Günter Born

Das MS-DOS-Programmierhandbuch MS-DOS 1.x - 6.x

Günter Born

Die Informationen in diesem Buch werden ohne Rücksicht auf einen eventuellen Patentschutz gemacht. Eventuell vorkommende Warennamen werden benutzt, ohne daß ihre freie Verwendbarkeit gewährleistet werden kann.

Das Buch wurde mit größter Sorgfalt erstellt und korrigiert. Dennoch können Fehler und Ungenauigkeiten nicht ausgeschlossen werden - wir sind auch nur Menschen.

Weder Verlag noch Autor können für fehlerhafte Angaben oder gar deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen. Bitte haben Sie Verständnis, dass ich Verbesserungsvorschläge oder Hinweise auf Fehler nicht mehr einarbeiten kann.

Satz: Günter Born

Herstellung: Günter Born

Inhaltsverzeichnis

Vorwort

Der eine oder andere Leser wird sich sicher die Frage stellen: wieso wird hier noch ein Buch über DOS geschrieben? Gibt es denn nicht schon genug Literatur zu diesem Thema, ganz abgesehen von den Unterlagen der Rechnerhersteller.

Zur Erklärung der Beweggründe möchte ich kurz auf die Entwicklungsgeschichte dieses Buches eingehen. Als ich Mitte 1983 im Rahmen eines Projektes mit einem der ersten IBM PC/XT in Europa konfrontiert wurde, war meine Welt noch in Ordnung. Als Betriebssysteme wurden damals CP/M 86 und ein unbekanntes System Namens MS-DOS mitgeliefert. Es handelte sich um die Version 1.0, die sich nicht besonders stark von CP/M unterschied, aber die vorhandene Festplatte nicht unterstützte. Nach wenigen Wochen kam dann DOS 2.0 hinzu, so daß die Festplatte auch betrieben werden konnte. Bemerkenswert war das hierarchische Dateisystem und die Möglichkeit, im Hintergrund zu drucken. Mit der Dokumentation gab es auch kaum Probleme, da sie nicht zur Verfügung stand. Um die Zeit bis zur Lieferung der DOS-Version 2.1 zu überbrücken, wurde vom Lieferanten schließlich die Fotokopie einer Microsoft-Dokumentation ausgegeben. Hierbei handelte es sich um einen gut 5 cm dicken Wälzer, der viele Informationen über das Betriebssystem, einschließlich der Beschreibung der INT-21-Aufrufe, enthielt. Außerdem bot die Softwareentwicklung mit dBase II genügend Probleme, so daß eine Beschäftigung mit den DOS-Interna nicht zur Debatte stand. Als nach einem Monat die PC-DOS-Version 2.1 geliefert wurde, war der Umfang der Original-Dokumentation bereits merklich geringer. Die Interna von DOS wurden nicht mehr beschrieben. Informationen über die Systemprogrammierung sind nur in zusätzlichen Handbüchern erhältlich. Selten werden jedoch die Randgebiete wie BIOS-Aufrufe, Graphikausgabe, Mausschnittstelle, etc. behandelt. Der Programmierer benötigt daher eine Reihe unterschiedlicher Werke zu den jeweiligen Themen.

Ein anderer Aspekt betrifft die Vollständigkeit der Informationen. Insbesondere in früheren DOS-Versionen wies das Betriebssystem eine Vielzahl undokumentierter Schnittstellen auf. Insbesondere die vielen als »reservedUm die Informationen aus den vielen verschiedenen Quellen für eigene Zwecke besser aufzubereiten, wurden die Erkenntnisse und Informationen in Form fliegender Blätter zusammengeschrieben. Mit wachsender Erfahrung konnten dann viele Informationen, die alleine betrachtet keinen Sinn ergaben, im Gesamtkontext bewertet werden, so daß das Puzzle Stück für Stück aufgedeckt wurde. MS-DOS liegt mittlerweile in verschiedenen Versionen (bis DOS 6.0) vor und ist weltweit mehr als 60 Millionen mal verbreitet. Da die Literatur andererseits kaum geschlossene Informationen über die verschiedenen Versionen und die undokumentierten Eigenschaften bietet, entstand die Idee, die bisher angesammelten Informationen in Form eines Buches aufzubereiten und einem breiteren Kreis zugänglich zu machen. Die erste Version dieses Buches wurde 1988 publiziert und eroberte sich schnell einen festen Platz auf den Schreibtischen der Entwickler.

In Laufe der folgenden Jahre wurde das Werk mehrfach überarbeitet. Ein eigenes Kapitel über die Funktionen der EGA- und VGA-Karten einschließlich der herstellerspezifischen Eigenheiten oder der Abschnitt über den *INT 2F* kamen in späteren Jahren hinzu. Das fehlende Wissen um das Thema Speichererweiterung war für mich der Grund, ein eigenes

Kapitel über Extended/Expanded Memory aufzunehmen. Weitere Themen wie die NetBIOS-Funktionen sollen das Wissen über neue Entwicklungen einem breiteren Kreis nahebringen. Erweiterungen um die neuen DOS 5.0-Funktionen, sowie die Erstellung residenter Programme sind weitere Stichpunkte.

Nun noch ein Wort zur Informationsbeschaffung: Als freier Autor stehen mir keinerlei Insiderinformationen des Herstellers zur Verfügung. Der Stoff besteht vielmehr aus einer Zusammenfassung allgemein zugänglicher Informationen, die durch eigene Erfahrungen ergänzt wurden. So viel zur Entstehungsgeschichte dieses Buches. Nun ist es an der Zeit, mich bei den Personen zu bedanken, die dieses Buch überhaupt erst ermöglicht haben.

An erster Stelle sei Mark Zbikovski (einem der Entwickler von DOS) gedankt, dessen Initialen mir bei so mancher nächtlichen Sitzung am Debugger begegneten. Sie verhalfen zu unerwarteten Erfolgserlebnissen und erlaubten die Orientierung im Innern von DOS.

Erwähnen möchte ich hier auch die Gruppe der Insider und Freaks, stellvertretend seien hier Ray Duncan und Adrew Schulman genannt, die viele Interna von DOS erforschten und ihre Erkenntnisse veröffentlichen.

Die Erstellung und Korrektur des Manuskriptes wäre ohne die tatkräftige Unterstützung meiner Frau und das Verständnis meiner Familie nicht möglich gewesen. Auch ihnen gilt mein Dank.

Und hier verließen ihn die Kräfte...

Denn es ist ein hoffnungsloses Unterfangen, alle Personen namentlich zu erwähnen.

Zum Abschluß nun noch mein Dank an die Firma Microsoft, die sich zur Veröffentlichung dieser Informationen entschloß und somit viele offene Fragen der Entwickler beantwortet.

Allen Lesern wünsche ich viel Erfolg und neue Erkenntnisse beim Umgang mit den bekannten und unbekannten Seiten von MS-DOS.

Günter Born

1 Einführung

Der Umgang mit MS-DOS gehört mittlerweile für viele PC-Besitzer zum Alltag. Dabei ist es oft unerläßlich, sich tiefer mit dem Betriebssystem zu beschäftigen. Für Software-Entwickler sind die Schnittstellen zum BIOS genau so wichtig, wie die DOS-Funktionsaufrufe. Dabei sind nicht nur die Kenntnisse rudimentärer Aufrufkonventionen wichtig, sondern es kommt vielfach auf das grundlegende Verständnis der internen Arbeitsweise des Betriebssystems an. Dieses grundlegende Verstehen der Abläufe kann auch bei der Verwendung von Anwenderprogrammen weiterhelfen, falls die Software nicht in der vorgesehenen Weise läuft (Leistungseinbußen, Fehlfunktionen, Fehlerabbrüche). Im Verlaufe der folgenden Kapitel werden die Schnittstellen zu MS-DOS (PC-DOS) und die BIOS-Aufrufe beschrieben. Weiterhin widmet sich das Buch der Erklärung der internen Abläufe.

1.1 Das Speicherabbild der 8086/8088-Prozessoren

MS-DOS hat sich innerhalb kürzester Zeit als Standard-Betriebssystem für die auf Intel-Prozessoren basierenden PCs durchgesetzt.

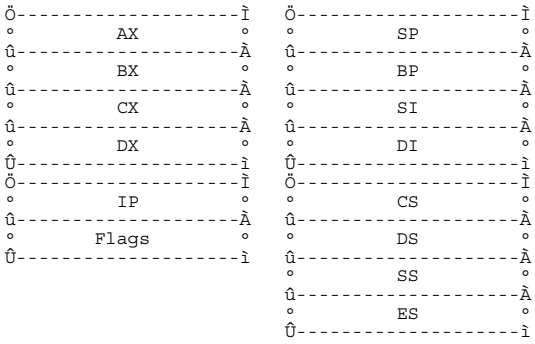


Bild 1.1: 8086- Registerstruktur;

Die Ursprünge von MS-DOS und PC-DOS gehen auf das von der Firma Seattle Computer für den 8086-Prozessor entwickelte QDOS (quick and dirty operating system) zurück. Später wurde das Produkt von Microsoft gekauft und unter dem Namen MS-DOS vermarktet. Auch wenn mittlerweile gravierende Erweiterungen vorgenommen wurden, lehnt sich MS-DOS nach wie vor stark an die Architektur der 8086/8088-Prozessoren an. Diese arbeiten intern mit 16 Bit breiten Adreß- und Datenpfaden. Dies gilt auch für die 80286- und 80386-CPUs, sofern sie im 8086-Real-Mode betrieben werden. Der interne Registeraufbau des 8086 wird in Bild 1.1 kurz skizziert.

Da mit den 16-Bit-Registern nur ein Adreßbereich von 64 Kbyte erreicht wird, haben sich die Entwickler etwas einfallen lassen, um auf einen Adreßraum von 1 Mbyte zu kommen.

Die 80x86/8088-Prozessoren benutzen zwei Register zur Erzeugung der physikalischen Adresse.

1.2 Die Segment Offset-Notation

Es sind insgesamt 20 Bit zur Adressierung des 1-Mbyte-Bereiches notwendig. Die Arbeitsregister (AX .. DX, SP, BP, SI, DI, IP) erreichen jedoch nur jeweils einen Ausschnitt von 64 Kbyte. Die CPU unterteilt nun den 1-Mbyte-Bereich in Segmente mit einer Größe von 16 Byte bis 64 Kbyte (Bild 1.2).

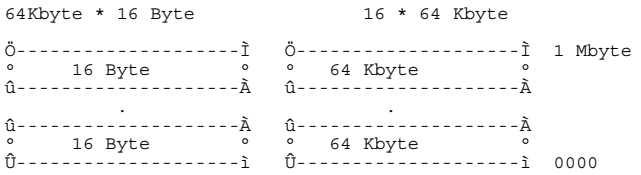


Bild 1.2: Aufteilung des 8086-Adreßraumes in Segmente

Die Basisadresse eines solchen Segments wird durch eines der vier Segmentregister (Codesegment, Datensegment, Stacksegment und Extrasegment) festgelegt. Eine Speicherzelle innerhalb dieses Segments wird nun durch den Offsetwert innerhalb der Arbeitsregister angesprochen.

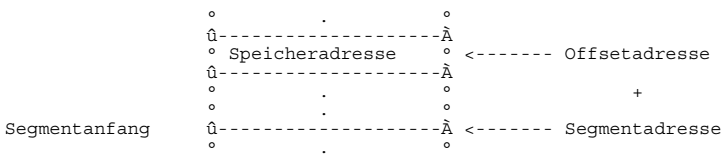


Bild 1.3: Adressierung einer Speicherzelle

Die Adreßberechnung der Businterface-Einheit (BIU) addiert daher bei jedem Speicherzugriff den Wert des Segmentregisters und den Offsetwert in einem Arbeitsregister. Hierbei spezifiziert das Segmentregister die Bits b4 bis b19, während der Offset die Bits b0 bis b15 belegt. Das Segmentregister ist also um 4 Bit nach »links(Segmentadresse:Offsetadresse).

```
Adresse in Segment:Offset-Schreibweise = F000:8000

      F000       Segment * 16
       8000      Offset
      -------
      F8000H     physikalische Adresse
```

Bild 1.4: Umrechnung der Segment:Offset-Adressen in absolute Adressen

Diese Segment:Offset-Notation wird häufig auch in der Literatur verwendet. Der Wert F8000H (20-Bit-Adresse) kann daher als F000:8000 oder F800:0000 angegeben werden.

Eine Umrechnung der Segment:Offset-Angaben in physikalische Adressen ist recht einfach. Hierzu wird der Wert des Segments mit 16 multipliziert und zum Offset addiert. Eine Multiplikation mit 16 (10H) läßt sich im Hexadezimalsystem leicht durch Verschieben um eine Stelle nach links erreichen. Das Beispiel in Bild 1.4 zeigt noch einmal diesen Vorgang.

Durch diese Verschiebung um vier Bit wird eine Adresse im 1-Mbyte-Bereich generiert. Andererseits bedingt dies auch, daß die minimale Segmentgröße 16 Byte beträgt. Die maximale Größe eines Segments wird durch die Breite der Arbeitsregister bestimmt und liegt bei 64 Kbyte.

1.3 Die Speicheraufteilung durch Intel

Die Firma Intel reservierte bereits beim Entwurf der Prozessoren bestimmte Speicherbereiche innerhalb des 1-Mbyte-Adreßraumes. Einmal handelt es sich um den Restartpunkt, an dem der Prozessor nach einem RESET mit der Abarbeitung des Programmes beginnt. Beim 8086/8088 sind hierfür die oberen 16 Adressen FFFF0H bis FFFFFH reserviert, wobei die erste Anweisung bei FFFF0H stehen muß. Bei den 80286/80386-Prozessoren werden im *Real Mode* die oberen physikalischen Adressen (24 Bit oder 32 Bit) in den 1-Mbyte-Bereich gespiegelt, so daß im Prinzip die gleichen Bedingungen vorliegen.

Der zweite reservierte Bereich liegt zu Beginn des Speicherbereiches und enthält die Interrupttabelle. Die Prozessoren können im 8086-Mode insgesamt 256 verschiedene Interrupts bearbeiten. Tritt eine Unterbrechung auf, verzweigt der Prozessor in eine Interrupt-Service-Routine. Deren Adresse ist für jeden Interrupt separat als 4-Byte-Vektor in einer Tabelle eingetragen. Diese Tabelle ist 1 Kbyte groß und befindet sich im Adreßbereich von 000 bis 3FFH. Intel hat innerhalb dieser Tabelle die ersten 128 Byte (00 bis 7FH) für eigene Zwecke reserviert. Das bedeutet, daß die Interrupts 0 bis 31 nicht vom Anwender belegt werden dürfen, falls eine Kompatibilität mit anderen Intel-Hard- und-Softwareprodukten gewährleistet werden soll.

Damit ergibt sich folgendes Speicherabbild innerhalb des 8086/8088-Adreßraumes.

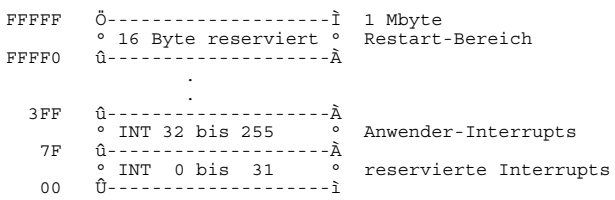


Bild 1.5: Reservierte Speicherbereiche des 8086/8088

Aus den Intel-Datenbüchern der 80x86-und 8088-Prozessoren ergibt sich die Belegung der reservierten Interrupt-Bereiche (die mit einem * markierten Interrupts werden nur vom 80286 benutzt, während die mit + markierten Interrupts durch den 80386-Prozessor belegt sind).

INT		Bemerkung
0		Divide Error
1		Single Step
2		NMI (Non Maskable Interrupt)
3		INT3 (Breakpoint Interrupt)
4		INT0 (Overflow Interrupt)
5	*	Bound Range Exeded Exeption
6	*	Invalid Opcode
7	*	Processor Extension not available (ESC, WAIT)
8	*	Interrupt Table Limit too small Exeption
	+	Double Fault
9	*	Processor Extension Segment Overrun
	+	Coprozessor Segment Overrun
10	+	Invalid TSS (JMP, CALL, IRET, RET)
11	+	Segment not present
12	+	Stack Fault
13	*	Segment Overrun Exeption
	+	General Protection Fault
14	+	Page Fault
-		INT 15 reserved
16	*	Processor Extension Error
	+	Coprozessor Error
-		INT 17-31 reserved
-		INT 32-255 user interrupt

Tabelle 1.1: Vorbelegung der Interrupts durch Intel

Bei der Entwicklung von Software sollten diese Spezifikationen berücksichtigt werden, um eine Kompatibilität hinsichtlich zukünftiger Hard- und Softwareprodukte zu gewährleisten.

Insbesondere die Tabelle zeigt, daß bereits beim Übergang vom 8086- auf den 80286-Prozessor die Interrupts 5,6,7,8,9,13 und 16 belegt wurden. Nur wenn die 80286-CPU im Real Mode (8086-Emulation) betrieben wird, sind diese Interrupts noch unbenutzt.

1.4 Die Speicherbelegung durch MS-DOS

Die Entwickler von MS-DOS konnten die Intel-Vorgaben natürlich nicht komplett ignorieren. Deshalb lehnt sich die Aufteilung des Speicherbereiches an Bild 1.5 an. Allerdings waren weitere Festlegungen bezüglich der Adreßbelegung erforderlich. Bild 1.6 gibt einen groben Überblick über die wichtigsten MS-DOS-Speicherstrukturen.

Die Adressen von 0000:0000 bis 9000:FFFF sind für den 640-Kbyte-MS-DOS-RAM-Bereich reserviert. Der verbleibende Adreßraum dient zur Aufnahme von Erweiterungen (Bildschirmadapter, BIOS etc.). Die Belegung ist etwas abhängig von der vorliegenden Hardwarekonfiguration. Insbesondere bei den Bildschirmadaptern spiegelt sich in der Adreßbelegung die technische Entwicklung der letzten Jahre wieder.

Insgesamt ist der Adreßraum von A000:0000H bis B000:FFFFH für Grafikadapterkarten reserviert. Tabelle 1.2 gibt einen Überblick über die heutige Belegung. Der Bereich von C000:0000 bis E000:FFFF ist für die ROMs der Hardwareerweiterungen wie z.B. Festplatten vorgesehen. Besitzt das System eine Festplatte, findet sich das BIOS-ROM des Plattenkontrollers meist oberhalb des Bildspeicherbereiches. Die Adressen variieren zwischen C800:0000 und E800:0000. Innerhalb dieses BIOS-ROMs sind neben den Routinen zur Ansteuerung der Platte auch Hilfsprogramme zur physikalischen Formatierung der Platte abgespeichert. Erst nach einer solchen Formatierung läßt sich die Platte mit FDISK und FORMAT für MS-DOS herrichten.

Adreßbereich	Bemerkung
B000:0000 B000:0FFF	Bildschirmspeicher Monochrom-Karte (4 Kbyte)
B800:0000 B800:3FFF	Bildschirmspeicher Color-Grafikadapter (16 Kbyte)
B000:0000 B000:7FFF B800:0000 B800:7FFF	Bildschirmspeicher Herkules-Grafikadapter (32 Kbyte Seite 0) Bildschirmspeicher Herkules-Grafikadapter (32 Kbyte Seite 1)
A000:0000 A000:FFFF	Bildschirmspeicher EGA-Adapter volle Aufrüstung. (64 Kbyte pro Seite)

Tabelle 1.2: Belegung des Adreßraumes durch die Grafikadapter

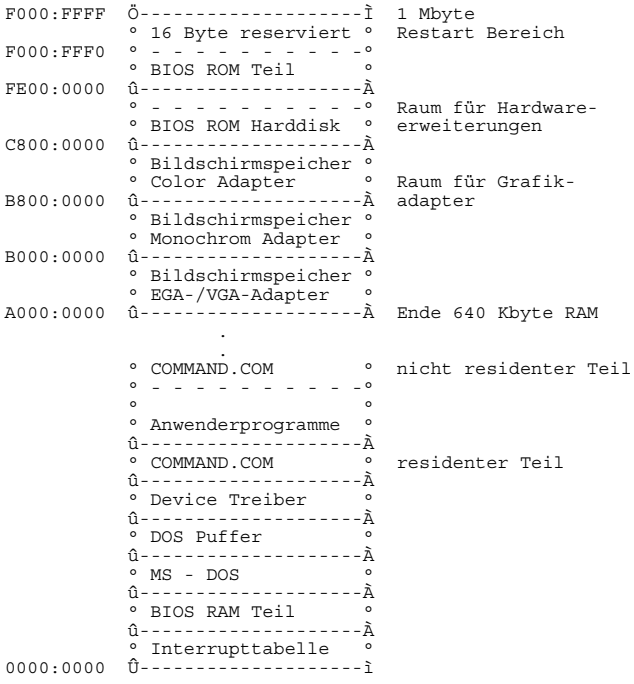


Bild 1.6: Aufteilung des 1-Mbyte-Speichers durch MS-DOS

Das BIOS-ROM für den PC beginnt ab der Adresse FE00:0000. Es enthält den hardwareabhängigen Teil und ist fest abgespeichert. Auf den im RAM-Bereich liegenden Ergänzungsteil (IO.SYS oder IBMBIO.COM) wird später noch eingegangen. Im BIOS-ROM findet sich auch die Restart-Routine für den Neuanlauf des Systems.

Beim IBM PC befindet sich im Bereich F000:0000 bis FE00:0000 das Basic-ROM. Bei kompatiblen PCs bleibt der Bereich entweder frei, oder er wird durch Zusatz-EPROMs (z.B. für Monitorprogramme) benutzt.

Damit ist die grobe Aufteilung des Speichers bekannt. Die Intel-Vorgaben bezüglich der oberen und unteren Speicherbelegung wurden offensichtlich berücksichtigt. Im folgenden Schritt soll die Organisation innerhalb des MS-DOS-RAMs untersucht werden.

1.5 Die Aufteilung des 640-Kbyte-MS-DOS-RAM-Bereiches

Die Entwickler von MS-DOS reservierten die unteren 640 Kbyte für das Betriebssystem und die Anwenderprogramme. Sie standen aber vor dem Problem, daß nicht alle Geräte mit den vollen 640 Kbyte ausgerüstet waren. Insbesondere die ersten IBM-PCs enthielten im Grundausbau nur 64 Kbyte RAM. Weiterhin besteht ab der Version 2.0 die Möglichkeit, daß Betriebssystem in gewissem Umfang durch den Benutzer zu konfigurieren (Device-Treiber, Puffer etc.). Eine starre Einteilung des RAM-Bereiches für bestimmte Funktionen war deshalb nicht möglich. Aus diesem Grunde wurde MS-DOS mit einer eigenen Speicherorganisation, die abhängig von der Systemkonfiguration die Aufteilung dynamisch vornimmt, ausgestattet. Damit lassen sich auch nur wenige Adressen angeben, an denen bestimmte Informationen stehen. Andererseits ist die relative Lage einzelner Systemteile im Speicherbereich aber immer gleich, so daß sich der schematische Aufbau angeben läßt.

Lage und Größe der Interrupttabelle wurden bereits durch Intel fixiert. Daran schließen sich die BIOS- und DOS-Datenbereiche an. Anschließend folgt der residente RAM-BIOS-Teil, der aus den Dateien IO.SYS oder IBMBIO.COM geladen wird. Dahinter findet sich das eigentliche MS-DOS, dessen Systemkern aus den Dateien MSDOS.SYS oder IBMDOS.COM geladen wird. Welche der beiden Dateien Verwendung findet, hängt vom PC ab. Das Betriebssystem der IBM-Personalcomputer (PC-DOS) verwendet die Dateien IBMBIO.COM und IBMDOS.COM. Innerhalb der kompatiblen Systeme gelangen die Dateien IO.SYS und MSDOS.SYS zum Einsatz. Die Bezeichnung der Dateien deutet schon an, daß zwischen MS-DOS und PC-DOS kleinere Unterschiede bestehen.

Um eine möglichst hohe Kompatibilität mit den IBM-Personalcomputern zu erreichen, verwenden die Hersteller kompatibler Systeme teilweise lizenzierte Versionen von IBMBIO.COM und IBMDOS.COM.

Oberhalb der DOS-Systemprogramme finden sich die Datenbereiche für die I/O-Puffer, sowie ein Kontrollbereich, in dem MS-DOS bestimmte Steuerinformationen ablegt. Daran schließt sich der Bereich an, in dem die residenten Treiber geladen werden. Diese werden durch Auswertung der Datei CONFIG.SYS aktiviert. Der Aufbau dieser Treiber wird in einem eigenen Kapitel beschrieben. Der residente Teil des Kommandointerpreters COMMAND.COM schließt den Systemteil ab. Der nicht residente Teil von COMMAND.COM wird immer an das Ende des RAM-Bereiches geladen.

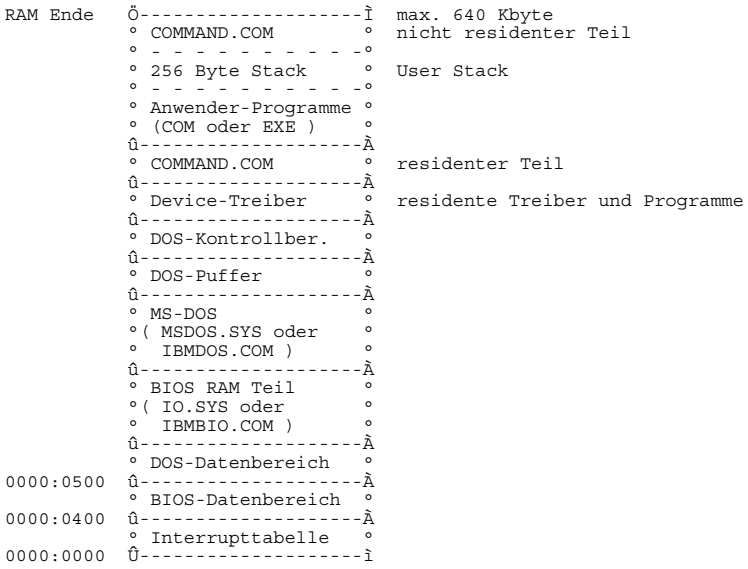


Bild 1.7: Aufteilung des 640-Kbyte-RAM durch MS-DOS

Anwenderprogramme finden sich zur Laufzeit zwischen dem residenten und nicht residenten Teil von COMMAND.COM. Reicht der freie Speicherbereich nicht aus, wird der nicht residente Teil von COMMAND.COM überschrieben. Nach Beendigung des Anwenderprogramms lädt DOS diesen Teil wieder nach.

Die komplette Verwaltung des Speichers erfolgt automatisch durch MS-DOS, ohne den Anwender zu belasten.

1.6 Die MS-DOS-Interupt-Vektor-Tabelle

Im Abschnitt 1.3 wurde bereits die Tatsache besprochen, daß der 8086 insgesamt 256 Interrupts besitzt, von denen die ersten 32 von Intel für Hard- und Softwareerweiterungen reserviert wurden. Der Einsprung in eine Interrupt-Service-Routine erfolgt über eine (1 Kbyte) Tabelle, die zu Beginn des Speicherbereiches liegt. Jedem Interrupt ist in dieser Tabelle ein 4-Byte-Adreßvektor mit folgendem Aufbau zugeordnet:

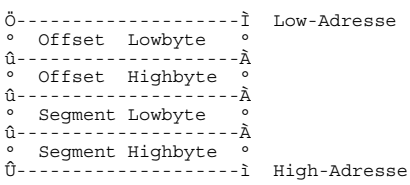


Bild 1.8: Aufbau des Interruptvektors

Der Prozessor rettet das Flagregister, sowie die Programmadresse auf den Stack. Dann wird zu der Programmstelle verzweigt, deren Adresse unter dem Interruptvektor abgespeichert ist. Nicht benutzte Interrupts sollten deshalb mit einem Vektor auf eine Routine mit einer IRET-Anweisung abgeschlossen werden. Wie aber aus folgendem Bild ersichtlich ist, besetzt MS-DOS den nicht genutzten Bereich teilweise mit den Werten 00 00 00 00. Tritt hier ein Interrupt auf, verzweigt das Programm zur Adresse 0000:0000 und stürzt ab, da hier kein Code vorhanden ist. Lediglich einige Interrupts, im nachfolgenden Bild mit dem Vektor 029C:1374 belegt, sind mit einer IRET-Anweisung abgefangen.

```
-d 0:0 3fff
0000:0000  88 51 9C 02 60 0C 70 00-A5 01 70 00 60 0C 70 00
0000:0010  60 0C 70 00 2C 0F 00 FC-39 0A 2F 0D 4D 3F 00 FC
0000:0020  36 09 2F 0D 1F 03 AB 0C-24 03 70 00 9E 03 70 00
0000:0030  18 04 70 00 92 04 70 00-0C 05 70 00 47 3F 00 FC
0000:0040  3B 05 2F 0D B8 0B 00 FC-AE 0B 00 FC D8 18 70 00
0000:0050  A4 0F 00 FC C2 0B 00 FC-00 11 00 FC AB 0E 00 FC
0000:0060  EF 0A 00 FC 37 20 70 00-C6 0C 00 FC 5A 0C 70 00
0000:0070  4D 3F 00 FC A4 30 00 FC-22 05 00 00 00 00 00 00
0000:0080  6E 13 9C 02 8F 13 9C 02-F5 02 CB 0E 2E 03 CB 0E
0000:0090  BD 02 CB 0E 0B 15 9C 02-4E 15 9C 02 00 5E 9C 02
0000:00A0  74 13 9C 02 7A 01 70 00-74 13 9C 02 74 13 9C 02
0000:00B0  74 13 9C 02 74 13 9C 02-57 02 B6 0B D5 1F 70 00
0000:00C0  EA 75 13 9C 02 13 9C 02-74 13 9C 02 BF 04 2F 0D
0000:00D0  74 13 9C 02 74 13 9C 02-74 13 9C 02 74 13 9C 02
0000:00E0  74 13 9C 02 74 13 9C 02-74 13 9C 02 74 13 9C 02
0000:00F0  74 13 9C 02 74 13 9C 02-74 13 9C 02 74 13 9C 02
0000:0100  59 2C 00 FC 00 00 00 00-00 00 00 00 00 00 00 00
0000:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
                   .
                   .
```

Bild 1.9: Auszug aus der MS-DOS-Interruptvektor-Tabelle

Weiterhin haben sich die Entwickler von MS-DOS nicht an die Intel-Spezifikation gehalten, wonach die ersten 32 Interrupts reserviert sind. Die folgende Tabelle zeigt die Belegung der Interruptvektoren durch die Hardware, das BIOS und MS-DOS.

INT	Adr	Bemerkungen
0	00	Division durch Null
1	04	Single-Step-Interrupt (z.B. von DEBUG belegt)
2	08	NMI (per Hardware für Parity Error belegt)
3	0C	Break-Point-Interrupt (z.B. von DEBUG belegt)
4	10	Arithmetik-Overflow
5	14	BIOS-Routine Print Screen
6	18	reserviert für Hardware (z.B. Mausabfrage beim Schneider PC)
7	1C	reserviert für die Hardware
8	20	System-Zeit (IRQ 0 Master PIC)
9	24	Tastatur-Interrupt (IRQ 1 Master PIC)
0A	28	reserviert für Hardware (IRQ 2 z.B. vom Slave PIC)
0B	2C	reserviert für Hardware (IRQ 3 z.B. COM 2 beim AT)
0C	30	reserviert für Hardware (IRQ 4 z.B. COM 1 beim AT)
0D	34	reserviert für Hardware (IRQ 5 z.B. LPT 2 beim AT)
0E	38	Disketten-Controller-Interrupt (IRQ 6 Master PIC)
0F	3C	reserviert für Hardware (IRQ 7 z.B. LPT 1 beim AT)
10	40	BIOS-Bildschirm-Ausgabe
11	44	BIOS-Konfiguration-Test
12	48	BIOS-Speichergröße ermitteln
13	4C	BIOS-Disketten und Disk I/O
14	50	BIOS-serielle Schnittstelle
15	54	BIOS (IBM-PC - Kassettenrecorder-Abfrage) (IBM-AT - Extended-Memory-Zugriffe) (Schneider PC - NVR und Graphic-Backplanes)
16	58	BIOS-Tastaturroutinen
17	5C	BIOS-Druckerroutinen
18	60	IBM-PC - ROM-Basic aktivieren, sonst unbelegt
19	64	BIOS-System-Warmstart
1A	68	BIOS-Zeit
1B	6C	Tastatur-Break-Interrupt
1C	70	Timer-Interrupt (Folgeroutine)
1D	74	Pointer auf Tabelle mit Bildschirmparametern
1E	78	Pointer auf Tabelle mit Floppy-Drive-Parametern
1F	7C	Pointer auf Tabelle mit Bildschirm-Grafikzeichen
20	80	DOS Program Terminate
21	84	DOS Function Request
22	88	DOS Terminate Process Exit Address
23	8C	DOS Control-C-Handler Address
24	90	DOS Critical Error Handler Address

25	94	DOS Disk Absolute Read
26	98	DOS Disk Absolute Write
27	9C	DOS Program Terminate and Stay Resident
----	------	INT 28 - INT 3F reserviert für DOS (undokumentiert)
28	A0	Idle Flag
29	A3	Character Output
.	.	reserviert
2E	B8	EXEC-CALL
2F	BC	Multiplexerinterrupt
.	.	reserviert
33	CC	Maustreiber
.	.	reserviert
34	D0	Turbo Lang. Floating Point Emulator (Opcode D8H)
35	D4	Turbo Lang. Floating Point Emulator (Opcode D9H)
36	D8	Turbo Lang. Floating Point Emulator (Opcode DAH)
37	DC	Turbo Lang. Floating Point Emulator (Opcode DBH)
38	E0	Turbo Lang. Floating Point Emulator (Opcode DCH)
39	E4	Turbo Lang. Floating Point Emulator (Opcode DDH)
3A	E8	Turbo Lang. Floating Point Emulator (Opcode DEH)
3B	EC	Turbo Lang. Floating Point Emulator (Opcode DFH)
3C	F0	Turbo Lang. Floating Point Emulator (mit ES:)
3D	F4	Turbo Lang. Floating Point Emulator (FWAIT..)
3E	F8	Turbo Lang. Floating Point Emulator
3F	FC	Overlay Manager Interrupt
		INT 40 - INT 5F reserviert von IBM für zukünftige (BIOS-)Erweiterungen
40	100	Disktreiber bei Festplattensystemen
41	104	Parametertabelle Festplatte Drive 0
42	108	Videostatustabelle EGA-Karten
43	10C	EGA-Grafik-Zeichensatztabelle
44	110	EGA-Zeichengenerator
45	114	reserviert
46	118	Parametertabelle Festplatte Drive 1
.	.	reserviert
4A	128	Timer Alarm (BIOS)
.	.	reserviert
50	140	Timer Alarm (BIOS)
.	.	reserviert
5F	17F	reserviert
60	180	frei für Anwenderprogramme und Erweiterungen (beim PC sind die Interrupts bis 7F frei für Anwenderprogramme). Die Interrupts werden teil- weise von Netzwerktreibern und VGA-Karten be- nutzt.)
67	19F	EMS-Treiber
68	1A0	Diese Interrupts sind beim PC frei für Anwender- programme. (Bei späteren Versionen sind sie als
6F	1BF	unbenutzt markiert)
70	1C0	IRQ-8-Eingang (REAL Time Clock beim AT)
71	1C4	IRQ-9-Eingang (BIOS Redirect beim AT)
72	1C8	IRQ-10-Eingang (z.B. COM 3 beim AT)
73	1CC	IRQ-11-Eingang (z.B. COM 4 beim AT)
74	1D0	IRQ-12-Eingang (Slave PIC beim AT)
75	1D4	IRQ-13-Eingang (Arithmetikprozessor beim AT)
76	1D8	IRQ-14-Eingang (Slave PIC beim AT)
77	1DC	IRQ-15-Eingang (Slave PIC beim AT)
78	1E0	frei für Anwenderprogramme beim PC, beim AT als unbenutzt markiert
7F	1FF	
80	200	reserviert für Basic (PC-DOS)
.	.	
85	217	

86	218	benutzt durch den Basic-Interpreter (PC-DOS)
.	.	
F0	3C3	
F1	3C4	nicht benutzt
.	.	
FF	3FF	

Tabelle 1.3: Belegung der 8086-Interrupts durch MS-DOS (PC-DOS)

Die ersten fünf Unterbrechungsvektoren (INT 0 bis INT 4) werden durch die 8088/8086- und 80286-Prozessoren belegt. Nachfolgend findet sich eine Kurzbeschreibung der Interrupt-Service-Routinen.

Division durch 0 (INT 0)

Der Vektor wird durch DOS initialisiert. Ein entsprechender Interrupt wird ausgelöst, falls das Ergebnis eines Divisionsbefehls (DIV oder IDIV) ungültig ist (Division durch 0 oder Überlauf). Das laufende Programm wird dann mit der Meldung:

```
Divide Overflow;
```

abgebrochen. Anschließend erscheint der MS-DOS Prompt (C>), um Bedienereingaben zu erlauben.

Single Step (INT 1)

Die Intel-80x86-und 80x88-Prozessoren besitzen die Möglichkeit, Programme im Einzelschrittbetrieb abzuarbeiten. Sobald das *Trap Flag* gesetzt ist, führt der Prozessor nach jedem Befehl einen INT 1 aus. Der Interruptvektor kann dann auf eine Routine zeigen, die die Registerbelegung und den letzten Befehl auswertet. Der Vektor wird von DOS immer so initialisiert, daß er auf eine IRET-Routine zeigt. Sollte ein Programm den *Single-Step-Mode* aktivieren, bleibt dies dann ohne Folgen, da nach jedem Interrupt das Programm an der alten Stelle fortgesetzt wird. Lediglich die Ablaufgeschwindigkeit verringert sich. Das DOS-Hilfsprogramm DEBUG benutzt diesen Interrupt für die Trace- und Procedure-Befehle, um so den Programmablauf anzuzeigen.

NMI (Non Maskable Interrupt INT 2)

Dieser Interrupt wird durch einen externen Signaleingang aktiviert und kann nicht durch den CLI (Clear Interrupt Enable Flag)-Befehl gesperrt werden. Bei vielen PCs wird der Interrupt zur Anzeige von Speicherfehlern benutzt. Der RAM-Bereich ist hierfür mit einer einfachen Paritätsprüfung ausgestattet, die bei Fehlern den NMI-Eingang des Prozessors setzt. Der INT 2 aktiviert eine BIOS-Routine, die mit der Meldung:

```
Parity Check 1 oder Parity Check 2
```

reagiert und anschließend das Interruptsystem sperrt und den Prozessor mit einem HLT-Befehl stoppt.

Break-Point-Interrupt (INT 3)

Neben den Interrupts, die intern durch den Prozessor ausgelöst werden (INT 0 ...), sowie den externen Interupts (NMI, INTR), besteht die Möglichkeit per Software eine Unterbrechung zu erzeugen. Diese Unterbrechung wirkt im Programm wie ein Sprung zu einer Speicherstelle, wobei aber die Konditionen für Interrupts gelten. Ein *Software-Interrupt* läßt sich deshalb nicht sperren. Für einen Software-Interrupt sind normalerweise zwei Operationscodebyte notwendig, was im Gegensatz zu einem FAR CALL einige Byte spart. Andererseits sind zwei Byte zum Setzen von Unterbrechungspunkten beim Programmtest noch zuviel. Hierfür existiert deshalb ein Interrupt (INT 3), der sich durch ein Opcodebyte aktivieren läßt. DOS initialisiert den Interruptvektor so, daß er auf eine IRET-Anweisung zeigt. Das MS-DOS-Hilfsprogramm DEBUG benutzt den INT 3 zur Erzeugung von Breakpoints. Hierzu wird an der betreffenden Stelle der Opcode durch einen INT 3 ausgetauscht. Der MS-DOS-Debugger kann bis zu 10 Unterbrechungspunkte gleichzeitig verwalten.

Overflow (INT 4)

Diese Unterbrechung wird ausgeführt, falls in einem Programm der Befehl INTO (Interrupt if Overflow) auftritt. MS-DOS setzt den Vektor auf eine IRET-Anweisung, da die Behandlung von Overflows Aufgabe der Anwendersoftware ist.

Systemzeit 8253 Timer (INT 8)

Der Timer des IBM-PC unterbricht über den INT 8 das laufende Programm alle $^{5}/_{91}$ Sekunden (zirka 18,2mal). Innerhalb dieser Service-Routine werden die Speicherzellen für Timer Low und Timer High erhöht. Beim 24-Stunden-Übertrag wird das Overflow Flag gesetzt und die Zeit wird für zirka 9,65 Sekunden angehalten. Dies ist erforderlich, um die Gangabweichung der Uhrzeit halbwegs zu korrigieren. Allerdings können bei kompatiblen PCs andere Taktraten als $^{5}/_{91}$ Sek. gelten.

Tastatur (INT 9)

Das BIOS enthält eine Service-Routine, die die Kommunikation mit der Tastatur über diesen Interrupt abwickelt. Dieser Interrupt ist durch die Anwendersoftware kaum nutzbar und wird deshalb nicht weiter besprochen.

Diskette (INT 0E)

Hier sind die Interrupt-Service-Routinen für die Kommunikation mit dem Diskettencontroller abgelegt. Dieser Interrupt ist ebenfalls per Anwenderprogramm kaum zu nutzen, weshalb er nicht weiter besprochen wird.

Aus dieser Tabelle geht weiterhin hervor, daß fast alle der von Intel reservierten Interrupts durch das BIOS oder MS-DOS mit anderen Funktionen belegt wurden. Dies bringt bereits beim 80286 Probleme, wie ein Vergleich mit Tabelle 1.1 zeigt. Da die Interrupts 5 bis 7 belegt oder für Hardwarezwecke reserviert sind, läßt sich der 80286 nicht im *Protected Mode* betreiben, da hier andere Bedingungen gelten. Der INT 15 ist beim IBM-PC noch für

das Kassettenrecorder-Interface reserviert. Moderne PCs besitzen keinen Recorder, so daß der Interrupt für andere Zwecke verwendet wird. Beim AT liegt hier zum Beispiel der BIOS-Einsprung für Schreib-Lese-Zugriffe auf dem Extended Memory Bereich (EMS).

In den nachfolgenden Kapiteln werden die Schnittstellen zu den BIOS- und DOS-Interrupts noch detaillierter behandelt.

1.7 Der BIOS-Datenbereich

An die Interruptvektor-Tabelle schließt sich ein Datenbereich von 256 Byte an, in dem das BIOS bestimmte Systemdaten verwaltet. Leider gibt die von Microsoft veröffentlichte Dokumentation kaum Informationen über den Aufbau der Datenstrukturen. Dies ist einerseits sinnvoll, da ja das BIOS prinzipiell die Verwaltung der Daten übernimmt. Da andererseits die Entwicklung neuer BIOS-Versionen auf die Kompatibilität zum Original-IBM-PC-BIOS Rücksicht nehmen muß, bleibt die Belegung in der Regel erhalten.

Aus diesem Grunde wird nachfolgend der Aufbau dieses Datenbereiches, sofern er bekannt ist, aufgezeigt. Es sei aber nochmals darauf hingewiesen, daß es sich bei der Aufstellung um nicht von Microsoft dokumentierte Daten handelt, die nicht bei jedem PC stimmen müssen.

Adr 0:Ofs.	Bemerkungen
0400 - 0401	Port-Adresse COM1 (Wert normal = 03F8H)
0402 - 0403	Port-Adresse COM2 (Wert normal = 02F8H)
0404 - 0405	Port-Adresse COM3 (erst ab DOS 3.3 benutzt)
0406 - 0407	Port-Adresse COM4 (in DOS nicht benutzt)
0408 - 0409	Port-Adresse LPT1 (Wert normal = 03BCH)
040A - 040B	Port-Adresse LPT2 (Wert normal = 03B8H)
040C - 040D	Port-Adresse LPT3 (Wert normal = 02B8H)
040E - 040F	Port-Adresse LPT4 (in DOS nicht benutzt)
0410 - 0411	Equipment-Flag (Hardwarekonfiguration)
0412	Test-Flag während der Initialisierung
0413 - 0414	freie Speicherkapazität in Kbyte (beim PC = Kapazität auf der Systemplatine)
0415 - 0416	Speicher in Kbyte auf Erweiterungskarten (PC), beim AT Fehlerflag für Herstellercodes und das Err-Flag
0417 - 0418	Tastatur-Flag
0419	ALT-Keypad-Flag (Status CTRL, ALT, Shift etc.)
041A - 041B	Zeiger auf den Beginn des Tastaturpuffers
041C - 041D	Zeiger auf das Ende des Tastaturpuffers
041E - . - 043D	Tastatur-Ringpuffer von 32 Byte für 16 Tastatur-codes à 2 Byte
043E	Seek Status 1 Bit per Floppy Drive, 0 recalibrate
043F	Drive Motor Status, 1 Bit per Drive, 1 = Motor on Bit 7 = 1 -> Write in Progress
0440	Count Down Value Motor off
0441	Status of last Operation
0442 - .	Status des Floppy Disk Controllers 765
0449	CRT-Mode
044A - 044B	Zahl der Bildschirmspalten
044C - 044D	Länge der Bildschirmseite für den Refresh
044E - 044F	Startadresse (Offset) Bildschirmseite, die refresht werden soll
0450 - 0451	Cursorposition Bildschirmseite 1
0452 -	Cursorposition Bildschirmseite 2

0453	
0454 - 0455	Cursorposition Bildschirmseite 3
0456 - 0457	Cursorposition Bildschirmseite 4
0458 - 0459	Cursorposition Bildschirmseite 5
045A - 045B	Cursorposition Bildschirmseite 6
045C - 045D	Cursorposition Bildschirmseite 7
045E - 045F	Cursorposition Bildschirmseite 8
0460 - 0461	Cursorgröße in Rasterzeilen
0462	Nummer der aktuellen Bildschirmseite
0463 - 0464	Basis-I/O-Adresse des aktiven Bildschirmadapters
0465	Bildschirmmode des 3x8-Registers
0466	aktuelle Farbpalette beim Color Graphics Adapter
0467 - 0468	Taktlänge Kassettenrecorderaufzeichnung (IBM-PC) Zeiger auf Zusatz ROM-Initialisierung (IBM-AT)
0469 - 046A	CRC-Register Kassettenaufzeichnung (IBM-PC) Zeiger auf das I/O-ROM-Segment (IBM-AT)
046B	letzter Eingabewert Kassettenaufzeichnung (IBM-PC) Interruptflag (IBM-AT)
046C - 046D	Timer Low Word
046E - 046F	Timer High Word
0470	Timer Overflow Byte
0471	Break-Flag: Bit 7 = 1 Break-Taste gedrückt
0472 - 0473	Tastatur-Reset-Flag; Wert = 1234H -> Reset
0474	unbelegt beim PC mit Diskettenlaufwerken Disk Status beim PC/AT mit Harddisk
0475	unbelegt beim PC mit Diskettenlaufwerken Harddisk-Nummer beim PC/AT mit Harddisk
0476	unbelegt beim PC mit Diskettenlaufwerken Kontrollflag beim PC/AT mit Harddisk
0477	unbelegt beim PC mit Diskettenlaufwerken Portadresse Disk AUS beim PC/AT mit Harddisk
0478- 047B	unbelegt beim PC Printer (LPT) Time - Out bei neueren BIOS-ROMs
047C- 047F	unbelegt beim PC RS 232 (COM) Time - Out bei neueren BIOS-ROMs
0480	Zusatztastatur Pufferanfang
0482	Zusatztastatur Pufferende
0484- 0486	--
0487	EGA-Video-Mode

0488	EGA-Statusbyte
0489- 048A	--
048B	letzte Datentransferrate bei Floppy Disk
048C	Statusbyte-Festplatte
048D	Errorregister-Festplatte
048E	Interruptflag-Festplatte
048F	Kontrollflag-Festplatte
0490	Drive 0 - 90 Medium-Status (zusätzl. Disk-Daten)
0491	Drive 1 - 91 Medium-Status (zusätzl. Disk-Daten)
0492	Drive 0 - 90 Start-Status (zusätzl. Disk-Daten)
0493	Drive 1 - 91 Start-Status (zusätzl. Disk-Daten)
0494	Drive 0 - 90 aktueller Zylinder
0495	Drive 1 - 91 aktueller Zylinder
0496	Tastaturflag Byte 3 beim AT
0497	Tastatur-LED-Anzeigeflag
0498- 0499	Offsetadresse User Wait Flag (Real Time Clock)
049A- 049B	Segmentadresse User Wait Flag (Real Time Clock)
049C- 049D	Low Word User Wait Flag (Real Time Clock)
049E- 049F	High Word User Wait Flag (Real Time Clock)
04A0	Wait Aktiv Flag (Real Time Clock)
04A1 - 04EF	reserviert
04F0 - 04FF	Kommunikationsbereich für Zwischenanwendungen (Programm- und Intertask-Kommunikation)

Tabelle 1.4: Belegung des BIOS-Datenbereiches

Interessant sind Adressen, die über die Systemkonfiguration Auskunft geben. Es zeigt sich, daß die Entwickler von MS-DOS ursprünglich im BIOS die Unterstützung von je vier seriellen und parallelen Schnittstellen vorgesehen hatten, die dann allerdings nicht implementiert wurden.

Für die Bildschirmdarstellung sind insgesamt 8 Seiten im BIOS reserviert. Dies resultiert noch aus den Ursprüngen der PCs, als man mit TV-Sichtgeräten nur 25 Zeilen zu je 40 Zeichen darstellen konnte. Die Betriebsart läßt sich mit dem Kommando:

```
MODE 40
```

selektieren. Die Größe eines Zeichens wird in etwa verdoppelt. Bei Verwendung eines Color-Graphics-Adapter (CGA) reichen die 16 Kbyte Speicher, um insgesamt 8 Bildschirmseiten anzulegen. Im 25 x 80-Zeichen-Modus sind immerhin noch 4 Seiten

wählbar. Nur im Grafikmodus und beim Monochromadapter ist die Darstellung auf eine Seite begrenzt.

Auch die Lage des Tastaturpuffers mag für manche Anwendungen interessant sein.

Im Laufe der Zeit wurden weitere Datenbereiche belegt. So kamen beim Übergang vom PC auf den AT die Echtzeituhr und die Festplatte hinzu, die eigene Datenbereiche belegen. Ähnliches gilt für den Status der EGA-Karten, die den vormals unbelegten BIOS-Datenbereich ab 0487H benutzen.

Die letzten 16 Byte sind als Kommunikationsbereich für Zwischenanwendungen reserviert. Damit können zum Beispiel zwei Programme Daten ohne größeren Aufwand untereinander austauschen, da ja der Datenbereich an einer absoluten Speicheradresse steht. Bei umfangreicheren Daten reicht es, einen Adreßzeiger im BIOS-Datenbereich abzulegen. Dieser läßt sich von allen Prozessen lesen. Weiterhin können residente Programme ihre Signatur dort ablegen (1).

Der überwiegende Teil der Daten läßt sich per BIOS-Aufruf lesen oder modifizieren. So existiert z.B. ein eigener BIOS-Aufruf, um die Systemkonfiguration zu lesen (2,3,4). Die genaue Beschreibung dieser BIOS-Funktionen ist in einem getrennten Kapitel enthalten.

1.8 Der DOS-Datenbereich

Ein zweiter Datenbereich mit 256 Byte schließt sich an den BIOS-Datenbereich an. Dort verwaltet MS-DOS seine Systemdaten. Auch hier gibt die von Microsoft veröffentlichte Dokumentation kaum Informationen über den Aufbau der Datenstrukturen.

Aus diesem Grunde werden nachfolgend Auszüge aus der Belegung dieses Datenbereiches aufgezeigt. Es sei auch hier darauf hingewiesen, daß es sich um nicht dokumentierte Funktionen handelt.

Adr 0:Ofs.	Bemerkungen
0500	Print Screen Status-Flag 0 Print Screen nicht aktiv oder fertig 1 Print Screen arbeitet gerade 255 Fehler in der Print-Screen-Funktion
0501	Durch Basic belegt (IBM-PC)
0504	Single Drive Mode Status Byte 0 Disklaufwerk A: zuletzt benutzt 1 Disklaufwerk B: zuletzt benutzt
050F	Durch Basic belegt (IBM-PC) 1 = Basic geladen
0510 - 0511	Durch Basic belegt (IBM-PC) Standard Segment
0512 - 0515	Basic Interrupt Vektor für die CLOCK
0516 - 0519	Basic Interrupt Vektor für Tastatur Break
051A - 051D	Basic Interrupt Vektor für Disk Error
051E	--
0521	
0522	DOS Diskparam. Tab. Steppertime = D, HD Unload = F
0523	" HD Load = 1, Mode = DMA
0524	" Motor Wait
0525	" n * 256 Byte pro Sektor
0526	" Last Sector per Track
0527	" GAP length ID-Data
0528	" Disk Transfer length
0529	" GAP length for Format
052A	" Fill Byte for Format
052B	" Head settle Time (ms)
052C	" Motor Start Time (1/8) Sek.
052D - 052F	--
0530 - 0533	benutzt durch das Mode-Kommando in PC-DOS
0534 - 05FF	Belegung nicht bekannt

Tabelle 1.5: Belegung des DOS-Datenbereiches

An der Systemadresse 0000:0500 befindet sich das Print-Screen-Status-Flag. Hier kann leicht überprüft werden, ob die *PrtScr*-Funktion noch aktiv ist, oder ob beim letzten Aufruf ein Fehler auftrat.

Der zweite interessante Bereich ist die Disk-Parameter-Tabelle, die üblicherweise ab 0000:0522 beginnt. Hier verwaltet MS-DOS die Einstellparameter für die Diskettenlaufwerke. Zwar legt das BIOS bereits eine Tabelle an, aber DOS bietet die Möglichkeit, mit Hilfe dieser Tabelle die Parameter an die Laufwerke anzupassen.

In verschiedenen PC-DOS-Dokumentationen wird erwähnt, daß der Bereich 0000:530 bis 0000:0533 durch das Mode-Programm belegt wird. Dies konnte aber bei MS-DOS nicht eindeutig nachvollzogen werden.

Der Bereich zwischen 600H und 6FFH ist in der Regel unbelegt. Sie können dies mit dem DOS-DEBUG- oder MEM-Programm leicht verifizieren.

1.9 Die Funktion des Programmes IO.SYS (IBMBIO.COM)

Da MS-DOS (PC-DOS) ein hardwareunabhängiges Betriebssystem ist, muß eine Anpassungssoftware zwischen Hard- und Software existieren. Diese Aufgabe übernimmt das Programm IO.SYS bei MS-DOS oder IBMBIO.COM bei PC-DOS. Es stellt die Verbindung zwischen der Hardware, den ROM-BIOS-Routinen und MS-DOS her. Insbesondere besteht die Möglichkeit in IO.SYS Anpassungen an veränderte Hardwarekonfigurationen vorzunehmen, ohne daß das BIOS-ROM geändert werden muß. Dies hat sich als sehr hilfreich erwiesen, da sich damit Fehler innerhalb der ROM-BIOS-Routinen ausgleichen lassen.

Beim Programmstart lädt eine Routine im ROM-BIOS den Boot-Record von der Systemdiskette in den Speicher. Dieses Boot-Programm prüft, ob die Dateien IO.SYS (IBMBIO.COM) und MSDOS.SYS (IBMDOS.COM) auf der Diskette an bestimmten Stellen vorhanden sind und lädt diese anschließend in den Speicher.

Dann wird das Programm IO.SYS gestartet, welches aus drei Teilen besteht:

- Initialisierungsmodul
- residenter Teil
- SYSINIT

Zuerst wird ein Initialisierungsteil durchlaufen, der den Hardwarestatus ermittelt und die Diskettenlaufwerke zurücksetzt. Diese Aufgabe wurde zwar bereits teilweise durch das BIOS-ROM ausgeführt. Da aber im Laufe der Zeit immer neue Hardwarekomponenten eingeführt werden, bietet eine Wiederholung die Möglichkeit, auch solche neuen Komponenten zu unterstützen, deren Treiber nicht im BIOS-ROM stehen. Die Version 3.3 des Betriebssystems MS-DOS (PC-DOS) unterstützt z.B. erstmals 3,5-Zoll-Diskettenlaufwerke. Es ist nun ohne weiteres möglich, einen älteren PC auf DOS 3.3 mit 3,5-Zoll-Laufwerken hochzurüsten, ohne das BIOS-ROM auszutauschen. Die Anpassung wird durch IO.SYS vorgenommen. Nur wer in DOS-Versionen unterhalb 3.2 diese Laufwerke einsetzt, muß das BIOS selbst erw itern.

Im nächsten Schritt sind die Einheitentreiber zu installieren. Hierfür kennt DOS zwei Möglichkeiten. Der residente Teil von IO.SYS besteht im wesentlichen aus den »residentenh Anweisungen der Form:

```
DEVICE = Treibername (ab DOS 2.0)
DEVICEHIGH = Treibername (ab DOS 5.0)
```

eintragen, die durch IO.SYS ausgewertet werden. Wird eine entsprechende Anweisung gefunden, dann lädt IO.SYS diesen Treiber oberhalb des Programmes MSDOS.SYS in den Speicher. Für diese Auswertung wird das Modul SYSINIT benötigt. SYSINIT bearbeitet zuerst die installierbaren Treiber aus CONFIG.SYS, bevor es die Standardtreiber aus IO.SYS konfiguriert. Diese Option ermöglicht die Installation zusätzlicher Treiber (Maus, Plotter, etc.). Andererseits lassen sich damit die residenten Treiber aus IO.SYS leicht

deaktivieren und durch eigene Routinen ersetzen. In einem weiteren Schritt wird der untere Bereich der Interruptvektor-Tabelle (BIOS-Interrupts bis INT 1F) initialisiert.

Dann wird das Programm MSDOS.SYS (IBMBIO.COM) direkt an den residenten Teil von IO.SYS verschoben (relocate) und gestartet, um DOS zu initialisieren. Ab DOS 5.0 läßt sich ein Teil von MSDOS.SYS mit der Anweisung:

```
DOS=HIGH
```

in den HMA-Bereich (siehe Kapitel 12.2) verlagern. MSDOS.SYS gibt später die Kontrolle an IO.SYS wieder zurück, damit der Kommandoprozessor COMMAND.COM geladen werden kann. Im letzten Schritt wird durch einen Sprung zum ersten Codebyte des Kommandoprozessors die Kontrolle an diesen übergeben.

Der Speicherbereich von IO.SYS enthält ab Offset 0 einen Sprung in den Programmbereich mit dem Initialisierungscode. Da nach der Initialisierung dieser Bereich nicht mehr notwendig ist, wird er als Datenbereich und für den Kommandoprozessor benutzt. Nachfolgender Speicherauszug zeigt einen solchen Datenbereich:

```
-d 0:500 1B0
0000:0500  00 00 20 20 00 00 20 20-53 59 53 25 00 00 00 00 ..  ..  SYS%....
0000:0510  00 00 00 00 00 00 90 1B-21 00 02 00 55 40 00 00 ........!...U@..
0000:0520  4D 53 DF 02 25 02 09 2A-FF 50 F6 01 02 00 00 00 MS..%..*.P......
0000:0530  00 00 00 00 00 00 00 00-49 0E 0B 00 40 6F 00 00 ........ ..@o..
0000:0540  43 4F 4D 4D 41 4E 44 20-43 4F 4D 20 00 00 00 00 COMMAND COM ....
0000:0550  00 00 00 00 00 00 17 92-49 0E 19 00 EC 5D 00 00 ........ ....]..
0000:0560  44 4F 53 20 20 20 20 20-53 59 53 25 00 00 00 00 DOS     SYS%....
0000:0570  00 00 00 00 00 00 90 1B-21 00 25 00 55 40 00 00 ........!.%.U@..
0000:0580  4D 53 44 4F 53 20 20 20-20 20 10 00 00 00 00 00 MSDOS       .....
0000:0590  00 00 00 00 00 00 91 1B-21 00 2E 00 00 00 00 00 ........!.......
0000:05A0  44 4F 53 50 4C 55 53 20-20 20 10 00 00 00 00 00 DOSPLUS     .....
0000:05B0  00 00 00 00 00 00 92 1B-21 00 2F 00 00 00 00 00 ........!./.....
0000:05C0  47 45 4D 42 4F 4F 54 20-20 20 10 00 00 00 00 00 GEMBOOT     .....
0000:05D0  00 00 00 00 00 00 92 1B-21 00 30 00 00 00 00 00 ........!.0.....
0000:05E0  47 45 4D 41 50 50 53 20-20 20 10 00 00 00 00 00 GEMAPPS     .....
0000:05F0  00 00 00 00 00 00 93 1B-21 00 31 00 00 00 00 00 ........!.1.....
0000:0600  47 45 4D 44 45 53 4B 20-20 20 10 00 00 00 00 00 GEMDESK     .....
0000:0610  00 00 00 00 00 00 00 93 1B-21 00 32 00 00 00 00 00 ........!.2.....
0000:0620  47 45 4D 53 59 53 20 20-20 20 10 00 00 00 00 00 GEMSYS      .....
0000:0630  00 00 00 00 00 00 94 1B-21 00 34 00 00 00 00 00 ........!.4.....
0000:0640  49 4D 41 47 45 53 20 20-20 20 10 00 00 00 00 00 IMAGES      .....
0000:0650  00 00 00 00 00 00 95 1B-21 00 35 00 00 00 00 00 ........!.5.....
0000:0660  42 41 53 49 43 32 20 20-20 20 10 00 00 00 00 00 BASIC2      .....
0000:0670  00 00 00 00 00 00 00 95 1B-21 00 36 00 00 00 00 00 ........!.6.....
0000:0680  45 58 45 43 55 54 45 20-43 4F 4D 20 00 00 00 00 EXECUTE COM ....
0000:0690  00 00 00 00 00 00 D7 9D-06 0F F4 0D 1B 34 00 00 ............4..
```

Bild 1.10: Speicherauszug aus dem DOS- und IO.SYS-Bereich

Ab Adresse 0:500 beginnt der DOS-Datenbereich, an den sich bei Adresse 0:700 das Programm IO.SYS anschließt. Es ist deutlich zu sehen, daß der Speicherbereich ab 0:500 mit einem Auszug aus dem Inhaltsverzeichnis der Festplatte überschrieben wurde. Lediglich innerhalb der DOS-Daten finden sich einige Parameter (z.B. die Disk-Parameter-Tabelle ab Adresse 0:522), die durch DOS neu initialisiert wurden.

Der residente Teil von IO.SYS befindet sich noch im Speicher und wird auch nicht überschrieben. Dieser Teil bearbeitet die MS-DOS I/O-Aufträge und verwaltet die Peripheriegeräte auf der hardwarenahen Ebene.

1.10 Die Funktion des Programmes MSDOS.SYS (IBMDOS.COM)

Das Programm MSDOS.SYS bildet den Kern des DOS-Systems und wird oberhalb des IOSYS-Programmteils geladen. Dies erfolgt während des Startvorganges durch das Bootprogramm. Die Lage im Hauptspeicher ist abhängig von der DOS-Version und kann nicht allgemeingültig angegeben werden. Sobald die Initialisierung durch IO.SYS abgeschlossen ist, wird der DOS-Initialisierungsteil von MSDOS.SYS aufgerufen. Er baut die internen DOS-Arbeitstabellen auf, initialisiert die DOS-Interruptvektoren (INT 20 bis INT 27). Ab den DOS-Versionen 3.1 bis 6.0 initialisiert MSDOS.SYS zusätzlich die Interruptvektoren 0FH bis 3FH. Dann wird im niedrigsten freien Speicherbereich ein Program-Segment-Prefix; (PSP) für den MS-DOS Kommandoprozessor eingerichtet. Dann geht die Kontrolle wieder an das Programm IO.SYS zurück, welches nun den Kommandoprozessor (COMMAND.COM) lädt.

Der Initialisierungsteil von MSDOS.SYS wird nicht mehr gebraucht, so daß dieser Bereich teilweise als DOS-Datenpuffer genutzt wird. Der residente Teil von MSDOS.SYS enthält das MS-DOS-Interface zu den Anwenderprogrammen (INT 20 bis INT 27). Hierunter befinden sich die Dateiverwaltungsroutinen, die Datenaufbereitung (Blocking/ Deblocking) für File I/O, sowie interne Funktionen. Diese Funktionen lassen sich durch Anwenderprogramme ansprechen. Die Parameter finden sich in Registern oder Kontrollblöcken. Das Programm transferiert diese Informationen in BIOS-Aufrufe und wickelt den Auftrag ab. Eine genaue Beschreibung der Interrupts und Funktionen findet sich in den nachfolgenden Abschnitten.

1.11 Die Funktion des Programmes COMMAND.COM

Die Verbindung zwischen Anwender und dem DOS-Betriebssystem wird durch den Kommandoprozessor COMMAND.COM abgewickelt. Dieser besteht aus vier separaten Teilen:

Ein residenter Teil findet sich oberhalb der DOS-Puffer und Datenbereiche. Dieser Teil enthält die Interrupt-Service-Routinen für INT 22 (Terminate Adresse), INT 23 (Control-C-Handler) und INT 24 (Critical Error Handler).

Weiterhin ist ein Programm vorhanden, um den transienten Teil von COMMAND.COM bei Bedarf nachzuladen. Dieser transiente Teil befindet sich am oberen Ende des RAM-Bereiches und kann von Anwenderprogrammen überschrieben werden. Über eine Checksum-Prüfung läßt sich feststellen, ob Teile des Programmcodes überschrieben wurden. Sobald ein Anwender- oder ein externes DOS-Programm beendet wurde, wird diese Checksumme überprüft und der zerstörte Teil nachgeladen. Die Standard-DOS-Fehlerbehandlung (Abort, Retry, or Ignore) wird durch den residenten Teil (INT 24) abgewickelt. In den DOS-Versionen 3.0 bis 6.0 enthält der residente Teil neben der Routine zum Laden des transienten Teils von COMMAND.COM auch die EXEC-Funktion. Mit dieser Funktion lassen sich beliebige Programme laden und starten.

An den residenten Teil schließt sich ein Initialisierungsprogramm an, welches beim Programmstart aktiviert wird. Dieser Initialisierungsteil wertet z.B. die Datei AUTO-EXEC.BAT aus. Weiterhin bestimmt es das unterste freie Speichersegment, ab dem ein

Anwenderprogramm geladen werden kann. Diese Adresse liegt meist noch im Initialisierungsteil von COMMAND.COM, da dieser nicht länger benötigt wird und folglich von Anwenderprogrammen belegt werden kann.

Neben diesen zwei Teilprogrammen findet sich ein nicht residenter Teil von COMMAND.COM, der immer an die RAM-Obergrenze geladen wird. Bei Bedarf kann dieser Bereich durch Anwenderprogramme überschrieben werden. Sobald diese Programme beendet sind, lädt der residente Teil von COMMAND den transienten Teil nach. Es werden hier zwei Funktionen zur Verfügung gestellt:

Einmal findet sich hier der Kommandointerpreter, der den Systemprompt (z.B. C>) ausgibt. Dann werden die Anwendereingaben decodiert. Dieser Teil enthält auch alle internen DOS-Kommandos (DIR, DEL, COPY, etc.). Weiterhin findet sich hier der BATCH-File-Prozessor. Sofern ein externes Kommando erkannt wurde, wird die EXEC-Funktion aufgerufen, um das Programm zu laden und auszuführen.

Weiterhin findet sich hier in DOS 2.x eine Routine, um externe Kommandos (COM- oder EXE-Programme) zu laden und zu starten. Diese Routine übernimmt die Entrelativierung von EXE-Dateien, da diese in ihrer Urform nicht lauffähig sind. Der Lader befindet sich im höchsten Speicherbereich und wird durch die EXEC-Funktion aktiviert.

Die Größe des DOS-Kommandoprozessors (COMMAND.COM) ist abhängig von der DOS-Version.

DOS 2.1

In dieser Version ist die EXEC-Funktion im transienten Teil von COMMAND.COM enthalten, so daß 17 Kbyte permanent belegt werden.

DOS 3.0-6.0

Diese Versionen halten die EXEC-Funktion im residenten Teil von COMMAND.COM. Die DOS 3.0 Version benötigt ebenfalls ca. 17 Kbyte Speicher für den Kommandoprozessor, während ab DOS 3.1 zirka 23 Kbyte belegt werden.

2 Die BIOS-Funktionen

Zur Abwicklung der I/O-Anforderungen an die Hardware dient das BIOS (Basic Input Output System). Es gliedert sich in einen Teil, der im IBM-PC im ROM ab der Adresse FE00:0000 liegt und die hardwarenahen Funktionen übernimmt. Ein weiterer Teil wird durch die Dateien IO.SYS (MS-DOS) oder IBMBIO.COM (PC-DOS) direkt oberhalb des DOS-Datenbereichs geladen. Weiterhin blenden die Adapter (z.B. EGA-Karte) BIOS-Bereiche im Adressraum von C000H bis FDFFH ein.

Das BIOS stellt Funktionen wie die Ausgabe von Zeichen auf dem Bildschirm, einlesen des Tastaturpuffers, etc. zur Verfügung. Diese Funktionen werden per Interrupt aktiviert. Im Rahmen der Systemprogrammierung sind sie oft nützlich und werden deshalb nachfolgend kurz vorgestellt.

Es sei aber darauf hingewiesen, daß nicht alle BIOS-Funktionen bei jedem kompatiblen Personalcomputer implementiert wurden. Neuere Geräte enthalten aber aus Kompatibilitätsgründen meist BIOS-ROM's (AMI, PHÖNIX), die die beschriebenen Schnittstellen aufweisen.

2.1 Print-Screen-Funktion (INT 5)

Diese Routine wird jedesmal aktiviert, wenn die Taste µPrtSc- gedrückt wurde. Das Interruptsystem bleibt während der Abarbeitung freigegeben. Anhand des Status-Flags (Adr 0000:0500) wird geprüft, ob der letzte Aufruf erfolgreich beendet wurde. Ist das Flag <> 0, bricht die Routine ab. Das Modul verwendet den INT 10 zur Ausgabe der einzelnen Zeichen auf dem Bildschirm. Die benutzten Register (AX, BX, CX, DX, DS) werden beim Aufruf gesichert, während die restlichen Register zerstört werden.

2.2 BIOS-Bildschirmausgabe (INT 10)

Diese Funktion erlaubt die Ansteuerung des Bildschirmadapters zur Cursoreinstellung, Zeichenausgabe, Bildscroll etc.). Die Register CS, SS, DS, ES, BX, CX und DX bleiben beim Aufruf erhalten, während die restlichen Registerinhalte zerstört werden. An Hand der Information aus dem BIOS-Konfigurationswort ermittelt die Routine die Adresse des Bildschirmadapters und schreibt den Wert (B000,B800,...) in das Register ES. Ist im System eine EGA-Karte vorhanden, befindet sich das BIOS zur Bildschirmsteuerung auf dieser Karte. Beim Systemstart wird der entsprechende Vektor von der POST-Routine (Power-On-Systemstart) eingetragen (siehe Abschnitt über den Systemstart). Es gelten aber die gleichen Funktionsschnittstellen wie bei den nachfolgend beschriebenen Routinen. Die Auswahl der jeweiligen Unterfunktion wird durch das Register AH selektiert. Es sind folgende Funktionen implementiert.

Bildschirmmodus wählen (AH = 00H)

Dieser Aufruf ermöglicht es, den Bildschirmmodus (Mono, Color, Grafik) einzustellen. Der Wert des Registers AL selektiert den Modus.

```
Ö--------------------------------Ì
º      CALL:  INT 10             º
º                                º
º AH:  00H (Mode select)         º
º AL:  Bildschirmmodus           º
û--------------------------------À
º      RETURN                    º
º --                             º
Û--------------------------------ì
```

Die folgende Tabelle gibt die möglichen Einstellungen wieder.

```
Register° Bildschirmmodus
--------é-------------------------------------
AL = 0  ° Text 40 Zeichen x 25 Zeilen monochrom
AL = 1  ° Text 40x25 color
AL = 2  ° Text 80x25 monochrom
AL = 3  ° Text 80x25 color
AL = 4  ° Grafik 320x200 color
AL = 5  ° Grafik 320x200 monochrom
AL = 6  ° Grafik 640x200 monochrom
AL = 7  ° CRT Modus 80x25 monochrom (intern)
```

Tabelle 2.1: Auswahl des Bildschirmmodus per INT 10

Der Modus AL = 7 wird intern vom BIOS zur monochromen 80x25-Zeichen-Darstellung benutzt. Bei Aufruf der Funktion AH = 0 wird der Bildschirm gelöscht.

Cursor-Größe definieren (AH = 01H)

Dieser Aufruf dient dazu, die Größe des sichtbaren Cursors zu definieren. Die Form ist abhängig von der Zahl der Linien und wird durch das Register CX spezifiziert.

```
Startzeile     Ö----Ì
               °    °  Cursorfeld aus n Linien
Endzeile       Û----ì
```

Bild 2.1: Aufbau des Cursorfeldes

Der Cursor kann dabei maximal die Größe eines Zeichenfeldes einnehmen. Bei der Colordarstellung umfaßt ein Zeichen die Größe von 5x8 Punkten in einer 8x8-Matrix. Beim Monochromadapter ist die Größe auf 5x14 Punkte erweitert. Der Cursor läßt sich nun aus mehreren Zeilen innerhalb des Zeichenfeldes zusammensetzen.

```
Ö-------------------------------Ì
°      CALL:  INT 10            °
°                               °
° AH: 01H (Cursor Size)         °
° CH: Bit 0-4 Startzeile Cursor °
° CL: Bit 0-4 Endzeile Cursor   °
û-------------------------------À
°      RETURN                   °
° --                            °
Û-------------------------------ì
```

Beim Coloradapter liegen die Werte zwischen 0 und 7, während sie beim Monochromadapter zwischen 0 und 13 variieren dürfen.

Einen Blockcursor (Rechteck) erhält man mit den Werten:

```
CH  = 0  CL = 7  (Color)   CH = 0  CL = 13 (Mono)
```

Soll der Cursor durch zwei Striche am oberen und unteren Zeichenrand dargestellt werden, gilt die Einstellung:

```
CH = 7  CL = 0  (Color)   CH = 13  CL = 0 (Mono)
```

Mit den Werten:

```
CH = 6  CL = 7  (Color)   CH = 12  CL = 13 (Mono)
```

wird der Cursor als Unterstrich dargestellt. Diese unterschiedlichen Wertebereiche für den Monochrom- und Coloradapter sind bei der Softwareerstellung zu beachten.

Bit 5,6 in CH werden bei einigen Grafikadaptern zum Ein- und Ausblenden des Cursors benutzt. Es gilt folgende Vereinbarung:

```
Ö-------------------------------Ì
º Bit 6,5                      º
û-------------------------------À
º     0 0: Cursor blinkend     º
º     0 1: Cursor statisch     º
º     1 0: Cursor statisch     º
º     1 1: Cursor ausgeschaltet º
Û-------------------------------ì
```

Bit 7 sollte beim Aufruf zu 0 gesetzt werden.

Cursor positionieren (AH = 02H)

Der Cursor läßt sich mit dieser Funktion auf dem Bildschirm positionieren. Die Koordinaten werden durch das DX-Register angegeben. Die Position (0,0) befindet sich in der linken oberen Bildschirmecke.

```
Ö-------------------------------Ì
º      CALL:  INT 10           º
º                              º
º AH: 02H (Set Cursor)         º
º BH: Bildschirmseite          º
º DH: Zeilennummer             º
º DL: Spaltennummer            º
û-------------------------------À
º      RETURN                  º
º --                           º
º                              º
Û-------------------------------ì
```

Da die Colorkarte im Gegensatz zur Monochromkarte 16-Kbyte-Speicher enthält, ein Textbildschirm aber nur 4 Kbyte belegt, sind insgesamt 4 Bildschirmseiten à 24x80 Zeichen möglich. Durch den Inhalt des BH-Registers wird die Bildschirmseite definiert, für die die neue Cursorposition gilt. Das BIOS sieht im Datenblock insgesamt 8 Bildschirmseiten vor.

Dies resultiert aus der Tatsache, daß im 25x40-Zeichen-Modus insgesamt 8 Bildschirmseiten vorhanden sind. In der Praxis wird man jedoch meist die 25x80-Zeichen-Anzeige verwenden. Bei Anwahl des Grafikmodus muß der Wert von BH = 0 gesetzt werden, da hier nur eine Seite existiert.

Cursorposition ermitteln (AH = 03H)

Die Position des Cursors innerhalb einer Bildschirmseite läßt sich mit der Unterfunktion AH = 3 abfragen. Das Register BH spezifiziert dabei die Bildschirmseite (bei Grafik = 0).

```
Ö---------------------------------Ì
º      CALL : INT 10              º
º                                 º
º AH:  03H (Get Cursor)           º
º BH:  Bildschirmseite            º
û---------------------------------À
º      RETURN                     º
º                                 º
º DH:  Zeilennummer               º
º DL:  Spaltennummer              º
º CH:  Startzeile Cursor          º
º CL:  Endzeile Cursor            º
Û---------------------------------ì
```

Die Funktion liefert als Ergebnis Lage und Größe des Cursors zurück. Die Cursorgröße findet sich analog zur Funktion AH = 1 im CX-Register.

Position Lichtgriffel (AH = 04H)

Diese BIOS-Funktion ermöglicht die Positionsbestimmung eines angeschlossenen Lichtgriffels innerhalb des Bildschirms. Es werden folgende Parameter übergeben:

```
Ö---------------------------------Ì
º      CALL: INT 10               º
º                                 º
º   AH: 04H (Light Pen)           º
û---------------------------------À
º      Return:                    º
º                                 º
º AH:  0 keine Position gefunden  º
º      1 gültige Position         º
º DH:  Zeilenposition (Textmodus) º
º DL:  Spaltenposition (Textmodus)º
º CH:  Rasterzeile (Grafikmodus)  º
º BX:  Rasterspalte (Grafikmodus) º
Û---------------------------------ì
```

In der Praxis dürften die wenigsten Geräte mit Lichtgriffeln ausgestattet sein. Die BIOS-Funktion wurde beim IBM-PC vermutlich nur auf Grund der Möglichkeiten des Anzeigebausteins eingeführt. Bei kompatiblen PCs ist dieser Aufruf nicht immer implementiert.

Bildschirmseite selektieren (AH = 05H)

Befindet sich der Bildschirmadapter im Textmodus (siehe Funktion AH = 00H), läßt sich mit dieser Funktion die aktuelle Bildschirmseite umschalten. Der Wert des Registers AL bestimmt die Bildschirmseite.

```
Ö---------------------------------Ì
º                                 º
º   CALL:  INT 10                 º
º                                 º
º AH:  05H (Set Page)             º
º AL:  Bildschirmseite            º
û---------------------------------À
º      RETURN                     º
º  --                             º
Û---------------------------------ì
```

Im 40x25-Zeichen-Modus existieren 8 Seiten, während der 80x25-Modus nur 4 Seiten erlaubt.

Bildschirmfenster Scroll Up (AH = 06H)

Innerhalb der aktiven Bildschirmseite läßt sich im Textmodus ein Fenster einblenden, in dem mit dieser Funktion der Text nach oben gescrollt wird. Dabei läßt sich der Ausschnitt auf den kompletten Bildschirm vergrößern, um die Scrollfunktion auf die gesamte Seite auszudehnen. Im unteren Fensterbereich werden beim Scroll Leerzeilen eingefügt. Es sind folgende Register zu definieren:

```
Ö---------------------------------Ì
º      CALL : INT 10              º
º                                 º
º AH: 06H (Scroll Up)             º
º AL: Anzahl der Zeilen, um die   º
º     nach oben geschoben wird    º
º CH: Eckzeile oben links         º
º CL: Eckspalte oben links        º
º DH: Eckzeile unten rechts       º
º DL: Eckspalte unten rechts      º
º BH: Attribut der Leerzeile      º
û---------------------------------À
º      RETURN                     º
º  --                             º
----------------------------------ì
```

Der Bildschirm wird im 80x25-Modus in 25 Zeilen zu je 80 Spalten aufgeteilt.

```
      0 1  . . . Spalten   . . . . . 79
     Ö-Ú-Ú-------------------------Ú-Ú-Ì
    0 û-é-é                       -é-À
Z   1 û-é  Ö- - - - - - - - - -Ì     é-À
e   2 û                             º
i           º                 º
l                Fenster
e           º                 º
n
            Û- - - - - - - - - -ì
   23 û-é-                         é-é-À
   24 Û-Ù-Ù-Ù-----------------------Ù-Ù-ì
```

Bild 2.2: Bildschirmraster im 80x25-Textmodus

Mit den Werten in den Registern CX, DX werden die beiden Eckpunkte des ausgewählten Bildschirmfensters spezifiziert. Die Koordinaten (0,0),(24,79) setzen das Fenster über den gesamten Bildschirm. Das AL-Register gibt an, wie viele Zeilen nach oben gescrollt werden. Ist der Wert = 0, dann wird das Fenster gelöscht. Der gleiche Effekt tritt auf, wenn der Wert in AL der Zeilenzahl innerhalb des Fensters entspricht, da ja von unten Leerzeilen eingefügt werden. Mit dem Wert des Registers BH wird das Attribut der Leerzeile (blinkend, unterstrichen, invers etc.) festgelegt. Die Kodierung der Attribute wird bei der Unterfunktion (AH = 8) vorgestellt. Einige Implementierungen besitzen einen Bug, so daß das BP-Register zerstört wird.

Bildschirmfenster Scroll Down (AH = 07H)

Analog der Unterfunktion (AH = 6) existiert die Möglichkeit, einen Bildschirmausschnitt um mehrere Zeilen nach unten zu scrollen. Am oberen Fensterrand werden Leerzeilen eingefügt.

```
Ö--------------------------------Ì
º      CALL : INT 10             º
º                                º
º AH: 07H (Scroll Down)          º
º AL: Anzahl der Zeilen, um die  º
º     nach unten geschoben wird  º
º CH: Eckzeile Ecke oben links   º
º CL: Eckspalte Ecke oben links  º
º DH: Eckzeile Ecke unten rechts º
º DL: Eckspalte Ecke unten rechts º
º BH: Attribut der Leerzeile     º
û--------------------------------À
º      RETURN                    º
º  --                            º
Û--------------------------------ì
```

Es gelten die gleichen Bedingungen wie bei der Unterfunktion (AH = 6). Bei einigen Versionen wird BP beim Aufruf zerstört.

Zeichen und Attribute an Cursorposition lesen (AH = 08H)

Im Textmodus läßt sich das an der aktuellen Cursorposition abgespeicherte Zeichen, sowie das zugehörige Attributbyte lesen. Der Inhalt des BH-Registers spezifiziert dabei die Bildschirmseite. Das Ergebnis wird im AX-Register zurückgegeben.

```
Ö--------------------------------Ì
º      CALL : INT 10             º
º                                º
º AH: 08H (Get Char Attribut)    º
º BH: Bildschirmseite (Textmodus) º
û--------------------------------À
º      RETURN                    º
º AL: gelesenes Zeichen          º
º AH: Attributbyte (Textmodus)   º
Û--------------------------------ì
```

In diesem Zusammenhang soll nun auf die Kodierung des Attributbytes eingegangen werden.

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙÙÚÙ-ÙÚÙÙÚÙÙ-ÙÚÙì
 º Û-Ú-ì º Û---Ù--  Vordergrundfarbe
 º   º   Û--------  Intensitätsbit
 º   Û------------  Hintergrundfarbe
 Û----------------  Blink Bit
```

Bild 2.3: Kodierung des Attributbytes bei der Bildschirmausgabe

```
Ö-----------------------------------------------Ì
º Darstellung      b7  b6 b5 b4  b3  b2 b1 b0 º
û-----------------------------------------------À
º normal           b   0  0  0   i   1  1  1  º
º invers           b   1  1  1   i   0  0  0  º
º unterstrichen    b   0  0  0   i   0  0  1  º
º weiß & weiß      b   0  0  0   i   0  0  0  º
º schwarz & schw.  b   1  1  1   i   1  1  1  º
û-----------------------------------------------À
º normal : weißes Zeichen auf schwarzem         º
º          Hintergrund                          º
º invers : schwarzes Zeichen auf weißem         º
º          Hintergrund                          º
û-----------------------------------------------À
º b = 1   Zeichen blinkend                      º
º i = 1   erhöhte Intensität                    º
Û-----------------------------------------------ì
```

Tabelle 2.2: Attribute beim Monochromadapter

Ein am Bildschirm angezeigtes Zeichen besteht aus 2 Byte, dem Code für das Zeichen und auf der folgenden Adresse das dazugehörige Attributbyte. Dieses Byte gibt an, wie das Zeichen darzustellen ist (Intensität, unterstrichen, Farbe etc.). Für das Attributbyte ist folgende Kodierung festgelegt.

Das Zeichen blinkt, falls Bit 7 = 1 gesetzt wird. Weiterhin kann über Bit 3 entschieden werden, ob die Intensität erhöht (Bit 3 = 1) wird. Die weitere Darstellung hängt davon ab, ob ein Monochrom- oder Coloradapter zur Verfügung steht. Die Tabelle 2.2 gibt die Möglichkeiten für die Monochromdarstellung wieder.

```
7   R   G   B    I   R   G   B
°   Û---Ú---ì    Û-----Ú-----ì
°       °            Û-------- Vordergrundfarbe
°       Û---------------------- Hintergrundfarbe
Û------------------------------ Blinkbit

I R G B
0 0 0 0  Schwarz
0 0 0 1  Blau
0 0 1 0  Grün          Vorder- und
0 0 1 1  Cyanblau
0 1 0 0  Rot           Hintergrundfarbe
0 1 0 1  Violett
0 1 1 0  Braun
0 1 1 1  Weiß
1 0 0 0  Dunkelgrau
1 0 0 1  Hellblau
1 0 1 0  Hellgrün
1 0 1 1  Hellcyanblau
1 1 0 0  Hellrot       Vordergrundfarbe
1 1 0 1  Hellviolett
1 1 1 0  Gelb
1 1 1 1  Weiß
```

Tabelle 2.3: Kodierung der Colorkarte

Colorkarten-Kodierung

An Hand dieser Tabelle lassen sich nun verschiedene Darstellungskombinationen selektieren. Wird an Stelle der Monochromkarte ein Farbgrafikadapter verwendet, ergibt sich eine andere Belegung, da hier 16 Vorder- und 8 Hintergrundfarben wählbar sind. Die Farben werden dabei aus den Grundfarben Rot, Grün, Blau gemäß dem Schlüssel in Tabelle 2.3 kombiniert.

Die 16 Vordergrundfarben setzen sich im Grunde aus den 8 Hintergrundfarben zusammen. Durch das Intensitätsbit läßt sich aber eine zweite Palette mit hellen Farben erzeugen.

Zeichen und Attribut an Cursorposition schreiben (AH = 09H)

Diese Funktion erlaubt die Ausgabe von Zeichen in den Bildspeicher, wobei jedem Zeichen ein Attributbyte angehängt wird. Es gelten folgende Registerbelegungen:

```
Ö---------------------------------Ì
º      CALL :  INT 10             º
º                                 º
º AH:  09H (Set Char & Attribut)  º
º AL:  Zeichencode                º
º BH:  Bildschirmseite der Aus-   º
º      gabe (nur im Textmodus)    º
º BL:  Attributbyte des Zeichens  º
º CX:  Zahl der Kopien            º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Die Kodierung des Attributbytes wurde bereits bei der Funktion (AH = 08) vorgestellt. Der Wert im Register CX spezifiziert, wie oft das Zeichen ausgegeben wird. Der Wert 0003 schreibt das Zeichen ab der aktuellen Cursorposition 3 mal in die durch das Register BH selektierte Ausgabeseite. Bei der Ausgabe wird der Cursor automatisch mitgeführt. Im Graphikmode führt ein Wert von CX > Zahl der Spalten bis zum Zeilenende zu unvorhersagbaren Resultaten.

Zeichen an Cursorposition schreiben (AH = 0AH)

Für die Fälle, wo lediglich Zeichen ohne Attributbyte ausgegeben werden müssen, läßt sich diese Funktion benutzen. Es gilt folgende Registerbelegung:

```
Ö---------------------------------Ì
º      CALL :  INT 10             º
º                                 º
º AH:  0AH (Write Char)           º
º AL:  Zeichencode                º
º BH:  Bildschirmseite der Aus-   º
º      gabe (nur im Textmodus)    º
º CX:  Zahl der Kopien            º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Das im Bildschirmspeicher gesetzte Attributbyte bleibt beim Schreibvorgang erhalten. Schreib-Lese-Vorgänge im Grafikmodus lassen sich ebenfalls über die gerade vorgestellten Funktionen abwickeln. Die Zeichengenerierung erfolgt aber aus dem BIOS-ROM. Allerdings lassen sich so nur die ersten 128-ASCII-Zeichen ausgeben.

Sollen Sonderzeichen angezeigt werden, ist der Grafikvektor des INT 1F auf eine Tabelle mit den entsprechenden Bitmustern umzulenken. Zumindest beim IBM-ROM (Vers. 24. 4. 81) tritt ein weiteres Problem auf. Falls der Wiederholfaktor in CX größer als 1 ist, werden die Zeichen nur in der aktuellen Zeile korrekt ausgegeben. Eine Fortsetzung in der Folgezeile erbringt dann keine korrekten Resultate mehr.

Farbpalette für Grafik setzen (AH = 0BH)

Im Grafikmodus lassen sich Vorder- und Hintergrundfarbe definieren. Es gilt folgende Registerbelegung:

```
Ö--------------------------------Ì
º      CALL :  INT 10            º
º                                º
º AH:  0BH (Select Palette)      º
º BH:  Code der zu setzenden     º
º      Farbpalette               º
º BL:  Farbwert (0 - 15) für     º
º      diese Palette             º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Über das BH-Register wird also spezifiziert, ob der Farbwert für die Vordergrund- oder Hintergrundfarbe gilt. Mit dem Wert BH = 0 wird die Hintergrundfarbe selektiert. In BL ist ein Wert von 0-15 aus der Farbscala (siehe Tabelle 2.3) zu definieren. Um die Vordergrundfarbpalette zu definieren, ist BH = 1 zu setzen.

Mit dem Register BL wird dann die Farbpalette spezifiziert:

```
Ö-----Ú----------------------------------------Ì
º BL  º Palette                                º
û-----é----------------------------------------À
º  0  º Satz mit Grün, Rot, Braun, (Gelb)      º
û-----é----------------------------------------À
º  1  º Satz mit Cyan, Magenta, Weiß           º
Û-----Ù----------------------------------------ì
```

Tabelle 2.4: Farbpaletten im Colormodus

Grafikpunkt setzen (AH = 0CH)

Im Grafikmodus lassen sich einzelne Punkte mit einer bestimmten Farbe setzen. Die Koordinaten dieses Punktes werden in der Notation (Zeile, Spalte) angegeben. Die folgende Tabelle gibt die gültigen Werte für die entsprechenden Farben wieder:

```
Ö-----------Ú---------Ú--------------Ì
º Auflösung º  Zeile  º  Spalte      º
û-----------é---------é--------------À
º 320x200   º 0-199   º  0-319       º
û-----------é---------é--------------À
º 640x200   º 0-199   º  0-639       º
Û-----------Ù---------Ù--------------ì
```

Tabelle 2.5: Werte für Zeile und Spalte im Grafikmodus

Die Position (0,0) liegt in der linken oberen Bildschirmecke.

Um den Punkt zu setzen, sind die folgenden Register zu definieren.

```
Ö--------------------------------Ì
º      CALL :  INT 10            º
º                                º
º AH:  0CH  (Set Pixel)          º
º DX:  Zeile                     º
º CX:  Spalte                    º
º AL:  Bit 0 - 6 : Farbe         º
º      Bit 7 = 1 Verknüpfung des º
º      Farbwertes über XOR mit   º
º      dem Punkt im RAM          º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Interessant ist Bit 7 im AL-Register. Falls der vorgegebene Farbwert mit dem Wert im Bildschirmspeicher übereinstimmt, läßt sich durch Setzen des Bit 7 = 1 der angegebene Punkt löschen (XOR-Verknüpfung).

Grafikpunkt lesen (AH = 0DH)

Im Grafikmodus lassen sich einzelne Punkte mit dieser Funktion lesen. Dies ist z.B. dann erforderlich, falls ein Punkt gelöscht werden soll (AH = 6) und die Farbe nicht bekannt ist. Es sind die folgenden Register zu definieren.

```
Ö--------------------------------Ì
º      CALL : INT 10             º
º                                º
º AH:  0DH  (Get Pixel)          º
º DX:  Zeile                     º
º CX:  Spalte                    º
º AL:  Bit 0 - 6 : Farbe         º
û--------------------------------À
º      RETURN                    º
º AL:  Farbwert des Punktes      º
Û--------------------------------ì
```

Der Farbwert wird im AL-Register zurückgegeben.

Zeichen ausgeben (AH = 0EH)

Mit dieser Funktion wird ein Teletype-Interface für die Bildschirmausgabe simuliert. Pro ausgegebenem Zeichen wird der Cursor um eine Stelle weiter bewegt. Die Funktion erwartet folgende Eingaben.

```
Ö--------------------------------Ì
º      CALL  : INT 10            º
º                                º
º AH:  0EH  (Write Char)         º
º AL:  Zeichen                   º
º BL:  Vordergrundfarbe im       º
º      Grafikmodus               º
º BH:  Ausgabeseite im           º
º      Textmodus                 º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Der Bildschirmmodus (25x40 oder 25x80 Zeichen) muß separat über die Funktion AH = 0 gesetzt werden.

Bildschirmstatus lesen (AH = 0FH)

Als Komplement zur Funktion AH = 0 (setze Bildschirmmodus) läßt sich hier der Modus wieder auslesen. Die Funktion gibt folgende Werte zurück.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  0FH  (Read Status)         º
û---------------------------------À
º      RETURN                     º
º AL:  Bildschirmmodus            º
º AH:  Bildschirmspalten (40 /80) º
º BH:  aktuelle Bildschirmseite   º
Û---------------------------------ì
```

Die Kodierung des Bildschirmmodus wurde bereits bei der Funktion AH = 0 vorgestellt. Bei der Abfrage kann auch der Modus 7 (interne 80x25 Monochromdarstellung) auftreten.

Zeichenkette ausgeben (AH = 13H)

Die Aufrufe AH = 10H, 11H und 12H sind bisher nicht belegt. Bei einigen BIOS-Versionen existiert allerdings noch der Aufruf 13H, um ganze Zeichenketten auszugeben. Es gilt folgende Aufrufkonvention.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  13H  (Write String)        º
º AL:  Untercode                  º
º CX:  Stringlänge in Bytes       º
º DX:  Cursorposition             º
º BH:  Bildschirmseite            º
º BL:  Attributcode               º
º ES:BP Anfangsadresse String     º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Der Aufruf kennt mehrere Unterfunktionen, die über den Wert im AL-Register selektiert werden.

AL = 0

Die Zeichenkette enthält nur die auszugebenden Zeichen. Im Register BL wird das Attributbyte übergeben. Das Attribut gilt dann für den gesamten String. Die Ausgabe erfolgt ab der aktuellen Cursorposition. Zeichen wie CR, LF, BS und Bell werden innerhalb des Strings als Steuerzeichen für die Cursorbewegung interpretiert. Das DX-Register bleibt bei diesem Aufruf unbelegt.

AL = 1

Die Zeichenkette enthält ebenfalls nur die auszugebenden Zeichen, wobei das Attributbyte per BL-Register gesetzt wird und für den ganzen String gilt. Bei diesem Aufruf wird der Cursor vor der Ausgabe auf die in DX angegebene Position gesetzt.

AL = 2

In der Zeichenkette findet sich immer ein Zeichen mit dem jeweiligen Attribut im folgenden Byte:

```
<Zeichen, Attribut, Zeichen, Attribut, . . . , Zeichen, Attribut>
```

Damit läßt sich jedes Zeichen mit Attributen belegen. Das BL-Register wird bei diesem Aufruf nicht benutzt. Die Funktion gibt den String an der aktuellen Cursorposition aus, so daß das Register DX ebenfalls nicht belegt ist.

AL = 3

Wie beim Aufruf AL = 2 finden sich im String sowohl Zeichen, als auch die zugehörigen Attribute. Das BL-Register bleibt unbenutzt. Der Wert in DX setzt den Cursor vor der Ausgabe auf die spezifizierte Position.

Weitere Informationen über die EGA-, VGA-BIOS-INT 10-Aufrufe finden sich im Kapitel über die EGA-/VGA-Erweiterungen.

Microsoft Maustreiber EGA Support (AH = F0H)

Der Aufruf erlaubt ein Register der EGA-Chips auszulesen. Der Wert wird bei der Rückgabe in BL übergeben. Der Code im Register BL definiert beim Aufruf die Registernummer. Der Inhalt von DX definiert den Gruppenindex.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  F0H  (Read one Register)   º
º BL:  Register Nummer            º
º BH:  00                         º
º DX:  Index Gruppe               º
û---------------------------------À
º      RETURN                     º
º BL: Daten                       º
Û---------------------------------ì
```

Für den Gruppenindex gelten folgende Belegungen:

```
Registergruppen
00H  CRT Controller (25 Register) 3B4H Mono  Mode
                                  3D4H Color Mode
08H  Sequencer (5 Register) 3C4H
10H  Graphik Controller (9 Register)   3CEH
18H  Attribut Controller (20 Register) 3C0H

Einzelregister
20H  Ausgaberegister 3C2H
28H  Controlregister (3BAH Mono Mode, 3DA Color Mode)
30H  Graphik 1 Positions Register 3CCH
38H  Graphik 2 Positions Register 3CAH
```

Die Belegung der Registernummer ist allerdings nicht genau bekannt.

Microsoft Maustreiber EGA Support (AH = F1H)

Der Aufruf erlaubt ein Register der EGA-Chips zu beschreiben. Beim Aufruf eines Einzelregisters wird der Wert in BL übergeben.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  F1H  (Write one Register)  º
º BL:  Register Nummer            º
º BH:  Ausgabewert                º
º DX:  Index Gruppe               º
û---------------------------------À
º      RETURN                     º
º BL: Daten                       º
Û---------------------------------ì
```

Beim Aufruf eines Registers einer Gruppe, enthält BL die Registernummer und BH den Wert.

Microsoft Maustreiber EGA Support (AH = F2H)

Der Aufruf erlaubt es, mehrere Register des EGA-Chips zu lesen. Die Werte werden in den mit ES:BX übergebenen Puffer gespeichert.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  F2H  (Read Register Range) º
º CH:  Register Startnummer       º
º CL:  Registerzahl (> 1)         º
º DX:  Index Gruppe               º
º ES:BX Pufferadresse             º
û---------------------------------À
º      RETURN                     º
º ----                            º
Û---------------------------------ì
```

Für den Code in DX gilt dabei:

```
00H  CRTC (3B4H Mono Mode, 3D4H Color Mode)
08H  Sequencer 3C4H
10H  Graphik Controller 3CEH
18H  Attribut Controller 3C0H
```

Das genaue Format des Aufrufes ist allerdings nicht bekannt.

Microsoft Maustreiber EGA Support (AH = F3H)

Der Aufruf erlaubt es, mehrere Register des EGA-Chips zu beschreiben.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  F3H  (Write Register Range)º
º CH:  Register Startnummer       º
º CL:  Registerzahl (> 1)         º
º DX:  Index Gruppe               º
º ES:BX Pufferadresse             º
û---------------------------------À
º      RETURN                     º
º ----                            º
Û---------------------------------ì
```

Die Werte sind in dem mit ES:BX übergebenen Puffer abzuspeichern. Es gilt die gleiche Belegung für DX wie bei der Funktion F2H.

Microsoft Maustreiber EGA Support (AH = F4H)

Der Aufruf erlaubt mehrere Register der EGA-Chips auszulesen.

```
Ö--------------------------------Ì
º      CALL  :  INT 10           º
º                                º
º AH:  F4H  (Read  Register Set)  º
º CX:  Register Zahl              º
º ES:DX Adresse Buffer            º
û--------------------------------À
º      RETURN                    º
º ----                           º
Û--------------------------------ì
```

Die Werte werden bei der Rückgabe in den durch ES:DX adressierten Puffer gespeichert.

Für den Puffer gilt folgende Belegung:

```
Registergruppen
00H  CRTC (3B4H Mono Mode, 3D4H Color Mode)
08H  Sequencer (3C4H)
10H  Graphik Controller (3CEH)
18H  Attribut Controller (3C0H)

Einzelregister
20H  Ausgaberegister 3C2H
28H  Controlregister (3BAH Mono Mode, 3DA Color Mode)
30H  Graphik 1 Positions Register 3CCH
38H  Graphik 2 Positions Register 3CAH
```

Die Zahl der Registernummern ist allerdings nicht genau bekannt.

Microsoft Maustreiber EGA Support (AH = F5H)

Der Aufruf erlaubt mehrere Register der EGA-Chips zu beschreiben.

```
Ö--------------------------------Ì
º      CALL  :  INT 10           º
º                                º
º AH:  F1H  (Write  Register Set) º
º CX:  Registerzahl               º
º ES:DX Adresse Buffer            º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Es gelten die Aufrufkonvention wie bei der Funktion F4H.

Microsoft Maustreiber EGA Support (AH = F6H)

Der Aufruf stellt die ursprünglichen Werte wieder ein.

```
Ö--------------------------------Ì
º      CALL  :  INT 10           º
º                                º
º AH:  F6H  (Revert to default)  º
û--------------------------------À
º      RETURN                    º
º ----                           º
Û--------------------------------ì
```

Microsoft Maustreiber EGA Support (AH = F7H)

Der Aufruf erlaubt es, die Register mit den Werten der Tabelle zu initialisieren. Die Adresse der Tabelle wird in ES:DX übergeben.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH: F7H (Define default Table)  º
º DX: Port Nummer                 º
º ES:BX Tabelle                   º
û---------------------------------À
º      RETURN                     º
º ----                            º
Û---------------------------------ì
```

Für den Code in DX gilt dabei:

```
Registergruppen
00H CRT Controller (25 Register) 3B4H Mono Mode
                                 3D4H Color Mode
08H Sequencer (5 Register) 3C4H
10H Graphik Controller (9 Register)   3CEH
18H Attribut Controller (20 Register) 3C0H

Einzelregister
20H Ausgaberegister 3C2H
28H Controlregister (3BAH Mono Mode, 3DA Color Mode)
30H Graphik 1 Positions Register 3CCH
38H Graphik 2 Positions Register 3CAH
```

Das genaue Format des Aufrufes ist allerdings nicht bekannt.

Microsoft Maustreiber EGA Support (AH = FAH)

Der Aufruf erlaubt die Prüfung, ob der Treiber aktiv ist.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  FAH  (Interrogate Driver)  º
º BX:  0000                       º
û---------------------------------À
º      RETURN                     º
º BX: 0000 Treiber nicht da       º
º ES:BX Pufferadresse             º
º                                 º
Û---------------------------------ì
```

Ist BX nach dem Aufruf = 0, ist kein Treiber aktiv. Andernfalls enthält der Puffer (ES:BX) die EGA Register Interface Versions Nummer. Byte 0 enthält die Hauptversionnummer, während Byte 2 die Unterversionsnummer enthält.

Die Funktionen des INT 10 werden durch das BIOS der Graphikkarte (und g.g.f. den Maustreiber) zur Verfügung gestellt. Ergänzende EGA/VGA-Funktionen zum INT 10 werden in Kapitel 13 beschrieben.

2.3 Hardwarekonfiguration (INT 11)

Beim Booten des Systems wird die aktuelle Hardwarekonfiguration ermittelt und im BIOS-Datenbereich eingetragen. Über den INT 11 läßt sich dieser Konfigurationswert auslesen. Es sind keine Register zu setzen.

```
Ö--------------------------------Ì
º      CALL :  INT 11            º
º                                º
º    (Get Hardware Konfiguration) º
û--------------------------------À
º      RETURN                    º
º AX:  Hardwarekonfiguration     º
Û--------------------------------ì
```

Das Ergebnis wird im AX-Register zurückgegeben. Die Hardwarekonfiguration ist gemäß Bild 2.4 kodiert.

Die Angabe der Speichergröße auf der Hauptplatine (Bits 2, 3) ist historisch zu sehen, da heute kein PC mehr mit 64 Kbyte bestückt wird. Andere BIOS-Versionen können eine abweichende Kodierung verwenden. Bei einigen Versionen sind die Bits sogar unbelegt. Auch Bit 12 (Spieleadapter) ist oft nicht benutzt. In neueren BIOS-Versionen zeigt Bit 1 die Präsenz eines Arithmetikprozessors an.

```
15   13  11  9 8 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ø-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙÚÙÚÙÚÙÚÙÚÙ-ÙÚÙÚÅÚÙÚÙÚÙÚÙÚÙÚÙÚÙÚÙÚÙì   0 = kein Diskettenlaufwerk
 ÛÚÙì º º Û-Ú-ì º ÛÚÙì ÛÚÙì ÛÚÙì º Û--  1 = Disklaufwerk vorhanden
  º  º º   º   º  º   º   º  º
  º  º º   º   º  º   º   º  Û----  Coprozessor, oder unbelegt
  º  º º   º   º  º   º   º
  º  º º   º   º  º   º   º         Speicher auf der Hauptplatine
  º  º º   º   º  º   º   Û-------  00 = 16 Kbyte, 01 = 32 Kbyte
  º  º º   º   º  º   º             10 = 48 Kbyte, 11 = 64 Kbyte
  º  º º   º   º  º   º
  º  º º   º   º  º   Û-----------  Videomodus
  º  º º   º   º  º                 00  unbelegt
  º  º º   º   º  º                 01  40x25 S/W auf Colorkarte
  º  º º   º   º  º                 10  80x25 S/W auf Colorkarte
  º  º º   º   º  º                 11  80x25 S/W auf Monokarte
  º  º º   º   º  º
  º  º º   º   º  º                 Zahl der Diskettenlaufwerke
  º  º º   º   º  Û---------------  00 = 1, 01 = 2, 10 = 3, 11 =4
  º  º º   º   º                    (nur gültig, falls Bit1 = 1)
  º  º º   º   º
  º  º º   º   Û------------------  x unbelegt
  º  º º   º
  º  º º   Û----------------------  Zahl d. seriellen Schnittst.
  º  º º
  º  º Û---------------------------  1 Spieladapter vorh., oder
  º  º                              0 unbelegt
  º  Û-----------------------------  x unbelegt
  º
  Û-------------------------------  Zahl der parallelen Schnittstellen
```

Bild 2.4: Kodierung des Konfigurationswertes

2.4 Speichergröße ermitteln (INT 12)

Diese BIOS-Funktion ermittelt die Speichergröße, die beim Systemstart festgestellt wurde.

```
Ö---------------------------------Ì
º      CALL :  INT 12             º
º                                 º
º --  (Get RAM Size)              º
û---------------------------------À
º      RETURN                     º
º AX:  Speichergröße in Kbytes    º
Û---------------------------------ì
```

Das Ergebnis wird im AX-Register zurückgegeben. Der Wert spezifiziert die Größe in Kbyte. Allerdings kann es bei kompatiblen BIOS-ROMs zu Problemen kommen, falls andere Kodierungen verwendet wurden.

2.5 Disketten- und Festplatten-Steuerung (INT 13)

Der BIOS-Interrupt 13 wird teilweise durch den Disketten-/Festplattencontroller verwaltet und bietet einige Low-Level-Funktionen zur Steuerung der Disketten- und Festplattenlaufwerke. Er erlaubt das Lesen, Schreiben und Überprüfen von Einzelsektoren, sowie das Formatieren einer Diskette/Festplatte. Ähnlich wie beim INT 10 steuert der Inhalt des AH-Registers die Unterfunktion. Allerdings sei noch auf eine Besonderheit hingewiesen. Ursprünglich war nur die Unterstützung von mehreren Diskettenlaufwerken vorgesehen. Im Laufe der Entwicklung mußten aber auch Festplatten eingesetzt werden. Die Funktionen zur Unterstützung dieser Festplatten befinden sich meist in einem eigenen BIOS-ROM auf dem Festplattencontroller. Das BIOS durchsucht beim Systemstart den Bereich von C000:0000 bis E000:0000 auf zusätzliche BIOS-ROMs. Befindet sich dort ein BIOS für eine Festplatte, wird die Initialisierungsroutine dieses BIOS aktiviert, die dann den INT 13-Vektor auf eigene Routinen umsetzt. Der ursprüngliche Vektor auf die Diskettenfunktionen wird dann auf den INT 40 umgelegt.

Da aber der INT 13 für beide Speichermedien benutzt wird, muß ein Selektionskriterium zur Auswahl der jeweiligen Einheit vorhanden sein. Da auch die Kompatibilität zu älteren BIOS-Versionen erhalten bleiben mußte, griffen die Entwickler zu einem Trick. Das Register DL gibt den Laufwerkstyp an und wurde früher bei allen Aufrufen auf Werte < 80H gesetzt. In den BIOS-Routinen zur Unterstützung der Festplatte wird deshalb der Inhalt von DL untersucht. Ist er kleiner als 80H, bezieht sich der Aufruf auf die Diskettenlaufwerke und ein INT 40 wird ausgelöst. Andernfalls bezieht sich der Aufruf auf die Festplatte, deren Routinen im ROM stehen. Damit ergeben sich allerdings kleinere Unterschiede beim Aufruf der INT 13-Funktionen.

Disk-System zurücksetzen (AH = 00H)

Mit diesem Aufruf läßt sich der Controller zurücksetzen, falls eine Operation abgebrochen werden soll oder falls ein Fehler auftritt.

```
Ö-------------------------------Ì
º      CALL :  INT 13           º
º                               º
º AH: 00H (Reset Disk)          º
º DL: Laufwerkstyp              º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º                               º
Û-------------------------------ì
```

Falls keine Festplatte im System vorhanden ist, bezieht sich der Aufruf immer auf die Diskettenlaufwerke. In diesem Fall ist das Register DL ohne Bedeutung. Findet sich eine Platte im System, wird bei DL < 80H ein INT 40 ausgelöst, der die Einsprungadresse der Diskettenroutinen enthält. Werte >= 80H beziehen sich auf das BIOS der Festplatte.

Status lesen (AH = 01H)

Der Status des Controllers nach der letzten Operation wird ausgelesen und im AL-Register zurückgegeben. Es gilt folgende Aufrufkonvention:

```
Ö-------------------------------Ì
º      CALL :  INT 13           º
º                               º
º AH: 01H (Get Status)          º
º DL: Laufwerktyp               º
û-------------------------------À
º      RETURN                   º
º AX: Status des Controllers    º
Û-------------------------------ì
```

Besitzt das System nur Diskettenlaufwerke, wird das DL-Register nicht belegt. Andernfalls spezifiziert es mit DL < 80H die Diskettenlaufwerke und sonst die Platte.

Der Status wird bei der Funktion AH = 1 als Byte im Register AX zurückgegeben. Welche Registerhälfte belegt ist, hängt von der BIOS-Version ab. Beim IBM-PC ohne Festplatte findet sich der Wert in AL. Neue Versionen unterscheiden zwischen Diskette und Platte. Bei Diskettenlaufwerken findet sich das Statusbyte in AH, während die Platte es im AL-Register zurückgibt. Das Statusbyte besitzt folgende Belegung:

```
Ö-------------------------------------------------------------Ì
° 7 6 5 4 3 2 1 0                                             °
°Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì                                             °
°ÛÚÙÚÙÚÙÚÙÚÙÚÙÚÙÚì   Operation nicht durchgeführt               °
° ° ° ° ° ° ° ÛÚì    01 ungültiger Befehl, oder Versuch über eine °
° ° ° ° ° ° °  Û--      64-Kbyte-Grenze zu übertragen (Bit 3 = 1) °
° ° ° ° ° ° °        10 Adreßmarke nicht gefunden               °
° ° ° ° ° ° °        11 Write auf eine schreibgesch. Diskette   °
° ° ° ° ° ° °                                                 °
° ° ° ° ° ° Û-----   angegebener Sektor nicht gefunden          °
° ° ° ° ° °                                                   °
° ° ° ° ° Û-------   DMA-Überlauf, oder 64-Kbyte-Grenze erreicht, °
° ° ° ° °            falls Bit 1 = 1 ist.                       °
° ° ° ° °                                                     °
° ° ° ° Û---------   CRC-Fehler beim Lesen                      °
° ° ° °                                                       °
° ° ° Û-----------   Controllerfehler                           °
° ° °                                                         °
° ° Û-------------   SEEK-Operation erfolglos                   °
° °                                                           °
° Û---------------   Time-Out-Fehler des Laufwerks              °
°                                                             °
û-------------------------------------------------------------À
°                      Status-Codes                           °
û-------------------------------------------------------------À
°  00H     successful completion                              °
°  01H     bad command                                        °
°  02H     address mark not found                             °
°  03H     tried to write on write-protected disk             °
°  04H     sector not found                                   °
°  05H     reset failed (hard disk)                           °
°  06H     diskette removed or changed                        °
°  07H     bad parameter table (hard disk)                    °
°  08H     DMA overrun                                        °
°  09H     attempt to DMA across 64K boundary                 °
°  0AH     bad sector detected (hard disk)                    °
°  0BH     bad track detected (hard disk)                     °
°  0CH     unsupported track                                  °
°  0DH     invalid number of sectors on format (hard disk)    °
°  0EH     control data address mark detected (hard disk)     °
°  0FH     DMA arbitration error (hard disk)                  °
°  10H     bad CRC/EEC on read                                °
°  11H     data ECC corrected                                 °
°  20H     controller failure                                 °
°  40H     seek failed                                        °
°  80H     timeout                                            °
°  AAH     drive not ready (hard disk)                        °
°  BBH     undefined error (hard disk)                        °
°  CCH     write fault     (hard disk)                        °
°  E0H     status error    (hard disk)                        °
°  FFH     sense operation failed (hard disk)                 °
Û-------------------------------------------------------------ì
```

Bild 2.5: Belegung des Floppy- und Hard-Disk-Statusbyte;s

Die Bits können einerseits auf Grund von Hardwarefehlern, oder bei falschen Aufrufen der Diskettenfunktionen, gesetzt werden. Der Versuch, den Sektor 20 einer (360 Kbyte) Diskette zu lesen, bringt den Fehler Bit 2 (Sektor nicht gefunden), da eine Diskette maximal 9 Sektoren besitzt.

Im BIOS-Datenbereich finden sich weitere Informationen über den Status des Diskettenlaufwerkes (Tabelle 2.6).

Adresse 0:x	Bedeutung
0490	Drive 0 - 90 Medium Status (zusätzl. Disk Daten)
0491	Drive 1 - 91 Medium Status (zusätzl. Disk Daten)
0492	Drive 0 - 90 Start Status (zusätzl. Disk Daten)
0493	Drive 1 - 91 Start Status (zusätzl. Disk Daten)
0494	Drive 0 - 90 aktueller Zylinder
0495	Drive 1 - 91 aktueller Zylinder

Tabelle 2.6: Disk-Statusbyte; im BIOS-Datenbereich

Für das *Medium Status Feld* gilt dabei folgende Kodierung:

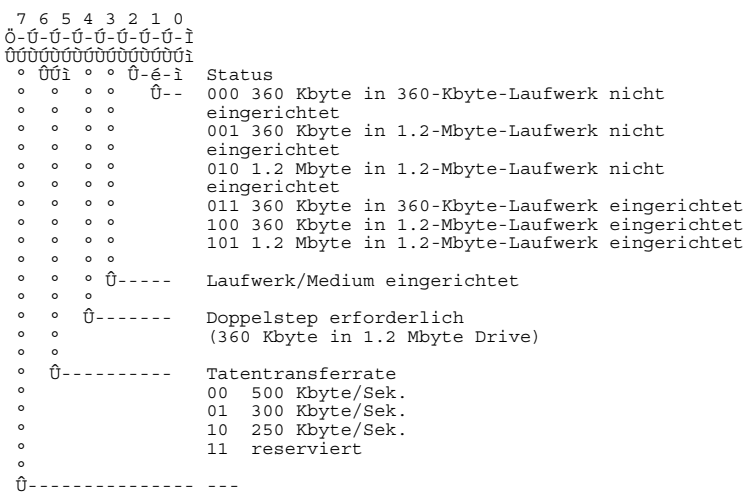


Bild 2.6: Belegung des Medium-Statusbytes bei Disketten

Diese Kodierung wurde aber erst bei Systemen mit 1,2-Mbyte-Laufwerken eingeführt und ist damit nicht immer gültig.

Sektor Read (AH = 02H)

Mit diesem Befehl läßt sich ein absoluter Sektor von einer Diskette/Platte lesen. Jedes Medium wird in einzelne Sektoren aufgeteilt, die in konzentrischen Kreisen (Spuren) aufgebracht werden. Die Zahl der Sektoren pro Spur ist abhängig von der Aufzeichnungsdichte.

Kbyte	Spuren/Seite	Seiten	Sektor/Spur	Sektoren	Bemerkungen
160	40	1	8	320	5 1/4 Zoll
180	40	1	9	360	5 1/4 Zoll
320	40	2	8	640	5 1/4 Zoll
360	40	2	9	720	5 1/4 Zoll
1200	80	2	15	2400	5 1/4 Zoll

Tabelle 2.7: Sektoraufteilung einzelner Diskettenformate

Um einen solchen Sektor zu lesen, müssen die folgenden Register vor dem Aufruf initialisiert werden:

```
CALL : INT 13

AH: 02H (Sektor Read)
DL: Drive-Nummer
DH: Kopfnummer
CH: Spur-(Zylinder-)Nummer
CL: Sektornummer
AL: Zahl der Sektoren
ES:BX  Adresse Lesepuffer

RETURN

CY: 0
    kein Fehler
CY: 1
AH: Fehlercode
```

Im Register DL wird die Laufwerksnummer (0 = A:, 1 = B: etc.) übergeben. Da die Funktion den Inhalt auf Gültigkeit überprüft, sind nur bestimmte Werte zulässig. Soll die Festplatte angesprochen werden, sind die Eingaben 80H und 81H erlaubt. Bei Disketten haben die Werte 00H und 01H Gültigkeit. DH enthält die Kopfnummer (Diskette 0-1, Platte 0-15), von der gelesen werden soll. Dieser Wert wird nicht überprüft. In das Register CH wird die Spurnummer geladen, während CL die Sektornummer innerhalb der Spur enthält. Bei Festplatten spricht man allerdings nicht mehr von einer Spur, sondern von einem Zylinder. Hier ergibt sich auch die Abweichung, daß die Zylindernummer 10 Bit umfaßt, wobei die unteren 8 Bit sich im Register CH (Spurnummer) befinden. Die beiden höherwertigen Bits der Zylindernummer werden in den oberen Bits des Registers CL (Sektornummer) übergeben. Die Sektornummern reichen in Abhängigkeit vom Medium nur von 1 bis 17, belegen also kein ganzes Byte. Beide Werte werden nicht überprüft. Im Register AL ist die Zahl der zu lesenden Sektoren anzugeben. Es können immer nur logisch aufeinanderfolgende Sektoren gelesen werden. Der Wert wird nicht überprüft. Weiterhin muß ein Puffer zur Aufnahme der gelesenen Werte angelegt werden, dessen Anfangsadresse als Zeiger im Registerpaar ES:BX übergeben wird.

Das Carry-Flag zeigt an, ob beim Aufruf der Funktion ein Fehler auftrat. Bei gesetztem Carry-Flag findet sich im Register AH ein Fehlercode, dessen Bedeutung bereits bei der Funktion (AH = 1 Status lesen) besprochen wurde. Im Fehlerfall ist anschließend das Laufwerk mit dem Aufruf AH = 0 zurückzusetzen. Anschließend kann der Leseaufruf wiederholt werden, um eventuelle Probleme durch die Motor-Hochlaufzeit auszuschließen.

Sektor Write (AH = 03H)

Analog der Lesefunktion existiert die Möglichkeit, einen absoluten Sektor auf die Diskette/ Festplatte zu schreiben. Hierzu sind die Register wie bei der Funktion AH = 02 zu setzen.

```
Ö--------------------------------Ì
º      CALL  :  INT 13           º
º                                º
º AH: 03H (Sektor Write)         º
º DL: Drive-Nummer               º
º DH: Kopfnummer                 º
º CH: Spur-(Zylinder-)Nummer     º
º CL: Sektornummer               º
º AL: Zahl der Sektoren          º
º ES:BX  Adresse Datenpuffer     º
û--------------------------------À
º      RETURN                    º
º CY: 0                          º
º     kein Fehler                º
º CX: Zahl der 512-Byte-Blocks,  º
º DX: die geschrieben wurden     º
º CY: 1                          º
º AH: Fehlercode                 º
Û--------------------------------ì
```

Im Register DL wird die Laufwerksnummer (0 = A:, 1 = B: etc.) übergeben. Da die Funktion den Inhalt auf Gültigkeit überprüft, sind nur bestimmte Werte zulässig. Soll die Festplatte angesprochen werden, sind die Eingaben 80H und 81H erlaubt. Bei Disketten haben die Werte 00H und 01H Gültigkeit. DH enthält die Kopfnummer (Diskette 0-1, Platte 0-15), auf die geschrieben werden soll. Dieser Wert wird nicht überprüft. In das Register CH wird die Spurnummer geladen, während CL die Sektornummer innerhalb der Spur enthält. Bei Festplatten wird die 10-Bit-Zylindernummer im Register CH (untere 8 Bit) und im Register CL (obere 2 Bit) übergeben. Es gelten die gleichen Konventionen wie bei AH = 02H. Die Sektornummern reichen in Abhängigkeit vom Medium nur von 1 bis 17, belegen also kein ganzes Byte. Beide Werte werden nicht überprüft. Im Register AL ist die Zahl der zu schreibenden Sektoren anzugegeben. Es können immer nur logisch aufeinanderfolgende Sektoren geschrieben werden. Der Wert wird nicht überprüft. Weiterhin muß ein Puffer mit den zu schreibenden Daten angelegt werden, dessen Anfangsadresse als Zeiger im Registerpaar ES:BX übergeben wird. Beim Aufruf werden dann die Daten aus dem Puffer in die angegebenen Sektoren übertragen.

Das Carry-Flag zeigt an, ob beim Aufruf der Funktion ein Fehler auftrat. Bei gesetztem Carry-Flag findet sich im Register AH ein Fehlercode, dessen Bedeutung bereits bei der Funktion (AH = 1 Status lesen) besprochen wurde. Im Fehlerfall ist anschließend das Laufwerk mit dem Aufruf AH = 0 zurückzusetzen. Anschließend kann der Schreibaufruf wiederholt werden, um eventuelle Probleme durch die Motor-Hochlaufzeit auszuschließen. Erst nach dreimaligem Versuch sollte die Operation abgebrochen werden.

Bei Festplatten enthält das Registerpaar CX:DX nach dem Aufruf die Zahl der geschriebenen 512-Byte-Blocks.

Sektor Verify (AH = 04H)

Mit dieser Funktion läßt sich prüfen, ob die Daten korrekt auf die Diskette geschrieben wurden. Hierzu sind die Register ebenfalls, wie bei der Read-Funktion beschrieben, vor dem Aufruf zu initialisieren.

```
Ö--------------------------------Ì
º     CALL  :  INT13             º
º                                º
º AH: 04H (Sektor Verfiy)        º
º DL: Drive-Nummer               º
º DH: Kopfnummer                 º
º CH: Spur-(Zylinder)-Nummer     º
º CL: Sektornummer               º
º AL: Zahl der Sektoren          º
º ES:BX  Adresse Datenpuffer     º
û--------------------------------À
º   RETURN                       º
º CY: 0 kein Fehler              º
º AH: Statuscode                 º
º AL: verifizierte Sektoren      º
º CY: 1                          º
º AH: Fehlercode                 º
Û--------------------------------ì
```

Das Ergebnis der Verify-Operation wird über das Register AH und das Carry-Flag (siehe Funktion Read) zurückgegeben. Im Fehlerfall ist anschließend das Disksystem mit dem Aufruf AH = 0 zurückzusetzen. Anschließend kann der Leseaufruf wiederholt werden, um eventuelle Probleme durch die Motor Hochlaufzeit auszuschließen.

Spur formatieren (AH = 05H)

Um eine Diskette/Platte zu formatieren, läßt sich diese BIOS-Unterfunktion benutzen. Sie ist aber nicht mit dem MS-DOS-Programm FORMAT zu verwechseln, welches neben der physikalischen Formatierung der Diskette auch noch logische Informationen, wie Boot-Records oder Directory-Informationen, auf dem Medium ablegt. Pro Aufruf läßt sich eine Spur physikalisch formatieren, d.h. neben der Einteilung in Sektoren werden Informationen in den Kopf der einzelnen Sektoren geschrieben.

```
Ö--------------------------------Ì
º     CALL : INT 13              º
º                                º
º AH: 05H (Sektor Format)        º
º ES:BX  Zeiger auf Tabelle      º
û--------------------------------À
º                                º
º     RETURN                     º
º CY: 0                          º
º     kein Fehler                º
º CY: 1                          º
º AH: Fehlercode                 º
Û--------------------------------ì
```

Für die Format-Operation benötigt die BIOS-Routine einen Zeiger auf die Tabelle mit Steueroperationen. Dieser Zeiger wird im Registerpaar ES:BX, in der Segment:Offset-Notation übergeben (ES = Segment, BX = Offset).

Pro Sektor muß ein Eintrag in der Tabelle vorhanden sein. Jeder Eintrag besteht aus vier Byte mit folgender Kodierung:

```
Byte º Bemerkung
-----é-----------------------------------
  0  º Spurnummer
  1  º Kopfnummer
  2  º Sektornummer
  3  º Bytes pro Sektor (00=128, 01=256,
     º                   02=512, 03=1024)
```

Tabelle 2.8: Belegung des Formatsteuerblock;s

Zur Formatierung einer 360-Kbyte-Diskette muß die Tabelle 2.8 also 9 Einträge zu je 4 Byte enthalten, da die Diskette pro Spur 9 Sektoren besitzt. Interessant ist in diesem Zusammenhang die Möglichkeit, innerhalb einer Spur die Zahl der Bytes pro Sektor zu variieren. MS-DOS speichert normalerweise 512 Byte pro Sektor. Die Informationen aus der Tabelle finden sich im Kopf eines jeden Sektors und dienen später zur Identifizierung des Sektors bei Schreib-Lese-Operationen. Allerdings gibt es Unterschiede in den einzelnen BIOS-Versionen. Sobald 1,2-Mbyte-Laufwerke unterstützt werden, ist vor dem Formatieren die Funktion AH = 01H aufzurufen, um den Diskettentyp zu setzen. Sollen 320/360-Kbyte-Disketten in einem 1,2-Mbyte-Laufwerk formatiert werden, sind vor dem Aufruf die folgenden Parameter in der Disk-Parameter-Tabelle zu modifizieren.

```
GAP Length           =  050H
Last Sektor/Track    =  8 oder 9 Sektoren
```

Die Adresse der Disk-Parameter-Tabelle findet sich in der DOS-Interruptvektor-Tabelle ab Adresse 0000:0078H. Der Aufbau ist im Abschnitt 1.8 (Der DOS-Datenbereich) beschrieben. Anschließend sind die alten Werte wieder zu restaurieren.

Bei einer Festplatte muß der Vektor in ES:BX auf eine 512-Byte-Tabelle zeigen, die für jeden Sektor zwei weitere Bytes enthält. Im ersten Byte wird markiert, ob der erste Sektor benutzbar oder gesperrt ist:

```
00H  : der Sektor ist benutzbar
80H  : der Sektor ist gesperrt
```

Im zweiten Byte steht die Sektornummer, wobei bei Festplatten ein *Interleave-Faktor***Fehler! Verweisquelle konnte nicht gefunden werden.**en. Bei aufeinanderfolgenden Sektoren müßte der Controller warten, bis der folgende Sektor nach einer Umdrehung wieder unter dem Schreib-Lese-Kopf vorbeikommt. Der Versatz um einige physikalische Sektoren stellt sicher, daß der Controller wieder lesebereit ist, wenn der folgende logische Sektor unter dem Kopf vorbeikommt. Damit lassen sich alle logischen Sektoren einer Spur mit einer Umdrehung der Platte lesen, was die Zugriffszeiten erheblich minimiert.

```
   1    5    2    6    3    7    4  <- logische Sektoren
Ö----Ú----Ú----Ú----Ú----Ú----Ú----Ì
Û----Ù----Ù----Ù----Ù----Ù----Ù----ì
  1    2    3    4    5    6    7  <- physikalische Sektoren
```

Bild 2.7: Interleave-Faktor; (2) bei Festplatten

Die Daten für den Aufbau der Festplatte bezieht die BIOS-Funktion aus der Disk-Parameter-Tabelle.

Im BIOS-ROM des IBM-PC/AT wurden weitere Funktionen unter dem INT 13 implementiert, die nachfolgend ebenfalls kurz besprochen werden. Die Funktionscodes AH = 06H und 07H sind unbelegt.

Harddisk-Format Track (AH = 06H)

Diese Funktion wird nur von einigen Harddisk-Controllern (z.B. XT) unterstützt. Sie formatiert eine Spur und setzt die *bad sector flags*. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 06H (Format Track)        º
º AL  Interleave-Faktor (nur XT)º
º CH  Zylindernummer            º
º CL  Sektornummer              º
º DH  Kopf                      º
º DL  Drive                     º
º ES:BX  512-Byte-Format-Buffer º
û-------------------------------À
º      RETURN                   º
º AH: Statuscode                º
Û-------------------------------ì
```

Die Bits 8,9 der Zylindernummer werden in den Highbits des Registers CL übergeben. Im 512-Byte-Format-Buffer enthalten die ersten 2*(Sector/Track) die Kennungen f,n für jeden Sektor:

```
f  0FH Sektor benutzbar
   80H Bad Sektor
n  Sektornummer
```

Die Funktion gibt in AH den Statuscode der Operation zurück.

Harddisk-Format ab n. Spur (AH=07H)

Dieser Aufruf ist ebenfalls nur bei einigen Harddisks (z.B. PC 2, XT) vorhanden. Er formatiert eine Platte ab der gewünschten Spur.

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 07H (Format Disk)         º
º AL  Interleave-Faktor (nur XT)º
º CH  Zylindernummer            º
º CL  Sektornummer              º
º DH  Kopf                      º
º DL  Drive                     º
º ES:BX  512-Byte-Format-Buffer º
û-------------------------------À
º      RETURN                   º
º AH: Statuscode                º
Û-------------------------------ì
```

Die Bits 8, 9 der Zylindernummer werden in den Highbits des Registers CL übergeben. Im 512-Byte-Format-Buffer enthalten die ersten 2*(Sector/Track) die Kennungen f,n für jeden Sektor:

```
f  0FH Sektor benutzbar
   80H Bad-Sektor
n  Sektornummer
```

im Register AL ist der Interleave-Faktor (1-10H beim XT) zu übergeben. Die Zylindernummer darf zwischen 0 und 3FFFH liegen, während die Kopfnummer bei 0-7 liegt. Die Funktion gibt in AH den Statuscode der Operation zurück.

Get Current Drive Parameter (AH = 08H)

```
Ö--------------------------------Ì
º     CALL : INT 13              º
º                                º
º AH: 08H (Get Drive Parameter)  º
º DL: Drive Nr (0-2)             º
û--------------------------------À
º     RETURN                     º
º CY: 1 --> Fehler               º
º AH: Fehlercode                 º
º CY: 0 --> kein Fehler          º
º CL: Laufwerksnummer (0 - 2)    º
º DH: Kopfzahl                   º
º DL: Zahl der Drives            º
º CH: Zylindernummer             º
º CL: Sektornummer               º
Û--------------------------------ì
```

Dieser Funktionsaufruf erlaubt es, die Parameter des aktuell eingestellten Laufwerkes abzufragen. Die Funktion ist nur in Verbindung mit einer Festplatte aufrufbar, wobei nebenstehende Parameter zu übergeben sind. Ist in DL beim Aufruf Bit 7 gesetzt, bezieht sich die Angabe auf eine Harddisk.

Die Zylindernummer teilt sich wieder auf die Register CH und CL auf, wobei die beiden oberen Bits in CL die Bits 8 und 9 der Zylindernummer bilden. (CH = Bit 0-7, CL = Bit 8-9). Die Werte sind nur gültig, falls kein Carry-Flag gesetzt ist.

Init Drive Characteristics (AH = 09H)

Das BIOS legt auch für die Festplatten Datenblöcke an, die die Eigenschaften eines Laufwerkes charakterisieren. Die Adressen dieser Datenblöcke finden sich in den Interruptvektoren:

```
INT 41 : Adresse Datenblock Drive 0
INT 46 : Adresse Datenblock Drive 1
```

Der Aufruf beeinflußt immer die Daten für ein Laufwerkspaar, da die Festplattencontroller meist zwei Einheiten unterstützen.

```
Ö--------------------------------Ì
º     CALL : INT 13              º
º                                º
º AH: 09H (Init Drive)           º
û--------------------------------À
º     RETURN                     º
º CY: 1 --> Fehler               º
º AH: Fehlercode                 º
º CY: 0 --> kein Fehler          º
Û--------------------------------ì
```

Die Urdaten sind im CMOS-RAM der Echtzeituhr gespeichert. Die Routine liest die Daten aus diesem Bereich und kopiert sie in in die Tabelle. Der Adreßvektor wird dann unter den Interruptvektoren eingetragen. Die Struktur ist in Tabelle 2.9 dargestellt.

Nach dem Aufruf sind die Tabellen mit den Werten aus dem CMOS-RAM belegt. Um andere Werte einzustellen, ist eine zweite Tabelle aufzubauen. Dann ist der Vektor auf diese Tabelle umzusetzen.

Offset	Bytes	Feld
00	2	maximale Zylinderzahl
02	1	maximale Kopfzahl
03	2	---
05	2	Startzylinder zum Schreiben
07	1	maximale Datenlänge für ECC
08	1	Kontrollbyte: Bit 7 - 6 disable retry Bit 5 = mehr als 8 Köpfe
09	3	---
0C	2	Landezone für Köpfe
0E	1	Sektoren pro Spur

Tabelle 2.9: Aufbau der Disk-Parameter-Tabelle; bei Platten

Read Long (AH = 0AH)

Mit diesem Aufruf lassen sich mehr als die Sektoren einer Spur auf einmal lesen. Ein Read-Long-Aufruf kann zwischen 1 und 79 Sektoren lesen. Es gelten folgende Aufrufkonventionen:

```
CALL : INT 13

AH: 0AH (Read Long)
DL: Laufwerksnummer (80H,81H)
DH: Kopfnummer (0 - 15)
CH: Zylindernummer (0 - 1023)
CL: Sektornummer (0 - 17)
AL: Sektoren zu lesen
ES:BX Lesepuffer für Daten

RETURN
CY: 1 --> Fehler
AH: Fehlercode
CY: 0 --> kein Fehler
```

Für die Registerbelegungen gelten die üblichen Konventionen. Die Daten werden in einem Lesepuffer abgelegt, dessen Anfangsadresse im Registerpaar ES:BX übergeben wird. Die Zahl der zu lesenden Sektoren findet sich in AL. Tritt ein Fehler auf, wird das Carry-Flag gesetzt. Der Aufruf der Long-Read-Funktion umgeht den Error-Correction-Code (ECC), der sich normalerweise auf 512 Byte bezieht.

Write Long (AH = 0BH)

Mit diesem Aufruf lassen sich mehr als die Sektoren einer Spur auf einmal schreiben. Ein Write-Long-Aufruf kann zwischen 1 und 79 Sektoren bearbeiten. Es gelten folgende Aufrufkonventionen:

```
Ö--------------------------------Ì
º     CALL : INT 13              º
º                                º
º AH: 0BH (Write Long)           º
º DL: Laufwerksnummer (80H,81H)  º
º DH: Kopfnummer (0 - 15)        º
º CH: Zylindernummer (0 - 1023)  º
º CL: Sektornummer (0 - 17)      º
º AL: Sektoren zu schreiben      º
º ES:BX Datenpuffer              º
û--------------------------------À
º     RETURN                     º
º CY: 1 --> Fehler               º
º AH: Fehlercode                 º
º CY: 0 --> kein Fehler          º
Û--------------------------------ì
```

Für die Registerbelegungen gelten die üblichen Konventionen. Die Daten werden in einem Puffer übergeben, dessen Anfangsadresse sich im Registerpaar ES:BX findet. Die Zahl der zu schreibenden Sektoren findet sich in AL. Tritt ein Fehler auf, wird das Carry-Flag gesetzt. Der Aufruf der Long-Write-Funktion umgeht den Error-Correction-Code (ECC), der sich normalerweise auf 512 Byte bezieht.

Seek (AH = 0CH)

Mit diesem Aufruf läßt sich eine SEEK-Funktion ausführen. Es gelten folgende Aufrufkonventionen.

```
Ö--------------------------------Ì
º     CALL : INT 13              º
º                                º
º AH: 0CH (SEEK)                 º
º DL: Laufwerksnummer (80H,81H)  º
º DH: Kopfnummer (0 - 15)        º
º CH: Zylindernummer (0 - 1023)  º
º CL: Sektornummer (0 - 17)      º
û--------------------------------À
º     RETURN                     º
º CY: 1 --> Fehler               º
º AH: Fehlercode                 º
º CY: 0 --> kein Fehler          º
Û--------------------------------ì
```

Für die Registerbelegungen gelten die üblichen Konventionen. Tritt ein Fehler auf, wird das Carry-Flag gesetzt.

Im BIOS der Festplatte sind weitere Funktionsaufrufe implementiert:

```
AH = 0CH   Alternate Disk Reset
AH = 10H   Test Drive Ready
AH = 11H   Recalibrate
AH = 14H   Controller internal Diagnosis
```

die nicht näher diskutiert werden sollen. Es gelten beim Aufruf die üblichen Konventionen. Für weitere Informationen sei auf die Literatur der jeweiligen Hersteller verwiesen.

Alternate Disk Reset (AH = 0DH)

Mit diesem Aufruf wird eine Festplatte zurückgesetzt. Es gelten folgende Aufrufparameter:

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 0DH (Reset  Disk)         º
º DL  Drive                     º
û-------------------------------À
º      RETURN                   º
º CF -> 0 kein Fehler           º
º CF -> 1 Fehler                º
º AH: Statuscode                º
Û-------------------------------ì
```

Die Funktion kann beim PC und bei PS/2-Modellen mit ESDI-Platten nicht benutzt werden.

Read Sector Buffer (AH = 0EH)

Dies ist ebenfalls ein neuer Aufruf, der beim XT und den PS/2-Modellen eingeführt wurde.

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 0EH (Read Sektor Buffer)  º
º AL  Zahl der Sektoren         º
º CH  Zylindernummer            º
º CL  Sektornummer              º
º DH  Kopf                      º
º DL  Drive                     º
º ES:BX  Zeiger auf Puffer      º
û-------------------------------À
º      RETURN                   º
º CF -> 0 kein Fehler           º
º CF -> 1 Fehler                º
º AH: Statuscode                º
Û-------------------------------ì
```

Der Aufruf transferiert die Daten aus dem Puffer des Controllers in den angegebenen Puffer im Hauptspeicher. Es werden allerdings keine Daten von der Platte gelesen! Der Aufruf wird bei PS/2-Systemen im wesentlichen für Diagnosezwecke benutzt.

Write Sector Buffer (AH = 0FH)

Der Aufruf ist beim XT, PS/2 und einigen Portablen vorhanden.

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 0FH (Write Sector Buffer) º
º AL  Zahl der Sektoren         º
º CH  Zylindernummer            º
º CL  Sektornummer              º
º DH  Kopf                      º
º DL  Drive                     º
º ES:BX  Zeiger auf Puffer      º
û-------------------------------À
º      RETURN                   º
º CF -> 0 kein Fehler           º
º AL: Zahl der transf. Sektoren º
º CF -> 1 Fehler                º
º AH: Statuscode                º
Û-------------------------------ì
```

Der Aufruf transferiert die Daten aus dem Puffer des Speichers in den Puffer des Controllers. Er sollte vor einem Formatieraufruf benutzt werden, um den Controller zu initialisieren. Der Aufruf wird bei PS/2-Systemen im wesentlichen für Diagnosezwecke eingesetzt.

Test For Drive Ready (AH = 10H)

Der Aufruf prüft, ob das angegebene Laufwerk bereit ist.

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 10H (Test Drive Ready)    º
º DL  Drive                     º
û-------------------------------À
º      RETURN                   º
º CF -> 0 kein Fehler           º
º CF -> 1 Fehler                º
º AH: Statuscode                º
Û-------------------------------ì
```

Recalibrate Drive (AH = 11H)

Der Aufruf ruft ein Recalibrate des Laufwerkes auf.

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 11H (Recalibrate Drive)   º
º DL  Drive                     º
û-------------------------------À
º      RETURN                   º
º CF -> 0 kein Fehler           º
º CF -> 1 Fehler                º
º AH: Statuscode                º
Û-------------------------------ì
```

Der Controller sucht auf dem spezifizierten Laufwerk den Zylinder 0.

Controller RAM Diagnose (AH = 12H)

Der Aufruf dient zur Überprüfung des Controllers bei XT, PS/2 und einigen Portablen.

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 12H (RAM Diagnose Kontr.) º
û-------------------------------À
º      RETURN                   º
º CF -> 0 kein Fehler           º
º CF -> 1 Fehler                º
º AH: Statuscode                º
Û-------------------------------ì
```

Drive Diagnose (AH = 13H)

Der Aufruf dient zur Überprüfung des Laufwerkes bei XT, PS/2 und einigen Portablen.

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 13H (Drive Diagnose)      º
û-------------------------------À
º      RETURN                   º
º CF -> 0 kein Fehler           º
º CF -> 1 Fehler                º
º AH: Statuscode                º
Û-------------------------------ì
```

Controller internal Diagnose (AH = 14H)

Der Aufruf dient zur Überprüfung des Controllers bei XT, PS/2 und einigen Portablen.

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 14H (Controller Diagnose) º
û-------------------------------À
º      RETURN                   º
º CF -> 0 kein Fehler           º
º CF -> 1 Fehler                º
º AH: Statuscode                º
Û-------------------------------ì
```

Der Aufruf wird für Western Digital 1003-WA2 Hard/Floppy-Disk-Controller in AT und XT/286 benutzt. Er ist auch in PS/2-Systemen vorhanden.

Read DASD Type (AH = 15)

Mit diesem Aufruf läßt sich der Laufwerkstyp bei Disketten feststellen. Daher ist dieser Aufruf nur für Diskettenlaufwerke erlaubt. Es wird auch geprüft, ob die Selektleitung für zwei Laufwerke umschaltbar ist.

```
Ö-------------------------------Ì
º     CALL : INT 13             º
º                               º
º AH: 15H (Read DASD Type)      º
º DL: Laufwerksnummer (< 80H)   º
º DH: Kopfnummer (1 - 2)        º
º CH: Spurnummer                º
º CL: Sektornummer (0 - 17)     º
û-------------------------------À
º     RETURN                    º
º CY: 1 --> Fehler              º
º AH: Fehlercode                º
º DL: Laufwerk                  º
º CY: 0 --> kein Fehler         º
º AH: Status                    º
Û-------------------------------ì
```

Falls kein Carry-Flag gesetzt ist, enthält das Register AH den Status:

```
0 : nicht vorhanden
1 : Diskette besitzt keine Change Line
2 : Diskette besitzt eine Change Line
3 : Festplatte
```

Bei Diskettenlaufwerken existiert eine Leitung, mittels derer der Controller zwischen den Laufwerken A und B umschaltet. Der Aufruf prüft, ob diese Umschaltung unterstützt wird.

Die Aufrufe:

```
AH = 16H  Disk Change Line Status
AH = 17H  Set DASD Typ for Format
```

beeinflussen ebenfalls diese Leitung. Es gelten die üblichen Aufrufkonventionen. Nähere Informationen finden sich in den jeweiligen Unterlagen der Hersteller.

Damit sind die Aufrufe innerhalb des Festplatten/Disketten-BIOS-Teils besprochen. Falls bei gesetztem Carry-Flag der Fehlercode 11H im Register AH auftritt, ist dies ein Hinweis auf einen Lesefehler, der durch den Error-Correction-Code kompensiert werden konnte. Das Anwenderprogramm besitzt die Möglichkeit, die Häufigkeit solcher Fehler zu erfassen und auszuwerten, um Aussagen über die Güte des Mediums zu treffen.

Change of Disk Status (Diskette) (AH = 16H)

Der Aufruf wird nicht beim PC und XT unterstützt. Er prüft, ob die Diskette ausgewechselt wurde. Dabei gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL  :  INT 13           º
º                                º
º AH: 16H (Disk Change Status)   º
û--------------------------------À
º      RETURN                    º
º CF -> 0 kein Fehler            º
º CF -> 1 Fehler                 º
º AH: Statuscode                 º
º DL: Drive                      º
Û--------------------------------ì
```

Der Statuscode besitzt folgende Kodierung:

```
00H   no disk change
01H   disk changed
```

Im Register DL wird die Laufwerksnummer übergeben, dessen Diskette gewechselt wurde.

Set Disk Type for Format (Diskette) (AH = 17H)

Der Aufruf wird nicht bei PC- und XT-Systemen unterstützt. Er setzt den Diskettentyp vor einer Formatierung.

```
Ö--------------------------------Ì
º      CALL  :  INT 13           º
º                                º
º AH: 17H (Set Disk Typ)         º
º AL: Disk Typ                   º
º DL: Drive                      º
û--------------------------------À
º      RETURN                    º
º CF -> 0 kein Fehler            º
º CF -> 1 Fehler                 º
º AH: Statuscode                 º
Û--------------------------------ì
```

Im Register AL ist der Code für den Diskettentyp gemäß folgender Aufstellung zu übergeben:

```
00H  no disk
01H  360-Kbyte-Diskette in 360-Kbyte-Drive
02H  360-Kbyte-Diskette in 1.2-Mbyte-Drive
03H  1.2-Mbyte-Diskette in 1.2-Mbyte-Drive
04H  720-Kbyte-Diskette in 720-Kbyte-Drive
```

In DL ist die Nummer des Laufwerkes zu übergeben.

Set Media Type for Format (Diskette) (AH = 18H)

Der Aufruf wird bei AT-, XT/286- und PS/2-Modellen unterstützt.

```
Ö--------------------------------Ì
º      CALL  :  INT 13           º
º                                º
º AH: 18H (Set Media Typ)        º
º CH  Zahl der Spuren (Lowbyte)  º
º CL  Bit 6/7 Tracks  0-5 Sekt.  º
º DL  Drive                      º
û--------------------------------À
º      RETURN                    º
º AH: Statuscode                 º
º ES:DI  Zeiger auf Parameter    º
Û--------------------------------ì
```

In CH werden die unteren 8 Bit der Spurzahl übergeben. Die Bits 6 und 7 in CL enthalten die höheren Bits der Spurzahl, während in den Bits 0 bis 5 die Zahl der Sektoren pro Spur steht. In DL wird die Laufwerksnummer übergeben. Nach dem Aufruf steht in AH der Statuscode mit folgender Bedeutung:

```
00H  if requested combination supported
01H  if function not available
0CH  if not suppported or drive type unknown
80H  if there is no media in the drive
```

In Registerpaar ES:DI findet sich ein Zeiger auf eine 11 Byte lange Parametertabelle.

Park Hard Disk Heads (AH = 19H)

Mit diesem Aufruf lassen sich die Köpfe in eine Parkposition bringen (XT/286, PS/2).

```
Ö--------------------------------Ì
º      CALL  :  INT 13           º
º                                º
º AH: 19H (Park Disk Heads)      º
º DL  Drive                      º
û--------------------------------À
º      RETURN                    º
º CF -> 0 kein Fehler            º
º CF -> 1 Fehler                 º
º AH: Statuscode                 º
Û--------------------------------ì
```

ESDI Harddisk-Format (AH = 1AH)

Dieser Aufruf wird nur bei PS/2-Systemen unterstützt.

```
Ö-------------------------------Ì
º      CALL  :  INT 13          º
º                               º
º AH: 1AH (Format Disk)         º
º AL  Defect Table Count        º
º CL  Format Modifier           º
º DL  Drive                     º
º ES:BX  Zeiger auf Tabelle     º
û-------------------------------À
º      RETURN                   º
º CF -> 0 kein Fehler           º
º CF -> 1 Fehler                º
º AH: Statuscode                º
Û-------------------------------ì
```

In AL wird die Zahl der defekten Bereiche angegeben, während in ES:BX der Zeiger auf die Tabelle mit den defekten Bereichen steht. In CL ist ein *Format Code* gemäß folgender Kodierung zu übergeben:

```
Format modifier
------Ú--------------------------
bit 0:º ignore primary defect map
bit 1:º ignore secondary defect map
bit 2:º update secondary defect map
bit 3:º perform surface analysis
bit 4:º generate periodic interrupt
```

Falls bei diesem Aufruf der periodische Interrupt angewählt wurde, löst die Funktion einen INT 15H mit AH=0FH nach jeder Formatierung eines Zylinders aus.

Der INT 13H wird weiterhin von verschiedenen Cache-Programmen abgefangen (PC-Kwik, PC-Cache, IBMCACHE.SYS). Wegen der Vielzahl der Produkte wird hier auf eine Beschreibung der Aufrufschnittstelle verzichtet.

2.6 Serielle Schnittstelle (INT 14)

Die Abwicklung der Ein-/Ausgaben zu den zwei seriellen Schnittstellen COM1 und COM2 kann über diese BIOS-Funktion erfolgen. Wie bei anderen Funktionen erfolgt die Parameterübergabe per Register. Zur Auswahl der Schnittstelle dient das DX-Register:

```
DX= 0  Schnittstelle COM1:
DX= 1  Schnittstelle COM2:
```

Mit dem AH-Register lassen sich die Unterfunktionen selektieren.

Schnittstelle initialisieren (AH = 00H)

Bei der seriellen Datenübertragung sind bestimmte Parameter wie Übertragungsgeschwindigkeit (Baudrate), Zahl der Datenbits, Stoppbits etc. festzulegen. Es gelten folgende Aufrufparameter:

```
Ö------------------------------Ì
º      CALL :  INT 14          º
º                              º
º AH: 00H  (Init V24)          º
º AL: Initialisierungswerte    º
º DX: Schnittstellen-Nummer    º
û------------------------------À
º      RETURN                  º
º AX:    Statuswort            º
Û------------------------------ì
```

Der Wert AH = 0 aktiviert die Unterfunktion zur Initialisierung der seriellen Schnittstelle. Im Register DX wird die Schnittstelle selektiert (0 = COM1:, 1 = COM2:). Das Initialisierungsbyte zur Einstellung der Baudrate, Stoppbits etc. findet sich im Register AL. Es gilt folgende Kodierung:

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙ-ÙÚÙÙÚÙÚÙÙÚÙÙÚÙì   Zahl der Datenbits
 Û-Ú-ì ÛÚÌ º  ÛÚÌ    10 = 7 Datenbits
   º    º  º  Û--    11 = 8 Datenbits
   º    º  º
   º    º  Û-----    Zahl der Stoppbits
   º    º            0 = 1 Stoppbit
   º    º            1 = 2 Stoppbits
   º    º
   º    Û--------    Parity Check
   º                 x0  No Parity Check
   º                 01  Odd Parity Check
   º                 11  Even Parity Check
   º
   Û-------------    Baudrate
                     000 = 110    100 = 1200
                     001 = 150    101 = 2400
                     010 = 300    110 = 4800
                     011 = 600    111 = 9600
```

Bild 2.8: Modus-Byte zur Einstellung der seriellen Schnittstelle

Nach der Rückkehr findet sich im AX-Register das Statuswort, welches bei der Unterfunktion AH = 3 (Status lesen) besprochen wird.

Zeichen ausgeben (AH = 01H)

Mit dieser Unterfunktion kann ein Zeichen in das Senderegister des 8250-Bausteines übertragen werden. Das Zeichen wird im AL-Register übergeben.

```
Ö------------------------------Ì
º      CALL  :  INT 14         º
º                              º
º AH: 01H  (Send Data)         º
º AL: Zeichen                  º
º DX: Schnittstellen-Nummer    º
û------------------------------À
º      RETURN                  º
º AH:    Statuswort            º
Û------------------------------ì
```

Im Register DX wird wieder die entsprechende Schnittstelle selektiert. Der Status der Operation wird im AH-Register zurückgegeben. Falls Bit 7 gesetzt ist, konnte das Zeichen nicht übertragen werden. Die Kodierung der restlichen Bits wird bei der Funktion AH = 03H (lese Status) besprochen.

Zeichen lesen (AH = 02H)

Die Funktion liest ein Zeichen ein und gibt es im Register AL zurück. Der Status wird im AH-Register zurückgegeben, wobei die Kodierung innerhalb der AH = 03H-Funktion vorgestellt wird.

```
Ö------------------------------Ì
°      CALL  :  INT 14         °
°                              °
° AH: 02H  (Read Data)         °
° DX: Schnittstellen-Nummer    °
û------------------------------À
°      RETURN                  °
° AH: Statuswort               °
° AL: Zeichen                  °
Û------------------------------ì
```

Allerdings sind die Bits 5, 6 nicht definiert. Bit 7 = 1 zeigt an, daß das Data-Set-Ready-Signal nicht gesetzt war. Nur wenn der Registerinhalt AH = 00H ist (No Error), ist das Zeichen in AL gültig.

Status lesen (AH = 03H)

Zur Übertragungssteuerung verfügt der 8250-Baustein über mehrere externe Anschlüsse. Weiterhin können Fehler beim Empfang eines Zeichens auftreten. Der 8550-Baustein legt alle Informationen in Statusregistern ab, die sich mit der Funktion AH = 03H abfragen lassen.

```
Ö------------------------------Ì
°      CALL  :  INT 14         °
°                              °
° AH: 03H  (Read Status)       °
° DX: Schnittstellen-Nummer    °
û------------------------------À
°      RETURN                  °
° AX: Statuswort               °
Û------------------------------ì
```

Die Kodierung der Bits entspricht nachfolgender Darstellung. Der Status der externen Signalanschlüsse (Modem Control) wird im Register AL zurückgegeben.

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙÚÙÚÙÚÙÚÙÚÙÚÙÚÙì
 ° ° ° ° ° ° ° Û-  Delta Clear to Send
 ° ° ° ° ° ° Û---  Delta Data Set Ready
 ° ° ° ° ° Û-----  Trailing Edge Ring Indicator
 ° ° ° ° Û-------  Receive Line Signal detect
 ° ° ° Û---------  Clear to Send
 ° ° Û-----------  Data Set Ready
 ° Û-------------  Ring Indicator
 Û---------------  Received Line Signal detect
```

Bild 2.9: Modem-Control-Byte (serielle Schnittstelle)

Der Status des Kommunikationsports (8250-Baustein) wird im AH-Register zurückgegeben.

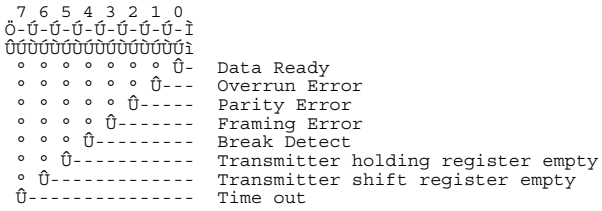


Bild 2.10: Kommunikationsport; Statusbyte (serielle Schnittstelle)

Die Schreib-Lese-Funktionen geben dieses Statusbyte ebenfalls nach dem Aufruf zurück.

Extended Initialize (AH = 04H)

Dieser Aufruf existiert beim PC Convertible / PS/2 und besitzt folgende Aufrufschnittstelle:

```
CALL:   INT 14
AH: 04H (Extended Initialize)
AL: Break Status
BH: Parity
BL: Stoppbits
CH: Zahl der Datenbits
CL: Baud-Rate
RETURN
AH: Kommunikationsport Status
AL: Modem-Controll-Byte-Status
```

Für den Break-Status gilt folgende Kodierung:

```
01H  Break
00H  No Break
```

In BH wird die Parity-Prüfung selektiert:

```
00H  No parity
01H  Odd parity
02H  Even parity
03H  Stick Parity odd
04H  Stick Parity even
```

In BL steht die Zahl der Stoppbits:

```
00H  1 Stoppbit
01H  2 Stoppbit
```

In CH wird die Zahl der Datenbits pro Zeichen gesetzt. Bei einer Übertragung von 5 Datenbit werden an Stelle von 2 Stoppbit nur $1^1/_2$ Stoppbit gesendet. Für die Einstellung der Datenbits gilt folgende Kodierung:

```
00H  5 Bit
01H  6 Bit
02H  7 Bit
03H  8 Bit
```

Die Baudrate wird in CL übergeben, wobei folgende Kodierung gilt:

```
00H  110
01H  150
02H  300
03H  600
04H  1200
05H  2400
06H  4800
07H  9600
08H  19200
```

Im Register AX stehen die Statusbytes gemäß der Kodierung in Bild 2.9 und 2.10.

Extended Communication Port Control (AH = 05H)

Dieser Aufruf existiert nur beim PS/2 und besitzt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º       CALL:   INT 14           º
º                                º
º AH: 05H (Extended Port Control)º
º AL: 00 Read Modem Control Reg. º
º     01 Write Modem Control Reg.º
º BL: Modem Control Register     º
û--------------------------------À
º           RETURN               º
º AH: Status                     º
Û--------------------------------ì
```

Mit dem Aufruf AL = 0 wird das Modem-Control-Register gelesen. Das Ergebnis wird in BL zurückgegeben. Mit AL= 1 wird der in BL übergebene Inhalt in das Modem-Control-Register geschrieben. Für BL gilt dabei folgende Kodierung:

```
Bits  º  Bedeutung
------é-------------------------
 0    º  data terminal ready
 1    º  request to send
 2    º  out1
 3    º  out2
 4    º  loop
 5,6,7º  reserved
```

Nach dem Aufruf steht im Register AH der Statuscode.

Einige Netzwerke fangen ebenfalls den INT 14 ab, um eine Kommunikation per serieller Schnittstelle zu erlauben (z.B. 3COM).

2.7 INT 15

Diese Funktion wurde beim IBM-PC für die Ein-/Ausgaberoutine des Kassettenrecorders benutzt. Bei Folgemodellen bleibt die Funktion frei oder wird für andere Zwecke belegt. Im BIOS des IBM-PC/AT sind z.B. weitere Funktionen im INT 15 implementiert.

Die einzelnen Funktionen werden über den Inhalt des Registers AH selektiert. Die Codes 00H bis 03H sind durch die Kassetten-I/O-Funktionen belegt. Sie geben im Register AH jeweils den Wert 86H zurück, falls der Ausgang für den Recorder fehlt. Gleichzeitig wird das Carry-Flag gesetzt.

Die Aufrufe AH = 04H bis 7FH sind bei älteren BIOS-Versionen nicht implementiert und geben ebenfalls den Wert 86H im Register AH zurück, wobei ebenfalls das Carry-Flag gesetzt wird. Erweiterungen ergeben sich nur bei neueren BIOS-Versionen (AT03, PS/2 etc.).

Auch Betriebssystemerweiterungen wie TopView und DesqView benutzen einige Funktionen des Interrupt 15. Da diese Softwareprodukte in Deutschland nicht allgemein verbreitet sind, wird auf die Schnittstellen nicht eingegangen. Bei einigen Systemen benutzt das Programm PRINT.COM die Funktion 20H des INT 15 zur Speicherung interner Flags.

Die Funktionsaufrufe 80H, 81H, 82H und 85H wurden im BIOS der älteren IBM PC/AT-Computer bereits reserviert. Sie sind aber mit einem einfachen IRET-Statement abgeschlossen. In den erweiterten BIOS-Versionen lassen sich hier besondere Funktionen einbringen, die nachfolgend beschrieben werden.

ABIOS Build System Parameter Table (AH = 04)

Eine der Ausnahmen im Bereich der Funktionscodes AH=04H bis 7FH bieten die Aufrufe der erweiterten ABIOS-Funktionen der PS/2-Systeme. Der INT 15 mit der Funktion AH = 04H besitzt folgende Übergabeschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 04H (Build Table)         º
º ES:DI: Ergebnispuffer         º
º DS: Segment RAM               º
û-------------------------------À
º      RETURN                   º
º CY: 0 o.k.                    º
º AH: 00                        º
º CY: 1 Fehler                  º
Û-------------------------------ì
```

Beim Aufruf ist in ES:DI die Adresse eines 32 Byte (20H) großen Ergebnispuffers zu übergeben. Dieser Puffer besitzt folgende Struktur:

```
Offset Bytes Bedeutung

00H    04   FAR Adr. ABIOS Common Startroutine
04H    04   FAR Adr. ABIOS Interrupt Routine
08H    04   FAR Adr. ABIOS Time-out Routine
0CH    02   Stackgröße in Byte
0EH    16   reserviert
1EH    02   Einträge in der Init. Tabelle
```

Nach dem Aufruf finden sich in diesem Puffer die Daten, sofern das Carry-Flag gelöscht ist und AH den Wert 0 enthält. In DS ist das Segment für die ABIOS-RAM-Erweiterungen zu übergeben. Der Wert 0 in DS bedeutet, daß keine Erweiterungen vorgenommen werden sollen.

ABIOS Build Initialisation Table (AH = 05)

Dieser Aufruf erzeugt die Initialisierungs-Tabelle für die ABIOS-Routinen. Es gilt folgende Übergabeschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 05H (Build  Init Table)   º
º ES:DI: Adresse Puffer         º
º DS: Segment RAM               º
û-------------------------------À
º      RETURN                   º
º CY: 0 o.k.                    º
º AH: 00                        º
º CY: 1 Fehler                  º
Û-------------------------------ì
```

Beim Aufruf ist in ES:DI die Adresse eines n (18H * Zahl der Einträge) Byte großen Puffers zu übergeben. Dieser Puffer besitzt für jeden Eintrag folgende Struktur:

```
Offset Byte Bedeutung
  00H   02  Device ID
  02H   02  Zahl der logische ID's
  04H   02  Device Blocklänge (0 bei gepatchten
            Funktionen oder Erweiterungen)
  06H   04  Init Routine Device Block und
            Funktions Transfer Tabelle
  0AH   02  Request Block Länge
  0CH   02  Länge Funktions Transfer Tabelle
            (0 = Funktion gepatcht)
  0EH   02  Länge Datenzeiger
  10H   01  Second Device ID (HW Level)
  11H   01  Revision Number
  12H   06  reserviert
```

Nach einem erfolgreichen Aufruf (CY = 0 und AH = 0) enthält der Puffer die Ergebnisdaten.

ESDI-Format-Unit-Interrupt (AH = 0FH)

Eine weitere Ausnahme im Bereich der Funktionscodes AH=04H bis 7FH bildet der Aufruf der Funktion AH = 0FH der PS/2-ESDI-Schnittstelle. Er wird bei den PS/2-Systemen beim Low-Level-Format über den INT 13 periodisch aufgerufen, sobald ein Zylinder formatiert ist. Hierbei gilt folgende Übergabeschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 0FH (Format Interrupt)    º
º AL: Phasencode                º
û-------------------------------À
º      RETURN                   º
º CY: 0 Formatierung fortsetzen º
º CY: 1 beende Formatierung     º
Û-------------------------------ì
```

Der Interrupt 15 wird von der BIOS-INT-13-Routine ausgelöst, wobei im Register AL ein Code für die aktuelle Phase des Formatiervorganges übergeben wird:

```
AL =  00H  reserviert
      01H  Oberflächenanalyse
      02H  Formatierung
```

Das Anwenderprogramm kann den Formatiervorgang durch Setzen des Carry-Flags unterbrechen.

Tastatur-Interrupt (AH = 4FH)

Eine weitere Ausnahme bietet der Tastatur-Interrupt (AH = 4FH). Diese Funktion läßt sich durch den Anwender jedoch nicht aufrufen. Vielmehr löst die INT 9-Routine des BIOS bei neueren Versionen einen INT 15 mit der Funktion 4FH aus. Hierbei gilt folgende Übergabeschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 4FH (Key Interrupt)       º
º AL: Scan Code                 º
º CY: 1                         º
û-------------------------------À
º      RETURN                   º
º CY: 0 ignore Scan Code        º
Û-------------------------------ì
```

Während der Abarbeitung des INT 9-Aufrufes ermittelt die zugehörige BIOS-Routine den Scan-Code der betreffenden Taste. Vor der Weitergabe an den Tastaturpuffer aktiviert die Routine dann den INT 15 mit der Funktion 4FH. Hier kann das Betriebssystem nun eine eigene Routine installieren, die den im Register AL übergebenen Scan Code auswertet und gegebenenfalls modifiziert. Löscht die Routine das Carry-Flag, ignoriert der INT 9-Handler anschließend den Scan Code im Register AL. So lassen sich zum Beispiel einzelne Tasten softwaremäßig sperren. Bei einem gesetzten Carry-Flag speichert die INT 9-Routine anschließend den Scan Code im Tastaturpuffer.

Open Device (AH = 80H)

Mit dieser Funktion läßt sich eine Geräteeinheit öffnen. Der Funktionscode ist bei allen BIOS-Versionen bereits für diese Aufgabe reserviert. Es gelten folgende Aufrufparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 80H (Open Device)         º
º BX: Device ID                 º
º CX: Prozess ID                º
û-------------------------------À
º      RETURN                   º
º CY: 0  ok                     º
º AH: 0                         º
º CY: 1  Fehler                 º
º AH: Status                    º
º  80H invalid function         º
º  86H unsupported function     º
Û-------------------------------ì
```

Im Register BX ist die Identifikationsnummer des zu öffnenden Gerätes zu hinterlegen, während in CX der Kenncode des zugehörigen Prozesses steht. Das Carry-Flag ist vor dem Aufruf zu löschen.

Close Device (AH = 81H)

Mit dieser Funktion läßt sich eine per INT 15 geöffnete Geräteeinheit wieder schließen. Der Funktionscode ist bei allen BIOS-Versionen bereits für diese Aufgabe reserviert. Es gelten folgende Aufrufparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 81H (Close Device)        º
º BX: Device ID                 º
º CX: Prozess ID                º
û-------------------------------À
º      RETURN                   º
º CY: 0  ok                     º
º AH: 0                         º
º CY: 1  Fehler                 º
º AH: Status                    º
º  80H invalid function         º
º  86H unsupported function     º
Û-------------------------------ì
```

Im Register BX ist die Identifikationsnummer des zu öffnenden Gerätes zu hinterlegen, während in CX der Kenncode des zugehörigen Prozesses steht. Das Carry-Flag ist vor dem Aufruf zu löschen.

Program Termination (AH = 82H)

Mit diesem Aufruf läßt sich ein Programm mit der angegebenen Devicenummer abbrechen. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 82H (Program Termination) º
º BX: Device ID                 º
û-------------------------------À
º      RETURN                   º
º CY: 0  ok                     º
º AH: 0                         º
º CY: 1  Fehler                 º
º AH: Status                    º
º  80H invalid function         º
º  86H unsupported function     º
Û-------------------------------ì
```

Die Aufrufe 80H bis 82H geben keine Parameter zurück. Lediglich der Status der Operation findet sich im Carry und AH. Die Funktionen werden vom BIOS reserviert. Einige Hersteller benutzen die Codes bei erweiterten BIOS-Versionen. Diese Funktionen lassen sich von Multitasking Erweiterungen abfangen.

Event Wait (AH = 83H)

Mit dieser Funktion können Zeitaufträge an das BIOS abgesetzt werden. Das BIOS reagiert dann auf den Ablauf der Uhr mit dem Setzen des Event-Flags. Es gelten folgende Aufrufparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 83H (Event Wait)          º
º AL: Steuercode (0 oder 1)     º
º CX:DX Wartezeit in Mikrosek.  º
º ES:BX Zeiger auf Flagadresse  º
û-------------------------------À
º      RETURN                   º
º                               º
º CY: Fehler                    º
Û-------------------------------ì
```

Mit dem Wert AL = 0 wird der Zeitauftrag an das BIOS abgesetzt. Das Registerpaar CX:DX enthält dabei die Wartezeit in Mikrosekunden. Nach Ablauf der spezifizierten

Zeitdauer setzt die Funktion das höchste Bit des Event-Flags. Dieses Event-Flag ist als Bytefeld vom Anwenderprozeß einzurichten, wobei die Adresse im Registerpaar ES:BX übergeben wird. Das Bit muß vorher gelöscht worden sein, so daß der Anwenderprozeß durch Abfrage des Bits auf das Ereignis warten kann.

Wird die Funktion mit AL = 1 aufgerufen, löscht die Routine den letzten Zeitauftrag aus der Tabelle (Cancel). Dann wird das Bit nicht mehr gesetzt.

Im Fehlerfall ist das Carry-Flag nach dem Aufruf gesetzt. Dies ist zum Beispiel der Fall, wenn im Register AL ein Wert ungleich 0 oder 1 übergeben wird. Eine andere Fehlermöglichkeit besteht darin, bei einem noch laufenden Zeitauftrag die Funktion erneut mit dem Code AL = 0 aufzurufen. Dann ist nach dem Aufruf das Carry-Flag gesetzt.

Joystick Support (AH = 84H)

Mit diesem Aufruf läßt sich ein eventuell angeschlossener Joystick abfragen. Für den Aufruf gelten folgende Parameter:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 84H (Joystick Support)    º
º DX: Steuercode (0 oder 1)     º
û-------------------------------À
º      RETURN                   º
º CY = 1 -> kein Joystick       º
º CY = 0 -> Joystick vorhanden  º
º bei DX = 1 gilt:              º
º AX: Wert A(x) Koordinate      º
º BX: Wert A(y) Koordinate      º
º CX: Wert B(x) Koordinate      º
º DX: Wert B(y) Koordinate      º
º bei DX = 0                    º
º AL: Status                    º
Û-------------------------------ì
```

Falls kein Joystick vorhanden ist, wird das Carry-Flag gesetzt. Mit DX = 0 wird der aktuelle Status des Joysticks abgefragt. Das Ergebnis findet sich im AL-Register in den Bits 4-7.

Mit DX = 1 lassen sich die Koordinaten der zwei Joystickhebel abfragen. Diese finden sich anschließend in den Registern AX, BX, CX und DX.

System Request Key (AH = 85H)

```
Mit dieser Funktion des INT 15 läßt sich der Zustand der System-Request-TasteFehler!
Verweisquelle konnte nicht gefunden werden.Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 85H (Get System Request)  º
º AL: Key Status                º
û-------------------------------À
º      RETURN                   º
º CF: 0 ok                      º
º CF: 1 Fehler                  º
º AH: Fehlerstatus              º
Û-------------------------------ì
```

Die Funktion wird durch den Tastaturtreiber aufgerufen. Im Register AL wird der Status der System-Request-Taste zurückgegeben. Dabei gilt folgende Kodierung:

```
Taste gedrückt    :   AL = 0
Taste losgelassen :   AL = 1
```

Dabei ist das Carry-Flag gelöscht. Normlerweise wird dieser Aufruf sofort durch den Standard Handler beendet. Programme können sich aber in den Funktionsaufruf einklinken. Auch dieser Aufruf wird nur in erweiterten BIOS-Versionen unterstützt.

Wait (AH = 86H)

Ähnlich der Funktion 83H (Wait Event) kann mit diesem Aufruf eine Zeitfunktion des BIOS genutzt werden. Es gelten folgende Übergabeparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 86H (Wait)                º
º CX:DX Delay-Zeit in Mikrosek. º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Im Registerpaar CX:DX wird die Verzögerungszeit in Mikrosekunden angegeben. Während die Funktion 83H (Wait Event) aber sofort die Kontrolle an den rufenden Prozeß zurückgibt und nach Ablauf der Wartezeit das Event-Flag setzt, suspendiert dieser Aufruf den Anwender-Task für die angegebene Zeitspanne. Erst dann kann der Anwender-Task wieder aktiv werden. Damit läßt sich ein Programm für eine definierte Zeit anhalten.

Move Block (AH = 87H)

Ab dem 80286-Prozessor besteht die Möglichkeit, einen Speicherbereich bis 16 Mbyte zu adressieren. MS-DOS erlaubt jedoch nur den Real-Modus des 8088/8086-Prozessors, in dem 1 Mbyte adressiert werden kann. Es gibt aber Systeme, die den Bereich oberhalb 1 Mbyte belegen. Die Funktion 87H erlaubt es, aus MS-DOS diesen erweiterten (Extended Memory) Bereich anzusprechen. Es gelten folgende Aufrufkonventionen.

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 87H (Move Block)          º
º CX: Zahl der Worte im Puffer  º
º ES:SI Adresse Descriptor Tab. º
û-------------------------------À
º      RETURN                   º
º CY: 0 kein Fehler             º
º CY: 1 Fehler                  º
º AH: Status                    º
Û-------------------------------ì
```

Im Register CX wird die Zahl der zu transferierenden Wörter spezifiziert. Es läßt sich immer nur ein Bereich von maximal 64 Kbyte (32 Kwörter) transferieren, wodurch der maximale Wert in CX auf 8000H begrenzt ist.

Das Registerpaar ES:DI zeigt auf eine Tabelle mit Steuerparametern für den Transfer. Da die Speicherbereiche oberhalb der 1-Mbyte-Grenze liegen, erfolgt der Transfer über 24-Bit-Adressen im *Protected Modus* des 80286-Prozessors. Die Tabelle mit den Steuerparametern (Descriptor-Tabelle) enthält Einträge für die Quelle und das Ziel. Es ergibt sich folgende Struktur:

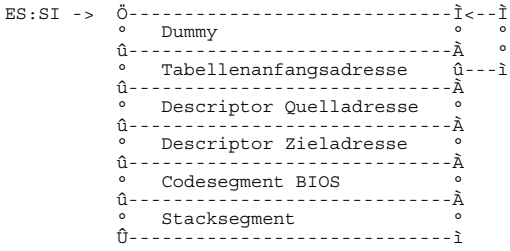


Bild 2.11: Der Aufbau der Descriptor-Tabelle bei der Move-Funktion

Das erste Feld besitzt eine Länge von 8 Byte und ist vom Anwenderprogramm mit den Werten 00H aufzufüllen (Dummy-Feld).

Ab Offset 08H findet sich ein 8-Byte-Feld mit der Segmentadresse des Datenbereiches. Das Feld ist vom Benutzer zu null zu setzen. Später findet sich hier die Anfangsadresse der Tabelle.

Die Beschreibung der Quell- und Zieladressen (Descriptoren) besteht aus je einem Feld mit folgendem Aufbau:

Bytes	Feld
2	Segmentgröße in Bytes (1-65536)
2	Low Word 24 Bit Segmentadresse
1	Highbyte 24 Bit Segmentadresse
1	Flag mit Zugriffsrechtkennung
2	reserviert

Tabelle 2.10: Der Aufbau der Descriptoren mit den Quell- und Zieladressen bei der Move-Funktion

Die Segmentgröße muß kleiner als 64 Kbyte sein. Im Feld »Zugriffsrecht**Fehler! Verweisquelle konnte nicht gefunden werden.**91H nur lesen
93H schreiben und lesen

abgespeichert.

Der Sourcedescriptor findet sich ab Offset 10H und besitzt den in Tabelle 2.10 beschriebenen Aufbau. Ab Offset 18H findet sich der Descriptor für die Zieladresse mit der gleichen Struktur.

Die restlichen zwei Descriptoren sind vom Anwender mit den Werten 00H zu füllen. Der erste dieser Descriptoren findet sich ab Offset 20H und dient zur Aufnahme eines »virtuellen**Fehler! Verweisquelle konnte nicht gefunden werden.**Nach dem Aufruf zeigt das Carry-Flag den Fehlerstatus an. Ist es gesetzt, findet sich im AH-Register der Statuscode aus Tabelle 2.11.

Code	Status
00	Transfer ohne Fehler beendet
01	RAM - Parity Error (gelöscht)
02	Exception Interrupt Error
03	Gate Address Line 20 gestört

Tabelle 2.11: Fehlercodes bei der Move-Funktion (INT 15)

Die Funktion sichert alle Registerinhalte bis auf das AX-Register. Es lassen sich also bestimmte Datensegmente im 16-Mbyte-Adreßbereich transferieren. Die Größe eines einzelnen Segments darf dabei 64 Kbyte nicht überschreiten. Quell- und Zielsegment müssen dabei eine Mindestgröße von 2*CX-1 Byte aufweisen, da sonst Daten verlorengehen.

Während des Transfers ist das komplette Interruptsystem wegen der in Kapitel 1 beschriebenen Inkompatibilitäten mit den Intel-Spezifikationen gesperrt. Die Interrupt-Descriptor-Tabelle für den 80286 »Protected ModusEs ist zu beachten, daß eventuell die Uhrzeit durch das gesperrte Interruptsystem nachgeht. Der Abgleich ist Aufgabe des Anwenderprogrammes. In der Compatibility Box von OS/2 wird die Funktion nicht unterstützt. Falls Sie auf das Extended Memory oberhalb 1 Mbyte zurückgreifen möchten, sollten Sie die Funktionen des XMS-Treibers (siehe Abschnitt 12.3) nutzen.

Extended Memory Size (AH = 88H)

Dieser Aufruf des INT 15 erlaubt die Größe eines erweiterten Speicherbereiches (Extended Memory) im Bereich oberhalb von 1 Mbyte festzustellen. Es gelten folgende Aufrufkonventionen:

CALL: INT 15
AH: 88H (Extend. Memory Size)
RETURN
CY: 0 kein Fehler
AX: Speichergröße in Kbyte
CY: 1 Fehler
AH: Status

Die Speichergröße oberhalb 1 Mbyte wird beim Systemstart ermittelt und ab Adresse 20H und 31H im CMOS-RAM des Uhrenbausteins abgelegt. Dieser Extended-Memory-Bereich kann aber erst ge-nutzt werden, wenn das Gerät mit mindestens 512 Kbyte im Grundbereich ausgerüstet ist.

Nach dem Aufruf enthält das AX-Register die Größe des Extended Memory in Kbyte, sofern das Carry-Flag gelöscht ist. Bei gesetztem Carry-Flag enthält AH einen Fehlercode (80H ungülitges Kommando, 86H Funktion wird nicht unterstützt).

Virtual Modus (AH = 89H)

Mit dem Aufruf dieser INT 15-Funktion läßt sich die 80286 CPU aus DOS heraus in den *Protected Modus* schalten. Nach dem Aufruf findet sich der Prozessor im virtuellen Modus und das im Register ES:SI übergebene Segment wird abgearbeitet. Es gelten folgende Aufrufkonventionen:

```
Ö------------------------------Ì
º      CALL:  INT 15           º
º                              º
º AH: 89H (Virtual Modus)      º
º ES:SI Zeiger auf Descriptor- º
º       Tabelle                º
º BX: Offset-INT-Tabelle       º
º CX: Offset-Codesegment       º
û------------------------------À
º      RETURN                  º
º CY: 0 -> ok                  º
º AH: 0 -> Virtual Modus ok    º
º CY: 1 -> Fehler              º
º AH: FF                       º
Û------------------------------ì
```

Beim Aufruf werden alle Register zerstört. In dem Registerpaar ES:SI wird wieder die Adresse der Descriptor-Tabelle übertragen. Diese besitzt folgenden Aufbau:

```
ES:SI ->  Ö----------------------------Ì<--Ì
          º   Dummy                    º   º
          û----------------------------À   º
          º   Tabellenanfangsadresse   º---ì
          û----------------------------À
          º   Interrupt-Tabelle        º
          û----------------------------À
          º   User Datensegment        º
          û----------------------------À
          º   User Extrasegment        º
          û----------------------------À
          º   User Stacksegment        º
          û----------------------------À
          º   User Codesegment         º
          û----------------------------À
          º   internal Codesegment     º
          Û----------------------------ì
```

Bild 2.12: Der Aufbau der Descriptor-Tabelle; im Virtual Modus

Die Descriptoren sind gemäß den Konventionen in Tabelle 2.10 aufgebaut. Die ersten 8 Byte sind mit 00 zu füllen. Die folgenden 8 Byte der Global-Descriptor-Tabelle sind ebenfalls mit 00 zu initialisieren.

Der dritte Descriptor zeigt auf die Interrupttabelle, die vom Anwenderprozeß anzulegen ist. Im Register BH findet sich der Offset in diese Tabelle, ab der die ersten 8 Hardware-Interrupts (IRQ8) beginnen. Der Wert in BL spezifiziert den Offset in die Tabelle mit den folgenden 8 Hardware-Interrupts (IRQ0).

Die weiteren Descriptoren dienen zum Spezifizieren der DS-, ES-, SS- und CS-Register im Virtual Modus. Sie sind durch den Anwenderprozeß vor dem Aufruf zu definieren.

Lediglich der letzte Descriptor wird für interne Zwecke benutzt und ist vor dem Aufruf zu null zu setzen.

Im Register CX ist die Adresse des Codesegmentes für den Protected-Mode-Betrieb anzugeben. Die CPU beginnt dort mit der Abarbeitung der Programmanweisungen. Weiterhin ist das Carry-Flag vor dem Aufruf zu löschen.

Nach dem Aufruf dieser Funktion steht kein BIOS mehr zur Verfügung, d.h., das Anwenderprogramm muß alle Ein-/Ausgabefunktionen selbst übernehmen. Auch die Interruptvektoren müssen gemäß den Konventionen des 80286 verändert und initialisiert werden. Es besteht keine offizielle Möglichkeit mehr, den 80286-Prozessor in den *Real Modus* zurückzuschalten, es sei denn, der Rechner wird abgeschaltet. Im BIOS nutzt man den Trick, einen Reset-Befehl im CMOS-RAM abzulegen und diesen dann auszuführen. Dann wird das System an dem alten BIOS-Punkt wieder aufgesetzt. Alternativ besteht die Möglichkeit, die Länge der Interrupt-Descriptor-Tabelle auf 0 zu setzen. Wird dann ein Software-Interrupt ausgeführt, gerät das System in einen illegalen Zustand (triple fault), worauf die CPU einen internen Reset ausführt.

Die Funktionen 90H (Device Busy) und 91H (Interrupt Complete Flag) sind im AT-BIOS bereits für zukünftige Erweiterungen vorgesehen, aber nicht implementiert. In erweiterten BIOS-Versionen gilt folgende Aufrufschnittstelle:

Device Busy (AH = 90H)

Mit dem Aufruf läßt sich prüfen, ob eines der angeschlossenen Geräte belegt (busy) ist. Hierbei gelten folgende Aufrufparameter.

```
Ö--------------------------------Ì
º      CALL:  INT 15             º
º                                º
º AH: 90H (Device Busy)          º
º AL: Type Code                  º
º ES:BX Zeiger auf Request       º
º       Kontrollblock            º
û--------------------------------À
º      RETURN                    º
º CY: 0 -> Gerät frei            º
º CY: 1 -> Geräte busy           º
Û--------------------------------ì
```

Vor dem Aufruf ist weiterhin das Carry-Flag zu löschen.

Der Aufruf prüft im Grunde, ob beim angeschlossenen Gerät ein Time Out aufgetreten ist. Im Register AL ist hierzu der Code für das angeschlossene Gerät zu übergeben. Dabei gilt die folgende Kodierung:

Code	Gerät
00	Festplatte (Time Out)
01	Diskettenlaufwerk (Time Out)
02	Tastatur (kein Time Out)
03	Zeiger-Einheit (Time Out)
80	Netzwerk (kein Time Out) ES:BX zeigt auf den Netzwerkkontrollblock
FC	Festplatten-Reset für PS/2 (Time Out)
FD	Diskettenlaufwerk Motorstart (Time Out)
FE	Drucker (Time Out)

Tabelle 2.12: Type Codes der INT 15-Funktion 90H

Unterstützt das BIOS den Aufruf nicht, gibt das BIOS im Register AH den Wert 86H zurück und setzt gleichzeitig das Carry-Flag.

Die Device-Typen werden in drei Gruppen unterteilt:

```
00-7FH  nicht reentrante Einheiten, Zugriffs-
        koordinierung per Betriebssystem
80-BFH  reentrante Einheiten, ES:BX zeigt dann
        auf einen Controll Block
C0-FFH  Wait only Aufrufe
```

Die Aufrufe im Bereich C0-FFH besitzen kein Äquivalent bei der Funktion 91H.

Bei Device-Typen im Bereich zwischen 80H und BFH muß ES:BX auf einen Kontrollblock zeigen. Bezieht sich der Aufruf zum Beispiel auf eine Netzwerkeinheit (Code=80H), dann ist im Registerpaar ES:BX ein Zeiger auf den zugehörigen Netzwerkkontrollblock in der Segment:Offset-Notation zu übergeben. Innerhalb des Netzwerkes wird allerdings kein Time Out gemeldet.

Nach dem Aufruf zeigt ein gesetztes Carry-Flag an, daß bei der Einheit ein Time Out aufgetreten ist. Die Funktion kann von Multitasking System abgefangen werden. Aufrufe mit den Codes zwischen C0H und FFH erlauben zum Beispiel, daß ein Task ausgeführt wird, wenn das BIOS auf das Ende einer I/O-Operation wartet.

Set Complete Interrupt Flag (AH=91H)

Mit diesem Aufruf unterstützt das erweiterte BIOS den Abschluß eines Interrupts durch Setzen des *Interrupt Complete Flag*. Dabei gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 91H (Set Interrupt Flag)  º
º AL: Type Code                 º
º ES:BX Zeiger auf Request      º
º       Kontrollblock           º
û-------------------------------À
º      RETURN                   º
º AH = 00                       º
Û-------------------------------ì
```

Der Aufruf gibt keine Parameter zurück. Im Register AL ist der Code für die gewünschte Einheit zu übergeben. Es gilt dabei die in Tabelle 2.12 gezeigte Kodierung (00 -> Disk, 01 -> Diskette etc.). Die Codes in AL lassen sich dabei gemäß folgender Tabelle in drei Gruppen zusammenfassen:

```
Ö----------Ú-----------------------------------------Ì
º  AL      º Bemerkung                                º
û----------é-----------------------------------------À
º 00 - 7FH º Einheiten, auf die vom Betriebssystem    º
º          º sequentiell zugegriffen wird. Das Be-    º
º          º triebssystem übernimmt dabei die Ko-     º
º          º ordinierung der Aufrufe.                 º
û----------é-----------------------------------------À
º 80 - BFH º Reentrante Aufrufe, die sich im wesent-  º
º          º lichen auf das Netzwerk beziehen. Es     º
º          º lassen sich daher mehrere Aufrufe zur    º
º          º gleichen Zeit bearbeiten. Die Übergabe-  º
º          º parameter sind deshalb vom rufenden Pro- º
º          º zeß in Kontrollblöcken zu speichern,     º
º          º deren Adresse als Zeiger übergeben wird. º
û----------é-----------------------------------------À
º C0 - FFH º Diese Aufrufe beziehen sich auf Verzö-   º
º          º gerungszähler (z.B. Diskette Motor       º
º          º Start oder Printer Time Out)             º
Û----------Ù-----------------------------------------ì
```

Tabelle 2.13: Zusammenfassung der Type-Codes in Gruppen

Unterstützt das BIOS den Aufruf nicht, setzt es das Carry-Flag und gibt den Wert 86H im Register AH zurück.

Get Configuration Parameter (AH=C0H)

Mit diesem Aufruf läßt sich die Adresse eines Parameterblocks mit Informationen über die BIOS- und Systemkonfiguration abfragen. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: C0H (Get Configuration)   º
û-------------------------------À
º      RETURN                   º
º CY: 1  Fehler                 º
º AL: 86 Funktion fehlt         º
º CY: 0  ok                     º
º ES:BX  Adresse Parameterblock º
Û-------------------------------ì
```

Es wird beim Aufruf der Funktionscode C0H im Register AH übergeben. Ist die entsprechende Funktion im BIOS nicht implementiert, setzt dieses das Carry-Flag und gibt in AH den Wert 86H zurück.

Bei einem erfolgreichen Aufruf ist das Carry-Flag gelöscht und das Registerpaar ES:BX enthält einen Zeiger auf den Parameterblock im BIOS. Dieser Parameterblock besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00	2	Länge der Parametertabelle in Byte
02	1	Code des Systemmodells
03	1	Unterversion des Systemmodells
04	1	BIOS-Systemrevisionsnummer
05	1	Systeminformation
		Bit 7 Disk BIOS benutzt DMA Chan. 3
		Bit 6 kaskadierter Interrupt (2)
		Bit 5 CMOS Clock vorhanden
		Bit 4 Tastaturumleitung aktiv
		Bit 3 Wait ext. Event implementiert
		Bit 2 extend. BIOS-Ber. reserviert
		Bit 1 PS/2 I/O-Channel
		Bit 0 reserviert
06	4	reserviert für Systeminformationen

Tabelle 2.14: Kodierung der Systeminformationen;

In der Tabelle sind verschiedene Informationen über die Systemhardware und die aktuelle Systemkonfiguration abgelegt. Diese lassen sich dann durch das Betriebssystem oder die Anwendersoftware abfragen.

Die Länge der Tabelle wird im ersten Byte zurückgegeben, so daß diese zukünftig variiert werden kann. Die folgenden zwei Byte enthalten eine Information über das Modell des verwendeten Rechners. Daran schließt sich die Revisionsnummer des BIOS an, die vom Hersteller vergeben wird. Der Wert 00H bezeichnet die erste Ausgabe.

Ab Offset 05H findet sich ein Byte mit Informationen über den Hardwarezustand. Jedem einzelnen Bit sind dabei bestimmte Informationen zugeordnet. So signalisiert Bit 7 = 1, daß das Festplatten-BIOS den DMA-Kanal 3 des Systems zum Datentransfer benutzt. Mit Bit 6 = 1 steht zum Beispiel fest, daß zwei Interrupt-Controller kaskadiert wurden.

Die restlichen Bytes sind für zukünftige Erweiterungen reserviert.

Return Extended BIOS-Data Area Segment Adresse (AH = C1H)

CALL: INT 15
AH: C1H (Get Adress)
RETURN
CY: 1 Fehler
CY: 0 ok
ES: Adresse

Mit der Unterfunktion AH = C1H läßt sich die Adresse des Extended-BIOS-Datensegmentes abfragen.

PS/2-BIOS Maus Interface (AH = C2H)

Die Funktionen mit dem Code AH=C2H sind nur bei IBM PS/2-Systemen implementiert und unterstützen rudimentäre Funktionen zur Ansteuerung der intergrierten Maus. Hierbei gelten folgende Unterfunktionen.

PS/2-BIOS Maus Enable/Disable(AX = C200H)

Dieser Aufruf erlaubt es, den Maustreiber aus- oder einzuschalten.

```
Ö--------------------------------Ì
°      CALL:  INT 15             °
°                                °
° AH: C2H (Get Adress)           °
° AL: 00H                        °
° BH: 00H disable                °
°     01H enable                 °
û--------------------------------À
°      RETURN                    °
° CY: 1 Fehler                   °
° AH: Status                     °
° CY: 0 ok                       °
Û--------------------------------ì
```

Nach dem Aufruf wird im Carry-Flag der Status zurückgegeben. Bei gesetztem Carry-Flag enthält AH einen Statuscode.

Für diesen Code gilt folgende Belegung:

```
Code  Status
00H   Aufruf erfolgreich
01H   ungültige Funktion
02H   ungültige Eingabe
03H   Interface Fehler
04H   Abfrage muß wiederholt werden
05H   kein Device Treiber installiert
```

Weitere Informationen enthalten die Unterlagen über das PS/2-BIOS.

PS/2-BIOS Maus Reset (AX = C201H)

Dieser Aufruf erlaubt es, den Maustreiber des BIOS zurückzusetzen.

```
Ö--------------------------------Ì
°      CALL:  INT 15             °
°                                °
° AH: C2H                        °
° AL: 01H (Reset)                °
û--------------------------------À
°      RETURN                    °
° CY: 1 Fehler                   °
° AH: Status                     °
° CY: 0 ok                       °
° BH: Device ID                  °
Û--------------------------------ì
```

Nach dem Aufruf wird im Carry-Flag der Status zurückgegeben. Bei gesetztem Carry-Flag enthält AH einen Statuscode gemäß Funktion C200H. Bei gelöschten Carry-Flag wurde der Treiber zurückgesetzt und in BH findet sich die ID-Nummer.

PS/2-BIOS Set Sampling Rate(AX = C202H)

Dieser Aufruf erlaubt es, die Abtastrate des Maustreibers im BIOS zu setzen.

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: C2H                       º
º AL: 02H (Set Sampling Rate)   º
º BH: Sampling Rate             º
û-------------------------------À
º      RETURN                   º
º CY: 1 Fehler                  º
º AH: Status                    º
º CY: 0 ok                      º
Û-------------------------------ì
```

Im Register BH ist die Abtastrate für den Maustreiber zu übergeben. Nach dem Aufruf wird im Carry-Flag der Status zurückgegeben. Bei gesetztem Carry-Flag enthält AH einen Statuscode gemäß Funktion C200H. Bei gelöschten Carry-Flag wurde die neue Abtastrate eingestellt.

Für die in BH übergebene Abtastrate gilt folgende Codierung:

```
BH = 00H Sampling Rate 10/Sekunde
     01H Sampling Rate 20/Sekunde
     02H Sampling Rate 40/Sekunde
     03H Sampling Rate 60/Sekunde
     04H Sampling Rate 80/Sekunde
     05H Sampling Rate 100/Sekunde
     06H Sampling Rate 200/Sekunde
```

PS/2-BIOS Set Resolution(AX = C203H)

Dieser Aufruf erlaubt es, die Auflösung des Maustreibers im BIOS zu setzen.

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: C2H                       º
º AL: 03H (Set Resulution)      º
º BH: Resolution                º
û-------------------------------À
º      RETURN                   º
º CY: 1 Fehler                  º
º AH: Status                    º
º CY: 0 ok                      º
Û-------------------------------ì
```

Im Register BH ist die Auflösung in mm für den Maustreiber zu übergeben. Nach dem Aufruf wird im Carry-Flag der Status zurückgegeben. Bei gesetztem Carry-Flag enthält AH einen Statuscode gemäß Funktion C200H. Bei gelöschten Carry-Flag wurde die neue Auflösung eingestellt.

Für die in BH übergebene Auflösung gilt folgende Codierung:

```
BH = 00H 1 Count  pro Millimeter
     01H 2 Counts pro Millimeter
     02H 4 Counts pro Millimeter
     03H 8 Counts pro Millimeter
```

PS/2-BIOS Get Type(AX = C204H)

Dieser Aufruf erlaubt es, den Typ des Treibers abzufragen.

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: C2H                       º
º AL: 04H (Get Type)            º
û-------------------------------À
º      RETURN                   º
º CY: 1 Fehler                  º
º AH: Status                    º
º CY: 0 ok                      º
º BH: Type                      º
Û-------------------------------ì
```

Im Register BH wird bei fehlerfreiem Aufruf (Carry-Flag = 0) der Type zurückgegeben.

PS/2-BIOS Initialize(AX = C205H)

Dieser Aufruf erlaubt die Initialisierung des Maustreibers.

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: C2H                       º
º AL: 05H (Initialize)          º
º BH: Paketgröße                º
û-------------------------------À
º      RETURN                   º
º CY: 1 Fehler                  º
º AH: Status                    º
º CY: 0 ok                      º
Û-------------------------------ì
```

Der Fehlerstatus wird im Carry-Flag zurückgegeben. Es gilt die Fehlercodierung wie bei der Funktion C200H.

PS/2-BIOS Get/Set Scaling Faktor (AX = C206H)

Dieser Aufruf erlaubt es, die Skalierungsfaktoren des Maustreibers im BIOS zu setzen oder abzufragen.

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: C2H                       º
º AL: 06H (Get/Set Scale Faktor)º
º BH: Subfunktion               º
û-------------------------------À
º      RETURN                   º
º CY: 1 Fehler                  º
º AH: Status                    º
º CY: 0 ok                      º
º BL: Treiber Status            º
º CL: Auflösung                 º
º DL: Abtastrate                º
Û-------------------------------ì
```

Im Register BH ist der Subcode für die Unterfunktion zu übergeben. Nach dem Aufruf wird im Carry-Flag der Status zurückgegeben. Bei gesetztem Carry-Flag enthält AH einen Statuscode gemäß Funktion C200H. Bei gelöschten Carry-Flag wurde die Funktion fehlerfrei ausgeführt.

Für die in BH übergebenen Untercodes gilt folgendes:

```
BH = 00H Return Device Status
 Rückgabe
 in BL:
    Bit 0: rechte Taste gedrückt
    Bit 1: reserviert
    Bit 2: linke Taste gedrückt
    Bit 3: reserviert
    Bit 4: 0 = 1:1 Skalierung, 1 = 2:1 Skalierung
    Bit 5: Device enabled
    Bit 6: 0 = Stream Mode, 1 = Remote Mode
    Bit 7: reserviert
 in CL: Auflösung (s. C203H)
 in DL: Abtastrate (Reports / Sekunde)

BH = 01H Set Scaling Rate 1:1
BH = 02H Set Scaling Rate 2:1
```

Weitere Informationen sind den BIOS-Unterlagen zu entnehmen.

PS/2-BIOS Set Device Handler Adresse (AX = C207H)

Dieser Aufruf erlaubt es, einen neuen Treiber aus einer Anwendung zu installieren.

```
Ö--------------------------------Ì
º      CALL:  INT 15             º
º                                º
º AH: C2H                        º
º AL: 07H (Set Device Handler)   º
º ES:BX: Handler Adresse         º
û--------------------------------À
º      RETURN                    º
º CY: 1 Fehler                   º
º AH: Status                     º
º CY: 0 ok                       º
Û--------------------------------ì
```

Im Registerpaar ES:BX ist die Segment:Offset-Adresse des neuen Treibers zu übergeben. Ist das Carry-Flag nach dem Aufruf gelöscht, wurde die neue Routine als Maushandler aktiviert. Andernfalls findet sich in AH der Fehlerstatus mit der Codierung gemäß Funktion C200H.

Die Funktionen AH = C3H und C4H sind vom BIOS der PS/2 Modelle 50 bis 80 belegt. Näheres findet sich in den BIOS-Unterlagen des Herstellers.

EISA-ROM Funktionen(AH = D8H)

Dieser Aufruf des INT 15 erlaubt bei Systemen mit dem EISA-Bus verschiedene Statusabfragen der installierten Adapterplatinen. Im folgenden werden die entsprechenden Aufrufe beschrieben.

EISA-ROM Read Slot Configuration(AX = D800H)

Diese Unterfunktion erlaubt die Überprüfung der Konfigurierung des Bussystems. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 15             º
º                                º
º AH: D8H                        º
º AL: 00H (Read Configuration)   º
º CL: Slot Nummer                º
û--------------------------------À
º      RETURN                    º
º CY: 1 Fehler                   º
º AH: Status                     º
º CY: 0 ok                       º
º AL: Bit Flags                  º
º BX: Version                    º
º CX: Checksum                   º
º DX: Infobytes                  º
º SI:DI 4 Byte ID                º
Û--------------------------------ì
```

Beim Aufruf ist im Register CL die Nummer des abzufragenden Steckplatzes (Slot) zu übergeben. Bei einem fehlerfreien Aufruf (Carry-Flag = 0) enthalten die angegebenen Register die Informationen.

In AL wird ein Bitfeld mit folgender Kodierung zurückgegeben:

```
AL Bit 7   : 1 bei duplicate ID's
   Bit 6   : 1 Produkt ID lesbar
   Bit 4,5 : Typ des Steckplatzes (Slots)
             00 = expansion, 01 = embedded
             10 = virtual device
   Bit 0-3 : duplicate ID-Nummer, falls Bit 7
             gesetzt ist.
```

Jeder Steckplatz muß eine Karte mit einer eindeutigen Busnummer besitzen. Ist dies nicht der Fall, wird Bit 7 gesetzt und die Nummer ist in den Bits 0 bis 3 enthalten.

Im Register BX wird die Versionnummer der Konfigurierutility zurückgegeben. Hierbei gilt:

```
BH: Hauptrevisionsnummer Konfigurier-Utility
BL: Unterversionsnummer  Konfigurier-Utility
```

In CX findet sich nach einem fehlerfreien Aufruf die Checksumme der Konfigurierungsdatei. Das Register DH enthält die Nummer der Device Funktion, erlaubt also die Identifikation, um welchen Kartentyp es sich handelt. DL enthält ein Informationsbyte, dessen Kodierung zur Zeit nicht bekannt ist.

In SI:DI wird eine komprimierte 4 Byte lange ID-Nummer zurückgegeben. Für die Belegung gelten die üblichen Intel Konventionen. Wird die Funktion im 16-Bit-Mode aufgerufen, ist der Code D800H in AX zu übergeben. Bei 32-Bit-CS-Adressen ist der Code in AX in D880H zu ändern.

Ist das Carry-Flag nach dem Aufruf gesetzt, liegt ein Fehler vor. Der entsprechende Code wird in AH zurückgegeben:

```
Fehlercode in AH
80H  ungültige Slotnummer
82H  EISA CMOS Inhalt ungültig
83H  leerer Slot
86H  ungültiger BIOS-Firmware Funktionsaufruf
87H  ungültige Systemkonfiguration
```

Bei gelöschtem Carry-Flag ist der Inhalt von AH auf den Wert 00H zurückgesetzt.

EISA-ROM Read Funktion Configuration(AX = D801H)

Diese Unterfunktion erlaubt die Abfrage der Konfigurierinformationen eines Busteilnehmers. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 15             º
º                                º
º AH: D8H                        º
º AL: 01H (Read Configuration)   º
º CH: Funktionsnummer            º
º CL: Slot Nummer                º
º DS:SI Bufferadresse            º
û--------------------------------À
º      RETURN                    º
º CY: 1 Fehler                   º
º AH: Status                     º
º CY: 0 ok                       º
º DS:SI Buffer                   º
Û--------------------------------ì
```

Beim Aufruf ist im Register CH eine Nummer für die zu lesende Funktion zu übergeben. CL enthält die Nummer des abzufragenden Steckplatzes (Slot). Weiterhin muß in DS:SI die Adresse eines 320 Byte umfassenden Ergebnisspuffers für den Standardkonfigurationsdatenblock angegeben werden. Bei einem fehlerfreien Aufruf (Carry-Flag = 0) enthalten die angegebenen Register die Informationen.

Bei gesetztem Carry-Flag enthält AH den Fehlercode mit der Belegung gemäß Funktion D800H. Beim Aufruf der Funktion wird der Inhalt von BX zerstört. Ein Einsatz im 32-Bit-Mode erfordert beim Aufruf in AL den Wert 81H.

EISA-ROM Clear EISA-CMOS-RAM(AX = D802H)

Diese Unterfunktion erlaubt es, daß EISA-CMOS-RAM zu löschen. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 15             º
º                                º
º AH: D8H                        º
º AL: 02H (Clear CMOS-RAM)       º
º BX: Revision Level             º
û--------------------------------À
º      RETURN                    º
º CY: 1 Fehler                   º
º AH: Status                     º
º CY: 0 ok                       º
º AH: 00H                        º
Û--------------------------------ì
```

Beim Aufruf ist im Register BX die Revisionsnummer der Konfigurierutility zu übergeben. BH einhält die Hauptversion, während BL die Unterversion aufnimmt.

Bei fehlerfreiem Aufruf wird das CMOS-RAM gelöscht und das Carry-Flag ist nach der Rückkehr auf 0 gesetzt. Bei gesetztem Carry-Flag liegt ein Fehler vor. Dann gilt folgende Belegung:

```
Fehlercode in AH
84H  Fehler beim Löschen des CMOS-RAM's
86H  ungültiger BIOS-Firmware Funktionsaufruf
88H  Konfigurierutility Version nicht unterstützt
```

Wird die Funktion im 32-Bit-Mode aufgerufen, ist in AL der Code 82H zu übergeben.

EISA-ROM Write EISA-CMOS-RAM(AX = D803H)

Diese Unterfunktion erlaubt es, daß EISA-CMOS-RAM zu beschreiben. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: D8H                       º
º AL: 03H (Write CMOS-RAM)      º
º CX: Länge Daten               º
º DS:SI Datenpuffer             º
û-------------------------------À
º      RETURN                   º
º CY: 1 Fehler                  º
º AH: Status                    º
º CY: 0 ok                      º
º AH: 00H                       º
Û-------------------------------ì
```

Beim Aufruf ist im Register CX die Zahl der Datenbytes im Puffer anzugeben. Ein Wert von 0000H bedeutet, daß ein Slot leer ist.

Die Konfigurierungsdaten befinden sich in einem Puffer, dessen Adresse im Registerpaar DS:SI übergeben wird.

Bei fehlerfreiem Aufruf werden die Daten in das CMOS-RAM übertragen und das Carry-Flag ist nach der Rückkehr auf 0 gesetzt. Bei gesetztem Carry-Flag liegt ein Fehler vor. Dann gilt folgende Belegung:

```
Fehlercode in AH
84H  Fehler beim Löschen des CMOS-RAM's
85H  EISA-CMOS-RAM ist voll
86H  ungültiger BIOS-Firmware Funktionsaufruf
```

Wird die Funktion im 32-Bit-Mode aufgerufen, ist in AL der Code 82H zu übergeben.

EISA-ROM Read Physical Slot(AX = D804H)

Diese Unterfunktion erlaubt die Überprüfung der Daten einer Steckkarte. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: D8H                       º
º AL: 04H (Read Configuration)  º
º CL: Slot Nummer               º
û-------------------------------À
º      RETURN                   º
º CY: 1 Fehler                  º
º AH: Status                    º
º CY: 0 ok                      º
º SI:DI 4 Byte ID               º
Û-------------------------------ì
```

Beim Aufruf ist im Register CL die Nummer des abzufragenden Steckplatzes (Slot) zu übergeben. Hierbei kann es sich auch um eine embedded oder virtual Slotnummer handeln. Bei einem fehlerfreien Aufruf (Carry-Flag = 0) wird in den Registern SI:DI eine komprimierte 4-Byte-ID-Nummer zurückgegeben.

Bei gesetztem Carry-Flag liegt ein Fehler vor. Dann gilt folgende Belegung:

```
Fehlercode in AH
80H  ungültige Slot Nummer
83H  leerer Slot
86H  ungültiger BIOS-Firmware Funktionsaufruf
```

Wird die Funktion im 32-Bit-Mode aufgerufen, ist in AL der Code 84H zu übergeben.

EISA-ROM 32-Bit CS-Adress-Mode-Calls(AX = D88xH)

Diese Unterfunktionen sind identisch zu den Funktionen D800H und D804H. Sie sollten aber verwendet werden, falls das System im 32-Bit-Mode betrieben wird. Dann nutzen die Zeiger das Register ESI an Stelle des SI-Registers.

Compac Systempro Multiprozessor(AX = ExxxH)

Einige Funktionen im Bereich ExxxH werden von Compac für Multiprozessor Operationen genutzt. Nähere Hinweise finden sich in den Compac BIOS-Unterlagen.

2.8 Tastatur abfragen (INT 16)

Die BIOS-Funktion erlaubt es, den Status des Tastaturpuffers abzufragen sowie Zeichen aus dem Puffer zu lesen. Die Steuerung erfolgt über das AH-Register, mit dem sich die einzelnen Funktionen selektieren lassen. Beim Aufruf werden alle Register bis auf AX erhalten.

Zeichen aus Puffer lesen (AH = 00H)

Mit der Unterfunktion AH = 0 läßt sich ein Zeichen aus dem Tastaturpuffer lesen. Es gelten folgende Aufrufkonventionen.

```
Ö--------------------------------Ì
º      CALL:  INT 16             º
º                                º
º AH: 00H (Keyboard Read)        º
û--------------------------------À
º      RETURN                    º
º AH: Scan Code                  º
º AL: Zeichen                    º
Û--------------------------------ì
```

Das ASCII-Zeichen findet sich in AL, während das AH-Register den Scan Code enthält. Dies ist eine interne Darstellung der Zeichen auf der Tastatur.

Status des Puffers abfragen (AH = 01H)

Hiermit kann der Status des Tastaturpuffers abgefragt werden.

```
Ö-------------------------------Ì
º      CALL:  INT 16            º
º                               º
º AH: 01H (Get Status)          º
û-------------------------------À
º      RETURN                   º
º AH: Scan Code                 º
º AL: Zeichen                   º
º Zero Flag:                    º
º  1 : kein Zeichen im Puffer   º
º  0 : Zeichen im Puffer        º
Û-------------------------------ì
```

Das Ergebnis wird über das Zero-Flag angezeigt. Ist dieses Flag gesetzt, liegen keine Zeichen im Tastaturpuffer vor. Falls Z = 0 ist, wird im AX-Register das Zeichen und der Scan Code (wie bei Funktion AH = 0) zurückgegeben. Das Zeichen bleibt aber im Puffer erhalten. Mit dieser Funktion läßt sich offenbar der Puffer auf Zeichen abfragen, ohne daß diese zerstört oder entfernt werden.

Shift-Status abfragen (AH = 02H)

Mit dieser Abfrage läßt sich prüfen, ob die Shift- oder Alt-Tasten gedrückt sind. Im BIOS-Datenbereich ist hierfür ein eigenes Statusbyte reserviert.

```
Ö-------------------------------Ì
º      CALL  :  INT 16          º
º                               º
º AH: 02H (Read Shift Status)   º
û-------------------------------À
º      RETURN                   º
º AL: Alt Keyboard Flag         º
Û-------------------------------ì
```

Das Ergebnis wird im AL-Register zurückgegeben. Es gilt folgende Belegung:

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙÙÙÙÚÙÙÚÚÙÙÙÙÚi
 º º º º º º º Û-  1 = rechte Shift-Taste gedrückt
 º º º º º º Û---  1 = linke Shift-Taste gedrückt
 º º º º º Û-----  1 = CTRL-Taste gedrückt
 º º º º Û-------  1 = ALT-Taste gedrückt
 º º º Û---------  1 = SCROLL-LOCK-Status Ein
 º º Û-----------  1 = NUM-LOCK-Status Ein
 º Û-------------  1 = CAPS-LOCK-Status Ein
 Û---------------  1 = Insert-Modus Ein
```

Bild 2.13: Kodierung des Keyboard-Statusflags

Durch Auswertung des Statusbytes läßt sich feststellen, ob bestimmte Tastenkombinationen gedrückt wurden.

Wiederholrate ändern (AH = 03H)

Bei moderneren BIOS-Versionen existiert die vielfach nicht dokumentierte Möglichkeit, die Wiederholrate der Tastatur zu verändern. Die Funktion wird mit dem Code 03H in AH aufgerufen und besitzt folgende Schnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 16            º
º                               º
º AH: 03H (Repeat Rate)         º
º AL: 05H                       º
º BL: Wiederholrate             º
º BH: Verzögerung in ms         º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Im Register AL ist immer der Wert 05H zu übergeben. Das Register BH enthält einen Code für die Verzögerungszeit in Millisekunden, vom Beginn des Tastendrucks, bis die Repeatfunktion aktiviert wird. Es sind die Werte erlaubt:

```
Ö--------Ú----------------------Ì
º   Code º Verzögerungszeit (ms) º
û--------é----------------------À
º     00 º 250                   º
º     01 º 500                   º
º     02 º 750                   º
º     03 º 1000                  º
Û--------Ù----------------------ì
```

In BL ist ein Code für die Wiederholrate (Zeichen/Sekunde) zu übergeben. Dabei gilt folgende Belegung:

```
Ö------Ú---------------Ú------Ú---------------Ì
º Code º Wiederholrate º Code º Wiederholrate º
û------é---------------é------é---------------À
º  00  º 30.0          º  10  º  7.5          º
º  01  º 26.7          º  11  º  6.7          º
º  02  º 24.0          º  12  º  6.0          º
º  03  º 21.8          º  13  º  5.5          º
º  04  º 20.0          º  14  º  5.0          º
º  05  º 18.5          º  15  º  4.6          º
º  06  º 17.1          º  16  º  4.3          º
º  07  º 16.0          º  17  º  4.0          º
º  08  º 15.0          º  18  º  3.7          º
º  09  º 13.3          º  19  º  3.3          º
º  0A  º 12.0          º  1A  º  3.0          º
º  0B  º 10.9          º  1B  º  2.7          º
º  0C  º 10.0          º  1C  º  2.5          º
º  0D  º  9.2          º  1D  º  2.3          º
º  0E  º  8.6          º  1E  º  2.1          º
º  0F  º  8.0          º  1F  º  2.0          º
Û------Ù---------------Ù------Ù---------------ì
```

Zeichenfolge speichern (AH = 05H)

Bei moderneren BIOS-Versionen existiert eine weitere Funktion zum Speichern von Zeichenfolgen im System. Hierbei gelten folgende Übergabekonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 16            º
º                               º
º AH: 05H (Zeichen speichern)   º
º CH: Scan Code                 º
º CL: ASCII-Code                º
û-------------------------------À
º      RETURN                   º
º AL: Statuscode                º
Û-------------------------------ì
```

In den Registern CL und CH werden das Zeichen und der zugehörige Scan Code übergeben. Dabei gelten die gleichen Werte wie bei der Funktion AH = 00H. Nach dem Aufruf enthält das Register AL einen Statuscode:

```
Code ° Status
-----é-----------------------------------------
 00  ° Das Zeichen wurde im Puffer gespeichert
 01  ° Zeichen nicht gespeichert, da Puffer voll
```

In den Puffer lassen sich maximal 15 Zeichen ablegen. Allerdings können parallel von der Tastatur Zeichen einlaufen, so daß die Puffergröße nochmals reduziert wird. Mit dem Aufruf lassen sich Tastaturmakros recht einfach programmieren.

MF2-Tastatur abfragen (AH = 10H)

Dieser Aufruf ist nur in BIOS-Version der AT 03- und PS/2-Modelle implementiert und erlaubt die Sondertasten der MF2-Tastaturen abzufragen. Es gilt folgende Schnittstelle:

```
Ö-------------------------------Ì
°      CALL:  INT 16            °
°                               °
° AH: 10H (MF2-Tastatur)        °
û-------------------------------À
°      RETURN                   °
° AH: Scan Code                 °
° AL: ASCII-Zeichen             °
Û-------------------------------ì
```

Die Funktion gibt den Scan-Code und das Zeichen im Register AX zurück.

Zeichen in Puffer (AH = 11H)

Dieser Aufruf ist nur in BIOS-Version der AT 03- und PS/2-Modelle implementiert und erlaubt die Prüfung, ob Zeichen im Puffer vorliegen. Es gilt folgende Schnittstelle:

```
Ö-------------------------------Ì
°      CALL:  INT 16            °
°                               °
° AH: 11H (Check Puffer)        °
û-------------------------------À
°      RETURN                   °
° ZF: 0 ->  Zeichen gelesen     °
° AH: Scan Code                 °
° AL: ASCII-Zeichen             °
Û-------------------------------ì
```

Die Funktion gibt den Scan-Code und das Zeichen im Register AX zurück. Die Werte sind jedoch nur definiert, falls das Zero-Flag nach dem Aufruf gelöscht ist. Andernfalls war der Puffer leer und es werden keine Zeichen zurückgegeben.

Check extended Status (AH = 12H)

Dieser Aufruf ist nur in BIOS-Version der AT 03- und PS/2-Modelle implementiert und erlaubt die Prüfung, ob die Caps-, Num- oder Scroll-Lock-Taste gedrückt ist. Es gilt folgende Schnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 16            º
º                               º
º AH: 12H (Check Status)        º
û-------------------------------À
º      RETURN                   º
º AX: Status der Tasten         º
Û-------------------------------ì
```

Die Funktion 02H gibt nur die Information zurück, ob die Tasten Caps-, Num- oder Scroll-Lock gerade aktiv sind, d.h. irgendwann gedrückt wurden. Die Funktion 12H prüft jedoch, ob die Tasten aktuell gedrückt sind. Dabei gilt folgende Kodierung des im Register AH zurückgegebenen Statusbytes.

```
Bit 7 6 5 4 3 2 1 0
   Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
   ÛÚÙÚÙÚÙÚÙÚÙÚÙÚÙÚì
    º º º º º º º Û-  linke CTRL-Taste gedrückt
    º º º º º º Û---  linke ALT-Taste gedrückt
    º º º º º Û-----  rechte CTRL-Taste gedrückt
    º º º º Û-------  rechte ALT-Taste gedrückt
    º º º Û---------  SCROLL LOCK gedrückt
    º º Û-----------  NUM LOCK gedrückt
    º Û-------------  CAPS LOCK gedrückt
    Û---------------  SYS-REQUEST-Taste gedrückt
```

Die Kodierung des Registers AL entspricht der Belegung von Bild 2.13.

Bei Compac-Rechnern werden die Funktionscodes FxxxH des INT 16 durch das BIOS verwaltet. Hier läßt sich zum Beispiel die CPU-Geschwindigkeit abfragen oder setzen. Weiterhin benutzen die PC Tools mit dem DESKTOP API den INT 16 mit den Funktionscodes FFxxH für bestimmte Befehle.

2.9 Druckerschnittstelle (INT 17)

Zur Steuerung der parallelen Druckerschnittstellen existiert der BIOS-Interrupt 17. Ähnlich wie bei den seriellen Schnittstellen selektiert der Registerinhalt in DX die betreffende Einheit:

```
DX º Drucker
---é--------
0  º LPT1:
1  º LPT2:
2  º LPT3:
```

Über das AH-Register werden dann die Unterfunktionen aktiviert. Alle Register, bis auf AH, bleiben beim Aufruf erhalten.

Zeichen ausgeben (AH = 00H)

Das im AL-Register befindliche Zeichen wird über die Schnittstelle ausgegeben. Die Schnittstelle wird durch den Inhalt des DX-Registers ausgewählt (0 = LPT1:, 1 = LPT2: etc.).

```
Ö------------------------------Ì
°      CALL:  INT 17           °
°                              °
° AH: 00H  (Print Character)   °
° AL: Zeichen                  °
° DX: Schnittstelle            °
û------------------------------À
°      RETURN                  °
° AH: Statusbyte               °
Û------------------------------ì
```

Das Ergebnis dieser Operation findet sich im AH-Register. Falls AH = 1 gesetzt ist, konnte das Zeichen nicht ausgegeben werden. Die Kodierung des Statusbytes findet sich unter der Funktion (02H Status lesen).

Printer Port initialisieren (AH = 01H)

Bei diesem Aufruf wird die Schnittstelle für die Ausgabe initialisiert. Es gelten folgende Aufrufkonventionen.

```
Ö------------------------------Ì
°      CALL:  INT 17           °
°                              °
° AH: 01H  (Init Port)         °
° DX: Schnittstelle            °
û------------------------------À
°      RETURN                  °
° AH: Statusbyte               °
Û------------------------------ì
```

Die Schnittstelle wird durch das Register DX selektiert (0 = LPT1:, 1 = LPT2:, 2 = LPT3:). Im AH-Register wird anschließend der Status zurückgegeben.

Printer Status lesen (AH = 02H)

Der Status eines angeschlossenen Druckers läßt sich mit dieser Funktion abfragen. Das Ergebnis wird im AH-Register zurückgegeben.

```
Ö------------------------------Ì
°      CALL:  INT 17           °
°                              °
° AH: 02H  (Read Status)       °
° DX: Schnittstelle            °
û------------------------------À
°      RETURN                  °
° AH: Statusbyte               °
Û------------------------------ì
```

Die Belegung der einzelnen Bits entspricht nachfolgendem Bild.

```
  7 6 5 4 3 2 1 0
 Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
 ÛÚÙÚÙÚÙÚÙÚÙÚÙÚÙÚì
  ° ° ° ° ° ° ° Û-  Time out
  ° ° ° ° ° ° Û---  ---
  ° ° ° ° ° Û-----  ---
  ° ° ° ° Û-------  I/O Error
  ° ° ° Û---------  Drucker selektiert
  ° ° Û-----------  Papierende
  ° Û-------------  Acknowledge
  Û---------------  not busy
```

Bild 2.14: Belegung des Drucker-Statusbyte;s

Die Bits 1, 2 sind im Statusbyte nicht belegt.

2.10 ROM-Basic (INT 18)

Dieser Interrupt ist beim IBM-PC für das ROM-Basic vorgesehen. Es bleibt bei den meisten kompatiblen PCs unbelegt und wird deshalb auch nicht weiter besprochen.

2.11 Bootstrap (INT 19)

Der Interrupt aktiviert die Bootstrap-Routine im BIOS-ROM. Diese lädt den Code der Systemdiskette von Spur 0, Sektor 0 in den Speicher ab Adresse 0:700. Anschließend wird das Programm ab dieser Adresse ausgeführt. Damit wird faktisch das MS-DOS-Betriebssystem neu geladen. Um einen Warmstart auszuführen muß im BIOS-RAM ab Adresse 0040:0072H der Wert 1234H eingetragen werden. Fehlt dieser Wert im BIOS, führt das System einen Kaltstart aus. Das PC-DOS-Programm VDISK.SYS benutzt diesen Interrupt zur Kommunikation mit einer RAM-Disk. Konkret wird über den INT 19 abgefragt, wieviel Extended Memory durch VDISK belegt wurde. Die 3 Bytes ab Offset 2CH im VDISK-INT 19-Treiber enthalten die Adresse des ersten freien Bytes im Extended Memory.

2.12 Systemzeit (INT 1A)

Das BIOS führt die Systemzeit als 4-Byte-Variable im BIOS-Datenbereich. Sie wird aus dem Timer-Interrupt gewonnen. Mit dem INT 1A läßt sich der Wert dieser Zeitvariablen lesen und stellen. Hierzu gelten folgende Konventionen:

Zeit lesen (AH = 00H)

Mit diesem Aufruf läßt sich der Inhalt der Timervariablen lesen.

```
Ö-------------------------------Ì
°      CALL:  INT 1A            °
°                               °
° AH: 04 (Get Time)             °
û-------------------------------À
°      RETURN                   °
° CX: Zähler High Word          °
° DX: Zähler Low Word           °
° AL: Übertragsflag             °
Û-------------------------------ì
```

Die Uhrzeit wird in Ticks zurückgegeben (siehe auch Zeit setzen). Im Register AL wird ein Flag zurückgegeben, welches einen 24-Stunden-Zeitüberlauf anzeigt. AL = 0 bedeutet, daß kein 24-StundenÜbertrag seit dem letzten Lesen der Zeit aufgetreten ist. Bei AL = 1 liegt ein Überlauf vor.

Zeit setzen (AH = 01H)

Mit dieser Unterfunktion läßt sich die Timervariable im BIOS-Datenbereich setzen. Es gilt folgende Aufrufkonvention:

```
Ö--------------------------------Ì
º      CALL:  INT 1A             º
º                                º
º AH: 01H (Set Time)             º
º CX: Zähler High Word           º
º DX: Zähler Low Word            º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Im Register CX findet sich das Timer-High-Word, welches die Zeit in Stunden (0-23) angibt. Das Register DX enthält die Zeit in Ticks vom Grundtakt. Beim IBM-PC sind dies 1/18,2 Sekunden. Mittlerweile besitzen viele PCs CMOS-Uhrenbausteine mit Batteriepufferung, die dann meist auch durch das BIOS-ROM unterstützt werden. Damit finden sich bei diesen Systemen erweiterte Funktionsaufrufe beim -INT 1A.

Read Real Time Clock (AH = 02H)

Mit diesem Aufruf läßt sich die interne Zeit der CMOS-Uhr lesen. Es gelten folgende Aufrufkonventionen.

```
Ö--------------------------------Ì
º      CALL:  INT 1A             º
º                                º
º AH: 02H (Read RTC Time)        º
û--------------------------------À
º      RETURN                    º
º CY:1 -> keine RTC              º
º CY:0 -> RTC gefunden           º
º CH: Stunden im BCD-Format      º
º CL: Minuten im BCD-Format      º
º DH: Sekunden im BCD-Format     º
Û--------------------------------ì
```

Die Werte werden in BCD-Notation zurückgegeben. Dies ist zu beachten, falls die Werte in den BIOS-Datenbereich geschrieben werden sollen (AH = 01H Set Clock).

Set Real Time Clock (AH = 03H)

Dieser Funktionsaufruf erlaubt es, die interne Zeit der CMOS-Uhr zu setzen. Es gelten folgende Aufrufkonventionen:

```
Ö--------------------------------Ì
º      CALL:  INT 1A             º
º                                º
º AH: 03H (Set RTC Time)         º
º CH: Stunden im BCD-Format      º
º CL: Minuten im BCD-Format      º
º DH: Sekunden im BCD-Format     º
º DL: 1 = 24 Std.  0 = 12 Std.   º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Die Werte werden in BCD-Notation übergeben. Im Register DL wird das Zeitformat eingestellt. Der Wert 1 setzt die Uhr auf die 24-Stunden-Anzeige, während bei DL = 0 nach

12 Stunden ein Uhrzeitüberlauf auftritt. Falls keine CMOS-Clock vorhanden ist, bleibt der Aufruf ohne Folgen.

Read Date from RTC (AH = 04H)

Mit diesem Aufruf läßt sich das Datum der Real-Time-Clock lesen. Es gelten folgende Aufrufkonventionen.

```
Ö-------------------------------Ì
º      CALL:  INT 1A            º
º                               º
º AH: 04H (Read RTC Date)       º
û-------------------------------À
º      RETURN                   º
º CY:1 -> keine RTC             º
º CY:0 -> RTC gefunden          º
º CH: Jahrhundert im BCD-Format º
º CL: Jahr im BCD-Format        º
º DH: Monat im BCD-Format       º
º DL: Tag im BCD-Format         º
Û-------------------------------ì
```

Die Werte werden in BCD-Notation zurückgegeben. Dies ist zu beachten, falls die Werte weiterverarbeitet werden sollen. Das Ergebnis im Jahrhundertfeld bewegt sich zwischen 19 und 20, während die restlichen Werte im gewohnten Rahmen liegen.

Set Date into Real Time Clock (AH = 05H)

Dieser Funktionsaufruf erlaubt es, das interne Datum der CMOS-Uhr zu setzen. Es gelten folgendende Aufrufkonventionen.

```
Ö-------------------------------Ì
º      CALL:  INT 1A            º
º                               º
º AH: 05H (Set RTC Date)        º
º CH: Jahrhundert im BCD-Format º
º CL: Jahr im BCD-Format        º
º DH: Monat im BCD-Format       º
º DL: Tag im BCD-Format         º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Die Werte werden in BCD-Notation übergeben. Der Wert in CH muß auf 19 oder 20 gesetzt werden. Falls keine CMOS-Clock vorhanden ist, bleibt der Aufruf ohne Folgen.

Set the Alarm (AH = 06H)

Neuere BIOS-Versionen erlauben es, bestimmte Zeitaufträge an die CMOS-Uhr abzusetzen. Nach Ablauf dieser Zeit wird ein Alarm ausgelöst. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 1A            º
º                               º
º AH: 06H (Set Alarm)           º
º CH: Stunde im BCD-Format      º
º CL: Minuten im BCD-Format     º
º DH: Sekunden im BCD-Format    º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Die Werte werden in BCD-Notation übergeben. Es ist möglich, Zeiten zwischen 00:00:00 und 23:59:59 Uhr anzugeben. Die Werte werden in den Uhrzeit-baustein übertragen. Sobald die Alarmzeit erreicht wird, löst der Baustein per Hardware (Level 8) einen INT 50 aus. Diese Routine behandelt dann den Alarm.

Das Anwenderprogramm muß den INT 4A-Vektor auf eine eigene Routine setzen und so die Alarmbedingung weiterverarbeiten.

Reset the Alarm (AH = 07H)

Falls mit dem Aufruf 06H (Set Alarm) ein entsprechender Auftrag abgesetzt wurde, läßt sich dieser mittels der Funktion 07H löschen. Es gelten folgende Aufrufkonventionen.

```
Ö-------------------------------Ì
º      CALL:  INT 1A            º
º                               º
º AH: 07H (Reset Alarm)         º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Die BIOS-Funktion löscht dann den Auftrag aus der CMOS-Uhr (Reset).

In einigen Systemen sind weitere Funktionen im Bereich AH = 08H bis 0BH implementiert um einzelne spezielle Funktionen des RTC-Bausteins anzusprechen. Da diese Funktionen aber nicht allgemein vorhanden sind, möchte ich sie an dieser Stelle übergehen.

2.13 Tastatur-Break (INT 1B)

Diese Unterbrechung wird aktiviert, sobald die Tasten <Ctrl> und <Break> gedrückt werden. Im BIOS wird der Interruptvektor auf eine IRET-Routine initialisiert. MS-DOS setzt dann diesen Vektor auf die DOS-Control-C-Routine um.

2.14 Timerinterrupt (INT 1C)

Bei jedem Überlauf des Timers 0 wird der INT 8 ausgelöst. Anschließend setzt die Software einen INT 1C (Timer-Folgeinterrupt). Dieser Vektor wird vom BIOS auf eine IRET-Anweisung initialisiert. Es besteht dann die Möglichkeit, den Vektor auf Anwenderprogramme umzulegen, die dann bei jedem Timertick aktiviert werden.

2.15 Bildschirm-Initialisierung (INT 1D)

Hierbei handelt es sich nicht um einen echten Interruptvektor. Vielmehr findet sich hier ein Zeiger, der in eine Tabelle mit den Initialisierungswerten für den Bildschirmcontroller zeigt.

2.16Disk-Parameter (INT 1E)

Der Vektor zeigt auf den aktuellen Parameterblock mit den Laufwerksdaten. Das BIOS setzt ihn auf eine Tabelle im BIOS-ROM. MS-DOS schaltet den Vektor dann auf eine zweite Tabelle im DOS-Bereich um. Zweck dieser Aktion ist die Anpassung der Laufwerksparameter an die eingesetzten Laufwerke. Durch Optimierung der Parameter kann die Zugriffsgeschwindigkeit der Laufwerke erhöht werden. Die Belegung dieser Tabelle ist nachfolgend aufgeführt:

IBM PC	PC 1512	Bedeutung
CF	DF	Steppertime = D, HD Unload = F
02	02	HD Load = 1, Modus = DMA
25	25	Motor Wait
02	02	n * 256 Byte pro Sektor
08	09	Last Sector per Track
2A	2A	GAP length ID - Data
FF	FF	Disk Transfer length
50	50	GAP length for Format
F6	F6	Fill Byte for Format
25	0F	Head settle Time ms
04	02	Motor Start Time (1/8) Sek.

Tabelle 2.15: Aufbau der Disk-Parameter-Tabelle

Die Aufstellung enthält die Werte, die im BIOS des IBM-PC abgelegt sind, und zum Vergleich die Einstellungen des Amstrad PC 1512. Das erste Byte spezifiziert in den obersten 4 Bit den Wert für die Stepperzeit des Motors. Interessant sind auch die Bytes 10 und 11. Während in Byte 11 die Wartezeit nach dem Einschalten des Motors in $^1/_8$-Sekunden-Schritten abgelegt wurde, enthält Byte 10 die Kopf-Beruhigungszeit in Millisekunden. Tabelle 2.15 zeigt bereits deutlich die Verbesserungen der Laufwerke seit Einführung des IBM-PC. Byte 2 enthält die Konstanten für Head Load und DMA Modus, während in Byte 3 die Verzögerungszeit der Motorabschaltung steht. Der Wert in Byte 4 gibt an, wie viele Blöcke à 256 Byte ein Sektor umfaßt. Im MS-DOS-Format sind dies üblicherweise
2 Blöcke, womit sich 512 Byte/Sektor ergeben. Byte 5 spezifiziert die Sektoren pro Spur. Während im IBM-BIOS noch die Zahl 8 steht, um auch 320-Kbyte-Disketten zu bearbeiten, weisen neuere Rechner nur noch 9 Sektoren pro Track (360 Kbyte) auf. In Byte 6 und 8 sind die Lücken (GAPs) spezifiziert, die zwischen den Sektoren (beim Formatieren) und zwischen der Identifikationsmarke und den Daten liegen.

Auch hier wird der Interrupt nicht aktiviert, sondern der Vektor dient lediglich als Zeiger in die Tabelle.

2.17 Grafik-Tabelle (INT 1F)

Auch hier handelt es sich nicht um einen echten Interrupt. Der Vektor dient lediglich als Zeiger in eine Tabelle mit den Grafikzeichen. Das BIOS initialisiert den Vektor mit dem Wert 0:0. Es gibt aber Programme, wie z.B. GRAFTABL.COM, die den Vektor umleiten.

3 Die MS-DOS-Interrupts

Oberhalb des BIOS-Teils (IO.SYS) wird das eigentliche Betriebssystem MS-DOS geladen. Es besteht aus dem Code der Datei MSDOS.SYS, den Datenbereichen für Puffer, den Systemtreibern und dem Kommandointerpreter COMMAND.COM. Ähnlich wie das BIOS stellt MS-DOS bestimmte Systemfunktionen zur Verfügung, die über Interrupts aktiviert werden. Der Bereich von INT 20 bis INT 3F ist bereits ab MS-DOS 2.0 in folgende Bereiche aufgeteilt:

INT	Bemerkungen
20	Interrupts, die durch Anwenderprogramme oder DOS
.	benutzt werden dürfen. Die Interrupts sind durch
27	Microsoft dokumentiert.
28	Interrupts, die von DOS für interne Zwecke be-
.	nutzt werden. Diese Interrupts sind nur teil-
2F	weise durch Microsoft dokumentiert.
30	Interrupts, die für DOS reserviert wurden. Die
.	meisten sind nicht belegt und wurden auch nicht
3F	durch Microsoft dokumentiert.

Tabelle 3.1: Aufteilung der INT 20 bis INT 3F durch DOS

Der Bereich der Interrupts 28-3F ist für MS-DOS-interne Zwecke reserviert und normalerweise für den Anwender nicht zugänglich. Die bisherige Microsoft-Dokumentation enthält nur wenige Informationen über diesen Bereich. Trotzdem sind mittlerweile einige Informationen über die reservierten Interrupts verfügbar. Die nachfolgende Tabelle enthält (soweit bekannt) die Belegung der offiziellen und reservierten DOS-Interruptvektoren.

INT	Adr	Bemerkungen
20	80	DOS Program Terminate
21	84	DOS Function Request
22	88	DOS Terminate Process Exit Address
23	8C	DOS Control-C-Handler Address
24	90	DOS Critical Error Handler Address
25	94	DOS Disk Absolute Read
26	98	DOS Disk Absolute Write
27	9C	DOS Program Terminate and Stay Resident
28	A0	Start a Resident Process
29	A4	Screen Output
2A	A8	reserviert für MS-DOS, aber meist mit einer IRET-Anweisung abgeschlossen
.	.	
2D	B4	
2E	B8	Start a Child Process
2F	BC	Multiplexerinterrupt
30	C0	reserviert für MS-DOS, aber meist mit einer IRET-Anweisung abgeschlossen
.	.	
3F	FF	

Tabelle 3.2: Belegung der MS-DOS-Interrupts

Ein Großteil der Interrupts 20-27 wurde aus Kompatibilitätsgründen zu DOS 1.x erhalten. Mittlerweile stehen über den INT 21 aber wesentlich leistungsfähigere Ersatzfunktionen zur Verfügung, die bei Neuentwicklungen von Software benutzt werden sollten.

Bevor wir allerdings die Schnittstellen dieser Interrupts diskutieren, sollen noch kurz einige Begriffe angerissen werden. Der genauen Beschreibung sind eigene Kapitel gewidmet.

Ein laufendes Programm wird unter MS-DOS als Prozeß bezeichnet. MS-DOS bietet nun die Möglichkeit, aus einem laufenden Prozeß, auch als *parent process* bezeichnet, einen weiteren Subprozeß zu erzeugen. Dieser Subprozeß wird allgemein als *child process* bezeichnet. Während der Ausführung des child process ruht der *parent process* so lange, bis der *child process* beendet wird und die Kontrolle zurückgibt.

Ein weiterer Begriff betrifft das *Program Segment Prefix* (PSP). Hierbei handelt es sich um einen Datenbereich von 256 Byte, der vor jedem geladenen Programm im Speicher steht. Hier legt MS-DOS bestimmte Informationen über die Umgebung des Prozesses ab. Der Aufbau und die Bedeutung dieses PSP werden in einem eigenen Kapitel beschrieben.

3.1 Program Terminate (INT 20)

Mit diesem Interrupt wird ein laufender Prozeß beendet und die Kontrolle geht an den aufrufenden *parent process* zurück. Dieser ist in der Regel der MS-DOS-Kommandoprozessor (COMMAND.COM). Hierdurch wird die DOS-Kommandoebene wieder aktiviert. Der Interrupt besitzt folgende Parameter:

```
Ö---------------------------------Ì
º      CALL:  INT 20              º
º                                 º
º CS: Segmentadresse des Program  º
º     Segment Prefix (PSP)        º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Vor dem Aufruf muß im Codesegment-Register die Segmentadresse des PSP geladen werden. Der INT 20 restauriert aus diesem PSP-Bereich nebenstehende Adressen.

```
Program Terminate Address
Control-C Handler Address
Critical Error Handler Address
```

Bei Programmen mit der Fileextension .COM befindet sich bereits der Wert des PSP im CS-Register, da hier eine Größe von 64 Kbyte nicht überschritten wird.

Vor Aufruf des INT 20 sind alle geöffneten Dateien zu schließen, da die Funktion dies nicht übernimmt. Die Ursache ist in der DOS-Dateiverwaltung begründet. Wird eine Datei verändert, trägt DOS die Informationen über die Dateilänge, das Datum des letzten Schreibzugriffs, sowie Zeit und Lage auf dem Speichermedium erst beim CLOSE-Befehl in die Directories ein. Weiterhin wird sichergestellt, daß vorher alle Pufferinhalte abgespeichert werden. Wird der INT 20 aktiviert, ohne vorher die geöffneten Dateien zu schließen, befinden sich alle modifizierten Dateien in einem inkonsistenten Zustand, wobei die abgespeicherten Daten teilweise verloren gehen können.

Der INT 20 wurde aus Kompatibilitätsgründen zu früheren MS-DOS-Versionen (1.x) übernommen. Bei Neuentwicklungen sollte die Funktion 4CH des INT 21 benutzt werden. Diese benötigt die Information über die Lage des PSP nicht mehr und ermöglicht zusätzlich die Übergabe eines *return codes* an den *parent process*.

3.2 MS-DOS-Funktionsdispatcher (INT 21)

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH: Funktionsnummer             º
º     weitere Register siehe      º
º     Unterfunktion               º
û---------------------------------À
º      RETURN                     º
º siehe Unterfunktion             º
Û---------------------------------ì
```

Dies ist aus Sicht des Anwenderprogrammierers der wohl wichtigste Interrupt. Über INT 21 lassen sich zahlreiche Funktionen aktivieren, wobei die Parameterübergabe per Register erfolgt. Die Funktionen werden über das Register *AH* selektiert.

Wegen der zahlreichen Funktionen erfolgt die Beschreibung des INT 21 in einem eigenen Kapitel.

3.3 Terminate Process Exit Address (INT 22)

MS-DOS legt unter dem Vektor des INT 22 die Adresse der Folgeroutine ab. Sobald der laufende Prozeß terminiert, verzweigt DOS zu der angegebenen Programmadresse. Beim Laden eines neuen Programmes erzeugt MS-DOS das Program Segment Prefix. Ab Offsetadresse 0AH findet sich im PSP auch die unter dem INT 22-Vektor eingetragene Adresse.

Der Interrupt sollte nicht direkt durch Anwenderprogramme verändert werden, da er durch die MS-DOS-EXEC-Funktion belegt ist.

Falls die Notwendigkeit besteht, einen eigenen *Terminate Process Handler* einzubinden, stellt der INT 21 hierfür die Funktionen:

```
Funktion 35H:  lese  den Interruptvektor
Funktion 25H:  setze den Interruptvektor
```

zur Verfügung, um den INT 22 auf die eigene Routine zu setzen. Mit der Funktion 35H wird der eingetragene Vektor gelesen. Dann kann der alte Wert gesichert werden. Anschließend läßt sich der neue Vektor per Funktion 25H setzen. Diese Funktionen garantieren, daß die gelesenen/geschriebenen Vektoren konsistent sind. Ein direkter Zugriff per Anwenderprogramm kann dies nicht sicherstellen.

3.4 Control-C Exit Handler Address (INT 23)

In MS-DOS ist die Möglichkeit vorgesehen, ein laufendes Programm durch die Tastenkombinationen <Ctrl><C> oder <Ctrl><Break> zu unterbrechen. Hierfür sind in MS-DOS zwei verschiedene Module zuständig. Da der Control-C-Interrupt nicht per Hardware erzeugt wird, muß eine Softwareroutine die Tastencodes auf Control-C untersuchen. Wird diese Kombination erkannt, setzt das Programm intern das Control-C-Flag. Dieses Flag wird durch verschiedene DOS-I/O-Funktionen abgefragt. Bei gesetztem Flag lösen diese Funktionen per Software einen INT 23 aus, um den Control-C-Handler zu aktivieren. Durch diese Strategie wird vermieden, daß der Aufruf des Handlers innerhalb einer kritischen Phase, z.B. während eines Diskzugriffs, erfolgt. Das Erkennen einer Control-C-Bedingung und die Aktivierung des Interrupts ist Bestandteil von MS-DOS, während der Interruptservice durchaus per Anwenderprogramm abgewickelt werden kann.

Hierzu wird der Adreßvektor des entsprechenden Interrupt-Handlers in der Interrupttabelle unter dem INT 23 (Adresse: 0000:008CH) eingetragen. Dies sollte allerdings nur unter Verwendung der INT 21-Funktionen 35H (Get-Interruptvektor) und 25H (Set Interrupt Vektor) erfolgen. Nur so kann die Konsistenz der Vektortabelle sichergestellt werden.

Die Abfrage des Control-C-Flags erfolgt softwaremäßig bei jedem Aufruf der DOS-I/O-Funktionen. Die meisten der INT 21-Funktionsaufrufe, bis auf Funktion 06H und 07H, prüfen den Status und lösen einen INT 23 aus. Falls keine I/O-Funktionen ausgeführt werden, oder falls ein Programm »abgestürztInnerhalb des Handlers dürfen beliebige weitere DOS-Funktionen aufgerufen werden, da das System nicht in einem kritischen Bereich unterbrochen wurde. Allerdings sind die Registerinhalte vor Eintritt in die Serviceroutine zu sichern, um sie gegebenenfalls zu restaurieren. Falls Control-C während einer Tastaturabfrage oder Bildschirmausgabe auftritt, oder falls innerhalb des Handlers die BIOS-Funktionen 09H (Tastaturabfrage) und 0FH (Bildschirmausgabe) benutzt werden,

gibt der INT 23 die Codes 03H, 0DH, 0AH an den Bildschirm aus. Auf diesem erscheint die Zeichenkombination:

```
^C
```

und ein Linefeed (Zeilenvorschub).

Der Control-C-Interrupt-Handler kann mit einer IRET-Anweisung abgeschlossen werden. In diesem Fall müssen die Register des Prozessors in den Zustand vor Eintritt in die Serviceroutine, versetzt werden. Anschließend wird das Programm an der unterbrochenen Stelle fortgesetzt. Es kann aber auch auf die Betriebssystemebene per INT 20 zurückgekehrt werden.

Es besteht die Möglichkeit, den DOS-INT 23-Handler durch eine benutzerspezifische Service-Routine zu ersetzen, die dann durch einen RET-FAR-Befehl beendet wird. In diesem Fall wird das Carry-Flag benutzt, um MS-DOS zu signalisieren, ob das Programm abzubrechen ist:

```
Carry-Flag gesetzt  -> Programm abbrechen
Carry-Flag gelöscht -> Programm fortsetzen.
```

Der Wert des INT 23-Vektors wird von MS-DOS beim Erstellen des PSP an die Offsetadresse 0EH kopiert. Falls ein laufender Prozeß einen *child process* erzeugt, wird diesem auch ein eigenes PSP zugewiesen. Falls der *child process* eine eigene Control-C-Routine installiert, wird der INT 23-Vektor verändert. Beim Prozeß-Exit restauriert MS-DOS den Original-Vektor, bevor der *parent process* aktiviert wird, so daß dieser seinen alten Control-C-Handler vorfindet.

Ein Beispiel für die Funktion der Control-C-Routine liefert das DOS-Batch-Management. Wird die Taste <Ctrl><C> betätigt, erscheint die Meldung:

```
^C
Stapeljob beenden (J/N) ?
```

Die Eingabe *J* aktiviert den MS-DOS-Kommandoprozessor, während bei *N* der Stapeljob fortgesetzt wird.

3.5 Critical Error Handler Address (INT 24)

Mit dieser Routine lassen sich »kritischest wird. Der INT 24-Vektor wird beim Erzeugen des Program Segment Prefix in die Offsetadresse 12H kopiert. Der INT 24 (Critical-Error-Handler) ist für die interne Benutzung durch DOS reserviert und sollte deshalb nie durch ein Anwenderprogramm ausgelöst werden. Allerdings besteht die Möglichkeit, den DOS-Critical-Error-Handler durch eine benutzerspezifische Serviceroutine zu ersetzen.

Falls ein I/O-Zugriff innerhalb DOS dreimal mit einem Fehler endet, wird der INT 24 ausgelöst. Beim Eintritt in den Handler wird das Interruptsystem gesperrt. In den Registern AH, SI und BP liefert der DOS-Handler Informationen über die Fehlerart zurück:

```
Ö---------------------------------Ì
º      CALL: INT 24               º
º                                 º
û---------------------------------À
º      RETURN                     º
º AX:   Fehlercode                º
º DI:   erweiterter Fehlercode    º
º BP:SI Zeiger auf den Device     º
º       Header                    º
Û---------------------------------ì
```

Die Register AX und DI enthalten Fehlercodes, die Hinweise auf die Fehlerursache geben. Im Registerpaar BP:SI findet sich ein Zeiger auf den Device Header Block; der gestörten Einheit. Hier lassen sich eventuell weitere Informationen hinsichtlich der Ursache finden.

Ein anwendungsspezifischer INT 24-Handler muß als erstes das Flagregister auf dem Anwenderstack sichern, um dann mit einem FAR CALL den Original DOS-INT 24-Handler aufzurufen. Die Adresse dieses Handlers muß bei der Installation des Anwender-Handlers gesichert werden. Der DOS-INT 24-Handler gibt die Meldung:

```
Abort, Retry, or Ignore
```

aus und wartet auf eine Benutzereingabe. Sobald dies erfolgt ist, geht die Kontrolle wieder an den anwendungsspezifischen INT 24-Handler zurück. Dieser sollte nun die Register auf den Stack sichern. Innerhalb der Routine dürfen nur die DOS-INT-21-Funktionen 01H bis 0CH, sowie 30H und 59H verwendet werden, da nicht bekannt ist, ob sich DOS in einem kritischen Bereich befindet. Die restlichen DOS-Funktionen verändern den System-Stack und hinterlassen einen unkonsistenten Zustand.

Auf dem Anwenderstack finden sich folgende Informationen:

```
Top
IP    -Ì
CS     û-  durch den INT 24 vom Prozessor automatisch gesicherte Register
Flags -ì
AX    -Ì
BX     º
CX     º
DX     º
SI     û-  Register, gesichert durch die INT 21-Funktion, Zustand vor
DI     º   Eintritt in die INT 21-Routine
BP     º
DS     º
ES    -ì
IP    -Ì
CS     û-  durch den INT 21 vom Prozessor automatisch gesicherte Register
Flags -ì
```

Falls nun die Kontrolle direkt durch den Handler an das Anwenderprogramm zurückgegeben werden soll, sind vorher die Register zu restaurieren. Hierbei ist die oben beschriebene Struktur des Stacks zu beachten. Es sind alle Informationen bis auf die drei letzten Einträge:

```
IP    -Ì
CS     û-  durch den INT 21 vom Prozessor automatisch gesicherte Register
Flags -ì
```

vom Stack zu entfernen. Anschließend ist eine IRET-Anweisung auszuführen, womit das Programm hinter der unterbrochenen I/O-Anforderung fortgesetzt wird. Allerdings befindet sich DOS so lange in einem unstabilen Zustand, bis eine INT 21-Funktion oberhalb 0CH aufgerufen wird.

Die zweite Möglichkeit besteht darin, die Kontrolle vom Handler durch eine IRET-Anweisung an DOS zu übergeben. In diesem Fall restauriert MS-DOS die

Prozessorregister vor Eintritt in das Programm. Der Handler kann durch den Wert im AL-Register festlegen, welche Aktionen innerhalb DOS ablaufen:

```
AL  °  Bemerkung
----é-----------------------------------------------------
 0  °  ignoriere den Fehler und setze das Programm fort
 1  °  versuche die Operation zu wiederholen
 2  °  Programmabbruch über den INT 23
 3  °  Systemaufruf abbrechen
```

Tabelle 3.3: Steuercodes für den INT 24

Es ist zu beachten, daß der Befehl *Ignore* zu unvorhersehbaren Resultaten führen kann, da DOS die Operation als erfolgreich ausgeführt betrachtet.

Bei Aufruf des INT 24 wird im Register AX ein Fehlercode übergeben. Ist Bit 7 im AH-Register gelöscht, handelt es sich um einen DISK-Fehler.

Disk-Error-Code

Bei Bit 7 = 0 im AH-Register, resultiert der Fehler aus einem Diskzugriff. Im Register AL findet sich die Nummer des fehlerhaften Laufwerks (0 = A:, 1 = B:, etc.). Die restlichen Bits des AH-Registers sind dann gemäß folgender Darstellung kodiert:

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙÚÙÚÙÚÙÚÙÚÙÚÙÚì  0 Fehler bei Read-Zugriff
 ° ° ° ° ° ÛÚì Û-  1 Fehler bei Write-Zugriff
 ° ° ° ° °  °
 ° ° ° ° °  °     00 Fehler im MS-DOS-Bereich
 ° ° ° ° °  Û---- 01 Fehler in der FAT
 ° ° ° ° °        10 Fehler in der Directory
 ° ° ° ° °        11 Fehler im Datenbereich
 ° ° ° ° °
 ° ° ° ° °        0 Fehler nicht erlaubt
 ° ° ° ° Û------- 1 Fehler zulässig
 ° ° ° °
 ° ° ° °          0 Wiederholung nicht erlaubt
 ° ° ° Û--------- 1 Wiederholung möglich
 ° ° °
 ° ° °            0 Ignore nicht erlaubt
 ° ° Û----------- 1 Ignore zulässig
 ° °
 ° Û------------- ---
 °                0 Fehler in Diskdrive
 Û--------------- 1 Fehler in FAT oder
                    in einem Characterdevice
```

Bild 3.1: Belegung des Disk-Error-Code

Die Bits 3, 4, 5 geben den gültigen Bereich für die Aktionen Retry, Ignore, Abort und Fail an. Ab DOS 3.0 wird der Wert im Register AL überprüft und gegebenenfalls korrigiert. Es gelten folgende Festlegungen:

```
Bit 5 = 0  Ignore nicht erlaubt -> falls AL = 0 ist,
           dann setzt DOS den Wert auf AL = 3 (Fail).
Bit 4 = 0  Retry nicht erlaubt -> falls AL = 1 ist,
           dann setzt DOS den Wert auf AL = 3 (Fail).
Bit 3 = 0  Fail nicht erlaubt -> falls AL = 3 ist,
           dann setzt DOS den Wert auf AL = 2 (Abort)
```

Die Übergabe des Wertes Wert AL = 2 (Abort) an DOS ist immer zulässig.

Device-Error-Code

Falls Bit 7 in AH gesetzt (1) ist, dann liegt entweder ein Fehler in der File-Allocation-Tabelle (FAT) einer Disk vor, oder eine zeichenorientierte Einheit ist gestört. Als Beispiel für die zweite Möglichkeit sei die Ausgabe auf einen Drucker genannt, der kein Papier enthält. Im Registerpaar BP:SI findet sich ein Zeiger auf den Datenblock des Treibers, in dem weitere Informationen über die Einheit gespeichert sind.

Offset	Bemerkung
00	Zeiger auf den Folgetreiber
04	Attributword Bit 15 = 0 Blockdevice 1 = Zeichendevice Bit 14 IOCTL Bit Bit 0 = 1 aktueller Standard Inputdevice Bit 1 = 1 aktueller Standard Outputdev. Bit 2 = 1 aktuelles Nulldevice Bit 3 = 1 aktuelles Clockdevice
06	Zeiger auf die Strategieroutine
08	Zeiger auf Interruptservice Teil
0A	8 Byte Einheiten Name (ASCII)

Tabelle 3.4: Belegung des Device-Headers

Ab Offset 04H findet sich das Attributbyte, welches in Bit 15 angibt, ob es sich um eine zeichen- oder blockorientierte Einheit handelt. Bei blockorientierten Treibern (Disketten) resultiert ein Fehler meist aus einem ungültigen FAT-Abbild; im Speicher.

Zeichenorientierte Treiber setzen Bit 15 = 0. Im Lowbyte des DI-Registers findet sich ein erweiterter Fehlercode.

Code	Bemerkung
00H	Schreibversuch auf eine schreibgeschützte Diskette
01H	unbekannte Einheit
02H	Laufwerk nicht bereit
03H	unbekanntes Kommando
04H	CRC-Datenfehler
05H	ungültige Aufrufstruktur
06H	SEEK-Fehler bei Disketten
07H	unbekanntes Datenmedium
08H	Sektor nicht gefunden
09H	Papierende Drucker
0AH	Schreibfehler
0BH	Lesefehler
0CH	allgemeiner Fehler
0DH	sharing violation (ab DOS 3.3)
0EH	lock violation (ab DOS 3.3)
0FH	invalid disk change
10H	FCB nicht verfügbar (ab DOS 3.3)
11H	sharing Puffer Überlauf (DOS 3.3)

Tabelle 3.5: Erweiterte Fehlercodes bei kritischem DOS-Fehler

Das Highbyte in DI ist nicht definiert. Auch der Inhalt des Registers AL ist bei zeichenorientierten Einheiten nicht definiert. Es besteht die Möglichkeit, durch die DOS-Funktion 58H einen erweiterten DOS-Fehlercode zu lesen.

Allgemein gelten für den INT 24-Handler folgende Bedingungen:

- Alle Register sind innerhalb des Handlers zu sichern.
- Während des Aufrufes ist das Interruptsystem gesperrt.
- Bei Disk-Fehlern wird ab DOS 3.0 der Zugriff 5mal versucht, bis ein INT 24 ausgelöst wird.
- Bei einem Fehler in der FAT wird ein Zugriff 3mal versucht, bis ein INT 24 ausgelöst wird.
- Der INT 24 wird nur bei I/O-Aufrufen durch INT 21-Funktionen aktiviert.
- Die Interrupts 25 (Absolute Read) und 26 (Absolute Write) benutzen eine Fehlerroutine des Kommandoprozessors.
- Der Handler darf nur die INT 21-Funktionen 01H-0CH und 59H benutzen. Alle anderen Funktionen zerstören den Systemstack.
- Falls der Handler die Kontrolle direkt an das Programm zurückgibt, sind vorher die Register und der Stack zu restaurieren. Anschließend ist innerhalb des Anwenderprogramms eine INT 21-Funktion oberhalb von 0CH aufzurufen. Weiterhin ist das Interruptsystem freizugeben.

3.6 Absolute Disk Read (INT 25)

Dieser Interrupt erlaubt es, absolute Sektoren direkt von einer Platte oder Diskette zu lesen. Die Parameterübergabe erfolgt über Register:

```
Ö---------------------------------Ì
º      CALL:  INT 25              º
º                                 º
º AL:    Laufwerksnummer          º
º DS:BX  Disk Transfer Address    º
º CX:    zu lesende Sektoren      º
º DX:    erster Sektor            º
û---------------------------------À
º      RETURN                     º
º CY: 1  Lesefehler               º
º AX:    Fehlerstatus             º
º CY: 0  kein Fehler              º
º ---                             º
Û---------------------------------ì
```

Im Register AL wird das Laufwerk spezifiziert (0 = A:, 1 = B:, etc.). Die Register DS:BX enthalten einen Zeiger auf die Disk Transfer Area; (DTA). Dies ist ein Pufferbereich, in den die gelesenen Daten abgespeichert werden. DOS legt standardmäßig im PSP ab Adresse 80H eine DTA von 128 Byte an. Diese Länge reicht aber nicht aus, um einen Sektor von 512 Byte zu speichern.

Das Anwenderprogramm ist dafür verantwortlich, daß ein genügend großer Pufferbereich vorhanden ist, um alle gelesenen Sektoren (à 512 Byte) aufzunehmen. Der Aufruf liest ab dem Sektor DX die Daten von ein oder mehreren logisch aufeinanderfolgenden Sektoren. Die Zahl der zu lesenden Sektoren steht in CX. Die Anordnung der Sektoren auf einer Diskette ist in einem getrennten Kapitel beschrieben. Die Eingabeparameter werden nicht überprüft, weshalb die Funktion mit Vorsicht zu behandeln ist. Fehler beim Aufruf des INT 25 führen zu falschen Daten oder zum Systemabsturz. Es wird auch nicht der Critical-

Error-Handler aufgerufen, sondern der Interrupt übergibt die Kontrolle sofort an den entsprechenden Einheitentreiber. Dieser ruft im Fehlerfall eine Routine in COMMAND.COM auf.

Alle Register, außer den Segmentregistern, werden beim Aufruf zerstört. Lediglich das Flag-Register wird automatisch beim INT 25 durch den Prozessor auf dem Anwender-Stack gesichert. Nach dem Aufruf enthält es Informationen über den Fehlerstatus:

Tritt während des Ablaufs ein Fehler auf, wird das Carry-Flag gesetzt, und im Register AH findet sich ein Fehlercode mit folgender Bedeutung:

Code	Bemerkung
80	Zugriffsfehler
40	SEEK-Fehler
20	Controller Fehler
10	CRC-Fehler beim Lesen
08	DMA Fehler
04	spezifizierter Sektor nicht gefunden
03	Schreibversuch auf eine geschützte Diskette (INT 26)
02	bad Adress Marke
01	falsches Kommando

Tabelle 3.6: Fehlercodes des INT 25

Die oben aufgeführten Fehlercodes werden im AH-Register übergeben. Im AL-Register finden sich gültige MS-DOS-Fehlercodes, die im Kapitel über die DOS-Fehler beschrieben werden. Die gleichen Codes werden biem INT 24 im Register DI übergeben.

Tritt kein Fehler auf, ist das Carry-Flag gelöscht. Nur in diesem Fall sind die gelesenen Daten gültig. Microsoft empfiehlt daher, diesen Interrupt nur in absolut notwendigen Fällen zu nutzen, insbesondere da im INT 21 mehrere Funktionen zur Dateibehandlung implementiert sind. Es wird auch keine Aufwärtskompatibilität mit neueren MS-DOS-Versionen garantiert.

Ab DOS 4.0 ergeben sich einige Änderungen beim direkten Diskzugriff per INT 25. Bis zur DOS-Version 3.3 wird die Sektoradresse im Register DX übergeben. Der Speicherbereich von Festplatten ist deshalb auf (16-Bit-Sektoradresse) 65536 Sektoren * 512 Byte = 32 Mbyte begrenzt. Ab DOS 4.0 umfaßt die Sektoradresse nun 32 Bit, so daß bis zu 2 Gbyte Plattenspeicher verwaltet werden können. (In DOS 4.X werden zwar nur Platten bis 512 Mbyte unterstützt, aber ab DOS 5.0 liegt die Obergrenze bei den 2 Gbyte). Dies bedingt aber einen etwas geänderten Aufruf des INT 25.

CALL:	INT 25
AL:	Laufwerksnummer
DS:BX	Zeiger auf Parameterblock
CX:	-1
RETURN	
CY: 1	Lesefehler
AX:	Fehlercode
CY: 0	kein Fehler

Im Register AL wird wieder das Laufwerk spezifiziert (0 = A:, 1 = B:, etc.). Da DOS 4.0 auch die Formate der älteren Versionen unterstützt, dient das Register CX zur Selektion der Aufrufkonventionen. Bis DOS 3.3 findet sich in diesem Register die Zahl der zu lesenden

Sektoren. Besitzt das Register vor dem Aufruf den Wert -1 (FFFFH), interpretiert DOS 4.0 dies als einen Befehl zum Zugriff auf Platten mit erweiterten Partitionen größer als 32 Mbyte.

Dann enthält das Registerpaar DS:BX einen 32-Bit-Zeiger auf einen erweiterten Parameterblock. In diesem Parameterblock sind die Informationen für den Zugriff auf das Medium gespeichert. Der Block ist vom rufenden Prozeß anzulegen und besitzt folgende Struktur:

Byte	Bedeutung
00-01	Low Word der ersten zu lesenden Sektornummer
02-03	High Word der ersten zu lesenden Sektornummer
04-05	Zahl der zu lesenden Sektornummern
06-07	Low Word Disk Transfer Address
08-09	High Word Disk Transfer Address

Tabelle 3.7: Aufbau des Parmeterblocks

In den ersten vier Byte findet sich eine 32-Bit-Sektoradresse, ab der gelesen wird. Die Zahl der zu lesenden Sektoren wird dann ab Offset 04H als 16-Bit-Zahl abgelegt. Die Adresse des Disk-Transfer-Buffers findet sich als 32-Bit-Vektor ab Offset 08H. In diesen Puffer legt der INT 25 die gelesenen Daten ab.

Tritt während des Aufrufes ein Fehler auf, ist anschließend das Carry-Flag gesetzt. Im Register AX steht dann ein Fehlercode, der ab DOS 4.0 folgendermaßen zu interpretieren ist:

Der Wert in AH entspricht der Codierung gemäß Tabelle 3.6. Steht in AL ein Wert, ist dieser gemäß folgenden Konventionen zu decodieren:

Code	Fehler
00	Zugriff auf schreibgeschützte Diskette
01	unbekannte Einheit
02	Laufwerk nicht bereit
03	unbekanntes Kommando
04	Datenfehler (CRC-Fehler)
05	fehlerhafte Aufrufstruktur-Länge
06	Seek-Fehler
07	unbekannter Medium-Typ
08	Sektor nicht gefunden
09	(Printer Papierende)
0A	Schreibfehler
0B	Lesefehler
0C	genereller Fehler

Tabelle 3.8: Fehlercodes des INT 25/INT 26 in AL

Wird der INT 25 mit den DOS-3.3-Konventionen aufgerufen, obwohl ein Medium mit Partitionen größer 32 Mbyte vorliegt, findet sich anschließend im Register AX der Fehlercode 0207H.

Es ist wichtig, daß die Zahl der zu lesenden Bytes den Bereich von 64 Kbyte nicht überschreitet, da sonst undefinierte Resultate auftreten. Dies ist bei der Definition der Zahl der zu lesenden Sektoren zu beachten, da ja die Zahl der Byte pro Sektor variieren kann. Weiterhin muß der reservierte Buffer (Disk Transfer Area (DTA)) genügend groß sein, um alle Daten aufzunehmen. Dies ist der Pufferbereich, in den die gelesenen Daten abgespeichert werden. DOS legt standardmäßig im PSP ab Adresse 80H eine DTA von 128 Byte an. Diese Länge reicht aber nicht aus, um einen Sektor von 512 Byte zu speichern. Das Anwenderprogramm ist dafür verantwortlich, daß ein genügend großer Pufferbereich vorhanden ist, um alle gelesenen Sektoren (à 512 Byte) aufzunehmen.

Die Numerierung der logischen Sektoren beginnt mit dem Wert 0 bei Spur 0, Kopf 0, physikalischer Sektor 0. Alle folgenden Sektoren werden dann in aufsteigender Reihenfolge gezählt. Diese Zählweise weicht von der Nomenklatur unter DOS 1.1 ab. Weitere Hinweise finden sich in einem getrennten Kapitel.

Die Eingabeparameter werden nicht überprüft, weshalb die Funktion mit Vorsicht zu behandeln ist. Alle Register, außer dem Segmentregister, werden beim Aufruf zerstört. Lediglich das Flag-Register wird automatisch beim INT 25 durch den Prozessor auf dem Anwenderstack gesichert. Der Treiber gibt ab DOS 4.0 die Kontrolle an den Anwenderprozeß über eine einfache RET-Anweisung zurück. Dies bedeutet, daß der Zustand der Flags vor dem Aufruf auf dem Stack verbleibt. Nach dem Aufruf enthält der Stack deshalb noch einen 16-Bit-Eintrag, der durch eine POP-Anweisung zu entfernen ist.

```
INT 25
JC  ERROR
POP AX     ; Stack bereinigen
  ..
```

Es ist aber zu beachten, daß bei der Stackbereinigung mittels des POP-AX-Befehls der Fehlercode in AX überschrieben wird. Deshalb sollte zuerst das Carry-Flag geprüft werden. Liegt ein Fehler vor, ist zuerst eine Fehlerbehandlung erforderlich. Bei gelöschtem Carry-Flag kann der POP-AX-Befehl direkt ausgeführt werden.

3.7 Absolute Disk Write (INT 26)

Dieser Interrupt erlaubt es, absolute Sektoren direkt auf eine Platte oder Diskette zu schreiben. Die Parameterübergabe erfolgt über Register:

```
Ö---------------------------------Ì
º      CALL:  INT 26              º
º                                 º
º AL:    Laufwerksnummer          º
º DS:BX  Disk Transfer Address    º
º CX:    zu schreibende Sektoren  º
º DX:    erster Sektor            º
û---------------------------------À
º      RETURN                     º
º CY: 1  Schreibfehler            º
º AL:    Fehlercode               º
º CY: 0  kein Fehler              º
º ---                             º
Û---------------------------------ì
```

Im Register AL wird das Laufwerk spezifiziert (0 = A:, 1 = B:, etc.). Die Register DS:BX enthalten einen Zeiger auf die Disk-Transfer-Area (DTA). Dies ist ein Pufferbereich, in den die zu schreibenden Daten abgespeichert sind. DOS legt standardmäßig im PSP ab Adresse 80H eine DTA von 128 Byte an. Diese Länge reicht aber nicht aus, um einen Sektor von 512 Byte zu speichern.

Der Aufruf schreibt ab dem Sektor DX die Daten für ein oder mehrere logisch aufeinanderfolgende Sektoren auf das Speichermedium. Die Zahl der zu schreibenden Sektoren steht in CX. Die Anordnung der Sektoren auf einer Diskette/Platte ist in einem getrennten Kapitel beschrieben. Die Eingabeparameter werden nicht überprüft, weshalb die Funktion mit Vorsicht zu behandeln ist. Fehler beim Aufruf des INT 26 führen zu falschen Daten oder zum Systemabsturz. Es wird auch nicht der Critical Error Handler aufgerufen, sondern der Interrupt übergibt die Kontrolle sofort an den entsprechenden Einheitentreiber. Dieser ruft im Fehlerfall eine Routine in COMMAND.COM auf.

Alle Register, außer dem Segmentregister, werden beim Aufruf zerstört. Lediglich das Flag-Register wird automatisch beim INT 26 durch den Prozessor auf dem Anwenderstack gesichert. Nach dem Aufruf enthält es Informationen über den Fehlerstatus.

Tritt während des Ablaufs ein Fehler auf, wird das Carry-Flag gesetzt und im Register AX findet sich ein Fehlercode mit folgender Bedeutung:

Code	Bemerkung
80	Zugriffsfehler
40	SEEK-Fehler
20	Controller Fehler
10	CRC-Fehler beim Lesen
08	DMA Fehler
04	spezifizierter Sektor nicht gefunden
03	Schreibversuch auf eine geschützte Diskette (INT 26)
02	bad Adress Marke
01	falsches Kommando

Tabelle 3.9: Fehlercodes des INT 26 im Register AH

Die oben aufgeführten Fehlercodes werden im AH-Register übergeben. Im AL-Register finden sich gültige MS-DOS-Fehlercodes, die im Kapitel über die DOS-Fehler beschrieben werden.

Tritt kein Fehler auf, ist das Carry-Flag gelöscht. Nur in diesem Fall wurden die Daten erfolgreich abgespeichert. Microsoft empfiehlt daher, diesen Interrupt nur in absolut notwendigen Fällen zu nutzen, insbesondere da im INT 21 mehrere Funktionen zur Dateibehandlung implementiert sind. Es wird auch keine Aufwärtskompatibilität mit neueren MS-DOS-Versionen garantiert.

Ab DOS 4.0 ergeben sich auch Änderungen beim direkten Schreibzugriff auf Disketten- oder Plattenbereiche. Analog zum INT 25 gelten die folgenden Aufrufkonventionen.

CALL: INT 26	
AL:	Laufwerksnummer
DS:BX	Zeiger auf Parameterblock
CX:	-1
RETURN	
CY: 1	Schreibfehler
AX:	Fehlercode
CY: 0	kein Fehler

Im Register AL wird wieder das Laufwerk spezifiziert (0 = A:, 1 = B:, etc.). Da DOS 4.0 auch die Formate der älteren Versionen unterstützt, dient das Register CX wieder zur Selektion der Aufrufkonventionen. Besitzt das Register vor dem Aufruf den Wert -1 (FFFFH), bezieht sich der Aufruf auf die erweiterten Partitionen von DOS 4.X. Das Registerpaar DS:BX enthält einen 32-Bit-Zeiger auf den erweiterten Parameterblock.

In diesem Parameterblock sind die Informationen für den Zugriff auf das Medium gespeichert. Der Block ist vom rufenden Prozeß anzulegen und besitzt die gleiche Struktur wie beim INT 25.

In den ersten vier Byte findet sich eine 32-Bit-Sektoradresse, ab der geschrieben wird. Die Zahl der zu schreibenden Sektoren wird dann ab Offset 04H als 16-Bit-Zahl abgelegt. Die Adresse des Disk Transfer Buffers findet sich als 32-Bit-Vektor ab Offset 08H. In diesem Puffer müssen die zu schreibenden Daten stehen.

Tritt während des Aufrufes ein Fehler auf, ist anschließend das Carry-Flag gesetzt. Im Register AX steht dann ein Fehlercode, der ab DOS 4.0 folgendermaßen zu interpretieren ist:

Der Wert in AH entspricht der Codierung gemäß Tabelle 3.6. Steht in AL ein Wert, ist dieser gemäß Tabelle 3.8 zu decodieren.

Wird der INT 26 mit den DOS-3.3-Konventionen aufgerufen, obwohl ein Medium mit Partitionen größer 32 Mbyte vorliegt, findet sich anschließend im Register AX der Fehlercode 0207H.

Die Länge des Schreibpuffers darf 64 Kbyte nicht übersteigen. Es ist deshalb erforderlich, die Zahl der zu schreibenden Sektoren auf die Pufferlänge zu justieren, da sonst undefinierte Resultate auftreten.

Die Numerierung der logischen Sektoren beginnt mit dem Wert 0 bei Spur 0, Kopf 0, physikalischer Sektor 0. Alle folgenden Sektoren werden dann in aufsteigender Reihenfolge gezählt.

Die Eingabeparameter werden nicht überprüft, weshalb die Funktion mit Vorsicht zu behandeln ist. Alle Register, außer dem Segmentregister, werden beim Aufruf zerstört. Lediglich das Flag-Register wird automatisch beim INT 26 durch den Prozessor auf dem Anwenderstack gesichert. Der Treiber gibt ab DOS 4.0 die Kontrolle an den Anwenderprozeß über eine einfache RET-Anweisung zurück. Dies bedeutet, daß der Zustand der Flags vor dem Aufruf auf dem Stack verbleibt. Nach dem Aufruf enthält der Stack deshalb noch einen 16-Bit-Eintrag, der durch eine POP-Anweisung zu entfernen ist.

```
INT 26
JC  ERROR
POP AX     ; Stack bereinigen
  ..
```

Es ist aber zu beachten, daß bei der Stackbereinigung mittels des POP-AX-Befehls der Fehlercode in AX überschrieben wird. Deshalb sollte zuerst das Carry-Flag geprüft werden. Liegt ein Fehler vor, ist zuerst eine Fehlerbehandlung erforderlich. Bei gelöschtem Carry-Flag kann der POP-AX-Befehl direkt ausgeführt werden.

3.8 Terminate But Stay Resident (INT 27)

```
Dieser Interrupt erlaubt es, daß ein Prozeß die Kontrolle an den residenten Teil von
COMMAND.COM zurückgibt, ohne daß der zugehörige Programm- und Datenbereich freigegeben
wird. Das heißt, es werden keine anderen Programme in diesen Bereich geladen. Der
Prozeß verbleibt im Speicher und wird lediglich in den Zustand »ruhend
```

Fehler! Verweisquelle konnte nicht gefunden werden.

```
Ö---------------------------------Ì
º      CALL:  INT 27              º
º                                 º
º CS:DX Zeiger auf das Ende des   º
º       belegten Speichers        º
º                                 º
û---------------------------------À
º      RETURN                     º
º ---                             º
º                                 º
Û---------------------------------ì
```

Im Registerpaar CS:DX wird ein Zeiger übergeben, der auf das erste freie Byte hinter dem belegten Speicherbereich zeigt. MS-DOS behandelt dann den Bereich, beginnend bei der Segmentadresse und dem Offset 0, bis zur in CS:DX übergebenen Adresse als erweiterten internen Speicher, in den keine anderen Programme geladen werden können.

Sobald nun der INT 27 aufgerufen wird, terminiert der Programmablauf und die Kontrolle geht an DOS zurück. Alle geöffneten Dateien bleiben erhalten, und der Programmcode

steht auch weiterhin im Speicher. Das Programm kann später z.B. durch einen Interrupt aktiviert werden, wobei die Register CS, DS und ES auf die gleiche Segmentadresse gesetzt werden müssen.

Die Größe des Programmes darf 64 Kbyte nicht überschreiten. EXE-Dateien können nicht mit dem Aufruf beendet werden, da sie einerseits nicht auf 64 Kbyte begrenzt sind und andererseits in den oberen freien Speicherbereich geladen werden.

Der INT 27 restauriert (siehe auch die INT 21-Funktion 30) die folgenden Adreßvektoren:

```
INT 22  Programm Terminate Address
INT 23  Control-C Handler Address
INT 24  Critical Error Handler Address
```

auf den Zustand vor dem Laden des Programmes, welches den INT 27 ausgibt. Aus diesem Grund ist es nicht möglich, über den INT 27 einen der drei oben genannten Handler durch benutzerspezifische Routinen zu ersetzen. Die Handler würden zwar resident im Speicher liegen, aber die Interruptvektoren zeigen auf die Original-DOS-Serviceroutinen.

Der Interrupt läßt sich aber verwenden, um benutzerspezifische residente Anwenderprogramme zu laden.

Da viele dieser Programme nicht unbedingt Informationen über ihre Umgebung benötigen, kann der durch den Environmentblock belegte Speicher an MS-DOS zurückgegeben werden. Hierzu ist die im PSP ab Offset 2CH gespeicherte Segmentadresse des Environments zu lesen und der Speicher durch die INT 21-Funktion 49H (free allocated memory) zurückzugeben. Manche Programme ermöglichen es auch den PSP-Bereich über diese Funktion freizugeben.

Dieser Interrupt wurde aus Kompatibilitätsgründen (MS-DOS 1.x) erhalten. Für EXE-Programme und Neuentwicklungen sollte die Funktion 31H des INT 21 (Keep Process) benutzt werden.

3.9 Start a Resident Process (INT 28 undokumentiert)

Der Interrupt wird intern durch DOS benutzt und die Funktion wurde erst ab DOS 5.0 offiziell durch Microsoft publiziert. Der Interrupt wird bereits ab DOS 2.0 verwendet. Wozu dient nun aber dieser Interrupt?

Die Funktionen von DOS sind nicht reentrant programmiert, was bedeutet, daß innerhalb einer Interrupt-Service-Routine keine DOS-Funktionen oberhalb 0CH aufgerufen werden können. Wenn DOS bestimmte Operationen wie Diskzugriffe ausführt, dann legt es temporäre Werte nicht auf dem Stack, sondern in internen Datenbereichen ab. Wird nun die Funktion vor ihrer Beendigung ein zweites Mal aufgerufen, speichert sie die neuen Zwischenwerte ebenfalls in diesem Speicherbereich. Die Interruptroutine rettet ja nur die Register des Prozessors und nicht die Datenbereiche. Nach Beendigung dieses Aufrufes sind nun für den unterbrochenen Prozeß die Daten zerstört, was im günstigsten Fall zu falschen Ergebnissen führt, meist aber in einem Systemabsturz endet. Da aber Programme wie PRINT.COM Diskzugriffe parallel zu DOS durchführen, muß es Mechanismen geben, diese kritischen Zustände in DOS zu erkennen und abzufangen. Hierzu existiert unter anderem auch der INT 28. Der Aufruf besitzt keine Parameter.

```
Ö-------------------------------Ì
º      CALL:  INT 28            º
º                               º
º ---                           º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Jedesmal, wenn DOS Zeichen über die INT 21-Funktionsaufrufe 01H bis 0CH bearbeitet, liegt ein Zustand vor, in dem eine Benutzung der INT 21-Funktionen 00H und oberhalb 0CH erlaubt ist. Um diesen »sicheren Zustand**Fehler! Verweisquelle konnte nicht gefunden werden.**Ein residenter Prozeß kann nun die eigene Startadresse unter dem INT 28-Vektor ein-tragen. Hierzu sind die INT 21-Funktionen 35H (Get Interrupt Vector) und 25H (Set Inter-rupt Vector) zu nutzen. Jedesmal, wenn DOS einen Funktionsaufruf zur Zeichenbearbeitung (INT 21-Funktion 01H bis 0CH) ausführt, wird dann der residente Prozeß aktiviert. Innerhalb dieses Prozesses dürfen dann die INT 21-Funktionen 00H und oberhalb 0CH benutzt werden.

Die Ursache für diesen INT 28-Aufruf liegt wohl in der Tatsache begründet, daß DOS bei den Funktionen 01H bis 05H und 08H bis 0CH das interne Control-C-Flag abfragt. Lediglich die Funktionen 06H und 07H verzichten darauf. Im Rahmen dieser Abfrage wird der INT 28 ausgelöst.

Nun besteht aber noch die Frage, wann INT 21-Funktionsaufrufe im Bereich 01H bis 0CH aufgerufen werden dürfen. Hier bietet der INT 28 einen weiteren Mechanismus zur Ermittlung des sicheren Zustandes an. Vor dem Aufruf des INT 28 wird ein Flag auf dem Stack gesichert. Dessen Inhalt signalisiert, ob DOS im sicheren Zustand ist, oder ob kritische Operationen noch offenstehen. Das Flag befindet sich 6 Byte oberhalb des Stackpointers, wobei nur das untere Bit definiert ist. Es gilt folgende Kodierung:

```
Flag = 0   Es dürfen keine INT 21-Funktionen im Bereich zwischen
           01H und 0CH genutzt werden.
Flag = 1   DOS befindet sich in einem sicheren Zustand und alle
           INT 21-Funktionsaufrufe lassen sich nutzen.
```

Zusammen mit dem INT 21-Aufruf 34H (Get Critical Region Flag) läßt sich in einem residenten Programm prüfen, ob ein sicherer Zustand vorliegt. Vorsicht ist allerdings bei Verwendung der INT 21-Funktion 51H (Get Active PSP) geboten. Erfolgt ein Aufruf dieser (vor DOS 5.0) undokumentierten Funktion aus einem INT 28-Handler, kommt es zu einem Systemabsturz, da die Funktion einen Fehler enthält, der den Stack nicht korrekt bearbeitet. Ab DOS 3.0 läßt sich ersatzweise die INT 21-Funktion 62H zur Ermittlung des aktiven PSP benutzen. Weitere Hinweise zu dieser Thematik und zum Umgang mit residenten Programmen finden sich in dem entsprechenden Kapitel und in /1/. Der Interrupt wird ab DOS 5.0 durch die INT 2F-Funktion 1680H ersetzt und sollte nicht mehr verwendet werden.

3.10 Screen Output (INT 29 undokumentiert)

Dieser Interrupt wird intern durch DOS benutzt und dessen Funktion wurde bisher nicht offiziell durch Microsoft publiziert. Er wird aber zumindest ab DOS 2.0 unterstützt.

DOS muß natürlich die Ausgabe von Zeichen abwickeln. Hierfür bedient es sich neben den BIOS-Funktionsaufrufen auch des Interrupts 29H. Der Interrupt gibt selbst nur ein

einzelnes Zeichen an der Cursorposition aus. Das Zeichen muß im AL-Register stehen, wobei alle anderen Register vom Interrupt-Handler gesichert werden.

```
Ö-------------------------------Ì
°      CALL:  INT 29            °
°                               °
° AL:  Zeichen                  °
û-------------------------------À
°      RETURN                   °
° ---                           °
°                               °
Û-------------------------------ì
```

Da aber der INT 21 genügend Funktionen zur Zeichenausgabe besitzt, ist der INT 29 normalerweise uninteressant. Es gibt aber DOS-Versionen (z.B. PC-DOS), die auch die Zeichenattribute über den INT 29 beeinflussen können.

Die Serviceroutine verwaltet dann die Daten für das Attributbyte und die Bildschirmseite auf folgenden Adressen:

```
Adresse   Bedeutung
CS:115H   Attributbyte für das auszugebende Zeichen
CS:116H   Bildschirmseite der Ausgabe
```

Die Segmentadresse bezieht sich auf das Codesegment der Serviceroutine. Eine Ausgabe von Attributen kann auf folgende Art erfolgen:

```
Segmentadresse INT 29 ermitteln
Werte an Offset 115H und 116H sichern
gewünschte Einstellung setzen
String zeichenweise ausgeben
alte Werte an Offset 115H restaurieren
```

Die Daten müssen nicht unbedingt gesichert werden, falls die Einstellung für längere Zeit gelten soll. Allerdings können bestimmte DOS-Programme sowie der IBM-Basic-Interpreter die Einstellung zurücksetzen. Es ist aber zu beachten, daß es sich hier um eine undokumentierte Funktion handelt, die nicht in allen DOS-Implementierungen unterstützt wird. Weiterhin können sich die Offsetadressen innerhalb der verschiedenen DOS-Versionen ändern.

In den Versionen DOS 3.2 und 3.3 vergleicht COMMAND.COM die Vektoren des INT 29 und INT 20. Liegt das INT 20-Segment oberhalb des INT 29-Segmentes, nimmt DOS an, daß ANSI.SYS geladen ist.

3.11 Microsoft Netzwerk-Session Layer Interrupt (INT 2A undokumentiert)

Dieser Interrupt wird von der Microsoft Netzwerksoftware benutzt. Es sind mir allerdings nur wenige Einzelheiten über die Aufrufschnittstelle bekannt.

Der Aufruf bietet folgende Funktionen:

```
 AH º Funktion
----é-------------------------------------------------
 00Hº Netzwerk installiert?
    º Return: AH <> 0 falls installiert
----é-------------------------------------------------
 01Hº NETBIOS-Aufruf
----é-------------------------------------------------
 02Hº Setze Printer-Mode im Netz
----é-------------------------------------------------
 03Hº Lese shared device status (check direct I/O)
    º AL = 00H
    º DS:SI Zeiger auf den ASCII Disk Namen
    º Return: CF  0 falls shared device erlaubt
----é-------------------------------------------------
 04Hº NETBIOS-Aufruf
    º AL  =  0 Error retry
    º        1 No retry ----é-------------------------------------------------
    º ES:BX  Zeiger auf den NCB (Network Control Block)
    º Return: AX  0  -> kein Fehler
    º         AH  1  -> Fehler
    º         AL       Fehlercode
----é-------------------------------------------------
 05Hº Lese Netzwerk-Resourcen-Informationen
    º AL      00H
    º Return: AX  reserved
    º         BX  Zahl der Netzwerknamen
    º         CX  Zahl der Kommandos
    º         DX  Zahl der Sessionen
----é-------------------------------------------------
 06Hº NETBIOS-Netzwerk Print Stream Control
    º AL      01H Setze Concatenations Mode
    º         02H Setze Truncations Mode (default)
    º         03H Truncate Print Stream
    º Return: CY = 1
    º         AX = Fehlercode
----é-------------------------------------------------
 80Hº NETBIOS Begin DOS Critical Section
    º AL = 01 bis 06  kritische Sektionsnummer
----é-------------------------------------------------
 81Hº NETBIOS End DOS Critical Section
    º AL = 01 bis 06  kritische Sektionsnummer
----é-------------------------------------------------
 82Hº NETBIOS Server Hook
    º Stack: AX vom INT 21-Aufruf
    º Wird von den INT 21-Funktionen bis auf 0CH
    º und 59H aufgerufen
----é-------------------------------------------------
 84Hº NETBIOS Keyboard Busy Loop
    º Vergleichbar mit dem INT 28
```

Weitere Informationen über die Netzwerkfunktionen finden sich im Kapitel mit der Beschreibung der NetBIOS-Funktionen.

3.12 Start a Child Process (INT 2E undokumentiert)

Bei diesem Interrupt handelt es sich ebenfalls um einen undokumentierten Interrupt, der ab DOS 2.x reserviert ist. Dieser Interrupt bietet eine Hintertür, um interne DOS-Prozeduren wie DIR und DEL über COMMAND.COM, den Kommandointerpreter, zu starten. Der Interrupt wird über die Register DS und SI mit Parametern versorgt.

```
Ö-------------------------------Ì
º      CALL:  INT 2E            º
º                               º
º DS:SI Zeiger auf einen        º
º       Kommandostring          º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Es wird ein Zeiger übergeben, der auf einen Kommandostring mit gültigen DOS-Kommandos zeigt. Dieser String besitzt ein festes Format:

```
Ö------Ú-------------------------------------------------------Ì
º Byte º Bemerkung                                             º
û------é-------------------------------------------------------À
º   1  º Länge des Strings in Byte, ohne d. Abschlußzeichen (n-1)º
û------é-------------------------------------------------------Ä
º   2  º Kommandostring                                         º
º   .  º    .                                                  º
º   n  º    .                                                  º
û------é-------------------------------------------------------À
º n+1  º Carrige Return (0DH) als Abschlußzeichen              º
Û------Ù-------------------------------------------------------ì
```

Tabelle 3.10: Kommandostring; für den INT 2E-Aufruf

Ein gültiger Kommandostring wäre zum Beispiel:

```
03,"DIR",0D
```

Sobald der INT 2E aufgerufen wird, führt der residente Teil von COMMAND.COM die Funktion aus. Dies kann dazu genutzt werden, um aus einem laufenden Programm heraus andere Funktionen aufzurufen.

So läßt sich das *Master Environment* aus einem *child process* leicht mit dem internen DOS-Kommando:

```
0D,"SET ... = ....",0DH
```

beeinflussen. Dies ist ansonsten nur durch direkte Zugriffe in den Speicherbereich des Environments möglich. Allerdings ist die Adresse des Master-Environments nur schwierig zu ermitteln (über die MCB-Kette). Eine Änderung per INT 21-Funktion 4BH (Load and Execute-Programm) mit einem Aufruf des DOS-Kommandointerpreters (COMMAND.COM) über die Anweisung:

```
/C SET ... = ...
```

ist nicht möglich, da dann ja eine zweite Kopie des Kommandoprozessors geladen wird. Diese erhält auch einen eigenen Environmentbereich, auf den sich alle anschließenden SET-Kommandos beziehen. Damit bietet der INT 2E die einzige Möglichkeit, diese Änderung aus einem laufenden Prozeß schnell und elegant vorzunehmen.

Der INT 2E übernimmt keine Speicherverwaltung, so daß nur die »internen INT 2E es erlaubt, einen DOS-Befehl auszuführen, ohne eine zweite Kopie des Kommandointerpreters (COMMAND.COM) zu laden, wie dies bei der INT 21-Funktion 4B00H (Load and Execute-Programm) der Fall ist.

Falls externe Programme aufzurufen sind (z.B. PRINT), muß vorher über die INT 21-Funktion 4AH ein entsprechender Speicherbereich freigegeben werden. Falls das geladene

Programm länger als der freie Speicherbereich ist, werden Teile von DOS überschrieben und es kann zu einem Systemabsturz kommen. Daher sollte der Interrupt nur für die Aktivierung interner DOS-Funktionen benutzt werden.

3.13 Mulitplexerinterrupt (INT 2F undokumentiert)

Dies ist ein intern durch MS-DOS benutzter Interrupt, der in MS-DOS 2.X überhaupt nicht und in den anderen DOS-Versionen nur teilweise offiziell dokumentiert ist. Der Interrupt erlaubt die Kommunikation zwischen zwei laufenden Prozessen. In DOS 2.X muß ein Prozeß vor einer Benutzung prüfen, ob der Vektor bereits initialisiert wurde. Ab DOS 3.0 übernimmt das Betriebssystem die Initialisierung.

Der INT 2F besitzt, ähnlich dem INT 21, einen Funktionsdispatcher, der über den Inhalt des Registers AH die jeweilige Unterfunktion aufruft. Jeder dieser Unterfunktionen wird eine eigene Codenummer zugeordnet. Weitere Parameter lassen sich mittels der restlichen Register übergeben. Leider existieren keine festgelegten Mechanismen zur Vergabe dieser Codenummern, so daß der Softwareentwickler darauf zu achten hat, daß eine Nummer nicht durch zwei Applikationen belegt wird. In diesem Fall muß eine der Applikationen mit einer anderen Codenummer versehen werden. Zur groben Orientierung sei folgende Tabelle angegeben:

Code	Bemerkung
00 - 7F	reserviert für MS-DOS-Zwecke
80 - BF	reserviert für zukünftige Erweiterungen
C0 - FF	frei für Anwenderprogramme

Tabelle 3.11: Zuordnung der Codenummern des INT 2F

Ab DOS 4.0 sind die Codes 00H bis BFH für das Betriebssystem reserviert. Der dokumentierte Teil des Multiplexerinterrupt 2FH wird zur Zeit im wesentlichen zur Kommunikation mit der PRINT-Utility genutzt. Weitere Informationen finden sich im Kapitel über den Multiplexerinterrupt.

3.14 Der INT 30H

Der INT 30 besitzt keinen Vektor, vielmehr legt DOS unter der entsprechenden Adresse in der Interruptvektor-Tabelle einen 5 Byte umfassenden JMP-FAR-xxx:xxx-Befehl ab. Die Adresse zeigt in den DOS-Teil mit den CP/M-orientierten Funktionen. Der CALL FAR CS:0005 zum Beenden eines Prozesses führt zum Beispiel einen FAR JMP auf diese Adresse aus und wird dann zu den entsprechenden Routinen geleitet. Allerdings zeigt der Vektor unter DOS 2.x zwei Byte unter den Einsprungpunkt.

3.15 Der INT 31H

Damit ist der INT 31H ebenfalls nicht mehr benutzbar, da ein Byte des vorhergehenden JMP-FAR-Befehls in den Adreßraum des INT 31H reicht.

Wird in DOS eine Software eingesetzt, die das DOS-Protected-Mode-Interface (DPMI) unterstützt, ist den INT 31 nicht mehr frei. Die DPMI-Spezifikation definiert den INT 31 als Einsprungpunkt für den Dispatcher. Genauere Informationen finden sich im Kapitel über die Speicherverwaltung, wo die DPMI-Spezifikation detailliert beschrieben wird.

3.16Der INT 32H

Dieser Interrupt ist in MS-DOS nicht belegt und reserviert.

3.17 Der Maus-Treiber-Interrupt (INT 33)

Die Interrupts 30H bis 3FH sind für MS-DOS reserviert. Allerdings belegen verschiedene Produkte diese Interrupts. Dies ist auch beim von Microsoft entwickelten Maus-Treiber der Fall. Der MS-MOUSE-Treiber wird meist in CONFIG.SYS oder AUTOEXEC.BAT gestartet. Er installiert sich resident und hängt sich in die Treiberkette ein. Der Interrupt kann dann als erweiterte BIOS- oder DOS-Funktion, zur Abfrage der Maus, gesehen werden.

Nachfolgend soll die Softwareschnittstelle dieses Treibers beschrieben werden.

Maus INIT (AX = 00H)

Mit diesem Aufruf wird der MS-Maus-Treiber initialisiert. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º       CALL:  INT 33             º
º                                 º
º AX: 00H (Maus INIT)             º
û---------------------------------À
º       RETURN                    º
º AX:  0 kein Treiber installiert º
º     FF Treiber vorhanden        º
º BX:  Anzahl der Maustasten      º
Û---------------------------------ì
```

Falls der Treiber nicht vorhanden ist, gibt das Register AX den Wert 00H zurück. Ist er installiert, enthält BX die Zahl der Maustasten. Es gilt dabei folgende Nomenklatur:

```
BX:  FFFFH   2 Maustasten
     0000H   Maus besitzt keine 2 Tasten
     0003H   Mouse System / Logitech Maus
```

Der Wert 00 bedeutet, daß die Maus keine 2 Tasten besitzt.

Mauszeiger einblenden und löschen (AX = 01H, AX = 02H)

Mit diesem Aufruf wird das Mauszeigersymbol auf dem Anzeigeschirm eingeblendet (AX = 01H) oder gelöscht (AX = 02H).

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 01H (Show Cursor)           º
º     02H (Clear Cursor)          º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Es werden sonst keine weiteren Parameter übergeben. Ein mehrfacher Aufruf der Funktion 02H bedeutet, daß auch die Funktion 01H mehrfach aufzurufen ist.

Get Parameter (AX = 03H)

Mit diesem Aufruf lassen sich die Koordinaten des Mauscursors und die Zustände der Tasten abfragen. Es gelten folgende Aufrufkonventionen:

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 03H (Get Parameter)         º
û---------------------------------À
º      RETURN                     º
º CX: X - Koordinate Cursor       º
º DX: Y - Koordinate Cursor       º
º BX: Tastenstatus                º
Û---------------------------------ì
```

In den Registern CX und DX werden die X- und Y-Koordinaten des Mauszeigers mit der Auflösung der jeweiligen Grafikkarte zurückgegeben. Das Register BX enthält die Maske mit dem Status der Maustasten. Es gilt folgende Belegung:

```
Bit 0 :  Linke Taste
Bit 1 :  Rechte Taste
Bit 2 :  Mittlere Taste
```

Ein gesetztes Bit signalisiert, daß die Taste gerade gedrückt wurde. Falls die Maus nur 2 Tasten besitzt, ist Bit 2 undefiniert.

Set Cursor (AX = 04H)

Diese Funktion positioniert den Mauszeiger auf die angegebenen Koordinaten. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 04H (Set Cursor)            º
º CX: X - Koordinate Cursor       º
º DX: Y - Koordinate Cursor       º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Die Cursorkoordinaten beziehen sich auf die Auflösung der verwendeten Grafikkarte.

Get Cursorposition and Pressed Buttons (AX = 05H)

Diese Funktion erlaubt es, die Position des Mauscursors beim letzten Tastendruck abzufragen. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 05H (Get Cursorposition)    º
º BX: 0 -> linke Taste            º
º     1 -> rechte Taste           º
û---------------------------------À
º      RETURN                     º
º CX: X - Position                º
º DX: Y - Position                º
º BX: Anzahl Tastendrucke         º
º AX: Tastenstatus                º
Û---------------------------------ì
```

Die Cursorkoordinaten beziehen sich auf die Auflösung der verwendeten Grafikkarte. Im AX-Register wird der Status der Maustasten zurückgegeben. Es gilt folgende Nomenklatur:

```
Ö-----Ú---------------------------Ì
º Bit º Status                    º
û-----é---------------------------À
º  0  º linke Taste gedrückt      º
û-----é---------------------------À
º  1  º rechte Taste gedrückt     º
û-----é---------------------------À
º  2  º mittlere Taste gedrückt   º
Û-----Ù---------------------------ì
```

Tabelle 3.15: Status der Maustasten

Im Register BX findet sich die Zahl der Tastendrucke seit dem letzten Funktionsaufruf. Die Register CX und DX enthalten die Koordinaten des Mauscursors beim letzten Tastendruck. Eine Maus mit 3 Tasten gibt den Wert AX = 4 zurück, falls die mittlere Taste gedrückt wird.

Get Cursorposition and Free Buttons (AX = 06H)

Diese Funktion arbeitet komplementär zur Funktion 05H, denn sie erlaubt es, Informationen über die freigegebenen Tasten abzufragen.

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 06H (Get Status)            º
º BX: 0 -> linke Taste            º
º     1 -> rechte Taste           º
º     2 -> mittlere Taste         º
û---------------------------------À
º      RETURN                     º
º CX: X - Position                º
º DX: Y - Position                º
º BX: Anzahl Tastenfreigaben      º
º AX: Tastenstatus                º
Û---------------------------------ì
```

Im Register BX ist ein Wert für die abzufragende Taste zu übergeben. Der Code 02H gilt dabei nur für die Mouse System Maus. Die zurückgegebenen Cursorkoordinaten beziehen sich auf die Auflösung der verwendeten Grafikkarte. Im AX-Register wird der Status der Maustasten zurückgegeben. Es gilt folgende Nomenklatur:

Bit	Status
0	linke Taste gedrückt
1	rechte Taste gedrückt
2	mittlere Taste gedrückt

Tabelle 3.16: Kodierung der Maustasten

Im Register BX findet sich die Zahl der Tastenfreigaben seit dem letzten Funktionsaufruf. Die Register CX und DX enthalten die Koordinaten des Mauscursors bei der letzten Tastenfreigabe.

Set X-Koordinates (AX = 07H)

Diese Funktion setzt die minimalen und maximalen Positionen der X-Achse, zwischen denen sich der Cursor bewegen kann.

```
CALL:  INT 33

AX: 07H (Set X - Koordinates)
CX: X - Position Minimum
DX: X - Position Maximum

RETURN

---
```

Die Cursorkoordinaten beziehen sich auf die Auflösung der verwendeten Grafikkarte. Der minimale Wert wird im Register CX übergeben, während das Register DX den Maximalwert enthält.

Set Y-Koordinates (AX = 08H)

Diese Funktion setzt die minimalen und maximalen Positionen der Y-Achse, zwischen denen sich der Cursor bewegen kann.

```
CALL:  INT 33

AX: 08H (Set Y - Koordinates)
CX: Y - Position Minimum
DX: Y - Position Maximum

RETURN

---
```

Die Cursorkoordinaten beziehen sich auf die Auflösung der verwendeten Grafikkarte. Der minimale Wert wird im Register CX übergeben, während das Register DX den Maximalwert enthält.

Define Cursor Symbol (AX = 09H)

Diese Funktion erlaubt es, das Aussehen des Mauscursors zu definieren. Dieser Cursor besteht aus einem Bitfeld zur Darstellung der Form. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 09H (Define Cursor Symbol)  º
º BX: X - Koordinate Mausspitze   º
º CX: Y - Koordinate Mausspitze   º
º ES:DX Zeiger auf Symbolmaske    º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Das Register BX enthält die X-Koordinate des Mauszeigers innerhalb der Symbolmaske. Das Register CX enthält analog die Y-Koordinate des Mauszeigers. Im Registerpaar ES:DX findet sich ein Zeiger auf den Anfang zweier Bitfelder, die die Form des Mauscursors und das Muster des Bildschirmhintergrundes spezifizieren. Das erste Feld von 16 Bit * 16 Byte definiert das Muster des Bildschirmhintergrundes. Jedes gesetzte Bit entspricht dabei einem hellen Punkt. Man beachte, daß Grafikoberflächen häufig schwarze Zeichen auf hellem Hintergrund ausgeben (Inversdarstellung). An dieses Feld schließt sich ein zweites Feld von 16 Bit * 16 Byte an, in denen die Pixel des Maussymbols abgelegt werden. Jedes gesetzte Bit entspricht einem hellen Punkt auf dem Bildschirm. Der Bildschirmausschnitt an der Mausposition wird bei der Ausgabe mit der 16 Bit * 16 Byte Maske des Cursorhintergrundes über die AND-Funktion verknüpft. Das Ergebnis wird anschließend noch mit der Cursormaske über XOR verknüpft. Der Wert FFFFH in der Maske des Bildschirmhintergrundes und gesetzte Bits in der Cursormaske löschen das entsprechende Pixel, so daß der Cursor invertiert dargestellt wird.

Define Textcursor Symbol (AX = 0AH)

Diese Funktion definiert das Aussehen des Textcursors auf dem Bildschirm. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 0AH (Set Textcursor Symbol) º
º BX: 0                           º
º CX: Bildschirmmaske             º
º DX: Cursormaske                 º
º BX: 1                           º
º CX: 1. Cursor Rasterzeile       º
º DX: n. Cursor Tasterzeile       º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Das Register BX steuert die Definition des Textcursors. Falls BX = 0 ist, wird im Register CX die Bildschirmmaske übergeben. Das Register DX enthält die Cursormaske. Im Textmode bestehen die zwei Masken jeweils aus einem 16 Bit Feld mit folgender Kodierung:

Bit	Bedeutung
0-7	ASCII - Zchn. d. Cursors
8-A	Vordergrundfarbe (RGB)
B	Intensität
C-E	Hintergrundfarbe
F	blinkende Darstellung

Tabelle 3.17: Kodierung des Mauscursors; im Textmode

Im Textmode wird vom Cursor jeweils nur ein Zeichenfeld belegt. Der Mode BX = 0 bedeutet, daß der Cursor softwaremäßig erzeugt wird.

Falls BX = 1 ist, wird ein Blockcursor über die Hardware dargestellt. Das Register CX gibt an, in welcher Rasterzeile der Cursor beginnt (s. auch Beschreibung des INT 10). Die letzte Rasterzeile wird in DX spezifiziert.

Read Steps (AX = 0BH)

Diese Funktion ermittelt die Zahl der Mausschritte seit dem letzten Funktionsaufruf. Es gelten folgende Konventionen:

```
CALL:  INT 33

AX: 0BH (Read Steps)

RETURN
CX: X - Schritte
DX: Y - Schritte
```

Die Werte der internen Schrittzähler finden sich in den Registern CX und DX. Ein Schritt entspricht dabei einer Distanz von ca 1/200 Inch und wird als Mickey bezeichnet.

Set Event-Interrupt (AX = 0CH)

Durch diesen Aufruf kann dem Treiber die Adresse einer neuen Serviceroutine übergeben werden. Diese wird bei bestimmten Ereignissen aktiviert.

```
CALL:  INT 33

AX:   0CH (Set New Interrupt)
CX:   Aktivierungsmaske
ES:DX FAR Routine Address

RETURN
---
```

Im Registerpaar ES:DX ist die Adresse der neuen Serviceroutine zu übergeben. Diese wird über einen CALL FAR-Aufruf vom Treiber aktiviert. Das Register CX enthält die Aktivierungsmaske, die definiert, wann die Routine aufzurufen ist. Es gilt folgende Belegung:

Bit	Aktion
0	Änderung Cursorposition
1	linke Taste gedrückt
2	linke Taste freigegeben
3	rechte Taste gedrückt
4	rechte Taste freigegeben
5	mittlere Taste gedrückt
6	mittlere Taste freigeg.

Tabelle 3.18: Belegung der Aktivierungsmaske (Maustreiber)

Die Bits für die mittlere Taste gelten nur bei der Mouse-Systems-Maus. Sobald eine der spezifizierten Bedingungen auftritt, aktiviert der Treiber den definierten Interrupt. Beim Aufruf der neuen Service-Routine sind die Register mit folgenden Werten:

Reg	Wert
AX	Aktivierungsmaske
BX	Zustand Maustaste
CX	x - Koordinate Cursor
DX	y - Koordinate Cursor
SI	horizont. Auflösung
DI	vertikale Auflösung

Tabelle 3.19: Belegung der Register beim Event-Interrupt

belegt. Die Routine kann dann entsprechend auf den Registerinhalt reagieren.

Emulate Lightpen (AX = 0DH, 0EH)

Durch diesen Aufruf kann der Treiber einen Lichtgriffel emulieren.

```
Ö---------------------------------Ì
°      CALL:  INT 33              °
°                                 °
° AX: 0DH (Emulate Lightpen ON)   °
°     0EH (Emulate Lightpen OFF)  °
û---------------------------------À
°      RETURN                     °
° ---                             °
Û---------------------------------ì
```

Mit AH = 0DH wird die Emulation eingeschaltet, während AH = 0EH die Emulation beendet.

Set Step Size (AX = 0FH)

Durch diesen Aufruf kann der Treiber die Maus auf die Standard Schrittweite einstellen. Es gelten folgende Übergabeparameter:

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 0FH (Set Step Size)        º
º CX:   X - Step Size (1 - 7FFFH) º
º DX:   Y - Step Size (1 - 7FFFH) º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Im Registerpaar CX und DX wird die Schrittweite in X- und Y-Richtung in 1/200 Inch pro Step (Mickey) eingestellt. Bei einem X-Wert von 16, bezogen auf 8 Bildpunkte, muß die Maus um 6,4 Inch bei 640 Bildpunkten bewegt werden, um den gesamten Bildschirm in X-Richtung zu überqueren.

Restore Old Screen (AX = 10H)

Soll der Cursor ausgeblendet werden, kann die Funktion 02H benutzt werden. Eine andere Möglichkeit besteht mit dem Aufruf 10H, womit der Treiber den Zustand des Bildschirms ohne das Maussymbol restauriert, die Maus wird also nicht mehr angezeigt. Es gelten folgende Parameter:

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 10H (Restore Old Screen)   º
º CX: obere X - Koordinate       º
º DX: obere Y - Koordinate       º
º SI: untere X - Koordinate      º
º DI: untere Y - Koordinate      º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Im Registerpaar CX und DX findet sich die linke obere Ecke des Fensters, welches restauriert werden soll. Die rechte untere Ecke wird in den Registern SI und DI übergeben. Innerhalb dieses Fensters wird der alte Bildschirminhalt restauriert. Der Mauscursor muß nach dem Aufruf wieder explizit eingeschaltet werden.

PC-Mouse Set Large Graphics Cursor (AX = 12H)

Dieser Aufruf ist bei der Microsoft-Maus nicht implementiert. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 12H (Set large Graphic Cur.)º
º BH: Cursorbreite in Worten     º
º CH: Cursorhöhe in Zeilen       º
º BL: horizontale Mausspitze     º
º CL: vertikale Mausspitze       º
º ES:DX Bit Map Cursor           º
û--------------------------------À
º      RETURN                    º
º AX: -1 ok                      º
Û--------------------------------ì
```

Der Aufruf entspricht der Funktion 09H bei der Microsoft-Maus.

Define Speed (AX = 13H)

Die Geschwindigkeit der Bewegung des Mauszeigers auf dem Bildschirm läßt sich einstellen. Mit diesem Aufruf kann die maximale Geschwindigkeit spezifiziert werden.

```
Ö---------------------------------Ì
°      CALL:  INT 33              °
°                                 °
° AX: 13H (Define Speed)          °
° DX: Geschwindigkeit             °
û---------------------------------À
°      RETURN                     °
° ---                             °
Û---------------------------------ì
```

Die Geschwindigkeit der Bewegung des Mauszeigers wird in Schritten pro Sekunde angegeben und im Register DX übergeben.

Die folgenden Aufrufe des MS-Treibers sind nicht alle dokumentiert, so daß eine Verwendung mit Vorsicht zu handhaben ist.

(MS-Maus) Exchange Interrupt Subroutines (AX = 14H)

Mit diesem Aufruf läßt sich die Adresse einer neuen Interrupt-Subroutine übergeben. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 33              °
°                                 °
° AX: 14H (Exchange Interrupt)    °
° CX: CALL Maske                  °
° ES:DX Adresse neue Routine      °
û---------------------------------À
°      RETURN                     °
° CX: alte CALL Maske             °
° ES:DX Adresse alte Routine      °
Û---------------------------------ì
```

Im Register ES:DX ist die Adresse der neuen Serviceroutine zu übergeben, während das Register CX die Information über die Aufrufmaske (siehe Funktion 0CH) enthält. Nach dem Aufruf enthalten die Register CX und ES:DX die Werte des alten Treibers.

(MS-Mouse) Return Driver Storage Requirements (AX = 15H)

Die Funktion ermittelt die Puffergröße zur Rückgabe des Treiberstatus.

```
Ö---------------------------------Ì
°      CALL:  INT 33              °
°                                 °
° AX: 15H (Storage Requirements)  °
û---------------------------------À
°      RETURN                     °
° BX: Buffer Size                 °
Û---------------------------------ì
```

Im Register BX steht nach dem Aufruf die Puffergröße, die zur Aufnahme des Treiberstatus benötigt wird.

(MS-Mouse) Save Driver State (AX = 16H)

Die Funktion dient zur Abfrage des aktuellen Treiberstatus.

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 16H (Save Driver State)    º
º ES:DX Adresse Puffer           º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Im Register ES:DX ist die Adresse des Puffers zu übergeben, in dem der Maustreiber die Statusinformationen ablegt.

(MS-Mouse) Restore Driver State (AX = 17H)

Die Funktion dient zur Restaurierung des Treiberstatus.

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 17H (Restore Driver State) º
º ES:DX Adresse Puffer           º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Im Register ES:DX ist die Adresse des Puffers zu übergeben, in dem die Statusinformationen abgelegt sind.

(MS-Maus) Set alternate Mouse User Handler (AX = 18H)

Mit diesem Aufruf läßt sich die Adresse einer User-Interrupt-Subroutine übergeben. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 18H (Exchange Interrupt)   º
º CX: CALL Maske                 º
º ES:DX Adresse neue Routine     º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Der Aufruf ist leider von Microsoft nicht dokumentiert. Im Register ES:DX ist die Adresse der neuen Serviceroutine zu übergeben, während das Register CX die Information über die Aufrufmaske enthält. Für die Werte in CX gilt folgende Festlegung:

```
 Bit º Bedeutung
-----é-------------------------------------------
  0  º Aufruf, falls Alt-Taste gedrückt
  1  º Aufruf, falls Ctrl-Taste gedrückt
  2  º Aufruf, falls Shift-Taste gedrückt
  3  º Aufruf, falls rechte Maustaste freigegeben
  4  º Aufruf, falls rechte Maustauste gedrückt
  5  º Aufruf, falls linke Maustaste freigegeben
  6  º Aufruf, falls linke Maustaste gedrückt
  7  º Aufruf, falls Maus sich bewegt
```

Beim Aufruf des Treibers sind folgende Register definiert:

Register	Bedeutung
AX	Bedingungsmaske wie beim Aufruf in CX
BX	Status der Maustasten
CX	Y-Position Cursor
DX	X-Position Cursor
DI	horizontale Auflösung (in Mickeys)
SI	vertikale Auflösung (in Mickeys)

Es sind bis zu drei Handler über diese Funktion installierbar.

(MS-Mouse) Return User alternate Interrupt Vektor (AX = 19H)

Mit diesem Aufruf läßt sich prüfen, ob ein User-Treiber mit dem Aufruf 18H installiert wurde.

CALL: INT 33
AX: 19H (Return Interrupt Adr.)
CX: CALL Maske
RETURN
BX:DX Adresse Routine
CX: CALL Maske

Im Register BX:DX wird die Adresse des User-Treibers zurückgegeben. Ist kein Treiber installiert, enthält CX den Wert 00H. Durch die Maske in CX läßt sich einer der drei Handler selektieren. Stimmt die Maske nicht überein, wird CX = 0 zurückgegeben.

(MS-Mouse) Set Mouse Sensitivity (AX = 1AH)

Mit dem Aufruf läßt sich die Empfindlichkeit und die Auflösung einer MS-Maus einstellen.

CALL: INT 33
AX: 1AH (Mouse Sensitivity)
BX: horizontale Geschwindigk.
CX: vertikale Geschwindigk.
DX: Schranke für 2 x Geschw.
RETURN

In BX und CX wird die Geschwindigkeit angegeben, mit der der Cursor über das Bild bewegt wird. In DX steht eine Schwelle, ab der die Cursorgeschwindig-keit auf den doppelten Wert angehoben wird. Dies wird bei der Funktion 13H wirksam.

(MS-Mouse) Get Mouse Sensitivity (AX = 1BH)

Mit dem Aufruf läßt sich die Empfindlichkeit und die Auflösung einer MS-Maus abfragen.

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 1BH (Mouse Sensitivity)    º
û--------------------------------À
º      RETURN                    º
º BX: horizontale Geschwindigk.  º
º CX: vertikale   Geschwindigk.  º
º DX: Schranke für 2 x Geschw.   º
Û--------------------------------ì
```

In BX und CX wird die Geschwindigkeit angegeben, mit der der Cursor über das Bild bewegt wird (siehe auch Funktion 0FH). In DX steht eine Schwelle, ab der die Cursorgeschwindigkeit auf den doppelten Wert angehoben wird. Dies wird bei der Funktion 13H wirksam.

(MS-Mouse) Set Interrupt Rate (AX = 1CH)

Der Aufruf wird nur bei der *Inport Maus* unterstützt und setzt die Interruptrate des Treibers.

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 1CH (Set Interrupt Rate)   º
º BX: Interrupt Rate             º
û--------------------------------À
º      RETURN                    º
Û--------------------------------ì
```

Für den in BX übergebenen Wert gilt:

```
BX = 00H  keine Interrupt erlaubt
     01H   30 Interrupts / Sekunde
     02H   50 Interrupts / Sekunde
     03H  100 Interrupts / Sekunde
     04H  200 Interrupts / Sekunde
```

Werte größer als BX = 4 verursachen undefinierte Ergebnisse.

(MS-Mouse) Define Display Page Number (AX = 1DH)

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 1DH (define display page)  º
º BX: Display Page Nummer        º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Der Aufruf selektiert, auf welcher Bildschirmseite der Cursor ausgegeben wird.

(MS-Mouse) Return Display Page Number (AX = 1EH)

```
Ö---------------------------------Ì
°      CALL:  INT 33              °
°                                 °
° AX: 1EH (select display page)   °
û---------------------------------À
°      RETURN                     °
° BX: Display Page Nummer         °
Û---------------------------------ì
```

Der Aufruf prüft, auf welcher Bildschirmseite der Cursor ausgegeben wird.

(MS-Mouse) Disable Mouse Driver (AX = 1FH)

Der Aufruf blockiert den Maustreiber.

```
Ö---------------------------------Ì
°      CALL:  INT 33              °
°                                 °
° AX: 1FH (diable driver)         °
û---------------------------------À
°      RETURN                     °
° AX: FFFFH Fehler                °
°     1F00H ok                    °
° ES:BX INT 33 Vektor             °
Û---------------------------------ì
```

Läßt sich der Treiber abschalten, enthält das Register AX den Wert vor dem Aufruf. Sonst wird der Wert FFFFH zurückgegeben. In ES:BX steht der Vektor, der vor der Installation des Treibers in der Interruptvektor-Tabelle geladen war. Der Aufruf restauriert die Vektoren des INT 10H und des INT 71H bei 8086-Systemen. Bei 80286/80386-Systemen wird der INT 74H restauriert. Das Anwenderprogramm kann anschließend den in ES:BX zurückgegebenen Wert unter dem INT 33 eintragen. Dann ist der Treiber komplett abgeschaltet.

(MS-Mouse) Enable Mouse Driver (AX = 20H)

Der Aufruf gibt den Maustreiber wieder frei.

```
Ö---------------------------------Ì
°      CALL:  INT 33              °
°                                 °
° AX: 20H (enable driver)         °
û---------------------------------À
°      RETURN                     °
° ---                             °
Û---------------------------------ì
```

Der Aufruf gibt die mit der Funktion 1FH restaurierten Vektoren wieder frei.

(MS-Mouse) Software Reset (AX = 21H)

Mit dieser Funktion läßt sich der Maustreiber zurücksetzen.

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 21H (Software Reset)        º
û---------------------------------À
º      RETURN                     º
º AX = Status                     º
Û---------------------------------ì
```

Im Register AX gibt der Treiber einen Statuscode zurück. Wurde das Register beim Aufruf nicht verändert, ist der Treiber nicht installiert. Andernfalls findet sich in AX der Wert FFFFH. Die Funktion ist mit dem Aufruf AH = 00H identisch, mit der Einschränkung, daß der Mausstatus nicht zurückgelesen wird.

(MS-Mouse) Set Language for Messages (AX = 22H)

Mit dieser Funktion läßt sich die Sprache für die Ausgabemeldungen selektieren. Im Register BX ist hierzu ein entsprechender Landescode zu übergeben.

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 22H (Set Language)          º
º BX: Languagecode)               º
û---------------------------------À
º      RETURN                     º
Û---------------------------------ì
```

Für den Landescode gilt folgende Belegung:

```
 BX = 00H  englisch
      01H  französich
      02H  holländisch
      03H  deutsch
      04H  schwedisch
      05H  finnisch
      06H  spanisch
      07H  portugiesisch
      08H  italienisch
```

Diese Funktion ist nur bei internationalen Versionen des Microsoft Maustreibers verfügbar.

(MS-Mouse) Get Language for Messages (AX = 23H)

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 23H (Get Language)          º
û---------------------------------À
º      RETURN                     º
º BX: Languagecode)               º
Û---------------------------------ì
```

Mit dieser Funktion läßt sich die Sprache für die Ausgabemeldungen abfragen. Im Register BX wird ein entsprechender Landescode zurückgegeben. Es gilt dabei die Kodierung gemäß Funktion 22H. Die US-Version gibt hier immer den Wert 0 zurück.

(MS-Mouse) Get Software Version and Mouse Typ (AX = 24H)

Mit dieser Funktion läßt sich die Mausversion und der -typ abfragen.

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 24H (Software Reset)        º
û---------------------------------À
º      RETURN                     º
º AX = Status                     º
º BH = Hauptversion               º
º BL = Unterversion               º
º CH = Maustyp                    º
º CL = Interrupt                  º
Û---------------------------------ì
```

Im Register AX gibt der Treiber einen Statuscode zurück. Enthält das Register den Wert FFFFH, liegt ein Fehler vor und die restlichen Register sind undefiniert. Alle anderen Werte deuten auf einen erfolgreichen Aufruf hin. Im Register BH steht dann die Hauptversionsnummer des Maustreibers, während BL die Unterversion enthält. In CH findet sich der Maustyp (1 = Bus, 2 = serial, 3 = InPort, 4 = PS/2, 5 = HP). Der Interrupttyp wird in CL kodiert (0 = PS/2, 2 = IRQ2, 3 = IRQ3,...., 7 = IRQ7).

(MS-Mouse) Set Acceleration Profile (AX = 2CH)

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 2CH (Set Profile)           º
º ES:DX Filename                  º
û---------------------------------À
º      RETURN                     º
Û---------------------------------ì
```

Diese Version setzt ein Profil für die Beschleunigung der Maus. Die Datei mit dem Maustreiber MOUSEPRO.FIL enthält ein Beispiel für ein solches Profile. Der Name der Datei mit den Profildefinitionen wird in ES:DX übergeben.

(MS-Mouse) Select Acceleration Profile (AX = 2DH)

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 2DH (Select Profile)        º
º BX = Level                      º
û---------------------------------À
º      RETURN                     º
Û---------------------------------ì
```

Dieser Aufruf selektiert ein Profil aus der Definitionsmenge. Der Wert für die Beschleunigung ist über das Register BX beim Aufruf zu übergeben. Die Werte dürfen zwischen 01H und 04H liegen.

(PC-Mouse) Get MS-Mouse storage Requirements (AX = 42H)

Dies ist ein erweiterter Aufruf des PC-Maustreibers. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 42H (Get storage Requirem.) º
û--------------------------------À
º      RETURN                    º
º AX = Status                    º
º BX = Buffer Size               º
Û--------------------------------ì
```

Das Register AX enthält nach dem Aufruf einen Statuscode. Mit dem Wert 4200H steht fest, daß die erweiterten Funktionsaufrufe mit den Codes 42H, 50H und 52H nicht unterstützt werden. Der Wert 0000H signalisiert, daß es sich um keine MS-Maus handelt. Nur falls der Wert FFFFH zurückgegeben wird, enthält das Register BX die benötigte Puffergröße in Byte für die Aufrufe 50H und 52H.

(MS-Mouse/Logitech) Return Copyright String (AX = 4DH)

Mit dieser Funktion läßt sich die Copyright Meldung abfragen.

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 4DH (Return Copyright)     º
û--------------------------------À
º      RETURN                    º
º ES:DI: Stringadresse           º
Û--------------------------------ì
```

Mit dieser Funktion läßt sich eine Copyright Nachricht abfragen. Nach dem Aufruf enthält ES:DI einen Zeiger auf den entsprechenden String *Copyright 1983 Microsoft ***.* Dieser String wird auch von anderen Treibern zurückgeliefert.

(PC-Mouse) Save MS-Mouse State (AX = 50H)

Mit dem Aufruf läßt sich der Status des Treibers in einem Puffer sichern. Es gilt folgende Aufrufschnittstelle.

```
Ö--------------------------------Ì
º      CALL:  INT 33             º
º                                º
º AX: 50H (Save Status)          º
º BX: Puffergröße in Byte        º
º ES:DX Adresse Puffer           º
û--------------------------------À
º      RETURN                    º
º AX = FFFFH ok                  º
Û--------------------------------ì
```

In BX ist die mit dem Aufruf 42H ermittelte Puffergröße in Byte zu übergeben. Der Puffer ist im Anwenderprozeß einzurichten. Die Adresse wird vor dem Aufruf in ES:DX gespeichert. Nach einem erfolgreichen Aufruf enthält AX den Wert FFFFH.

(PC-Mouse) Restore MS-Mouse State (AX = 52H)

Mit dem Aufruf läßt sich der Status des Treibers aus dem Puffer restaurieren. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 52H (Restore Status)        º
º BX: Puffergröße in Byte         º
º ES:DX Adresse Puffer            º
û---------------------------------À
º      RETURN                     º
º AX = FFFFH ok                   º
Û---------------------------------ì
```

In BX ist die mit dem Aufruf 42H ermittelte Puffergröße in Byte zu übergeben. Der Puffer enthält die mit dem Aufruf 50H gelesenen Statusinformationen. Die Adresse wird vor dem Aufruf in ES:DX gespeichert. Nach einem erfolgreichen Aufruf enthält AX den Wert FFFFH.

(MS-Mouse/Logitech) Get Version String (AX = 6DH)

Mit dieser Funktion läßt sich die Copyright Meldung abfragen.

```
Ö---------------------------------Ì
º      CALL:  INT 33              º
º                                 º
º AX: 6DH (Return Copyright)      º
û---------------------------------À
º      RETURN                     º
º ES:DI: Stringadresse            º
Û---------------------------------ì
```

Mit dieser Funktion läßt sich eine Copyright Nachricht bei einem Logitech Treiber abfragen.

Bei Verwendung obiger Mausaufrufe ist zu prüfen, ob der aktuelle Treiber diese Funktionen unterstützt, da kleinere Abweichungen zwischen den einzelnen Treibern auftreten können.

Bei Logitech-Treibern sind zusätzlich die Funktionsaufrufe AX=1DXXH, AX=20XXH bis AX=24XXH implementiert. Auf die Beschreibung wurde jedoch verzichtet, da die MS-Aufrufe einen Herstellerstandard darstellen.

3.18 INT 34 bis INT 3E

Diese Interrupts bilden das Interface für die Borland/Microsoft-Floation-Point-Emulationsaufrufe. Dabei gilt folgende Abbildung der Interrupts:

INT	Floating Point Code
34H	Opcode D8H
35H	Opcode D9H
36H	Opcode DAH
37H	Opcode DBH
38H	Opcode DCH
39H	Opcode DDH
3AH	Opcode DEH
3BH	Opcode DFH
3CH	Emulation mit Override Anw.
3DH	FWAIT
3EH	shortcut calls

Der Interrupt 3C bildet die Schnittstelle für Floating Point Aufrufe, die ein Segment Override benutzen. Es wird die Codefolge CD 3C .... generiert. Dabei lassen sich die 4 Segmentregister in der Codefolge angeben.

Der INT 3E wird für Aufrufe innerhalb der Librarys benutzt.Die genaue Aufrufschnittstelle ist aber nicht dokumentiert.

3.19 Overlay Manager (INT 3FH)

Dieser Interrupt wird vom Microsoft LINK.EXE zur Verwaltung von Overlays benutzt. Borlands Turbo Linker belegt diesen Interrupt ebenfalls. Weiterhin nutzt der Microsoft-Dynamik-Link-Library-Manager den Interrupt. Die genaue Aufrufschnittstelle ist allerdings nicht bekannt.

3.20 Disktreiber (INT 40)

Falls im System eine Festplatte vorhanden ist, wird der BIOS-INT 13-Vektor zur Diskettensteuerung auf den INT 40 umgeschaltet. Dies erfolgt beim Systemstart durch das BIOS des Festplattentreibers. Alle Aufrufe zur Diskettensteuerung werden aber vom Anwenderprozeß über den INT 13 abgewickelt. Dieser Treiber unterscheidet dann zwischen Platte/Diskette und löst gegebenenfalls einen INT 40 aus, um die Diskettenfunktionen zu aktivieren. Weitere Informationen finden sich bei der INT 13-Beschreibung der BIOS-Funktionen.

3.21 Parametertabelle Festplatte (INT 41, INT 46)

Hier handelt es sich eigentlich nicht um einen Interruptvektor, vielmehr wird analog dem INT 1EH (Diskette Parameters) von der BIOS-Funktion ein Adreßvektor der Disk-Parameter-Tabelle eingetragen (INT 41 = Drive 0, INT 46 = Drive 1). Die Tabelle besitzt folgende Struktur:

Offset	Bytes	Feld
00	2	maximale Zylinderzahl
02	1	maximale Kopfzahl
03	2	---
05	2	Startzylinder zum Schreiben
07	1	maximale Datenlänge für ECC
08	1	Kontroll Byte: Bit 7 disable access retry Bit 6 disable ECC retry Bit 5 set defect map = zylinder+1 Bit 4 = 0 Bit 3 = 1 -> mehr als 8 Köpfe Bit 0-2 = Drive Option (XT)
09	1	Timeout (XT, sonst 0)
0A	1	Format Timeout (XT, sonst 0)
0B	1	Drive Check Timeout (XT, sonst 0)
0C	2	Landezone für Köpfe
0E	1	Sektoren pro Spur
0F	1	0

Tabelle 3.19: Aufbau der Disk-Parameter-Tabelle; bei Platten

Sollen die Werte verändert werden, ist eine zweite Tabelle anzulegen. Die Anfangsadresse dieser Tabelle kann dann mit der INT 21-Funktion 25H (Set Interrupt Vector) umgesetzt werden.

3.22 EGA-Videostatus (INT 42)

Dieser Vektor wird durch das BIOS einiger EGA-Karten belegt. Der Vektor zeigt auf eine 64 Byte große Tabelle mit den Funktions- und Videostatusinformationen. Der Inhalt dieser Tabelle läßt sich aber auch über die INT 10-Funktion 1BH lesen. Eine Beschreibung der Tabelle findet sich in Kapitel 13 (EGA-BIOS-Erweiterung).

3.23 EGA-Grafikzeichensatz (INT 43)

Unter diesem Vektor legt das BIOS einiger EGA-Karten einen 32-Bit-Vektor auf die Tabelle mit den ersten 128 Zeichen des grafischen Zeichensatzes ab. Dieser Satz wird in den Modi 04H, 05H und 06H des EGA-Adapters benutzt. Bei anderen Grafikmodi enthält die Tabelle alle 256 Zeichen.

3.24 EGA-Interrupt (INT 44)

Dieser Vektor wird durch den Charactergenerator der EGA-Karten belegt. Die Bedeutung des Vektors ist allerdings nicht eindeutig klar.

Die Novell NetWare belegt diesen Interrupt ebenfalls für das Hochsprachen API.

3.25 Timer-Alarm (INT 4A, INT 50)

Das BIOS der 80286 Prozessoren erlaubt meist die Funktion der Real-Time-Clock für Alarmfunktionen zu nutzen. Der INT 50 wird vom Timer mit ca. 1024 Hz angesprochen (externer Interrupteingang). Mittels der Funktion AH = 06 des Interrupts 1AH läßt sich der INT 50 so einstellen, daß zu einer bestimmten Uhrzeit ein Alarm ausgelöst wird. Der Anwenderprozeß muß vorher eine eigene Alarmbehandlungsroutine unter dem Vektor des INT 4A installieren, welche diesen Alarm dann auswertet. Weitere Informationen finden sich bei der Beschreibung des INT 1A. Diese Funktion ist jedoch nicht bei allen BIOS-Versionen implementiert.

3.26 VDS Virtual DMA Specification (INT 4BH)

Diese Spezifikation eröffnet einen Weg um im *Protected Mode* mit virtueller Speicherverwaltung einen DMA-Transfer durchzuführen. Innerhalb der Spezifikation wird der INT 4B mit folgenden Funktionsaufrufen belegt.

VDS Get Version (AX=8102H)

Mit diesem Aufruf läßt sich die aktuelle Version ermitteln. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 8102H                       º
º DX: 0000H                       º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
º AH = Hauptversion               º
º AL = Unterversion               º
º BX = Produkt Nummer             º
º CX = Revisons Nummer            º
º DX = Flags                      º
º SI:DI Puffer Größe              º
Û---------------------------------ì
```

Beim Aufruf ist das Registerpaar DX mit dem Wert 0000H zu belegen. Das Carry-Flag spezifiziert, ob der Aufruf erfolgreich war. Bei gesetztem Flag enthält AL einen Fehlercode mit folgender Codierung:

```
AL  ° Fehlerursache
-----------------------------------------------------
01H ° region not in contigous memory
02H ° region crossed a physical alignment boundary
03H ° unable to lock pages
04H ° no buffer available
05H ° region too large for buffer
06H ° buffer currently in use
07H ° invalid memory region
08H ° region was not locked
09H ° number of physical pages greater as table length
0AH ° invalid buffer ID
0BH ° copy out of buffer range
0CH ° invalid DMA channel number
0DH ° disable count overflow
0EH ° disable count underflow
0FH ° function not supported
10H ° reserved flag bits set in DX
```

Ist das Carry-Flag nach dem Aufruf gelöscht, enthalten die Register AX, BX und CX Informationen über die Versionsnummer und den Revisionsstand. In DX werden verschiedene Flags zurückgegeben. Diese besitzen folgende Kodierung:

```
DX Flagcodierung

 Bit 0: PC/XT Bus (1 Mbyte Adressraum)
     1: physical buffer/remap region in 1. Mbyte
     2: automatic remap enabled
     3: all memory physically contigous
  4-15: reserviert
```

Das Registerpaar SI:DI enthält die maximale Größe des DMA-Puffers.

Bei verschiedenen Subfunktionen werden Descriptor-Tabellen zur Beschreibung der DMA-Parameter verwendet. Diese Tabellen besitzen folgenden Aufbau:

```
Format der DMA descriptor struktur (DDS)

Offset Bytes Bedeutung
 00H     4   Größe der Region
 04H     4   Offset
 08H     2   Segment / Selektor
 0AH     2   Puffer ID
 0CH     4   physikalische Adresse
```

Einige Aufrufe nutzen eine erweiterte DMA-Descriptor-Struktur (EDDS).

```
Format der Extended DMA descriptor struktur (EDDS)

Offset Bytes Bedeutung
 00H     4   Größe der Region
 04H     4   Offset
 08H     2   Segment / Selektor
 0AH     2   reserviert
 0CH     2   verfügbare Nummer
 0EH     2   benutzte Nummer
 10H     4   Region 0 physikalische Adresse
 14H     4   Region 0 Größe in Byte
 18H     4   Region 1 physikalische Adresse
 1CH     4   Region 1 Größe in Byte
  ......
```

Falls in der Extended-DMA-Descriptor-Struktur Einträge in der *Page Table* vorliegen, gilt folgende Struktur:

```
Format der Extended DMA descriptor struktur (EDDS) mit page table entries.

Offset Bytes Bedeutung
 00H     4   Größe der Region
 04H     4   Offset
 08H     2   Segment / Selektor
 0AH     2   reserviert
 0CH     2   verfügbare Nummer
 0EH     2   benutzte Nummer
 10H     4   Page Table Entry 0 (386 page table)
 14H     4   Page Table Entry 1
   ......
```

Die Bits 1-12 eines *Page Table Entries* sollten zu 0 gesetzt werden. Bit 0 ist zu setzen, falls die Seite vorhanden und im Status *locked* ist.

VDS Lock DMA Region (AX=8103H)

Mit diesem Aufruf läßt sich eine Region im Speicher für einen DMA-Transfer fixieren.

```
Ö---------------------------------Ì
°      CALL:  INT 4B              °
°                                 °
° AX: 8103H                       °
° DX: Flags                       °
° ES:SI DMA Descriptor            °
û---------------------------------À
°      RETURN                     °
° CF = 1 Error                    °
° AL = Fehlercode                 °
° CF = 0 ok                       °
Û---------------------------------ì
```

Beim Aufruf wird das Register DX als Flag verwendet, welches die Aktion steuert. Für die einzelnen Bits gilt folgende Belegung:

```
Codierung des Flagregisters DX beim Aufruf

Bit 0: reserviert (0)
    1: Daten in Puffer kopieren
       (wird ignoriert falls Bit 2 = 1)
    2: Puffer nicht allocieren, falls Region
       nicht zusammenhängend ist, oder die
       pyhsikalischen Grenzen (Bit 4,5) über-
       schreitet
    3: kein Versuch eines automatischen Remaps
    4: Region darf 64 Kbyte nicht überschreiten
    5: Region darf 128 Kbyte nicht überschreiten
 6-15: reserviert (0)
```

In ES:SI wird die Adresse auf die DMA-Descriptor-Struktur (DDS) übergeben. Der Aufbau dieser Struktur ist beim Aufruf AX=8102H beschrieben. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der Kodierung gemäß AX=8102H. Das *region size field* in der DDS ist mit der maximalen zusammenhängenden Speichergröße gefüllt. Bei gelöschtem Carry-Flag ist der Bereich gesperrt. Dann kann die physiklische Seite nicht ausgelagert werden. In der DDS sind das »physikalische Adress-FeldVDS Unlock DMA Region (AX=8104H)

Mit diesem Aufruf läßt sich eine Region im Speicher nach einen DMA-Transfer freigeben.

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 8104H                       º
º DX: Flags                       º
º ES:DI DMA Descriptor            º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
Û---------------------------------ì
```

Beim Aufruf wird das Register DX als Flag verwendet, welches die Aktion steuert. Für die einzelnen Bits gilt folgende Belegung:

```
Codierung des Flagregisters DX beim Aufruf

Bit 0: reserviert (0)
    1: Daten aus Puffer kopieren
 2-15: reserviert (0)
```

In ES:SI wird die Adresse auf die DMA-Descriptor-Struktur (DDS) übergeben. Der Aufbau dieser Struktur ist beim Aufruf AX=8102H beschrieben. Die Felder *region size*, *physical address* und *buffer id* sind zu setzen. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der Kodierung gemäß AX=8102H. Das *region size field* in der DDS ist mit der Größe des maximal zusammenhängenden Speicherblocks gefüllt. Bei gelöschtem Carry-Flag ist der Bereich freigegeben. In der DDS ist das *physical adress field* und das *buffer id field* belegt. (ein Wert von 0 im *buffer id field* signalisiert, daß kein Puffer allociert war).

VDS Scatter/Gather Lock Region (AX=8105H)

Dieser Aufruf erlaubt es, Teile des Speichers zu sperren. Dies nützlich, falls Teile des virtuellen Speichers ausgelagert wurden.

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 8105H                       º
º DX: Flags                       º
º ES:DI DMA Descriptor            º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
Û---------------------------------ì
```

Beim Aufruf wird das Register DX als Flag verwendet, welches die Aktion steuert. Für die einzelnen Bits gilt folgende Belegung:

```
Codierung des Flagregisters DX beim Aufruf

Bit 0-5: reserviert (0)
      6: EDDS mit page table entries zurückgeben
      7: nur vorhandene Pages sperren (lock),
         nicht vorhandene Seiten mit 0000 be-
         legen.
   8-15: reserviert (0)
```

In ES:SI wird die Adresse auf die erweiterte DMA-Descriptor-Struktur (EDDS) übergeben. Dabei sind die Felder *region size*, *linear segment*, *linear offset* und *number available* zu belegen. Der Aufbau dieser Struktur ist beim Aufruf AX=8102H beschrieben. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der

Kodierung gemäß AX=8102H. Das *region size field* in der EDDS ist mit der Größe des maximal zusammenhängenden Speicherblocks gefüllt. Bei gelöschtem Carry-Flag wurde der gewünschte Bereich gesperrt. In der EDDS ist die Zahl der belegten Seiten im Feld *number used* gespeichert. Ist Bit 6 gesetzt, enthalten die untersten 12 Bit des Registers BX den Offset auf die erste Seite (first page).

VDS Scatter/Gather Unlock Region (AX=8106H)

Dieser Aufruf erlaubt es, gesperrte Teile des Speichers wieder freizugeben.

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 8106H                       º
º DX: Flags                       º
º ES:DI DMA Descriptor            º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
Û---------------------------------ì
```

Beim Aufruf wird das Register DX als Flag verwendet, welches die Aktion steuert. Für die einzelnen Bits gilt folgende Belegung:

```
Codierung des Flagregisters DX beim Aufruf

Bit 0-5: reserviert (0)
      6: EDDS enthält page table entries
      7: EDDS kann nicht vorhandene Seiten
         enthalten.
   8-15: reserviert (0)
```

In ES:SI wird die Adresse auf die erweiterte DMA-Descriptor-Struktur (EDDS) übergeben. Diese Struktur wurde beim Aufruf AX=8105H initialisiert. Der Aufbau dieser Struktur ist beim Aufruf AX=8102H beschrieben. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der Kodierung gemäß AX=8102H.

VDS Request DMA Buffer (AX=8107H)

Dieser Aufruf erlaubt es, einen DMA-Puffer anzulegen.

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 8107H                       º
º DX: Flags                       º
º ES:DI DMA Descriptor            º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
Û---------------------------------ì
```

Beim Aufruf wird das Register DX als Flag verwendet, welches die Aktion steuert. Für die einzelnen Bits gilt folgende Belegung:

```
Codierung des Flagregisters DX beim Aufruf

Bit 0:  reserviert (0)
    1:  Daten in Puffer kopieren
 2-15:  reserviert (0)
```

In ES:SI wird die Adresse auf die DMA-Descriptor-Struktur (DDS) übergeben. Das Feld *region size* ist dabei zu besetzen. Wurde Bit 1 in DX gesetzt, sind weiterhin die Felder *region offset* und *region segment* zu initialisieren. Der Aufbau der DDS-Struktur ist beim Aufruf AX=8102H beschrieben. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der Kodierung gemäß AX=8102H. Bei erfolgreichem Aufruf ist das Carry-Flag gelöscht und in der DDS sind die Felder *physical address* und *buffer id* gesetzt, sowie das Feld *region size* mit der Bufferlänge belegt.

VDS Release DMA Buffer (AX=8108H)

Dieser Aufruf erlaubt es, einen DMA-Puffer wieder freizugeben.

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 8108H                       º
º DX: Flags                       º
º ES:DI DMA Descriptor            º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
Û---------------------------------ì
```

Beim Aufruf wird das Register DX als Flag verwendet, welches die Aktion steuert. Für die einzelnen Bits gilt folgende Belegung:

```
Codierung des Flagregisters DX beim Aufruf

Bit 0:  reserviert (0)
    1:  Daten aus Puffer kopieren
 2-15:  reserviert (0)
```

In ES:SI wird die Adresse auf die DMA-Descriptor-Struktur (DDS) übergeben. Das Feld *buffer id* ist dabei zu besetzen. Wurde Bit 1 in DX gesetzt, sind weiterhin die Felder *region size*, *region offset* und *region segment* zu initialisieren. Der Aufbau der DDS-Struktur ist beim Aufruf AX=8102H beschrieben. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der Kodierung gemäß AX=8102H. Bei erfolgreichem Aufruf ist das Carry-Flag gelöscht.

VDA Copy into DMA Buffer (AX=8109H)

Dieser Aufruf erlaubt es, Daten in einen DMA-Puffer zu speichern.

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 8109H                       º
º DX: 0000                        º
º ES:DI DMA Descriptor            º
º BX:CX Offset                    º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
Û---------------------------------ì
```

Beim Aufruf ist das Register DX = 0000 zu setzen. Die Kombination BX:CX enthält den Offset in den DMA-Puffer.

In ES:SI wird die Adresse auf die DMA-Descriptor-Struktur (DDS) übergeben. Die Felder *buffer ID*, *region offset*, *region segment* und *region size* sind zu initialisieren. Der Aufbau ist beim Aufruf AX=8102H beschrieben. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der Kodierung gemäß AX=8102.

VDA Copy out of DMA Buffer (AX=810AH)

Dieser Aufruf erlaubt es, Daten aus einen DMA-Puffer zu transferieren.

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 8109H                       º
º DX: 0000                        º
º ES:DI DMA Descriptor            º
º BX:CX Offset                    º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
Û---------------------------------ì
```

Beim Aufruf ist das Register DX = 0000 zu setzen. Die Kombination BX:CX enthält den Offset in den DMA-Puffer.

In ES:SI wird die Adresse auf die DMA-Descriptor-Struktur (DDS) übergeben. Die Felder *buffer ID*, *region offset*, *region segment* und *region size* sind zu initialisieren. Der Aufbau ist beim Aufruf AX=8102H beschrieben. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der Kodierung gemäß AX=8102.

VDA Disable DMA Translation (AX=810BH)

Dieser Aufruf schaltet die DMA Übertragung aus. In BX ist die Nummer des DMA-Kanals zu übergeben.

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 810BH                       º
º DX: 0000                        º
º BX: DMA Kanal                   º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
Û---------------------------------ì
```

Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der Kodierung gemäß AX=8102.

VDA Enable Translation (AX=810CH)

Dieser Aufruf schaltet die DMA Übertragung ein. In BX ist die Nummer des DMA-Kanals zu übergeben.

```
Ö---------------------------------Ì
º      CALL:  INT 4B              º
º                                 º
º AX: 810CH                       º
º DX: 0000H                       º
º BX: DMA Kanal                   º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AL = Fehlercode                 º
º CF = 0 ok                       º
Û---------------------------------ì
```

Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt. Der Fehlercode in AL entspricht der Kodierung gemäß AX=8102. Bei gelöschtem Carry-Flag war der Aufruf erfolgreich. Falls der Zustand *disable count = 0* wird das Zero-Flag bei der Rückkehr gesetzt.

3.27 Common Access Method SCSI Interface (INT 4FH)

Dieser Interrupt wird durch die Spezifikation für das SCSI-Interface belegt. Die Spezifikation liegt nach meinen Informationen jedoch erst im Entwurf vor, so daß an dieser Stelle auf eine Beschreibung verzichtet wird.

3.28 GSS Computer Graphic Interface (INT 59H)

Dieser Interrupt wird durch das *Computer Graphic Interface* belegt. Über den Interrupt werden die Aufrufe aus verschiedenen Hochsprachen (Language Bindings) zum Kontroller abgewickelt. Der IBM-Graphic-Development-Kit benutzt diesen Interrupt ebenfalls.

```
Ö---------------------------------Ì
º      CALL:  INT 59              º
º                                 º
º DS:DX Puffer mit Array Pointer  º
û---------------------------------À
º      RETURN                     º
º CF = 1 Error                    º
º AX = Fehlercode                 º
º CF = 0 ok                       º
º AX = Returncode                 º
Û---------------------------------ì
```

Vor dem Aufruf wird in DS:DX die Adresse auf einen Puffer mit den Zeigern auf das Ausgabearray geladen. Das Carry-Flag zeigt nach dem Aufruf den Fehlerstatus an. Weitere Informationen sind den entsprechenden Herstellerunterlagen zu entnehmen.

3.29 AT&T Starlan Extended NetBIOS (INT 5BH)

Dieser Interrupt wird durch die erweiterte NetBIOS-Schnittstelle des Starlan Netzwerkes belegt. Da dieses Netzwerk im deutschsprachingen Raum kaum eingesetzt wird, verzichte ich an dieser Stelle auf eine Beschreibung.

3.30 NETBIOS Interface (INT 5CH)

Dieser Interrupt wird vom NETBIOS für Netzwerkzugriffe benutzt. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 5C              º
º                                 º
º ES:BX Adresse NCB               º
û---------------------------------À
º      RETURN                     º
º AL = Status                     º
Û---------------------------------ì
```

Im Registerpaar ES:BX wird die Adresse des Netzwerk-Control-Blocks übergeben. Die Beschreibung der Funktionen ist im Abschnitt über die Netzwerkfunktionen enthalten.

3.31 PC/TCP Packet Driver (INT 60H)

Dieser Interrupt wird von einigen Netzwerken (z.B. TCP) belegt.

3.32 EMS-Treiber (INT 67H)

Der LIMS-EMS-Treiber belegt diesen Interrupt zur Kommunikation mit dem EMS-Speicher. Die Aufrufschnittstelle ist im Kapitel über die Speichererweiterungen enthalten.

Für 80386-Rechner existieren weiterhin Programme (Virtual Control Programm Interface) zum Zugriff auf die erweiterten Eigenschaften des Prozessors. Diese benutzen ebenfalls einige Unterfunktionen des INT 67H. Genauere Informationen finden sich im Kapitel über die Speichermanager.

3.33 DECnet DOS CTERM Schnittstelle (INT 69H)

Unter DECnet DOS benutzt der Terminal Emulator den Interrupt zur Kommunikation.

3.34DECnet DOS LAT Schnittstelle (INT 6AH)

Unter DECnet DOS benutzt der Local Area Transporttreiber (LAT) den Interrupt zur Kommunikation.

3.35 Novell serial I/O (INT 6BH)

Dieser Interrupt wird von Novell zur Kommunikation über serielle Leitungen belegt.

3.36 DOS 3.2 Realtime Clock Update (INT 6CH)

Unter DOS 3.2 wird hier ein Treiber zur Kommunikation der Real Time Clock hinterlegt.

3.37 VGA-Interrupt (INT 6DH)

Dieser Interrupt wird von einigen VGA-Karten von IBM, Paradise, Video7 und NCR für interne Zwecke benutzt.

3.38 HP ES-12 Extended BIOS (INT 6FH)

Die Firma HP benutzt den INT 6FH für die Kommunikation mit dem Extended BIOS der ES-12-Rechner. Es sind folgende Unterfunktionen implementiert:

Read CMOS-Memory (AH = 22H)

Mit dieser Unterfunktion läßt sich ein Byte aus dem CMOS-RAM lesen. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 6F              °
°                                 °
° AH:  22H (Read CMOS RAM)        °
° BP:  12H                        °
° BL:  Adresse CMOS Byte          °
û---------------------------------À
°      RETURN                     °
° AH = Status                     °
° AL = gelesenes Byte             °
Û---------------------------------ì
```

Im Register AL wird das gelesene Byte aus dem CMOS-RAM zurückgegeben.

Write CMOS-Memory (AH = 24H)

Mit dieser Unterfunktion läßt sich ein Byte in das CMOS-RAM schreiben. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 6F              °
°                                 °
° AH:  24H (Write CMOS RAM)       °
° AL = neuer Wert                 °
° BP:  12H                        °
° BL:  Adresse CMOS Byte          °
û---------------------------------À
°      RETURN                     °
° AH = Status                     °
Û---------------------------------ì
```

Im Register AL wird das neue Byte für das CMOS-RAM übergeben.

Weitere Interrupts werden durch Netzwerktreiber und andere Software belegt. Novell Netware belegt zum Beispiel den INT 6F und den INT 7A. Der INT 7A bildet die Hauptschnittstelle zum IPX-Treiber. Weiterhin belegt der BASIC-Interpreter der IBM-PC's einige Vektoren im Bereich zwischen INT 83 und INT FF.

4 Die MS-DOS-Funktionen (INT 21)

Der MS-DOS-Interrupt 21 besitzt aus Sicht des Anwendungsprogrammierers die wohl wichtigsten Funktionen. Über diesen Interrupt wickelt DOS fast alle Systemdienste ab. Beginnend mit der Version DOS 2.0 werden immer mehr Funktionen geboten, wobei allerdings einige für interne Zwecke reserviert sind und deshalb nicht dokumentiert wurden. Sofern die Parameterübergabe und die Funktion bekannt ist, wird sie nachfolgend mit beschrieben.

Die Funktionen dienen zur Programmverwaltung, Datenein-/ausgabe, Speicherverwaltung, etc. Historisch gesehen kann man zwischen den aus der CP/M-Welt übernommenen Diensten und den mehr XENIX orientierten Aufrufen unterscheiden. Dies resultiert aus der Entwicklung von DOS, welches in der Version 1.x sehr nahe mit CP/M verwandt ist. In den späteren Versionen wurde dann der Schwerpunkt auf die Erweiterungen Richtung XENIX und Netzwerkverwaltung gelegt. Die alten Aufrufe blieben aus Kompatibilitätsgründen aber erhalten. Nachfolgende Tabelle enthält einen kurzen Überblick über die Funktionsgruppen des INT 21.

```
  Code     ° Funktionsgruppe
-----------é-------------------------------------------------
  00H      ° Programm Terminate
  01 - 0CH ° Traditionelle Zeichen-Ein-/Ausgabe
  0D - 24H ° CP/M orientierte Dateiverwaltung
  25 - 26H ° Traditionelle nicht device-orientierte Funktionen
  27 - 29H ° CP/M orientierte Dateiverwaltung
  2A - 2EH ° Traditionelle nicht device-orientierte Funktionen
  2F - 38H ° Erweiterte Funktionen
  39 - 3BH ° Directory-Funktionen
  3C - 46H ° Erweiterte Dateiverwaltungsfunktionen
  47H      ° Directory-Funktionen
  48 - 4BH ° Speicherverwaltungsfunktionen
  4C - 4FH ° Erweiterte Funktionen
  54 - 57H ° Erweiterte Funktionen
  57 - 6CH ° Erweiterte Funktionen
```

Tabelle 4.1: Funktionsgruppen des INT 21

Im Text wird auf diese Funktionen hingewiesen, da für Programmneuentwicklungen nur noch die XENIX-orientierten Aufrufe verwendet werden sollten. Die nachfolgende Tabelle zeigt eine Gegenüberstellung der neuen und der alten Funktionen des INT 21.

```
-------------------------------------------------
Alter Aufruf                    Neue Funktion
Code Funktion                   Code Funktion
-------------------------------------------------
00H  Terminate Program          4CH  End Process
0FH  Open File                  3DH  Open Handle
10H  Close File                 3EH  Close Handle
11H  Search for First Entry     4EH  Find First File
12H  Search for Next Entry      4FH  Find Next File
13H  Delete File                41H  Delete Directory
14H  Sequential Read            3FH  Read Handle
15H  Sequential Write           40H  Write Handle
16H  Create File                3CH  Create Handle
17H  Rename File                56H  Change Directory Entry
21H  Random Read                3FH  Read Handle
22H  Random Write               40H  Write Handle
23H  Get File Size              42H  Move File Pointer
24H  Set Relative Record        42H  Move File Pointer
26H  Create New PSP             4BH  Load and Execute Program
27H  Random Block Read          3FH  Read Handle
28H  Random Block Write         40H  Write Handle
```

Tabelle 4.2: Alte und neue DOS-Funktionsaufrufe

Manche dieser Funktionen besitzen ihrerseits Unterfunktionen, die in der Regel über den Code im AL-Register ausgewählt werden. Die Funktionen werden grundsätzlich über den INT 21 verwaltet. Es bestehen aber aus Sicht des Anwendungsprogramms mehrere Möglichkeiten, diese Funktionen anzusprechen.

a) Intrasegment CALL 0005H

Aus Kompatibilitätsgründen zu früheren DOS-Versionen wurde die Möglichkeit dieses Aufrufes über den DOS-Funktions-Dispatcher beibehalten. Die Nummer der gewünschten Funktion ist im Register CL zu übergeben. Weiterhin sind die restlichen Register gemäß den Anforderungen der jeweiligen Unterfunktion zu setzen. Anschließend muß ein *Intrasegment CALL* auf die Adresse 05H ausgeführt werden. Dort findet sich das aktuelle *Program Segment Prefix,* welches ab Offset 05H den Code für einen *Long CALL* zum DOS-Dispatcher enthält. Diese Aufrufart besitzt jedoch verschiedene Nachteile:

- Es sind nur Aufrufe innerhalb des 64-Kbyte-Segmentbereiches möglich. Wird das Programm verschoben, findet sich das PSP nicht mehr im gleichen Segment. Ein CALL auf diese Adresse führt dann zu einem Systemabsturz.
- Das Register AX wird beim Aufruf zerstört, während bei anderen Aufrufarten die Zustände der Register vor dem Aufruf gesichert werden. Weiterhin wird in diesen Fällen im Register AX ein Fehlercode übergeben.
- Der Aufruf ist nur bei den INT 21-Funktionen 00H bis 24H erlaubt.
- Es sind im Prinzip 2 CALL-Aufrufe notwendig, bis der Dispatcher aktiviert wird (code- und zeitaufwendig).
- Die Aufwärtskompatibilität zu neuen DOS-Versionen ist nicht gewährleistet.

Aus diesen Gründen sollte diese Aufrufkonvention nicht mehr benutzt werden.

b) Intersegment CALL PSP:0050H

Die zweite Möglichkeit besteht darin, den Funktionscode im Register AH zu plazieren. Die anderen Register sind gemäß den Anforderungen der jeweiligen Funktion zu initialisieren. Dann ist ein *Intersegment Long CALL* zur Offsetadresse 0050H des aktiven Program-Segment-Prefix (PSP) durchzuführen. In diesem PSP steht ab Offsetadresse 50H der Programmcode für einen INT 21-Aufruf, der den DOS-Funktions-Dispatcher dann aktiviert. Die Benutzung dieser Aufrufart ist ebenfalls mit einigen Nachteilen behaftet:

- Wird der Wert des Codesegmentregisters (CS) geändert, ist die Adresse des PSP nicht unbedingt im Programm bekannt.
- Der Long CALL benötigt insgesamt 5 Byte, was bei vielen Aufrufen die Programmlänge vergrößert.
- Der indirekte Aufruf über einen Long CALL mit nachgeschaltetem INT 21 ist zeitintensiv.

Da im Grunde der INT 21 zum Aufruf des Dispatchers benutzt wird, sollte diese Aufrufart nicht weiter benutzt werden.

c) Aufruf über den INT 21

Dies ist die zu bevorzugende Aufrufart für die Aktivierung des DOS-Funktions-Dispatchers. Hierzu ist der Code der Funktion im Register AH zu definieren. Die restlichen Register sind gemäß den Anforderungen der jeweiligen Unterfunktion zu initialisieren. Dann ist ein INT 21-Befehl auszuführen. Dies ist die kürzeste und einfachste Form des Aufrufes, die folgende Vorteile bietet:

- Der INT 21 kann aus jedem Codebereich, ohne Kenntnis der Lage des PSP, aktiviert werden.
- Der INT 21 benötigt lediglich zwei Codebytes und veranlaßt, daß neben der Rückkehradresse auch der Zustand der Flagregister auf dem Stack gesichert wird.

Der Aufruf des DOS-Dispatchers über den Interrupt 21 wird von Microsoft für zukünftige Softwareentwicklungen empfohlen.

Nachfolgende Tabelle enthält die Aufstellung aller INT 21-Funktionsaufrufe, die anschließend detailliert besprochen werden.

Code	DOS	Funktion
00	2.0 - 6.x	Terminate Program
01	2.0 - 6.x	Read Keyboard and Echo
02	2.0 - 6.x	Display Character
03	2.0 - 6.x	Auxiliary Input
04	2.0 - 6.x	Auxiliary Output
05	2.0 - 6.x	Print Character
06	2.0 - 6.x	Direct Console I/O
07	2.0 - 6.x	Direct Console Input
08	2.0 - 6.x	Read Keyboard
09	2.0 - 6.x	Display String
0A	2.0 - 6.x	Buffered Keyboard Input
0B	2.0 - 6.x	Check Keyboard Status
0C	2.0 - 6.x	Flush Buffer, Read Keyboard
0D	2.0 - 6.x	Reset Disk
0E	2.0 - 6.x	Select Disk
0F	2.0 - 6.x	Open File
10	2.0 - 6.x	Close File
11	2.0 - 6.x	Search for First Entry
12	2.0 - 6.x	Search for Next Entry
13	2.0 - 6.x	Delete File
14	2.0 - 6.x	Sequential Read
15	2.0 - 6.x	Sequential Write
16	2.0 - 6.x	Create File
17	2.0 - 6.x	Rename File
19	2.0 - 6.x	Get Current Disk
1A	2.0 - 6.x	Set Disk Transfer Address
1B	2.0 - 6.x	Get Default Drive Data
1C	2.0 - 6.x	Get Drive Data
1F	2.0 - 6.x	Get Default Disk Parameter Block
21	2.0 - 6.x	Random Read
22	2.0 - 6.x	Random Write
23	2.0 - 6.x	Get File Size
24	2.0 - 6.x	Set Relativ Record
25	2.0 - 6.x	Set Interrupt Vector
26	2.0 - 6.x	Create New PSP
27	2.0 - 6.x	Random Block Read
28	2.0 - 6.x	Random Block Write
29	2.0 - 6.x	Parse File Name
2A	2.0 - 6.x	Get Date
2B	2.0 - 6.x	Set Date
2C	2.0 - 6.x	Get Time
2D	2.0 - 6.x	Set Time
2E	2.0 - 6.x	Set / Get Verify Flag
2F	2.0 - 6.x	Get Disk Transfer Address
30	2.0 - 6.x	Get MS-DOS Version Number
31	2.0 - 6.x	Keep Process
32	2.0 - 6.x	Get Disk Parameter Block
33	2.0 - 6.x	Control-C Check
34	2.0 - 6.x	Get DOS Critical Interval Flag
35	2.0 - 6.x	Get Interrupt Vector
36	2.0 - 6.x	Get Free Disk Space
37	2.0 - 6.x	Set Switch Character
38	2.0 - 6.x	Get Country Data
38	3.0 - 6.x	Set Country Data
39	2.0 - 6.x	Create Directory
3A	2.0 - 6.x	Remove Directory
3B	2.0 - 6.x	Change Current Directory
3C	2.0 - 6.x	Create Handle
3D	2.0 - 6.x	Open Handle

Tabelle 4.3: INT 21-Funktionsaufrufe

Code	DOS	Funktion
3E	2.0 - 6.x	Close Handle
3F	2.0 - 6.x	Read from a File or Device
40	2.0 - 6.x	Write to a File or Device
41	2.0 - 6.x	Delete (Unlink) Directory Entry
42	2.0 - 6.x	Move File Pointer
43	2.0 - 6.x	Get/Set File Attributes
4400	2.0 - 6.x	IOCTL Data
4402	2.0 - 6.x	IOCTL Character
4404	2.0 - 6.x	IOCTL Block
4406	2.0 - 6.x	IOCTL Status
4408	3.0 - 6.x	IOCTL is Changeable
4409	3.1 - 6.x	IOCTL is Redirected Block
440A	3.1 - 6.x	IOCTL is Redirected Handle
440B	3.0 - 6.x	IOCTL Retry
440C	3.3 - 6.x	Generic IOCTL Handle Request
440D	3.2 - 6.x	Generic IOCTL Request
440E	3.2 - 6.x	I/O Control for Device
440F	3.2 - 6.x	Set Logical Drive
4410	5.0 - 6.x	Query Support Handle
4411	5.0 - 6.x	Query Support Block
45	2.0 - 6.x	Duplicate File Handle
46	2.0 - 6.x	Force Duplicate File Handle
47	2.0 - 6.x	Get Current Directory
48	2.0 - 6.x	Allocate Memory
49	2.0 - 6.x	Free Allocated Memory
4A	2.0 - 6.x	Set Block
4B00	2.0 - 6.x	Load and Execute
4B01	2.0 - 6.x	Load Program
4B03	2.0 - 6.x	Load Overlay
4B05	5.0 - 6.x	Enter EXEC State
4C	2.0 - 6.x	Terminate a Process
4D	2.0 - 6.x	Get Returncode
4E	2.0 - 6.x	Find First Matching File
4F	2.0 - 6.x	Find Next File
50	2.0 - 6.x	Set Active PSP
51	2.0 - 6.x	Get Active PSP
52	2.0 - 6.x	Get DOS Data Address
53	2.0 - 6.x	Set Disk Parameter Block
54	2.0 - 6.x	Get Verify State
56	2.0 - 6.x	Rename File
57	2.0 - 6.x	Get / Set File Date and Time
58	3.0 - 6.x	Get / Set Allocation Strategy
59	3.0 - 6.x	Get Extended Error
5A	3.0 - 6.x	Create Temporary File
5B	3.0 - 6.x	Create New File
5C	3.0 - 6.x	Lock / Unlock File Access
5D00	3.1	Server Call
5D01	3.1	Update all Files
5D02	3.1	Close File by Name
5D03	3.1	Close Maschines File
5D04	3.1	Close Process File
5D05	3.1	Get Open File List Entry
5D06	3.0	Get DOS Variable Area
5D07	3.1	Redirect Printer
5D08	3.1	Redirect Printer
5D09	3.1	Redirect Printer
5D0A	3.1	Set Extended Error
5D0B	3.1	Get DOS Variable Areas
5E00	3.1 - 6.x	Get Machine Name
5E02	3.1 - 6.x	Set Printer Setup
5E03	3.1 - 6.x	Get Printer Setup
5E04	3.1 - 6.x	Set Printer Tabs
5E05	3.1 - 6.x	Get Printer Tabs
5F00	3.1 - 6.x	Set Redirection Mode
5F01	3.1 - 6.x	Get Redirection Mode
5F02	3.1 - 6.x	Get Redirection List Entry
5F03	3.1 - 6.x	Redirect Device
5F04	3.1 - 6.x	Cancel Redirection
5F05	4.0 - 6.x	Get Extended Redirect List
5F07	4.0 - 6.x	Activate Local Drive
5F08	4.0 - 6.x	Delete Local Drive
60	3.0 - 6.x	Expand Filename
62	3.0 - 6.x	Get Program Segment Prefix Address
63	2.25, 6.x	Get Lead Byte Table, DCBS Support
65	3.3 - 6.x	Get Extended Country Information
66	3.3 - 6.x	Get Global Code Page
66	3.3 - 6.x	Set Global Code Page

```
° 67   ° 3.3 - 6.x ° Set Handle Count                        °
° 68   ° 3.3 - 6.x ° Commit File                             °
° 69   ° 4.0 - 6.x ° Get/Set Disk Serial Number              °
° 6A   ° 4.0 - 6.x ° Update Network Units                    °
° 6C   ° 5.0 - 6.x ° Extended Open/Create                    °
Û------Ù-----------Ù-----------------------------------------ì
```

Tabelle 4.3: INT 21-Funktionsaufrufe (Ende)

In der Spalte »DOS4.1 Terminate Program (Funktion 00H, DOS 2.0-6.x)

Diese Funktion ermöglicht es, ein Programm zu beenden und gibt die Kontrolle an den Parent Process zurück. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö--------------------------------Ì
°      CALL:  INT 21             °
°                                °
° AH:  00H                       °
° CS:  Segmentadresse des PSP    °
°                                °
û--------------------------------À
°      RETURN                    °
° --                             °
°                                °
Û--------------------------------ì
```

Im Codesegment-Register muß die Adresse des Program-Segment-Prefix stehen. Beim Aufruf ist zu beachten, daß das Programm nicht länger als 64 Kbyte sein sollte. Dies ist unbedingt bei EXE-Dateien zu relevant. Wird die Funktion aufgerufen, ohne daß CS die Segmentadresse des PSP enthält, stürzt das System ab. Weiterhin müssen alle Dateien vorher geschlossen werden, da sonst die Änderungen nicht im Inhaltsverzeichnis der Diskette/Platte eingetragen werden und so verlorengehen.

Lediglich der Inhalt der DOS-Ausgabedateipuffer wird auf das Speichermedium ausgelagert und offene Handles werden noch geschlossen.

Die Funktion restauriert folgende Adressen aus dem Program-Segment-Prefix-Bereich:

```
Offset   Bedeutung

0AH      Program Terminate Address
0EH      Control-C Handler Address
12H      Critical Error Handler Address
```

auf den Wert des Parent Process. Wurde zum Beispiel die Adresse des Critical-Error-Handlers vom Child Process verändert, wird anschließend die alte Adresse restauriert.

Es gelten im Prinzip die gleichen Bedingungen, wie sie bereits beim INT 20 diskutiert wurden. Letztendlich ruft der INT 20 die Funktion 00 des INT 21 auf. Bei neuen Programmen sollte die INT 21-Funktion 4CH (End Process) benutzt werden.

4.2 Read Keyboard and Echo (Funktion 01H, DOS 2.0-6.x)

Diese Funktion erlaubt die Zeicheneingabe per Tastatur bei gleichzeitiger Ausgabe auf der Ausgabeeinheit. Es gelten folgende Registerbelegungen:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:  01H                      º
û-------------------------------À
º      RETURN                   º
º AL: eingelesenes Zeichen      º
Û-------------------------------ì
```

Die Funktion liest das Zeichen immer von der Standardeingabeeinheit aus. Dies ist in der Regel die Tastatur. Die Eingabe kann aber z.B. auch über das DOS-Kommando:

```
C>PROGR < AUX:
```

von der Auxillary Einheit erfolgen. Eine weitere Möglichkeit besteht in der Zuordnung von COM1: zur Standardeingabeeinheit CON:. Die Umleitung ist dagegen nicht zulässig, da anschließend die Endekennung (Ctrl+Z) nicht mehr erkannt wird. Die Funktion wartet, bis ein Zeichen vorliegt, bevor sie es auf der Standardausgabeeinheit ausgibt. Dann wird zum aufrufenden Programm zurückgekehrt. Das eingelesene Zeichen befindet sich im AL-Register.

Nun tritt bei der Tastatur aber das Problem auf, daß bestimmte Tastenkombinationen nicht mehr mit 8 Bit darstellbar sind (z.B. Funktionstasten). Um diese Tastencodes an ein Programm zu übergeben, wurde ein erweiterter (extended) ASCII-Code definiert. Dieser besteht aus zwei Byte. Das erste Byte eines extended ASCII-Codes ist immer 00H. Im folgenden Byte steht dann der Code der jeweiligen Taste. Die Zuordnung der Tasten ist den jeweiligen Rechnerunterlagen zu entnehmen.

Um einen erweiterten ASCII-Code von der Tastatur zu lesen, sind zwei Aufrufe der Funktion 01H notwendig. Falls ein erweiterter Code vorliegt, wird beim ersten Aufruf der Wert 00H zurückgegeben. Es wird kein Fehlerstatus durch die Funktion zurückgegeben. Auch wenn keine Zeichen empfangen werden, bricht die Funktion nicht ab, sondern wartet auf ein eintreffendes Zeichen. Das rufende Pogramm kann somit nicht abgearbeitet werden.

Beim Aufruf wird die Control-C-Routine aktiviert. Falls diese Tasten gedrückt wurden, löst DOS einen INT 23 aus.

4.3 Display Character (Funktion 02H, DOS 2.0-6.x)

Diese Funktion erlaubt es, ein Zeichen im DL-Register an die Standard-Ausgabeeinheit zu senden. Es gelten folgende Registerbelegungen.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:  02H                      º
º DL:  auszugebendes Zeichen    º
û-------------------------------À
º      RETURN                   º
º --                            º
Û-------------------------------ì
```

Es wird keine Status- oder Fehlermeldung zurückgegeben. Die Funktion ermittelt lediglich beim Aufruf den Control-C-Status und löst gegebenenfalls einen INT 23 aus.

Bei Ausgaben in Dateien funktioniert <Ctrl>+<C> nur, falls Break = ON gesetzt wird. Bei einer Ausgabe in eine Datei kann eine weitere Eigenart zu Problemen führen. Die Funktion

führt keine Fehlerprüfung aus, so daß ein volles oder schreibgeschütztes Medium nicht erkannt wird. Ab DOS 2.0 sollte deshalb die Funktion 40H des INT 21 benutzt werden.

4.4 Auxiliary Input (Funktion 03H, DOS 2.0-6.x)

Diese Funktion erwartet ein Zeichen von der Standard-Auxiliary-Einheit. Als Auxiliary-Einheiten lassen sich in DOS die AUX:, COM1: und COM2: angeben. Es ist zu beachten, daß DOS für die Auxiliary-I/O-Devices weder Puffer anlegt, noch einen interruptgesteuerten Ablauf zuläßt. Falls ein Zeichen empfangen wird, gibt die Routine es im AL-Register zurück. Vor der Benutzung müssen die Übertragungsparameter der Schnittstelle durch das DOS-Programm MODE gesetzt werden.

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:  03H                       º
û--------------------------------À
º      RETURN                    º
º AL: empfangenes Zeichen        º
Û--------------------------------ì
```

Es wird kein Fehlerstatus von dieser Funktion zurückgegeben.

Die Funktion fragt beim Aufruf ebenfalls den Control-C-Status des Systems ab und löst gegebenenfalls einen INT 23 aus.

Ab DOS 2.0 sollte die Funktion 3FH benutzt werden.

4.5 Auxiliary Output (Funktion 04H, DOS 2.0-6.x)

Diese Funktion gibt ein Zeichen auf der Standard-Auxiliary-Einheit aus. Es wird kein Fehler- oder Statuscode zurückgegeben.

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:  04H                       º
º DL:  auszugebendes Zeichen     º
û--------------------------------À
º      RETURN                    º
º --                             º
Û--------------------------------ì
```

Als Auxiliary-Einheiten lassen sich in DOS die AUX:, COM1: und COM2: angeben. Es ist zu beachten, daß DOS für die Auxiliary-I/O-Devices weder Puffer anlegt, noch einen interruptgesteuerten Ablauf zuläßt. Ab DOS 2.0 sollte die Funktion 40H benutzt werden.

Die Funktion fragt beim Aufruf ebenfalls den Control-C-Status der Tastatur ab und löst gegebenenfalls einen INT 23 aus. Die Übertragungsparameter der Schnittstelle müssen vor der Benutzung durch den Mode-Befehl gesetzt werden. Die Funktion wartet bei belegter Ausgabe solange, bis das BIOS den AUX:-Treiber als frei meldet.

4.6 Print Character; (Funktion 05H, DOS 2.0-6.x)

Diese Funktion überträgt Zeichen an den Standard-Druckertreiber. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:  05H                       º
º DL:  Zeichen                   º
û--------------------------------À
º      RETURN                    º
º ----                           º
Û--------------------------------ì
```

Das auszugebende Zeichen wird im *DL*-Register übergeben. Die Funktion prüft den Control-C-Status und löst gegebenenfalls einen INT 23 aus.

Es wird kein Status oder Fehlercode zurückgegeben, d.h. bei abgeschaltetem Drucker erfährt das Programm nichts von dem Fehler.

DOS wartet vielmehr bis der Drucker bereit ist, oder die Time-Out-Zeit abgelaufen ist. Ab DOS 2.0 ist deshalb die Funktion 40H zu bevorzugen.

4.7 Direct Console I/O (Funktion 06H, DOS 2.0-6.x)

Diese Funktion erlaubt die Zeichen-Ein-/Ausgabe über die Standard-Konsole. Der Inhalt des Registers DL bestimmt, ob ein Zeichen gelesen oder ausgegeben werden soll. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:  06H                       º
º DL:  Zeichen oder  Steuerflag  º
û--------------------------------À
º      RETURN                    º
º AL:  gelesenes Zeichen         º
º Z :  Flag, ob Zeichen gültig   º
Û--------------------------------ì
```

Enthält das Register DL vor Aufruf der Funktion den Wert 0FFH, dann wird versucht ein Zeichen einzulesen. Das Zero-Flag zeigt den Status an. Ist es gesetzt, dann konnte kein Zeichen gelesen werden. Falls Z = 0 ist, dann wird das Zeichen im AL-Register zurückgegeben.

Werden über die Funktion 06H Zeichen gelesen, dann ist zu prüfen, ob erweiterte ASCII-Codes vorliegen. In diesem Fall ist der erste gelesene Wert = 00H. Dann kann der Code des Zeichens erst beim zweiten Aufruf gelesen werden.

Falls der Inhalt des DL-Registers <> 0FFH ist, wird der Wert als Zeichen interpretiert und an die Standard-Ausgabeeinheit weitergegeben.

Die Funktion 06H prüft nicht den Control-C-Status des Systems.

4.8 Direct Console Input (Funktion 07H, DOS 2.0-6.x)

Diese Funktion erlaubt es, ein Zeichen von der Standard-Eingabeeinheit zu lesen. Die Funktion wartet so lange, bis ein Zeichen eintrifft. Bei I/O-Umleitung werden die Zeichen aus einer Datei gelesen. Die Funktion erkennt kein EOF (Ctrl+C), wodurch sich das Programm aufhängt.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:  07H                      º
û-------------------------------À
º      RETURN                   º
º AL:  gelesenes Zeichen        º
Û-------------------------------ì
```

Das Zeichen findet sich anschließend im AL-Register. Es wird kein Fehler- oder Statuscode zurückgegeben. Die Funktion überprüft auch nicht den Control-C-Status. Das eingegebene Zeichen wird auch nicht als Echo an der Standard-Ausgabeeinheit angezeigt.

4.9 Read Keyboard (Funktion 08H, DOS 2.0-6.x)

Diese Funktion erwartet ein Zeichen von der Standard-Eingabe. Es wird so lange gewartet, bis das Zeichen eintrifft. Bei Verwendung der DOS-I/O-Umleitung liest die Funktion Zeichen aus der angegebenen Datei und bleibt hängen, da EOF nicht erkannt werden kann.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:  08H                      º
û-------------------------------À
º      RETURN                   º
º AL:  empfangenes Zeichen      º
Û-------------------------------ì
```

Bei erfolgreichem Aufruf wird das Zeichen im AL-Register zurückgegeben. Es wird kein Fehlerstatus von der Funktion ausgegeben. Die Funktion fragt beim Aufruf lediglich den Control-C-Status der Tastatur ab und löst gegebenenfalls einen INT 23 aus.

Die Verarbeitung von erweiterten ASCII-Codes erfordert einen zweimaligen Aufruf der Funktion. Beim ersten Aufruf wird in diesem Fall der Wert 00H zurückgegeben. Erst ein weiterer Aufruf liefert den Code der jeweiligen Sondertaste. Ist das *interim console flag* gesetzt (siehe AX=6301H), werden einzelne Bytes einer Double Byte Character Sequence (DCBS) zurückgegeben (gilt nur für asiatische DOS-Versionen).

4.10 Display String (Funktion 09H, DOS 2.0-6.x)

Diese Funktion gibt einen String, der mit dem Zeichen $ endet, auf der Standard-Ausgabeeinheit aus. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:   09H                    º
º DS:DX Zeiger auf den String- º
º       anfang                 º
û------------------------------À
º      RETURN                  º
º ---                          º
Û------------------------------ì
```

Das $-Zeichen wird nicht mit ausgegeben. Das Registerpaar DS:DX enthält einen Zeiger auf die Anfangsadresse des Strings, wobei in DX der Offset steht. Die Stringlänge ist auf das aktuelle Datensegment begrenzt (maximal 64 Kbyte). Die Funktion gibt weder Fehler- noch Statusmeldungen zurück. Es wird lediglich der Control-C-Status des Systems abgefragt und gegebenenfalls ein INT 23 ausgelöst. Die Funktion verändert das Register AL bei der Ausgabe. Bei I/O-Umleitung in Dateien ist zu beachten, daß die Funktion keine Status- oder Fehlermeldungen zurückgibt. Ist zum Beispiel der Datenträger voll, hängt sich das Programm auf. Die Ausgabe erfolgt über das BIOS, wobei die Zeichencodes 07, 08, 09, 0A und 0D interpretiert werden.

4.11 Buffered Keyboard Input (Funktion 0AH, DOS 2.0-6.x)

Diese Funktion liest einen String von der Standard-Eingabeeinheit und überträgt ihn in den angegebenen Puffer im Hauptspeicher. Es gelten folgende Übergabeparameter.

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    0AH                   º
º DS:DX  Zeiger auf den Puffer- º
º        anfang                º
û------------------------------À
º      RETURN                  º
º --                           º
Û------------------------------ì
```

Das Registerpaar DS:BX enthält einen Zeiger auf den Pufferbereich, in den der String übertragen werden soll.

Dieser Puffer muß folgenden Aufbau besitzen:

```
DS:DX    Ö------------------------------Ì
  ------>º Pufferlänge in Bytes          º
         û------------------------------À
         º Zeichenzähler                 º
         û------------------------------À
         º Puffer                        º
         º   .                           º
         º   .                           º
         Û------------------------------ì
```

Bild 4.1: Der DOS- Tastaturzwischenpuffer

Die Länge des Puffers wird immer ohne die zwei ersten Byte für Pufferlänge und Zeichenzahl angegeben. Im Byte für die Zeichenzahl findet sich ein Wert, der proportional zur Zahl der im Puffer abgespeicherten Zeichen ist. Daran schließt sich ein Bereich von n Byte an, der für die Aufnahme der Zeichen reserviert wurde. Dies ist die im ersten Byte spezifizierte Pufferlänge. So lassen sich insgesamt bis zu 255 Zeichen ablegen. Die

angegebene Pufferlänge muß größer als Null sein, wobei ja die ersten beiden Byte (Len und Count) nicht mitgerechnet werden.

Die Funktion liest Zeichen für Zeichen ein und überträgt diese in den Puffer. Dabei werden die Informationen im Pufferkopf mit verwaltet. Tritt ein RETURN (0DH) auf, bricht die Funktion ab. Vorher wird dieses RETURN-Zeichen aber noch abgespeichert. Dieses RETURN wird allerdings nicht innerhalb des Zeichenzählers berücksichtigt. Falls der Puffer überläuft, bevor ein RETURN erkannt wurde, ignoriert die Funktion die zuletzt eingegebenen Zeichen und sendet jeweils ein BELL (07H) Signal an die Ausgabeinheit, bis ein RETURN eingelesen wird. Bei einer Zeicheneingabe per Konsole lassen sich diese so lange editieren, bis ein RETURN empfangen wurde. Falls die Tastenkombination Control-C betätigt wird, löst die Funktion einen INT 23 aus. Falls keine I/O-Umleitung vorgenommen wurde, kann DOS den INT 28 aufrufen, solange kein Zeichen von der Tastatur kommt. Beim Einlesen aus einer Datei (I/O-Umleitung) besteht keine Möglichkeit zur Statusabfrage. Sind die Eingabezeilen länger als 254 Zeichen, gehen Zeichen verloren.

4.12 Check Keyboard Status (Funktion 0BH, DOS 2.0-6.x)

Diese Funktion überprüft den Status der Standard-Eingabeeinheit und gibt diesen im AL-Register zurück.

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:    0BH                     º
Û--------------------------------À
º      RETURN                    º
º AL:  Tastaturstatus            º
Û--------------------------------ì
```

Sind Zeichen im Tastaturpuffer vorhanden, wird dies mit dem Wert AL = FFH signalisiert, andernfalls ist AL = 0. Falls die Eingabe nicht umgeleitet wurde, fragt die Funktion den Zwischenpuffer im BIOS-Datenbereich ab. Beim Einlesen aus Dateien (I/O-Umleitung) bedeutet AL=FFH, daß das Dateiende noch nicht erreicht ist.

In asiatischen DOS-Versionen werden Zeichen bei zeichenorientierten Geräten in 2 Byte, auch als DBCS bezeichnet (Double Byte Character Sequence), abgelegt. Ist das *interims console flag* gesetzt, liefert der Funktionsaufruf ein Byte. Gleichzeitig ist AL = FFH gesetzt um anzuzeigen, daß noch ein zweites Byte zu lesen ist.

Steht in diesem die Control-C-Sequenz, wird ein INT 23 ausgelöst. Die Funktion 0BH erlaubt somit die Abfrage, ob Zeichen im Puffer vorliegen, bevor die Funktion 0AH aufgerufen wird.

4.13 Flush Buffer, Read Keyboard (Funktion 0CH, DOS 2.0-6.x)

```
Ö-------------------------------Ì
º      CALL:   INT 21           º
º                               º
º AH:  0CH                      º
º AL:  Code der Subfunktion     º
û-------------------------------À
º      RETURN                   º
º AL:  0 -> Puffer ist leer     º
Û-------------------------------ì
```

Der Standard-Eingabepuffer wird beim Aufruf gelöscht (Flush). Weitere Aktionen hängen dann vom Wert des Registers AL ab.

Falls die Eingabe nicht umgeleitet wurde, löscht der Funktionscode AL=00H den Tastaturpuffer im BIOS-Datenbereich.

Andere Werte in AL erlauben es, die MS-DOS-Funktionen:

```
01  Read Keyboard and Echo
06  Direct Console I/O
07  Direct Console Input
08  Read Keyboard
0A  Buffered Keyboard Input
```

nach dem Löschen des Puffers zu aktivieren. In diesem Fall gelten die Registerbelegungen der jeweiligen Subfunktion. Andere als die oben spezifizierten Werte im Register AL werden von der Funktion abgewiesen, so daß nur der Puffer gelöscht wird. In diesem Fall zeigt das Register AL nach der Rückkehr den Wert 0. Ansonsten gilt für AL die Belegung wie bei den Subfunktionen 01, 06, 07 und 08, d.h. AL enthält das gelesene Zeichen. Bei Löschaktionen (AL=00H) und bei AL = 0AH bleibt AL unverändert erhalten.

Abgesehen von fehlerhaften Subfunktionscodes im Register AL gibt die Routine keine weiteren Status- oder Fehlermeldungen zurück. Lediglich bei Anwahl der Unterfunktionen wird der Control-C-Status des Systems überprüft und gegebenfalls ein INT 23 ausgelöst. Diese Funktion läßt sich sehr gut bei Sicherheitsabfragen (z.B. alle Dateien löschen [J/N] ?) verwenden, da vor dem Einlesen der Benutzereingabe erst der Tastaturpuffer geleert wird. Der Benutzer ist also gezwungen, explizit auf die Abfrage zu reagieren und kann nicht im voraus Zeichen eingeben.

Die Löschfunktion bezieht sich nur auf den Tastaturpuffer. Bei Verwendung der I/O-Umleitung wird die Eingabedatei nicht beeinflußt, d.h. hier empfiehlt es sich die gewünschte Eingabefunktion (01, 06, 07, etc.) direkt aufzurufen. Zu beachten ist weiterhin, daß durch das Löschen eventuell im Puffer befindliche Ctrl+C-Zeichen entfernt werden, d.h. das Programm läßt sich unter Umständen nicht abbrechen.

4.14 Reset Disk (Funktion 0DH, DOS 2.0-6.x)

Der Aufruf setzt die interne DOS-Dateiverwaltung in einen definierten Zustand zurück. Hierzu werden alle Blocktreiber mit dem Kommando »FLUSHe-Befehl zuständig (s. auch INT 21, Funktion 3EH oder 5D01H, 68H, 6AH, INT 2F, Funktion AX=1120).

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:  0DH                      º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Der Aufruf *Reset Disk* wird meist benutzt, um das System in einen bekannten Zustand zu versetzen. Der Control-C-Handler sollte z.B. diese Funktion benutzen, um vor der Rückkehr nach DOS das Dateisystem in einen definierten Zustand zu versetzen. Es wird kein Status- oder Fehlercode zurückgegeben.

4.15 Select Disk (Funktion 0EH, DOS 2.0-6.x)

Diese Funktion selektiert ein Standard-Laufwerk.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:   0EH                     º
º DL:   Laufwerksnummer         º
û-------------------------------À
º      RETURN                   º
º AL: Zahl der logischen Dives  º
Û-------------------------------ì
```

Das Register DL spezifiziert das logische Laufwerk, welches selektiert werden soll. Es gilt die Codierung A: = 0, B: = 1, etc. Die Funktion weicht damit in der Kodierung der Laufwerke (0 = A:, 1 = B:, etc.) von den üblichen DOS-Konventionen (1 = A:, 2 = B:, etc.) ab. Lediglich die Funktion 19H des INT 21 benutzt die gleiche Kodierung wie die Funktion 0EH.

Nach dem Aufruf bezieht DOS alle folgenden INT 21-I/O-Anforderungen ohne explizite Laufwerksangabe auf das per Funktion 0EH selektierte Laufwerk. Wird in DL ein nicht existierendes Laufwerk adressiert, ignoriert DOS den Funktionsaufruf und läßt die alte Einstellung bestehen. Da die Funktion keinen Status zurückgibt erhält der rufende Prozess über diesen Vorgang allerdings keine Rückmeldung.

Im AL-Register gibt die Funktion allerdings die Zahl der logischen Laufwerke innerhalb des Systems zurück. Wird dieser Wert um 1 erniedrigt, entspricht der Wert dem Code des letzten logischen Laufwerkes. Mit AL = 2 stehen dann die Diskettenlaufwerke A: und B: zur Verfügung. Allerdings muß der Wert mit Vorsicht betrachtet werden, da im Prinzip nur die Nummer des höchsten Laufwerkes angegeben wird. Falls z.B. der Wert 5 zurückgegeben wird, ist nicht sicher, daß die Laufwerke A B C D E auch wirklich existieren. Es ist zusätzlich zu beachten, daß neben der Zahl der physikalisch vorhandenen Einheiten auch die Angabe LASTDRIVE = xx, innerhalb der Datei CONFIG.SYS, den Wert in AL beeinflußt. Die Funktion gibt den jeweils größeren Wert aus beiden Einstellungen (Laufwerkszahl und LASTDRIVE) zurück. Systeme mit einem Diskettenlaufwerk ordnen diesem die Einheitennummern 0 = A:, 1 = B: zu. Ab DOS 3.0 gibt die Funktion immer eine Minimalanzahl von 5 Laufwerken im Register AL zurück. Bei Novell NetWare wird beim Aufruf in AL dagegen immer der Wert 32 (20H) zurückgegeben, was der Zahl der maximal unterstützten Laufwerke entspricht. Dies wird von vielen Zusatz-Tools nicht berücksichtigt, so daß beim *Sysinfo* oft falsche Zahlen auftauchen.

4.16 Open File (Funktion 0FH, DOS 2.0-6.x)

Diese Funktion dient zur Eröffnung einer Datei. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö-------------------------------Ì
°      CALL:  INT 21            °
°                               °
° AH:   0FH                     °
° DS:DX Zeiger auf FCB          °
û-------------------------------À
°      RETURN                   °
° AL:  Statusflag               °
Û-------------------------------ì
```

Im Registerpaar DS:DX findet sich ein Zeiger auf einen ungeöffneten File-Control-Block (FCB). Dieser FCB ist nichts anderes als eine Datenstruktur im Speicher, die von CP/M übernommen wurde und in der vor dem Aufruf der Dateiname hinterlegt wurde.

Diese Struktur umfaßt in der Standard Version 37 Byte und wird in der erweiterten Version auf 44 Byte verlängert. Die DOS-Funktionen speichern in diesem Bereich weitere Informationen über die Datei ab. Der Aufbau dieses FCB wird in einem eigenen Kapitel beschrieben. Beim Aufruf der Funktion wird das Inhaltsverzeichnis der Disk nach dem Dateinamen durchsucht. Falls ein Eintrag vorliegt, gibt die Funktion im Register AL den Wert 0 zurück. Anschließend werden folgende Daten in den FCB geschrieben:

Ist die Laufwerksnummer (Offset 0) auf den Wert 0 gesetzt, was per Konvention *Default Drive* bedeutet, dann ersetzt die Funktion den Wert durch 1 = A:, 2 = B:, etc. Dies ist notwendig, da ja bei einem späteren Dateizugriff die Einstellung des Default-Drive geändert sein kann, womit die geöffnete Datei nicht mehr gefunden wird.

Der *Current Block* (Offset 0CH-0DH) wird auf den Wert 0 initialisiert, womit dieser Zeiger auf den ersten Record innerhalb der Datei zeigt. Die Recordlänge (Offset 0EH-0FH) wird auf den Standardwert von 128 Byte pro Satz gesetzt.

Dateigröße (Offset 10H), Datum (Offset 14H) und Zeit (Offset 16H) werden mit den Werten aus dem Inhaltsverzeichnis der Diskette/Platte belegt.

Falls kein Directory-Eintrag gefunden wird, oder falls die Datei mit dem Attribut *System* belegt ist, gibt die Funktion den Wert 0FFH im Register AL zurück. Bei erfolgreichem Aufruf enthält AL den Wert 00H.

Vor einem ersten sequentiellen Zugriff auf die Datei ist der Wert des *Current Record Field* (Offset 20H) auf den ersten zu lesenden Block zu setzen. Bei einem Random Zugriff ist das *Relativ Record Field* (Offset 21H) auf den entsprechenden Wert zu setzen. Stimmt die Recordlänge von 128 Byte nicht, kann sie durch einen Eintrag ab Offset 0EH angepaßt werden. Alle Einstellungen sind nach dem *Open File Call*, aber vor dem ersten Dateizugriff vorzunehmen. Die Funktion 0EH (Open File) gehört zu den älteren CP/M orientierten Dateioperationen. Sie unterstützt keine netzwerkorientierten Dienste (SHARE) und keine Aufrufe mit Pfadangaben. Um mit den XENIX-Aufrufen (per Handle) kompatibel zu sein, darf der Zugriff nur im Kompatibilitäts-Modus erfolgen. In diesem Mode darf die Datei zwar beliebig oft, aber nur von einem Prozeß, geöffnet werden. Falls die Datei bereits in einem anderen Mode eröffnet wurde, resultiert der File-Open-Call im Kompatibilitätsmode in einem INT 24 (Critical Error). Auf dem Bildschirm erscheint die Meldung *Drive not ready*. Weiterhin wirkt sich bei Verwendung der FCB-Funktionen ein DOS-Programmierfehler aus. Der FCB wird standardmäßig im Program-Segment-Prefix

(PSP) ab Offset 5CH angelegt. Das PSP ist eine 256 Byte lange Datenstruktur, die von DOS vor jedem Programm im Speicher ab Offset 0000H angelegt wird. Da die FCB-Struktur aber 37 Byte umfaßt, reicht das letzte Byte bis zur Adresse 80H. Hier beginnt aber bereits ein Bereich (80H bis FFH), der durch DOS für zwei weitere Zwecke benutzt wird. Einmal wird hier beim Start des Programmes die Kommandozeile des Aufrufes gespeichert. Damit können beim Aufruf Parameter (z.B. DIR /W) an das Programm übergeben werden. Weiterhin benutzt DOS den Bereich ab Offset 80H als Disk-Transfer-Area (DTA). Hierbei handelt es sich um einen 128-Byte-Puffer, in dem Daten für die Disketten-/Platten-Ein-/Ausgabe zwischengespeichert werden. Damit kommt es bei Verwendung von FCB's zu Überlappungen der einzelnen Puffer, die durch den Anwenderprogrammierer abzufangen sind. Ein Ausweg ist die Verlagerung des FCB's in einen anderen Speicherbereich, wobei dann die neue Adresse beim Funktionsaufruf in DS:DX anzugeben ist. Weitere Hinweise über den Aufbau der FCB's und des PSP finden sich im Kapitel über die DOS-Datenstrukturen. Einzige interessante Eigenschaft der Funktion 0FH ist die Tatsache, daß im Filenamen Wildcardzeichen (*,?) auftreten dürfen. DOS öffnet dann die erste Datei, auf die das Suchmuster paßt. Da dieser Effekt allerdings nicht immer gewollt ist und angesichts der oben beschriebenen Probleme sollte für Neuentwicklungen die Funktion 3DH (Open Handle) verwendet werden, da diese wesentlich flexibler ist.

4.17 Close File (Funktion 10H, DOS 2.0-6.x)

Diese Funktion ist das Gegenstück zum Open File-Aufruf, da sie eine Datei schließt. Hierzu ruft die Funktion die Blocktreiber mit dem Code für FLUSH auf. Hierdurch werden eventuell noch im Puffer befindliche Daten auf das Medium ausgelagert. Als wichtigste Aufgabe übernimmt die Funktion es, bei veränderter Dateilänge diese Information, sowie das aktuelle Datum und die Uhrzeit in das Directory einzutragen. Es gelten folgende Parameterübergaben:

```
Ö-------------------------------Ì
º      CALL:   INT 21           º
º                               º
º AH:   10H                     º
º DS:DX Zeiger auf FCB          º
û-------------------------------À
º      RETURN                   º
º AL:   Status Flag             º
Û-------------------------------ì
```

Die Register DS:DX dienen wieder als Zeiger auf einen geöffneten FCB. Das Inhaltsverzeichnis wird nach dem angegebenen Namen durchsucht. Wird ein Eintrag gefunden, vergleicht die Funktion die Informationen innerhalb des FCB mit den Einträgen des Directory.

Bei Abweichungen werden die Daten aus dem FCB auf die Platte in den Directory-Bereich übertragen. Anschließend wird der Wert AL = 0 zurückgegeben. Falls kein Eintrag gefunden wird, enthält das Register AL den Wert FFH. Dies kann z.B. vorkommen, falls die Diskette seit dem *Open Call* gewechselt wurde, oder falls das aktuelle Unterverzeichnis auf der Platte gewechselt wurde. Bei gewechselten Disketten muß davon ausgegangen werden, daß DOS diesen Wechsel nicht sofort merkt. Der Close-Aufruf lagert dann erst einmal die Pufferdaten auf das Medium aus. Dabei werden die in internen Tabellen gespeicherten Positionsangaben benutzt, die natürlich bei der neuen Diskette nicht mehr

passen. Dies kann dazu führen, daß bereits auf der Diskette befindliche Daten zerstört werden. Die der Datei zugeordneten Ausgabepuffer werden gelöscht und zur weiteren Verwendung freigegeben. Der Aufruf dieser Funktion sollte immer dann erfolgen, wenn eine Datei nicht länger benötigt wird. Es ist dann sichergestellt, daß alle Änderungen auf dem Medium abgespeichert sind. Der Systemaufruf stammt aus der Menge der CP/M-orientierten Funktionen und sollte bei Neuentwicklungen durch den Aufruf 3EH (Close File Handle) ersetzt werden.

4.18 Search for First Entry (Funktion 11H, DOS 2.0-6.x)

Diese Funktion bezieht sich auf das aktuelle Inhaltsverzeichnis eines Speichermediums und erlaubt es, einen bestimmten Dateinamen zu suchen. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:    11H                     º
º DS:DX  Zeiger auf einen FCB    º
Û--------------------------------À
º      RETURN                    º
º AL:    Statusbyte              º
Û--------------------------------ì
```

Der Zeiger DS:DX muß auf einen ungeöffneten File Control Block (FCB) zeigen. In DX steht dabei die Offsetadresse dieses FCB. Der Dateiname innerhalb des FCB wird für den Vergleich verwendet. In DOS 2.x sind nur Fragezeichen (?) als Sonderzeichen innerhalb des Dateinamens erlaubt.

Ab DOS 3.0 dürfen alle globalen Zeichen, einschließlich des (*) als Wildcard, im Dateinamen vorkommen. Wird kein Eintrag im Inhaltsverzeichnis gefunden, enthält das AL-Register den Wert FFH als Status. Andernfalls wird 0 als Status zurückgegeben. Wenn ein Eintrag gefunden wird, dann legt die Funktion eine Kopie des FCB in der Disk-Transfer-Area (DTA) ab. Der Aufbau richtet sich danach, ob ein normaler oder ein erweiterter FCB verwendet wird. Erweiterte FCBs müssen z.B. benutzt werden, um System oder Hidden Files zu bearbeiten. Bei einem erweiterten FCB wird das erste Byte in der Disk-Transfer-Area (DTA) auf den Wert FFH gesetzt, während die folgenden 5 Byte 0 bleiben. Das sechste Byte erhält die Kopie des Attributbytes aus der Directory. Der restliche Bereich entspricht der Belegung eines normalen FCB, d.h., im nächsten Byte findet sich der Code für das Laufwerk (1 = A:, 2 = B:, etc.). Die verbleibenden 32 Byte enthalten den Directory-Eintrag.

Bei Verwendung von erweiterten FCBs kann durch das Attributbyte die Suche nach dem gewünschten Dateitype gesteuert werden. Mit dem Wert = 0 sucht die Funktion nur nach normalen Dateien, während die Namen von Unterverzeichnissen, sowie Dateien mit den Attributen *Hidden* oder *System*, nicht erkannt werden. Werte von 02H (Hidden-File), 04H (System-File) oder 10H (Directory-Eintrag) lassen sich ebenfalls angeben. In diesem Fall werden alle Dateien, die die gesetzten Bedingungen erfüllen, z.B. normale und Hidden-Files, zurückgegeben. Das Bitfeld läßt sich so setzen, daß alle Dateitypen, mit Ausnahme des *Volume Labels* gefunden werden. Lediglich bei der expliziten Suche nach diesem Volume Label wird nur dieses zurückgegeben. Eine detaillierte Beschreibung des FCB erfolgt in einem eigenen Kapitel.

Falls die Funktion 12H benutzt werden soll, darf der mit dem Zeiger DS:DX bezeichnete FCB zwischenzeitlich nicht verändert werden. Die Funktion 11H gehört noch zur Gruppe der älteren CP/M-orientierten Dateioperationen und unterstützt keine Pfadangaben. Bei Programmneuentwicklungen sollte die Funktion 4EH (Find First Matching File) benutzt werden.

4.19 Search for Next Entry (Funktion 12H, DOS 2.0-6.x)

Diese Funktion bezieht sich auf das aktuelle Inhaltsverzeichnis eines Speichermediums und wird nach der Funktion 11H benutzt, um weitere Einträge mit dem Dateinamen zu finden.

```
Ö------------------------------Ì
°      CALL:  INT 21           °
°                              °
° AH:    12H                   °
° DS:DX  Zeiger auf einen FCB  °
°                              °
û------------------------------À
°      RETURN                  °
° AL:    Statusbyte            °
°                              °
Û------------------------------ì
```

Es gelten die gleichen Bedingungen wie bei der Funktion 11H. Im Registerpaar DS:DX findet sich der Zeiger auf den ungeöffneten FCB, der beim Aufruf der Funktion 11H definiert wurde. Wichtig ist, daß der bei der Funktion 11H benutzte FCB zwischenzeitlich nicht durch andere Diskoperationen verändert wurde (z.B. durch Öffnen der Datei).

Wird ein weiterer Eintrag gemäß den in der Funktion 11H spezifizierten Bedingungen gefunden, enthält das Statusbyte im Register AL den Wert 00H. Ist die Suche erfolglos, wird der Wert 0FFH zurückgegeben. Die DTA enthält bei erfolgreichem Aufruf eine FCB-Struktur mit dem Namen der gefundenen Datei.

Der Aufruf 12H gehört zu den CP/M-orientierten Funktionen, und wurde mittlerweile durch den Aufruf 4FH (Find Next Matching File) ersetzt.

4.20 Delete File (Funktion 13H, DOS 2.0-6.x)

Mit der Funktion 13H läßt sich eine Datei im aktuellen Verzeichnis löschen. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö------------------------------Ì
°      CALL:  INT 21           °
°                              °
° AH:    13H                   °
° DS:DX  Zeiger auf einen FCB  °
û------------------------------À
°      RETURN                  °
° AL:    Statusbyte            °
Û------------------------------ì
```

Der Zeiger DS:DX ist einem FCB zugeordnet, der Laufwerk und Dateiname enthält. Wichtig ist, daß die zu löschende Datei geschlossen ist, d.h., der durch den Zeiger DS:DX adressierte FCB darf nicht geöffnet sein. Das Directory wird nach dem (innerhalb) des FCBs angegebenen Namen durchsucht.

Falls dieser existiert, wird er aus dem Inhaltsverzeichnis gelöscht und im Register AL wird 0 zurückgegeben. Andernfalls enthält AL den Wert FFH. Als Dateiname sind auch Wildcards (*,?) möglich. Wird keine Datei mit dem Muster gefunden, oder sind alle Files schreibgeschützt, dann enthält AL bei der Rückkehr den Wert FFH. Es ist zu beachten, daß nicht die Datei, sondern nur der Eintrag im Inhaltsverzeichnis gelöscht wird. Im Grunde wird sogar nur das erste Byte des Dateinamens auf den Wert E5H gesetzt (siehe Kapitel über das DOS-Inhaltsverzeichnis). Falls die Datei nicht geschlossen war, tritt bei der anschließenden Close-Operation ein Fehler auf, da ja die Datei nicht mehr existiert. Weiterhin darf bei der Datei nicht das *Read Only*-Attribut gesetzt sein. Ist dieses Attribut definiert, kann die Datei nicht gelöscht werden. Bei Einsatz innerhalb einer Netzwerkumgebung (ab DOS 3.1) benötigt die Funktion das Zugriffsrecht *Create Access*, um eine Datei zu löschen.

Die Funktion besitzt allerdings noch einen recht unangenehmen Effekt. Wird ein erweiterter FCB benutzt, kann der Filename und die Extension nach den Konventionen mit Wildcards ????????.??? auffüllt werden. Sind zusätzlich noch alle Attribute freigegeben, löscht ein Aufruf mit diesen Einstellungen (bis zur Version 6.0) alle Dateien im aktuellen Directory. Hierzu gehören auch Dateien, deren Directory-Attribut gesetzt ist, die demnach die Einträge für die Dateien im Unterverzeichnis enthalten. Dadurch geht das Unterverzeichnis und die darin befindlichen Dateien verloren, ohne das diese explizit gelöscht wurden. Hierdurch werden die durch diese Dateien belegten Cluster nicht mehr freigegeben. Das erste Byte wird dabei nicht auf E5H, sondern auf 00H gesetzt. Damit lassen sich die Dateien auch mit Hilfstools nicht mehr restaurieren. Weiterhin befinden sich anschließend jede Menge verlorene Cluster auf der Platte, die nur mit CHKDSK wieder freigegeben werden können.

Der Aufruf wurde mittlerweile durch die xenixorientierte Funktion 41H (Delete a File from the spezified Device) ersetzt.

4.21 Sequential Read (Funktion 14H, DOS 2.0-6.x)

Diese Funktion liest einen Satz (Record) aus der spezifizierten Datei im aktuellen Verzeichnis. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö------------------------------Ì
º      CALL:   INT 21          º
º                              º
º AH:    14H                   º
º DS:DX  Zeiger auf einen FCB  º
û------------------------------À
º      RETURN                  º
º AL:    Statusbyte            º
Û------------------------------ì
```

Das Registerpaar DS:DX enthält einen Zeiger auf den FCB. Die im FCB angegebene Datei muß bereits eröffnet sein. Der zu lesende Satz wird durch die FCB-Einträge *Current Block Field* (Offset 0CH) und *Current Record* (Offset 20H) spezifiziert. Der adressierte Satz wird dann gelesen und in der Disk-Transfer-Area abgelegt.

Die Länge eines Satzes ist im *Record Size Field* des FCB spezifiziert. Im Register AL wird der Status der Leseoperation zurückgegeben. Es gilt folgende Belegung:

```
 Code ° Status
------é-------------------------------------------------
   0  ° Funktion ohne Fehler ausgeführt
   1  ° Versuch, über das Dateiende hinaus zu lesen,
      ° keine Daten gelesen (EOF)
   2  ° Nicht genügend Platz in der DTA, der Lese-
      ° befehl wurde abgebrochen
   3  ° Dateiende wurde erreicht, ein Teilsatz wurde
      ° gelesen, die restlichen Bytes in der DTA wurden
      ° mit 00 aufgefüllt
```

Es ist zu beachten, daß die Standard DTA von DOS im PSP ab Adresse 80H angelegt wird. Damit stehen aber nur 128-Byte-Puffer zur Verfügung. Werden längere Datensätze im FCB spezifiziert, läuft der Pufffer in den Codebereich des zugehörigen Programmes über, d.h. das Programm wird zerstört. Hier muß eine neue DTA vom Programm angelegt werden, dessen Adresse mittels der INT 21-Funktion 1AH an DOS übergeben wird.

Ein Einsatz innerhalb eines Netzwerkes (ab DOS 3.1) setzt voraus, daß die Funktion das Zugriffsrecht *Read Access* auf der Datei besitzt. Bei neuen Programmen sind die Handle orientierten Funktionen des INT 21 zu nutzen.

4.22 Sequential Write (Funktion 15H, DOS 2.0-6.x)

Diese Funktion schreibt einen Satz (Record) in die spezifizierte Datei, die sich im aktuellen Verzeichnis befinden muß. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö--------------------------------Ì
°      CALL:  INT 21             °
°                                °
° AH:    15H                     °
° DS:DX  Zeiger auf einen FCB    °
û--------------------------------À
°      RETURN                    °
° AL:    Statusbyte              °
Û--------------------------------ì
```

Die im FCB angegebene Datei muß bereits eröffnet sein. Der zu schreibende Satz wird durch die FCB-Einträge Current Block Field (Offset 0CH) und Current Record (Offset 20H) spezifiziert. Der Satz wird dann aus der DTA gelesen und im adressierten Dateibereich ab der aktuellen Schreibposition gespeichert.

Die Lage des Schreibzeigers (current block * current record) bestimmt, ob die Daten an das Dateiende angehängt werden, oder ob die Bytes innerhalb der Datei überschrieben werden.

Die Länge eines Satzes ist im Record-Size-Field des FCB spezifiziert. Falls diese Länge kleiner als die Sektorgröße des Speichermediums ist, legt die Funktion die Daten in einem MS-DOS-Zwischenpuffer ab. Erst wenn dieser Puffer gefüllt ist, oder falls die Datei geschlossen wird, oder falls das Disk-System zurückgesetzt wird (Funktion 0DH), dann werden die Daten auf die Platte geschrieben. Nach diesem Schreibvorgang wird anschließend das FCB-Feld *Current Record* auf den neuen Wert gesetzt. Das Register AL enthält nach dem Aufruf den Status der Schreiboperation. Es gilt folgende Kodierung:

```
 Code ° Status
------é-----------------------------------------------
   0  ° Funktion ohne Fehler ausgeführt
   1  ° Disk voll, Schreibversuch abgebrochen
   2  ° Nicht genügend Platz in der DTA, der Schreib-
      ° befehl wurde abgebrochen
```

Beim Zugriff auf Dateien innerhalb eines Netzwerkes (ab DOS 3.1), benötigt der aufrufende Prozeß das Write-Access-Privileg. Wegen der Größe der DTA ist die Verwendung der Funktion etwas problematisch. Deshalb sollte bei Neuentwicklungen der Aufruf 40H (Write to a File or Device) verwendet werden.

4.23 Create File (Funktion 16H, DOS 2.0-6.x)

Mit dieser Funktion läßt sich eine Datei im aktuellen Verzeichnis neu anlegen. Hierzu muß im Register DS:DX ein Zeiger auf einen ungeöffneten FCB eingerichtet werden. Im Filenamen (im FCB) sind keine Wildcardzeichen (*,?) erlaubt. Für den Aufruf gelten folgende Übergabeparameter:

```
Ö-------------------------------Ì
°      CALL:  INT 21            °
°                               °
° AH:    16H                    °
° DS:DX  Zeiger auf einen FCB   °
°                               °
û-------------------------------À
°      RETURN                   °
° AL:    Statusbyte             °
°                               °
Û-------------------------------ì
```

MS-DOS prüft nach dem Aufruf, ob bereits eine Datei mit dem angegebenen Namen vorliegt. Falls kein Eintrag vorliegt, wird einer im Inhaltsverzeichnis angelegt. Existiert bereits eine Datei mit dem angegebenen Namen, dann setzt die Funktion die Länge auf 0, d.h., das bestehende File wird gelöscht und eine neue leere Datei wird angelegt.

Gleichzeitig wird die Datei geöffnet. Falls im Inhaltsverzeichnis kein freier Eintrag für die Datei gefunden wird, bricht die Funktion mit einer Fehlermeldung im Register AL ab. Hierfür existieren nur zwei Werte:

```
   Code ° Bedeutung
--------é--------------------------------------
    00  ° Leereintrag gefunden, Datei angelegt
    FF  ° Directory voll, Datei nicht angelegt
```

Die Dateiattribute lassen sich über erweiterte FCBs setzen. Für die Eröffnung der Datei gelten die gleichen Bedingungen wie bei der Funktion 0FH (Open File).

Falls auf eine Datei innerhalb eines Netzwerkes zugegriffen wird, benötigt die Funktion das Zugriffsrecht *Create Access*, um die Dateilänge auf den Wert 0 zu begrenzen.

Bei Softwareneuentwicklungen ist vorzugsweise die Funktion 3CH (Create a File) zu verwenden.

4.24 Rename File (Funktion 17H, DOS 2.0-6.x)

Mit diesem Aufruf läßt sich der Name einer existierenden Datei (im aktuellen Verzeichnis) ändern. Das Registerpaar DS:DX muß auf einen modifizierten FCB zeigen. In diesem FCB müssen Laufwerksnummer, erster Dateiname und zweiter Dateiname angegeben werden. Der modifizierte FCB besitzt dabei folgenden Aufbau:

```
  Bytes ° Bedeutung
--------é-----------------------------------------
    01  ° altes Laufwerk (0 = default, 1 = A:, etc.)
    08  ° alter Filename (Wildcards ? erlaubt)
    03  ° alte Extension
    01  ° neues Laufwerk = altes Laufwerk
    08  ° neuer Filename (Wildcards ? erlaubt)
    03  ° neue Extension
```

Die Laufwerksangabe zwischen altem und neuem Drive muß identisch sein. Umfaßt der Filename oder eine Extension nicht die maximale Länge von 8/3 Zeichen, sind die restlichen Byte mit dem Wert 00H aufzufüllen. In DOS 2.x sind nur ? als Wildcards erlaubt, während ab DOS 3.x auch * als Wildcards akzeptiert werden. Für die Funktion gelten folgende Aufrufkonventionen.

```
Ö-------------------------------Ì
°      CALL:  INT 21            °
°                               °
° AH:    17H                    °
° DS:DX  Zeiger auf einen FCB   °
û-------------------------------À
°      RETURN                   °
° AL:    Statusbyte             °
Û-------------------------------ì
```

MS-DOS durchsucht das Inhaltsverzeichnis nach dem ersten Dateinamen. Wird dieser gefunden, prüft DOS, ob eine Datei mit dem zweiten Dateinamen vorkommt. Ist dies nicht der Fall, wird die Datei auf den zweiten Namen umbenannt. Bei Verwendung von Wildcardzeichen im ersten Filenamen wird die Rename-Operation auf jede Datei angewandt, auf die das Suchmuster zutrifft.

Im AL-Register wird dann der Wert 00 zurückgegeben. Wurden im zweiten Dateinamen Wildcards (*) angegeben, bleiben die zugehörigen Zeichen im Dateinamen erhalten. Mit dieser Funktion lassen sich nur normale DOS Dateien umbenennen, d.h. der Versuch, einen Hidden-, System- oder Volumenamen umzubenennen, wird abgewiesen. Die Funktion 17H ignoriert allerdings ein gesetztes Read-only-Attribut und benennt die Datei auf den neuen Namen um. Die Fehlercodes im AL-Register besitzen folgende Bedeutung:

```
 Code °  Bedeutung
------é-----------------------------------------------------------
   0  °  Funktion ohne Fehler ausgeführt
  FF  °  Datei nicht gefunden, oder Dateiname existiert bereits
```

Wurden Wildcardzeichen verwendet, bedeutet AL=00H bei der Rückkehr, daß mindestens eine Datei umbenannt wurde. Bei Fehlern (AL = FFH) können verschiedene Ursachen vorliegen, die sich ab DOS 3.0 mit der INT 21-Funktion 59H abfragen lassen.

Um die Datei umzubenennen, wird ein modifizierter FCB benutzt. Die Laufwerksangabe steht wie üblich ab Offset 0H. Der erste Dateiname findet sich bei Offset 01H, und der zweite Name wird ab Ofset 11H gespeichert (s. oben).

Bei Zugriffen auf Dateien innerhalb eines Netzwerkes muß die Funktion das Create-Access-Privileg besitzen.

Der Aufruf 17H wurde durch die Funktion 56H (Rename a File) abgelöst und sollte deshalb bei Neuentwicklungen nicht mehr zum Einsatz gelangen.

Die nicht dokumentierte Funktion 18H ist intern für DOS reserviert, ist aber nicht belegt.

4.25 Get Current Drive (Funktion 19H, DOS 2.0-6.x)

Der Aufruf dieser Funktion gibt die Nummer des aktuell eingestellten Laufwerkes zurück.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    19H                    º
û-------------------------------À
º      RETURN                   º
º AL:    Laufwerksnummer        º
Û-------------------------------ì
```

Im Register AL findet sich die Nummer des Default-Laufwerkes mit der Kodierung A: = 0, B: = 1, etc.

Die Funktion bildet das Gegenstück zum Aufruf 0EH des INT 21, der das Laufwerk setzt. Hier weicht die Laufwerksnotation (0 = A:, 1 = B:, etc.) ebenfalls von den DOS-Konventionen ab.

4.26 Set Disk Transfer Address (Funktion 1AH, DOS 2.0-6.x)

Die CP/M kompatiblen Funktionen zur Datei-Ein-/Ausgabe verwenden einen Bereich als Zwischenpuffer, der nachfolgend als Disk-Transfer-Area (DTA) bezeichnet wird. Standardmäßig reserviert DOS beim Programmstart eine 128 Byte große DTA im PSP ab Offset 80H. Die Funktion 1AH teilt MS-DOS die Lage dieses Datenbereiches (DTA) mit.

```
Ö-------------------------------Ì
º      CALL:   INT 21           º
º                               º
º AH:    1AH                    º
º DS:DX  Zeiger auf die DTA     º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Der DTA-Bereich erstreckt sich von der angegebenen Adresse (DS:DX) bis zum Ende des Datensegments (DS:FFFFH). Die DTA darf allerdings nicht über die Segmentgrenze hinausreichen. Wie groß der genutzte Bereich der DTA wirklich ist, wird mit der Recordgröße im FCB definiert.

Ein Überlauf auf das nachfolgende Segment ist genauso ausgeschlossen, wie ein Überlauf auf den Anfang des aktuellen Datensegments. Falls die Lage des DTA nicht explizit angegeben wird, benutzt DOS den Bereich im PSP ab 80H als DTA. Die Lage des DTA läßt sich mit der Funktion 2FH ermitteln. Es ist aber zu beachten, daß DOS intern immer nur die Lage einer DTA speichert. Probleme gibt es bei (residenten) Programmen, die auf eine fixe Lage des DTA eingestellt sind und abstürzen, wenn sich die DTA-Adresse ändert. Dies ist zum Beispiel nach dem Start eines Subprozesses über die EXEC-Funktion der Fall.

Um die diversen Probleme im Zusammenhang mit den FCB's und der DTA zu umgehen, empfiehlt sich ein Verzicht auf die CP/M orientierten Filefunktionen.

4.27 Get Default Drive Data (Funktion 1BH, DOS 2.0-6.x)

Mit dieser Funktion lassen sich verschiedene Informationen über das aktuelle Laufwerk, b.z.w die File-Allocation-Table (FAT) abfragen.

```
Ö-------------------------------Ì
°      CALL:   INT 21           °
°                               °
° AH:    1BH                    °
û-------------------------------À
°      RETURN                   °
° AL: Sektoren per Cluster      °
° CX: Bytes per Sektor          °
° DX: Cluster per Drive         °
° DS:BX Pointer auf FAT ID Byte °
Û-------------------------------ì
```

Die Daten über das aktuell selektierte Laufwerk werden in den Registern AL, CX, DX und DS:BX zurückgegeben. Im Register AL findet sich die Größe eines Clusters. Bei diesen Clustern handelt es sich um eine Speichereinheit, in die eine Datei aufgeteilt wird (siehe DOS-Dateiverwaltung).

Damit kann die Datei nie kleiner als dieser Wert sein. Die belegte Speichergröße auf einer Platte umfaßt dann immer ein Vielfaches dieser Clustergröße. Sollen z.B. 10 Byte bei einer Clustergröße von 2 Sektoren a 512 Byte abgespeichert werden, belegt die Datei insgesamt 1 Kbyte auf dem Speichermedium. Die Größe eines Sektors in Byte findet sich im Register CX. Standardmäßig legt MS-DOS diese mit 512 Byte pro Sektor fest. Weiterhin enthält das Register DX die Zahl der Cluster auf dem Speichermedium. Im Registerpaar DS:BX findet sich ein Zeiger auf einen Datenbereich, der die File-Allocation-Tabelle beinhaltet und dessen erstes Byte den Disk Typ identifiziert. Für dieses Byte gilt folgende Kodierung:

```
Wert °    Diskettenformat
-----é-------------------------------------------
FF   ° Double sided Diskette, 8 Sektoren pro Spur
     ° 40 Spuren, (320 Kbyte,  5 1/4")
FE   ° Single sided Diskette, 8 Sektoren pro Spur
     ° 40 Spuren, (160 Kbyte,  5 1/4")
FD   ° Double sided Diskette, 9 Sektoren pro Spur
     ° 40 Spuren, (360 Kbyte,  5 1/4")
FC   ° Single sided Diskette, 9 Sektoren pro Spur
     ° 40 Spuren, (180 Kbyte,  5 1/4")
FB   ° Double sided Diskette, 8 Sektoren pro Spur
     ° 80 Spuren,  (640 Kbyte, 3 1/2")
FA   ° Single sided Diskette, 8 Sektoren pro Spur
     ° 80 Spuren, (330 Kbyte,  3 1/2")
F9   ° Double sided Diskette, 15 Sektoren pro Spur
     ° 40 Spuren, (600 Kbyte,  5 1/4")
     ° Double sided Diskette, 9 Sektoren pro Spur
     ° 80 Spuren, (720 Kbyte,  3 1/2")
F8   ° Fixed Disk (mit nicht näher bestimmter
     ° Formatierung)
F0   ° andere Medien
     ° Double sided Diskette, 15 Sektoren pro Spur
     ° 80 Spuren, (1,2 Mbyte,  5 1/4")
     ° Double sided Diskette, 18 Sektoren pro Spur
     ° 80 Spuren, (1,44 Mbyte,  3 1/2")
     ° Double sided Diskette, 36 Sektoren pro Spur
     ° 80 Spuren, (2,88 Mbyte,  3 1/2")
```

Tabelle 4.4: Kodierung des Media-Byte

Ab DOS 5.0 kommt noch der Code für 2,8 Mbyte Disketten hinzu.

Um die Daten eines speziellen Laufwerkes abzufragen, existiert die Funktion 1CH. Weiterhin liefert die Funktion 36H (Get Disk Free Space) ähnliche Informationen.

4.28 Get Drive Data (Funktion 1CH, DOS 2.0-6.x)

Mit dieser Funktion lassen sich analog der Funktion 1BH verschiedene Informationen über ein spezielles Laufwerk abfragen.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    1CH                    º
º DL:    Laufwerksnummer        º
û-------------------------------À
º      RETURN                   º
º AL: Sektoren per Cluster      º
º CX: Bytes per Sektor          º
º DX: Cluster per Drive         º
º DS:BX Pointer auf FAT ID Byte º
Û-------------------------------ì
```

Die Daten über das Laufwerk werden in den Registern AL, CX, DX und DS:BX zurückgegeben. Falls im Register AL der Wert FFH steht, war die angegebene Laufwerksnummer ungültig.

Diese wird vor dem Aufruf im Register DL eingestellt, wobei gilt:

```
 Code º Laufwerk
------é-------------------------
   0  º Default Drive
   1  º Drive A:
   2  º Drive B:
    .
```

Die Kodierung der zurückgegebenen Daten (FAT-ID-Byte) wurde bereits im Rahmen der Funktion 1BH besprochen. Im Registerpaar DS:BX findet sich der Zeiger auf den Datenbereich, der die File-Allocation-Tabelle beinhaltet und den Disk-Typ identifiziert. Es gilt die Kodierung aus Tabelle 4.4.

Bei Fehlern (z.B. wenn eine Diskette im abgefragten Laufwerk fehlt), löst die Funktion einen INT 24 aus, der gegebenenfalls durch das Programm abzufangen ist. Wird ein ungültiger Laufwerkscode in AL übergeben, enthält AL nach dem Aufruf den Wert FFH. Dann lassen sich ab DOS 3.0 die erweiterten Fehlercodes mit der INT 21-Funktion 59H ermitteln.

Um die Daten des Default-Laufwerkes abzufragen, existiert die Funktion 1BH. Ähnliche Daten liefert auch die Funktion 36H (Get Disk free Space).

Die nicht dokumentierten Funktionen 1CH und 1EH sind für DOS reserviert, aber nicht implementiert.

4.29 Get Default Disk Parameter Block (Funktion 1FH, DOS 2.0 - 5.0)

Hierbei handelt es sich um eine Funktion, die intern durch DOS benutzt wird und deren Funktion erst ab DOS 5.0 offiziell durch Microsoft dokumentiert wurde. Mit dieser Funktion läßt sich der Disk-Parameter-Block (DPB) des Default-Laufwerkes lesen. Es gelten folgende Aufrufparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    1FH                    º
û-------------------------------À
º      RETURN                   º
º AL: 0 kein Fehler             º
º DS: Segmentadresse DPB        º
º BX: Offsetadresse DPB         º
Û-------------------------------ì
```

Bei dem Disk-Parameter-Block (DPB) handelt es sich um eine durch MS-DOS verwaltete Datenstruktur, die Informationen über die Einheitennummer, den Aufbau des Speichermediums etc. enthält. In DOS verwalten die beim Start installierten Blocktreiber für jedes logische Laufwerk einen solchen DPB. Die Funktion liest beim Aufruf dann den Inhalt des Bootsektors des Mediums vom angegebenen Laufwerk. Im Registerpaar DS:BX wird anschließend ein Adreßvektor auf diesen Speicherbereich zurückgegeben.

Falls das Register nach dem Aufruf den Wert *0* erhält, dann ist dieser Vektor gültig. Die Datenstruktur variiert etwas mit der DOS-Version und besitzt folgenden Aufbau:

Offset	Bytes	Bemerkung
00	1	Drive-Nummer (Nummer des Treibers)
01	1	Einheiten-Nummer (Subunit im DPB)
02	2	Bytes pro Sektor
04	1	Sektors pro Cluster - 1
05	1	Cluster to Sector-Shift (2**n)
06	2	Zahl der reservierten Boot-Sektoren
08	1	Zahl der FATs
09	2	Zahl der Einträge im Hauptverzeichnis
0B	2	Sektornummer des ersten Datenclusters
0D	2	Clusterzahl + 1
		DOS 2.x
0F	1	Zahl der Sektoren pro FAT
10	2	Start-Sektor der Root-Directory
12	4	Zeiger auf den Device-Header
16	1	Media Descriptor Byte
17	1	Zugriffsbyte (FFH = DPB neu aufbauen)
18	4	Zeiger auf den folgenden DPB (FFFFH = letzter Block)
1C	2	Start Cluster aktives Verzeichnis
		0 = Stamverzeichnis
1E	64	ASCIIZ-String mit dem Namen des aktiven Verzeichnisses
		DOS 3.x
0F	1	Zahl der Sektoren für die FATs
10	2	Start-Sektor der Root-Directory
12	4	Zeiger auf den Device-Header
16	1	Media Descriptor Byte
17	1	Zugriffsbyte (FFH = DPB neu aufbauen)
18	4	Zeiger auf den folgenden DPB (FFFFH = letzter Block)
1C	2	Cluster bei dem die Suche nach freien Blocks beginnt
1E	2	freie Cluster (FFFF = unbekannt)
		ab DOS 4.0 - 6.0
0F	2	Zahl der Sektoren für die FATs wird als Word gespeichert
11	2	Start-Sektor der Root-Directory
13	4	Zeiger auf den Device-Header
17	1	Media Descriptor Byte
18	1	Zugriffsbyte (FFH = DPB neu aufbauen)
19	4	Zeiger auf den folgenden DPB (xxxx:FFFFH = letzter Block)
1D	2	Start Cluster (free space Suche)
1F	2	Anzahl der freien Cluster FFFFH = unbekannt

Tabelle 4.5: Aufbau des Disk-Parameter-Block;s (DPB)

DOS legt die DPB der einzelnen Laufwerke hintereinander im Speicher ab. Mittels der Funktion 32H lassen sich die Daten eines jeden Laufwerkes abfragen. Bei Disketten müssen die Daten beim Aufruf erst ermittelt werden. Aus diesem Grunde bringt DOS einen INT 24-Fehler, falls keine Diskette im Laufwerk eingelegt ist. Bei Festplatten legt DOS den DPB bereits beim Bootvorgang an.

Im ersten Byte wird der Laufwerkscode (0 = A:, 1 = B:, etc.) zurückgegeben. Das folgende Feld enthält den *Unit Code*, der dem Treiber bei der Installation zugeordnet wird (siehe Kapitel über die Einheitentreiber). Dieser Code ist relevant, falls der Treiber mehr als ein logisches Laufwerk unterstützt. Das dritte Feld enthält die Zahl der Bytes pro Sektor, die in DOS meist auf 200H (512) gesetzt wird. Ab Offset 04H steht die Zahl der Sektoren pro

Cluster, wobei die Zahl 0 einem Sektor pro Cluster entspricht. Bei 4 Sektoren pro Cluster findet sich also der Wert 03H.

Das Feld *Cluster to Sector-Shift* beschreibt wieviele Sektoren ein Cluster besitzt. Der Wert x ist als 2**x zu interpretieren, d.h. die Zahl 0 bedeutet 1 Sektor pro Cluster, die Zahl 1 bedeutet 2 Sektoren pro Cluster usw.

Offset 06H gibt die Zahl der Boot-Records an, d.h. der Wert entspricht dem Startcluster der ersten FAT. Darauf folgt das Feld mit der Zahl der FATs (Standard = 2). Die Zahl der möglichen Dateieinträge im Hauptverzeichnis findet sich ab Offset 09H.

Dann kommen die Felder mit den Nummern des ersten und letzten belegten Datenclusters. Ab Offset 0FH ist die Zahl der Sektoren für die FAT gespeichert. Hier beginnt der versionsspezifische Teil der Datenstruktur. Bis DOS 3.3 wurde die Zahl der Sektoren pro FAT als Byte gespeichert. Ab DOS 4.0 umfaßt dieser Wert dann 2 Byte, wodurch sich die restlichen Einträge um ein Byte verschieben. Der Startsektor des Hauptinhaltsverzeichnisses schließt sich daran an. Als nächstes findet sich der Adreßzeiger auf den Kopf des Gerätetreibers. Das Media-Descriptor-Byte (Offset 16H/17H ab DOS 4.0) beschreibt grob den Speichertyp. Ein Flag ab Offset 17H (18H bie DOS 4.0) dient zur Markierung des Diskzugriffes, welcher nach Beendigung der Operation gelöscht wird. Ein Wert von FFH signalisiert, daß der DPB neu aufzubauen ist (z.B. nach dem Wechsel der Diskette). Zum Abschluß findet sich ein Adreßzeiger (Segment:Offset) auf den DPB des nächsten Laufwerkes. Ein Wert XXXX:FFFFH zeigt das Ende der DPB-Kette an. Das folgende Wort ist abhängig von der DOS-Version unterschiedlich belegt. In DOS 2.X steht hier die Nummer des Start-Clusters des aktuellen Unterverzeichnisses. Der Wert 0 signalisiert, daß das Hauptverzeichnis selektiert ist. In DOS 2.X beginnt die Suche nach freien Clustern grundsätzlich am Anfang der FAT. Um die Fraktionierung möglichst gering zu halten, wurde ab DOS 3.0 eine Optimierung eingebaut. Nur beim Systemstart beginnt die Suche nach freien Blöcken am Beginn der FAT. Anschließend merkt sich DOS die Lage des letzten beschriebenen Clusters, und setzt bei den folgenden Aufrufen mit der Suche an diesem Punkt wieder auf. Ab DOS 3.0 findet sich deshalb ab Offset 1CH (1DH bei DOS 4.0) ein Wort mit der Nummer des Start-Clusters, ab dem die Suche nach freien Clustern beginnt.

Die Datenstruktur wird in DOS 2.X mit dem Namen des aktuellen Verzeichnisses (64 Byte ASCIIZ-String) beendet. Ab DOS 3.0 entfällt dieser Namen und es folgt ein Wort mit der Zahl der freien Cluster des Mediums. Ein Wert von FFFFH steht für den Zustand »unbekannt«»Die nicht dokumentierte Funktion 20H ist für interne Zwecke von DOS reserviert, aber nicht implementiert.

4.30 Random Read (Funktion 21H, DOS 2.0-6.x)

Diese Funktion liest einen Satz (Record) aus der spezifizierten Datei. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    21H                   º
º DS:DX  Zeiger auf einen FCB  º
û------------------------------À
º      RETURN                  º
º AL:    Statusbyte            º
Û------------------------------ì
```

Die im FCB angegebene Datei muß bereits eröffnet sein. Der zu lesende Satz wird durch den FCB-Eintrag *Relativ Record Field* (Offset 21H) spezifiziert. Der adressierte Satz wird gelesen und in den Disk-Transfer-Bereich (DTA) übertragen.

Die Zeiger im FCB für *Current Block Field* (Offset 0CH) und *Current Record Field* (Offset 20H) werden so gesetzt, daß sie mit dem durch die Variable *Relativ Record Field* (Offset 21H) adressierten Satz übereinstimmen. Die Satzlänge wird aus dem *Record Size Field* (Offset 0EH) des FCB gelesen. Im Register AL wird der Status der Leseoperation zurückgegeben. Es gilt folgende Belegung:

```
 Code º Bedeutung
------é-------------------------------------------------
   0  º  Funktion ohne Fehler ausgeführt
   1  º  Versuch über das Dateiende hinaus zu lesen,
      º  keine Daten gelesen (EOF)
   2  º  Nicht genügend Platz in der DTA, der Lese-
      º  befehl wurde abgebrochen
   3  º  Dateiende wurde erreicht, ein Teilsatz wurde
      º  gelesen, die restlichen Bytes in der DTA wurden
      º  mit 00 aufgefüllt
```

Wird die Operation auf einer Datei innerhalb eines Netzwerkes ausgeführt, muß der Prozeß das Read-Access-Privileg besitzen, andernfalls wird der Lesezugriff abgewiesen.

Für Programmneuentwicklungen sollte die Funktion 3FH (Read from a File or Device) verwendet werden, da die Funktion 21H zu den älteren CP/M-orientierten I/O-Aufrufen zählt.

4.31 Random Write (Funktion 22H, DOS 2.0-6.x)

Diese Funktion schreibt einen Satz (Record) in die spezifizierte Datei. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    22H                   º
º DS:DX  Zeiger auf einen FCB  º
û------------------------------À
º      RETURN                  º
º AL:    Statusbyte            º
Û------------------------------ì
```

Die im *FCB* angegebene Datei muß bereits eröffnet sein. Der zu schreibende Satz wird durch den FCB-Eintrag *Relativ Record Field* (Offset 21H) spezifiziert. Der Satz wird dann aus der DTA gelesen und im adressierten Dateibereich abgelegt.

Die Länge eines Satzes ist im *Record Size Field* des FCB spezifiziert. Falls diese Länge kleiner als die Recordlänge des Speichermediums ist, legt die Funktion die Daten in einem MS-DOS Zwischenpuffer ab. Erst wenn dieser Puffer gefüllt ist, oder falls die Datei geschlossen wird, oder falls das Disk-System zurückgesetzt wird (Funktion 0DH), dann

werden die Daten auf die Platte geschrieben. Im Register AL wird der Status der Schreiboperation zurückgegeben. Es gilt folgende Belegung:

```
Code ° Bedeutung
-----é-------------------------------------------------
  0  ° Funktion ohne Fehler ausgeführt
  1  ° Disk voll, Schreibversuch abgebrochen
  2  ° Nicht genügend Platz in der DTA, der Schreib-
     ° befehl wurde abgebrochen
```

Die Einträge im FCB *Current Block* (Offset 0CH) und *Current Record* (Offset 20H) werden so justiert, daß sie auf den gleichen Satz zeigen, in den gerade geschrieben wurde. Die Positionierung des Ausgabezeigers ist allerdings problematisch. Zeigt er auf Daten innerhalb der Datei, werden die folgenden Datenbytes überschrieben. Zeigt er hinter das Dateiende, füllt DOS den Zwischenbereich mit undefinierten Daten auf. Falls die Datei mit dem Attribut *Read Only* versehen ist, wird ein Schreibzugriff abgewiesen.

Bei Zugriff auf Dateien innerhalb eines Netzwerkes (ab DOS 3.1), benötigt der Prozeß das Write Access-Privileg.

Da es sich bei der Funktion um eine ältere CP/M-orientierte I/O-Anweisung mit allen ihren Problemen handelt, sollte bei Neuentwicklungen von Programmen der Aufruf 40H (Write to a File or Device) verwendet werden.

4.32 Get File Size (Funktion 23H, DOS 2.0-6.x)

Diese Funktion ermittelt die Größe einer angegebenen Datei und gibt diese zurück. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö-------------------------------Ì
°      CALL:  INT 21            °
°                               °
° AH:    23H                    °
° DS:DX  Zeiger auf einen FCB   °
û-------------------------------À
°      RETURN                   °
° AL:    Statusbyte             °
Û-------------------------------ì
```

Der Zeiger DS:DX muß auf einen ungeöffneten FCB zeigen. In diesem FCB müssen vor dem Aufruf folgende Felder definiert worden sein: Im Laufwerksbyte muß ein gültiger Wert für das Laufwerk stehen. Der Filename muß als maximal 8 Byte langer ASCII-String eingetragen werden.

Fehlende Zeichen sind mit 00H aufzufüllen. Im Namen sind Wildcardzeichen (? ab DOS 2.x, * ab DOS 3.X) erlaubt. Das Feld mit der Dateiextension muß belegt, oder mit 00 00 00 gefüllt werden. Im Feld *record size* (Offset 0EH) ist vor dem Aufruf die Größe eines Satzes (1..FFFFH Byte) korrekt einzusetzen.

Wurde eine Datei gefunden, gibt die Funktion im Feld »relativ recordn:

```
Code ° Bedeutung
-----é-----------------------------------
 00  ° Operation fehlerfrei durchgeführt
 FF  ° keine Datei gefunden
```

Der Wert des Feldes *Record Size* muß vor dem Aufruf auf einen definierten Wert gesetzt werden, sonst ist das Ergebnis nicht gültig. Der Wert kann zum Beispiel durch den Aufruf

Open File initialisiert werden. Wegen der Probleme mit den FCB orientierten Filezugriffen sollten die handleorientierten INT 21-Funktionen für Neuentwicklungen verwendet werden. Zur Ermittlung der Dateilänge eignet sich die INT 21-Funktion 42H.

4.33 Set Relativ Record (Funktion 24H, DOS 2.0-6.x)

Mit dieser Funktion läßt sich das Feld *Relativ Record Field* des FCB setzen.

```
Ö------------------------------Ì
°      CALL:  INT 21           °
°                              °
° AH:    24H                   °
° DS:DX  Zeiger auf einen FCB  °
û------------------------------À
°      RETURN                  °
° ---                          °
Û------------------------------ì
```

Durch Aufruf der Funktion wird das *Relativ Record Field* (Offset 21H) so justiert, daß es den gleichen Record adressiert, der durch die Werte der Felder *Current Block Field* (Offset 0CH) und *Current Record Field* (Offset 20H) referiert wird.

Der Zeiger DS:DX muß auf einen ungeöffneten FCB zeigen. Die Funktion sollte vor der Benutzung der Random-Read- und Random-Write-Funktionen (Aufrufe 21H, 22H, 27H und 28H) benutzt werden. Die Funktion gibt keine Status- oder Fehlermeldung nach dem Aufruf zurück.

4.34 Set Interrupt Vektor (Funktion 25H, DOS 2.0-6.x)

Der direkte Zugriff aus Anwenderprogrammen auf die Interruptvektor-Tabelle ist nicht zulässig. Tritt zum Beispiel während des Setzens eines neuen Vektors ein Interrupt auf, der diesen Vektor benutzt, befindet sich das System in einem inkonsistenten Zustand. Probleme gibt es ebenfalls, wenn Software unter DOS-Emulatoren (WINDOWS, OS/2) läuft und die Vektoren direkt modifiziert. Um dies zu vermeiden, bietet das MS-DOS-Betriebssystem die Funktion 25H (Set-Interruptvektor). Es gelten folgende Übergabeparameter:

```
Ö------------------------------Ì
°      CALL:  INT 21           °
°                              °
° AH:    25H                   °
° AL:    Interruptnummer       °
° DS:DX  Interruptvektor       °
û------------------------------À
°      RETURN                  °
° ---                          °
Û------------------------------ì
```

Beim Aufruf wird im Register AL die Nummer des gewünschten Interrupts angegeben. Die Funktion lädt dann die im Registerpaar DS:DX befindliche Adresse in die Interruptvektor-Tabelle. Es gilt dabei folgende Zuordnung:

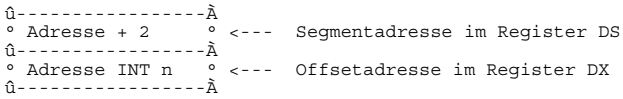


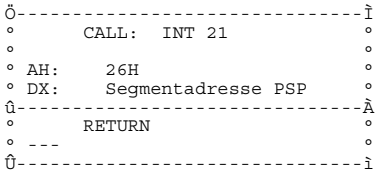
Bild 4.2: Anordnung eines Interruptvektors

Die Funktion gibt keine Status- oder Fehlermeldungen zurück. Um den aktuellen Wert des Interruptvektors zu sichern, läßt sich die Funktion 35H (Get Interrupt Vektor) benutzen.

Die PharLap 386-DOS-Extender nutzen die Funktion 25H des INT 21 ebenfalls als Eintrittspunkt für Systemaufrufe.

4.35 Create New PSP (Funktion 26H, DOS 2.0-6.x)

Mit dieser Funktion läßt sich ein neues Program-Segment-Prefix (PSP) erzeugen. Es sind folgende Parameter zu übergeben:



Der Wert in DX spezifiziert die Segmentadresse, wo der neue PSP angelegt wird. Der PSP-Bereich besitzt eine Größe von 256 Byte (0FFH) und befindet sich ab der Segmentadresse 0 im Codesegment. Ab Offset 100H beginnt dann der Programmcode.

Beim Erstellen des PSP-Segments kopiert DOS die Werte der Interruptvektoren 22H (Programm Exit Address), 23H (Control-C Exit Address) und 24H (Critical Error Exit Address) in diesen Bereich. Dadurch ist es nach Abbruch des laufenden Prozesses möglich, diese Vektoren zu restaurieren, auch wenn diese vorher durch den Prozeß variiert wurden. In DOS 2.x nimmt das Betriebssystem an, daß in CS die Segmentadresse des PSP steht. Bei EXE-Files kann es hier zu Problemen kommen.

Der Systemaufruf sollte bei Programmneuentwicklungen nicht mehr verwendet werden, da die Funktion 4B00H (Load- and -Execute-Programm) automatisch ein PSP anlegt.

4.36 Random Block Read; (Funktion 27H, DOS 2.0-6.x)

Diese Funktion liest einen oder mehrere Sätze (Records) aus der spezifizierten Datei. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    27H                   º
º DS:DX  Zeiger auf einen FCB  º
º CX:    Zahl der Records die  º
º        zu lesen sind         º
û------------------------------À
º      RETURN                  º
º AL:    Statusbyte            º
º CX:    Zahl der gelesenen    º
º        Records               º
Û------------------------------ì
```

Die im FCB angegebene Datei muß bereits eröffnet sein. Der zu lesende Satz wird durch den FCB-Eintrag *Relativ Record Field* (Offset 21H) spezifiziert. Dieser Wert muß vorher durch Aufruf der Funktion 24H gesetzt werden. Die Zahl der zu lesenden Sätze wird in CX angegeben. Die Recordlänge wird durch DOS auf 128 Byte gesetzt, kann aber explizit im FCB zwischen 1 und FFFFH variiert werden.

MS-DOS ermittelt die Zahl der zu lesenden Bytes durch Multiplikation der Recordzahl * Recordlänge. Nach der Operation enthält das Register CX die Zahl der wirklich gelesenen Sätze. Die Einträge im FCB *Current Block Field*, *Current Record Field* und *Relativ Record Field* werden auf den nächst folgenden Satz gesetzt. Falls der Wert in CX = 0 ist, wird kein Satz gelesen. Im Register AL erscheint die Statusinformation:

```
Code º Bedeutung
-----é-------------------------------------------------
  0  º Funktion ohne Fehler ausgeführt
  1  º Versuch über das Dateiende hinaus zu lesen,
     º keine Daten gelesen (EOF)
  2  º Nicht genügend Platz in der DTA, der Lese-
     º befehl wurde abgebrochen
  3  º Dateiende wurde erreicht, ein Teilsatz wurde
     º gelesen, die restlichen Bytes in der DTA wurden
     º mit 00 aufgefüllt
```

Beim Zugriff auf eine Datei innerhalb eines Netzwerkes muß der Prozeß das Read-Access-Privileg besitzen. Die Funktion gehört zur Gruppe der älteren CP/M-orientierten Aufrufe und sollte bei Neuentwicklungen nicht weiter verwendet werden.

4.37 Random Block Write (Funktion 28H, DOS 2.0-6.x)

Diese Funktion schreibt einen oder mehrere Sätze (Records) in die spezifizierte Datei. Vor dem Aufruf sind folgende Parameter zu setzen:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    28H                   º
º DS:DX  Zeiger auf einen FCB  º
º CX:    Zahl der Records die  º
º        zu schreiben sind     º
û------------------------------À
º      RETURN                  º
º AL:    Statusbyte            º
º CX:    Zahl der geschriebenen º
º        Records               º
Û------------------------------ì
```

Die im FCB angegebene Datei muß bereits eröffnet sein. Die zu schreibenden Sätze werden durch den FCB-Eintrag *Relativ Record Field* (Offset 21H) spezifiziert. Dieser Wert muß vorher durch Aufruf der Funktion 24H gesetzt werden. Die Zahl der zu lesenden Sätze

wird in CX angegeben. Die Recordlänge kann im FCB variiert werden und ist standardmäig auf 128 Byte gesetzt.

Falls der Wert in CX = 0 ist, werden keine Daten geschrieben, sondern MS-DOS setzt die Dateigröße im FAT-Bereich auf den Wert des Feldes *Relativ Record Field* im FCB. Es werden dann die entsprechenden Cluster, je nach Dateigröße, belegt oder freigegeben. Bei Werten größer 0 werden entsprechend viele Sätze geschrieben.

MS-DOS ermittelt die Zahl der zu schreibenden Bytes durch Multiplikation der Recordzahl * Recordlänge. Nach der Operation enthält das Register CX den Wert der wirklich geschriebenen Sätze. Die Einträge *Current Block Field, Current Record Field* und *Relativ Record Field* im FCB werden auf den nächst folgenden Satz gesetzt. Falls der Wert in CX = 0 ist, wird kein Satz geschrieben. Im Register AL erscheint die Statusinformation:

```
Code º Bedeutung
-----é-------------------------------------------------
  0  º Funktion ohne Fehler ausgeführt
  1  º Disk voll, keine Sätze geschrieben
  2  º Nicht genügend Platz in der DTA, der Schreib-
     º befehl wurde abgebrochen
```

Falls nicht genügend Speicherplatz auf dem Medium vorhanden ist, enthält das AL-Register den Statuscode 01H. In diesem Fall werden keine Daten auf die Einheit geschrieben. Wird der Schreibzeiger hinter das Dateiende positioniert, füllt DOS die Lücke zwischen EOF und der aktuellen Position mit undefinierten Zeichen.

Bei Verwendung innerhalb eines Netzwerkes (ab DOS 3.1) muß der Prozeß das Zugriffsrecht *Write Access* besitzen. Die Funktion sollte bei Programmneuentwicklungen nicht weiter verwendet werden.

4.38 Parse File Name (Funktion 29H, DOS 2.0-6.x)

Diese Funktion analysiert Zeichenketten (Strings) auf gültige Filenamen. Die Strings müssen das Format:`Device:Filename.Extension`

besitzen. Es sind folgende Übergabeparameter zu definieren:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    29H                    º
º AL:    Steuercode             º
º DS:SI  Zeiger auf den String  º
º ES:DI  Zeiger auf einen Puffer º
º        für den FCB            º
û-------------------------------À
º      RETURN                   º
º AL:    Statusbyte             º
º DS:SI  Zeiger auf das analys. º
º        Stringende             º
º ES:DI  Zeiger auf den FCB     º
Û-------------------------------ì
```

Im Registerpaar DS:SI wird die Adresse (Segment:Offset) der zu analysierenden Zeichenkette abgelegt. Das Registerpaar ES:DI enthält einen Zeiger auf einen Pufferbereich, in dem die Funktion einen ungeöffneten FCB anlegen kann. Wird ein gültiger Dateiname im String gefunden, baut die Funktion einen FCB in diesem Bereich

auf. Mit dem Wert des Registers AL wird die Funktion des Parsers gesteuert, wobei nur die untersten 4 Bit belegt sind.

```
 3 2 1 0
Ö-Ú-Ú-Ú-Ì
ÛÚÙÚÙÚÙÚÌ   0 Analyse abbrechen, falls ein Fileseparator gefunden
 º º º Û--    wird (Fileseparator = Blank, Komma, etc.)
 º º º      1 Ignoriere führende Separatoren
 º º º
 º º º      0 Setze Default Drive im FCB, falls im String kein
 º º Û---    Drive angegeben wurde
 º º        1 Drive Nummer nicht verändern, falls keine Angabe
 º º          im String
 º º
 º º       0  Ersetze den Dateinamen im FCB durch 8 Blanks, falls
 º Û-----     im String kein Dateiname vorkommt
 º          1  Dateiname im FCB nicht verändern, falls im String
 º             kein Name vorkommt
 º
 º          0  Ersetze die Extension im FCB durch 3 Blanks, falls
 Û------       im String keine Extension vorkommt
            1  Extension im FCB nicht verändern, falls im String
               keine Extension vorkommt
```

Bild 4.3: Steuercodes für die Funktion Parse File Name

Die restlichen Bits im Register AL sind vor dem Aufruf auf den Wert 0 zu setzen. Falls im String Wildcards (*) vorkommen, werden alle betroffenen Zeichen (Filename oder Extension) durch ein ? ersetzt.

Als gültige Fileseparatoren sind folgende Zeichen erlaubt:

```
: . ; , = + / " [ ] \ < > Space Tab
```

Weiterhin gelten alle Control-Zeichen als Fileseparatoren. Innerhalb eines Dateinamens dürfen diese Separatoren nicht verwendet werden, da sonst die Analyse an dieser Stelle abbricht.

Falls ein gültiger Dateiname im String gefunden wurde, setzt die Funktion folgende Registerinhalte:

```
AL    º Bemerkungen
------é-----------------------------------------------------
0     º falls Dateiname und Extension keine Wildcards
      º enthalten
1     º falls Dateiname oder Extension Wildcards enthalten
FF    º falls die Laufwerksangabe falsch ist
------é-----------------------------------------------------
DS:SI º Zeiger auf das erste Zeichen im String, welches noch
      º nicht analysiert wurde
ES:DI º Zeiger auf das erste Byte des ungeöffneten FCB
```

Falls der String keinen gültigen Laufwerksnamen enthält, wird das Register AL auf den Wert FFH gesetzt. Bei ungültigen Dateinamen zeigt ES:DI+1 auf eine Speicherzelle die ein Blank (20H) enthält.

4.39 Get Date (Funktion 2AH, DOS 2.0-6.x)

Mit dieser Funktion läßt sich das Datum des Systems abfragen. Es gelten folgende Übergabeparameter.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    2AH                    º
û-------------------------------À
º      RETURN                   º
º CX:    Jahr (1980 - 2099)     º
º DH:    Monat (1 - 12)         º
º DL:    Tag (1 - 31)           º
º AL:    Wochentag (0 - 6)      º
Û-------------------------------ì
```

Die Funktion gibt das aktuelle Datum in den Registern zurück. Die Kodierung der Wochentage erfolgt mittels des Registers AL:

```
 Code º Wochentag
------é------------
   0  º Sonntag
   1  º Montag
   2  º Dienstag
   3  º Mittwoch
   4  º Donnerstag
   5  º Freitag
   6  º Samstag
```

Das Jahr befindet sich im 16-Bit-Register CX und liegt zwischen 1980 und 2099. Ein Überlauf der Uhrzeit (24:00) bewirkt, daß das Datum um einen Tag weitergezählt wird. Monats-, Jahresüberlauf und Schaltjahre werden durch die Funktion automatisch berücksichtigt.

4.40 Set Date (Funktion 2BH, DOS 2.0-6.x)

Mit dieser Funktion läßt sich das Datum des Systems setzen. Dies gilt auch, wenn eine CMOS-Uhr im System vorhanden ist. Es sind folgende Übergabeparameter zu definieren:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    2BH                    º
º CX:    Jahr (1980 - 2099)     º
º DH:    Monat (1 - 12)         º
º DL:    Tag (1 - 31)           º
û-------------------------------À
º      RETURN                   º
º AL:  00H Datum gültig         º
º      FFH Datum ungültig       º
Û-------------------------------ì
```

Die Funktion setzt das in den Registern übergebene Datum. Falls ein ungültiger Wert übergeben wurde (z.B. Jahr 1900), enthält das Register AL anschließend den Wert FFH. In diesem Fall wird das Datum nicht verändert. Sobald der Wert 00H im Register AL zurückgegeben wird, ist das neue Datum gesetzt. Ab DOS 3.3 wird bei Systemen mit Real-Time-Clock auch die Uhr auf dem Chip gestellt, falls das BIOS dies erlaubt.

4.41 Get Time (Funktion 2CH, DOS 2.0-6.x)

Mit dieser Funktion läßt sich die Zeit des Systems lesen. Es gelten folgende Übergabeparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    2CH                    º
û-------------------------------À
º      RETURN                   º
º CH:    Stunden (0 - 23)       º
º CL:    Minuten (0 - 59)       º
º DH:    Sekunden (0 - 59)      º
º DL:    1/100 Sek. (0 - 99)    º
Û-------------------------------ì
```

Die Funktion gibt die aktuelle Zeit über die Register zurück. Das Registerpaar CX:DX enthält die Zeit in Stunden, Minuten, Sekunden und $^1/_{100}$ Sekunden. Die Stundenwerte stehen im Register CH, während der Wert im Register DL die $^1/_{100}$ Sekunden enthält. Das Format ist so organisiert, daß Anzeigen und Rechenoperationen möglichst einfach durchzuführen sind.

Achtung: Das Ergebnis der Funktion kann von der verwendeten Hardware abhängen. Falls die Uhr z.B. keine $^1/_{100}$ Sekunden hat, wird DL immer den Wert 0 enthalten.

4.42 Set Time (Funktion 2DH, DOS 2.0-6.x)

Mit dieser Funktion läßt sich die Zeit des Systems setzen. Dies gilt auch für den Fall, daß das System standardmäßig mit einer CMOS-Uhr ausgerüstet ist. Es gelten folgende Übergabeparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    2DH                    º
º CH:    Stunden (0 - 23)       º
º CL:    Minuten (0 - 59)       º
º DH:    Sekunden ( 0 - 59)     º
º DL:    1/100 Sek. (0 - 99)    º
û-------------------------------À
º      RETURN                   º
º AL:  00 Zeit  gültig          º
º      FF Zeit  ungültig        º
Û-------------------------------ì
```

Die Funktion setzt die in den Registern übergebene Zeit. Falls ein ungültiger Wert übergeben wurde (z.B. Minuten 79), enthält das Register AL anschließend den Wert FFH. In diesem Fall wird die Zeit nicht verändert. Der Wert AL = 00H zeigt an, daß ein gültiger Wert für die Zeiteinstellung übernommen wurde. Ab DOS 3.3 wird auch die Uhrzeit auf dem Chip gestellt, falls das BIOS dies zuläßt.

Achtung: Das Ergebnis der Funktion kann von der verwendeten Hardware abhängen. Falls die Hardware bestimmte Werte nicht unterstützt, sind diese Register irrelevant. So kann z.B. die Zeit in $^1/_{100}$ Sekunden nicht von jeder Hardwareuhr gestellt werden.

4.43 Set/Reset Verify Flag (Funktion 2EH, DOS 2.0-6.x)

Mit dieser Funktion läßt sich das MS-DOS *Verify-Flag* setzen oder löschen. Es gelten folgende Übergabeparameter:

```
 Verify-Flag setzen

 Verify-Flag löschen
```

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    2EH                    º
º AL:    0 = No Verify          º
º        1 = Verify             º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Bei Schreiboperationen in Dateien besteht die Möglichkeit, daß das Ergebnis dieser Operation verifiziert wird. Dies kann z.B. in DOS durch den Befehl VERIFY = ON erzwungen werden. Mit der Funktion 2EH läßt sich dies ebenfalls erreichen. Falls AL = 0 ist, wird der Verify-Befehl abgeschaltet. In DOS 2.x sollte DL=0 gesetzt werden.

Bei AL = 1 verifiziert DOS jeden geschriebenen Satz innerhalb einer Diskettendatei.

Das Flag wird normalerweise durch DOS ausgeschaltet, da die Operation, insbesondere bei Disketten, lange dauert. Der Zustand des Verify-Flags läßt sich durch die Funktion 54H abfragen. Bei Zugriffen auf Dateien innerhalb eines Netzwerkes wird die Verify-Operation nicht unterstützt.

4.44 Get Disk Transfer Address (Funktion 2FH, DOS 2.0-6.x)

Die Lage der Disk-Transfer-Area (DTA) läßt sich mit dieser Funktion ermitteln. Der DTA dient als Puffer für die Disk-Ein-/Ausgabedaten.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    2FH                    º
û-------------------------------À
º      RETURN                   º
º ES:BX  Zeiger auf die DTA     º
Û-------------------------------ì
```

Im Registerpaar ES:DX wird dann ein Zeiger (Segment:Offset) mit der DTA-Adresse zurückgegeben. In der Standardeinstellung benutzt DOS den Bereich ab Offset 80H im PSP als DTA, der demnach auf 128 Byte begrenzt ist.

Wurde die DTA in einen eigenen Speicherbereich umgesetzt, läßt sich die betreffende Adresse mit der Funktion abfangen.

4.45 Get MS-DOS Version Number (Funktion 30H, DOS 2.0-6.x)

Die Funktion gibt die Versionsnummer des aktuell geladenen Systems zurück. Bei Aufruf sollte das Register AL auf den Wert 0 gesetzt werden.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    30H                    º
º AL:    00H                    º
û-------------------------------À
º      RETURN                   º
º AL     Haupt - Versionsnummer º
º AH     Unter - Versionsnummer º
º BH     OEM - Seriennummer     º
ºBL:CX   24 Bit User Seriennr.  º
Û-------------------------------ì
```

Im AX-Register wird dann die Versionsnummer zurückgegeben, wobei die Hauptnummer in AL steht (Beispiel: Version 3.1 -> AL = 3, AH = 10). Es ist zu beachten, daß die Werte als Hexzahlen zurückgegeben werden. Bei DOS 3.1 enthält das Register die Werte AL = 03H, AH = 0AH. Bei DOS 3.2 werden die Werte AL = 03H und AH = 14H zurückgegeben.

Falls die Versionsnummer kleiner als 2.0 ist, enthält AL den Wert 0. Die restlichen Register sind für die Seriennummern des Herstellers reserviert (FFH für MS-DOS, 00H für PC-DOS). Das hierfür reservierte Registerpaar BL:CX wird aber üblicherweise immer auf 00:00 gesetzt, da bisher kein Hersteller diese vorgesehene Option nutzt.

Diese Aussagen gelten allerdings nur für die Versionen 2.x und 3.x des Betriebssystems. Ab DOS 4.0 modifizierte Microsoft die Abfrage der Version, um Kompatibilitätsprobleme mit Altsoftware in den Griff zu bekommen. Ziel ist es, einem Programm eine vom Benutzer definierbare Versionsnummer zurückzugeben. In DOS 4.x findet sich eine interne Tabelle (im DOS-Datenbereich), in dem die Namen der betreffenden (Microsoft) Programme abgelegt werden. Fragt eines dieser Programme die Versionsnummer ab, erhält es die Version 3.40 zurückgeliefert.

Ab DOS 5.0 besteht sogar die Möglichkeit, mittels des Befehls SETVER, jedem Programm eine beliebige Versionsnummer zuzuordnen. SETVER installiert sich resident als Treiber (SETVERXX) im Speicher und überwacht die Versionsabfrage. Es baut sich zu diesem Zweck eine interne Tabelle mit den Programmnamen und den Versionsnummern auf. Der Treiber SETVERXX kann nur die Funktion INIT (Code 00H) ausführen. Die Tabelle mit den Programmnamen findet sich im Treiber ab Offset 4BH. Der transiente Teil von SETVER vermag die Liste zu modifizieren. Die Original Versionsnummer kann ab DOS 5.0 über eine Erweiterung der Funktion 33H ermittelt werden.

In AL läßt sich beim Aufruf ebenfalls ein Steuercode definieren. Bei AL <> 1 wird die OEM-Seriennummer in BH zurückgegeben. Bei AL = 1 erfolgt die Abfrage des Versionsflags. Das Flag wird ebenfalls in BH zurückgegeben und signalisiert, ob eine ROM-Variante des Betriebssystems benutzt wird. Die Unterscheidung erfolgt in einem Bit von BH. Ist BH = 08H liegt die ROM-Version vor, bei BH = 00H wird eine normale DOS-Version benutzt.

4.46 Keep Process (Funktion 31H, DOS 2.0-6.x)

Die Funktion erlaubt es, Terminate-and-Stay-Resident-Programme zu erzeugen. Es gelten folgende Übergabeparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    31H                    º
º AL:    Return Code            º
º DX:    belegte Paragraphen    º
û-------------------------------À
º      RETURN                   º
º ---                           º
Û-------------------------------ì
```

Ein geladenes Programm läßt sich mit diesem Aufruf beenden, ohne daß es aus dem Speicher entfernt wird. Diese Technik wird häufig benutzt, um spezielle Interrupt-Treiber zu installieren. Diese lassen sich anschließend durch den entsprechenden Interrupt aktivieren.

Die Funktion ersetzt den INT 27 (Terminate But Stay Resident) und erlaubt auch Programme, die mehr als 64 Kbyte Speicher belegen, zu installieren.

Im AL-Register kann ein Returncode an den *Parent Process* zurückgegeben werden. Falls es sich bei diesem Prozeß um MS-DOS handelt, läßt sich der Code mit dem Befehl ERRORLEVEL lesen. Jeder andere Parent Prozeß kann den Returncode mittels der INT 21-Funktion 4DH abfragen. Im Register DX steht die Zahl der belegten Speicherparagraphen (16-Byte-Blöcke). Es ist jedoch darauf zu achten, daß bei EXE-Programmen neben dem Codebereich auch der Datenbereich in die reservierte Speichergröße eingerechnet wird. Weiterhin muß auch der Speicherbereich des PSP (10H Paragraphen) berücksichtigt werden. Minimal muß DX = 06 gesetzt werden (PSP-Init).

MS-DOS beendet das Programm und versucht die angeforderten Speicherblöcke diesem Prozeß zuzuordnen. Es werden aber keine von dem Programm reservierten Blöcke freigegeben. Falls das Programm vorher mehr Paragraphen belegt hat, müssen diese explizit freigegeben werden. Dateien, die durch diesen Prozeß bereits geöffnet wurden, bleiben auch weiterhin erhalten.

Weiterhin kann der durch das Environment belegte Speicher in vielen Fällen freigegeben werden. Hierfür existiert die Funktion 48H, der im Register ES die Segmentadresse des Environments übergeben wird. Diese Adresse findet sich ab Offset 2CH im PSP-Block.

4.47 Get Disk Parameter Block (Funktion 32H, DOS 2.0-6.x)

Bei diesem Aufruf handelt es sich um eine interne DOS-Funktion, die den Disk-Parameter-Block (DPB) liest. Der DPB ist eine interne Datenstruktur, die Informationen über das Speichermedium, die Einheit etc. enthält. Die Aufrufparameter, sowie die Struktur der Parameterblöcke wurde erst ab DOS 5.0 von Microsoft offiziell publiziert. Die bisherigen DOS-Versionen unterstützen aber diese Funktion. In der OS/2-Kompatibility-Box wird die Funktion DOS 3.3 konform abgebildet. Lediglich die Datenstruktur zeigt in der Adresse des Device Treibers einen falschen Wert. Es gelten folgende Übergabeparameter für die Funktion:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    32H                   º
º DL:    Laufwerkscode         º
û------------------------------À
º      RETURN                  º
º Carry: gesetzt -> Fehler     º
º AL:    Fehlercode            º
º Carry: gelöscht -> ok        º
º DS: Segmentadresse DPB       º
º BX: Offsetadresse DPB        º
Û------------------------------ì
```

Im Register DL wird die Laufwerksnummer (0 = Default, 1 = A:, 2 = B:, etc.) angegeben. Tritt ein Fehler auf, wird das Register AL zur Rückgabe des Fehlercodes benutzt. Ein gesetztes Carry-Flag zeigt einen Fehler an. Es gelten folgende Fehlercodes:

```
Code º Bemerkung
-----é--------------------------------
  15 º ungültige Laufwerksangabe in DL
```

Andere Fehlercodes sind den DOS-Error-Code-Tabellen zu entnehmen.

Falls der Aufruf fehlerfrei abgearbeitet wird (Carry gelöscht und AX = 0), dann findet sich im Registerpaar DS:BX ein Zeiger auf den Disk-Parameter-Block (DPB). Diese Datenstruktur besitzt folgenden Aufbau:

Offset	Bytes	Funktion
00	1	Drive-Nummer (Nummer des Treibers)
01	1	Einheiten-Nummer (Subunit im DPB)
02	2	Byte pro Sektor
04	1	Sektor pro Cluster - 1
05	1	Cluster to Sector-Shift
06	2	Zahl der reservierten Boot-Sektoren
08	1	Zahl der FATs
09	2	Zahl der Einträge im Hauptverzeichnis
0B	2	Sektornummer des ersten Daten-Clusters
0D	2	maximale Clusternummer
----	---	--- Aufbau für DOS 2.x ---
0F	1	Zahl der Sektoren für eine FAT
10	2	Start-Sektor des Root Directories
12	4	Zeiger auf den Device Header des Laufwerks
16	1	Media Descriptor Byte
17	1	Zugriffsbyte (0 = Disk Access, FFH rebuild Block)
18	4	Zeiger auf den folgenden DPB (FFFFH = last Block)
1C	2	Start Cluster current Directory (0 = Root)
1E	64	ASCIIZ-String aktueller Directory-Pfad
----	---	--- Aufbau für DOS 3.x ---
0F	1	Zahl der Sektoren für eine FAT
10	2	Start-Sektor des Root Directories
12	4	Zeiger auf den Device Header des Laufwerks
16	1	Media Descriptor Byte
17	1	Zugriffsbyte (0 = Disk Access, FFH rebuild Block)
18	4	Zeiger auf den folgenden DPB (FFFFH = last Block)
1C	2	Startcluster für Free Space Suche
1E	2	Zahl der freien Cluster (FFFFH = unbekannt)
----	---	--- Aufbau ab DOS 4.x ---
0F	2	Zahl der Sektoren für die FATs

Offset	Bytes	Funktion
11	2	Start-Sektor des Root Directories
13	4	Zeiger auf den Device Header
17	1	Media Descriptor Byte
18	1	Zugriffsbyte (0 = Disk Access, FFH rebuild Block)
19	4	Zeiger auf den folgenden DPB (FFFFH = last Block)
1D	2	letzter geschriebener Cluster
1F	2	Zahl der freien Cluster (FFFFH = unbekannt)

Tabelle 4.6: Der Aufbau des Disk-Parameter-Blocks (DPB)

Die nachfolgende Abbildung zeigt einen Speicherdump des DPB einer Festplatte.

```
-d 029C:6E20
029C:6E20  02 02 00 02 03 02 01 00-02 00 02 73 00 73 28 29
029C:6E30  53 00 02 10 70 00 F8 00-FF FF FF FF 00 00 FF FF
```

Bild 4.4: Speicherdump des DCB einer Platte

DOS legt die DPB der einzelnen Laufwerke hintereinander im Speicher ab. Mittels der Funktion 32H lassen sich die Daten eines jeden Laufwerkes abfragen. Bei Disketten müssen die Daten beim Aufruf erst ermittelt werden. Aus diesem Grunde bringt DOS einen Fehler, falls keine Diskette im Laufwerk eingelegt ist. Bei Festplatten legt DOS den DPB bereits beim Bootvorgang an. Der DPB des Default-Laufwerkes läßt sich mit der Funktion 1FH abfragen. Dort findet sich auch eine ausführliche Beschreibung des DPB.

4.48 Control-C Check (Funktion 33H, DOS 2.0-6.x)

Diese Funktion wurde im Laufe der verschiedenen DOS-Versionen schrittweise erweitert. Nachfolgend werden die verschiedenen Unterfunktionen beschrieben.

4.48.1 Control-C Check (Funktionen AX=3300H/3301H)

Die Subfunktion ist ab DOS 2.0 implementiert und besitzt folgende Aufrufsyntax:

CALL: INT 21	
AH:	33H
AL:	1. Unterfunktion
DL:	2. Unterfunktion
RETURN	
AL:	Fehlercode
DL:	Status des Flags

Mit dieser Funktion läßt sich bei allen DOS-Versionen der Zustand des internen Control-C Flags prüfen. Es gilt nebenstehende Übergabeschnittstelle.

Das Register AL spezifiziert, ob der Status gelesen oder gesetzt werden soll:

```
 Code ° Bemerkung
------é-------------------------------------------------
AL= 0 ° lese den Status des Flags in das Register DL
    1 ° setze den Status des Flags aus dem Register DL
```

Das Register DL dient zur Übergabe des gewünschten Wertes. Bei der Leseoperation (AL = 0) wird der aktuelle Wert in DL zurückgegeben, während bei der Schreiboperation der Wert des Registers den Zustand des Flags beeinflußt. Es gilt folgende Zuordnung:

```
DL ° Bedeutung
---é---------------------
0  ° Control-C Check Aus
1  ° Control-C Check Ein
```

Falls beim Aufruf das Register AL nicht die Werte 0 oder 1 enthält, terminiert die Funktion und gibt den Fehlercode FFH in AL zurück. Der Control-C Check wird normalerweise nur bei Aufrufen der INT 21-Funktionen 01H bis 0CH ausgeführt. Ob die jeweilige Funktion den Status abfragt, ist in der entsprechenden Beschreibung vermerkt. Falls der Control-C Status gesetzt ist, wird bei jedem INT 21-Funktionsaufruf auf Control-C abgeprüft.

Achtung: **Falls ein Programm die Funktionen 06H oder 07H (Direct Console I/O) benutzt und das Zeichen Ctrl-C als Wert eingelesen werden soll, muß vorher das Flag auf »Aus**Fehler! Verweisquelle konnte nicht gefunden werden.**4.48.2 Get/Set State (Funktion AX=3302H)**

Ab DOS 3.0 erlaubt die Funktion den undokumentierten Aufruf mit AL = 02H. Der Aufruf wird in der Regel nur intern durch DOS benutzt, wobei folgende Aufrufschnittstelle gilt:

```
Ö--------------------------------Ì
°      CALL:  INT 21             °
°                                °
° AH: 33H (Control-C Check)      °
° AL: 02H (Get/Set State)        °
° DL: 00H Set Break OFF          °
°     01H Set Break ON           °
û--------------------------------À
°      RETURN                    °
° AL: Status                     °
° DL: alter Wert von Break       °
Û--------------------------------ì
```

Mit diesem Aufruf wird der Zustand des Break-Flags gesetzt und gleichzeitig der alte Zustand zurückgegeben (exchange). Vor dem Aufruf ist in DL der neue Zustand für Break (0 = OFF, 1 = ON) zu setzen. Nach dem Aufruf enthält das Register DL dann den alten Zustand (0 = OFF, 1 = ON) des Break-Flags. Findet sich im Register AL der Wert FFH, trat beim Aufruf ein Fehler auf.

Ab DOS 3.0 benutzt die Funktion keinen der internen DOS-Stacks, so daß ein Aufruf aus TSR-Programmen und Interruptroutinen erlaubt ist.

4.48.3 Get Boot Drive (Funktion AX=3305H)

In DOS 4.x wurde der Funktionsumfang nochmals erweitert. Ab DOS 4.0 läßt sich mittels der INT 21-Funktion 33H auch das Bootlaufwerk abfragen. Hierzu gelten folgende Aufrufparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH: 33H (Control-C Check)     º
º AL: 05H (Get Boot Device)     º
û-------------------------------À
º      RETURN                   º
º AL: FF Fehler                 º
º AL: 00 -> ok                  º
º DL: Bootdrive                 º
Û-------------------------------ì
```

Vor dem Aufruf ist der Inhalt des Registers AL auf den Wert 05H zu setzen. Enthält AL nach dem Aufruf den Code FFH, wird die Funktion nicht unterstützt. Ist der Wert in AL = 5, wurde das Laufwerk ermittelt. Anschließend gibt die Funktion im Register DL den Code des Bootlaufwerkes zurück. Hierbei gilt:

```
DL = 1  ->  A:
     2  ->  B:
     3  ->  C:
```

4.48.4 Get Version/Location (Funktion AX=3306H)

Diese Subfunktion ist erst ab DOS 5.0 implementiert und gibt die wirkliche DOS-Versionsnummer, sowie ein Flag zurück. Hierzu gelten folgende Aufrufparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH: 33H (Get Version)         º
º AL: 05H                       º
û-------------------------------À
º      RETURN                   º
º AL: FF -> Fehler              º
º AL: 00 -> ok                  º
º BX: DOS-Version               º
º DX: Bitflag                   º
Û-------------------------------ì
```

Vor dem Aufruf ist der Inhalt des Registers AL auf den Wert 06H zu setzen. Enthält AL nach dem Aufruf den Code FFH, wird die Funktion nicht unterstützt. Ist der Wert in AL = 6, wurde die wirkliche DOS-Version ermittelt. In BH steht dann die Unterversion und in BL die Hauptversion. Der Inhalt von DX ist als Bitfeld zu interpretieren.

Hierbei gilt folgende Belegung:

```
DX Bit 12: 1 DOS im HMA-Bereich geladen
       11: 1 ROM-Version des Betriebssystems
     3-10: 0 reserviert
     0- 2:   Revisionsnummer (0 .. 7)
```

Interessant ist vor allem Bit 12, mit dem sich prüfen läßt, ob Teile von DOS im HMA-Bereich geladen wurden. Alle Aufrufe der Funktion 33H sind reentrant implementiert.

Wird bei der Funktion 33H in AL ein ungültiger Code (ungleich 0, 1, 2, 5 oder 6) übergeben, enthält AL nach dem Aufruf den Wert FFH.

4.49 Get DOS Critical Interval Flag (Funktion 34H, DOS 2.0-6.x)

Hier handelt es sich ebenfalls um eine erst ab DOS 5.0 dokumentierte Funktion, die normalerweise nur intern durch DOS benutzt wird. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    34H                    º
û-------------------------------À
º      RETURN                   º
º ES:    Segmentadresse Flag    º
º BX:    Offsetadresse  Flag    º
Û-------------------------------ì
```

Nach dem Aufruf gibt DOS die Adresse des *Critical Interval Flags* im Registerpaar ES:BX zurück. Das Flag ist gemäß folgender Notation kodiert:

```
  Flag º Bedeutung
-------é-----------------------------------------------------
    0  º Die DOS-Resourcen sind für den rufenden Prozeß frei
 >= 1  º DOS befindet sich gerade in einem kritischen Bereich
       º Der rufende Prozeß kann keine Betriebsmittel belegen
```

Was hat es nun aber mit diesem Flag auf sich? DOS wurde als Single-User, Single-Tasking-Betriebssystem entworfen. Um die Implementierung möglichst einfach zu halten, sind die Funktionsaufrufe nicht reentrant ausgeführt. Ruft nun ein aktiver Prozeß eine INT 21-Funktion oberhalb 0CH auf, legt DOS Zwischenergebnisse in einem internen Arbeitsbereich ab. Vor einer Rückkehr in den laufenden Prozeß werden diese Daten wieder restauriert. Es besteht aber durchaus die Möglichkeit, daß ein zweiter Prozeß (z.B. durch einen Interrupt gestartet) die DOS-Funktion unterbricht. Ruft dieser Prozeß ebenfalls die INT 21-Funktionen auf, wird der Inhalt des internen DOS-Arbeitsbereiches nicht auf dem Stack gesichert, sondern die Ergebnisse werden einfach überschrieben. Sobald der gerade laufende Prozeß beendet ist, kann auch der unterbrochene Prozeß wieder aktiviert werden. Leider ist der interne DOS-Arbeitsbereich zerstört, so daß die Funktion in der Regel falsche Ergebnisse liefert.

Um Programme wie PRINT von MS-DOS dennoch im Hintergrund betreiben zu können, besitzt DOS ein internes Flag, welches anzeigt, ob sich gerade eine INT 21-Funktion in einem kritischen Bereich befindet. Ist dies der Fall, darf der INT 21 nicht ein zweites Mal aufgerufen werden. Der rufende Prozeß muß so lange warten, bis das Flag den Wert 0 besitzt. Erst dann kann z.B. eine Disk-I/O-Operation über den INT 21 abgesetzt werden.

Das Byte an der Adresse ES:BX+1 wird in der DOS-Version 2.x als Critical-Error-Flag verwaltet. In DOS 3.x befindet sich dieses Flag an der Adresse ES:BX-1. Nur bei der DOS-3.0-Version von Compac befindet sich das Flag an der Adresse ES:BX-1AAH. Ab DOS 3.1 läßt sich die Lage des Critical-Error-Flag über die undokumentierte Funktion AX=5D06H ermitteln. Ein Programm sollte immer prüfen, ob Critical-Interval-Flag und Critical-Error-Flag den Wert 0 besitzen, bevor sie einen laufenden DOS-Funktionsaufruf unterbrechen. Weitere Hinweise bezüglich des Umgangs mit residenten Programmen und den Flags finden sich im Abschnitt über die TSR-Programmierung und in (1).

4.50 Get Interrupt Vektor (Funktion 35H, DOS 2.0-6.x)

Soll ein Interruptvektor gelesen werden, muß die Funktion 35H benutzt werden, um die Konsistenz sicherzustellen.

```
°      CALL:  INT 21            °
°                               °
° AH:    35H                    °
° AL:    Interrupt Nummer       °
û--------------------------------À
°      RETURN                   °
° ES:BX  Interrupt Vektor       °
Û--------------------------------ì
```

Im Register AL ist die entsprechende Interrupt-Nummer zu hinterlegen.

Der gelesene Vektor besteht aus einer Segmentadresse, die in ES zurückgegeben wird. Die Offsetadresse findet sich in BX.

Es besteht natürlich die Möglichkeit, einen Interruptvektor direkt zu lesen. Falls aber bei dieser Operation ein Interrupt auftritt, der den Vektor verändert, dann ist das Ergebnis nicht definiert. Aus diesem Grund sollen alle Operationen auf der Interrupt-Tabelle mittels der Funktionen 25H und 35H erfolgen.

4.51 Get Free Disk Space (Funktion 36H, DOS 2.0-6.x)

Diese Funktion gibt die Zahl der freien Cluster innerhalb des Speichermediums zurück. Damit läßt sich der freie Speicherbereich auf einer Diskette/Platte bestimmen. Es gelten folgende Aufrufkonventionen:

```
°      CALL:  INT 21            °
°                               °
° AH:    36H                    °
° DL:    Drive Nummer           °
û--------------------------------À
°      RETURN                   °
° AX:   Sektoren per Cluster    °
° BX:   Freie Cluster           °
° CX:   Byte pro Sektor         °
° DX:   Cluster pro Drive       °
Û--------------------------------ì
```

Im Register DL wird das Laufwerk angegeben, wobei die übliche MS-DOS-Kodierung (0 = Default Drive, 1 = A:, 2 = B:, ...) gilt. Falls das angegebene Laufwerk nicht existiert, enthält das Register AX nach dem Aufruf den Wert 0FFFFH als Fehlercode. Bei Zugriffen auf Diskettenlaufwerke löst DOS einen INT 24 aus, falls die Diskette nicht eingelegt ist. Ansonsten gibt die Funktion in den Registern AX, BX, CX und DX verschiedene Informationen über die Belegung des Speichermediums zurück.

DOS unterteilt eine Diskette/Platte in logische Speichereinheiten (Cluster), in denen die Daten dann abgelegt werden. Ein Cluster besitzt immer ein Vielfaches der Speichergröße eines Sektors. Die Zahl der Sektoren pro Cluster wird im AX-Register zurückgegeben. Im Register BX findet sich die Zahl der freien Cluster innerhalb des Speichermediums, während DX die Gesamtzahl der Cluster des Speichermediums anzeigt. In CX steht noch die Zahl der Byte pro Sektor (512 Byte Standard).

Die freie Kapazität in Byte läßt sich leicht durch Multiplikation der Registerinhalte DX * AX * CX ermitteln.

Mit dieser Funktion werden die älteren Aufrufe 1BH und 1CH ersetzt, die bei Neuentwicklungen nicht mehr benutzt werden sollten.

4.52 Get/Set Switch Character (Funktion 37H, DOS 2.0-6.x)

Die nicht dokumentierte Funktion 37H ist für den internen MS-DOS-Gebrauch reserviert. Sie dient dazu, das Switch-Zeichen zu setzen oder zu lesen. Allerdings macht die Verwendung nur unter DOS 2.x einen Sinn.

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:    37H                     º
º AL:    Subcode                 º
º DL:    Hilfsregister           º
û--------------------------------À
º      RETURN                    º
º CY:    1 = Fehler              º
º AX:    Fehlercode              º
º CY:    0 = kein Fehler         º
º DL:     Ergebnisregister       º
Û--------------------------------ì
```

Der DIR-Befehl benutzt zum Beispiel standardmäßig ein solches Zeichen (/). Mit dem Befehl DIR /W wird dann eine erweiterte Bildschirmdarstellung selektiert. In DOS 2.x ist es möglich, dieses Zeichen zu beeinflussen. So kann das /-Zeichen durch einen Bindestrich oder ein Gleichheitszeichen = ersetzt werden. Für die Funktion 37H gelten dabei folgende Aufrufparameter:

Im Register AL wird dabei ein Code zur Selektion der jeweiligen Unterfunktion übergeben. Der Aufruf erlaubt es, die Unterfunktionen 0 bis 3 aufzurufen.

AL = 00H (Get Switch Character)

Mit dieser Unterfunktion läßt sich das eingestellte Zeichen abfragen. Die Funktion gibt dann den ASCII-Code des Zeichens im Register DL zurück. Dies erfolgt jedoch nur, falls das Carry-Flag nicht gesetzt ist. Andernfalls findet sich in AX ein Fehlercode.

AL = 01H (Set Switch Character)

Mit dieser Unterfunktion läßt sich die Einstellung des Switchcharacters ändern. Das neue Zeichen wird dabei im Register DL übergeben. Nach einem fehlerfreien Aufruf enthält das Register DL dann das eingestellte Zeichen. Andernfalls findet sich im Register AX ein Fehlercode.

AL = 02H (Get Device Access Mode)

In DOS 2.x besteht die Möglichkeit, Ein-/Ausgaben von und zu Geräten über Pseudo-Dateien abzuwickeln. Dabei wird ein Gerätename in einem Unterverzeichnis \DEV eingetragen. Anschließend behandelt DOS dieses Gerät wie eine Datei, wobei aber die Zeichen an das Gerät weitergereicht werden. Eine Eingabe \DEV\LPT1 leitet dann die Zeichen für den Drucker in die Pseudo-Datei LPT1 um.

Physikalisch gelangen die Daten aber zum Drucker. Bei DOS 2.x muß vorher in CONFIG.SYS allerdings die Einstellung:

```
AVAILDEV = FALSE
```

gesetzt werden. Der Befehl wurde in DOS 3.x nicht übernommen, da es Probleme in einer Netzwerkmgebung geben kann. Allerdings blieb der INT 21-Funktionsaufruf 37H erhalten. Nach einem Aufruf mit dem Untercode AL = 02H enthält das Register DL den Status. Der Wert 00 signalisiert, daß die Geräte nur über die Pseudo-Einheit erreichbar sind. Andernfalls wird 0FFH zurückgegeben. In DOS 4.x und 5.x liefert die Funktion den Wert AL=FFH zurück.

AL = 03H (Set Device Access Mode)

Mit dem Unterfunktionscode 03H läßt sich per INT 21 die Einstellung der Funktion AVILDEV beeinflussen. Der neue Wert wird per Register DL übergeben. Mit 00H lassen sich anschließend die Geräte nur noch über die Pseudo-Einheit \DEV erreichen. Die Eingabe DL = 0FFH schaltet die Einstellung wieder zurück. Nach dem Aufruf enthält das DL-Register die aktuelle Einstellung. Der Aufruf wird ab DOS 4.0 komplett ignoriert.

Die Funktion wurde bei DOS 2.x eingeführt. Allerdings unterstützen nicht alle Programme die modifizierten Switchcodes. Ab DOS 3.x wurde dann die AVILDEV-Funktion wieder entfernt, um Konsistenzprobleme in Netzwerkumgebungen vorzubeugen. DOS 4.x speichert noch das Trennzeichen, wertet es aber nicht aus. In DOS 5.0 wird selbst das Zeichen nicht mehr gespeichert. Der Funktionsaufruf 37H blieb aber erhalten.

4.53 Get/Set Country Data (Funktion 38H)

Die Funktion wird bereits ab DOS 2.0 unterstützt, besitzt aber je nach Version verschiedene Übergabeparameter. Sie dient dazu, die landesspezifischen Informationen abzufragen und zu verändern.

4.53.1 Get Country Data (Funktion 38H, DOS 2.x)

Für DOS 2.1 gelten folgende Übergabeparameter, um die Daten zu lesen.

```
Ö------------------------------Ì
º      CALL:  INT 21 (DOS 2.x)  º
º                               º
º AH:    38H                    º
º AL:    00H                    º
º DS:DX  Zeiger auf einen Puffer º
û------------------------------À
º      RETURN                   º
º CY:    1 = Fehler             º
º AX:    Fehlercode             º
º CY:    0 = kein Fehler        º
º DS:DX  Zeiger auf einen Puffer º
Û------------------------------ì
```

Der Wert im Register AL muß vor dem Aufruf (DOS 2.1) zu Null gesetzt werden. Im Registerpaar DS:DX findet sich ein Zeiger auf einen 32-Byte-Puffer, der vom rufenden Prozeß angelegt wird. In diesen Puffer übergibt die Funktion die landesspezifischen Daten, die gemäß folgender Aufstellung kodiert werden:

Bytes	Bedeutung
2	Datums- und Zeitformat
2	ASCIIZ-String für das Währungssymbol
2	ASCIIZ-String für die Tausender Markierung
2	ASCIIZ-String für die Dezimal Markierung
24	reserviert

Tabelle 4.7: Puffer für die Landesinformationen; (DOS 2.x)

Für das Datums- und Zeitformat gelten in DOS 2.1 folgende Kodierungen:

Code	Format
0	USA Std:Min:Sek Monat/Tag/Jahr
1	Europa Std:Min:Sek Tag/Monat/Jahr
2	Japan Std:Min:Sek Jahr/Monat/Tag

Tabelle 4.8: Kodierung der Uhrzeit; (DOS 2.x)

Die nachfolgenden Felder enthalten ASCIIZ-Strings (ASCII-Zero-String), deren erstes Byte jeweils das Zeichen für das Währungssysmbol, die Tausender-Markierung und das Dezimalzeichen enthält. Die ASCIIZ-Strings werden durch ein Nullbyte (00H) abgeschlossen. Die restlichen 24 Byte werden bei DOS 2.1 nicht benutzt.

Bei DOS 2.0 gelten die gleichen Aufrufkonventionen, wobei die ASCII-Strings nur aus einem Byte bestehen. Es handelt sich also nicht um ASCIIZ-Strings, so daß das Nullbyte fehlt. In DOS 2.0 bleiben 27 Byte innerhalb der Datentabelle unbenutzt.

Bei allen DOS-Versionen unter 3.0 lassen sich nur die landesspezifischen Informationen abfragen, aber nicht setzen.

4.53.2 Get Country Data (Funktion 38H, DOS 3.0-6.x)

Ab DOS 3.0 lassen sich mit dieser Funktion landesspezifische Informationen setzen und abfragen. Es gelten folgende Übergabeparameter, um die Daten zu lesen.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    38H                    º
º AL:    Steuercode             º
º BX:    Landescode (16 Bit)    º
º DS:DX  Zeiger auf einen Puffer º
û-------------------------------À
º      RETURN                   º
º CY:    1 = Fehler             º
º AX:    Fehlercode             º
º CY:    0 = kein Fehler        º
º BX:    Landescode             º
Û-------------------------------ì
```

Das Betriebssystem MS-DOS ist in der Lage, sich auf landesspezifische Besonderheiten (Datumsschreibweise, Währungen, Separatoren für Zeit etc.) einzustellen. Der Wert im Register AL steuert dabei die Aktionen der Funktion 38H. Es gelten folgende Belegungen:

```
Ö--------------Ú-------------------------------------------------Ì
º AL = 0       º lese die Daten des aktuell eingestellten Landes º
û--------------é-------------------------------------------------À
º AL = 1 - FEH º spezifiziert den gewünschten Landescode         º
û--------------é-------------------------------------------------À
º AL = FFH     º Der Landescode besitzt einen Wert über 255 und  º
º              º wird im Register BX übergeben                   º
û--------------é-------------------------------------------------À
º BX           º 16 Bit Landescode, falls der Wert > 255 ist     º
Û--------------Ù-------------------------------------------------ì
```

Tabelle 4.9: Übergabe des Landescodes (DOS 3.x - 6.x)

Das Register AL besitzt eine Mehrfachfunktion. Falls der Wert = 0 ist, werden die Daten des aktuell eingestellten Landes gelesen. Werte zwischen 01H und 0FEH spezifizieren einen bestimmten Landescode, dessen Parameter gelesen werden sollen. Falls der Landescode über 255 liegt, wird der Wert von AL auf 0FFH gesetzt. Dies signalisiert, daß sich der eigentliche Landescode im Register BX befindet.

Der Landescode lehnt sich an der internationalen Telefonvorwahl an. Folgende kleine Tabelle enthält einen Auszug aus diesen Codes.

```
Ö---------------------Ú----------------------Ú--------------------Ì
º 01  USA / Kanada    º   34  Spanien         º 47  Norwegen       º
º 07  UDSSR           º   39  Italien         º 48  Polen          º
º 30  Griechenland    º   41  Schweiz         º 49  Deutschland    º
º 31  Niederlande     º   44  Großbritannien  º 61  Australien     º
º 32  Belgien         º   45  Dänemark        º 81  Japan          º
º 33  Frankreich      º   46  Schweden        º                    º
Û---------------------Ù----------------------Ù--------------------ì
```

Tabelle 4.10: Auszug aus der Landescode-Tabelle

Da die Funktion verschiedene Daten zurückgibt, muß im rufenden Programm ein Speicherbereich von 34 Byte reserviert werden. Das Registerpaar DS:DX enthält einen Zeiger auf diesen Speicherblock.

Die landesspezifischen Daten werden dann in dem 32-Byte-Feld abgelegt. Es gilt folgender Aufbau des Feldes:

Offset	Bytes	Bemerkung
00	2	Datumsformat
02	5	Währungssymbol (ASCIIZ - String)
07	2	Trennzeichen Tausender (ASCIIZ - String)
09	2	Dezimalzeichen (ASCIIZ - String)
0B	2	Trennzeichen Datum (ASCIIZ - String)
0D	2	Trennzeichen Zeit (ASCIIZ - String)
0F	1	Bitfeld für weitere Kodierungen
10	1	Währungsformat
11	1	Zeitdarstellung
12	4	Case Map Call Adresse für Codes > 7FH
16	2	Data List Separator (ASCIIZ - String)
18	10	reserviert

Tabelle 4.11: Belegung des Feldes mit den landesspezifischen Informationen

Bei den ASCIIZ-Strings handelt es sich um Zeichenketten, die durch ein Nullbyte (00H) abgeschlossen werden. Die Bedeutung der einzelnen Einträge läßt sich noch weiter aufgliedern. Das Datumsformat (Offset 02H) wird in drei verschiedenen Formaten ausgegeben:

```
0 = USA (m/d/y)
1 = Europa (d/m/y)
2 = Japan (y/m/d)
```

Für das Währungssymbol ist ein String von 5 Zeichen (Offset 02H) vorgesehen. Die Darstellung innerhalb der Ausgabe wird durch das Bitfeld kodiert.

```
 2 1 0
Ö-Ú-Ú-Ì
ÛÚÙÚÙÚÙÚì
 º º Û-   0  Währungssymbol vor die Zahl stellen
 º º      1  Währungssymbol hinter die Zahl stellen
 º º
 º º      0  kein Leerzeichen zwischen Zahl und Währungssymbol
 º Û---   1  Leerzeichen zwischen Zahl und Währungssymbol
 º
 º
 Û-----   1  Das Dezimalzeichen wird durch das Währungssymbol
             ersetzt
```

Bild 4.5: Kodierung des Bitfeldes

Die restlichen Bits sind undefiniert. Das Feld Währungssymbol (Offset 10H) gibt an, wie viele Stellen nach dem Dezimalpunkt auszugeben sind (Beispiel: 2 Stellen bei DM).

Das Zeitformat ermöglicht eine Ausgabe im 12- oder 24-Stunden-Format:

Code	Darstellung
0	12 Stunden Uhrzeit
1	24 Stunden Uhrzeit

Die *Case Map Call Address* bezeichnet einen Vektor (Segment:Offset), der auf eine Routine zeigt, die die Konvertierung von Klein- in Großbuchstaben vornimmt. Dies sind die Zeichen im Bereich 80H bis FFH. Da diese Konvertierung landesspezifisch ist, existiert hier eine eigene Prozedur. Diese wird mit einem FAR CALL auf diese Adresse aktiviert.

```
Ö------------------------------Ì
º      FAR  CALL  auf Case-    º
º      MAP Adresse             º
º                              º
º AL:  zu wandelndes Zeichen   º
û------------------------------À
º      RETURN                  º
º AL:  gewandeltes Zeichen     º
Û------------------------------ì
```

Das zu wandelnde Zeichen wird im AL-Register übergeben. Falls ein Großbuchstabe zu diesem Zeichen existiert, gibt die Funktion das Ergebnis im AL-Register zurück. Falls kein Zeichen existiert, oder der Wert kleiner als 80H war, bleibt der Urwert im Register AL erhalten.

Der Versuch das Zeichen ß zu konvertieren, dürfte zu einem solchen Ergebnis führen. Die Umsetzungsroutine benutzt nur das AL-Register und die Flags.

Falls beim Aufruf der Funktion 38H ein Fehler auftritt, wird das Carry-Flag gesetzt. Im Register AL findet sich dann ein Fehlercode mit folgender Bedeutung:

```
 Code º Bedeutung
------é-------------------------------------
   2  º falscher oder unbekannter Landescode
```

Den erweiterten Fehlercode kann man über den Funktionsaufruf 59H (Get Extended Error) abfragen.

4.53.3 Set Country Data (Funktion 38H, DOS 3.0-6.x)

Die Funktion 38H dient auch dazu, bestimmte landesspezifische Einstellungen zu setzen. Es gelten dann folgende Übergabeparameter:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    38H                   º
º DX:    0FFFFH -> setze Daten º
º AL:    Steuercode            º
º BX:    Landescode            º
û------------------------------À
º      RETURN                  º
º CY:    1 = Fehler            º
º AX:    Fehlercode            º
º CY:    0 = kein Fehler       º
Û------------------------------ì
```

Falls das Register DX den Wert -1 (0FFFFH) enthält, setzt die Funktion den im Register übergebenen Landescode. Da DX normalerweise die Offsetadresse des Ergebnispuffers enthält, scheint sich hier ein Problem zu ergeben. Da aber der Puffer nicht über eine Segmentgrenze hinausreichen kann, ist der größte erlaubte Offsetwert 0FFFDH. Der Code 0FFFFH ist also frei.

Die Kodierung des Landes erfolgt analog dem oben beschriebenen Muster:

AL = 1 - FEH	spezifiziert den gewünschten Landescode
AL = FFH	Der Landescode besitzt einen Wert über 255 und wird im Register BX übergeben
BX	16 Bit Landescode, falls der Wert > 255 ist

Tabelle 4.12: Übergabe Landescode (DOS 3.x - 4x)

Tritt ein Fehler auf, z.B. der Landescode existiert nicht, ist das Carry-Flag nach dem Aufruf gesetzt. Im Register AL steht der Wert 2 (Landescode ungültig oder es existiert keine Tabelle für diesen Code).

Weiterhin lassen sich die erweiterten Fehlercodes über den Aufruf 59H (Get Extended Error) abfragen.

DOS kann intern nur die Landescodes 01 (USA) und 61 (Australien) bearbeiten. Andere Ländercodes werden über die speziellen Tastaturtreiber (z.B. KEYBGR, oder COUNTRY.SYS) unterstützt. Diese sind beim Systemstart mittels der Datei AUTOEXEC.BAT zu aktivieren. Weiterhin muß das Programm NLSFUNCTION geladen sein. Neben der Umcodierung der Tastatur generieren sie auch die landesspezifischen Daten.

Wenn nun ein solcher Treiber aktiviert wird, lädt DOS den Code in den unteren Bereich des Anwenderprogrammspeichers. Von der Routine wird der Interruptvektor der Tastatur auf den Treiber umgesetzt. Damit ist die landesspezifische Umsetzung der Tastatureingaben installiert. Durch gleichzeitiges Betätigen der Tasten <Ctrl><Alt><F1> läßt sich die Original-US-Darstellung reaktivieren. Mit <Ctrl><Alt><F2> gelangt man wieder in die landesspezifische Darstellung zurück. Die Umschaltung erfolgt innerhalb des Treibers, der zusätzlich die Umsetztabelle für die US-Version besitzt. Falls die Tastaturanpassung lediglich über BIOS-ROM-Routinen unterstützt wird, ist die Umschaltung allerdings meist nicht möglich.

4.54 Create Directory (Funktion 39H, DOS 2.0-6.x)

Diese Funktion erzeugt ein Unterverzeichnis in der File-Allocation-Tabelle (FAT).

```
 Unterverzeichnis erzeugen

Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:    39H                     º
º DS:DX  Zeiger auf den ASCIIZ-  º
º        String mit dem Pfad     º
û--------------------------------À
º      RETURN                    º
º Carry: gesetzt -> Fehler       º
º AX:    Fehlercode              º
º Carry: nicht gesetzt           º
º        kein Fehler             º
Û--------------------------------ì
```

Im Register DS:DX findet sich ein Zeiger, der auf einen ASCIIZ-String mit dem Pfadnamen zeigt. Dieser wird mit einem Nullbyte abgeschlossen. Falls bei der Operation ein Fehler auftritt, ist bei der Rückkehr das Carry-Flag gesetzt.

Im AX-Register findet sich ein Fehlercode:

```
 Code º Bedeutung
------é-----------------------------------------------------
   3  º Pfad nicht gefunden
   5  º Unterverzeichnis nicht anlegbar, Zugriff abgewiesen
```

Der Fehlercode 5 kann mehrere Ursachen haben:

- Es ist kein Eintrag im Hauptinhaltsverzeichnis; mehr frei.
- Es existiert bereits eine Datei mit dem Namen des Unterverzeichnisses.
- Der Pfad gibt ein Laufwerk ohne Subdirectory an.

In allen diesen Fällen kann kein Unterverzeichnis angelegt werden.

Über die Funktion 59H lassen sich die erweiterten Fehlercodes abfragen. Bei Anwendungen innerhalb eines Netzwerkes (ab DOS 3.1) muß der Prozeß das Create-Access-Zugriffsrecht besitzen. Die maximale Länge des ASCIIZ-Strings mit dem Pfadnamen darf 64 Zeichen nicht überschreiten.

4.55 Remove Directory (Funktion 3AH, DOS 2.0-6.x)

Diese Funktion löscht ein Unterverzeichnis in der File-Allocation-Tabelle (FAT), falls das Verzeichnis keine Dateien mehr enthält.

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:    3AH                     º
º DS:DX  Zeiger auf den ASCIIZ -º
º        String mit dem Pfad     º
û--------------------------------À
º      RETURN                    º
º Carry: gesetzt -> Fehler       º
º AX:    Fehlercode              º
º Carry: nicht gesetzt           º
º        kein Fehler             º
Û--------------------------------ì
```

Im Register DS:DX findet sich ein Zeiger, der auf einen ASCIIZ-String mit dem Pfadnamen zeigt. Falls bei der Operation ein Fehler auftritt, ist bei der Rückkehr das Carry-Flag gesetzt. Im AX-Register findet sich ein Fehlercode:

```
 Code ° Bedeutung
------é------------------------------
   3  ° Pfad nicht gefunden
   5  ° Zugriff abgewiesen
  16  ° aktuelles Inhaltsverzeichnis
```

Der Fehlercode 5 kann mehrere Ursachen haben:

- Das Unterverzeichnis ist nicht leer.
- Der Pfad gibt kein Unterverzeichnis an.
- Es wurde versucht, das Hauptverzeichnis zu löschen.
- Es existiert kein Eintrag mit dem angegebenen Namen.

Der Fehler 16 gibt an, daß versucht wurde, daß aktuelle Unterverzeichnis zu löschen. Es ist jedoch nur möglich, aus dem aktuellen Verzeichnis die jeweiligen Unterverzeichnisse zu löschen. Die Ursache liegt unter anderem in der Directory-Struktur begründet.

Über die Funktion 59H lassen sich die erweiterten Fehlercodes ermitteln. Falls kein Fehler auftritt, wird das Unterverzeichnis aus dem Inhaltsverzeichnis entfernt.

Bei Zugriffen innerhalb eines Netzwerkes wird vom rufenden Prozeß das Create-Access-Privileg verlangt.

4.56 Change Current Directory (Funktion 3BH, DOS 2.0-6.x)

Diese Funktion selektiert das angegebene Unterverzeichnis. Es sindfolgende Übergabeparameter definiert:

```
Ö------------------------------Ì
°      CALL:  INT 21           °
°                              °
° AH:    3BH                   °
° DS:DX  Zeiger auf den ASCIIZ -°
°        String mit dem Pfad   °
û------------------------------À
°      RETURN                  °
° Carry: gesetzt -> Fehler     °
° AX:    Fehlercode            °
° Carry: nicht gesetzt         °
°        kein Fehler           °
Û------------------------------ì
```

Im Register DS:DX findet sich ein Zeiger, der auf einen ASCIIZ-String mit dem Pfadnamen zeigt. Hierbei kann es sich auch um eine unvollständige Pfadangabe handeln. Falls bei der Operation ein Fehler auftritt, ist bei der Rückkehr das Carry-Flag gesetzt. Im AX-Register findet sich ein Fehlercode:

```
   Code ° Fehler
--------é---------------------
    3   ° Pfad nicht gefunden
```

Die erweiterten Fehlercodes lassen sich über die Funktion 59H ermitteln. Der String des Pfadnamens ist auf 64 Zeichen begrenzt. Diese Beschränkung gilt auch für Pfadnamen, die innerhalb der DOS-Kommandoebene eingegeben werden. Falls der Pfad nicht existiert, oder falls der Pfad unkorrekt ist, dann bricht die Funktion ohne Änderungen ab. Im Pfadnamen darf kein Zugriff auf einen Netzwerkknoten spezifiziert werden.

4.57 Create Handle (Funktion 3CH, DOS 2.0-6.x)

Diese Funktion gehört zu den neueren MS-DOS-File-I/O-Befehlen. Sie erzeugt eine leere Datei und ordnet dieser den ersten freien Filehandle zu. Der Begriff Handle wird in einem eigenen Kapitel beschrieben. Es sind folgende Übergabeparameter definiert:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    3CH                    º
º DS:DX  Zeiger auf den ASCIIZ -º
º        String mit dem Pfad    º
º CX:    Fileattribut           º
Û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: nicht gesetzt          º
º AX:    Handle                 º
Û-------------------------------ì
```

Im Registerpaar DS:DX findet sich ein Zeiger auf den ASCIIZ-String mit dem Pfadnamen. In CX lassen sich die Dateiattribute ablegen. Diese Attribute werden im Rahmen der Funktion 43H besprochen.

Falls die angegebene Datei noch nicht existiert, legt DOS sie an. Andernfalls wird die bestehende Datei auf die Länge 0 begrenzt. Damit geht der Dateiinhalt verloren und läßt sich mit Undelete-Tools nicht wieder restaurieren.

Deshalb ist ab DOS 3.0 die Funktion 5BH zu bevorzugen.

Die Attribute in CX werden dann dieser Datei zugewiesen. Falls ein Fehler auftritt, ist bei der Rückkehr das Carry-Flag gesetzt. Im Register AX findet sich ein Fehlercode:

```
 Code º Bedeutung
------é--------------------------
   3  º Pfad nicht gefunden
   4  º zu viele offene Dateien
   5  º Zugriff abgewiesen
```

Der Fehler 5 tritt auf, falls kein Platz im Inhaltsverzeichnis für eine neue Datei vorhanden ist, oder falls der Name bereits für ein Unterverzeichnis vergeben wurde. Weiterhin kann der Zugriff abgewiesen werden, falls eine Datei mit dem Namen besteht, deren Attribute aber eine Veränderung nicht zulassen (z.B. Read Only). Mit der DOS-Funktion 59H lassen sich die erweiterten Fehlercodes abfragen.

Tritt kein Fehler auf, öffnet die Funktion die Datei für Schreib-/Leseoperationen und positioniert den Zeiger auf das erste Byte der Datei. Soll das Dateiattribut geändert werden, läßt sich die Funktion 43H (Change Mode) verwenden. Der zugewiesene Filehandle wird im Register AX zurückgegeben.

Bei Einsatz innerhalb eines Netzwerkes ist das Create-Access-Zugriffsrecht erforderlich.

Der Funktion 3CH fehlt allerdings eine Festlegung mit welchen SHARE-Flags die Datei zu öffnen ist. Ab DOS 4.0 wurde deshalb die Funktion 6CH eingeführt. Eine Datei wird durch die Funktion 3CH mit dem Code 02H (read/write) für den aktiven Prozeß geöffnet. Der Zugriff ist für andere Netzwerkteilnehmer gesperrt. Diese Eigenschaft wird auch an Subprozesse vererbt.

4.58 Open Handle (Funktion 3DH)

Bei diesem Aufruf handelt es sich um eine Xenix-orientierte Form der Dateibehandlung.

4.58.1 Open Handle (Funktion 3DH, DOS 2.x)

In DOS 2.x gelten jedoch etwas andere Aufrufkonventionen als bei den 3.x Versionen. Diese Funktion öffnet eine bereits vorhandene Datei.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    3DH   (DOS 2.x)        º
º AL:    Zugriffscode           º
º DS:DX  Zeiger auf den ASCIIZ -º
º        String mit dem Pfad    º
û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: nicht gesetzt          º
º AX:    Filehandle             º
Û-------------------------------ì
```

Der Vektor DS:DX zeigt auf einen ASCIIZ-String (ASCII-Zero-String) mit dem Pfadnamen. Das Register AL enthält den Zugriffscode (Accesscode):

```
 Code º Zugriffsart
------é-----------------------------------------
   0  º Datei für Lesezugriffe öffnen
   1  º Datei für Schreibzugriffe öffnen
   2  º Datei für Schreib-/Lesezugriffe öffnen
```

Der Aufruf bezieht sich auf Dateien mit dem Attribut *normal* oder *hidden*, die zusätzlich mit dem im Pfad spezifizierten Namen übereinstimmen müssen. Falls der Name mit einem Doppelpunkt (:) endet (z.B. PRN:), dann wird die Datei nicht geöffnet. Der Schreib-/ Lesezeiger wird auf das erste Byte der Datei positioniert und die Größe eines Records wird auf den Wert 1 Byte initialisiert. Mittels der Funktion 42H läßt sich anschließend der Zeiger innerhalb der Datei positionieren. Das Dateiattribut kann per Funktionsaufruf 43H neu gesetzt werden, während der Aufruf 57H die Einträge für Uhrzeit und Datum modifiziert.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und im AX-Register befindet sich ein Fehlercode. Es gelten die gleichen Fehlercodes wie bei der Funktion 59H (Get Extended Error).

Läßt sich die Datei öffnen, befindet sich in AX anschließend der zugewiesene Handlecode. Dieser wird für spätere I/O-Zugriffe auf diese Datei benötigt.

4.58.2 Open Handle (Funktion 3DH, DOS 3.0-6.x)

Dieser Aufruf gehört zu den Xenix-orientierten Systemdiensten, die Ein-/Ausgaben auf Dateien oder I/O-Einheiten (Printer, Console etc.) ermöglichen. Die Funktion öffnet eine vorhandene Datei, unabhängig von ihren Attributen, für Ein-/Ausgaben. Bei Namen, die in DOS mit einem Doppelpunkt (:) enden (z.B. COM1:) wird kein Versuch unternommen, eine entsprechende Datei zu öffnen, da ja eine Einheit angesprochen wurde. Allerdings darf der Doppelpunkt beim Open-Aufruf nicht im String auftauchen, da er nur auf der Kommandoebene gültig ist. Es gelten folgende Übergabekonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    3DH (DOS 3.x)          º
º AL:    Access Code            º
º DS:DX  Zeiger auf den ASCIIZ -º
º        String mit dem Pfad    º
û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: nicht gesetzt          º
º AX:    Filehandle             º
Û-------------------------------ì
```

Im Registerpaar DS:DX befindet sich ein Zeiger auf einen ASCIIZ-String mit dem Pfadnamen der Datei. Das Register AL enthält den Zugriffscode für die Datei. Hierbei handelt es sich um ein Byte mit folgender Kodierung:

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙÚÙ-ÙÚÙÚÙÚÙ-ÙÚì
 º Û-Ú-ì º Û-Ú-ì
 º   º   º   Û----  Access Code
 º   º   º
 º   º   Û--------  reserviert
 º   º
 º   Û------------  Sharing Code
 º
 Û----------------  Inherit (vererbbar) Bit
```

Bild 4.6: Zugriffscodes für den Open-Handle-Aufruf;

Das Byte gibt an, welche Zugriffsrechte auf die Datei existieren.

Zugriffs-(Access-)Codes

Die Bits 0 bis 2 spezifizieren, ob die Datei zum Lesen oder Schreiben geöffnet wurde:

```
Ö-------Ú---------Ú-----------------------------------------------Ì
º  Wert º Zugriff º Bemerkung                                      º
û-------é---------é-----------------------------------------------À
º  000  º  read   º Nur ein lesender Zugriff auf die Datei        º
û-------é---------é-----------------------------------------------À
º  001  º  write  º Nur ein schreibender Zugriff auf die Datei    º
û-------é---------é-----------------------------------------------À
º  010  º  both   º Schreib- und Lesezugriffe sind auf die Datei  º
º       º         º erlaubt.                                      º
Û-------Ù---------Ù-----------------------------------------------ì
```

Tabelle 4.13: File Access Codes;

Sharing Mode

Dieses Bitfeld (Bit 4, 5, 6) legt die Zugriffsberechtigung für verschiedene Prozesse innerhalb eines Netzwerkes fest. Um zu den netzwerklosen DOS-Programmen kompatibel zu bleiben, existiert ein *Compatibility Mode*. Wird eine Datei mit diesem Zugriffsmode geöffnet, kann kein anderer Netzwerkteilnehmer mehr auf diese Datei zugreifen. Um einen gleichzeitigen Zugriff von mehreren Netzwerkteilnehmern zu erlauben, kann ein *Sharing Code* angegeben werden. Um diese Funktion allerdings zu nutzen, muß vorher das DOS-Kommando SHARE.COM geladen werden, um die netzwerkspezifischen Funktionen zu installieren. Die folgende Tabelle spezifziert die Share-Zugriffsstufen sowie die Kodierung des Bitfeldes:

```
 Code ° Zugriffsmode
------é---------------
 000  ° compatibility
 001  ° deny both
 010  ° deny write
 011  ° deny read
 100  ° deny non
```

Es sind nur diese Kombinationen zulässig. Innerhalb der einzelnen Zugriffsmodi gelten unterschiedliche Bedingungen.

Compatibility Mode

Dieser Mode ermöglicht eine Kompatibilität zu Softwareprogrammen, die für einen netzwerklosen Betrieb ausgerichtet sind. Ein Prozeß kann die Datei beliebig oft öffnen, Zugriffe von anderen Netzwerkteilnehmern werden aber abgewiesen. Der *Compatibility Mode* wird einer Datei zugewiesen, falls sie per:

- CREATE-Funktionsaufruf
- FCB-Funktionsaufruf
- Handle-Funktionsaufruf mit *Compatibility Mode*

eröffnet wird. Falls die Datei bereits in einem anderen Mode eröffnet wurde, führt ein weiterer Versuch mit dem Attribut *Compatibility* zu einem INT 24 (Critical Error). Es wird die Meldung *Drive not ready* ausgegeben und eine Abfrage des Extended Error Codes zeigt einen *Sharing Violation Error* an. Neben dem *Compatibility Mode*, welcher alle Zugriffe fremder Prozesse abweist, läßt sich das Zugriffsrecht auch explizit spezifizieren.

Deny read/write

Mit diesem Mode kann ein Prozeß einen exklusiven Zugriff auf die Datei erwirken. Falls die Datei erfolgreich mit diesem Mode geöffnet wird, kann nur noch der öffnende Prozeß auf die Datei zugreifen. Alle Schreib-/Lesezugriffe anderer Prozesse (Netzwerkteilnehmer) werden abgewiesen. Es ist auch nicht möglich, das Zugriffsrecht durch andere Prozesse zu ändern. Lediglich der aktuelle Prozeß darf die Datei beliebig oft mit beliebigen Attributen öffnen. Der Mode wird automatisch zurückgesetzt, falls die Datei geschlossen wird.

Deny write

Wird eine Datei mit diesem Attribut geöffnet, hat nur noch der aktuelle Prozeß das Schreibzugriffsrecht, bis die Datei geschlossen wird. Alle Schreiboperationen anderer Prozesse werden abgewiesen. Der Versuch einer Open-File-Anweisung mit dem Mode *Deny Write* ist nicht möglich, falls die Datei bereits anderweitig mit dem Zugriffsrecht *Write Access* eröffnet wurde.

Deny read

Eine mit diesem Mode eröffnete Datei weist alle Lesezugriffe fremder Prozesse zurück. Der Mode wird automatisch beim *File Close* gelöscht. Der Versuch, die Datei mit *Deny Read* zu öffnen, wird abgebrochen, falls die Datei bereits anderweitig im *Compatibility Mode* oder mit dem Zugriffsrecht *Read Access* geöffnet wurde.

Deny none

Mit diesem Attribut werden keine Schreib-/Lesebeschränkungen für andere Prozesse gesetzt. Der Aufruf ist nicht möglich, falls die Datei bereits anderweitig im *Compatibility Mode* eröffnet wurde.

Grundsätzlich gilt für *Deny Read/Deny None*, daß innerhalb eines Netzwerkes keine lokalen Datenpuffer für die Dateien angelegt werden. Die Modes Deny Read/Write, Deny Write und Compatibility arbeiten dagegen innerhalb eines Netzwerkes mit lokalen Puffern.

Inherit Bit

Dieses Bit gibt an, ob die Datei an *Child Processes* vererbt werden kann. Falls das Bit = 0 ist, kann die Datei auch aus einem *Child Process* bearbeitet werden, andernfalls ist sie nur für den Hauptprozeß zugreifbar.

Das reservierte Bit 4 sollte immer auf den Wert 0 gesetzt werden.

Falls ein Fehler auftritt, wird das Carry-Flag gesetzt und AX enthält einen Fehlercode mit folgender Kodierung:

```
Code ° Bedeutung
-----é-------------------------------------------------------
  1  ° File Sharing muß geladen werden, um einen Sharing Code
     ° zu nutzen
     °
  2  ° Der Dateiname ist ungültig oder unbekannt
     °
  3  ° Der Pfadname ist ungültig oder unbekannt
     °
  4  ° Es ist kein Handle mehr frei, oder die intenen
     ° Systemtabellen laufen über
     °
  5  ° Das Programm versucht ein Unterverzeichnis oder das
     ° Volume Label als Datei zu öffnen
     °
 12  ° Der Zugriffcode ist nicht 0, 1 oder 2
```

Falls der Systemaufruf wegen eines *File Sharing Errors* abgebrochen werden muß, erzeugt die Funktion einen INT 24 mit dem Code 2 (Drive not ready). Dieser Code läßt sich durch die Funktion 59H abfragen.

Bei der Eröffnung der Datei müssen die Zugriffsrechte festgelegt werden. Standardmäßig gilt der *Compatibility Mode* für frühere DOS-Versionen, die keinen Fremdzugriff erlauben.

4.59 Close Handle (Funktion 3EH, DOS 2.0-6.x)

Diese Funktion schließt eine Datei, die mit den Befehlen 3DH (Open Handle) oder 3CH (Create Handle) eröffnet wurde.

```
Ö-------------------------------Ì
°      CALL: INT 21             °
°                               °
° AH:    3EH                    °
° BX:    Handle                 °
û-------------------------------À
°      RETURN                   °
° Carry: gesetzt -> Fehler      °
° AX:    6 = Invalid Handle     °
° Carry: nicht gesetzt          °
°        kein Fehler            °
Û-------------------------------ì
```

Im Register BX muß lediglich der Handle der Datei angegeben werden. Dieser Handlecode wird bei der Create- oder Open-Handle-Operation zurückgegeben.

DOS lagert den Inhalt der Schreibpuffer auf das Medium aus und aktualisiert die Filegröße, sowie Datum und Uhrzeit des letzten Schreibzugriffs. Im Netzwerk werden zusätzlich alle mit *Lock* gesetzten Sperren auf der Datei aufgehoben.

Bei einem fehlerhaften Handle wird das Carry-Flag gesetzt und das Register AX enthält den Fehlercode *6 Invalid Handle*. Über die Funktion 59H lassen sich die erweiterten Fehlercodes abfragen. Nach einem fehlerfreien Aufruf ist das Inhaltsverzeichnis aktualisiert und die internen Puffer sind gelöscht. Weiterhin ist der betreffende Handle wieder freigegeben (FFH im PSP).

4.60 Read from a File or Device (Funktion 3FH, DOS 2.0-6.x)

Diese Funktion erlaubt es, aus einer geöffneten Datei oder Einheit zu lesen. Es gelten die Übergabeparameter der folgenden Tabelle.

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:    3FH                     º
º BX:    Handle                  º
º CX:    Zahl der zu lesenden    º
º        Bytes                   º
º DS:DX  Zeiger auf einen Puffer º
û--------------------------------À
º      RETURN                    º
º Carry: gesetzt -> Fehler       º
º AX:    Fehlercode              º
º Carry: nicht gesetzt           º
º AX:    Zahl der gelesenen      º
º        Bytes                   º
Û--------------------------------ì
```

Neben der Nummer des Handle in BX wird die Zahl der zu lesenden Bytes im Register CX geführt. So lassen sich auf einen Schlag bis zu 64 Kbyte lesen. Im Registerpaar DS:DX findet sich ein Zeiger auf den Puffer, in dem die gelesenen Zeichen abgespeichert werden.

Der Lesezugriff auf Dateien erfolgt ab der aktuellen Position des Lesezeigers. Der Zeiger wird anschließend um die Zahl der gelesenen Bytes verschoben.

Falls ein Fehler auftritt, ist das Carry-Flag gesetzt und im AX-Register findet sich ein Fehlercode:

```
5  Handle nicht zum Lesen geöffnet
6  Handle nicht geöffnet, oder ungültig
```

Bei fehlerfreiem Zugriff ist das Carry-Flag = 0 und im Register AX steht die Zahl der gelesenen Bytes. Der Versuch, hinter der EOF-Marke zu lesen, führt zum Wert AX = 0. Es werden dann keine Zeichen gelesen. Die in CX angegebene Zeichenzahl kann nicht immer erreicht werden. Ein Einlesen aus dem Tastaturpuffer bricht z.B. ab, sobald ein CR auftritt.

Wird CX = 0 gesetzt, läßt sich prüfen ob das Handle für Leseoperationen geöffnet ist, ohne daß ein Zeichen übertragen wird.

Die Funktion erlaubt eine Umleitung der Einheiten. Bei Zugriffen auf Geräte unterscheidet die Funktion über Bit 5 des Geräteattributes zwischen den Modes *raw* und *cooked*. Im *raw mode* werden die gelesenen Daten nicht interpretiert. Für die Einheit CON: ist der Mode

cooked standardmäßig eingestellt. Dies bedeutet, daß bei I/O-Umleitungen über CON: die Eingaben durch ein RETURN-Zeichen abzuschließen sind. Andernfalls terminiert die Funktion nicht und das System muß neu gebootet werden.

Für einen Zugriff auf Dateien innerhalb eines Netzwerkes ist das Read-Access-Privileg erforderlich. Wurde SHARE (oder ein anderer Netwerktreiber) geladen, sind Zugriffe auf gesperrte Dateien oder Sätze unzulässig. Wurde die Datei im *Compatibility Mode* geöffnet und an einen Subprozess vererbt (inherit), dann löst ein Zugriff auf diese Datei einen INT 24 aus. Zugriffe auf nicht gesperrte Bereich sind dagegen immer möglich, auch wenn das inherit-Flag nicht gesetzt ist.

Bei Zugriffen auf gesperrte Dateien, die mit SHARE-Flags für Lesezugriffe anderer Prozesse geöffnet wurden bricht die Funktion mit einem Fehlercode AX=0005H ab.

4.61 Write to a File or Device (Funktion 40H, DOS 2.0-6.x)

Diese Funktion erlaubt es, in eine geöffnete Datei oder zu einer Einheit zu schreiben. Es gelten folgende Übergabeparameter.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    40H                    º
º BX:    Handle                 º
º CX:    Zahl der zu schreiben- º
º        den Bytes              º
º DS:DX  Zeiger auf einen Puffer º
û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: nicht gesetzt          º
º AX:    Zahl der geschriebenen º
º        Bytes                  º
Û-------------------------------ì
```

Neben der Handlenummer in BX wird die Zahl der zu schreibenden Bytes im Register CX geführt. So lassen sich auf einen Schlag bis zu 64 Kbyte schreiben. Im Registerpaar DS:DX findet sich ein Zeiger auf den Puffer, in dem die Zeichen zwischengespeichert sind.

Falls ein Fehler auftritt, ist das Carry-Flag gesetzt und im AX-Register findet sich der Fehlercode:

```
5  Handle nicht zum Schreiben geöffnet
6  Handle nicht geöffnet, oder ungültig
```

Die erweiterten Fehlercodes lassen sich durch die Funktion 59H abfragen.

Bei fehlerfreiem Zugriff ist das Carry-Flag = 0 und im Register AX steht die Zahl der geschriebenen Bytes. Falls AX nach dem Aufruf den Wert *0* hat, ist die Disk voll. Wenn der Wert kleiner als CX ist, trat während des Schreibvorgangs ein Fehler auf, ohne daß das Carry-Flag gesetzt wurde. Die Funktion erlaubt eine Ein-/Ausgabeumleitung der Ausgaben an verschiedene Einheiten, falls sich die Ausgabe auf die Standardausgabe bezieht.

Es wird ab der aktuellen Position des Schreibzeiger ausgegeben. Um eine Datei auf eine bestimmte Länge zu begrenzen, wird üblicherweise die gewünschte Zahl der Bytes ab dem Dateianfang geschrieben. Alternativ besteht die Möglichkeit, den Schreib-/Lesezeiger mittels der Funktion 42H (Move File R/W Pointer) neu zu positionieren. Anschließend

wird die Funktion 40H mit CX = 0 (Zahl der Bytes = 0) aufgerufen. Dies hat den Effekt, daß die Datei hinter dem Schreib-/Lesezeiger abgeschnitten wird. Die Dateilänge ist damit begrenzt. Wird der Schreibzeiger auf den Anfang der Datei positioniert, läßt sich mit obiger Methode die Datei löschen. Befindet sich der Schreibzeiger hinter dem Dateiende (z.B. per Funktion 42H), verlängert DOS die Datei und füllt den Zwischenraum zwischen EOF und der aktuellen Position mit undefinierten Zeichen.

Bei Zugriffen auf Geräteeinheiten unterscheidet DOS über das Bit 5 im Geräte-Attribut, ob der Zugriff im raw- oder cooked-Mode erfolgt. Im raw-Mode erfolgt keine Interpretation der Zeichen. Bei Ausgaben auf Druckern kann die Interpretation der Zeichen störend sein, zum Beispiel wenn das Zeichen EOF (26H) bei der Ausgabe von Graphikdaten herausgefiltert wird. Die Treiber für AUX: und PRN: arbeiten standardmäßig im *cooked*-Mode.

Falls die Datei nur das Read-Attribut besitzt, werden alle Schreibzugriffe abgewiesen. Innerhalb eines Netzwerkes benötigt der rufende Prozeß das Write-Zugriffsrecht. Bei Zugriffen auf gesperrte Dateien oder Bereiche in Netzwerken mit geladenem SHARE-Programm hängt die Reaktion des Betriebssystemes vom Dateimode ab. Wird die Datei im *Compatibility Mode* eröffnet und an einen Subprozess vererbt, dann löst DOS einen INT 24 aus. Wurde die Datei als *shared* eröffnet, dann bricht DOS mit einem Fehlercode AX=0005 ab, falls auf einen gesperrten Bereich zugegriffen wird.

4.62 Delete (Unlink) Directory Entry (Funktion 41H, DOS 2.0 - 6.x)

Mit der Funktion läßt sich eine Datei löschen, wobei folgende Aufrufparameter definiert sind.

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:    41H                     º
º DS:DX  Zeiger auf ASCIIZ-Stringº
º        mit dem Pfadnamen       º
û--------------------------------À
º      RETURN                    º
º Carry: gesetzt -> Fehler       º
º AX:    Fehlercode              º
º Carry: nicht gesetzt           º
º        kein Fehler             º
Û--------------------------------ì
```

Es wird allerdings nur der Eintrag im Inhaltsverzeichnis gelöscht (Unlink), die Datei bleibt weiter erhalten. Konkret wird das erste Zeichen des Filenamens in der FAT auf den Wert E5H gesetzt. Tools wie UNDELETE versuchen diesen Wert auf ein gültiges Zeichen zu setzen um so die gelöschte Datei wieder herzustellen. Im Registerpaar DS:DX findet sich ein Zeiger auf einen ASCIIZ-String mit dem Pfadnamen, der letztlich auch den Dateinamen enthält. Innerhalb des Dateinamens sind keine Wildcardzeichen (*) erlaubt. Der ASCIIZ-String ist mit einem Nullbyte abzuschließen.

Falls die Datei existiert und nicht als *Read Only* markiert ist, wird sie gelöscht. Tritt ein Fehler auf, wird das Carry-Flag gesetzt. Im AX-Register findet sich ein Fehlercode:

```
2  Pfad oder Dateiname existiert nicht
5  Der Pfad bezieht sich auf ein Unterverzeichnis
   oder einen Read-Only-File.
```

Wichtig ist, daß die Datei vor Aufruf der Funktion geschlossen wird. Andernfalls besteht die Möglichkeit, daß DOS in eine nicht mehr existierende Datei Daten auslagert.

Um einen Read-Only-File zu löschen, muß erst das Attribut mittels der Funktion 43H umgesetzt werden. Die erweiterten Fehlercodes lassen sich mit der Funktion 59H abfragen. Innerhalb eines Netzwerkes benötigt der Prozeß das Create-Access-Zugriffsrecht, um die Datei zu löschen. Ab DOS 3.1 kann die Funktion indirekt über die (undokumentierte) INT 21-Funktion 5D00H aktiviert werden, wobei dann auch Wildcards (*,?) im Filenamen auftreten dürfen.

4.63 Move File Pointer (Funktion 42H, DOS 2.0-6.x)

Mit dieser Funktion läßt sich der Schreib-/Lesezeiger innerhalb der Datei positionieren. Dies ist beim Zugriff über Filehandles notwendig.

```
Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:    42H                     º
º AL:    Methode der Bewegung    º
º BX:    Handle                  º
º CX:DX  Distanz in Bytes        º
Û--------------------------------À
º      RETURN                    º
º Carry: gesetzt -> Fehler       º
º AX:    Fehlercode              º
º Carry: nicht gesetzt           º
º DX:AX  neue Zeigerposition     º
Û--------------------------------ì
```

Die Handlenummer wird per Register BX übergeben. Das Registerpaar CX:DX enthält einen 32-Bit-Offsetwert, um den der Schreib-/Lesezeiger verschoben werden soll.

Der Code in AL spezifiziert die Positionierungsart:

```
0   º Positioniere ab dem Fileanfang um den Offset
    º
1   º Positioniere ab der aktuellen Position um den Offset
    º
2   º Positioniere ab dem Dateiende um den Offset
```

Falls ein Fehler auftritt, wird das Carry-Flag gesetzt. In AX steht ein Fehlercode:

```
1  AL war nicht 0, 1 oder 2
6  Der Handle war nicht geöffnet.
```

Die erweiterten Fehlercodes (Extended Errorcodes) lassen sich mittels der Funktion 59H abfragen.

Im fehlerfreien Fall findet sich im Registerpaar DX:AX die neue Position des Schreib-/Lesezeigers. Mit dem Steuercode AL = 2 (positioniere ab dem Dateiende um den Offset) und CX:DX = 0 (Offset = 0) läßt sich die aktuelle Dateilänge abfragen. Dies ist möglich, da mit dem Offset 0 der Zeiger nicht positioniert wird, die Funktion aber anschließend im Registerpaar DX:AX den Offsetwert relativ zum Dateianfang zurückgibt, was ja der Dateilänge in Byte entspricht. Der Positionszeiger läßt sich auch über das Dateiende hinaus positionieren, was bei Schreiboperationen zu undefinierten Daten in der Datei führt.

Bei Dateien innerhalb von Netzwerken gelten verschiedene Bedingungen hinsichtlich der Positionierung des Schreib-/Lesezeigers. Falls die Datei als *Shared* mit den Modes *Deny Read* oder *Deny None* eröffnet wurde, wird der Zeiger auf dem Rechner mit der physikalischen Datei positioniert. Bei allen anderen Modes wird die Information lediglich im Netzwerk gehalten.

4.64 Get/Set File Attributes (Funktion 43H, DOS 2.0 -6.x)

Mit dieser Funktion lassen sich die Dateiattribute lesen und verändern. Es gelten folgende Übergabeparameter.

```
Ö-------------------------------Ì
°      CALL: INT 21             °
°                               °
° AH:    43H                    °
° AL:    0 = Get  1 = Set       °
° CX:    Attribut bei Set       °
° DS:DX  Zeiger auf ASCIIZ -    °
°        String mit dem Pfad    °
û-------------------------------À
°      RETURN                   °
° Carry: gesetzt -> Fehler      °
° AX:    Fehlercode             °
° Carry: nicht gesetzt          °
° CX:    Attributbyte falls     °
°        AL = 0                 °
Û-------------------------------ì
```

Im Registerpaar DS:DX findet sich ein Zeiger auf einen ASCIIZ-String mit dem Pfadnamen, der auch die Dateibezeichnung enthält. Wildcards (*,?) sind hier nicht erlaubt. Das Register AL spezifiziert, ob ein Attribut gelesen (AL = 0) oder geschrieben (AL = 1) wird. Falls ein Schreibauftrag vorliegt, muß der Attributcode im Register CX übergeben werden.

Die Kodierung der Attribute erfolgt gemäß folgender Tabelle:

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙÚÙÚÙÚÙÚÙÚÙÚÙÚì
 ° ° ° ° ° ° ° Û-  Read Only
 ° ° ° ° ° ° Û---  Hidden File
 ° ° ° ° ° Û-----  System File
 ° ° ° ° Û-------  Volume Label
 ° ° ° Û---------  Subdirectory
 ° ° Û-----------  Archive Bit
 ° Û-------------  ---
 Û---------------  ---
```

Bild 4.7: Kodierung der Dateiattribute

Eine detailliertere Diskussion ist in dem Kapitel über die Dateiverwaltung enthalten. Es ist aber zu beachten, daß sich die Attribute *Volume Label* und *Subdirectory* nicht durch die Funktion ändern lassen. Diese Bits müssen beim Aufruf auf 0 gesetzt werden, sonst erfolgt eine Fehlermeldung.

Bei Zugriffen innerhalb eines Netzwerkes muß für das Verändern des *Archive Bits* keine besondere Zugriffsberechtigung vorliegen, während alle anderen Bits nur bei Vorliegen des Create-Access-Privilegs geändert werden können.

Tritt während des Funktionsaufrufes ein Fehler auf, wird das Carry-Flag gesetzt und im Register AX findet sich ein Fehlercode:

```
Code  º  Bemerkung
------é-----------------------------------------------------
  1   º AL ist nicht 0 (Get) oder 1 (Set)
      º
  2   º Datei nicht gefunden
      º
  3   º Der Pfad ist ungültig oder die Datei existiert nicht
      º
  5   º Das Attributbyte läßt sich nicht beschreiben, da der
      º Name
      º ein Unterverzeichnis oder ein Volume ID bezeichnet.
```

Falls das Carry-Flag nicht gesetzt ist und im Register AL der Wert 0 steht, wurde die Operation fehlerfrei ausgeführt. Beim Aufruf Get-Attribute findet sich dieses anschließend im Register *CX*.

4.65 IOCTL Data (Funktion 44H, DOS 2.0 -6.x)

Diese DOS Funktion ermöglicht es, Informationen über die geöffnete Einheit abzufragen oder Kontrollinformationen zum Treiber zu senden. Die Funktion 44H unterstützt verschiedene Unterfunktionen, die mit dem Code 00H bis 11H durch das Register AL selektiert werden. Die folgende Tabelle gibt einen kurzen Überblick über diese Unterfunktionen sowie die DOS-Version, ab der eine Unterstützung erfolgt.

```
Ö------Ú---------Ú-----------------------------------------Ì
º Code º Version º  Subfunktion                             º
û------é---------é-----------------------------------------À
º  00  º   2.0   º Get Device Information                   º
º  01  º   2.0   º Set Device Information                   º
º  02  º   2.0   º Read from a Character Device             º
º  03  º   2.0   º Write to a Character Device              º
º  04  º   2.0   º Read from a Block Device                 º
º  05  º   2.0   º Write to a Block Device                  º
º  06  º   2.0   º Get Input Status                         º
º  07  º   2.0   º Get Output Status                        º
º  08  º   3.0   º Is a particular Block Device changeable? º
º  09  º   3.1   º Is a logical Device local or remote?     º
º  0A  º   3.1   º Is a Handle local or remote?            º
º  0B  º   3.0   º Change Sharing Retry Code               º
º  0C  º   3.3   º Generic IOCTL Handle request           º
º  0D  º   3.2   º Block Device Generic IOCTL request     º
º  0E  º   3.2   º Get logical Drive                       º
º  0F  º   3.2   º Set logical Drive                       º
º  10  º   5.0   º Query Support (Handle)                  º
º  11  º   5.0   º Query Support (Block)                   º
Û------Ù---------Ù-----------------------------------------ì
```

Tabelle 4.14: IOCTL-(Funktion-44H)-Subfunktionen

Die Subfunktionen 00H-08H werden nicht innerhalb eines Netzwerkes unterstützt. Vor Aufruf der Subfunktion 0BH muß das File-Sharing-Kommando (SHARE) geladen werden. Die IOCTL-Funktionen tauschen in der Regel lediglich Informationen mit den Einheitentreibern aus. Es besteht zusätzlich die Möglichkeit, Aufrufe direkt auf Dateien zu beziehen. Dies ist aber auf die Codes 00H, 06H und 07H beschränkt. Andere Codes resultieren in einer Invalid-Function-Fehlermeldung.

4.65.1 IOCTL Data (Funktion 44H, Code 0 und 1, DOS 2.0-6.x)

Diese Unterfunktion erlaubt es, die interne MS-DOS Dateitabelle zur Devicekontrolle zu lesen oder zu verändern. Es gelten folgende Aufrufkonventionen.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    44H                    º
º AL:    0 Get Device Data      º
º        1 Set Device Data      º
º BX:    Handle                 º
º DX:    Device Daten           º
û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: nicht gesetzt          º
º DX:    Device Daten           º
Û-------------------------------ì
```

DX ist vor dem Aufruf nur zu setzen, falls AL = 1 ist.

Mit AL = 0 werden die Daten gelesen. Bei einem fehlerfreien Ablauf finden sie sich im Register DX. Im Register BX muß der Handle der Datei übergeben werden. Falls AL = 1 ist, werden die Daten aus dem Register DX in die interne DOS-Tabelle übertragen. Beim Schreiben muß das Register DH = 0 sein.

Die Kodierung der Devicedaten wurde im Laufe der verschiedenen DOS-Versionen erweitert. Als Einheiten werden Laufwerke (blockorientierte Einheiten) und Geräte wie LPT:, CON: etc. (zeichenorientierte Einheiten) unterschieden. Demnach hängt die Kodierung der Devicedaten von der Einheit ab. Bit 7 des Device-Words im Treiberkopf signalisiert, ob eine zeichenorientierte Einheit (Bit 7 = 1) oder einen blockorientierte Einheit (Bit 7 = 0) vorliegt.

Die in DX zurückgegebenen Devicedaten haben bei gesetztem Bit 7 folgendes Format:

Bit	Wert	Bemerkung
15	x	reserviert (wie Bit 7)
14	1	Das Device kann Kontroll-Zeichen bearbeiten, die mit der Funktion 44H Code 2 und 3 gesendet wurden. Das Bit kann nur gelesen aber nicht gesetzt werden.
13	x	reserviert
12	x	reserviert
11	1	Media not removable
10	x	reserviert
.	.	.
8	x	reserviert
7	1	Deviceflag = Gerät
6	0	End of File bei Input
5	1	keine Prüfung auf Kontroll-Zeichen (raw)
	0	Prüfung auf Kontroll-Zeichen (cooked)
4	1	Device nutzt INT 29H
3	1	Clock Device vorhanden
2	1	Null Device vorhanden
1	1	Console Output Device
0	1	Console Input Device

Tabelle 4.15: Kodierung der Device-Daten; bei zeichenorientierten Einheiten

Die Bits 8 bis 15 sind direkt aus dem Kopf des jeweiligen Gerätetreibers entnommen. Bit 15 hat in der Regel den gleichen Wert wie Bit 7. In einigen DOS-Versionen wird das Bit als reserviert markiert.

Bit 14 zeigt an, ob die Einheit Kontrollinformationen verarbeiten kann. Bit 14 = 0 bedeutet, die Einheit kann keine per Subfunktion AL = 02, 03, 04 oder AL = 05 übergebene Kontrollinformation bearbeiten. Ist das Bit gesetzt, dann lassen sich über die Aufrufe 4402H und 4403H Kontrollinformationen absetzen, die dann vom Treiber interpretiert werden. Das Bit läßt sich allerdings durch den Aufruf 4400H nicht setzen, da die Bearbeitung der Kontrollinformationen treiberspezifisch implementiert wird.

Die Bits 8 bis 13 sind bei vielen DOS-Versionen als reserviert markiert und werden ebenfalls aus dem Kopf des Treibers gelesen. Bit 13 = 1 bedeutet, daß der Treiber die Funktion 4410H unterstützt. Dies ist in MS-DOS 5.0/6.0 nur bei HIMEM.SYS und PRINTER.SYS der Fall.

Bit 12 ist seit DOS 3.1 definiert und markiert, daß das Gerät im Netzwerk betrieben wird (Bit 12 = 1), also nicht lokal ist.

Bit 11 = 1 bedeutet, daß der Treiber die Open- (440DH) und Close-(440EH) Funktionen unterstützt.

DIe Bits 9-10 sind meines Wissens unbenutzt (0). Bit 8 wird bei der Abfrage des Handles Nr. 1 benutzt, die Funktion ist aber unbekannt.

Bit 7 stammt aus dem Kopf des Treibers und selektiert zeichen- oder blockorientierte Einheiten. Falls Bit 7 gesetzt (1) ist, bezieht sich der Aufruf auf eine logische Einheit (Printer, Tastatur, etc.). Diese Einheit ist dem entsprechenden Handle zugeordnet. Für die Bits 0 - 6 gilt dann folgende Kodierung:

```
Bit 6 = 0  Die Einheit unterstützt die End-of-File Kennung
           beim Einlesen von Daten
        1  Es wird kein EOF unterstützt
Bit 5 = 0  Die Daten werden im ASCII-Mode transferiert, d.h.
           der String wird auf Steuerzeichen (CTRL-S, CTRL-P,
           CTRL-Z, CTRL-C) untersucht (cooked)
        1  Die Daten werden im Binär-Mode transferiert, d.h.
           es werden keine Steuerzeichen erkannt (raw). Der
           CTRL-C-Check muß zusätzlich auf OFF geschaltet
           sein (siehe INT 21, Funktion 33H)
Bit 3 = 1  CLOCK$ Einheit
Bit 2 = 1  NULL Einheit
Bit 1 = 1  Standard Ausgabe Einheit
Bit 0 = 1  Standard Eingabe Einheit
```

Bit 6 signalisiert im gelöschten Zustand, daß der Treiber bei Leseversuchen nur EOF zurückgibt (z.B. der Versuch von LPT: zu lesen). Bei gesetztem Bit kann die Einheit Daten zurückgeben.

Bit 5 definiert, ob der Treiber Steuerzeichen verarbeiten (cooked mode) oder ignorieren (raw mode) soll. Bei blockorientierten Einheiten ist immer der *raw mode* eingestellt. Bei zeichenorientierten Einheit ist der *cooked mode* vordefiniert.

Ist Bit 4 gesetzt, gibt der Treiber einzelne Zeichen über den INT 29 aus. Dies kann nur bei zeichenorientierten Geräten der Fall sein. In vielen DOS-Versionen ist dieses Bit als reserviert markiert.

Die Bits 0 bis 3 markieren, ob es sich bei dem Gerät um einen der DOS-Standardtreiber (CLOCK$, NUL, CON) handelt. Hier werden wieder die Bits des Treiberkopfes übernommen.

Falls Bit 7 = 0 ist, bezieht sich das Handle auf eine Datei und die restlichen Bits besitzen folgende Bedeutung:

Bit	Wert	Bemerkung
15	x	reserviert (Datei remote DOS 3.x)
14	0	Datum/Zeit nicht setzen (DOS 3.x)
13	x	reserviert
12	x	reserviert
11	1	Medium nicht wechselbar
10	x	reserviert
.	.	.
8	x	reserviert
7	0	Dateibit
6	0	Die Datei wurde nicht beschrieben
5	x	Laufwerknummer (0 = A:, 1 = B:, ...)
.	.	.
0	x	.

Tabelle 4.16: Kodierung der Device-Daten bei Dateien (Bit 7 = 0)

Die Bits 0-5 geben die Laufwerksnummer (0 = A:, 1 = B:, etc.), auf welches sich die Datei bezieht, an. Bit 6 wird durch Schreibzugriffe nach einem Open Aufruf gelöscht. Ist das Bit 6 = 0, wurde die Datei seit dem Öffnen der Datei noch nicht beschrieben. Die praktische Nutzung ist m.E. allerdings eingeschränkt. Die Bits 8-15 sind nach der DOS-Dokumentation als reserviert gekennzeichnet und werden aus dem Kopf des Treibers gelesen. Inoffiziell gilt aber folgende Belegung: Ein gesetztes Bit 11 signalisiert, daß das Medium nicht wechselbar ist. Bit 14 = 1 definiert, daß der Treiber beim Close einer Datei Datum und Uhrzeit nicht verändern soll. Bit 15 signalisiert ab DOS 3.0, daß die Datei im Netzwerk vorliegt (remote).

In der Praxis lassen sich die Funktionen 4400H und 4401H kaum nutzen. Über Bit 5 ist definierbar, ob die Daten vom Treiber im *raw mode* oder im *cooked mode* bearbeitet werden. Im *raw mode* werden Steuerzeichen ignoriert. Im *cooked mode* werden Zeichen wie Ctrl+A (Code 26H) als EOF interpretiert und ausgefiltert. Dies kann bei Graphikausgaben auf einen Drucker zu Problemen führen. Ein Programm könnte Bit 7 abfragen um festzustellen, ob die Ein- und Ausgaben der Standardeinheit von einem zeichenorientierten Treiber (CON) kommen, oder ob eine I/O-Umleitung zu Dateien vorliegt. Bit 2 erlaubt die Prüfung, ob ein Programm mit der Nulleinheit verbunden ist. Dies ist bei TSR-Programmen relevant, da hier die Standard-I/O-Handles oft unbenutzt bleiben und somit an DOS zurückgegeben werden können. Mit Bit 14 läßt sich bei Geräten testen, ob spezielle Treiber (PRINTER.SYS) geladen wurden.

Tritt bei der Operation ein Fehler auf, wird das Carry-Flag gesetzt und im Register AX findet sich ein Fehlercode:

```
Code ° Bemerkung
-----é-----------------------------------------------------------
 1   ° AL ist nicht 0 oder 1, oder DH ist nicht 0 bei AL = 1
 6   ° Der Handle in BX ist ungültig oder nicht geöffnet
15   ° ungültige Daten
```

Tritt kein Fehler auf, ist das Carry-Flag gelöscht. Erweiterte Fehlercodes lassen sich mit der Funktion 59H abfragen.

4.65.2 IOCTL Character (Funktion 44H, Code 2 und 3, DOS 2.0-6.x)

Diese Unterfunktionen erlauben es, Kontrollzeichen an eine zeichenorientierte Einheit (character device) zu senden oder zu empfangen. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
°      CALL:  INT 21            °
°                               °
° AH:    44H                    °
° AL:    2 Read CTRL- Character °
°        3 Write CTRL - Charact.°
° CX:    Zahl der transferierten°
°        Bytes                  °
° BX:    Handle                 °
° DS:DX  Zeiger auf Puffer      °
û-------------------------------À
°      RETURN                   °
° Carry: gesetzt -> Fehler      °
° AX:    Fehlercode             °
° Carry: nicht gesetzt          °
° AX:    Bytes transferiert     °
Û-------------------------------ì
```

Mit AL = 2 werden die Daten von der Einheit in den Puffer gelesen, mit AL = 3 werden sie zu der Einheit geschrieben. Im Registerpaar DS:DX findet sich ein Zeiger auf den

Datenpuffer. Im Register BX muß der Handle einer zeichenorientierten Einheit übergeben werden. Das Register CX gibt an, wieviele Bytes gelesen oder gesendet werden sollen. Nach einem fehlerfreien Ablauf, daß Carry-Flag ist nicht gesetzt, enthält AX die Zahl der transferierten Zeichen. Falls ein Fehler auftritt wird das Carry-Flag gesetzt und in AX findet sich ein Fehlercode.

```
Code ° Bemerkung
-----é-----------------------------------------------------
 1   ° AL ist nicht 2 oder 3, oder die Einheit kann nicht die
     ° aufgerufene Funktion ausführen.
 6   ° Der Handle in BX ist ungültig oder nicht geöffnet
13   ° Daten ungültig
```

Der Einheitentreiber muß so geschrieben sein, daß er IOCTL-Daten verarbeiten kann, andernfalls erfolgt eine Fehlermeldung.

4.65.3 IOCTL Block (Funktion 44H, Code 4 und 5, DOS 2.0 - 6.x)

Diese Unterfunktionen erlauben es, Kontrollzeichen an ein Blockdevice zu senden oder zu empfangen. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
°      CALL: INT 21             °
°                               °
° AH:    44H                    °
° AL:    4 Read CTRL-Charact.   °
°        5 Send CTRL-Charact.   °
° BL:    Laufwerksnummer        °
° CX:    Bytes zu transferie-   °
°        ren                    °
° DS:DX  Zeiger auf Puffer      °
û-------------------------------À
°      RETURN                   °
° Carry: gesetzt -> Fehler      °
° AX:    Fehlercode             °
° Carry: nicht gesetzt          °
° AX:    Bytes transferiert     °
°                               °
Û-------------------------------ì

 Kontrollzeichen senden

 Kontrollzeichen empfangen
```

Mit AL = 4 werden die Daten gelesen, mit AL = 5 werden sie gesendet. Im Registerpaar DS:DX findet sich der Zeiger auf den Zeichenpuffer. Im Register BL muß die Laufwerksnummer (0 = Default, 1 = A:, 2 = B:, etc.) angegeben werden. Das Register CX gibt an, wie viele Bytes gelesen oder gesendet werden sollen. Nach einem fehlerfreien Ablauf, daß Carry-Flag ist nicht gesetzt, enthält AX die Zahl der transferierten Zeichen.

Falls ein Fehler auftritt, wird das Carry-Flag gesetzt und in AX findet sich ein Fehlercode.

```
Code ° Bemerkung
-----é-----------------------------------------------------
 1   ° AL ist nicht 4 oder 5, oder die Einheit kann nicht die
     ° aufgerufene Funktion ausführen.
 5   ° Der Wert in BL spezifiziert keine gültige
     ° Laufwerksnummer
13   ° Daten ungültig
```

Der Device-Treiber muß so geschrieben sein, daß er IOCTL-Daten verarbeiten kann.

4.65.4 IOCTL Status (Funktion 44H, Code 6 und 7, DOS 2.0-6.x)

Diese Unterfunktion prüft, ob die per Handle selektierte Datei oder Treiber für eine Ausgabe bereit ist.

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    44H                    º
º AL:    6 Check Input Status   º
º        7 Check Output Status  º
º BX:    Handle                 º
û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: nicht gesetzt          º
º AL:    Statusbyte (00 oder FF)º
Û-------------------------------ì
```

Mit AL = 6 wird der Eingabestatus einer Einheit oder einer Datei abgefragt. Mit AL = 7 wird der Ausgabestatus des Device-Treibers geprüft. In BX muß die Handle-Nummer stehen. Es wird vorausgesetzt, daß die Einheit vorher mit Open eröffnet wurde (von dort wird der Handlecode übernommen).

Falls ein Fehler auftritt, ist anschließend das Carry-Flag gesetzt und im AX-Register findet sich ein Fehlercode:

```
Code º Bemerkung
-----é-----------------------------------------------------------
 1   º AL ist nicht 6 oder 7
 5   º Zugriff abgewiesen
 6   º Der angegebene Handle ist ungültig oder nicht eröffnet
 13  º ungültige Daten
```

Andernfalls ist das Carry-Flag gelöscht und im AL-Register findet sich ein Statusbyte mit folgender Bedeutung:

```
Wert   Device         Input File              Outputfile
-------------------------------------------------------
00H    Not ready      Pointer auf EOF         ready
FFH    ready          ready                   ready
```

Ein Ausgabefile gibt, bis auf den Ausnahmefall *Disk voll*, immer den Wert *ready* zurück.

Wird die Subfunktion 06H in DOS 4.0 auf eine Datei angewendet, gibt sie so lange den Wert F2H in AL zurück, bis das Dateiende (EOF) erreicht ist. Dann wird nur noch der Wert 00H zurückgegeben, bis die aktuelle Position mittels der Funktion 42H geändert wird.

4.65.5 IOCTL is Changeable (Funktion 44H, Code 8, DOS 3.0-6.x)

Diese Unterfunktion prüft, ob ein blockorientierter Einheitentreiber austauschbare Medien unterstützt. Es gelten die folgenden Übergabeparameter:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    44H                   º
º AL:    8 (Is Device Changable)º
º BL:    Laufwerksnummer       º
û------------------------------À
º      RETURN                  º
º Carry: gesetzt -> Fehler     º
º AX:    Fehlercode            º
º Carry: nicht gesetzt         º
º AX:    Statusbytes           º
Û------------------------------ì
```

Im Register BL wird die Nummer des Laufwerks angegeben (0 = Default, 1 = A:, 2 = B:, etc.).

Bei fehlerfreiem Aufruf, das Carry-Flag ist nicht gesetzt, enthält AX den Status :

```
0  Medium auswechselbar (Diskette oder Wechselplatte)
1  Medium fix (Festplatte)
```

Falls ein Fehler auftritt wird das Carry-Flag gesetzt und in AX findet sich ein Fehlercode.

```
Code º Bemerkung
-----é-----------------------------------------------------
  1  º Der Treiber unterstützt diese Abfrage nicht
 15  º Der Wert in BL ist keine gültige Laufwerksnummer
```

Falls der Fehlercode 1 zurückgegeben wird, kann das rufende Programm davon ausgehen, daß das Medium nicht wechselbar ist.

4.65.6 IOCTL is Redirected Block (Funktion 4409H, DOS 3.1-6.x)

Mit dieser Funktion läßt sich prüfen, ob ein angegebenes Laufwerk lokal zu einer Arbeitsstation innerhalb des Netzwerkes zugeordnet ist, oder ob es sich um ein Laufwerk eines Servers handelt, welches global verfügbar ist (remote).

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    44H                   º
º AL:    9 (Device local)      º
º BL:    Laufwerksnummer       º
û------------------------------À
º      RETURN                  º
º Carry: gesetzt -> Fehler     º
º AX:    Fehlercode            º
º Carry: nicht gesetzt         º
º DX:    Device Attribute Bits º
º                              º
Û------------------------------ì
```

Im Register BL muß die Laufwerksnummer (0 = Default, 1 = A:, 2 = B:, etc.) übergeben werden. Im DX-Register gibt die Funktion das Attributbyte der Einheit zurück. Dieses Register besitzt folgende Kodierung:

- Bei einer Remote-Einheit ist offiziell nur Bit 12 (1000H) benutzt, während die restlichen Bits = 0 sind.
- Bit 9 = 1 signalisiert in diesem Fall bei einigen DOS-Versionen, daß das Laufwerk von mehreren Netzwerkstationen benutzt wird. Dies wird über den INT 2A ermittelt.
- Bit 12 = 1 kennzeichnet ein Netzlaufwerk.

- Bit 15 = 1 bedeutet im Netzwerk, daß der Laufwerksbezeichner über SUBST erzeugt wurde.

Falls Bit 12 = 0 (lokal) ist, besitzt das Register folgende Kodierung:

- Bit 0 ist reserviert.
- Bit 1 = 1 signalisiert ab DOS 4.0, daß der Treiber Sektornummern mit 32 Bit unterstützt.
- Bit 2 .. 5 sind reserviert und gelöscht.
- Bit 6 = 1 definiert ab DOS 3.2, daß der Treiber die Funktionen 13H und 18H, b.z.w. die INT 21-Funktionen 440DH, 440EH und 440FH unterstützt.
- Bit 7 = 1 markiert ab DOS 5.0, daß der Treiber die Funktion 19H, b.z.w. die INT 21-Funktion 4411H unterstützt.
- Bit 8 wird von DIRVER.SYS benutzt, der Zweck ist aber unbekannt.
- Bit 9 = 1 markiert ein lokales Laufwerk, welches von anderen Stationen mit benutzt werden darf.
- Bit 10 ist reserviert.
- Bit 11 = 1 definiert ab DOS 3.0 das der Treiber die Funktionen 0DH und 0FH unterstützt und damit die INT 21-Funktion 4408H supported.
- Bit 12 = 0 kennzeichnet ein lokales Laufwerk.
- Bit 13 = 1 bedeutet, der Treiber erwartet beim Aufruf der Funktion 02H (Build BPB) ein neues Media-ID.
- Bit 14 = 1 bedeutet, der Treiber definiert die Funktionen 03H und 0CH und unterstützt damit die INT 21-Funktionen 4404H und 4405H.
- Bit 15 = 1 bedeutet, daß der Laufwerksbezeichner über SUBST erzeugt wurde.

Die Informationen in DX sind nur gültig, falls das Carry-Flag nicht gesetzt ist. Für Applikationsprogramme ist es gleich, ob ein Device *local* oder *remote* ist, da ein Zugriff immer möglich ist.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in AX findet sich der Fehlercode:

```
 Code ° Bemerkung
------é-----------------------------------------------------
   1  ° Filesharing muß erst geladen werden, bevor die
      ° Funktion
      ° aufgerufen werden kann.
  15  ° In BL ist ein ungültiges Laufwerk angegeben.
```

Mit der Funktion 59H lassen sich die erweiterten Fehlercodes abfragen.

4.65.7 IOCTL is Redirected Handle (Funktion 440AH, DOS 3.1-6.x)

Mit dieser Funktion läßt sich prüfen, ob ein angegebener Handle sich auf eine lokale Einheit (Datei oder Device) oder auf einen Server (remote) innerhalb eines Netzwerkes bezieht.

```
Ö-------------------------------Ì
º                               º
º      CALL:  INT 21            º
º                               º
º AH:    44H                    º
º AL:    0AH (Is Handle local)  º
º BX:    Handle                 º
û-------------------------------À
º                               º
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: nicht gesetzt          º
º DX:    IOCTL Bit Feld         º
º                               º
Û-------------------------------ì
```

Im Register BL muß die Laufwerksnummer (0 = Default, 1 = A:, 2 = B:, etc.) übergeben werden. Der Handlecode wird vor dem Aufruf in BX gesetzt. Im DX-Register gibt die Funktion das IOCTL-Bit-Feld zurück. Falls der Handle sich auf ein Remote-Device oder Remote-File bezieht, ist Bit 15 gesetzt (8000H).

Die Entscheidung, ob es sich um ein Gerät oder eine Datei handelt, erfolgt wie bei AX=4400H über Bit 7.

Die Informationen in DX sind nur gültig, falls das Carry-Flag nicht gesetzt ist. Für Applikationsprogramme ist es gleich, ob ein Device *local* oder *remote* ist, da ein Zugriff immer möglich ist.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in AX findet sich der Fehlercode:

```
 Code º Bemerkung
------é-------------------------------------------------------------
   1  º Das Netzwerk muß erst geladen werden, bevor die Funktion
      º aufgerufen werden kann.
   6  º In BL ist ein ungültiger Handle angegeben, oder der Handle
      º ist nicht geöffnet.
```

Mit der Funktion 59H lassen sich die erweiterten Fehlercodes abfragen. Bei Novell Netware 2.0 wird die Nummer des Fileservers, auf den sich das Handle bezieht, in CX zurückgegeben.

4.65.8 IOCTL Retry (Funktion 44H, Code 0BH, DOS 3.1-6.x)

Diese Funktion spezifiziert, wie oft MS-DOS eine Operation wie einen Diskzugriff wiederholt, falls ein Fehler vorkommt. Ein solcher Fehler kann z.B. dadurch auftreten, daß die Disk innerhalb des Netzwerkes bereits durch andere Teilnehmer belegt ist. Es gilt folgende Parameterversorgung:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    44H                   º
º AL:    0BH (Retry)           º
º DX:    Zahl der Wiederholung. º
º CX:    Wartezeit             º
û------------------------------À
º      RETURN                  º
º Carry: gesetzt -> Fehler     º
º AX:    Fehlercode            º
º Carry: nicht gesetzt         º
º ---    kein Fehler           º
Û------------------------------ì
```

Im Register DX muß die Zahl der Wiederholungen bei abgewiesenen Zugriffen stehen. MS-DOS versucht standardmäßig dreimal einen Zugriff, wenn das Device oder die Datei durch andere Netzwerkteilnehmer belegt sind (Share- oder Lock-Konflikt). Mit der Funktion 440BH läßt sich die Zahl der Zugriffe ändern. Falls der Zugriff nach n Versuchen nicht gelingt, gibt MS-DOS einen INT 24 aus, der von der Anwendersoftware abgefangen werden kann.

In CX findet sich die Wartezeit zwischen zwei Zugriffen. Der Effekt, den der Wert in CX auslöst, ist maschinenabhängig. Der Wert gibt nämlich an, wie oft eine Leerschleife auf der jeweiligen Maschine auszuführen ist. Der Schleifenzähler wird durch DOS standardmäßig auf den Wert 1 initialisiert. Daher variiert die Zeit, abhängig vom Prozessor und der Taktfrequenz.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in AX findet sich der Fehlercode:

```
 Code º Bemerkung
------é---------------------------------------
   1  º Filesharing muß erst geladen werden, bevor
     º die Funktion aufgerufen werden kann.
```

Mit der Funktion 59H lassen sich die erweiterten Fehlercodes abfragen.

4.65.9 Generic IOCTL Handle Request (Funktion 440CH, DOS 3.2-6.x)

Dieser Aufruf ist erst ab DOS 3.2 implementiert. Er erlaubt es, innerhalb eines Einheitentreibers verschiedene Unterfunktionen zu aktivieren, die Umschaltungen zwischen verschiedenen Zeichensätzen (Code Pages) unterstützen. Es gilt folgende Aufrufkonvention:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    44H                   º
º AL:    0CH (Generic IOCTL)   º
º BX:    Handle Nummer         º
º CH:    Hauptcode (Device)    º
º CL:    Untercode (Funktion)  º
º DS:DX  Zeiger auf Parameter  º
û------------------------------À
º      RETURN                  º
º Carry: gesetzt -> Fehler     º
º AX:  Fehlercode              º
º Carry: clear -> kein Fehler  º
º DS:DX Iteration Count        º
Û------------------------------ì
```

Ab MS-DOS 3.2 lassen sich bis zu 12 verschiedene Zeichensätze (Fonts) für die Ausgabetreiber verwenden. Mit dem Befehl DEVICE = COMMAND ... wird beim

Systemstart in CONFIG.SYS festgelegt, wieviele Fonts unterstützt werden. Durch die Aufrufe der Code-Page-Funktion lassen sich dann diese Zeichensätze installieren und aktivieren. Auch der INT 21 wurde mit einem Funktionsaufruf zur Installation und Aktivierung der Fonts ausgestattet.

Mit dem Aufruf 440CH wird diese Funktion aktiviert. Im Register BX wird der Handlecode einer geöffneten Einheit übergeben, auf die sich die folgenden Steueranweisungen beziehen. An Hand des Registers CX wird dann die gewünschte Aktion selektiert. Im CH-Teil findet sich der Hinweis auf die Art der jeweils angesprochenen Einheit (CON, COM, LPT). Es werden also nur zeichenorientierte Ausgabeeinheiten angesprochen, da nur hier die verschiedenen Zeichentypen Sinn ergeben. Der Registerteil CL spezifiziert, welche Unterfunktion ausgeführt werden soll. Es gilt dabei folgende Kodierung:

CH	Einheit
00	ungültig
01	COM x
03	CON
05	LPT x

CL	Unterfunktion
4CH	Prepare Start
4DH	Prepare End
4AH	Select
6AH	Query selected
6BH	Query prepare list
45H	Set Iteration Count
65H	Get Iteration Count
5FH	Set Display Inform. (DOS 4.0)
6FH	Get Display Inform. (DOS 4.0)

Tabelle 4.17: Kodierung der Code-Page-Unterfunktionen

Der Zeiger im Registerpaar DS:DX spezifiziert eine Tabelle, in der je nach Unterfunktion bestimmte Daten übergeben werden. Nachfolgend sollen nun die mit **CL** selektierten Unterfunktionen besprochen werden.

Set Iteration Count (CL = 45H)

Mit diesem Aufruf läßt sich dem Treiber der Wiederholungsfaktor übergeben. Dieser Wert spezifiziert, wie oft ein Treiber versucht, an die Einheit Daten zu senden, bis er den Zustand *device busy* annimmt. Der Parameterblock besitzt beim Aufruf folgenden Aufbau:

Offset	Byte	Bedeutung
00H	2	Wiederholungszähler für Ausgabe bei belegten Geräten

Die Funktion gibt folgende Fehlercodes zurück:

Code	Fehler
1	ungültige Subfunktion
5	Zugriff verweigert
6	Handle nicht definiert

Die Unterfunktion ist nach meinen Informationen nicht durch Microsoft dokumentiert.

Get Iteration Count (CL = 65H)

Mit diesem Aufruf läßt sich der Wiederholungsfaktor abfragen. Dieser Wert spezifiziert, wie oft ein Treiber versucht, an die Einheit Daten zu senden, bis er den Zustand *device busy* annimmt. Der Parameterblock besitzt beim Aufruf folgenden Aufbau:

Offset	Byte	Bedeutung
00H	2	Wiederholungszähler für Ausgabe bei belegten Geräten

Die Funktion gibt folgende Fehlercodes zurück:

Code	Fehler
1	ungültige Subfunktion
5	Zugriff verweigert
6	Handle nicht definiert

Die Unterfunktion ist nach meinen Informationen nicht offiziell durch Microsoft dokumentiert.

Prepare Start (CL = 4CH)

Mit diesem Aufruf wird der Treiber für n Zeichenfonts initialisiert. Die Zahl der Zeichenfonts wurde bereits beim Systemstart (CONFIG.SYS) spezifiziert und ist auf den Maximalwert von 12 begrenzt. Die durch das Registerpaar DS:DX adressierte Tabelle besteht aus mehreren Worten, die bei *Prepare Start* folgende Struktur besitzt:

Wert	Bedeutung
0	Flagregister
x	Länge in Bytes der folgenden Felder (n+1)*2
n	Zahl der reservierten »Code Pages« -1
	Code Page Nummer 1
.	.
-1	Code Page Nummer n

Tabelle 4.17: Datenstruktur bei CL = 4CH

Das Wort für die Flags ist beim Download mit dem Wert 0 zu initialisieren. Falls die verschiedenen Fonts hardwaremäßig (z.B. durch Cartridge-Kassetten) implementiert sind, ist der Wert 1 zu setzen. Die Zahl der zu installierenden *Code Pages* wird im dritten Wort spezifiziert (Maximalwert 12). Für jeden Zeichensatz schließt sich anschließend ein Statuswort an, welches mit dem Wert -1 zu initialisieren ist. Mit dem Wert -1 wird dem Einheitentreiber mitgeteilt, daß der eventuell bereits an dieser Position geladene Font nicht verändert wird. Soll ein anderer Font vorbereitet werden, muß an Stelle der -1 der jeweilige Code eingetragen werden. Da die Länge der Tabelle nicht implizit bekannt ist, wird im zweiten Wort die Zahl der Bytes, je zwei Byte pro Code Page und 2 Byte für die Code-Page-Zahl, übergeben. Bei 3 Code Pages steht hier also der Wert 8 (n+1)*2.

Um die zuletzt geladenen Fonts aufzufrischen, muß die Funktion 4CH mit den Werten -1 für alle Code Pages aufgerufen werden. Dies ist z.B. sofort nach dem Aufruf *Prepare End* erforderlich.

Tritt bei der Operation ein Fehler auf, wird bei der Rückkehr das Carry-Flag gesetzt. Mittels der Funktion 59H läßt sich dann der erweiterte Fehlercode abfragen. Es sind beim Aufruf *Prepare Start* folgende Fehlercodes implementiert:

```
Code ° Bedeutung
-----é-------------------------------------------------------
  01 ° ungültige Funktionsnummer
  22 ° unbekanntes Kommando
  27 ° die angegebene Code Page ist durch die Tastatur belegt
     ° oder es existiert kein File mit der entsprechenden Code
     ° Page
  29 ° Fehler in der Einheit
  31 ° Device Treiber besitzt keinen Font zum laden oder es
     ° wurde kein File mit dem Code Page gefunden (DOS 4.x)
```

Nur wenn das Carry-Flag nicht gesetzt ist, konnte die Funktion erfolgreich ausgeführt werden.

Refresh Font Um den Treiber dazu zu veranlassen, die aktuellen Fonts an das unterlagerte Gerät zu senden, ist die Prepare-Start-Funktion ebenfalls zu benutzen. Beim Aufruf muß jeder Code-Page-Eintrag in der Tabelle mit dem Wert -1 versehen werden. Dies ist z.B. sofort nach dem Aufruf *Prepare End* erforderlich.

Tritt bei der Operation ein Fehler auf, wird bei der Rückkehr das Carry-Flag gesetzt. Mittels der Funktion 59H läßt sich dann der erweiterte Fehlercode abfragen. Es sind beim Aufruf *Prepare Start* folgende Fehlercodes denkbar:

```
Code ° Bedeutung
-----é-----------------------------------------------
  27 ° Tastatur oder Code-Page-Konflikt
  29 ° Fehler in der Einheit
  31 ° Der Einheitentreiber besitzt keine Fontdaten
     ° die in die Einheit geladen werden können
```

Nur wenn das Carry-Flag nicht gesetzt ist, kann die Funktion erfolgreich ausgeführt werden.

Fonts laden

Nachdem die Funktion *Prepare Start* ausgeführt wurde, müssen die jeweiligen Fonts zum Einheitentreiber übertragen werden. Hierfür ist die bereits oben beschriebene Funktion 4403H (IOCTL Character) zu benutzen. Diese ermöglicht es, Kontrollzeichen an einen Treiber zu übergeben. Die Daten werden nun in den Treiber geladen, wobei das Format spezifisch für jede Einheit festgelegt ist. Die Übertragung ist mit dem weiter unten beschriebenen Aufruf *Prepare End* zu beenden.

Falls keine Daten übertragen werden, nimmt der Treiber an, daß die Fonts hardwaremäßig vorliegen. Dies ermöglicht es z.B., auswechselbare Zeichenfonts für Drucker mittels Cartridge-Kassetten zu unterstützen. Falls die Fonts hardwaremäßig fest vorgegeben sind, ist keine Prepare-Operation erforderlich.

Tritt bei der Operation ein Fehler auf, wird bei der Rückkehr das Carry-Flag gesetzt. Mittels der Funktion 59H läßt sich dann der erweiterte Fehlercode abfragen. Es sind beim Aufruf *Prepare Start* folgende Fehlercodes denkbar:

```
 Code ° Bedeutung
------é-------------------------------------------------------
  27  ° Einheitencode in der Fontdatei nicht gefunden, oder
      ° die Code-Page-Daten sind in der Datei nicht vorhanden
  29  ° Fehler in der Einheit
  31  ° Die Datei besitzt keine Fontdaten oder die Struktur
      ° der Datei ist zerstört
```

Nur wenn das Carry-Flag nicht gesetzt ist, kann die Funktion erfolgreich ausgeführt werden.

Prepare End (CL = 4DH)

Dieser Aufruf wird benutzt, um nach einem *Prepare Start* mit anschließendem Laden der Fontdaten über die Funktion 4403H dem Treiber zu signalisieren, daß die Initialisierung beendet ist. Der Aufruf erfolgt in der oben bereits beschriebenen Art mit der gleichen Registerstruktur. Lediglich der Zeiger DS:DX wird nicht mehr an den Anfang der Tabelle, sondern auf das Feld mit der Zahl der Code Pages positioniert. Damit hat die Tabelle nur noch folgenden Aufbau:

Byte	Bedeutung
2	Länge des Datenbereichs in Bytes
2	Code Page ID

Tabelle 4.18: Datenstruktur bei CL = 4DH, 4AH, 6AH

Tritt bei der Operation ein Fehler auf, wird bei der Rückkehr das Carry-Flag gesetzt. Mittels der Funktion 59H läßt sich dann der erweiterte Fehlercode abfragen. Es werden beim Aufruf *Prepare End* folgende Fehlercodes zurückgegeben:

Code	Bedeutung
1	Funktion unbekannt
6	Handle nicht definiert
19	Falsche Daten aus der Fontdatei gelesen
31	Kein Prepare Start ausgeführt

Nur wenn das Carry-Flag nicht gesetzt ist, konnte die Funktion erfolgreich ausgeführt werden.

Select (CL = 4AH)

Dieser Aufruf wird benutzt, um nach einem *Prepare Start* mit anschließendem Laden der Fontdaten über 4403H die Code Pages zu selektieren. Der Aufruf erfolgt in der bereits beschriebenen Weise mit der Ausnahme das der Zeiger DS:DX auf das Feld mit der Zahl der Code Pages positioniert wird. Damit hat die Tabelle folgenden Aufbau:

Byte	Bedeutung
2	Länge des Datenbereiches in Byte
2	Code Page ID
2n	DBCS lead byte range (DOS 4.0)
2	0000H Abschluß Datenbereich (DOS 4.0)

Tabelle 4.19: Datenstruktur bei CL = 4AH, 6AH

Hinter der Code Page ID findet sich ein Feld mit n Worten, in dem der Wertebereich für das erste Byte der DBCS (Double Byte Character Set) spezifiziert wird. Diese DBCS werden zum Beispiel zur Darstellung von Kanji-Schriftzeichen benutzt. Deshalb unterstützen nur Treiber mit der asiatischen Version von DOS 4.x diesen Aufruf.

Tritt bei der Operation ein Fehler auf, wird bei der Rückkehr das Carry-Flag gesetzt. Mittels der Funktion 59H läßt sich dann der erweiterte Fehlercode abfragen. Es sind beim Aufruf *select* folgende Fehlercodes belegt:

```
Code ° Bedeutung
-----é-------------------------------------------------
   1 ° Funktion unbekannt
   6 ° Handle nicht definiert
  26 ° Die Code Pages sind nicht initialisiert
  27 ° Der selektierte Tastaturtreiber unterstützt diese
     ° Code Page nicht
  29 ° Fehler in der Einheit
  65 ° Gerät läßt sich nicht auf die angegebene Codeseite umschalten
```

Nur wenn das Carry-Flag nicht gesetzt ist, kann die Funktion erfolgreich ausgeführt werden.

Query Selected (CL = 6AH)

Mit diesem Aufruf lassen sich die selektierten Code Pages abfragen. Der Aufruf erfolgt in der bereits beschriebenen Weise mit der Ausnahme, daß der Zeiger DS:DX auf das Feld mit der Zahl der Code Pages positioniert wird. Damit hat die Tabelle folgenden Aufbau:

```
  Byte ° Bedeutung
-------é---------------------------------------
   2   ° Länge des Datenbereiches in Byte
   2   ° Code Page ID
  2n   ° DBCS lead byte range (DOS 4.0)
   2   ° 0000H Abschluß Datenbereich (DOS 4.0)
```

Tabelle 4.20: Datenstruktur bei CL = 6AH

Tritt bei der Operation ein Fehler auf, wird bei der Rückkehr das Carry-Flag gesetzt. Mittels der Funktion 59H läßt sich dann der erweiterte Fehlercode abfragen. Es sind beim Aufruf *Query Selected* folgende Fehlercodes denkbar:

```
Code ° Bedeutung
-----é-----------------------
   1 ° Funktion unbekannt
   6 ° Handle nicht definiert
  26 ° Keine Page selektiert
  27 ° Device-Fehler
```

Nur wenn das Carry-Flag nicht gesetzt ist, konnte die Funktion erfolgreich ausgeführt werden.

Query Prepare List (CL = 6BH)

Mit diesem Aufruf lassen sich die eingestellten Code Pages abfragen. Der Aufruf erfolgt in der oben beschriebenen Weise. Allerdings muß die im Registerpaar DS:DX adressierte Tabelle eine erweiterte Struktur besitzen. Neben den maximal 12 ladbaren Fonts kann das System bis zu 12 weitere Hardwarefonts unterstützen. Die Tabelle muß so bemessen sein, daß sie maximal alle Daten (27 Wörter) aufnehmen kann. Diese Daten besitzen folgende Struktur:

```
 Wert ° Bedeutung
-------é-------------------------------------------------
```

*x ° Länge in Bytes der folgenden Felder ((n+1)+(m+1))*2*
n ° Zahl der Hardware Code Pages (max. 12)
*-1 ° Hardware »Code Page**Fehler! Verweisquelle konnte nicht gefunden werden.** . ° .*
*-1 ° Hardware »Code Page**Fehler! Verweisquelle konnte nicht gefunden werden.** m ° Zahl der ladbaren Code Pages (max. 12)*
*-1 ° geladene »Code Page**Fehler! Verweisquelle konnte nicht gefunden werden.** . ° .*
*-1 ° geladene »Code Page**Fehler! Verweisquelle konnte nicht gefunden werden.**Tabelle 4.21: Datenstruktur bei CL = 6BH*

Tritt bei der Operation ein Fehler auf, wird bei der Rückkehr das Carry-Flag gesetzt. Mittels der Funktion 59H läßt sich dann der erweiterte Fehlercode abfragen. Es sind beim Aufruf *Query Prepare List* folgende Fehlercodes denkbar:

Code	Bedeutung
1	Funktion unbekannt
6	Handle nicht definiert
26	Keine Code Pages selektiert
29	Fehler in der Einheit

Nur wenn das Carry-Flag nicht gesetzt ist, kann die Funktion erfolgreich ausgeführt werden.

Get/Set Display Informationen (CL=5FH oder 7FH)

Ab DOS 4.0 wurde der Aufruf AX=440CH um zwei weitere Funktionen erweitert. Die in Tabelle 4.17 angegebenen Code-Page-Unterfunktionen (CL) sind nur für die LPT-Einheit (CH = 5) gültig. Ab DOS 4.0 wurden für die CON-Einheit (CH = 3) zwei weitere Subfunktionen eingeführt.

CH	Einheit	CL	Unterfunktion
3	CON	5FH	Set display information
		7FH	Get display information

Die beiden Aufrufe lesen oder setzen die Display-Informationen der Standard-Ausgabeeinheit. Das Registerpaar DS:DX zeigt dabei auf einen Parameterblock mit folgender Struktur:

Offset	Byte	Feld
00	1	Info Level (immer 0)
01	1	reserviert (immer 0)
02	2	Restlänge Parameterblock (14)
04	2	Control-Flag Bit 0 = 0 Vordergrundfarbe 1 Blinken
06	1	Video Mode 1=Text Mode, 2 APA-Mode
07	1	reserviert (immer 0)
08	2	Zahl der Farben (Mono = 0)

Offset	Byte	Feld
0A	2	Display-Höhe in Pixeln (APA-Mode)
0C	2	Display-Breite in Pixeln (APA-Mode)
0E	2	Display-Höhe in Zeichen
10	2	Display-Breite in Zeichen

Die Informationen in diesem Parameterblock beeinflussen die Darstellung der Standard-Ausgabeeinheit. Bei der Leseoperation muß im Anwenderprogramm ein genügend großer Buffer zur Aufnahme des Parameterblocks reserviert werden.

4.65.10 Generic IOCTL Request (Funktion 44H, Code 0DH, DOS 3.2-6.x)

Dieser Aufruf ist erst ab DOS 3.2 implementiert. Er erlaubt es, innerhalb eines blockorientierten Einheitentreibers verschiedene Unterfunktionen zu aktivieren. Der Aufruf erfolgt mit folgenden Registerwerten:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    44H                    º
º AL:    0DH (Generic IOCTL)    º
º BL:    Laufwerknummer         º
º CH:    Hauptcode immer 08H    º
º CL:    Untercode (Funktion)   º
º DS:DX  Zeiger auf Parameter-  º
º        block                  º
û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º        clear -> kein Fehler   º
Û-------------------------------ì
```

Im Register AX steht immer der Wert 440DH. Das logische Laufwerk wird in BL spezifiziert (0 = Default, 1 = A:, 2 = B:, etc.). Das Register CX enthält den Steuercode für die jeweilige Unterfunktion. Das Register CH wird bei allen Aufrufen einheitlich mit dem Wert 08H initialisiert. Im Register CL findet sich dann ein Untercode, der die jeweilige Teilfunktion spezifiziert. In Abhängigkeit von der DOS-Version werden folgende Funktionen unterstützt:

```
Code º Funktion
-----é-----------------------------------------------
 40H º Set Device Parameters
 60H º Get Device Parameter
 41H º Write Track on a logical Device
 61H º Read Track on a logical Device
 42H º Format and Verify Track on a logical Device
 62H º Verify Track on a logical Device
 47H º Set Access Flag (DOS 4.0)
 67H º Get Access Flag (DOS 4.0)
 46H º Set Media ID (DOS 4.0)
 66H º Get Media ID (DOS 4.0)
 68H º Sense Media Type (DOS 5.0)
```

Das Registerpaar DS:DX enthält einen Zeiger auf einen Parameterblock. Dessen Struktur ist abhängig von der selektierten Unterfunktion.

Get / Set Device Parameters (CL = 60H / 40H)

Mit diesen Unterfunktionen lassen sich die Parameter einer Einheit lesen oder setzen. Bei Get (CL = 60H) und Set (CL = 40H) gilt der gleiche Aufbau des Parameterblocks:

```
 Bytes º  Bedeutung
-------é--------------------------------
   1   º  Bitfeld für Spezialfunktionen
   1   º  Einheitentyp
   1   º  Attribute des Treibers
   2   º  Zahl der Zylinder (Platte)
   1   º  Typ des Speichermediums
   x   º  Parameterblock der Einheit
   x   º  Spuraufbau der Einheit
```

Tabelle 4.22: Aufbau des Parameterblocks bei 440DH (CL = 40H oder 60H)

Die Datenfelder *Parameterblock der Einheit* und *Spuraufbau der Einheit* werden nachfolgend aufgeschlüsselt.

Bitfeld für Spezialfunktionen

Mit dem ersten Byte (Spezialfunktionen) lassen sich die Get- und Set-Funktionen steuern. Es gelten folgende Bedingungen:

Bei der Get-Operation (CL = 60H) ist nur Bit 0 definiert, alle anderen Bits sind beim Aufruf zu löschen. Mittels des Bits wird der zurückgegebene BIOS-Parameter-Block (BPB) selektiert:

Bit 0 :

1: Es wird der Build-BIOS-Parameter-Block zurückgegeben

0: Der Standard-BIOS-Parameter-Block wird zurückgegeben

Bei der Set-Operation (CL = 40H) sind die Bits 0, 1 und 2 belegt. Es gelten folgende Bedingungen:

Bit 0 :

1: Alle nachfolgenden Abfragen des *Build BPB* geben den *Device BPB* zurück. Wird ein weiteres Set Device Parameter mit Bit 0 = 0 abgesetzt, dann ist bei der Build-BPB-Kommando-Abfrage das aktuelle *Medium BPB* zurückzugeben.

0: Das Device-BPB-Feld enthält eine neue Standard Einstellung für die Einheit. Falls ein vorhergehender Set-Aufruf das Bit gesetzt hatte, wird die aktuelle Einstellung des *Media BPB* zurückgegeben. Andernfalls wird die Standard-BPB-Einstellung des Treibers bei der Build-BPB-Abfrage zurückgegeben.

Bit 1 :

1: Alle Felder des Parameterblocks, bis auf das Feld *Spuraufbau der Einheit*, sind zu ignorieren.

0: Alle Felder des Parameterblocks sind auszuwerten.

Bit 2 :

1: Alle Sektoren einer Spur besitzen die gleiche Länge. Weiterhin liegen alle Sektornummern zwischen 1 und n.

0: Die Sektoren einer Spur besitzen unterschiedliche Längen.

Die Bits 0 und 1 schließen sich gegenseitig aus, d.h., sie können nicht gleichzeitig gesetzt werden. Bei einem Set-Aufruf sollte Bit 2 immer gesetzt werden, da dann das Formatieren einer Spur wesentlich schneller erfolgen kann.

Einheitentyp

Das zweite Byte beschreibt die Art der physikalisch angeschlossenen Einheit. Es gilt folgende Kodierung:

Wert	Einheitentyp
0	320/360 Kbyte 51/4-Zoll-Diskette
1	1,2 Mbyte 51/4-Zoll-Diskette
2	720 Kbyte 31/2-Zoll-Diskette
3	8 Zoll einfache Dichte
4	8 Zoll doppelte Dichte
5	Festplatte
6	Bandlaufwerk
7	1,44 Mbyte-Laufwerk
8	2,88 Mbyte-Laufwerk
9	Anderes Speichermedium

Tabelle 4.23: Kodierung der Speichermedien

Diese Daten lassen sich lediglich über die Get-Funktion abfragen.

Attribute des Treibers

Dieses Feld umfaßt ein Byte, welches die Attribute der Einheit spezifiziert:

Bit 0:

1: Das Speichermedium ist nicht wechselbar.

0: Das Speichermedium ist auswechselbar.

Bit 1:

1: Ein Diskettenwechsel wird vom Treiber unterstützt.

0: Ein Diskettenwechsel wird nicht unterstützt.

Die Attribute lassen sich nicht setzen, da sie im Treiber definiert sind. Die restlichen Bits sind nicht definiert.

Zahl der Zylinder

In diesem Feld wird die maximale Zahl der durch den Treiber unterstützten Zylinder angegeben. Insbesondere bei Festplatten ist die Speicheroberfläche in mehrere Zylinder mit getrennten Schreib-Lese-Köpfen unterteilt. Der Wert in dem Feld ist unabhängig vom Speichermedium. Die Informationen lassen sich lediglich abfragen, da innerhalb des Treibers die Zahl der unterstützten Zylinder fest implementiert ist.

Typ des Speichermediums

Dieses Feld ist zur Unterstützung von Diskettenlaufwerken erforderlich. Spätestens seit der Einführung der 1,2-Mbyte-Disketten unterstützen die Laufwerke mehrere Formate. In dem Feld wird daher angegeben, ob eine 1,2-Mbyte- oder 360-Kbyte-Diskette im $5^1/_4$-Zoll-Laufwerk eingelegt ist. Es gilt folgende Kodierung:

0: Quad density (96 tpi) Disk mit 1,2 Mbyte

1: Double density (48 tpi) Disk mit 320/360 Kbyte

Als Standardeinstellung für diese Laufwerke wird eine 1,2-Mbyte-Diskette angenommen. Das Feld wird allerdings nur benutzt, wenn das aktuelle Speichermedium innerhalb des Laufwerkes nicht anders bestimmt werden kann.

Parameterblock der Einheit

Hier handelt es sich um eine komplette Datenstruktur, die alle Informationen über den Typ des Speichermediums sowie seine logische Aufteilung für MS-DOS enthält. Die Struktur besitzt folgenden Aufbau:

Bytes	Bedeutung
2	Bytes pro Sektor
1	Sektoren pro Cluster
2	Zahl der reservierten Sektoren
1	Zahl der File-Allocation-Tabellen (FAT)
2	Zahl der Einträge im Hauptverzeichnis
2	Gesamtzahl aller Directory-Einträge
1	Type des Speichermediums
2	Sektoren pro FAT
2	Sektoren pro Spur
2	Zahl der Köpfe
4	Zahl der unsichtbaren (hidden) Sektoren
4	reserviert 1
6	reserviert 2

Tabelle 4.24: Aufbau des Device-BIOS-Parameter-Blocks (BPB)

Abhängig vom Bit 0 im Spezialfunktionsfeld wird der BPB oder der Build BPB zurückgegeben. Die Bedingungen wurden in dem entsprechenden Absatz erläutert.

Spuraufbau der Einheit

Bei diesem Feld handelt es sich um eine Tabelle mit variabler Länge, die den Spuraufbau innerhalb des Speichermediums angibt. MS-DOS besitzt nur eine globale Beschreibungstabelle für alle Laufwerke. Falls sich die Attribute eines Mediums ändern, muß mit dem Set-Parameter-Befehl diese Tabelle aktualisiert werden. Die Tabelle wird unabhängig von der Spezifikation des Spezialfunktionsfeldes gesetzt. Beim Aufruf *Get Parameter* wird das Feld nicht benutzt.

Lediglich die Schreib-Lese-, Format- und Verify-Aufrufe benutzen diese Tabelle. Diese besteht aus n Worten und besitzt folgenden logischen Aufbau:

Wert	Bedeutung
n	Zahl der Sektoren in der Tabelle
1	Sektor Nummer 1
200H	Sektorlänge in Bytes
.	.
n	Sektor Nummer n
200H	Sektorlänge in Bytes

Tabelle 4.25: Aufbau des Track-Layout-Feldes

Für jeden Sektor einer Spur werden 2 Wörter in der Tabelle reserviert. Im ersten findet sich die Nummer des Sektors, während das zweite Wort die Sektorlänge angibt. Standardmäßig setzt MS-DOS die Länge auf 512 Byte (200H) pro Sektor. Die Zahl der Sektoren innerhalb der Tabelle wird im ersten Wort angegeben. Die Einträge für die Sektornummer dürfen demnach nicht größer als die Zahl der Sektoren in der Tabelle sein. Falls im Spezialfunktionsfeld das Bit 2 gesetzt ist, müssen die Werte für die Sektorlänge gleich sein.

Generell sollten die Einstellwerte für den Laufwerkstyp, die Attribute der Einheit und die Zahl der Köpfe nur dann geändert werden, wenn das physikalische Laufwerk gewechselt wurde.

Die Funktion besitzt folgende Fehlercodes:

```
 Code Fehler
   1  Funktion nicht definiert
  15  Laufwerk existiert nicht
```

Read / Write Track; on Logical Device (440DH CL = 41H und CL = 61H)

Mit diesen beiden Unterfunktionen lassen sich ganze Spuren einer logischen Einheit lesen oder schreiben. Die Unterfunktion wird durch den Wert in CL gesteuert:

```
CL = 61H Lese Spur
CL = 41H Schreibe Spur
```

Der Aufruf der Funktion erfolgt mit den gleichen Parametern wie bereits oben beschrieben wurde. Lediglich der Wert für das Register CL muß angepaßt werden. Außerdem besitzt der durch das Registerpaar DS:DX adressierte Parameterblock den folgenden Aufbau:

```
Bytes ° Bedeutung
------é------------------------------
  1   ° Bitfeld für Spezialfunktionen
  2   ° Kopfnummer
  2   ° Zylindernummer
  2   ° Erster Sektor
  2   ° Zahl der Sektoren
  4   ° Zeiger auf den Transferpuffer
```

Tabelle 4.26: Aufbau des Parameterblocks bei 440DH CL = 41H oder 61H

Das erste Byte muß auf den Wert 0 (Reset Bits) gesetzt werden. Der Kopf gibt an, auf welcher Oberfläche die Schreib-Lese-Operation stattfinden soll. Der Wert des Zylinders wird von 0 gezählt, d.h. für den zweiten Zylinder muß eine 1 eingetragen werden. Der erste Sektor, ab dem geschrieben werden soll, wird in dem Wort *erster Sektor* spezifiziert. Der Wert wird ebenfalls von 0 an gezählt. Das nächste Feld gibt an, wie viele Sektoren insgesamt zu schreiben oder zu lesen sind. Der Block wird abgeschlossen mit einem Zeiger auf den Schreib-Lese-Puffer (DTA). Der Puffer muß im rufenden Prozeß reserviert werden. Ein Schreib-Lese-Zugriff über die 64-Kbyte-Grenze ist nicht möglich. Es kann auch nur eine komplette Spur gelesen oder geschrieben werden.

Falls während der Operation Fehler auftreten, Carry-Flag gesetzt, lassen sich diese über die Funktion 59H abfragen. Hierbei sind folgende Fehler definiert:

```
 Code Fehler
   1  Funktion nicht definiert
  15  Laufwerk existiert nicht
```

Format / Verify Track on Logical Device (440DH CL = 42H und CL = 62H)

Mit dieser Unterfunktion lassen sich einzelne Spuren formatieren oder verifizieren. Die Aktion wird über den Wert des CL Registers gesteuert:

```
CL = 42H Formatiere und verifiziere die Spur
CL = 62H Verifiziere die Spur
```

Es gelten die gleichen Aufrufparameter wie bei den anderen Unterfunktionen. Der Wert für CL ist anzupassen und der durch DS:DX adressierte Parameterblock besitzt folgenden Aufbau:

Bytes	Bedeutung
1	Bitfeld für Spezialfunktionen
2	Kopfnummer
2	Zylindernummer

Tabelle 4.27: Aufbau des Parameterblocks bei 440DH CL = 42H/62H

Beim Aufruf wird Bit 0 des Spezialfunktionsfeldes ausgewertet. Es gilt folgende Nomenklatur:

Bit 0:

1 = Format-Status-Check

0 = Format/Verify-Start

Mit der Abfrage des Format-Status läßt sich prüfen, ob eine Kombination *Spuren* und *Sektoren pro Spur* bei der Format- und Verify-Operation unterstützt wird. Hierzu ist vorher ein Set-Device-Parameter-Aufruf mit einem korrekten BPB abzusetzen. Der Treiber gibt dann bei der Abfrage des Format-Status im Spezialfunktionsfeld zurück:

Bit

0: Die Funktion wird durch das ROM-BIOS unterstützt. Der Zugriff auf einzelne Spuren und die Angabe *Sektoren pro Spur* ist für Diskettenlaufwerke erlaubt.

1: Die Funktion wird nicht durch das ROM-BIOS unterstützt.

2: Die Funktion wird durch das ROM-BIOS unterstützt. Ein Zugriff über *Spuren* und *Sektoren pro Spur* ist bei Diskettenlaufwerken nicht erlaubt.

3: Die Funktion wird durch das ROM-BIOS unterstützt, aber der Treiber kann die Zahl der Spuren und die Zahl der Sektoren pro Spur nicht ermitteln, da das Diskettenlaufwerk leer ist.

Um eine Spur zu formatieren, sind folgende Schritte notwendig:

- Aufruf der Set-Parameter-Funktion, um die Formatparameter zu übergeben.
- Aufruf des Format-Status-Check, um die Zahl der Spuren des Laufwerks und die Zahl der Sektoren pro Spur zu überprüfen.
- Aufruf der Format- und Verify-Funktion mit Bit 0 im *Spezialfunktionsfeld = 0*, um die einzelnen Spuren zu bearbeiten.

Ab DOS 4.0 gibt der Aufruf folgende Informationen im Parameterblock zurück:

```
Bit 0:     1 falls die spezifizierte Spur unterstützt wird
    1:     1 keine BIOS-Unterstützung
    2:     1 Spur, Sektor/Spur wird nicht unterstützt
    3:     1 keine Disk im Laufwerk
    Word:  Zahl der Köpfe
    Word:  Zahl der Zylinder
```

Die Informationen im Byte 0 (Flags) stimmen mit den Daten von DOS 3.x überein.

Get/Set Access Flag Status (440DH, CL = 67H und CL = 47H)

Diese Unterfunktion ist erst ab DOS 4.0 implementiert. Mit dem Wert in CL wird die jeweilige Funktion selektiert:

```
CL = 67H: Get Access Flag Status
CL = 47H: Set Access Flag Status
```

Über den Zeiger DS:DX wird ein Parameterblock mit folgender Struktur adressiert:

```
Ö------Ú-----------------------------------Ì
º Byte º Feld                              º
û------é-----------------------------------À
º   0  º Spezialfunktionsfeld               º
û------é-----------------------------------À
º   1  º Disk Access Flag                   º
Û------Ù-----------------------------------ì
```

Solange ein Medium nicht formatiert wurde oder einen ungültigen Bootrecord enthält, sperrt das System jeden Zugriff per I/O-Funktion. Das Disk-Access-Flag weist dann den Wert 0 auf. Der Formataufruf über die Funktion CL = 47H setzt den Wert des Flags ungleich 0, sobald das Medium erfolgreich formatiert wurde. Im Fehlerfall bleibt das Flag = 0, um weitere Zugriffe zu blockieren. Dadurch läßt sich die Datenintegrität einer Platte leicht gewährleisten. Die Funktion definiert folgende Fehlercodes:

```
Code  Fehler
  1   Funktion nicht definiert
  5   Zugriff verweigert
 15   Laufwerk existiert nicht
```

Get/Set Media ID (440DH, CL = 66H und CL = 46H)

Diese Unterfunktionen sind erst ab DOS 4.0 implementiert. Sie erlauben es, die Seriennummer, den Typ des Dateisystems, sowie das Volumelabel im Boot-Record des Speichermediums zu setzen oder abzufragen. Mit dem Wert in CL wird die jeweilige Funktion selektiert:

```
CL = 66H: Get Media ID
CL = 46H: Set Media ID
```

Über den Zeiger DS:DX wird ein Parameterblock mit folgender Struktur adressiert:

```
Ö------Ú-----------------------------------Ì
º Byte º Feld                              º
û------é-----------------------------------À
º   2  º Info-Level (immer 0)               º
û------é-----------------------------------À
º   4  º Seriennummer                       º
û------é-----------------------------------À
º  11  º Media Namen                        º
û------é-----------------------------------À
º   8  º Typ des Dateisystems               º
Û------Ù-----------------------------------ì
```

Das erste Word sollte immer mit dem Wert 0 belegt werden. Ab Offset 02H folgt eine 4-Byte-Konstante mit der Seriennummer des Datenträgers. Daran schließt sich ein Feld mit 11 Zeichen an, welches den im Bootrecord einzutragenden Medium Namen enthält. Abgeschlossen wird die Struktur durch ein 8 Byte langes Feld mit dem Namen des Dateisystems (z.B. 'FAT12 ', 'FAT16 '). Fehlende Zeichen sind mit Blanks aufzufüllen.

Beim Get-Aufruf werden die Daten aus dem Boot-Record des Mediums gelesen und in obiger Datenstruktur zurückgegeben. Beim Set-Aufruf sind die Daten in obiger Struktur zu definieren. Sie werden anschließend in den Boot-Record ausgelagert.

Die Funktion definiert folgende Fehlercodes:

```
Code  Fehler
  1   Funktion nicht definiert
  5   Zugriff verweigert
 15   Laufwerk existiert nicht
```

Sense Media Type (440DH, CL = 68H)

Diese Unterfunktionen sind erst ab DOS 5.0 implementiert. Sie ermittelt, welches Format ein Speichermedium hat. Mit dem Wert CL = 68H wird die Funktion selektiert. Über den Zeiger DS:DX wird ein Parameterblock mit folgender Struktur adressiert:

```
Ö------Ú----------------------------------Ì
º Byte º Feld                            º
û------é----------------------------------À
º   1  º Format Type                       º
û------é----------------------------------À
º   1  º Size Type                         º
Û------Ù----------------------------------ì
```

Das erste Byte gibt an, ob es sich um ein Standard-DOS-Format (Wert = 0) oder um ein nicht standardisiertes Format (Wert = 1) handelt. Im folgenden Byte findet sich dann eine Information über die Kapazität des Speichermediums. Hierbei gilt folgende Kodierung:

```
Code Kapazität
02H  720 Kbyte
07H  1,44 Mbyte
09H  2,88 Mbyte
```

Die Funktion ist durch Microsoft nicht komplett dokumentiert, dürfte aber folgende Fehlercodes zurückgeben:

```
Code  Fehler
  1   Funktion nicht definiert
  5   Zugriff verweigert
 15   Laufwerk existiert nicht
```

4.65.11 I/O Control for Device (Funktion 440EH, DOS 3.2-6.x)

Dieser Aufruf ist erst ab DOS 3.2 implementiert. Er prüft, ob eine logische Einheit einer blockorientierten Einheit zugeordnet ist. Es gelten nebenstehende Übergabeparameter.

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    44H                   º
º AL:    0EH (I/O Control)     º
º BL:    Laufwerknummer        º
û------------------------------À
º                              º
º      RETURN                  º
º Carry: gesetzt -> Fehler     º
º AX:    Fehlercode            º
º Carry: clear -> kein Fehler  º
º AL:    Laufwerke             º
º                              º
Û------------------------------ì
```

Im Register BL wird die Nummer des gewünschten Laufwerkes angegeben (0 = Default, 1 = A:, 2 = B:, etc.). Falls kein Fehler auftritt, das Carry-Flag ist nach dem Aufruf gelöscht, enthält das Register AL die Nummer des letzten zugeordneten logischen Laufwerkes (1 = A:, 2 = B:, etc.). Falls nur eine Einheit zugeordnet ist, enthält AL den Wert 0.

Bei Systemen mit einem Diskettenlaufwerk sind diesem z.B. in der Regel die logischen Laufwerkskennbuchstaben A: und B: zugeordnet. Beim Aufruf wird deshalb das AL-

Register mit dem Wert des ersten Laufwerks belegt (1 = A:). Bei der Rückkehr enthält AL den Wert der letzten zugeordneten Einheit (2 = B:), d.h., dem Laufwerk sind die logischen Einheiten A: bis B: zugeordnet.

Tritt während des Aufrufes ein Fehler auf, ist das Carry-Flag gesetzt. Im Register AX wird dann ein Fehlercode mit folgender Bedeutung zurückgegeben.

```
Code  Fehler
  1   Funktion nicht definiert
 15   Laufwerk existiert nicht
```

Die Fehlerursache ist bei den Extended-DOS-Errors beschrieben und läßt sich auch durch die Funktion 59H abfragen.

4.65.12 Set Logical Drive (Funktion 440FH, DOS 3.2 - 6.x)

Auch dieser Aufruf ist erst ab DOS 3.2 implementiert. Er erlaubt es, einer blockorientierten Einheit den nächsten logischen Laufwerksbuchstaben zuzuordnen. Es gelten folgende Aufrufkonventionen:

```
 Laufwerksbuchstaben zuordnen;

Ö--------------------------------Ì
º      CALL:  INT 21             º
º                                º
º AH:    44H                     º
º AL:    0FH                     º
º BL:    Laufwerknummer          º
û--------------------------------À
º      RETURN                    º
º Carry: gesetzt -> Fehler       º
º AX:    Fehlercode              º
º Carry: clear -> kein Fehler    º
º AL:    Laufwerke               º
Û--------------------------------ì
```

Im Register BL wird die Nummer des gewünschten Laufwerkes angegeben (0 = Default, 1 = A:, 2 = B:, etc.). Beim Aufruf wird dann der nächste logische Laufwerksname zugeordnet. Falls kein Fehler auftritt, das Carry-Flag ist nach dem Aufruf gelöscht, enthält das Register AL die Nummer des nun zugeordnetenen logischen Laufwerkes (1 = A:, 2 = B:, etc.).

Mit der Funktion 440EH läßt sich prüfen, ob mehrere logische Einheiten zugeordnet sind. Jeder Aufruf schaltet demnach die Laufwerks-Kennung um einen Buchstaben weiter, wobei die Reihenfolge zirkular ist. Falls nur eine Einheit zugeordnet ist, enthält AL den Wert 0.

Dieser Aufruf wird z.B. bei Systemen mit einem Diskettenlaufwerk benutzt. Soll z.B. eine Diskette kopiert werden, gibt DOS die Meldung:

```
Insert diskette for drive A: and strike any key when ready
```

aus. Anschließend kommt die gleiche Meldung mit der Laufwerksbezeichnung B:, usw. Um diesen Wechsel zu unterstützten, läßt sich obige Funktion nutzen. Zuerst wird der Laufwerksname A: eingestellt. Nach der Leseoperation stellt ein Aufruf der Funktion 440FH das Diskettenlaufwerk auf die logische Einheit B: um und veranlaßt die DOS-Abfrage. Ein weiterer Aufruf schaltet die Einheit wieder auf A: zurück.

Tritt während des Aufrufes ein Fehler auf, ist das Carry-Flag gesetzt. Im Register AX wird dann ein Fehlercode mit folgender Belegung zurückgegeben.

```
Code  Fehler
  1   Funktion nicht definiert
 15   Laufwerk existiert nicht
```

Die Fehlerursache ist bei den Extended-DOS-Errors beschrieben und läßt sich auch durch die Funktion 59H abfragen.

4.65.13 Query Support (Handle) (Funktion 4410H, DOS 6.x)

Dieser Aufruf ist erst ab DOS 5.0 implementiert und ermittelt, ob ein Funktionsaufruf per 440CH vom entsprechenden Treiber unterstützt wird. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    44H                    º
º AL:    10H                    º
º BX:    Handle                 º
º CH:    Hauptcode (Gerät)      º
º CL:    Subcode (Operation)    º
º DS:DX Adresse Parameterblock  º
û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: clear -> kein Fehler   º
Û-------------------------------ì
```

Im Register BX ist der Handle des Gerätes zu übergeben. In CH findet sich der Hauptcode des Gerätes, welches überprüft werden soll. Es sind die Werte der Funktion 440CH als Hauptcode erlaubt. In CL ist der Code für die Unterfunktion einzutragen. Auch hier gilt die Codierung der Funktion 440CH.

In DS:DX ist die Adresse eines Parameterblocks zu übergeben. Für den Parameterblock gilt die gleiche Kodierung wie bei der betreffenden Subfunktion des Aufrufes 440CH.

Beim Aufruf prüft DOS, ob die Funktion 440CH die entsprechende Subfunktion unterstützt. Falls dies der Falls ist, ist das Carry-Flag nach dem Aufruf gelöscht und AX enthält den Wert 0. Im Fehlerfall setzt DOS das Carry-Flag und in AX befindet sich der Fehlercode:

```
Code  Fehler
 1    Funktion nicht unterstützt.
```

Die Ursache kann allerdings mehrdeutig sein (Funktion nicht unterstützt, DOS-Version falsch, Treiber unterstützt die Funktion 19H nicht).

4.65.14 Query Support (Block) (Funktion 4411H, DOS 6.x)

Auch dieser Aufruf ist erst ab DOS 5.0 implementiert und ermittelt, ob ein Funktionsaufruf per 440DH vom entsprechenden Treiber unterstützt wird. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 21            º
º                               º
º AH:    44H                    º
º AL:    11H                    º
º BL:    Laufwerk               º
º CH:    Hauptcode (Gerät = 08) º
º CL:    Subcode (Operation)    º
º DS:DX Adresse Parameterblock  º
û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: clear -> kein Fehler   º
Û-------------------------------ì
```

Im Register BL ist der Code für das betreffende Laufwerk (0 = default, 1 = A:, 2 = B:, etc.) zu übergeben. CH enthält beim Aufruf immer den Wert 08H. In CL ist der Code für die entsprechende Subfunktion der INT 21-Funktion 440DH zu übergeben. In DS:DX ist die Adresse eines Parameterblocks zu übergeben.

Für den Parameterblock gilt die gleiche Kodierung wie bei der betreffenden Subfunktion des Aufrufes 440DH.

Beim Aufruf prüft DOS, ob die Funktion 440DH die entsprechende Subfunktion unterstützt. Falls dies der Fall ist, ist das Carry-Flag nach dem Aufruf gelöscht und AX enthält den Wert 0. Im Fehlerfall setzt DOS das Carry-Flag und in AX befindet sich der Fehlercode:

```
Code  Fehler
 1    Funktion nicht unterstützt.
```

Die Ursache kann allerdings mehrdeutig sein (Funktion nicht unterstützt, DOS-Version falsch, Treiber unterstützt nicht die Funktion 19H).

Einige Zusatzprogramme unterstützen weitere Subfunktionen des INT 21, Funktion 44.

4.66 Duplicate File Handle (Funktion 45H, DOS 2.0-6.x)

Diese Funktion erzeugt ein zusätzliches Handle für eine Datei. Im Register BX muß das Handle einer geöffneten Datei angegeben werden. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
º      CALL:   INT 21           º
º                               º
º AH:    45H                    º
º BX:    Handle                 º
û-------------------------------À
º      RETURN                   º
º Carry: gesetzt -> Fehler      º
º AX:    Fehlercode             º
º Carry: nicht gesetzt          º
º AX:    neues Handle           º
Û-------------------------------ì
```

Falls kein Fehler auftritt, das Carry-Flag ist nicht gesetzt, gibt die Funktion den neuen Handle im AX-Register zurück. Mit diesem Code kann sich anschließend ein Prozeß auf die gleiche Datei beziehen. Selbst die internen Zeiger innerhalb der Datei werden übernommen. Eine Veränderung der Schreib-Lese-Zeiger wirkt sich auf beide Handles aus. Normalerweise wird diese Funktion benutzt, um die Standard I/O-Devices umzuleiten.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in AX findet sich der Fehlercode:

```
  Code ° Bemerkung
-------é-------------------------------------------------
   4   ° Zu viele offene Dateien, kein Handle mehr frei
   6   ° Der Handle ist nicht eröffnet oder ungültig
```

Die *Extended DOS Errors Codes* lassen sich mit der Funktion 59H abfragen. Mit dem Aufruf 45H des INT 21 läßt sich die erst ab DOS 3.3 implementierte Funktion *Commit File* in früheren DOS-Versionen nachbilden.

4.67 Force Duplicate File Handle (Funktion 46H, DOS 2.0-6.x)

Diese Funktion übernimmt das spezifizierte Handle, um auf die gleiche Datei zu zeigen, die bereits durch ein anderes Handle belegt ist.

```
Ö-------------------------------Ì
°      CALL: INT 21             °
°                               °
° AH:    46H                    °
° BX:    Handle 1               °
° CX:    Handle 2               °
û-------------------------------À
°      RETURN                   °
° Carry: gesetzt -> Fehler      °
° AX:    Fehlercode             °
° Carry: nicht gesetzt          °
° ---    kein Fehler            °
Û-------------------------------ì
```

Falls kein Fehler auftritt, das Carry-Flag ist nicht gesetzt, hat die Funktion den neuen Handle der im CX-Register steht, der geöffneten Datei zugewiesen. Falls die durch CX adressierte Datei geöffnet war, wird sie beim Aufruf geschlossen.

Nach diesem Aufruf werden die Schreib-Lese-Zeiger der beiden Handles auf gleichen Werten gehalten. Normalerweise wird diese Funktion benutzt, um die Standard-I/O-Ausgaben umzuleiten.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in AX findet sich der Fehlercode:

```
  Code ° Bemerkung
-------é-------------------------------------------------
   4   ° Zu viele offene Dateien, kein Handle mehr frei
   6   ° Der Handle ist nicht eröffnet oder ungültig
```

Mit der Funktion 46H lassen sich die erweiterten DOS-Fehlercodes abfragen. In DOS 3.3 kommt es zu einem Systemabsturz, falls CX = BX gesetzt wird.

4.68 Get Current Directory (Funktion 47H, DOS 2.0-6.x)

Diese Funktion gibt den Pfadnamen des aktuellen Directory zurück. Es gelten folgende Aufrufkonventionen:

```
Ö------------------------------Ì
º      CALL: INT 21            º
º                              º
º AH:    47H                   º
º DL:    Drive Nummer          º
º DS:SI  Zeiger auf einen Puffer º
û------------------------------À
º      RETURN                  º
º Carry: gesetzt -> Fehler     º
º AX:    Fehlercode            º
º Carry: nicht gesetzt         º
º ---    kein Fehler           º
Û------------------------------ì
```

In DL muß eine gültige Laufwerksnummer (0 = Default, 1 = A:, 2 = B:, etc.) stehen. Das Registerpaar DS:SI muß auf einen mindestens 64 Byte großen Puffer zeigen, in dem der Pfad abgelegt werden kann. Der Pfadname wird als ASCIIZ-String, der den Weg vom Hauptverzeichnis bis zum aktuellen Unterverzeichnis angibt, zurückgegeben. Der String enthält allerdings keine Laufwerksbezeichnung und beginnt nicht mit dem Zeichen \ .

Er wird als ASCIIZ-String mit einem Nullbyte (00H) abgeschlossen.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in AX findet sich der Fehlercode:

```
  Code º Bemerkung
-------é-------------------------------------------------
  15   º Das in DL angegebene Laufwerk in nicht gültig
```

Mittels der Funktion 59H lassen sich die erweiterten DOS-Fehlercodes abfragen. Die Funktion gibt bei einem erfolgreichen Aufruf AX = 0100H zurück, was aber nicht dokumentiert wurde. Viele von Microsoft entwickelte Windows-Programme nutzen aber diese undokumentierte Eigenschaft.

4.69 Allocate Memory (Funktion 48H, DOS 2.0-6.x)

Diese Funktion versucht, den angeforderten Speicherbereich zu reservieren. Es gelten folgende Aufrufkonventionen:

```
Ö------------------------------Ì
º      CALL:  INT 21           º
º                              º
º AH:    48H                   º
º BX:    Zahl der Paragraphen  º
û------------------------------À
º      RETURN                  º
º Carry: gesetzt -> Fehler     º
º AX:    Fehlercode            º
º Carry: nicht gesetzt         º
º AX:    Segment Adresse       º
º BX:    freie Paragraphen     º
Û------------------------------ì
```

Mit dem Wert im Register BX wird vor dem Aufruf die Zahl der zu reservierenden Paragraphen (16-Byte-Blöcke) spezifiziert. Falls kein Fehler auftritt, das Carry-Flag ist nicht gesetzt, reserviert die Funktion den angeforderten Speicherbereich. Im Register AX wird die Segmentadresse des Speicherbereiches zurückgegeben. Die erste freie Adresse des Speicherblocks findet sich somit bei AX:0000H.

Das Register BX enthält nach dem Aufruf die Zahl der noch freien Paragraphen (16-Byte-Blöcke) des größten freien Speicherbereiches.

Um festzustellen, wieviel Speicher noch frei ist, kann folgender Trick angewandt werden:

Die Funktion wird aufgerufen, wobei die Zahl der angeforderten Paragraphen auf FFFFH gesetzt wird. Dies bedeutet eine Anforderung von 1 Mbyte Speicher, was durch DOS nie erfüllt werden kann, d.h. es wird kein Speicher reserviert. Anschließend gibt die Funktion im Register BX die Zahl der noch freien Paragraphen zurück. Der angegebene Wert spezifiziert allerdings nur die Summe der Paragraphen im größten freien Speicherblock. Dies sagt aber nichts über die Lage und Größe der freien Bereiche aus. Die Lage ist abhängig von der in DOS eingestellten Freispeicherverwaltungsstrategie. Im Kapitel über den DOS-Memory-Manager finden sich weitere Informationen.

Tritt während des Aufrufs ein Fehler auf, wird das Carry-Flag gesetzt und in AX findet sich der Fehlercode:

```
  Code ° Bemerkung
-------é-----------------------------------------------
   7   ° Die Memory Control Blocks (MCB) sind zerstört
   8   ° Nicht genügend freier Speicherbereich um die
       ° angeforderte Menge zu reservieren.
```

Der Fehlercode 7 kann normalerweise nur vorkommen, wenn Anwenderprogramme sich nicht an die Konventionen halten und selbst Speicherblöcke belegen. Mittels der Funktion 59H lassen sich auch die erweiterten DOS-Fehlercodes abfragen. Die Zuordnung der MCB's zu einzelnen Programmen erfolgt über einen Zeiger im MCB-Kopf, der auf den zugehörigen PSP des Programmes verweist. Ab DOS 3.3 werden bei der Suche nach freiem Speicher (zur Zuweisung an COM-Programme) nebeneinander liegende freie Blöcke zusammengefaßt.

4.70 Free Allocated Memory (Funktion 49H, DOS 2.0-6.x)

Diese Funktion gibt den angegebenen Speicherbereich wiede frei. Es gelten folgende Aufrufkonventionen:

```
ö------------------------------ì
°      CALL:  INT 21           °
°                              °
° AH:    49H                   °
° ES:    Segmentadr. Speicher  °
û------------------------------À
°      RETURN                  °
° Carry: gesetzt -> Fehler     °
° AX:    Fehlercode            °
° Carry: nicht gesetzt         °
° ---    kein Fehler           °
Û------------------------------ì
```

Falls kein Fehler auftritt, das Carry-Flag ist nicht gesetzt, wird der reservierte Block, dessen Adresse durch das ES-Register spezifiziert wurde, an den Speicherpool freigegeben. Es genügt die Segmentadresse zu übergeben, da ja der Offset implizit 0000H ist. Die Länge des Speicherbereiches ist intern im MCB vermerkt.

Nebeneinander liegende freie Blöcke werden allerdings nicht zusammengefaßt. Dies ist in DOS 3.3 nur bei den Funktionen 48H und 4AH der Fall.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in AX findet sich der Fehlercode:

```
  Code ° Bemerkung
-------é-------------------------------------------------------------
   7   ° Die Memory Control Blocks (MCB) sind zerstört
   9   ° An der angegebenen Adresse findet sich kein Speicherblock,
       ° der durch die Funktion 48H reserviert wurde
```

Mit der Funktion *59H* lassen sich die erweiterten DOS-Fehlercodes abfragen.

4.71 Set Block (Funktion 4AH, DOS 2.0-4.x)

Diese Funktion verändert die Größe des reservierten Speicherblocks. Es gelten folgende Aufrufkonventionen:

```
ö-------------------------------Ì
°      CALL:  INT 21            °
°                               °
° AH:    4AH                    °
° BX:    Zahl der Paragraphen   °
° ES:    Segmentadresse Block   °
û-------------------------------À
°      RETURN                   °
° Carry: gesetzt -> Fehler      °
° AX:    Fehlercode             °
° BX:    freie Paragraphen      °
° Carry: nicht gesetzt          °
°  --    kein Fehler            °
Û-------------------------------ì
```

Im Register *ES* wird die Segmentadresse des zu verändernden Speicherblocks angegeben. *BX* enthält die gewünschte Speichergröße in Paragraphen (16-Byte-Blöcke). Falls kein Fehler auftritt, das Carry-Flag ist nicht gesetzt, reserviert die Funktion den angeforderten Speicherbereich. Falls dies nicht möglich ist (z.B. bei Vergrößerungen), gibt das Register *BX* die Zahl der Paragraphen des größten freien Blocks zurück.

Bei Vergößerungen muß hinter dem Block ein freier MCB liegen. Ab DOS 3.3 faßt das Betriebssystem nebeneinander liegende freie Blöcke bei der Suche nach freiem Speicher zusammen.

Im Fehlerfall wird das Carry-Flag gesetzt und im *AX*-Register findet sich ein Fehlercode.

```
  Code ° Bemerkung
-------é-------------------------------------------------------------
   7   ° Die Memory-Control-Blocks (MCB) sind zerstört
   8   ° Nicht genügend freier Speicherbereich, um die angeforderte Menge
       ° zu reservieren
   9   ° Falsche Adresse in ES, gibt keinen Speicherblock an
```

Mit der Funktion *59H* lassen sich die erweiterten Fehlercodes abfragen.

4.72 Load or Execute a Program (Funktion 4BH, DOS 2.0-6.x)

Mit der (*EXEC*) Funktion lassen sich unter DOS »Child Processe.

4.72.1 Load and Execute (AL = 0)

Dieser Aufruf ist in allen DOS-Versionen vorhanden und lädt ein ausführbares Programm (COM oder EXE) in den freien Speicher und startet dieses Programm als Subprozeß (Child Process). Der Vaterprozeß (Parent Process) ist während der Laufzeit suspendiert. Es gilt folgende Parameterübergabe:

```
Ö---------------------------------Ì
º      CALL:  INT 21               º
º                                  º
º AH:    4BH                       º
º AL:    00H (Load and Execute)    º
º DS:DX  Zeiger auf einen ASCIIZ -º
º        String mit dem Pfadnamen  º
º ES:BX  Zeiger auf einen          º
º        Parameterblock            º
û---------------------------------À
º      RETURN                      º
º Carry: gesetzt -> Fehler         º
º AX:    Fehlercode                º
º Carry: nicht gesetzt             º
Û---------------------------------ì
```

Das Register *AL* wird mit dem Steuercode 0 besetzt. Das Registerpaar *DS:DX* zeigt auf einen ASCIIZ-String, der Laufwerksbezeichnung, Pfad- und Dateiname des auszuführenden Programmes beinhaltet. Beim Laden des neuen Programmes wird automatisch ein eigenes Programm Segment Prefix (*PSP*) angelegt. Um dieses Segment zu initialisieren, benötigt der Lader allerdings verschiedene Informationen.

Diese müssen in einem Parameterblock abgelegt werden, der durch das Registerpaar *ES:BX* adressiert wird. Er besitzt folgenden Aufbau:

```
Ö--------Ú-------Ú-----------------------------------------------Ì
º Offset º Bytes º Bedeutung                                      º
û--------é-------é-----------------------------------------------À
º   00H  º   2   º Segmentadresse des Environment String          º
º        º       º 00 -> kopiere das »Parent
Environment««û--------é-------é-------------------------------------------------À
º   02H  º   4   º Zeiger (Segment:Offset) auf einen String mit  º
º        º       º einer Kommandozeile, die ab Offset 80H im     º
º        º       º neuen PSP abgelegt wird. Die Zeile darf nicht º
º        º       º länger als 128 Byte sein                      º
û--------é-------é-----------------------------------------------À
º   06H  º   4   º Zeiger auf einen File Control Block, der ab   º
º        º       º Offset 5CH (FCB1) im neuen PSP abgelegt wird  º
û--------é-------é-----------------------------------------------À
º   0AH  º   4   º Zeiger auf einen File Control Block, der ab   º
º        º       º Offset 6CH (FCB2) im neuen PSP abgelegt wird  º
Û--------Ù-------Ù-----------------------------------------------ì
```

Tabelle 4.28: Die Struktur der EXEC-Parametertabelle

DOS legt für den Subprozeß alle Informationen des Hauptprozesses offen, bzw. diese Informationen werden kopiert. So wird für den »Child Processe beginnen. Falls das zu ladende Programm im aktuellen Unterverzeichnis abgelegt ist, findet sich im Environment der Pfad zu diesem Programm. Im ersten Wort des durch *ES:BX* adressierten Parameterblocks ist die Lage des Environmentbereiches abgelegt. Falls der Wert = 0 ist, dann wird das Environment des »Parent Processes**Fehler! Verweisquelle konnte nicht gefunden werden.**Weiterhin sind auch alle geöffneten Dateien dem Subprozeß bekannt, es sei denn, daß »Inherit Bit*H* im »Parent Process**Fehler! Verweisquelle konnte nicht gefunden werden.**Child Processes**Fehler! Verweisquelle konnte nicht gefunden**

werden. Parent Processes**Fehler! Verweisquelle konnte nicht gefunden werden.**Um einen Subprozeß laden zu können, muß genügend Speicher frei sein. Da normalerweise ein Anwenderprogramm beim Laden den gesamten freien Speicher erhält, muß ein Teil mit der Funktion *4AH* freigegeben werden.

Die Funktion *4BH* benutzt den Lader des Kommandointerpreters COMMAN.COM. Dieser Lader befindet sich im transienten Bereich von COMMAND und wird vom Anwenderprogramm überschrieben. Ist nicht genügend Speicher vorhanden, um diesen Teil nachzuladen, dann bricht die Funktion mit einer Fehlermeldung ab. In diesem Fall ist das Carry-Flag gesetzt und im *AX*-Register findet sich ein Fehlercode:

```
   Code ° Fehler
--------é-----------------------------------------------------
    1   ° Subfunktion nicht unterstützt
    2   ° Die Programmdatei wurde nicht gefunden, oder
        ° der Pfad ist ungültig
    3   ° Suchweg nicht gefunden
    4   ° keine freien Handles
    8   ° Nicht genügend Speicher, um das Programm zu laden
   11   ° Die Programmdatei ist vom Typ EXE und enthält
        ° inkonsistente Informationen
```

Mittels der Funktion *59H* lassen sich die erweiterten DOS-Fehlercodes abfragen.

Um das Programm zu laden und auszuführen, kann die reine *EXEC*-Funktion benutzt werden. Dann muß das rufende Programm allerdings dafür sorgen, daß der richtige Zugriffspfad eingestellt ist, daß keine *EXE*-Dateien zu bearbeiten sind, etc. Deshalb ist es in der Regel einfacher, den Kommandoprozessor COMMAND.COM für diese Aufgabe zu nutzen. DOS erlaubt es, eine zweite Kopie des Kommandoprozessors zu laden. Als Name des auszuführenden Programms wird COMMAND.COM angegeben. Notfalls ist noch der Pfad zur dieser Datei anzugeben. Im Parameterblock wird unter Offsetadresse *2* ein Zeiger auf einen Kommandostring abgespeichert. Dieser muß das Format:

```
Länge /C Kommando 0DH
```

besitzen, damit der Kommandointerpreter den Reststring als ausführbare Anweisung interpretiert. Wegen der Komplexität wird das Thema in einem eigenen Abschnitt mit Hilfe von Beispielprogrammen erläutert.

Achtung: Bei allen DOS-Versionen unter 3.0 werden die Register, einschließlich Stackpointer und Stacksegment, zerstört.

4.72.2 Load (AL = 1 undokumentiert)

Diese Funktion ist in allen DOS-Versionen ab 2.0 vorhanden.

```
Ö----------------------------------Ì
º      CALL:  INT 21               º
º                                  º
º AH:    4BH                       º
º AL:    01H (Load)                º
º DS:DX  Zeiger auf einen ASCIIZ-  º
º        String mit dem Pfadnamen  º
º ES:BX  Zeiger auf einen          º
º        Parameterblock            º
û----------------------------------À
º      RETURN                      º
º Carry: gesetzt -> Fehler         º
º AX:    Fehlercode                º
º Carry: nicht gesetzt             º
Û----------------------------------ì
```

Dieser undokumentierte Aufruf entspricht im wesentlichen der Unterfunktion *4B00H*, wobei allerdings das Programm nur geladen und nicht gestartet wird. Es gelten die Aufrufparameter der nebenstehenden Abbildung.

Der durch *ES:BX* adressierte Parameterblock besitzt folgenden Aufbau:

```
Ö--------Ú-------Ú--------------------------------------------------Ì
º Offset º Bytes º Bedeutung                                        º
û--------é-------é--------------------------------------------------À
º   00H  º   2   º Segmentadresse des Environment Strings            º
º        º       º 00 -> kopiere das »Parent
Environment«»û--------é-------é-------------------------------------------------À
º   02H  º   4   º Zeiger (Segment:Offset) auf einen String mit  º
º        º       º einer Kommandozeile, die ab Offset 80H im     º
º        º       º neuen PSP abgelegt wird. Die Zeile darf nicht º
º        º       º länger als 128 Byte sein                      º
û--------é-------é-----------------------------------------------À
º   06H  º   4   º Zeiger auf einen File-Control-Block, der ab   º
º        º       º Offset 5CH (FCB1) im neuen PSP abgelegt wird  º
û--------é-------é-----------------------------------------------À
º   0AH  º   4   º Zeiger auf einen File-Control-Block, der ab   º
º        º       º Offset 6CH (FCB2) im neuen PSP abgelegt wird  º
û--------é-------é-----------------------------------------------À
º   0EH  º   4   º enthält nach dem Aufruf die Adresse des Stacks º
º        º       º SS:SP                                         º
û--------é-------é-----------------------------------------------À
º   12H  º   4   º enthält nach dem Aufruf die Anfangsadresse     º
º        º       º CS:IP des neuen Prozesses                     º
Û--------Ù-------Ù-----------------------------------------------ì
```

Die beiden letzten Einträge im Parameterblock sind vor dem Aufruf unbelegt. Die Funktion speichert hier dann die Anfangsadresse des neuen Prozesses und die erste Stackadresse ab. Nach dem Aufruf wird wieder zu aufrufenden Prozeß zurückgekehrt. Eine Verwendung ist nur in wenigen Fällen möglich, zum Beispiel in Verbindung mit einer Betriebssystem-erweiterung, die mehrere Programme im Speicher hält und abwechselnd aktiviert. Weitere Einsatzbeispiele sind Debugger, die genau obige Funktionalität benötigen.

Bei einem fehlerhaften Aufruf ist das Carry-Flag gesetzt und im *AX*-Register findet sich ein Fehlercode:

```
   Code ° Fehler
--------é-----------------------------------------------------
    1   ° Subfunktion nicht unterstützt
    2   ° Die Programmdatei wurde nicht gefunden, oder
        ° der Pfad ist ungültig
    3   ° Suchweg nicht gefunden
    4   ° keine freien Handles
    8   ° Nicht genügend Speicher, um das Programm zu laden
   10   ° ungültiges Environment
   11   ° ungültiges Format im Parameterblock (ES:BX)
```

Der Fehler 4 tritt auf, falls zu viele Dateien geöffnet sind. Da DOS bei jedem zu startenden Prozess standardmäßig 5 Handles belegt, kann bei einem Überlauf der System File Table (SFT) kein neuer Prozess mehr gestartet werden. Der Fehler 10 kann nur auftreten, falls ein selbst definiertes Environment übergeben wird. Mittels der Funktion *59H* lassen sich die erweiterten Fehlercodes bestimmen.

4.72.3 Load Overlay (AL = 3)

Diese Funktion ist ebenfalls ab DOS 2.0 vorhanden.

```
Ö---------------------------------Ì
°      CALL:  INT 21               °
°                                  °
° AH:    4BH                       °
° AL:    03H (Load Overlay)        °
° DS:DX  Zeiger auf einen ASCIIZ-  °
°        String mit dem Pfadnamen  °
° ES:BX  Zeiger auf einen          °
°        Parameterblock            °
û---------------------------------À
°      RETURN                      °
° Carry: gesetzt -> Fehler         °
° AX:    Fehlercode                °
° Carry: nicht gesetzt             °
Û---------------------------------ì
```

Mit diesem Aufruf (*AL = 3*) lädt die Funktion *4BH* eine Datei als Overlay in den freien Speicher. Es sind nebenstehende Parameter zu setzen.

MS-DOS nimmt an, daß das Overlayprogramm in den Adreßbereich des laufenden Prozesses geladen werden soll. Deshalb wird auch kein zusätzlicher Speicher benötigt und es werden auch keine PSP und Environmentbereiche angelegt. Daher unterscheidet sich der Aufbau des Parameterblocks von der oben diskutierten Form.

```
Ö--------Ú-------Ú-----------------------------------------------Ì
° Offset ° Bytes ° Bedeutung                                     °
û--------é-------é-----------------------------------------------À
°   00H  °   2   ° Segmentadresse, ab der das Overlayprogramm zu °
°        °       ° laden ist                                     °
û--------é-------é-----------------------------------------------À
°   02H  °   2   ° Informationen für die Relokation einer EXE-   °
°        °       ° Datei (Relocations Faktor)                    °
Û--------Ù-------Ù-----------------------------------------------ì
```

Tabelle 4.29: Die Struktur der Load-Overlay-Tabelle

Tritt während des Aufrufs ein Fehler auf, wird das Carry-Flag gesetzt und im AX-Register findet sich ein Fehlercode.

```
   Code º Fehler
--------é-------------------------------------------------------
    1   º Der Wert in AL ist nicht 0, 1 oder 3
    2   º Die Programmdatei wurde nicht gefunden,
        º oder der Pfad ist ungültig
    3   º Suchweg nicht gefunden
    4   º keine freien Handles
    8   º Nicht genügend Speicher um das Programm zu laden
```

Mittels der Funktion *59H* lassen sich die erweiterten DOS-Fehlercodes abfragen.

Ein Aufruf der Subfunktion 4B04H besitzt keine Wirkung, obwohl Microsoft C diesen Aufruf ab DOS 4.0 benutzt. Vielleicht existiert eine DOS-Version als OEM-Variante, die hier etwas mit anfangen kann.

4.72.4 EXEC: Enter EXEC-State (AL = 5)

Diese Funktion ist erst ab DOS 5.0 vorhanden.

```
Ö---------------------------------Ì
º      CALL:  INT 21               º
º                                  º
º AH:    4BH                       º
º AL:    05H (Enter Exec State)    º
º DS:DX  Zeiger auf einen Parame-  º
º        terblock                  º
û---------------------------------À
º      RETURN                      º
º Carry: gesetzt -> Fehler         º
º AX:    Fehlercode                º
º Carry: nicht gesetzt -> ok       º
Û---------------------------------ì
```

Mit diesem Aufruf (*AL = 5*) erlaubt die Funktion *4BH* einem Prozess ein Programm selbst zu laden und als Subprozess unter Umgehung von DOS zu starten. Die Subfunktion verändert einige intere Einstellungen in den DOS-Datentabellen. Damit kann ein Programm die COM- und EXE-Dateien selbst laden und starten.

In DS:DX ist die Adresse eines Parameterblocks mit folgendem Aufbau zu übergeben.

```
Ö--------Ú-------Ú-----------------------------------------------Ì
º Offset º Bytes º Bedeutung                                     º
û--------é-------é-----------------------------------------------À
º   00H  º   2   º reserviert (immer 0)                           º
û--------é-------é-----------------------------------------------À
º   02H  º   2   º Flags (Bit 0 = 1 : EXE-File)                   º
º        º       º (Bit 1 = 1 : Overlay File)                    º
û--------é-------é-----------------------------------------------À
º   04H  º   4   º Adresse ASCIIZ-String mit dem                  º
º        º       º Programmnamen                                 º
û--------é-------é-----------------------------------------------À
º   08H  º   2   º PSP-Segment des neuen Programmes               º
û--------é-------é-----------------------------------------------À
º   0AH  º   2   º IP Wert beim Start von DOS                     º
û--------é-------é-----------------------------------------------À
º   0CH  º   2   º CS Wert beim Start von DOS                     º
û--------é-------é-----------------------------------------------À
º   0EH  º   4   º Programmgröße in Byte                          º
Û--------Ù-------Ù-----------------------------------------------ì
```

Tabelle 4.29.1: Die Struktur der Enter EXEC-State-Tabelle

Ab Offset 04H ist die Adresse auf einen ASCIIZ-String einzutragen. Dieser String besitzt maximal 64-Byte-Länge und enthält den Programmnamen, sowie optional das Laufwerk und den Pfad. Ab Offset 0AH werden zwei Worte angelegt, in denen der von DOS zu setzende Startwert für CS:IP hinterlegt wird. Der letzte Eintrag enthält die Programmlänge, einschließlich PSP in Byte.

Nach dem Aufruf führt die Funktion folgende Aktionen durch:

· Prüft den Filenamen und emuliert gegebenfalls die per SETVER definierte DOS-Versionsnummer.

· Modifiziert interne DOS Datentabellen, falls das Programm in den untersten 64 Kbyte des Speichers geladen wird.

· Weiterhin können programmspezifische Patches erfolgen.

Das rufende Programm kann anschließend die Kontrolle an den Subprozess übergeben. Während des Zeitraumes nach der Rückkehr vom Aufruf 4B05H und dem Transfer zum Subprozess darf kein BIOS- oder DOS-Aufruf erfolgen. Interrupts dürfen auch keine ausgelöst werden.

Ist der DOS-Kern im HMA geladen, wird beim Aufruf der Funktion 4B05H die A20-Adressleitung deaktiviert (A20 = 0). Damit ist der DOS-Kern für den Parent Prozess nicht mehr verfügbar. Der nächste Aufruf einer INT 21-Funktion schaltet die A20-Adressleitung wieder ein. Dies sollte nur durch den Subprozess erfolgen, da im Hauptprogramm ein INT 21-Aufruf die Funktionen des Aufrufes 4B05H aufheben würde.

Ein Programm, welches die Subfunktion benutzt, muß folgende Schritte ausführen:

· Reservieren eines Speicherbereiches für das Programm, Definition eines PSP und Erzeugung der Environmenttabelle.

· Laden der Datei in den reservierten Bereich. Bei EXE-Files müssen die Adressbezüge zusätzlich reloziert werden.

· Einsetzen der Rücksprungadresse in das PSP. Hier kann die Adresse des Parent Prozesses stehen.

· Aufruf der Funktion 4B05H um das Betriebssystem abzuschalten. Es dürfen dann keine DOS- oder BIOS-Aufrufe mehr folgen, da sonst Nebeneffekte auftreten. Die Funktion 4B05H setzt zum Beispiel das neue PSP als aktives PSP.

· Setzen der Segmentregister und Verzweigung zum Startpunkt im Subprozess. Der erste INT 21-Aufruf im Subprozess gibt dann die A20-Adressleitung wieder frei. Damit steht der DOS-Kern wieder zu Verfügung und der neue PSP ist aktiv.

· Nach Beendigung des Subprozesses wird zur Rückkehradresse verzweigt (Definition im PSP).

· Freigabe des Speichers durch den Parent Prozess.

Tritt während des Aufrufs ein Fehler auf, wird das Carry-Flag gesetzt und im AX-Register findet sich ein Fehlercode.

```
   Code ° Fehler
--------é-----------------------------------------------------
    1   ° Der Wert in AL ist nicht 0, 1 oder 3
    2   ° Die Programmdatei wurde nicht gefunden, oder der
        ° Pfad ist ungültig
    3   ° Suchweg nicht gefunden
    4   ° keine freien Handles
    8   ° Nicht genügend Speicher um das Programm zu laden
```

Mittels der Funktion *59H* lassen sich die erweiterten DOS-Fehlercodes abfragen. Zur Zeit fehlen mir allerdings noch Informationen über die praktische Anwendung der Funktion.

Weiterhin wird die Funktion 4BH von einigen Virenprogrammen abgefangen. Die genauen Aufrufspezifikationen möchte ich an dieser Stelle (aus verschiedenen Gründen) nicht erläutern.

4.73 Terminate a Process (Funktion 4CH, DOS 2.0-6.x)

Mit dieser Funktion läßt sich ein laufender Prozeß beenden. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:    4CH                      °
° AL:    Returncode               °
û---------------------------------À
°      RETURN                     °
° ---                             °
Û---------------------------------ì
```

Im Register *AL* kann ein Returncode an den »Parent Process**Fehler! Verweisquelle konnte nicht gefunden werden.**Die Rückkehradresse ist im PSP (INT 22) definiert.

Gleichzeitig mit dem Programmabbruch werden alle durch den Subprozeß neu geöffneten Dateien geschlossen. Die vom Parent Prozess geerbten offenen Dateien bleiben allerdings weiterhin offen. Weiterhin gibt DOS den belegten Speicher des Subprozesses frei.

4.74 Get Returncode (Funktion 4DH, DOS 2.0-6.x)

Mit dieser Funktion läßt sich der Returncode eines Subprozesses lesen. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:    4DH                      °
û---------------------------------À
°      RETURN                     °
° AH:    Terminatorcode           °
° AL:    Returncode               °
Û---------------------------------ì
```

Der Fehlercode kann nur einmal gelesen werden. Er wird im *AX*-Register zurückgegeben, wobei das *AL* Register den von der Funktion *4CH* eingetragenen Wert enthält. Wird ein Programm über den INT 27 resident installiert, ist der EXIT-Code allerdings immer 0.

Der Fehlercode läßt sich über die DOS-Funktion ERRORLEVEL beliebig häufig abfragen.

Im *AH*-Register finden sich Informationen über die Ursache des Programmabbruchs:

```
 Code ° Bemerkung
------é-----------------------------------------
   0  ° normales Programmende
   1  ° Abbruch durch Control-C
   2  ° Abbruch als Ergebnis eines Critical Errors
   3  ° Abbruch durch Funktion 31H
```

Mit der Funktion *59H* lassen sich die erweiterten DOS-Fehlercodes abfragen.

4.75 Find First Matching File (Funktion 4EH, DOS 2.0-6.x)

Mit dieser Funktion läßt sich prüfen, ob eine Datei mit dem angegebenen Namen im Inhaltsverzeichnis abgelegt wurde. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:    4EH                      °
° DS:DX  Zeiger auf ASCIIZ-       °
°        String mit dem Pfadnamen °
° CX:    Attribute der Datei      °
û---------------------------------À
°      RETURN                     °
° Carry: gesetzt -> Fehler        °
° AX:    Fehlercode               °
Û---------------------------------ì
```

Der Zeiger im Registerpaar *DS:DX* adressiert einen ASCIIZ-String mit dem Pfadnamen, der auch die Dateibezeichnung enthält. Der Filename kann auch Wildcards (*,?) enthalten, hierbei darf es sich aber nicht um eine Datei innerhalb eines Netzwerkes handeln. *CX* gibt die Dateiattribute an, die bei der Suche zu berücksichtigen sind (normal, hidden, etc.). Wird eines der Attribute »Hidden« »System**Fehler! Verweisquelle konnte nicht gefunden werden.**Directory**Fehler! Verweisquelle konnte nicht gefunden werden.**Eine exclusive Suche (z.B. nach hidden Files) ist also nicht möglich.

Falls eine Datei mit dem spezifizierten Namen gefunden wurde, kopiert die Funktion die Daten in die Disk-Transfer-Area (*DTA*). Dieser Bereich befindet sich standardmäßig im *PSP* ab Adresse *80H*. Bei der Funktion *4EH* besitzt der Bereich folgende Struktur:

Offset	Bytes	Bedeutung
00H	21	reserviert für die Funktion 4FH (Find Next)
15H	1	Attributbyte der gefundenenen Datei
16H	2	Eintragzeit des letzten Schreibzugriffs
18H	2	Eintragdatum des letzten Schreibzugriffs
1AH	2	Low Word Dateigröße
1CH	2	High Word Dateigröße
1EH	13	Name und Extension der Datei als ASCIIZ-String

Tabelle 4.30: Kodierung des Dateinamen bei Find-Aufrufen

Die Werte im ersten Feld (21 Byte) variieren in Abhängigkeit von der DOS-Version und werden durch die Funktion 4FH ausgewertet.

Der zurückgegebene Dateiname wird mit einem Nullbyte abgeschlossen. Zwischen Name und Extension findet sich ein Punkt. Aus dem String sind eventuelle Blanks bereits entfernt.

Bei Attributen = 08H werden unter DOS 2.x auch andere Filenamen mit zurückgegeben, während DOS 3.x nur Volume Label Namen zurückgibt. Wird ein Gerät (keine Wildcards erlaubt) spezifiziert, gibt DOS 2.x das Attribut = 00, die Dateigröße = 0000 und das aktuelle Datum / Uhrzeit zurück.

Falls ein Fehler auftritt, wird das Carry-Flag gesetzt. Im Register *AX* findet sich ein Fehlercode:

Code	Fehler
2	Datei nicht vorhanden
3	Der angegebene Pfad existiert nicht oder ist ungültig
18	Kein Dateieintrag gefunden

Mit der Funktion *59H* lassen sich die erweiterten DOS-Fehlercodes abfragen. In DOS 3.x und 4.x versagt die Funktion wegen eines Bugs. Wird sie mehrfach hintereinander mit Gerätenamen aktiviert, ohne daß zwischenzeitlich DOS-Calls oder eine Directory Suche (ohne Volume Label Attribut) erfolgten, gibt die Funktion fehlerhafte Ergebnisse zurück (falls Attribut = 08H). Hier sollte ein Aufruf der INT 21-Funktionen 3CH, 3DH, 41H oder 46H eingeschoben werden.

4.76 Find Next File (Funktion 4FH, DOS 2.0-6.x)

Mit diesem Aufruf läßt sich ein Inhaltsverzeichnis nach dem nächst gültigen Dateinamen durchsuchen. Es gelten folgende Aufrufparameter:

CALL:	INT 21
AH:	4FH

```
°      RETURN                    °
° Carry: gesetzt -> Fehler       °
° AX:    Fehlercode              °
Û---------------------------------ì
```

Die *DTA* muß die Information enthalten, die durch die Funktion *4EH* erzeugt wurden. Sonst sind keine weiteren Angaben erforderlich. Die Ergebnisse werden in den ersten 13 Byte abgelegt. Wird keine weitere Datei gefunden, erscheint der Fehler *18* (dezimal) im *AX*-Register.

Falls ein Fehler auftritt, wird das Carry-Flag gesetzt. Im Register AX findet sich ein lercode:

```
Code ° Fehler
-----é-----------------------------------------------------
 2   ° Der angegebene Pfad existiert nicht oder ist ungültig
18   ° Kein Dateieintrag gefunden
```

Mit der Funktion *59H* lassen sich die erweiterten DOS-Fehlercodes abfragen.

4.77 Set Active PSP (Funktion 50, DOS 2.0-6.x)

Hier handelt es sich um eine interne DOS-Funktion die durch Microsoft erst ab DOS 5.0 offiziell publiziert wurde. Es gelten folgende Aufrufkonventionen:

```
Ö---------------------------------Ì
°      CALL:  INT 21             °
°                                °
° AH:    50H                     °
° BX:    Segmentadresse des neuen °
°        PSP                     °
û---------------------------------À
°      RETURN                    °
° Carry: gesetzt -> Fehler       °
° AX:    Fehlercode              °
° Carry: gelöscht -> ok          °
Û---------------------------------ì
```

Die Funktion übernimmt im Register *BX* die Segmentadresse des neuen Programm-Segment-Prefix. Der Offsetwert wird implizit zu *0000H* angenommen. DOS übernimmt das *PSP* als aktives *PSP*. Vor einem Aufruf der Funktion ist das alte *PSP* zu retten. Mit dieser Funktion läßt sich ein neues *PSP* generieren oder von anderen Prozessen übernehmen.

Diese Funktion speichert die in BX angegebenen Segmentadresse in einer internen Datenstruktur. Damit greift DOS anschließend auf das gesetzte neue PSP-Segment zu. Dies bei TSR-Programmen hilfreich (s. entsprechendes Kapitel), da hier normalerweise das PSP des Vordergrundprogrammes aktiv ist. Für Fileoperationen benötigt das TSR-Programm aber die eigene PSP.

In DOS 2.x darf die Funktion nicht aus dem INT 28H heraus aufgerufen werden, falls nicht vorher das Critical-Error-Flag gesetzt wurde (Funktion AH = 5D06H). Dann benutzt

die Funktion den »critcal error stack4.78 Get Active PSP (Funktion 51, DOS 2.0-6.x)

Auch hier handelt es sich um eine interne DOS-Funktion, die durch Microsoft erst ab DOS 5.0 offiziell publiziert wurde. Die Funktion existiert aber bereits seit DOS 2.0 und in OS/2. Es gelten folgende Aufrufkonventionen:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    51H                      º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º BX:    Segmentadresse des       º
º        aktuellen PSP            º
Û---------------------------------ì
```

Die Funktion ermittelt die Lage des aktuellen *PSP* und gibt die Segmentadresse im *BX*-Register zurück.

Die Funktion erlaubt es also, aus einem laufenden Prozeß heraus festzustellen, wo sich der Kopf des Programmes befindet. Bei COM-Dateien ist dies trivial, da diese ja nicht über Segmentgrenzen hinausreichen, aber bei EXE-Programmen leistet die Funktion gute Dienste.

Falls ein Fehler auftritt, wird das Carry-Flag gesetzt. Im Register *AX* findet sich ein Fehlercode. Dieser entspricht den mit der Funktion *59H* abfragbaren erweiterten Fehlercodes.

Im Hinblick auf die Aufwärtskompatibilität der Software empfiehlt es sich aber, ab DOS 3.0 bis 4.01 die offizielle Funktion *62H* (Get PSP Address) zu verwenden, da diese das gleiche leistet.

In DOS 2.x darf die Funktion nicht aus dem INT 28H heraus aufgerufen werden, falls nicht vorher das Critical-Error-Flag gesetzt wurde (Funktion AH = 5D06H). Dann benutzt die Funktion den »critcal error stack4.79 Get DOS List of Lists (Funktion 52, DOS 2.0-6.x, undokumentiert)

Auch hier handelt es sich um eine interne DOS-Funktion, die durch Microsoft nicht offiziell dokumentiert wurde. Es gelten die folgenden Aufrufkonventionen.

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    52H                      º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º ES:BX  Zeiger auf den DOS       º
º        Datenblock               º
Û---------------------------------ì
```

Die Funktion gibt die Lage des internen DOS-Datenblocks zurück. (Genauer: ab DOS 3.0 handelt es sich um 3 aufeinanderfolgende Speicherblöcke). Der Zeiger findet sich im Register *ES:BX*. Im Datenblock finden sich die Zeiger zur Speicherverwaltung des Memory-Managers.Über diese Zeiger führt DOS die Zugriffe auf die einzelnen Speichersegmente aus.

Falls ein Fehler auftritt, wird das Carry-Flag gesetzt. Im Register *AX* findet sich ein Fehlercode. Dieser entspricht den mit der Funktion *59H* abfragbaren erweiterten Fehlercodes.

Offset	Bytes	Feld
-0CH	2	SHARE-Wiederholungszähler (DOS 3.1 - DOS 3.3) läßt per sich INT 21-Funktion 440B setzen
-0AH	2	SHARE-Verzögerungszähler (DOS 3.1 - DOS 3.3) läßt per sich INT 21-Funktion 440B setzen
-08H	4	Zeiger auf aktuellen Diskpuffer erst ab DOS 3.0 definiert
-04H	2	(ab DOS 3.0) Index auf den Puffer mit den ungelesenen Eingaben der CON-Einheit der Wert 0000 bedeutet keine Eingaben
-02H	2	Zeiger auf ersten Memory-Control-Block (MCB)
00H	4	Zeiger auf den DOS-Disk-Parameter-Block (DPB)
04H	4	Adresse der DOS System File Table
08H	4	Zeiger auf den Kopf des CLOCK$-Treibers
0CH	4	Zeiger auf den Kopf des Consol-Treibers
---	-	DOS 2.x Datenstruktur
10H	1	Anzahl log. Laufwerke im System
11H	2	max. Byte/Block der Block Treiber
13H	4	Zeiger auf den ersten Sektorpuffer
17H	18	Begin des NUL-Device Headers
---	-	DOS 3.0 Datenstruktur
10H	1	Zahl der Blockeinheiten
11H	2	max. Blockgröße für alle Blocktreiber
13H	4	Zeiger auf den ersten Diskpuffer
17H	4	Zeiger auf ein Feld mit den aktuellen Directory Strukturen (CDS)
1BH	1	LASTDRIVE Wert (meist 5)
1CH	4	Zeiger auf den STRING= Arbeitsbereich
20H	2	Wert x von STRING = X aus CONFIG.SYS
22H	4	Zeiger auf die FCB Tabelle
26H	2	Wert y aus FCBS = x,y in CONFIG.SYS
28H	18	Kopf des NUL-Einheitentreibers
---	-	DOS 3.1 - 3.3 Datenstruktur

10H	2	max. Blockgröße (Byte / Sektor) für alle Blocktreiber
12H	4	Zeiger auf den ersten Diskpuffer in der Kette
16H	4	Zeiger auf ein Feld mit den aktuellen Directory Strukturen (CDS)
1AH	4	Zeiger auf die FCB Tabellen
1EH	2	Anzahl der geschützten FCB's (FCBS=x,y)
20H	1	Anzahl der installierten Bockeinheiten
21H	1	Anzahl der Laufwerksbuchstaben
22H	18	Kopf des NUL-Einheitentreibers
34H	1	Zahl der mit JOIN umgeleiteten Drives
---	-	DOS 4.x Datenstruktur
10H	2	max. Byte / Sektor der Blocktreiber
12H	4	Zeiger auf den Diskpuffer Info Record
16H	4	Zeiger auf ein Feld mit den aktuellen Directory Strukturen (CDA)
1AH	4	Zeiger auf die FCB Tabellen
1EH	2	Anzahl der geschützten FCB's (y, FCBS=X,Y)
20H	1	Anzahl der installierten Blockeinheiten
21H	1	LASTDRIVE Wert (meist 5)
22H	18	Kopf des NUL-Einheitentreibers
34H	1	Zahl der mit JOIN umgeleiteten Drives
35H	2	Zeiger auf das IBMDOS Codesegment mit der Liste der Programmnamen/ -versionen
37H	4	Zeiger auf eine FAR-Routine mit dem residenten IFS-Funktionen, werden von IFS-Funktionen aktiviert, die die Funktionen 20H, 24H-28H nicht unterstützen.
3BH	4	Zeiger auf die Kette der IFS-Treiber
3FH	2	x-Wert von BUFFERS x,y wird auf ein mehrfaches von 30 gerundet, falls der Buffer im EMS-liegt.
41H	2	y-Wert von BUFFERS x,y
43H	1	Boot Laufwerk (1=A:)
44H	1	CPU-Flag (01H 80386+)

45H	2	XMS-Größe in Kbyte
---	-	DOS 5.x Datenstruktur
10H	2	max. Byte / Sektor der Blocktreiber
12H	4	Zeiger auf den Diskpuffer Info Record
16H	4	Zeiger auf ein Feld mit den aktuellen Directory Strukturen (CDA)
1AH	4	Zeiger auf die FCB Tabelle
1EH	2	0000H
20H	1	Anzahl der installierten Blockeinheiten
21H	1	LASTDRIVE Wert (meist 5)
22H	18	Kopf des NUL-Einheitentreibers
34H	1	Zahl der mit JOIN umgeleiteten Drives
35H	2	0000H
37H	4	Zeiger auf die SETVER-Tabelle (oder 0:0)
3BH	2	unbekannt
3DH	2	unbekannt
3FH	2	x-Wert von BUFFERS x,y
41H	2	y-Wert von BUFFERS x,y
43H	1	Boot Laufwerk (1=A:)
44H	1	CPU-Flag (01H 80386+)
45H	2	XMS-Größe in Kbyte

Tabelle 4.31: Aufbau des DOS-Control-Blocks

Der Aufbau der Tabelle ist abhängig von der DOS-Version. Der Aufbau der internen DOS-Tabellen wird im Kapitel über den DOS-Memory-Manager detailliert diskutiert.

4.80 Translate BIOS Parameter Block (Funktion 53, DOS 2.0 - 6.x, undokumentiert)

Auch hier handelt es sich um eine interne DOS-Funktion die durch Microsoft nicht offiziell publiziert wurde. Es gelten folgende Aufrufkonventionen:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    53H                      º
º ES:BP  Zeiger auf den aufzubau- º
º        enden Parameterblock     º
º DS:SI  Zeiger auf den BIOS-     º
º        Parameter-Block          º
û---------------------------------À
º      RETURN                     º
º AX:    ----                     º
Û---------------------------------ì
```

Der Zeiger *ES:BP* ist auf die neu zu initialisierende Tabelle zu setzen, während in *DS:SI* die Adresse der Tabelle mit den Initialisierungswerten der BIOS-Parameter-Blocks steht. Die Funktion baut nun die neue Tabelle mit diesen Werten aus dem BIOS-Parameter-Block (*BPB*) auf. Diese Technik wird von DOS verwendet, um auch noch zur Laufzeit Informationen über die Einheit auszuwerten und die Einheit anzupassen.

Die Funktion führt offenbar keine Fehlerprüfung statt, d.h. das AX-Register ist nach dem Aufruf undefiniert.

Der BIOS-Parameter-Block besitzt, in Abhängigkeit von der DOS-Version, folgende Struktur:

```
Ö--------Ú------Ú-------------------------------------------------Ì
º Offset º Bytesº            Funktion                              º
û--------é------é-------------------------------------------------À
º   00   º   2  º Bytes pro Sektor                                  º
û--------é------é-------------------------------------------------À
º   02   º   1  º Sektor pro Cluster - 1                           º
û--------é------é-------------------------------------------------À
º   03   º   2  º Zahl der reservierten Boot-Sektoren              º
û--------é------é-------------------------------------------------À
º   05   º   1  º Zahl der FATs                                    º
û--------é------é-------------------------------------------------À
º   06   º   2  º Zahl der Einträge im Hauptverzeichnis           º
û--------é------é-------------------------------------------------À
º   08   º   2  º maximale Clusternummer (0 ab DOS 4.0 , > 32 MB) º
û--------é------é-------------------------------------------------À
º   0A   º   1  º Media ID                                         º
û--------é------é-------------------------------------------------À
º   0B   º   2  º Sektoren pro FAT                                 º
û--------é------é-------------------------------------------------À
º  ----  º  --- º --- Aufbau für DOS 3.x ---                      º
û--------é------é-------------------------------------------------À
º   0D   º   2  º Zahl der Sektoren pro Spur                       º
û--------é------é-------------------------------------------------À
º   0F   º   2  º Anzahl Schreib/Leseköpfe                         º
û--------é------é-------------------------------------------------À
º   11   º   4  º Anzahl reservierter Sektoren (hidden sectors)   º
û--------é------é-------------------------------------------------À
º   15   º  11  º reserviert                                       º
û--------é------é-------------------------------------------------À
º  ----  º  --- º --- Aufbau für DOS 4.x ---                      º
û--------é------é-------------------------------------------------À
º   15   º   4  º Gesamtzahl der Sektoren für Speichermedien mit  º
º        º      º mehr als 32 Mbyte Kapazität                     º
û--------é------é-------------------------------------------------À
º   19   º   6  º reserviert                                       º
û--------é------é-------------------------------------------------À
º   1F   º   2  º Anzahl der Zylinder                              º
û--------é------é-------------------------------------------------À
º   21   º   1  º Geräte Typ                                       º
û--------é------é-------------------------------------------------À
º   22   º   2  º Geräte Attribut                                  º
Û--------Ù------Ù-------------------------------------------------ì
```

Tabelle 4.32: Der Aufbau des BIOS-Parameter-Blocks (BPB)

Der Aufbau des Disk-Parameter-Block (*DPB*) wurde bereits detailliert bei der Funktion *32H* (Get DPB) besprochen. Die Kodierung des Media-ID-Byte ist bei der INT 21-Funktion 1BH beschrieben.

4.81 Get Verify State (Funktion 54H, DOS 2.0-6.x)

Mit dieser Funktion läßt sich der Zustand des Verify-Flags abfragen. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:    54H                      °
û---------------------------------À
°      RETURN                     °
° AL:  Status des Verify Flags    °
°      00H -> Verify OFF          °
°      01H -> Verify ON           °
Û---------------------------------ì
```

Nach dem Aufruf enthält das AL-Register den Zustand des Verify-Flags. Mit *AL = 0* ist Verify bei Disketten-Schreiboperationen ausgeschaltet, während mit *AL = 1* die Option eingeschaltet ist. Der Schalter läßt sich aus der Kommandoebene mit:

```
Verify = ON   /  OFF
```

beeinflussen. Die Funktion 2EH ermöglicht es ebenfalls den Wert des Verify-Flags zu verändern.

4.82 Install New Process (Funktion 55H, DOS 2.0-6.x, undokumentiert)

Die Funktion *55* wird intern durch DOS benutzt und ist nicht dokumentiert. Mit dieser Funktion läßt sich ein *PSP*-Bereich für einen neuen Prozeß anlegen. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:    55H (Install new process)°
° DX:    Segmentadresse new PSP   °
° SI:    Size field               °
û---------------------------------À
°      RETURN                     °
° CY:  0-> ok                     °
Û---------------------------------ì
```

Im Register DX ist die Segmentadresse, ab der der neue *PSP* abzulegen ist, zu übergeben. Die Funktion kopiert in DOS 2.x den Inhalt des bestehenden *PSP* in den angegebenen Bereich und modifiziert die Eintragungen dann nach den Anforderungen des Prozesses. Ab DOS 3.0 ist in SI der Wert für das »size field« welches sich ab DX:02 findet, einzutragen.»Dieser Aufruf wird durch die Funktion *4BH* abgedeckt, die aus Sicht des Anwenderprogrammierers zu bevorzugen ist. Die Funktion gibt keine Fehlercodes zurückzugeben.

4.83 Rename File (Funktion 56H, DOS 2.0-6.x)

Mit dieser Funktion läßt sich der Name einer Datei ändern. Es gelten folgende Aufrufparameter:

```
ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:    56H                      °
° DS:DX  Zeiger auf einen ASCIIZ- °
°        String mit altem Pfad    °
° ES:DI  Zeiger auf einen ASCIIZ- °
°        String mit neuem Pfad    °
û---------------------------------À
°      RETURN                     °
° Carry: gesetzt -> Fehler        °
° AX:    Fehlercode               °
° Carry: gelöscht -> ok           °
Û---------------------------------ì
```

Das Registerpaar *DS:DX* zeigt auf einen ASCIIZ-String mit dem Pfad- und Dateinamen der Datei, die umbenannt werden soll. Im *ES:DI* ist ein Zeiger auf einen zweiten ASCIIZ-String mit dem neuen Pfad und Dateinamen enthalten. Falls in diesem String ein Laufwerksname auftritt, muß er mit dem ursprünglich angegebenen Laufwerk übereinstimmen. Der Name des Unterverzeichnisses muß nicht übereinstimmen, wodurch eine Datei in ein anderes Unterverzeichnis kopiert werden kann.

Das Archive-Attribut wird dabei nicht gesetzt. Ab DOS 3.3 lassen sich mit der Funktion auch Unterverzeichnisse umbenennen. Ab DOS 3.1 läßt sich die Funktion über die undokumentierte Funktion 5D00H aktivieren, wobei dann Wildcardzeichen (*,?) in beschränktem Umfang im Filenamen auftreten dürfen. Bei erfolgreichem Aufruf wird der Fehlercode 18 (dezimal) »no more filesTritt ein Fehler auf, wird das Carry-Flag gesetzt und im *AX*-Register findet sich ein Fehlercode:

```
 Code ° Bemerkung
------é-------------------------------------------------------------------
  2   ° Einer der Pfadnamen ist ungültig oder nicht zugreifbar
  3   ° Suchweg nicht gefunden
  5   ° Der erste Pfadname spezifiziert ein Inhaltsverzeichnis,
      ° der zweite Pfadname eine Datei, oder die zweite Datei kann nicht
      ° geöffnet werden
 17   ° Die zwei Dateien befinden sich nicht auf der gleichen Einheit
      ° (Laufwerk)
```

Mittels der Funktion *59H* lassen sich die erweiterten Fehlercodes abfragen.

4.84 Get/Set File Date and Time (Funktion 57H, DOS 2.0-6.x)

Mit dieser Funktion lassen sich Datum und Zeit einer Datei lesen und bedingt ändern. Diese Parameter werden normalerweise bei jedem Schreibvorgang auf die Datei durch DOS aktualisiert. Es gelten folgende Aufrufparameter.

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    57H                      º
º AL:    00H (Get Date and Time)  º
º        01H (Set Date and Time)  º
º BX:    Filehandle               º
º CX:    Zeit bei AL = 01         º
º DX:    Datum bei AL = 01        º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º CX:    Zeit bei AL = 00         º
º DX:    Datum bei AL = 00        º
Û---------------------------------ì
```

Der Inhalt des Registers *AL* steuert die Aktion (*AL = 0* Get Data, *AL = 1* Set Data). Im Register *BX* wird der Filehandle übergeben. Im Register *DX* findet sich das Datum und im Register *CX* die Zeit. Das Format entspricht dem Eintrag innerhalb des Inhaltsverzeichnisses. Es wird im Kapitel über den Dateiaufbau beschrieben.

Falls ein Fehler auftritt, wird das Carry-Flag gesetzt und im Register *AX* findet sich ein Fehlercode:

```
  Code º Bemerkung
-------é-----------------------------
   1   º ungültiger Funktionsaufruf
   6   º ungültiger Handle
```

Mittels der Funktion *59H* lassen sich die erweiterten DOS-Fehlercodes abfragen.

Die Funktion 5101H erlaubt es nicht, direkt das Datum und die Zeit einer Datei zu setzen. Vielmehr werden die neuen Werte in der internen System-File-Table (SFT) der Datei zugewiesen. Erst ein Close-Aufruf überträgt die Daten auf das Medium.

Geräte besitzen ebenfalls einen Eintrag in der SFT, dessen Datum/Zeit beim Programmstart gesetzt wird. Damit läßt sich nachträglich die Startzeit eines Programmes abfragen.

4.85 Get/Set Allocation Strategie (Funktion 58H, DOS 2.0-6.x)

Diese Funktion erlaubt es, die Strategie des DOS-Memory-Managers zu lesen und zu verändern. Die Funktion wird für DOS interne Zwecke benutzt und ist selten beschrieben.

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    58H                      º
º AL:    00H (Get Strategie)      º
º        01H (Set Strategie)      º
º        02H (Get UMB link status)º
º        03H (Set UMB link status)º
º BX:    Strategiecode (bei Set)  º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: nicht gesetzt            º
º AX:    Strategiecode (bei Get)  º
Û---------------------------------ì
```

Es ist davon auszugehen, daß es sich ebenfalls um eine undokumentierte Funktion handelt. Es gelten die nebenstehenden Aufrufparameter.

Das Register *AL* bestimmt, ob die Einstellung (*AL = 00*) verändert (*AL = 01*) werden soll. Ab DOS 5.0 sind zusätzlich die Subcodes AL= 02H (Get UMB link state) und AL=03H (Set UMB link state) implementiert. Bei der Set Operation (AL=01H) ist im Register *BX* ein Code zur Einstellung der Verwaltungsstrategie zu übergeben. Bei der Get-Operation (AL = 00H) wird die aktuelle Einstellung im Register *AX* zurückgegeben.

Hierbei gilt folgende Kodierung:

```
AX = Code º Strategie
----------é-----------
   00H    º Low Memory First fit
   01H    º Low Memory Best fit
   02H    º Low Memory Last fit
  -----   º ab DOS 5.0
   40H    º High Memory First fit
   41H    º High Memory Best fit
   42H    º High Memory Last fit
   80H    º First fit, High/Low
   81H    º Best fit, High/Low
   82H    º Last fit, Hight/Low
```

Die drei Strategien bestimmen, wie eine Speicherplatzanforderung zu bearbeiten ist. Bei der »first fit«Methode beginnt DOS mit der Suche nach einem freien Block am unteren Speicherende. Der erste freie Block mit der entsprechenden Größe wird zugeordnet. Bei der »best fit«Methode wird der gesamte Speicher durchsucht und der Block der am besten von der Größe her paßt wird zugeordnet. Bei der »last fit«Strategie beginnt die Suche im oberen Bereich und der erste passende Block wird zugewiesen.»Ab DOS 5.0 wird weiterhin unterschieden, ob der konventionelle Speicher bis 640 Kbyte, oder Upper Memory bis 1 Mbyte zu benutzen ist. Ein Code von 80H bis 82H bewirkt, daß DOS zuerst eine Speicherreservierung im High Memory versucht. Mißlingt dies, wird der Speicher im Low Memory (bis 640 Kbyte) reserviert.

Normalerweise ist in MS-DOS die »first fit«Strategie eingestellt. Eine genaue Beschrei_bung der Strategien sowie der Konsequenzen findet sich im Kapitel über den DOS-Memory-Manager.»Mit der Subfunktion AL = 02H läßt sich ab DOS 5.0 abfragen, ob UMB benutzt wird. Das Ergebnis findet sich im Register AL:

AL = 00H UMB nicht Teil der DOS-Speicherkette
01H UMB ist Teil der DOS-Speicherkette

Mit der Subfunktion AL = 03 läßt sich ab DOS 5.0 definieren, ob UMB benutzt werden sollen. Die Strategie ist in BX zu übergeben:

BX = 0000H UMB aus DOS-Speicherkette entfernen
0001H UMB in DOS-Speicherkette aufnehmen

Tritt während der Operation ein Fehler auf, wird das Carry-Flag gesetzt und im *AX*-Register findet sich ein Fehlercode:

```
1  Ungültiger Funktionsaufruf
```

Die erweiterten DOS-Fehlercodes lassen sich mit der Funktion *59H* abfragen.

4.86 Get Extended Error (Funktion 59H, DOS 3.0-6.x)

Normalerweise übergibt DOS die Fehlercodes im Register AX und setzt das Carry-Flag. Ab Version 3.0 existiert die Möglichkeit, erweiterte Fehlercodes abzufragen, die auf die älteren Codes abgebildet wurden. Hierfür steht die Funktion 59 mit folgender Parameterschnittstelle zur Verfügung:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    59H                      º
º BX:    0 (DOS 3.0 - 6.x)        º
û---------------------------------Ã
º      RETURN                     º
º AX:    erweiterter Fehlercode   º
º BH:    Fehlerklasse             º
º BL:    vorgeschlagene Aktion    º
º CH:    Fehlerhinweis            º
Û---------------------------------ì
```

Der Fehlercode kann nur einmal abgefragt werden. Bei der Abfrage werden die Register *CL*, *DX*, *SI*, *DI*, *BP*, *DS*, *ES* zerstört, daß Anwenderprogramm muß also diese Register vor dem Aufruf retten. In jeder neuen DOS-Version werden weitere Fehlercodes implementiert. Ein bereits bestehendes Programm kann auch mit neueren DOS-Versionen arbeiten, da die alten Fehlermeldungen erhalten bleiben.

Um eine zukünftige Erweiterung der Funktionalität zu ermöglichen, wird beim Aufruf im Register *BX* ein Versionscode übergeben. In den DOS-Versionen 3.0 bis 6.x ist der Wert immer Null.

In den Registern *AX*, *BX* und *CX* wird dann ein erweiterter Fehlercode zurückgegeben, der genauere Hinweise auf die Fehlerursache und deren Beseitigung gibt. Im Register *AX* findet sich ein erweiterter Fehlercode. Die Funktion versucht hier die Fehlermeldungen älterer DOS-Versionen abzubilden. Der übergebene Wert entspricht den üblicherweise im Register AL übergebenen Fehlercodes. Falls dieser Code nicht mehr existiert, wird die am besten passende Fehlernummer zugeordnet. Eine Beschreibung der Fehlercodes in AX findet sich im Kapitel »Fehlerbehandlung in DOS«»Im Register *BH* findet sich ein Code, der die Fehlerklasse beschreibt. Es gilt folgende Nomenklatur:

```
 Code ° Fehlerklasse
------é-------------------------------------------------------------------
  01  ° Fehlende Resourcen (z.B. kein Speicher mehr frei, oder alle
      ° Kanäle belegt)
  02  ° Hierbei handelt sich sich nicht um einen Fehler, sondern um
      ° eine temporäre
      ° Situation, die einen einwandfreien Ablauf verhindert (z.B.
      ° Zugriffsversuch auf einen »locked  03  ° Fehlende oder falsche
Autorisierung (z.B. kein »Create
      ° Access
```

Fehler! Verweisquelle konnte nicht gefunden werden.

```
  04  °
Fehlerursache in der internen Systemsoftware
  05  ° Hardwarefehler
  06  ° Fehler in der Systemsoftware, der nicht durch den laufenden
      ° Prozeß bewirkt wurde (z.B. fehlende oder falsche Konfigurations-
      ° dateien)
  07  ° Fehler im Anwendungsprogramm
  08  ° Datei oder Einheit nicht gefunden
  09  ° Datei oder Einheit besitzen ein falsches Format oder Typ
  0A  ° Datei oder Einheit blockiert (locked)
  0B  ° Falsche Diskette im Laufwerk, oder die Diskette ist beschädigt,
      ° oder sonstige Probleme mit dem Speichermedium
  0C  ° Spezifizierter Name existiert bereits (z.B. Versuch einen Namen
      ° innerhalb eines Netzwerkes mehrfach zu vergeben)
  0D  ° andere Fehlerursache
```

Diese Hinweise lassen sich dann in eine entsprechende Fehlermeldung umsetzen.

Als weitere Hilfestellung liefert die Funktion einen Code im Register *BL*, der Hinweise zur weiteren Vorgehensweise enthält. Es gilt folgende Kodierung:

```
 Code ° Aktion
------é-------------------------------------------------------------
  01  ° Wiederhole Versuch und gebe dann einen Hinweis an den Benutzer
  02  ° Wiederhole den Versuch nach einer Pause
  03  ° Falls der Anwender Datei-, Pfad- oder Laufwerksnamen eingegeben
      ° hat, wiederhole die Abfrage
  04  ° Setze die benutzten Resourcen zurück und beende das Programm
      ° (z.B. Dateien schließen)
  05  ° Das System ist so stark gestört, daß das Programm sofort zu
      ° beenden ist (Dateien sollten nicht mehr geschlossen werden)
  06  ° Der gemeldete Fehler dient nur zur Information
  07  ° Fehlermeldung mit Abfrage an den Benutzer (z.B. Disketten
      ° einlegen), dann wiederhole den Versuch noch einmal
```

Als weitere Information findet sich im Register *CH* ein Hinweis zur Lokalisierung des Fehlers. Es gelten folgende Konventionen:

```
 Code ° Bemerkung
------é-----------------------------------------------------------------
  01  ° unbekannt
  02  ° Der Fehler trat bei einem wahlfreien Zugriff auf eine
      ° blockorientierte Einheit (z.B. Diskette) auf
  03  ° Der Fehler trat im Netzwerk auf
  04  ° Der Fehler trat bei einem sequentiellen Zugriff auf
      ° eine zeichenorientierte Einheit auf (z.B. Printer
      ° nicht bereit)
  05  ° Der Fehler trat bei einem Zugriff auf den RAM-Speicher auf
```

Die erweiterten Fehlercodes sollten nur innerhalb folgender Konstellationen benutzt werden:

- Innerhalb eines *INT 24*-Handlers, um auch Netzwerkfehler abzufangen

- Bei Verwendung von *INT 21*-Funktionsaufrufen, falls das Carry-Flag gesetzt ist und der Fehlercode im Register *AX* zurückgegeben wird.
- Bei *FCB*-Funktionsaufrufen, die als Fehler den Wert *FFH* im *AL*-Register zurückgeben.

An Hand der zurückgegebenen Werte, insbesondere im *BL*-Register, läßt sich dann die weitere Vorgehensweise bestimmen.

4.87 Create Temporary File (Funktion 5AH, DOS 3.0-6.x)

Dieser Aufruf legt eine temporäre Datei im angegebenen Verzeichnis an. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:    5AH                      °
° DS:DX  Zeiger auf einen ASCIIZ- °
°        String mit dem Pfad      °
° CX:    Dateiattribute           °
û---------------------------------À
°      RETURN                     °
° Carry: gesetzt -> Fehler        °
° AX:    Fehlercode               °
° Carry: gelöscht -> ok           °
° DS:DX  Zeiger auf einen ASCIIZ- °
°        String mit dem Pfad      °
Û---------------------------------ì
```

Bei Aufruf der Funktion zeigt das Registerpaar *DS:DX* auf einen ASCIIZ-String mit dem Pfadnamen. Dieser Pfad darf keinen Dateinamen enthalten und muß mit einem Backslash \ enden. Fehlt der Suchweg, legt DOS die Datei nicht im aktuellen, sondern im Stammverzeichnis an. Weiterhin sind in diesem String 13 Byte für den später zugeordneten Namen der temporären Datei zu reservieren. Im Register *CX* wird das Attributbyte der neuen Datei angegeben. Die Kodierung ist im Kapitel über die DOS-Dateistrukturen angegeben.

Die Funktion erzeugt nun einen Dateinamen und versucht die Datei anzulegen. Existiert bereits eine Datei des gleichen Namens, wird so lange ein neuer Name generiert, bis dieser nicht mehr im System vorkommt. Dann wird die Datei geöffnet. Der Name wird vermutlich aus Datum und Uhrzeit abgeleitet.

Nach dem fehlerfreien Aufruf enthält der Pfad den Namen der temporären Datei. Es ist aber zu beachten, daß die temporäre Datei nicht automatisch beim Programmende gelöscht werden.

Tritt während des Aufrufes ein Fehler auf, dann wird das Carry-Flag gesetzt und im Register *AX* findet sich ein Fehler.

```
  Code ° Fehler
-------é-----------------------------------
   03  ° Pfad nicht gefunden
   04  ° zuviele offene Dateien
   05  ° Zugriff auf die Einheit abgewiesen
   80  ° Datei existiert bereits
```

Um in einem Netzwerk eine temporäre Datei anzulegen, benötigt der rufende Prozeß das »Create Access«Privileg. Andernfalls kann z.B. der Fehler 5 auftreten. Die erweiterten Fehlercodes lassen sich mit der Funktion 59H abfragen.»4.88 Create New File (Funktion 5BH, DOS 3.0-6.x)

Mit dieser Funktion läßt sich eine neue Datei erzeugen. Er ist identisch zur älteren DOS-Funktion *3CH* mit der Ausnahme, daß keine Datei angelegt wird, falls der Name bereits existiert. Es gelten folgende Aufrufparameter:

```
ö--------------------------------ì
°      CALL:  INT 21             °
°                                °
° AH:    5BH                     °
° DS:DX  Zeiger auf einen ASCIIZ- °
°        String mit dem Pfad     °
° CX:    Dateiattribute          °
û--------------------------------À
°      RETURN                    °
° Carry: gesetzt -> Fehler       °
° AX:    Fehlercode              °
° Carry: gelöscht -> ok          °
° AX:    Filehandle              °
Û--------------------------------ì
```

Während die Funktion *3CH* eine bestehende Datei auf die Länge 0 setzt und damit neu anlegt, wird der Aufruf *4BH* abgewiesen, falls der angegebene Name bereits existiert. Damit ist sichergestellt, daß keine Datei versehentlich überschrieben wird. Im Registerpaar *DS:DX* wird ein Zeiger auf einen ASCIIZ-String übergeben. Dieser String enthält den Pfadnamen, einschließlich Laufwerks- und Dateibezeichnung. Der String ist mit einem Nullbyte abzuschließen.

Im Register *CX* können die Dateiattribute übergeben werden (siehe Kapitel über den DOS-Dateiaufbau).

Die neue Datei wird im »Compatibility«Modus angelegt, d.h. die Datei kann gelesen und beschrieben werden. Der Schreib-Lese-Zeiger wird auf das erste Byte positioniert. Inner_halb eines Netz_werkes ist das »Create Access«Privileg für die Funktion er_for_der_lich. Ab DOS 4.0 existiert die Funktion 6CH um eine Datei mit SHARE-Flags zu öffnen.»Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt. Im *AX*-Register findet sich ein Fehlercode:

```
  Code ° Fehler
-------é-----------------------------
   3   ° Ungültiger Pfad
   4   ° Zu viele offene Dateien
   5   ° Zugriff abgewiesen
  80   ° Die Datei existiert bereits
```

Die erweiterten Fehlercodes lassen sich mit der Funktion *59H* abfragen. Falls das Carry-Flag nicht gesetzt ist, wird im *AX*-Register der Filehandle zurückgegeben.

4.89 Lock/Unlock File Access (Funktion 5CH, DOS 3.0-6.x)

Mit Hilfe dieser Funktionen lassen sich bestimmte Sätze einer Datei für den Zugriff anderer Prozesse sperren und auch wieder freigeben. Dies ist insbesondere in Netzwerken notwendig, um zu vermeiden, daß zwei Prozesse die Daten gleichzeitig manipulieren und somit falsche Ergebnisse auftreten. Es gelten folgende Aufrufparameter:

```
Ö--------------------------------Ì
°      CALL:  INT 21             °
°                                °
° AH:    5CH                     °
° AL:    00H (Lock Datei)        °
°        01H (Unlock Datei)      °
° BX:    Filehandle              °
° CX:    Offset Highbyte         °
° DX:    Offset Lowbyte          °
° SI:    Länge Highbyte          °
° DI:    Länge Lowbyte           °
û--------------------------------À
°      RETURN                    °
° Carry: gesetzt -> Fehler       °
° AX:    Fehlercode              °
° Carry: gelöscht -> ok          °
Û--------------------------------ì
```

Im Register *BX* muß der Handlecode der jeweiligen Datei übergeben werden. Dieser Code wird beim Öffnen der Datei zurückgegeben. Der interessierende Bereich innerhalb der Datei wird mit den Registern *CX*, *DX*, *SI*, *DI* markiert. In *CX:DX* findet sich ein 4-Byte-Zeiger auf den Beginn des zu sperrenden/ freizugebenden Bereichs. Die Länge dieses Bereichs (Bytes) findet sich in *SI:DI*.

Der Wert im Register *AL* gibt an, ob der Dateibereich gesperrt oder freigegeben werden soll.

Die Funktion ist nur gültig, falls ein Netzwerktreiber, oder das DOS-Programm SHARE geladen ist.

Lock (AL = 00)

Mit dieser Unterfunktion läßt sich ein Dateibereich exclusiv für den Zugriff des gerade laufenden Prozesses reservieren, d.h. kein anderer Prozeß kann diese Daten lesen oder schreiben. Falls dies doch versucht wird, wird der Zugriff abgewiesen. Nach einigen Wiederholungen wird an diesen Prozeß eine Fehlermeldung abgesetzt. Die Zahl der Wiederholungen ist abhängig von der Systemeinstellung (siehe auch Funktion *440BH*). In diesem Fehlerfall kann der abgewiesene Prozeß die erweiterten Fehlercodes über die Funktion *59H*

abfragen.Es kann jeder beliebige Bereich innerhalb der Datei für Zugriffe anderer Prozesse gesperrt werden. Eine Sperre über das Dateiende hinaus wird nicht als Fehler betrachtet.

Die Sperre sollte durch den Prozeß nur für die Dauer des Zugriffs gesetzt werden, da dann alle anderen Prozesse vom Zugriff ausgeschlossen werden. Falls ein Prozeß mit geöffneten Dateien abgebrochen wird, treten undefinierte Resultate auf, wenn die Zugriffssperre vorher nicht aufgehoben wird. Dies ist insbesondere bei der Erstellung von Control-C und Critical-Error-Handlern zu beachten.

Wird ein Handle dupliziert, ist der gesperrte Bereich auch beim zweiten Filehandle bekannt. Wird ein Subprozeß mittels der EXEC-Funktion erzeugt, bleibt auch dieser Subprozeß vom Zugriff auf die gesperrten Daten ausgeschlossen.

Die Prüfung auf Zugriffsmöglichkeiten bei gesperrten Dateibereichen sollte über die Funktion *5CH* erfolgen. Ist der Bereich bereits gesperrt, wird das Carry-Flag gesetzt und im Register *AX* findet sich ein Fehlercode:

```
  Code ° Fehler
-------é-----------------------------------------------
    1  ° Das DOS-Share-Kommando ist nicht geladen
    6  ° Der Handle ist ungültig oder nicht geöffnet
   33  ° Der Bereich ist ganz oder teilweise gesperrt
```

Über die Funktion 59H lassen sich die erweiterten Fehlercodes abfragen. Die Funktion sollte nur auf Dateien angewandt werden, die mit dem »deny read Fehler! Verweisquelle konnte nicht gefunden werden.**deny none**Fehler! Verweisquelle konnte nicht gefunden werden.**Unlock (AL = 01)**

Hier handelt es sich um die Gegenfunktion, die einen gesperrten Bereich wieder freigibt. Der Bereich muß exakt mit den gleichen Parametern wie im Lock-Befehl spezifiziert werden. Ursache ist die Speicherung der gesperrten Bereich durch SHARE in internen Tabellen, die nur bei Übereinstimmung Einträge findet und löscht. Vor einem Programmabbruch muß die Unlockfunktion auf geöffnete Dateien ausgeführt werden, da sonst undefinierte Resultate auftreten. Dies ist insbesondere bei *INT 23*-und *INT 24*-Handlern zu beachten.

Wird ein Handle dupliziert, ist der freigegebene Bereich auch beim zweiten Filehandle bekannt.

Tritt ein Fehler auf, wird das Carry-Flag gesetzt und im Register *AX* findet sich der Fehlercode:

```
  Code ° Fehler
-------é-----------------------------------------------
    1  ° Das DOS-Share-Kommando ist nicht geladen
    6  ° Der Handle ist ungültig oder nicht geöffnet
   33  ° Der Bereich ist ganz oder teilweise durch den
       ° aktiven Prozeß gesperrt
```

Die Fehlermeldung 33 tritt auf, falls mit einem zweiten *LOCK*-Aufruf geprüft wird, ob der spezifizierte Bereich noch gesperrt ist. Sperren auf Geräte werden ignoriert, führen aber zu keiner Fehlermeldung.

Über die Funktion *59H* lassen sich die erweiterten Fehlercodes abfragen. Die Funktion sollte nur auf Dateien angewandt werden, die mit dem »deny read**Fehler! Verweisquelle konnte nicht gefunden werden.deny none**Fehler! Verweisquelle konnte nicht gefunden werden.4.90 Interne Funktion (Funktion 5DH, DOS 3.1 -6.x)

Diese Funktion ist nicht dokumentiert und wird intern ab DOS 3.1 benutzt. Die Funktion erlaubt verschiedene indirekte Aufrufe der INT 21-Funktionen (Server Calls). Es sind die Codes AL= 00H - 0FH belegt.

4.90.1 Indirect Function Call (AX = 5D00H)

Mit diesem Aufruf lassen sich ab DOS 3.1 interne Funktionen aktivieren. Es gilt folgende Aufrufschnittstelle:

```
Ö----------------------------------Ì
°       CALL:  INT 21               °
°                                   °
° AH:  5DH (Internal)               °
° AL:  00H (indirect CALL)          °
° DS:DX  Bufferadresse              °
û----------------------------------À
°       RETURN                      °
° ---                               °
Û----------------------------------ì
```

In DS:DX ist eine Adresse auf eine Parameterliste zu übergeben. Diese besitzt folgenden Aufbau:

```
Ö-------Ú------Ú---------------------------------------Ì
° Offs. ° Byte ° Feld                          °
û-------é------é---------------------------------------À
°  00H  °  2   ° AX                           °
°  02H  °  2   ° BX                           °
°  04H  °  2   ° CX                           °
°  06H  °  2   ° DX                           °
°  08H  °  2   ° SI                           °
°  0AH  °  2   ° DI                           °
°  0CH  °  2   ° DS                           °
°  0EH  °  2   ° ES                           °
°  10H  °  6   ° reserviert                    °
°  16H  °  2   ° ID der Maschine (0 = eigen)         °
°  18H  °  2   ° Prozess-ID (PSP-Seg. rufend. Prozess) °
Û-------Ù------Ù---------------------------------------ì
```

Diese Funktion erlaubt einen Server Call durch andere Maschinen, mit dem sich DOS-INT 21-Funktionen indirekt aufrufen lassen. Im vorletzten Wort der Parameterliste steht z.B. die ID der rufenden Maschine. Beim Wert 0 wurde der Aufruf durch die eigene Maschine abgesetzt. Die ersten Worte enthalten die Daten für die Registerbelegung. Die Funktion lädt die betreffenden Register mit den Werten und ruft die gewünschte INT 21-Funktion auf. Bei der Rückkehr sind die Register gemäß der aktivierten INT 21-Funktion gesetzt.

Enthält AX vor dem Aufruf ungültige Werte, stürzt das System ab. Die Funktion erlaubt zum Beispiel den Aufruf der INT 21-Funktionen 41H (Delete) und 56H (Rename) mit Wildcardzeichen im Dateinamen. Funktionen, die Filenamen benutzen, benötigen ein Format wie es die INT 21-Funktion 60H liefert. Weiterhin bleiben die durch SHARE gesetzten Wiederholungszeiten ungültig.

4.90.2 Commit all Files (AX = 5D01H)

Mit diesem Aufruf lassen sich ab DOS 3.1 alle Dateien für eine definierte Maschine oder einen Prozess aktualisieren (commit). Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:  5DH (Internal)             °
° AL:  01H (Commit Files)         °
° DS:DX  Bufferadresse            °
û---------------------------------À
°      RETURN                     °
° CY: 1 : Fehler                  °
° AX: Fehlercode                  °
° CY: 0 : ok                      °
Û---------------------------------ì
```

In DS:DX ist eine Adresse auf eine Parameterliste zu übergeben. Diese besitzt eine Struktur wie bei der Funktion 5D00H, wobei allerdings nur die beiden letzten Einträge *ID* und *Prozess ID* benutzt werden.

Wird diese Funktion aufgerufen, lagert DOS alle Puffer in die Dateien aus und aktualisiert Datum und Uhrzeit. Nach dem Aufruf signalisiert ein gesetztes Carry-Flag, daß ein Fehler aufgetreten ist. Der Fehlercode findet sich dann in AX, die Codierung entspricht der INT 21-Funktion 59H.

4.90.3 SHARE Close all Files by Name (AX = 5D02H)

Mit diesem Aufruf lassen sich ab DOS 3.1 Dateien die mit SHARE geöffnet wurden über ihren Namen schließen. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:  5DH (Internal)             °
° AL:  02H (Close Files)          °
° DS:DX  Bufferadresse            °
û---------------------------------À
°      RETURN                     °
° CY: 1 : Fehler                  °
° AX: Fehlercode                  °
° CY: 0 : ok                      °
Û---------------------------------ì
```

In DS:DX ist eine Adresse auf eine Parameterliste zu übergeben. Diese besitzt eine Struktur wie bei der Funktion 5D00H, wobei allerdings nur die beiden letzten Einträge ID und Prozess ID, sowie die Registerwerte DS und DX benutzt werden. In dem durch DS:DX adressierten Puffer ist in den Einträgen für die Register DS:DX die Adresse auf den Namen der zu schließenden Datei zu übergeben. Der Dateiname ist als ASCIIZ-String in

kanonischer Form abzulegen. Diese Form wird durch die Funktion AH = 60H des INT 21 erzeugt.

Falls SHARE nicht geladen ist, gibt die Funktion einen Fehlercode (Carry-Flag gesetzt) zurück. Der Fehlercode findet sich dann in AX, die Codierung ist der Funktion 59H des INT 21 zu entnehmen.

4.90.4 SHARE Close all Files for a Computer (AX = 5D03H)

Mit diesem Aufruf lassen sich ab DOS 3.1 alle Dateien einer angegebenen Station über SHARE schließen. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:  5DH (Internal)             °
° AL:  03H (Close all Files)      °
° DS:DX  Bufferadresse            °
û---------------------------------À
°      RETURN                     °
° CY: 1 : Fehler                  °
° AX: Fehlercode                  °
° CY: 0 : ok                      °
Û---------------------------------ì
```

In DS:DX ist eine Adresse auf eine Parameterliste zu übergeben. Diese besitzt eine Struktur wie bei der Funktion 5D00H, wobei allerdings nur der ID-Eintrag benutzt wird. Bei erfolgreichem Aufruf werden alle Dateien einer Station geschlossen.

Falls SHARE nicht geladen ist, gibt die Funktion einen Fehlercode (Carry-Flag gesetzt) zurück. Der Fehlercode findet sich dann in AX, die Codierung ist der Funktion 59H des INT 21 zu entnehmen.

4.90.5 SHARE Close all Files for a Process (AX = 5D04H)

Mit diesem Aufruf lassen sich ab DOS 3.1 alle Dateien eines angegebenen Prozesses über SHARE schließen. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:  5DH (Internal)             °
° AL:  04H (Close all Files)      °
° DS:DX  Bufferadresse            °
û---------------------------------À
°      RETURN                     °
° CY: 1 : Fehler                  °
° AX: Fehlercode                  °
° CY: 0 : ok                      °
Û---------------------------------ì
```

In DS:DX ist eine Adresse auf eine Parameterliste zu übergeben. Diese besitzt eine Struktur wie bei der Funktion 5D00H, wobei allerdings nur die Einträge Prozess-ID und ID benutzt werden. Bei erfolgreichem Aufruf werden alle Dateien eines Prozesses geschlossen.

Falls SHARE nicht geladen ist, gibt die Funktion einen Fehlercode (Carry-Flag gesetzt) zurück. Der Fehlercode findet sich dann in AX, die Codierung ist der Funktion 59H des INT 21 zu entnehmen.

4.90.6 SHARE Get Open File List Entry (AX = 5D05H)

Mit diesem Aufruf läßt sich ab DOS 3.1 die Liste der offenen Dateien ermitteln. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 21               º
º                                  º
º AH:  5DH (Internal)              º
º AL:  05H (Get File List)         º
º DS:DX  Bufferadresse             º
û---------------------------------À
º      RETURN                      º
º CY: 1 : Fehler                   º
º AX: Fehlercode                   º
º CY: 0 : ok                       º
º BX: Netzwerk ID                  º
º CX: Lock Zahl                    º
º ES:DI ASCIIZ Filename            º
Û---------------------------------ì
```

In DS:DX ist eine Adresse auf eine Parameterliste zu übergeben. Diese besitzt eine Struktur wie bei der Funktion 5D00H, wobei allerdings nur einige Felder belegt sind. Im Feld BX des Puffers ist ein Index in den Sharing Record (siehe Funktion 52H) einzutragen. CX (im Puffer) enthält einen Index der SFT in die Sharing Record SFT Liste.

Falls SHARE nicht geladen ist, gibt die Funktion einen Fehlercode (Carry-Flag gesetzt) zurück. Der Fehlercode findet sich dann in AX, die Codierung ist der Funktion 59H des INT 21 zu entnehmen. Falls der übergebene Index ungültig ist, gibt die Funktion den Fehlercode 18 (no more files) zurück.

Bei erfolgreichem Aufruf ist das Carry-Flag gelöscht und BX enthält die Nummer der Maschine im Netzwerk, welche die SFT besitzt. CX enthält die Zahl der Sperren (Locks) die der Besitzer in der SFT eingetragen hat. In ES:DI findet sich ein Zeiger auf den Filenamen, der als ASCIIZ-String abgelegt wurde.

4.90.7 Get Address of DOS Swappable Data Area (AX = 5D06H)

Dieser Aufruf liefert die Adresse und die Adresse des DOS-Datenbereiches zurück. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:  5DH (Internal)             º
º AL:  06H (Get DOS Data Adr.)    º
û---------------------------------À
º      RETURN                     º
º CY: 1 -> Fehler                 º
º AX: Fehlercode                  º
º CY: 0 -> ok                     º
º DS:SI Adresse DOS Datenbereich  º
º CX: Größe Bereich 1             º
º DX: Größe Bereich 2             º
Û---------------------------------ì
```

Der Aufruf wird ab DOS 3.1 unterstützt. Er gibt die Adresse des internen DOS-Datenbereiches an. Dieser Datenbereich enthält alle internen DOS-Daten und kann bei Benutzung reentranter Programme ausgetauscht werden. Nach dem Aufruf enthält DS:SI die Adresse des Bereiches. CX gibt die Größe des auszutauschenden Datenbereiches an, der während einer kritschen DOS-Phase (INDOS-Flag <> 0) auszutauschen ist. In DX steht die Größe des unbedingt auszutauschenden Datenbereiches.

Der DOS-Datenbereich besitzt folgenden Aufbau:

Offset	Bytes	Bedeutung
00H	1	Critical Error Flag, besitzt den Wert 00 falls DOS gerade über den INT 24 aufgerufen wurde.
01H	1	InDOS-Flag, hat während der Aktivitäten in DOS den Wert 1, während des Aufrufes von Routinen aus dem INT 28 den Wert 2 der Wert 0 zeigt an, daß DOS nicht aktiv ist.
02H	1	Laufwerk auf dem der Fehler aufgetreten ist, oder FFH.
03H	1	Ort des zuletzt aufgetretenen Fehlers (s. Funktion 59H)
04H	2	Extended Error Code (letzter Fehler)
06H	1	vorgeschlagene Aktion für den Fehler
07H	1	Fehlerklasse letzter Fehler
08H	4	wird von der Funktion 59H in ES:DI zurückgeliefert (Zeiger auf den letzten Fehler)
0CH	4	Adresse aktuelle DTA
10H	2	Segment aktuelles PSP
12H	2	Zwischenspeicher für SP beim INT 23
14H	2	Exitcode des zuletzt beendeten Prozesses
16H	1	Kennung aktuelles Laufwerk (0=A:,etc.)
17H	1	Extended Break Flag (0 Prüfung nur bei I/O-Operat. 1 Prüfung bei jedem DOS-Aufruf)
--- Datenblock 2 ---		
18H	2	Wert von AX beim INT 21 Aufruf
1AH	2	PSP-Segment beim Sharing/Netzwerk Aufruf
1CH	2	Maschinen Nummer im Netzwerk
1EH	2	First usable memory block bei Speicherallocierung
20H	2	Best usable memory block bei Speicherallocierung
22H	2	Last usable memory block bei Speicherallocierung
24H	2	Speichergröße in Paragraphen (nur beim Init)
26H	2	reserviert
28H	1	Fehler beim INT 24 Aufruf
29H	1	Bit Flag Aktionen beim INT 24
2AH	1	unbekannt
2BH	1	FFH Abbruch mit Ctrl-C, sonst 0
2CH	2	unbekannt
2EH	1	Tag
2FH	1	Monat
30H	1	Jahr (ab 1980)
32H	2	Zahl der Tage seit dem 1.1.1980
34H	1	Wochentag (0=Sonntag)
35H	1	SFT Zeiger (?)
36H	1	Flag <> 0: sicher um INT 28 aufzurufen
37H	1	Flag <> 0: INT 24 Abortabfrage wurde mit Fail quittiert
38H	26	Device Treiber Request Header
52H	4	Zeiger auf Device Treiber Entry Point
56H	22	Device Treiber Request Header
6CH	22	Device Treiber Request Header
82H	1	Typ der PSP-Kopie (00H= einfache Kopie FFH= make child)
83H	1	unbekannt
84H	3	24-Bit-User-Nummer
87H	1	OEM-Versionsnummer
88H	2	unbekannt
8AH	6	CLOCK$-Transfer-Record
90H	2	unbekannt
92H	128	Buffer für Filenamen
112H	128	Buffer für Filenamen
192H	21	Find first/ find next search Datenblock
1A7H	32	Directory Eintrag für gefundene Files
1C7H	81	Kopie der aktuellen Directory Struktur (CDS)
218H	11	FCB-Format Filename
223H	1	unbekannt
224H	11	Wildcard Zielname für Rename über FCB-Format

```
 22FH   9       unbekannt
 238H   1       Attribute Extended FCB
 239H   1       Typ des FCB (0 regulär, FF extended)
 23AH   1       Attribute Directory Suche
 23BH   4       unbekannt
 23FH   1       Flag: signalisiert wie DOS aufgerufen
                wurde (00: direkt per INT 21, FF: Server Call (AX=5D00H)
 240H   2       unbekannt
 242H   1       Flag 0:read 1:write
 243H   3       unbekannt
 246H   1       Flag: insert mode line edit
 247H   1       Flag = FFH -> canonischer Filename bezieht sich
                auf einen existierenden File
 248H   1       unbekannt
 249H   1       Typ der Prozess Termination (00 - 03H)
 24AH   1       unbekannt
 24BH   1       Wert der als 1. Zeichen in den Namen gelöschter
                Dateien eingetragen wird (5EH)
24CH    4       Zeiger auf den Drive Parameter Block für
                Aufrufe während eines kritischen Fehlers
250H    4       Zeiger auf den Stackbereich, welcher beim
                INT 21-Aufruf die User-Register enthält
254H    2       SP-Inhalt beim INT 24-Aufruf
256H    4       unbekannt
25AH    8       unbekannt
262H    1       Media ID-Byte, welches per INT 21, Funktion
                1BH oder 1CH zurückgegeben wird
263H    1       unbekannt
264H    4       Zeiger auf den Device Header
268H    4       Zeiger auf die aktuelle SFT
26CH    4       Zeiger auf die aktuelle Directorystruktur
270H    4       Zeiger auf den FCB des rufenden Prozesses
274H    2       Zahl der SFT auf die sich eine offene Datei bezieht
276H    2       temporärer Speicher für File Handles
278H    4       Zeiger auf den Beginn der JFT der Handle Tabelle
                des Prozesses
27CH    2       Offset im DOS Datensegment zum 1. Argument des
                Filenamens
27EH    2       Offset im DOS Datensegment zum 2. Argument des
                Filenamens
280H    2       Offset der letzten Komponente im Pfadnamen
                (oder FFFFH)
282H    14      unbekannt
292H    4       aktueller Offset in der Datei
296H    12      unbekannt
2A2H    4       Zahl der Bytes, die an die Datei angefügt werden
2A6H    4       unbekannt
2AAH    4       Zeiger auf die bearbeitete SFT
2AEH    2       INT 21 BX-Inhalt
2B0H    2       INT 21 DS-Inhalt
2B2H    2       temporäre Speicherzelle
2B4H    4       Zeiger auf vorhergehenden Call Frame,
                wird auch beim INT 24 aufgerufen
2B8H    21      FindFirst Suchpuffer
2CDH    32      Directory Eintrag für Filenamen, die umbenannt werden
2EDH    331     DOS Stack 1 für Critical Error Aufrufe
403H    35      Scratch SFT
438H    384     DOS Stack 2 Disk INT 21-Funktionen > 0CH, INT 25, INT 26
5B8H    384     DOS Stack 3 Character INT 21-Funktionen 01H bis 0CH
---- DOS 3.2 - 3.3 -----
738H    1       Device Treiber »lookahead«Flag»739H  3       unbekannt
```

Diese Daten können durch TSR-Programme komplett umkopiert werden, so daß eine Reetrance Fähigkeit erreicht werden kann.

Teil 2 des Datenbereiches ist nur dann von TSR-Programmen zu sichern, falls DOS gerade aktiv ist.

4.90.8 Get Redirected Printer Mode (AX = 5D07H)

Dieser Aufruf ist erst ab DOS 3.1 in Netzwerkumgebungen implementiert.

```
Ö---------------------------------Ì
º      CALL:  INT 21               º
º                                  º
º AH:  5DH (Internal)              º
º AL:  07H (Get Printer Mode)      º
û---------------------------------À
º      RETURN                      º
º DL: Mode                         º
Û---------------------------------ì
```

Mit diesem Aufruf läßt sich in einem Netzwerk der Mode eines umgeleiteten Druckers abfragen.

Im Register DL steht nach dem Aufruf der Mode:

```
  DL: 00H  redirected output is combined
      01H  redirected output in separate print jobs
```

Die Funktion gibt keinen Fehlerstatus zurück.

4.90.9 Set Redirected Printer Mode (AX = 5D08H)

Dieser Aufruf ist erst ab DOS 3.1 in Netzwerkumgebungen implementiert.

```
Ö---------------------------------Ì
º      CALL:  INT 21               º
º                                  º
º AH:  5DH (Internal)              º
º AL:  08H (Set Printer Mode)      º
º DL: Mode                         º
û---------------------------------À
º      RETURN                      º
º ----                             º
Û---------------------------------ì
```

Mit diesem Aufruf läßt sich in einem Netzwerk der Mode eines umgeleiteten Druckers setzen.

Im Register DL wird vor dem Aufruf der Mode:

```
  DL: 00H  redirected output is combined
      01H  redirected output in separate
           print jobs, start new job
```

übergeben. Die Funktion gibt keinen Fehlerstatus zurück.

4.90.10 Flush Redirected Printer Output (AX = 5D09H)

Dieser Aufruf ist erst ab DOS 3.1 in Netzwerkumgebungen implementiert.

```
Ö---------------------------------Ì
°       CALL:  INT 21              °
°                                  °
° AH:  5DH (Internal)              °
° AL:  09H (Flush Printer Output)  °
û---------------------------------Ä
°       RETURN                     °
° ----                             °
Û---------------------------------ì
```

Mit diesem Aufruf läßt sich in einem Netzwerk die Ausgabe des Puffers an den Drucker erzwingen.

Anschließend wird ein neuer Druckjob aufgesetzt. Die Funktion gibt keinen Fehlerstatus zurück.

4.90.11 Set Extended Error Info (AX = 5D0AH)

Der Aufruf ist ab DOS 3.1 vorhanden, ist aber erst ab DOS 5.0 offiziell dokumentiert.

```
Ö---------------------------------Ì
°       CALL:  INT 21              °
°                                  °
° AH:  5DH (Internal)              °
° AL:  0AH (Extended Error)        °
° DS:DX  Bufferadresse             °
û---------------------------------Ä
°       RETURN                     °
° ---                              °
Û---------------------------------ì
```

Mit diesem Aufruf setzt DOS 3.1 die internen Extended Error Codes.

Im Registerpaar DS:DX wird die Adresse des folgenden Puffer übergeben.

```
Ö-------Ú------Ú---------------------------------------Ì
° Offs. ° Byte ° Feld                          °
û-------é------é---------------------------------------À
°  00H  °  2   ° Wert für AX für Funktion 59H      °
°  02H  °  2   ° Wert für BX für Funktion 59H      °
°  04H  °  2   ° CX                          °
°  06H  °  2   ° DX                          °
°  08H  °  2   ° SI                         °
°  0AH  °  2   ° DI                          °
°  0CH  °  2   ° DS                          °
°  0EH  °  2   ° ES                          °
°  10H  °  6   ° reserviert                    °
Û-------Ù------Ù---------------------------------------ì
```

In den beiden ersten Worten werden die Codes übergeben, die anschließend bei der Abfrage der Funktion 59H in AX und BX zurückgegeben werden.

4.90.12 Get DOS Swappable Data Areas (AX = 5D0BH)

Dieser Aufruf ist erst ab DOS 4.x implementiert.

```
Ö---------------------------------Ì
º       CALL:  INT 21             º
º                                 º
º AH:  5DH (Internal)             º
º AL:  0BH (Get SWAP Areas)       º
û---------------------------------À
º       RETURN                    º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º DS:SI  Adresse                  º
Û---------------------------------ì
```

Mit diesem Aufruf läßt sich analog zum Aufruf AX=5D06H die Adresse des internen DOS-Datenbereiches ermitteln.

Tritt beim Aufruf ein Fehler auf, wird das Carry-Flag gesetzt. Der Fehlercode findet sich im Register AX. Bei gelöschtem Carry-Flag befindet sich im Registerpaar DS:SI ein Zeiger auf ein DOS-Datenliste (swappable-data-area-list) mit folgendem Format:

```
ö-------Ú------Ú---------------------------------------Ì
º Offs. º Byte º Feld                                   º
û-------é------é---------------------------------------À
º  00H  º  2   º Zahl der Datenbereiche                  º
û-------é------é---------------------------------------À
º ----  º ---- º für jeden Datenbereich                 º
º  xxH  º  4   º Adresse des Datenbereiches              º
º  xxH  º  2   º Länge und Typ des Datenbereiches       º
º       º      º Bit 15 = 1: Daten immer swappen        º
º       º      º        0: Daten nur per DOS swappen    º
º       º      º Bit 0-14: Länge in Byte                º
Û-------Ù------Ù---------------------------------------ì
```

Der Speicher enthält im ersten Wort einen Zähler, der die Zahl der DOS-Datenbereich definiert. Bisher sind in DOS 4.0 3 Datenbereiche vorhanden. Für jeden der Datenbereiche findet sich in der Tabelle dann eine 6-Byte-Datenstruktur. Die ersten 4 Byte enthalten den Adressvektor auf den Beginn der Datenstruktur. Daran schließt sich ein Flag (Word) mit Informationen über den Datenbereich an.

In DOS 4.0 bis 6.x hat der DOS-Datenbereich selbst folgendes Format:

Offset	Bytes	Bedeutung
00H	1	Critical Error Flag, besitzt den Wert 00 falls DOS gerade über den INT 24 aufgerufen wurde.
01H	1	InDOS-Flag, hat während der Aktivitäten in DOS den Wert 1, während des Aufrufes von Routinen aus dem INT 28 den Wert 2 der Wert 0 zeigt an, daß DOS nicht aktiv ist.
02H	1	Laufwerk auf dem der Fehler aufgetreten ist, oder FFH.
03H	1	Ort des zuletzt aufgetretenen Fehlers (s. Funktion 59H)
04H	2	Extended Error Code (letzter Fehler)
06H	1	vorgeschlagene Aktion Fehler
07H	1	Fehlerklasse letzter Fehler
08H	4	wird von der Funktion 59H in ES:DI zurückgeliefert (Zeiger auf den letzten Fehler)
0CH	4	Adresse aktuelle DTA
10H	2	Segment aktuelles PSP
12H	2	Zwischenspeicher für SP beim INT 23
14H	2	RETURN-Code des zuletzt beendeten Prozesses
16H	1	Kennung aktuelles Laufwerk (0=A:,etc.)

```
 17H   1       Extended Break Flag (0 Prüfung nur bei I/O-Operat.
               1 Prüfung bei jedem DOS-Aufruf)
 18H   2       reserviert
  --- Datenblock 2, nur swappen, falls in DOS  ---
 1AH   2       Wert von AX beim INT 21 Aufruf
 1CH   2       PSP-Segment beim Sharing/Netzwerk Aufruf
 1EH   2       Maschinen Nummer im Netzwerk
 20H   2       First usable memory block
               bei Speicherallocierung
 22H   2       Best usable memory block
               bei Speicherallocierung
 24H   2       Last usable memory block
               bei Speicherallocierung
 26H   2       Speichergröße in Paragraphen (nur beim Init)
 26H   8       reserviert
 30H   1       Tag
 31H   1       Monat
 32H   1       Jahr (ab 1980)
 34H   2       Zahl der Tage seit dem 1.1.1980
 36H   1       Wochentag (0=Sonntag)
 37H   3       unbekannt
 38H   30      Device Treiber Request Header
 58H   4       Zeiger auf Device Treiber Entry Point
 5CH   22      Device Treiber Request Header
 72H   30      Device Treiber Request Header
 90H   6       unbekannt
 96H   6       CLOCK$-Transfer-Record
 9CH   2       unbekannt
 9EH   128     Buffer für Filenamen
11EH   128     Buffer für Filenamen
19EH   21      Find first/ find next search Datenblock
1B3H   32      Directory Eintrag für gefundene Files
1D3H   88      Kopie der aktuellen Directory Struktur
22BH   11      FCB-Format Filename
236H   1       unbekannt
237H   11      Wildcard Zielname für Rename über FCB-Format
242H   9       unbekannt
24BH   1       Attribute Extended FCB
24CH   1       Typ des FCB (0 regulär, FF extended)
24DH   1       Attribute Directory Suche
24EH   1       File Open Mode
24FH   3       unbekannt
252H   1       Flag: signalisiert wie DOS aufgerufen
               wurde (00: direkt per INT 21, FF: Server Call (AX=5D00H)
253H   7       unbekannt
25AH   1       Flag = FFH -> canonischer Filename bezieht sich
               auf einen existierenden File
25BH   1       unbekannt
25CH   1       Typ der Prozess Termination (00 - 03H)
25DH   3       unbekannt
260H   4       Zeiger auf den Drive Parameter Block für
               Aufrufe während eines kritischen Fehlers
264H   4       Zeiger auf den Stackbereich, welcher beim
               INT 21-Aufruf die User-Register enthält
268H   6       unbekannt
26EH   2       Segment Disk-Puffer
270H   8       unbekannt
278H   1       Media ID-Byte, welches per INT 21, Funktion
               1BH oder 1CH zurückgegeben wird
279H   5       unbekannt
27EH   4       Zeiger auf die aktuelle SFT
282H   4       Zeiger auf die aktuelle Directorystruktur
286H   4       Zeiger auf den FCB des rufenden Prozesses
28AH   2       Zahl der SFT auf die sich eine offene Datei bezieht
28CH   2       temporärer Speicher für File Handles
28EH   4       Zeiger auf den Beginn der JFT der Handle Tabelle
               des Prozesses
292H   2       Offset im DOS Datensegment zum 1. Argument des
               Filenamens
294H   2       Offset im DOS Datensegment zum 2. Argument des
```

```
              Filenamens
 296H   30     unbekannt
 2B4H   2      Zahl der Byte im partiellen Sektor
 2B6H   2      Zahl der Sektoren
 2B8H   6      unbekannt
 2BEH   4      Zahl der Bytes, die an die Datei angefügt werden
 2C2H   8      unbekannt
 2AAH   4      Zeiger auf die bearbeitete SFT
 2CAH   2      INT 21 BX-Inhalt
 2CCH   2      INT 21 DS-Inhalt
 2CEH   2      temporäre Speicherzelle
 2D0H   4      Zeiger auf vorhergehenden Call Frame,
              wird auch beim INT 24 aufgerufen
 2D4H   2      Open Mode (INT 21,AX=6C00H)
 2D6H   3      unbekannt
 2D9H   4      Speicher für ES:DI bei INT 21, AX=6C00H
 2DDH   2      Extended File Open Action Code (AX=6C00H)
 2DFH   2      Extended File Open Attribute (AX=6C00H)
 2E1H   2      Extended File Open File Mode (AX=6C00H)
 2E3H   4      Zeiger auf Filenamen (AX=6C00H)
 2E7H   5      unbekannt
 2ECH   2      speichert DS zeitweise (Funkt. 52H)
 2EEH   5      unbekannt
 2F3H   4      Zeiger Anwender-Filename
 2F7H   4      unbekannt
 2FBH   4      SS:SP temporär (Funkt. 52H)
 2FFH   1      Flag <> 0, falls Stack umgeschaltet (Funkt. 52H)
 300H   21     Puffer für FindFirst Daten
 315H   32     Directory Eintrag für Rename Files
 334H   331    Critical Error Stack
 480H   384    DOS Stack 2 Disk INT 21-Funktionen > 0CH, INT 25, INT 26
 600H   384    DOS Stack 3 Character INT 21-Funktionen 01H bis 0CH
 780H   1      Device Treiber »lookahead«Flag» 781H 15     unbekannt
```

Eine Sicherung dieser Daten erlaubt einen Aufruf von DOS-Routinen durch TSR-Programme, sofern nicht gerade ein INT 2A ausgeführt wird.

4.91 Get Machine Name (Funktion 5EH, Code 00H, DOS 3.1 - 6.x)

Die Funktion *5EH* bezieht sich auf die Netzwerkfunktionen, die erst ab DOS 3.1 implementiert wurden. Mit dem Aufruf *5E00H* läßt sich der Name einer Maschine innerhalb des Netzwerkes ermitteln. Es gelten folgende Parameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5EH                      º
º AL:    00H                      º
º DS:DX  Zeiger auf einen Puffer  º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º CH:    Flag                     º
º CL:    Nummer                   º
Û---------------------------------ì
```

Das *AL*-Register selektiert mit dem Wert *00H* die Subfunktion »Get Maschine Name« Das Register_paar DS:DX enthält eine Pufferadresse, in dem der ASCIIZ-String mit dem Namen der Maschine zu_rückge_ge_ben wird. Hierbei handelt es sich um einen String der

Länge 15 (Byte), der mit einem Nullbyte abgeschlossen wird. Falls der Name der Mas»hine keine 15 Zeichen umfaßt, wird der restliche Bereich mit Blanks gefüllt. Im Register *CH* wird ein Flag zurückgegeben.

Ist der Name der Maschine nicht definiert, wird der Wert *00H* eingetragen. Dann ist die in *CL* befindliche Nummer ungültig. Andernfalls findet sich hier die NetBIOS-Nummer der Maschine.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in *AX* findet sich der Fehlercode:

```
   Code ° Fehler
--------é-----------------------------------------
     1  ° Die Netzwerksoftware ist nicht geladen
```

Allerdings prüfen die meisten DOS-Versionen den Netzwerktreiber nicht ab, so daß der Fehler kaum auftritt. Die erweiterten Fehlercodes lassen sich durch die Funktion *59H* abfragen. Es ist zu beachten, daß vor Aufruf der Funktion auch auf der lokalen Station die Netzwerksoftware geladen ist, da sonst die Resultate undefiniert sind.

2.92 Set Machine Name (Funktion 5EH, Code 01H, DOS 3.1 - 6.x)

Mit dem Aufruf *5E01H* läßt sich der Name einer Maschine innerhalb des Netzwerkes setzen. Es gelten folgende Parameter:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:    5EH                      °
° AL:    01H                      °
° CH:    Flag                     °
° CL:    Nummer                   °
° DS:DX  Zeiger auf einen Puffer  °
û---------------------------------À
°      RETURN                     °
° ---                             °
Û---------------------------------ì
```

Das *CL*-Register enthält beim Aufruf die Nummer der Maschine. In CH ist ein Flag zu übergeben, welches den Aufruf steuert. Mit CH=00 wird der aktuell eingestellt Name als ungültig markiert. Mit CH <> 00 wird der neue Name gesetzt. Dieser Name muß in einem 15 (Byte) langen Puffers stehen, der mit einem Nullbyte abgeschlossen wird.

Falls der Name der Maschine keine 15 Zeichen umfaßt, wird der restliche Bereich mit Blanks gefüllt. Im Registerpaar DS:SI ist die Adresse des Puffers zu übergeben.

Die Funktion gibt offenbar keine Fehlercodes zurück.

4.93 Set Printer Setup (Funktion 5EH, Code 02H, DOS 3.1-6.x)

Mit dem Aufruf 5E02H läßt sich ein Initialisierungsstring für den Netzwerk-Drucker definieren. Dieser String wird dann vor jeder Ausgabe an den Drucker des Netzwerkes ausgegeben. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5EH                      º
º AL:    02H                      º
º BX:    Zuordnungsliste          º
º CX:    Stringlänge (bis 64 Byte)º
º DS:SI  Zeiger auf einen Puffer  º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
Û---------------------------------ì
```

Im Register *BX* wird ein Zuordnungsindex für die Netzwerkeinheit übergeben. Der Wert kann durch den Funktionsaufruf *5F02H* (Get Redirection List) ermittelt werden. Das Registerpaar *DS:SI* enthält die Anfangsadresse eines Pufferbereiches, in dem der Initialisierungsstring abgelegt ist. Die Länge dieses Strings wird im Register *CX* übergeben, wobei die maximale Länge 64 Byte umfaßt.

Tritt während des Funktionsaufrufes ein Fehler auf, ist nach der Rückkehr das Carry-Flag gesetzt. Im Register *AX* findet sich folgender Fehlercode:

```
  Code º Fehler
-------é-----------------------------------------
   1   º Die Netzwerksoftware ist nicht geladen
  87   º ungültiger Parameter
```

Tritt der Fehler 87 (dezimal) auf, wurde in BX ein ungültiger Index in die Zuordnungsliste übergeben. Vor einem Aufruf der Funktion ist die Netzwerksoftware der Maschine zu laden. Die erweiterten DOS-Fehlercodes lassen sich mittels der Funktion *59H* abfragen.

Es ist zu beachten, daß der Zuordnungsindex für die Einheit bei jedem Aufruf der Funktion *5F03H* (Redirect Device) und bei Funktion *5F04H* (Cancel Redirection) geändert wird. Daher sollte die Set-Funktion sofort nach dem Aufruf *Get Printer Setup* abgesetzt werden.

Mit der Set-Funktion ist es möglich, daß mehrere Benutzer einen Drucker gleichzeitig benutzen. Da vor Ausgabe einer jeden Datei auf diesen Drucker ein Initialisierungsstring vorgeschaltet wird, läßt sich das Gerät individuell nach den Bedürfnissen eines jeden Benutzers einstellen.

4.94 Get Printer Setup (Funktion 5EH, Code 03H, DOS 3.1-6.x)

Mit dem Aufruf *5E03H* läßt sich der Initialisierungsstring für den Drucker abfragen. Dieser String wird vor jeder Ausgabe an den Drucker des Netzwerkes ausgegeben. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5EH                      º
º AL:    03H                      º
º BX:    Zuordnungsliste          º
º ES:DI  Zeiger auf einen Puffer  º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º CX:    Stringlänge (bis 64 Byte)º
Û---------------------------------ì
```

Im Register *BX* wird ein Zuordnungsindex für die Netzwerkeinheit übergeben. Der Wert kann durch den Funktionsaufruf *5F02H* (Get Redirection List Entry) ermittelt werden. Das Registerpaar *ES:DI* enthält die Anfangsadresse eines Pufferbereiches, in dem der Initialisierungsstring zurückgegeben wird. Die Länge dieses Strings findet sich dann im Register *CX*, wobei die maximale Länge 64 Byte umfaßt.

Tritt während des Funktionsaufrufes ein Fehler auf, ist nach der Rückkehr das Carry-Flag gesetzt. Im Register *AX* findet sich folgender Fehlercode:

```
  Code º Fehler
-------é-----------------------------------------
   1   º Die Netzwerksoftware ist nicht geladen
  87   º ungültiger Parameter
```

Vor einem Aufruf der Funktion *5E03H* ist die Netzwerksoftware auf der Maschine zu laden. Die erweiterten DOS-Fehlercodes lassen sich durch die Funktion *59H* abfragen.

Es ist zu beachten, daß der Zuordnungsindex für die Einheit bei jedem Aufruf der Funktion *5F03H* (Redirect Device) und bei Funktion *5F04H* (Cancel Redirection) geändert wird. Daher sollte die Set-Funktion sofort nach dem Aufruf »Get Printer Setup**Fehler! Verweisquelle konnte nicht gefunden werden.4.95Set Printer Mode (Funktion 5EH, Code 04H, DOS 3.1-6.x)**

Mit dem Aufruf 5E04H läßt sich der Mode für den Netzwerk-Drucker definieren. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5EH                      º
º AL:    04H                      º
º BX:    Redirection Index        º
º DX:    Printer Mode             º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
Û---------------------------------ì
```

Im Register *BX* wird ein Zuordnungsindex für die Redirectionsliste übergeben. In DX ist der Mode zu setzen. Hierbei gilt:

```
  DX  Bit 0 = 1: Binärmode
              0: Textmode
```

Im Textmode werden Tabulatoren in Leerzeichen gewandelt.

Tritt während des Funktionsaufrufes ein Fehler auf, ist nach der Rückkehr das Carry-Flag gesetzt. Im Register *AX* findet sich ein Fehlercode, dessen Bedeutung beim INT 21-Aufruf Funktion 59H beschrieben wurde.

4.96Get Printer Mode (Funktion 5EH, Code 05H, DOS 3.1-6.x)

Mit dem Aufruf 5E05H läßt sich der Mode für den Netzwerk-Drucker abfragen. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5EH                      º
º AL:    05H                      º
º BX:    Redirection Index        º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º DX:    Printer Mode             º
Û---------------------------------ì
```

Im Register *BX* wird ein Zuordnungsindex für die Redirectionsliste übergeben. Dieser Index läßt sich über die Funktion 5F02H ermitteln.

Nach einem fehlerfreien Aufruf enthält DX den Mode:

```
  DX  Bit 0 = 1: Binärmode
              0: Textmode
```

Im Textmode werden Tabulatoren in Leerzeichen gewandelt.

Tritt während des Funktionsaufrufes ein Fehler auf, ist nach der Rückkehr das Carry-Flag gesetzt. Im Register *AX* findet sich ein Fehlercode, dessen Bedeutung beim INT 21-Aufruf Funktion 59H beschrieben wurde.

4.97 Get Redirection Mode (Funktion 5FH, Code 00H, DOS 3.1-6.x)

Innerhalb eines Netzwerkes lassen sich bestimmte Einheiten (Drucker etc.) auf andere Rechnerknoten umleiten. Mit dem Aufruf läßt sich der Mode eines umgeleiteten Gerätes ermitteln. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5FH                      º
º AL:    00H                      º
º BL:    Umleitungstyp            º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º BH:    Umleitungsstatus         º
Û---------------------------------ì
```

Das *AL*-Register enthält den Subcode *00H*, während das Register *BL* den Typ des Gerätes enthält. Für die Einträge in BL gilt dabei: 03H = Drucker, 04H = Disk Laufwerk.

Falls der Aufruf ohne Fehler ausgeführt wurde, das Carry-Flag ist gelöscht, enthält das Register *BH* nach dem Aufruf den Status (00 = OFF, 01H = ON).

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in *AX* findet sich der Fehlercode der bei der Funktion 59H beschrieben wurde.

4.98 Set Redirection Mode (Funktion 5FH, Code 01H, DOS 3.1-6.x)

Mit dem Aufruf läßt sich der Mode eines umgeleiteten Gerätes setzen. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5FH                      º
º AL:    01H                      º
º BL:    Umleitungstyp            º
º BH:    Umleitungsmode           º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
Û---------------------------------ì
```

Das *AL*-Register enthält den Subcode *01H*, während das Register *BL* den Typ des Gerätes enthält. Für die Einträge in BL gilt dabei: 03H = Drucker, 04H = Disk Laufwerk. In BH ist der Status für die Umleitung zu übergeben. Hierbei gilt:

```
BH = 00: Off
     01: ON
```

Falls der Aufruf ohne Fehler ausgeführt wurde, ist das Carry-Flag gelöscht. Anernfalls enthält AX einen Fehlercode, der gemäß den Konventionen der Funktion 59H kodiert ist.

4.99 Get Redirection List Entry (Funktion 5FH, Code 02H, DOS 3.1-6.x)

Innerhalb eines Netzwerkes lassen sich bestimmte Einheiten (Drucker etc.) auf andere Rechnerknoten umleiten. Die Zuordnung zwischen logischen Gerätenamen und dem Kommunikationskanal erfolgt über eine Zuordnungsliste. Mit dem Aufruf läßt sich die Zuordnungsliste lesen. Die Umleitungsparameter werden zum Beispiel innerhalb Funktion *5EH* benutzt. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5FH                      º
º AL:    02H                      º
º BX:    Zuordnungsliste          º
º DS:SI  Zeiger auf Puffer mit demº
º        lokalen Einheitennamen   º
º ES:DI  Zeiger auf Puffer mit demº
º        Netzwerknamen            º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º BH:    Status der Einheit       º
º BL:    Typ der Einheit          º
º CX:    Parameterwert            º
Û---------------------------------ì
```

Das *AL*-Register enthält den Subcode *02H*, während das Register *BX* einen Indexwert in die Umleitungstabelle übergibt. Dieser Index spezifiziert, welcher Eintrag zu lesen ist (0 = erster Eintrag).

Als weiteres sind zwei Adreßzeiger für Puffer zu definieren. Der erste Puffer von 16-Byte-Länge dient zur Aufnahme des Namens der lokalen Einheit, die auf einen Netzwerkknoten umgeleitet wurde. Der Vektor wird im Registerpaar *DS:DI* (Segment:Offset) übergeben. Das Registerpaar *ES:DI* enthält die Adresse des zweiten 128-Byte-Puffers, in dem der Name der Netzwerkeinheit abgelegt wird.

Die Listen sind beim Aufruf leer. MS-DOS ermittelt die Namen an Hand der internen Umleitungstabelle, der Eintrag wird durch den Index in *BX* selektiert, und speichert sie in die Puffer als ASCII-Strings um. Pro Aufruf kann nur ein Eintrag der Umleitungstabelle gelesen werden. Diese Einträge sind vorher durch die Funktion *5F03H* zu setzen. Soll die gesamte Liste abgefragt werden, ist der Index (beginnend bei 0) nach jedem Aufruf um den Wert 1 zu erhöhen. Beim Erreichen des Listenendes gibt die Funktion die Fehlermeldung 18 zurück.

Falls der Aufruf ohne Fehler ausgeführt wurde, das Carry-Flag ist gelöscht, enthält das Register *CX* einen gespeicherten Wert, der bei der Definition der Tabelle durch den Benutzer vergeben werden kann. DOS verwaltet dann diesen Code und gibt ihn bei der Abfrage mit aus.

Im Register *BH* findet sich ein Statuswort mit folgender Kodierung:

Bit 0: 0 die abgefragte Einheit ist gültig
1 die Einheit ist nicht definiert

Bit 1-7: reserviert

Im Register *BL* findet sich ein Code zur Kennzeichnung des Typs der lokalen Einheit. Es gilt folgende Zuordnung:

CL	Typ der Einheit
3	Drucker
4	Laufwerk

Die Register *DX* und *BP* werden beim Aufruf der Funktion *5F02H* zerstört.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt und in *AX* findet sich der Fehlercode:

Code	Fehler
1	Die Netzwerksoftware ist nicht geladen
18	Keine weiteren Einträge in der Tabelle

Die erweiterten Fehlercodes lassen sich durch die Funktion *59H* abfragen.

Es ist zu beachten, daß vor Aufruf der Funktion auch auf der lokalen Station die Netzwerksoftware geladen ist, da sonst der Fehler 1 auftritt.

4.100Redirect Device (Funktion 5FH, Code 03H, DOS 3.1-6.x)

Mit dem Aufruf *5F03H* läßt sich eine neue Zuordnung innerhalb des Netzwerkes aufbauen. Dies bezieht sich sowohl auf die Zuordnung einzelner Inhaltsverzeichnisse im Netzwerk als auch auf die Zuordnung der Netzwerkdrucker. Es gelten folgende Übergabeparameter.

CALL: INT 21	
AH:	5FH
AL:	03H
BL:	Einheitentyp
CX:	Anwendercode
DS:SI	Zeiger auf einen Puffer mit den Einheitennamen
ES:DI	Zeiger auf einen Puffer mit Pfad und Password

RETURN	
Carry:	gesetzt -> Fehler
AX:	Fehlercode
Carry:	gelöscht -> ok

Im Register *AL* wird der Subcode *03H* abgelegt. Im Registerpaar *DS:SI* findet sich die Anfangsadresse eines ASCIIZ-Strings (16 Byte) mit dem Namen der umzuleitenden Einheit.

In *ES:DI* wird der Zeiger auf den ASCIIZ-String übergeben, der den Pfad innerhalb des Netzwerkes spezifiziert. Alle Ein-/Ausgaben zu der Quelleinheit werden dann von DOS zur Zieleinheit im Netzwerk umgeleitet. Der Pfad innerhalb des Netzwerkes muß in der Notation:

```
<Maschinenname><Pfadname><00H><Password><00H>
```

übergeben werden.

Das Register *BL* spezifiziert, um welchen Einheitentyp (03H = Printer, 04H = Disk) es sich handelt. Der Quellstring darf eine maximale Länge von 128 Byte nicht überschreiten.

Printer (BL = 03)

In diesem Fall wird ein Druckerausgang auf eine Einheit innerhalb des Netzwerkes umgeleitet. Im Register *CX* kann ein Anwendercode übergeben werden, den DOS verwaltet. Innerhalb eines IBM-Netzwerkes sollte dieser Wert aus Kompatibilitätsgründen auf Null gesetzt werden. Alle anderen Werte sind für das IBM-Netzwerkprogramm reserviert. Der Wert wird beim Aufruf der Funktion *5F02H* zurückgegeben. Alle Druckerausgaben werden gepuffert und zum Netzwerkprinter umgeleitet. Die Funktion bezieht sich dabei auf alle Ausgaben über den *BIOS-INT 17*, die durch die Einheitennamen *PRN*:, *LPT1*:, *LPT2*:, *LPT3*: spezifiziert werden. Die ASCIIZ-Strings, die die Druckereinheiten spezifizieren, sind mit einem Nullbyte abzuschließen. Der ASCIIZ-String mit der Zieleinheit muß ein Paßwort der Länge 1 bis 8 Zeichen enthalten.

Laufwerk (BL = 04)

Mit *BL = 04* wird ein Laufwerk innerhalb des Netzwerkes umdirigiert. Im Quellstring ist der Laufwerksname, gefolgt von einem Doppelpunkt und einem Nullbyte, abzulegen. Der ASCIIZ-String mit der Zieleinheit enthält einen Zugriffspfad zum Netzwerk und endet mit einem Nullbyte. Im Register *CX* läßt sich (wie bei *BL = 03*) ein anwenderspezifischer Code angeben. Wird ein Leerstring als Name des Quellaufwerkes angegeben, bleibt die Umleitung wirkungslos. Bei gültigen Laufwerksnamen werden alle Ein-/Ausgaben zum Netzwerk umgeleitet.

Tritt während des Aufrufes ein Fehler auf, wird das Carry-Flag gesetzt. Im Register *AX* findet sich ein Fehlercode:

```
 Code º Fehler
------é-------------------------------------------------
  1   º Falscher Funktionscode (BL nicht 1 .. 4), oder
      º die Netzwerksoftware ist nicht geladen
  3   º Zugriffspfad nicht vorhanden
  5   º Zugriff abgewiesen
  8   º Nicht genügend Speicher vorhanden
```

Die erweiterten DOS-Fehlercodes lassen sich mittels der Funktion *59H* abfragen.

4.101 Cancel Redirection (Funktion 5FH, Code 04H, DOS 3.1 - 6.x)

Mit dem Aufruf *5F04H* läßt sich eine Zuordnung innerhalb des Netzwerkes rückgängig machen. Dies bezieht sich sowohl auf die Zuordnung einzelner Inhaltsverzeichnisse im Netzwerk, als auch auf die Zuordnung der Netzwerkdrucker. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21               º
º                                  º
º AH:    5FH                       º
º AL:    04H                       º
º DS:SI  Zeiger auf einen Puffer   º
º        mit den Einheitennamen    º
û---------------------------------À
º      RETURN                      º
º Carry: gesetzt -> Fehler         º
º AX:    Fehlercode                º
º Carry: gelöscht -> ok            º
Û---------------------------------ì
```

Im Register *AL* wird mit dem Subcode *04H* belegt, während das Registerpaar *DS:SI* auf einen ASCIIZ-String mit dem Einheitennamen (max. 128 Byte) zeigt. Handelt es sich bei der Einheit um ein Laufwerk, nimmt DOS die Umleitung zurück und setzt sie auf die physikalische Einheit um. Druckerumleitungen werden ebenfalls vom Netzwerk auf den lokalen Druckerausgang zurückgeschaltet.

Der Versuch, eine Umleitung zurückzunehmen, während die Druckerausgabe noch läuft, oder falls der Zugriff auf den Netzwerkknoten abgeschaltet ist, wird abgewiesen.

Im Fehlerfall wird das Carry-Flag gesetzt und im Register *AX* findet sich ein Fehlercode:

```
 Code º Fehler
------é---------------------------------
  1   º Falscher Funktionscode
 15   º Umleitung zur Zeit unterbrochen
```

Die erweiterten DOS-Fehlercodes lassen sich über die Funktion *59H* abfragen.

4.102 Get Redirection List Extended Entry (Funktion 5FH, Code 05H, DOS 4.x - 6.x)

Mit dem Aufruf *5F05H* läßt sich die Liste mit den erweiterten Einträgen abfragen. Diese Funktion ist erst ab MS-DOS 4.X in Zusammenarbeit mit dem Microsoft Netzwerk definiert. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5FH                      º
º AL:    05H                      º
º BX:    Redirection List Entry   º
º DS:SI  Zeiger auf Quell-Puffer  º
º ES:DI  Zeiger auf Ziel-Puffer   º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º BH:    Device Status Flag       º
º BL:    Device Typ               º
º CX:    Parameter Wert           º
º BP:    NetBIOS Session Nummer   º
Û---------------------------------ì
```

Im Register *AL* wird mit dem Subcode *05H* belegt, während das Registerpaar *DS:SI* auf einen ASCIIZ-String mit dem Namen der Quelleinheit zeigt. In ES:DI ist ein Zeiger auf den Puffer einzurichten, der den ASCIIZ-String mit dem Netzwerkpfad aufnimmt.

Nach einem fehlerfreien Aufruf (Carry-Flag gelöscht), findet sich im Register BH das Geräte-Statusflag mit folgender Kodierung:

```
BH Bit 0: 0 das Gerät (Device) ist gültig
```

In BL steht dann der Type des Gerätes:

```
BL : 03H Printer
     04H Disklaufwerk
```

CX enthält den gespeicherten Parameterwert (user data). In BP steht die lokale NetBIOS Sessionsnummer.

Im Fehlerfall wird das Carry-Flag gesetzt und im Register *AX* findet sich ein Fehlercode gemäß der Kodierung der Funktion 59H.

4.103 Enable Drive (Funktion 5FH, Code 07H, DOS 6.x)

Mit dem undokumentierten Aufruf *5F06H* läßt sich in DOS 5.0 ein Laufwerk freigeben. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5FH                      º
º AL:    07H                      º
º DL:    Laufwerk (0=A:, etc.)    º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
Û---------------------------------ì
```

Im Register *AL* wird mit dem Subcode *06H* belegt, während das Register DL den Laufwerkscode aufnimmt.

Tritt beim Aufruf ein Fehler auf, wird bei der Rückkehr das Carry-Flag gesetzt. In AX findet sich dann ein Fehlercode, dessen Bedeutung der Nomenklatur der Funktion 59H entspricht.

4.104 Disable Drive (Funktion 5FH, Code 08H, DOS 6.x)

Mit dem undokumentierten Aufruf *5F06H* läßt sich ab DOS 5.0 ein Laufwerk sperren. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    5FH                      º
º AL:    08H                      º
º DL:    Laufwerk (0=A:, etc.)    º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
Û---------------------------------ì
```

Im Register *AL* wird mit dem Subcode *08H* belegt, während das Register DL den Laufwerkscode aufnimmt. Bei erfolgreichem Aufruf ist dann das Laufwerk für Zugriffe gesperrt.

Tritt beim Aufruf ein Fehler auf, wird bei der Rückkehr das Carry-Flag gesetzt. In AX findet sich dann ein Fehlercode, dessen Bedeutung der Nomenklatur der Funktion 59H entspricht.

Die restlichen Subcodes der Funktion 5F sind von Netzwerktreibern anderer Hersteller (z.B. Starlite) oder Microsofts LAN-Manager belegt. Die Kodierung wird hier jedoch weggelassen, da sie nicht genau bekannt ist.

4.105 Expand Filename (Funktion 60H, DOS 3.0-6.x, undokumentiert)

Der Aufruf *60H* gehört ebenfalls zu den undokumentierten MS-DOS-Funktionen. Er dient dazu, zu einer Datei den vollständigen Pfadnamen (kanonischer Filename) zu ermitteln. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    60H                      º
º DS:SI  ASCIIZ - Filename        º
º ES:DI  Adresse Ergebnispuffer   º
û---------------------------------À
º      RETURN                     º
º CY: 1  Fehler                   º
º AX:    Fehlercode               º
Û---------------------------------ì
```

Im Register *DS:SI* wird die Anfangsadresse des ASCIIZ-Strings mit dem unvollständigen Dateinamen übergeben. Weiterhin ist ein zweiter Puffer mit mindestens 128-Byte-Länge zu

reservieren. Die Funktion ermittelt den vollständigen Pfadnamen, einschließlich Laufwerks und Filename und gibt das Ergebnis als ASCIIZ-String im Puffer zurück. Dabei werden auch Umleitungen mit JOIN, SUBST oder APPEND oder Netzwerktreiber berücksichtigt.

Die Funktion ist immer dann hilfreich, wenn ein Dateiname nur unvollständig vorliegt. Der angegebene Pfad im Filenamen muß nicht unbedingt vorhanden sein. Die Funktion konvertiert alle Zeichen in Großbuchstaben, alle »/«Zeichen werden in Backslash (umgesetzt. Wird ein »*.**Fehler! Verweisquelle konnte nicht gefunden werden.**..**Fehler! Verweisquelle konnte nicht gefunden werden.**D:**Fehler! Verweisquelle konnte nicht gefunden werden.****Fehler! Verweisquelle konnte nicht gefunden werden.**In DOS 3.3 bis 6.0 wird der Name eines Gerätes in einer besonderen Notation zurückgegeben, falls der Name nicht implizit ein Unterverzeichnis oder die Angabe /DEV enthält. Der zurückgegebene String enthält den unveränderten Gerätenamen, dem die Extension »x:/**Fehler! Verweisquelle konnte nicht gefunden werden.**xDer Aufruf zerstört den Inhalt des AL-Registers. Der Ausgabepuffer enthält dann den Filenamen in der Form:

```
D:\PFAD\FILE.EXT
   oder
\\MASCHINE\PFAD\FILE.EXT
```

Tritt ein Fehler beim Aufruf auf, wird das Carry-Flag gesetzt. In AX findet sich dann ein Fehlercode:

Code	Fehler
02H	ungültige Angabe im Verzeichnispfad oder Laufwerksbuchstaben
03H	falsch aufgebaute Pfad- oder Laufwerksangabe

Die Funktion *61H* ist für MS-DOS reserviert aber nicht implementiert.

4.106 Get Program Segment Prefix Address (Funkt. 62H, DOS 3.0-6.x)

Ab DOS 3.0 existiert neben dem internen Funktionsaufruf (*51H*) die offizielle Funktion *62H* zur Ermittlung der Adresse des Program Segment Prefix (*PSP*). In DOS 5.0 wurde dann die Funktion 51H offiziell dokumentiert, während die Funktion 62H ab DOS 5.0 als undokumentiert gilt. Es gelten folgende Aufrufparameter:

```
Ö--------------------------------Ì
°      CALL: INT 21             °
°                      °
° AH:   62H                  °
û--------------------------------À
°    RETURN                  °
° BX:   Segmentadresse der      °
°      aktiven Prozesses       °
Û--------------------------------ì
```

Die Funktion gibt im Register *BX* die Segmentadresse des aktuellen *PSP* zurück. Der Offsetwert ist implizit *0*, während der Codebereich des laufenden Prozesses ab Offset *100H* beginnt.

Die Funktion benutzt ab DOS 3.0 keine internen Stacks und ist damit reentrant. Ab DOS 5.0 sollte allerdings die Funktion 51H benutzt werden.

4.107 DCBS-Support (Funktion 63H, undokumentiert)

Diese Funktion ist nicht dokumentiert und wird von verschiedenen DOS-Versionen benutzt, um Kanji-Zeichen zu bearbeiten.

4.107.1 Get Lead Byte Table (AL = 00H)

Mit diesem Aufruf liest die asisatische Version von DOS 2.25 das »Lead Byte Aufruf-schnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 21               º
º                                  º
º AH:  63H (DBCS)                  º
º AL:  00H (Get DBCS Table)        º
û---------------------------------À
º      RETURN                      º
º AL=00: ok                        º
º DS:SI  Bufferadresse             º
º AL=FF: nicht unterstützt         º
Û---------------------------------ì
```

Die Adresse der Tabelle wird nach dem Aufruf in DS:SI zurückgegeben. Bei DOS 2.25 finden sich hier die »Lead Bytes**Fehler! Verweisquelle konnte nicht gefunden werden.**

Offset	Bytes	Bedeutung
00H	2	low/high ends des »Leading Byte« Bereichs des betreffenden Zeichens
02H	2	low/high ends des »Leading Byte« Bereichs des betreffenden Zeichens
....		
xxH	2	Ende Flag (ist immer 0000H)

Mit den DBCS-Zeichen werden zum Beispiel Kanji-Zeichen ausgegeben. Die Tabelle enthält dabei die Konstruktionsvorschriften für diese komplexen Zeichen. Die Generierung der Zeichen erfolgt durch eigene Routinen, die aktiv werden, sobald ein Startzeichen (interim character) erkannt wird.

Falls die Funktion nicht unterstützt wird, enthält das Register AL nach dem Aufruf den Wert FFH. Bei erfolgreichem Aufruf ist AL = 00 ab DOS 3.2. In DOS 2.25 wird ein Fehlerstatus wie bei den anderen DOS-Funktionen über das Carry-Flag zurückgegeben. Hier findet sich bei gesetztem Carry-Flag ein Fehlercode in AX. Die Kodierung entspricht der Belegung in der Funktion 59H.

4.107.2 Set/Clear Interim Console Flag (AL = 01H)

Diese Funktion ist ebenfalls nur in verschiedenen asiatischen DOS-Versionen implementiert. Sie unterstützt die Eingabe von koreanischen Hongeul Zeichen. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:  63H (DBCS)                 °
° AL:  01H (Set/Clear Flag)       °
° DL:  00  Set Flag               °
°      01  Clear Flag             °
û---------------------------------À
°      RETURN                     °
° AL: 00 ok                       °
° AL: FF ungültiger Mode          °
Û---------------------------------ì
```

Mit diesem Aufruf setzen oder löschen die asiatischen Versionen von DOS 2.25 und ab DOS 3.2 das Interim-Zeichen-Flag. Diese Eigenschaft wird zum Beispiel benötigt, um koreanische Zeichen des Hongeul Alphabeths einzulesen.

Der Wert in DL bestimmt, ob das Flag gesetzt oder gelöscht wird.

```
  DL = 0:  nur komplette Zeichen zurückgeben
       1:  einzelne Bytes eines Zeichens
           zurückgeben
```

Die Belegung ist für alle asiatischen DOS-Versionen gleich. Auch die Fehlercodes werden nicht über das Carry-Flag zurückgegeben, sondern AL wird zur Statusanzeige genutzt.

Der Aufruf zerstört alle Register bis auf SS:SP.

4.107.3 Get Interim Console Flag (AL = 02H)

Auch dieser Aufruf ist nur in asiatischen Versionen des Betriebssystems implementiert.

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
°                                 °
° AH:  63H (DBCS)                 °
° AL:  02H (Get Interim Flag)     °
û---------------------------------À
°      RETURN                     °
° AL: FF Fehler                   °
° AL: 00 ok                       °
° DL: Flag                        °
Û---------------------------------ì
```

Mit diesem Aufruf liest DOS 2.25 und ab DOS 3.2 das Interim-Zeichen-Flag. Nach einem Aufruf findet sich in DL der Zustand des Flags, falls der Inhalt von AL auf 00 gesetzt war. Bei AL = FFH wird die Funktion nicht unterstützt. Für DL gilt folgende Belegung:

```
  DL = 0:  nur komplette Zeichen zurückgeben
       1:  einzelne Bytes eines Zeichens
           zurückgeben
```

Der Aufruf zerstört alle Register bis auf SS:SP.

4.108 Set Device Driver Lookahead Flag (Funktion 64H, DOS 3.2-6.x)

Diese Funktion wird ab DOS 3.2 intern von DOS benutzt. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    64H                      º
º AL:    Flag                     º
û---------------------------------À
º      RETURN                     º
º ----                            º
Û---------------------------------ì
```

Die Funktion definiert ob ein Einheitentreiber vor dem Aufruf der INT 21-Funktionen 01H, 08H und 0AH mit der Treiberfunktion 05 (non destructive read) aufgerufen wird. Die Einstellung erfolgt über den Wert in AL:

```
AL =  0: Aufruf der Treiberfunktion 05 vor den
         INT 21-Funktionsaufrufen 01H, 08H und
         0AH.
    <>0: kein Aufruf der Treiberfunktion 5
```

Die Funktion benutzt keinen der internen DOS-Stacks, ist also reentrant. Die Funktion wird ab DOS 3.3 durch PRINT.COM aufgerufen.

4.109 Get Extended Country Information (Funktion 65H, DOS 3.3-6.x)

In DOS 3.3 lassen sich mit dieser Funktion die erweiterten Landesinformationen abrufen. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:    65H                      º
º AL:    Subcode (1,2,4,5,6,7)    º
º BX:    Code Page Flag           º
º DX:    Landeskennzahl           º
º CX:    Zahl der zu lesenden     º
º        Daten                    º
º ES:DI  Vektor auf einen Puffer  º
º        für die Daten            º
û---------------------------------À
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º ES:DI  Datenpuffer mit den      º
º        Informationen            º
Û---------------------------------ì
```

Die Funktion erlaubt es, verschiedene Parametersets zu lesen. Die Selektion dieser Sets erfolgt über das Register *AL*. Im Register *DX* wird die Landeskennzahl angegeben, auf die sich der Aufruf bezieht. Diese Kennzahlen wurden bereits bei der Funktion 38H besprochen. Der Wert *-1* spezifiziert den gerade eingestellten Landescode.

Das Register *BX* spezifiziert den Code für den gewünschten Zeichenfont (Code Page). Ein Wert = *-1* bezieht die Anfrage auf den globalen Zeichenfont (der *CON*-Einheit).

Die Daten werden in einem Puffer zurückgegeben. Dieser Bereich ist vom rufenden Prozeß einzurichten. Der Wert im Registerpaar *ES:DI* spezifiziert einen Adreßvektor auf den Pufferanfang.

Das Register *CX* enthält einen Wert, der die Zahl der zurückgelesenen Bytes angibt. Es sind mindestens 5 Byte zu lesen.

Landesinformationen (AL = 01H)

Wird der Wert *01* im Register *AL* übergeben, liest die Funktion bestimmte Informationen über landesspezifische Formate. Die nachfolgende Tabelle gibt die Belegung des Ergebnisbuffers (*ES:DI*) wieder.

Bytes	Landesinformation
1	Identifikationscode Funktion = 01H
2	Pufferlänge
2	Landesidentifikationscode
2	Code Page
2	Datumsformat
5	Währungssymbol
2	Tausenderseparator
2	Dezimalzeichen
2	Datumsseparator
2	Zeitseparator
1	Währungsformatflag
1	Nachkommastellen im Währungsformat
1	Zeitformat
4	Einsprungadresse »Mono Case**Fehler! Verweisquelle konnte nicht gefunden werden.**
2	Separator der Datenliste
10	Nullbytes (00H)

Tabelle 4.34: Landesinformationen bei der Funktion 65H (Ende)

Die Bedeutung dieser Datenstruktur (z.B. Währungssymbol) wurde bereits im Rahmen der Funktion *38H* besprochen.

Codierungstabelle Großbuchstaben (AL = 02H)

Auf der Benutzerebene lassen sich Datei- und Einheitennamen sowohl in Klein- als auch in Großbuchstaben eingeben. Zur Umsetzung von Klein- auf Großbuchstaben bietet DOS 3.3 eine Umsetztabelle. Die Adresse dieser Tabelle läßt sich mittels *AL = 02H* ermitteln.

Bytes	Landesinformation
1	Identifikationscode Funktion = 02H
4	Adreßvektor der Tabelle

Tabelle 4.35: Kodierungstabelle Großbuchstaben bei der Funktion 65H

Die 5 Byte werden in den durch das Registerpaar *ES:DI* adressierten Puffer zurückgegeben. Das erste Byte enthält den im Register *AL* übergebenen Subcode. Der nachfolgende Adreßvektor zeigt auf die Tabelle mit den Großbuchstaben. Im ersten Wort der Tabelle steht deren Länge in Byte. Daran schließen sich 128-ASCII-Zeichen an. Insgesamt umfaßt die Tabelle somit 130 Byte.

Kodierungstabelle Filenamen (AL = 04H)

Ähnlich der Umsetztabelle für Großbuchstaben existiert eine zweite Tabelle für die Umsetzung von Dateinamen in Großbuchstaben. Sie läßt sich mittels der Unterfunktion *AL = 04H* abfragen. Im Register *ES:DI* findet sich die Pufferadresse, in dem die Daten abgelegt wurden.

Bytes	Landesinformation
1	Identifikationscode Funktion = 04H
4	Adreßvektor der Tabelle

Tabelle 4.36: Kodierungstabelle Filenamen bei der Funktion 65H

Der Puffer enthält wieder den Identifikationscode sowie den Adreßvektor auf die Umkodierungstabelle. Im ersten Wort der Tabelle steht wieder deren Länge in Byte. Diese ist auf 128 Byte begrenzt, so daß die Datenstruktur ebenfalls 130 Byte belegt.

Kodierungstabelle Filename-Terminator (AL = 05)

Diese Funktion wurde ab DOS 5.0 dokumentiert, ist aber in den früheren Versionen ab 3.3 undokumentiert. Die Funktion gibt die gleichen Informationen für alle Länder und Codepageeinstellungen zurück.

In *ES:DI* findet sich die Pufferadresse, in dem die Daten abgelegt wurden.

Bytes	Landesinformation
1	Wert 05H als Markierung für die ausgeführte Funktion
4	Adreßvektor der Tabelle mit unzulässigen Zeichen

Tabelle 4.37: Kodierungstabelle unerlaubte Zeichen im Filenamen bei der Funktion 6505H

Die Tabelle besitzt folgenden Aufbau:

Bytes	Bedeutung
2	Länge
1	---
1	niedrigster Zeichencode Filename
1	höchster Zeichencode Filename
1	---
1	erstes ungültiges Zeichen
1	letztes ungültiges Zeichen
1	---
1	Zahl der Terminatorzeichen
n	Tabelle mit Zeichen für Ende Filenamen
	. " / [ ] : | < > + = ...

Bei fehlerfreiem Aufruf ist das Carry-Flag gelöscht und der Puffer enthält wieder den Identifikationscode sowie den Adreßvektor auf die Tabelle. Im ersten Wort der Tabelle steht wieder deren Länge in Byte. Daran schließen sich die Bytes mit den unerlaubten Zeichen im Filenamen an (z.B. /, [,], etc.). Bei einem Fehler ist nach dem Aufruf das Carry-Flag gesetzt. AX enthält dann einen Fehlercode:

```
AX = 01: ungültige Funktion
     02: Datei nicht gefunden
```

Um die Funktion anzuwenden muß vorher das Programm NLSFUNC unter Angabe des vollständigen Namens der Datei COUNTRY.SYS geladen werden.

Collate Size (AL = 06H)

Mittels des Subcodes *AL = 06H* läßt sich die Adresse der »Collate SequenzTritt während des Funktionsaufrufes ein Fehler auf, wird das Carry-Flag gesetzt und im Register *AX* findet sich der Fehlercode.

Code	Fehlerart
1	Der Wert von CX ist kleiner als 5
2	Die Werte für den Landescode oder »Code Page**Fehler! Verweisquelle konnte nicht gefunden werden.**

In diesem Fall werden keine gültigen Daten zurückgegeben.

DBCS-Vektor (AL = 07H)

Ab DOS 4.0 ist bei der Funktion 65H eine weitere Unterfunktion hinzugekommen. Mit dem Wert AL = 07H wird die Adresse der DBCS-Tabelle abgerufen. Diese besitzt eine variable Länge und weist folgende Struktur auf:

Bytes	Bedeutung
2	Tabellenlänge in Worten
1	DBCS Vektor 1
1	DBCS Vektor 2
1	...
1	DBCS Vektor n
2	Endemarkierung 0,0

Die Bedeutung dieser Tabelle ist mir allerdings nicht ganz klar.

Country dependent Character Capitalization (AL = 20H, 21H, 22H)

Ab DOS 4.0 sind bei der Funktion 65H weitere undokumentierte Unterfunktionen hinzugekommen, die ab DOS 5.0 dokumentiert wurden. Die Funktionen besitzen folgende Belegung:

Convert Character (Funktion 6520H)

Die Funktion wandelt ein Zeichen unter Zuhilfenahme einer Tabelle in einen Großbuchstaben um. Es gelten folgende Registerbelegungen:

```
CALL:  INT 21
AH:    65H
AL:    20H  (Convert Character)
DL:    Zeichen

RETURN
Carry: gesetzt -> Fehler
AX:    Fehlercode
Carry: gelöscht -> ok
DL:    Zeichen
```

Mit Hilfe der Funktion lassen sich landesspezifische Zeichen wie Umlaute in Großbuchstaben wandeln. Bei fehlerhafter Ausführung wird das Carry-Flag gesetzt. In AX findet sich dann der Fehlercode:

```
AX = 01: Funktion nicht unterstützt.
```

Dann ist der zurückgegebene Wert in DL ungültig.

Convert String (Funktion 6521H)

Die Funktion wandelt einen String unter Zuhilfenahme einer Tabelle in Großbuchstaben um. Es gelten folgende Registerbelegungen:

```
Ö---------------------------------Ì
º                                 º
º      CALL:  INT 21              º
º                                 º
º AH:    65H                      º
º AL:    21H  (Convert Character) º
º DS:DX  Adresse String           º
º CX     Länge String             º
û---------------------------------À
º                                 º
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º DS:DX  Adresse String           º
Û---------------------------------ì
```

Mit Hilfe der Funktion lassen sich landesspezifische Zeichen wie Umlaute in Stringketten in Großbuchstaben wandeln. Die Länge des Strings ist in CX zu übergeben. Bei fehlerhafter Ausführung wird das Carry-Flag gesetzt. In AX findet sich dann der Fehlercode:

```
AX = 01: Funktion nicht unterstützt.
```

Dann wurde der String nicht konvertiert. Andernfalls wurde der String im Puffer konvertiert.

Convert ASCIIZ-String (Funktion 6522H)

Die Funktion wandelt einen ASCIIZ-String unter Zuhilfenahme einer Tabelle in Großbuchstaben um. Es gelten folgende Registerbelegungen:

```
Ö---------------------------------Ì
º                                 º
º      CALL:  INT 21              º
º                                 º
º AH:    65H                      º
º AL:    22H  (Convert Character) º
º DS:DX  Adresse String           º
û---------------------------------À
º                                 º
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º DS:DX  Adresse String           º
Û---------------------------------ì
```

Mit Hilfe der Funktion lassen sich landesspezifische Zeichen wie Umlaute in Stringketten in Großbuchstaben wandeln. Im Gegensatz zur Funktion 21H muß der String mit einem Nullbyte (00H) abgeschlossen sein. Dadurch kann die Angabe der Länge entfallen. Dateinamen werden häufig als ASCIIZ-String gespeichert und lassen sich so leicht in Großbuchstaben konvertieren.

Bei fehlerhafter Ausführung wird das Carry-Flag gesetzt. In AX findet sich dann der Fehlercode:

```
AX = 01: Funktion nicht unterstützt.
```

Dann wurde der String nicht konvertiert. Die Funktion wird nur intern durch DOS genutzt.

YES/NO Character Check (Funktion 6523H)

Die Funktion ist erst ab DOS 4.0 vorhanden und wurde bisher nicht dokumentiert. Sie prüft ob ein eingegebenes Zeichen einer Ja/Nein-Abfrage gültig ist. Diese Zeichen können landesspezifisch verschieden sein (z.B. Y(ES)/N(O), J(A)/NEIN), O(UI)/N(ON), etc.). Es gelten folgende Registerbelegungen:

```
Ö---------------------------------Ì
º                                 º
º      CALL:  INT 21              º
º                                 º
º AH:    65H                      º
º AL:    23H  (Check Character)   º
º DX     Zeichen                  º
û---------------------------------À
º                                 º
º      RETURN                     º
º Carry: gesetzt -> Fehler        º
º AX:    Fehlercode               º
º Carry: gelöscht -> ok           º
º AX:    Typ                      º
Û---------------------------------ì
```

In DX wird das zu untersuchende Zeichen übergeben. Bei 1-Byte-Zeichen (z.B: J/N) befindet sich das Zeichen in DL und DH ist 00H zu setzen. Bei DBCS-Zeichen befindet sich im Register DH das zweite DCBS-Zeichen, sofern vorhanden.

Bei fehlerhafter Ausführung wird das Carry-Flag gesetzt. In AX findet sich dann der Fehlercode:

```
AX = 01: Funktion nicht unterstützt.
```

Dann wurde der String nicht konvertiert. Andernfalls wird im Register AX der Status des Tests zurückgegeben:

```
AX = 00H: Eingabe entspricht YES
     01H: Eingabe entspricht NO
     02H. Eingabe weder YES noch NO
```

Damit kann ein Programm unabhängig von der Landessprache Ja/Nein-Abfragen formulieren.

Convert Country Dependent Filenames (Funktionen AX = 65A0H, 65A1H, 65A2H)

Die Funktionen sind erst ab DOS 4.0 vorhanden und wurden bisher nicht dokumentiert. Sie wandeln Filenamen in Großbuchstaben um. Es gilt die gleiche Aufrufschnittstelle wie bei den Funktionen AX=6520H bis 6522H. Der Aufruf funktioniert aber wegen eines Bugs in den DOS-Versionen 4.0 bis 5.0 nicht.

4.110 Get Global Code Page (Funktion 6601H, DOS 3.3-6.x)

```
Ab DOS 3.3 läßt sich mit dieser Funktion die »Code
PageÖ---------------------------------Ì
º                                     º
º      CALL:  INT 21                  º
º                                     º
º AH:    66H                          º
º AL:    01H  (Get Code Page)         º
û---------------------------------À
º                                     º
º      RETURN                         º
º Carry: gesetzt -> Fehler            º
º AX:    Fehlercode                   º
º Carry: gelöscht -> ok               º
º BX:    Aktive Code Page             º
º DX:    System Code Page             º
º                                     º
Û---------------------------------ì
```

Mit dem Wert *01H* im Register AL wird die gerade aktive »Code Page«Einstellung abgefragt. Im Feh_lerfall ist nach dem Aufruf das Carry-Flag gesetzt. Nur falls das Carry-Flag gelöscht ist, finden sich in den Registern BX und DX Informationen über die Ein_stel_lung.»Das Register *BX* enthält den Wert des aktuell eingestellten »Code Page«Sets, während in DX der »Code PageCode PageFehler! Verweisquelle konnte nicht gefunden werden.**4.111 Set Global Code Page (Funktion 6602H, DOS 3.3-6.x)**

Mit diesem Aufruf läßt sich die aktive Code Page Seite setzen. Es gilt folgende Aufrufschnittstelle.

```
Ö---------------------------------Ì
º      CALL:  INT 21                  º
º                                     º
º AH:    66H                          º
º AL:    02H  (Set Code Page)         º
º BX:    Aktive Code Page             º
û---------------------------------À
º      RETURN                         º
º Carry: gesetzt -> Fehler            º
º AX:    Fehlercode                   º
º Carry: gelöscht -> ok               º
º BX:    Aktive Code Page             º
º DX:    System Code Page             º
Û---------------------------------ì
```

Ab DOS 3.3 läßt sich mit dieser Funktion die »Code Page«Ein_stel_lung des aktuellen Landes verändern. Es gelten nebenstehende Über_gabe_parameter.»Mit dem Wert *02H* im Register *AL* wird der global gültige »Code Page**Fehler! Verweisquelle konnte nicht gefunden werden.**Code Page**Fehler! Verweisquelle konnte nicht gefunden**

werden. Code Page**Fehler! Verweisquelle konnte nicht gefunden werden.**Der Wert in BX bestimmt dabei die gewünschte Code Page Seite. Hierbei gilt folgende Kodierung:

```
  Code  Bedeutung
  437   US
  850   Multilingual
  852   slavische Sprachen
  860   Portugal
  861   Island
  863   Kanada (französischer Teil)
  865   Norwegen / Dänemark
```

Falls eine Einheit nicht umgeschaltet werden kann, wird das Carry-Flag gesetzt und im Register *AX* findet sich ein Fehlercode:

```
 Code ° Fehlerart
------é-----------------------------------------------------
  02  ° »Code Page
```**Fehler! Verweisquelle konnte nicht gefunden werden.**  65  ° Einheit kann nicht auf neuen »Code Page**Fehler! Verweisquelle konnte nicht gefunden werden.**  ° werden

Der »Code Page Fehler! Verweisquelle konnte nicht gefunden werden.**Code Page«Datei durch DOS ladbar sein. An_dern__falls erfolgt eine Fehlermeldung (02H).»Um die »Code Page«Einstellung der aktuellen Landeseinstellung zu verändern, besteht nur der Weg, die Einträge in CONFIG.SYS zu än_dern und das System neu zu starten.»Um diesen Funktionsaufruf zu nutzen, muß das Programm *NLSFUNC* vorher von der Kommandoebene aus gestartet worden sein. Weiterhin müssen die Einheiten in der Lage sein, verschiedene »Code Pages**Fehler! Verweisquelle konnte nicht gefunden werden.

4.112Set Handle Count (Funktion 67H, DOS 3.3-6.x)

Ab DOS 3.3 lassen sich mehr als 20 offene Dateien pro Prozeß verwalten. Mit dieser Funktion kann die Einstellung der maximal gleichzeitig zu öffnenden Dateien verändert werden (max. 65 534). Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
° AH:    67H                      °
° BX:    Zahl der offenen Handles °
û---------------------------------À
°      RETURN                     °
° Carry: gesetzt -> Fehler        °
° AX:    Fehlercode               °
° Carry: gelöscht -> ok           °
Û---------------------------------ì
```

Im Register *BX* wird die Größe der Handletabelle angegeben. Falls dieser Wert kleiner als die Zahl der eröffneten Handles ist, wird der neue Wert erst dann eingestellt, wenn die Zahl der geöffneten Handles unter die spezifizierte Grenze gesunken ist. Ein Wert kleiner als 20 wird nicht angenommen, sondern es werden immer 20 geöffnete Handles vorgesehen.

Im File CONFIG.SYS läßt sich die Zahl der geöffneten Handles nur zwischen 20 und 255 einstellen, obwohl die Funktion *67H* insgesamt bis zu 64 Kbyte für die Handles reserviert. Falls die Handletabelle größer als 20 ist, gibt DOS 3.3 die reservierten Speicherbereiche für die Handletabellen nicht wieder frei. Dies kann bei mehrfachem Aufruf zu einer Fragmentierung des Speichers führen. Weiterhin werden bei DOS 3.3 nur die ersten 20 Handles an einen Tochterprozess vererbt. In DOS 3.3 wurde auf Grund eines Bugs die Handletabelle mit 64 Kbyte reserviert, sofern die Zahl der benötigten Handles gerade war. Da DOS die Handletabelle außerhalb des PSP anlegt (sofern mehr als 20 Handles gefordert werden), muß der rufende Prozeß sicherstellen, daß freier Speicher hierfür vorhanden ist. Die Funktion prüft nicht, ob DOS genügend Einträge in der internen System-File-Table (SFT) besitzt um die reservierten Handles auch zu verwalten. Ist die SFT zu klein, wird beim Öffnen der n.-ten Einheit eine Fehlermeldung ausgelöst. Gegebenenfalls muß die SFT mit dem Eintrag FILES=xx in CONFIG.SYS vergrößert werden.

4.113 Commit File (Funktion 68H, DOS 3.3-6.x)

Mit dieser Funktion läßt sich ab DOS 3.3 erzwingen, daß alle Zwischenpuffer mit Daten auf die jeweiligen Dateien geschrieben werden. Es gelten folgende Übergabeparameter.

```
Ö---------------------------------Ì
°      CALL:  INT 21              °
° AH:    68H                      °
° BX:    Handle                   °
û---------------------------------À
°      RETURN                     °
° Carry: gesetzt -> Fehler        °
° AX:    Fehlercode               °
° Carry: gelöscht -> ok           °
° BX:    File Handle              °
Û---------------------------------ì
```

Mit diesem Aufruf läßt sich eine effizientere Dateiverwaltung, insbesondere bei Netzwerkapplikationen erreichen. Normalerweise kann nur durch eine Close-Anweisung

ein Auslagern der Puffer erzwungen werden. Falls die Datei aber weiterhin benötigt wird, muß anschließend ein Open-Aufruf erfolgen. Dies ist ein sehr zeitintensives Unternehmen.

Mit der Funktion *68H* (Commit File) kann das Problem auf elegante Weise umgangen werden, da beim Aufruf die modifizierten Puffer auf das Medium ausgelagert werden, ohne die Datei zu schließen. Zusätzlich besteht in einer Netzwerkumgebung die Sicherheit, daß die Datei für den aktiven Prozeß mit allen Zugriffsrechten erhalten bleibt, während dies bei der Close/Open-Sequenz nicht immer der Fall ist. Erlangt zum Beispiel ein zweiter Prozeß die Kontrolle über die Datei, nachdem der aktive Prozeß gerade die »Close«Anweisung aus_geführt hat, dann resultiert der anschließende »Open«Aufruf in einer Fehlermeldung. Alle Versuche, die Datei zu öffnen, werden so lange zurückgewiesen, bis der zweite Prozeß die Datei wieder freigibt. Beim Aufruf ist in BX der Handle einer offenen Datei zu übergeben. Der Fehlerstatus wird im Carry-Flag zur»ckgegeben. Bei einem gelöschten Carry-Flag wurden alle Daten des Puffers auf das Medium ausgelagert. Bei gesetztem Carry-Flag findet sich in AX ein Fehlercode, dessen Bedeutung der Funktion 59H entspricht.

4.114 Get/Set Disk Serial Number (Funktion 69H, DOS 4.0 - 6.x)

Diese Funktion ist nicht dokumentiert und wird ab DOS 4.0 benutzt. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 21              º
º                                 º
º AH:  69H (Serial Number)        º
º AL:  00H (Get serial number)    º
º      01H (Set serial number)    º
º BL:  Drive                      º
º DS:DX Adresse Disk Infoblock    º
û---------------------------------À
º      RETURN                     º
º CF:  1 Fehler                   º
º AX:  Fehlercode                 º
º CF:  0 ok                       º
º AX:  Status                     º
Û---------------------------------ì
```

Mit diesem Aufruf schreibt oder liest DOS eine zufällige Disketten-Seriennummer. Damit wird ein Diskettenwechsel zuverlässiger erkannt. Der Code in AL bestimmt, ob die Nummer gelesen (AL = 0) oder geschrieben (AL = 1) werden soll. In BL ist das Laufwerk (0 = default, 1 = A, etc.) zu übergeben. In DS:DX ist die Adresse eines Datenblocks mit folgendem Aufbau zu setzen:

```
Ö-------Ú-------Ú-----------------------------------Ì
° Offs. ° Bytes ° Bedeutung                         °
û-------é-------é-----------------------------------À
°  0H   °   2   ° Info Level (0000)                 °
û-------é-------é-----------------------------------À
°  2H   °   4   ° Serien-Nummer (binär)             °
°  6H   °  11   ° Volume Label                      °
°  11H  °   8   ° FAT-Typ String                    °
Û-------Ù-------Ù-----------------------------------ì
```

Die 11 Byte des Volume Labels enthalten normalerweise den Text des Disketten Labels. Ist dieser nicht bekannt, steht im Feld der Text "NO NAME   ". Im Feld mit dem FAT-Typ steht der Text "FAT12   " oder "FAT16   ". Die Funktion gibt den Fehlerstatus über das Carry-Flag zurück. Bei gesetztem Flag enthält AX den Fehlercode. Es wird kein kritscher Fehler ausgelöst, falls zum Beispiel kein Medium im Laufwerk vorliegt. Bei gelöschtem Carry-Flag ist der Inhalt von AX zerstört. In Abhängigkeit vom Wert in AL beim Aufruf wird der Inhalt des Puffers (DS:DX) entweder vom Medium gelesen oder zum Medium geschrieben.

Die Funktionen 6AH und 6BH werden ab DOS 4.0 zwar benutzt, ihre Belegung ist aber nicht bekannt.

4.115 Extended Open/Create (Funktion 6CH, DOS 4.0 - 6.x)

Diese Funktion steht erst ab DOS 4.0 zur Verfügung und bildet eine Kombination der Funktionen 3CH (Create File with Handle), 3DH (Open File with Handle), 5BH (Create New File) und 68H (Commit File). Sie sollte bei neuen Porgrammentwicklungen für DOS 4.x und höher verwendet werden, da sie prüft, ob eine Datei bereits existiert. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
°       CALL:  INT 21           °
°                               °
° AH: 6CH (Extended Open/Create)°
° AL: 00H  reserviert           °
° BX:      Open Mode            °
° CX:      Attribut             °
° DX:      Format               °
° DS:SI    Filename (ASCIIZ)    °
û-------------------------------À
°       RETURN                  °
° CY: 1  Fehler                 °
° AX:    Fehlercode             °
° CY: 0  ok                     °
° AX:    Handle                 °
° CX:    Statuscode             °
Û-------------------------------ì
```

Das Register AL ist beim Aufruf immer auf 0 zusetzen. In BX wird ein 16-Bit-Flag mit dem Open-Mode übergeben. Dieses Flag besitzt folgende Codierung:

```
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
°0°W°F°0°0°0°0°0°I°0°S°S°S°0°A°A°
Û-ÙÚÙÚÙ-Ù-Ù-Ù-Ù-ÙÚÙ-ÙÚÙ-ÙÚÙ-ÙÚÙ-ÙÚì
  ° °        °  Û-Ú-ì  Û-Ú-ì Access Code
  ° °        °   °      Û-  0 - Read
  ° °        °   °          1 - Write
```

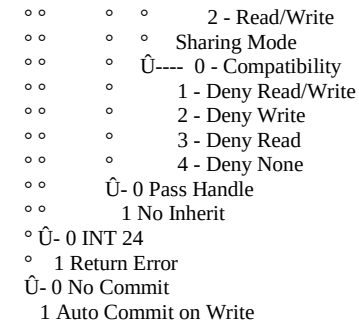


Bild 4.8: Kodierung des Open-Mode

Mit dem W-Bit läßt sich festlegen, ob nach jedem Schreibzugriff ein COMMIT (Update des Directory-Eintrages) erfolgt. Das F-Bit selektiert die Art der Fehlerbehandlung. Ist es gesetzt, gibt die Funktion die Fehlercodes im Register AX zurück. Der INT 24-Critical-Error-Handler ist damit gesperrt. Mit einem F-Bit = 0 fängt der INT 24 kritische Fehler ab.

Im Register CX wird das »Create AttributIn DX läßt sich ein Flag übergeben, welches das Verhalten von DOS spezifiziert, falls die Datei existiert/nicht existiert. Das Flag besitzt folgendes Format:

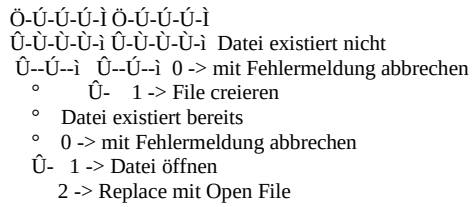


Bild 4.9: Kodierung des Open-Flags

Die vier unteren Bits spezifizieren die Aktionen von DOS, falls die Datei nicht existiert. Ist der Wert = 0, bricht die Funktion mit einer Fehlermeldung ab. Beim Wert = 1 wird die Datei kreiert. Das gleiche Prinzip gilt für die folgenden vier Bit, die für vorhandene Dateien ausgewertet werden.

Nach dem Aufruf zeigt das Carry-Flag den Fehlerstatus an. Ein gesetztes Flag deutet auf Fehler hin, deren Code in AX übergeben wird. Bei CY = 0 enthält AX den Handle des geöffneten Files. Im Register CX gibt die Funktion den Statuscode der Datei zurück. Es gilt folgende Codierung:

CX = 0 File open
1 File create / open
2 File replace / open

Durch Abfrage des Registerinhaltes läßt sich also feststellen, ob eine Datei neu angelegt oder geöffnet wurde.

Die restlichen Codes des INT 21 sind in DOS unbenutzt. Es gibt aber eine Reihe von Programmen, die Subfunktionen des INT 21 abfangen und mit eigenen Funktionen belegen. So benutzt Novells NetWare die Subfunktionen AX=BxxxH bis FFxxH.

5 Der DOS-Multiplexerinterrupt INT 2F

Dies ist ein intern durch MS-DOS benutzter Interrupt, der in MS-DOS 2.X überhaupt nicht und in den anderen DOS-Versionen nur teilweise offiziell dokumentiert ist. Der Interrupt erlaubt die Kommunikation zwischen zwei laufenden Prozessen. In DOS 2.X muß ein Prozeß vor einer Benutzung prüfen, ob der Vektor bereits initialisiert wurde. Ab DOS 3.0 übernimmt das Betriebssystem die Initialisierung.

Der INT 2F besitzt, ähnlich dem INT 21, einen Funktionsdispatcher, der über den Inhalt des Registers AH die jeweilige Unterfunktion aufruft. Jeder dieser Unterfunktionen wird eine eigene Codenummer zugeordnet. Weitere Parameter lassen sich mittels der restlichen Register übergeben. Leider existieren keine festgelegten Mechanismen zur Vergabe dieser Codenummern, so daß der Softwareentwickler darauf zu achten hat, daß eine Nummer nicht durch zwei Applikationen belegt wird. In diesem Fall muß eine der Applikationen mit einer anderen Codenummer versehen werden. Zur groben Orientierung sei folgende Tabelle angegeben:

Code	Bemerkung
00 - 7F	reserviert für MS-DOS-Zwecke
80 - BF	reserviert für zukünftige Erweiterungen
C0 - FF	frei für Anwenderprogramme

Tabelle 5.1: Zuordnung der Codenummern des INT 2F

Ab DOS 4.0 sind die Codes 00H bis BFH für das Betriebssystem reserviert.

Der dokumentierte Teil des Multiplexerinterrupt 2FH wird zur Zeit im wesentlichen zur Kommunikation mit DOS-Utilities wie PRINT, SHARE, etc. genutzt.

Falls ein Anwenderprogramm sich in den INT 2FH einhängen möchte, muß es bestimmte Konventionen beachten.

- Als erstes ist zu prüfen, ob der INT 2F-Vektor durch das Betriebssystem unterstützt wird. Dies ist bei allen DOS-Versionen oberhalb 2.X der Fall.
- Dann ist der Vektor des INT 2F mittels den INT 21-Funktion 35H zu lesen und intern zu speichern.
- Nun kann die Adresse des eigenen INT 2F-Handlers in der Vektortabelle eingetragen werden. Hierzu ist die INT 21-Funktion 25H zu benutzen.
- Wird ein Programm durch den INT 2F aktiviert, muß es den Wert in AH überprüfen. Besitzt das Programm den Code, muß es anschließend den Subcode in AL testen. Stimmt dieser Code mit den vereinbarten Werten überein, ist die Funktion auszuführen. Der Fehlerstatus ist über das Carry-Flag zurückzugeben.

- Stimmt der Identifikationscode in AH überein und ist AL = 0, dann muß der Handler entweder den Wert 01H oder FFH in AL an die rufende Routine zurückgeben (siehe unten). Das Programm sollte mit einem IRET-Befehl (oder RETF 2) terminieren.
- Falls der Wert in AH nicht mit der Identifikationsnummer des Programmes übereinstimmt, ist der folgende Treiber aufzurufen. Dessen Adresse muß beim Eintrag des eigenen Interruptvektors gesichert werden. DOS hat als letzten Treiber einen eigenen Handler in die Kette eingetragen. Wird dieser Handler erreicht, gibt er den Wert AL = 00H an den rufenden Prozess zurück.

Get Installationsstatus (AL = 0)

Um zu überprüfen ob ein Prozess unter dem INT 2F installiert ist, ist der betreffende Identifikationscode in AH zu setzen. Weiterhin muß in AL der Wert 00H eingetragen werden. Ein Aufruf mit diesen Parametern gilt vereinbarungsgemäß als Überprüfung des Installationsstatus.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: xxH                         º
º AL: 00H Get Installationsstatus º
û---------------------------------À
º      RETURN                     º
º AL: Statusbyte                  º
Û---------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Treibers.

- Falls AL = 0 ist, dann ist der residente Teil des Handlers nicht vorhanden und kann installiert werden.
- Falls AL = 1 ist, dann ist der residente Teil der Handlers nicht vorhanden, er läßt sich aber auch nicht installieren.
- Falls AL = FFH ist, dann ist der residente Teil des Handlers bereits installiert.

Die Funktion AL = 00H (Get Installationsstatus) sollte bei jedem residenten Programm implementiert werden, welches den Interrupt 2FH nutzt. Denn nur so läßt sich prüfen, ob der Handler bereits installiert ist.

Nachfolgend werden die einzelnen dokumentierten und undokumentierten Funktionen des INT 2F vorgestellt.

5.1 PRINT-Time-Slice (AX = 0080H, DOS 3.1 - 6.x)

Dieser undokumentierte Aufruf wird durch PRINT abgefangen. DOS aktiviert die Subfunktion um PRINT Rechenzeit zuzuteilen. Es gilt folgende Aufrufkonvention:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 00H  (PRINT)                º
º AL: 80H                         º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

PRINT gibt nach dem Aufruf die Kontrolle an den rufenden Prozess zurück.

5.2 Kommunikation with PRINT (AH = 01H, DOS 3.0 - 6.x)

Mit dem Code AH = 01 erlaubt der INT 2F die Kommunikation mit dem residenten Teil des Programmes PRINT.COM. Der Aufruf ist zwar bereits ab DOS 2.x vorhanden, die Schnittstellenbelegung ist aber erst ab DOS 3.0 definiert. Es gelten folgende Parameterübergaben:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 01H  (PRINT)                º
º AL: Unterfunktion               º
û---------------------------------À
º      RETURN                     º
º --- s. Funktion                 º
Û---------------------------------ì
```

Der Wert in AL selektiert die Unterfunktion des INT 2F-Handlers.

Für das Programm PRINT gelten folgende Konventionen:

```
Ö-----Ú-----------------------------Ì
º AL  º Bemerkungen                  º
û-----é-----------------------------À
º 00  º Get Installationsstatus      º
û-----é-----------------------------À
º 01  º Datei in Liste  einfügen     º
û-----é-----------------------------À
º 02  º Datei aus Liste entfernen    º
û-----é-----------------------------À
º 03  º Liste komplett löschen       º
û-----é-----------------------------À
º 04  º Status lesen                 º
û-----é-----------------------------À
º 05  º Ende der Statusabfrage       º
û-----é-----------------------------À
º 06  º Check Error (ab DOS 3.3)     º
Û-----Ù-----------------------------ì
```

Tabelle 5.2: Unterfunktionen der PRINT-Utility;

Die Funktionscodes AL = F8H bis FFH sind beim PRINT-Aufruf (AH=01H) für DOS reserviert und werden durch den INT 2F abgefangen.

Tritt ein Fehler auf, wird dies durch ein gesetztes Carry-Flag dem Anwenderprozeß signalisiert. In diesem Fall gibt der jeweils über das Register AL angesprochene Funktionsteil einen Fehlercode in AL zurück. Die nachfolgende Tabelle enthält die Belegung der Fehlercodes (dezimal) für die PRINT-Utility.

AL	Bemerkungen
00	kein Fehler
01	fehlerhafter Funktionsaufr.
02	Datei nicht gefunden
03	Pfad nicht gefunden
04	zu viele offene Dateien
05	Zugriff abgewiesen
08	Liste voll
09	Treiber Busy
12	Dateiname zu lang
15	falsches Laufwerk

Tabelle 5.3: Fehlercodes der PRINT-Utility

5.2.1 Get Installationsstatus (AL = 00H, DOS 3.0 - 6.x)

Mit dieser Unterfunktion kann überprüft werden, ob der residente Teil von PRINT installiert ist. Es gelten folgende Aufrufkonventionen:

CALL: INT 2F
AH: 01H (PRINT)
AL: 00H Get Installationsstatus
RETURN
AL: Statusbyte

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Treibers.

- Falls AL = 0 ist, dann ist der residente Teil von PRINT nicht vorhanden und kann installiert werden.
- Falls AL = 1 ist, dann ist der residente Teil von PRINT nicht vorhanden, er läßt sich aber auch nicht installieren.
- Falls AL = FFH ist, dann ist der residente Teil von PRINT bereits installiert.

Die Funktion AL = 00H (Get Installationsstatus) sollte bei jedem residenten Programm implementiert werden, welches den Interrupt 2FH nutzt. Denn nur so läßt sich prüfen, ob der Handler bereits installiert ist.

5.2.2 Datei in Liste einfügen (AL = 01H, DOS 3.0 - 6.x)

Mit diesem Aufruf läßt sich eine Datei in die PRINT-Ausgabeliste einfügen. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 01H (PRINT)                 º
º AL: 01H Datei in Liste einfügen º
º DS:DX Zeiger auf Submit-Tabelle º
û---------------------------------À
º      RETURN                     º
º CY: 0 -> kein Fehler            º
º AL: Submitcode                  º
º CY: 1 -> Fehler beim Aufruf     º
º AL: Fehlercode                  º
Û---------------------------------ì
```

Der Dateiname muß als ASCII-String mit Laufwerksbezeichnung und Pfadname angegeben werden. Der Zeiger im Registerpaar DS:DX zeigt auf eine Datenstruktur (Submit Paket) mit folgendem Aufbau:

```
Ö----------Ú-------------------------------------------Ì
º  Bytes   º  Feld                                      º
û----------é-------------------------------------------À
º    1     º Levelcode                                  º
û----------é-------------------------------------------À
º    4     º Zeiger auf den ASCIIZ-String mit dem       º
º          º Dateinamen                                 º
Û----------Ù-------------------------------------------ì
```

Tabelle 5.4: Aufbau der Submit-Tabelle für PRINT

Der Levelcode ist in den Versionen 3.0 bis 6.0 immer mit dem Wert 0 versehen. Vermutlich ist mit dem Levelcode eine Priorisierung der PRINT-Aufträge für spätere DOS-Versionen vorgesehen. Der Zeiger wird in der Segment:Offset-Notation verwendet. Der ASCIIZ-String muß das Laufwerk, den Pfad und den Dateinamen enthalten. Innerhalb des Dateinamens dürfen keine Wildcards (*) auftreten. Ist das Carry-Flag nach dem Aufruf gesetzt, deutet dies auf einen Fehler hin. Der Fehlercode findet sich dann im Register AL und ist gemäß Tabelle 5.3 kodiert. Bei gelöschtem Carry-Flag enthält AL einen Status, der folgende Kodierung aufweist:

```
AL = 01H Filename in Warteschlange eingetragen
     9EH Datei wird gedruckt
```

Damit kann festgestellt werden, ob die Warteschlange beim Aufruf leer war.

5.2.3 Datei aus Liste entfernen (AL = 02H, DOS 3.0 - 6.x)

Mit diesem Aufruf läßt sich eine Datei aus der PRINT-Ausgabeliste entfernen. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 01H (PRINT)                 º
º AL: 02H Datei aus Liste löschen º
º DS:DX Zeiger auf Dateiname      º
û---------------------------------À
º      RETURN                     º
º CY: 0 -> kein Fehler            º
º CY: 1 -> Fehler beim Aufruf     º
º AL: Fehlercode                  º
Û---------------------------------ì
```

Dies ist die komplementäre Funktion zu AL = 01H, mit der man eine Datei aus der PRINT-Ausgabeliste entfernen kann. Der Zeiger im Registerpaar DS:DX zeigt allerdings direkt auf

den ASCIIZ-String mit dem Dateinamen. Dieser Dateiname darf auch Wildcardzeichen (*) enthalten. Ein gefundener Name wird aus der Liste ausgetragen. Ist das Carry-Flag nach dem Aufruf gesetzt, liegt ein Fehler vor. Die Funktion gibt dann den Status im Register AL zurück. Es gelten die in Tabelle 5.3 beschriebenen Fehlercodes.

5.2.4 Lösche Liste (AL = 03H, DOS 3.0 - 6.x)

Dieser Aufruf löscht alle Einträge aus der PRINT-Ausgabeliste. Es gelten folgende Übergabeparameter:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 01H (PRINT)                º
º AL: 03H Liste löschen          º
û--------------------------------À
º      RETURN                    º
º CY: 0 -> kein Fehler           º
º CY: 1 -> Fehler beim Aufruf    º
º AL: Fehlercode                 º
Û--------------------------------ì
```

Der Aufruf benötigt keinen Dateinamen, da hier alle Einträge aus der Liste entfernt werden. Bei einem Fehler ist anschließend das Carry-Flag gesetzt und AL enthält einen Fehlercode gemäß Tabelle 5.3.

5.2.5 Lese Status (AL = 04H, DOS 3.0 - 6.x)

Mit diesem Aufruf wird die Ausgabe innerhalb des Modules PRINT unterbrochen, so daß die Ausgabeliste untersucht werden kann. Es gelten folgende Übergabeparameter:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 01H (PRINT)                º
º AL: 04H lese Status            º
û--------------------------------À
º      RETURN                    º
º CY: 1 -> kein Fehler           º
º AX: Fehlercode                 º
º CY: 0 -> kein Fehler           º
º DX: Fehlerzähler               º
º DS:SI Zeiger auf Ausgabeliste  º
Û--------------------------------ì
```

Im Register DX wird ein Wert zurückgegeben, der die Zahl der Fehlversuche bei der Ausgabe des letzten Zeichens spezifiziert. Dies tritt zum Beispiel auf, falls der Drucker ausgeschaltet ist. Treten keine Fehler auf, wird das Register DX vor der Rückkehr auf den Wert 0 gesetzt.

Im Registerpaar DS:SI befindet sich ein Zeiger auf die interne Ausgabeliste, welche für jeden Namen einen Puffer von 64 Byte enthält. Der Name der gerade auszugebenden Datei steht am Anfang dieser Liste. Ein Eintrag mit dem Wert 00 im ersten Byte zeigt das Ende der belegten Liste an.

Der nächste Aufruf der Funktion AH = 01H mit dem Subcode AL = 05H gibt die PRINT-Ausgabe wieder frei.

5.2.6 Ende Statusabfrage (AL = 05H, DOS 3.0 - 6.x)

Der Aufruf gibt explizit die PRINT-Ausgabe frei, falls sie durch AH = 04H blockiert war. Es gelten folgende Parameter:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 01H (PRINT)                 º
º AL: 05H Ende Statusabfrage      º
û---------------------------------À
º      RETURN                     º
º CY: 0 -> kein Fehler            º
º CY: 1 -> Fehler beim Aufruf     º
º AL: Fehlercode                  º
Û---------------------------------ì
```

Im Register AL findet sich der oben beschriebene Fehlercode, falls das Carry-Flag gesetzt ist.

5.2.7 Get Printer Device (AL = 06H, DOS 3.3 - 6.x)

Der Aufruf AH = 06H ist erst ab DOS 5.0 dokumentiert und wird durch PRINT.COM benutzt, um zu prüfen, ob die Warteschlange der Ausgabeeinheit voll ist.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 01H (PRINT)                 º
º AL: 06H Get Device              º
û---------------------------------À
º      RETURN                     º
º CY: 0 -> kein Fehler            º
º CY: 1 -> Fehler beim Aufruf     º
º AL: Fehlercode 08H              º
º DS:SI Adresse Device Treiber    º
Û---------------------------------ì
```

Falls dies zutrifft, ist das Carry-Flag gesetzt und in AX steht der Fehlercode 08H (queue full).

Weiterhin gibt die Funktion nach dem Aufruf in DS:SI die Adresse des Treiberkopfes der Ausgabeeinheit zurück. So läßt sich prüfen, auf welches Gerät PRINT seine Ausgaben umleitet.

Bei der INT 2F-Funktion AH = 01H sind alle Unterfunktionscodes von AL = F8H bis FFH für zukünftige DOS-Erweiterungen reserviert.

5.3 Kommunikation with PC LAN Programm REDIR/REDIRIFS (AH = 02H)

Dieser Aufruf wird durch das PC-Lan-Programm REDIR/REDIRIFS für interne Zwecke benutzt und belegt den Code AH = 02H. Die Schnittstelle ist nicht dokumentiert.

```
Ö----Ú-----------------------------------------Ì
° AL ° Aufruf                                  °
û----é-----------------------------------------À
° 00 ° Installations Check                     °
°    ° Return: AL = FFH -> installiert         °
Û----Ù-----------------------------------------ì
```

Nach dem Aufruf zeigt der Wert AL=FFH, daß die Treiber installiert sind.

Das Programm benutzt weiterhin die Untercodes AL = 01, 02, 03 und 04. Die Funktionen und Registerbelegungen sind allerdings nicht bekannt.

5.4 DOS 3.X Critical-Error-Handler (AH = 05H, DOS 3.0 - 6.x)

Der Code AH = 05H ist ab DOS 3.0 durch das Betriebssystem belegt. Über diesen Interrupt wird die Kommunikation mit dem Critical-Error-Handler abgewickelt. Die Schnittstelle wurde durch Microsoft nicht offiziell dokumentiert.

5.4.1 Get Installationsstatus (AL = 0)

Mit dieser Unterfunktion kann überprüft werden, ob der DOS 3.X Critical-Error-Handler installiert ist. Unter diesem Interrupt kann ein Anwenderprozeß unter Umständen eine eigene Fehlerbehandlungsroutine ablegen. Es gelten folgende Aufrufkonventionen:

```
Ö---------------------------------Ì
°      CALL:  INT 2F               °
°                                  °
° AH: 05H (Critical-Error-Handler)°
° AL: 00H Get Installationsstatus °
û---------------------------------À
°      RETURN                      °
° AL: Statusbyte                   °
Û---------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Handlers.

- Falls AL = 0 ist, dann ist der Handler nicht vorhanden und kann installiert werden.
- Falls AL = 1 ist, dann ist der Handler nicht installiert, er läßt sich aber auch nicht installieren.
- Falls AL = FFH ist, dann ist der Handler installiert.

Die Funktion AL = 00H (Get Installationsstatus) sollte vor Aufruf der folgenden Unterfunktion oder vor der Installation eines eigenen Fehlerhandlers benutzt werden, um zu prüfen, ob bereits ein Handler installiert ist.

5.4.2 Handle Error (AL = XXH)

Dieser Aufruf wird beim Auftreten eines Fehlers durch COMMAND.COM aktiviert. Ein Anwenderprogramm kann dann unter dem INT 2F, AH = 05H einen eigenen Handler installieren. Von COMMAND werden in DOS 3.x folgende Parameter übergeben.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 05H (Critical-Error-Handler)º
º AL: Extended Error Code <> 0    º
û---------------------------------À
º      RETURN                     º
º CY: 0 -> kein Fehler            º
º ES:DI Adresse Fehlertext        º
º CY: 1 -> benutze intern. Handlerº
Û---------------------------------ì
```

Im Register AL übergibt DOS 3.x den Extended-Error-Code mit der Fehlerursache. Bei einem gelöschten Carry-Flag muß der Fehler-Handler in ES:DI die Adresse auf einen ASCIIZ-String mit dem Fehlertext zurückgeben. Diese wird dann durch DOS auf der Konsole ausgegeben. Setzt die anwenderspezifische Critical-Error-Routine das Carry-Flag vor dem Rücksprung zu DOS, dann benutzt COMMAND den internen Critical-Error-Handler.

Ab DOS 4.0 gelten etwas andere Aufrufkonventionen:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 05H (Critical-Error-Handler)º
º AL: 1 -> Extend. Error Code<>0  º
º     2 -> Parameter Error        º
º BX:->  Error Code <> 0          º
û---------------------------------À
º      RETURN                     º
º CY: 0 -> kein Fehler            º
º ES:DI Adresse Fehlertext        º
º CY: 1 -> benutze intern. Handlerº
Û---------------------------------ì
```

Im Register AL übergibt DOS einen Statuscode. Ist dieser = 1, dann enthält BX den erweiterten Fehlercode. Mit AL = 2 findet sich in BX der *Parameter Errorcode*. Nach dem Aufruf ist vom Fehler-Handler in ES:DI die Adresse des Fehlerstrings zurückzugeben.

Dieser wird dann durch DOS auf der Konsole ausgegeben. Setzt die anwenderspezifische Critical-Error-Routine das Carry-Flag vor dem Rücksprung zu DOS, dann benutzt COMMAND den internen Critical-Error-Handler. Die Subfunktion 02H wird durch viele externe DOS-Programme aufgerufen.

5.5 Kommunikation with ASSIGN (AH = 06H, DOS 3.0 -6.x)

Ähnlich der Kommunikation mit PRINT läßt sich über die Funktion AH = 06H mit dem residenten Teil von ASSIGN kommunizieren. Dieser Aufruf wurde von Microsoft dokumentiert.

5.5.1 Get Installation Status (AL = 00H)

Zur Überprüfung des Installationsstatus gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 06H (ASSIGN)                º
º AL: 00H Get Installationsstatus º
û---------------------------------À
º      RETURN                     º
º AL: Statusbyte                  º
Û---------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Treibers.

- Falls AL = 0 ist, dann ist der residente Teil von ASSIGN nicht vorhanden und kann installiert werden.
- Falls AL = 1 ist, dann ist der residente Teil von ASSIGN nicht vorhanden, er läßt sich aber auch nicht installieren.
- Falls AL = FFH ist, dann ist der residente Teil von ASSIGN installiert.

ASSIGN erlaubt die Umleitung einzelner Laufwerksbezeichnungen auf andere physikalische Laufwerkseinheiten.

5.5.2 Get ASSIGN Memory Segment (AL = 01H)

Mit dieser Unterfunktion läßt sich das Segment mit dem ASSIGN-Arbeitsbereich ermitteln. Es gelten folgende Aufrufkonventionen:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 06H (ASSIGN)                º
º AL: 01H Get Memory Segment      º
û---------------------------------À
º      RETURN                     º
º ES: Memory Segment              º
Û---------------------------------ì
```

Der zurückgegebene Wert im Register ES enthält die Adresse des ASSIGN-Arbeitsbereiches mit den Zuweisungen. Ab DOS 3.3 findet sich ab Offset ES:103 eine 26 Byte lange Tabelle mit den Codes der Laufwerke. Die Tabelle wird mit den Werten 0 (=A:), 1 (=B:), etc. initialisiert. Bei einer Umleitung (z.B. durch ASSIGN A=B) wird an der betreffenden Stelle einfach der neue Laufwerkscode eingetragen.

Beachten Sie, daß ab DOS 5.0 das Programm ASSIGN nicht mehr als TSR-Programm installiert wird. Vielmehr wird beim Aufruf von ASSIGN intern der SUBST-Befehl eingesetzt. Ab DOS 6.0 entfällt ASSIGN komplett.

5.6 Kommunikation with DRIVER.SYS (AH = 08H, DOS 3.0 - 6.x)

Diese Funktion ist für DOS reserviert und übernimmt die Kommunikation mit dem Programm DRIVER.SYS.

5.6.1 Get Installation Status (AL = 00H)

Zur Überprüfung des Installationsstatus gelten folgende Übergabeparameter:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 08H (DRIVER.SYS)           º
º AL: 00H Get Installationsstatus º
û--------------------------------À
º      RETURN                    º
º AL: Statusbyte                 º
Û--------------------------------ì
```

Der zurückgegebene Wert im Register AL enthält Hinweise über den Status.

- Falls AL = 0 ist, dann ist der DRIVER.SYS nicht vorhanden und kann installiert werden.
- Falls AL = 1 ist, dann ist DRIVER.SYS nicht vorhanden, läßt sich aber auch nicht installieren.
- Falls AL = FFH ist, dann ist DRIVER.SYS installiert.

Der Treiber ermöglicht die Installation neuer Block-Devices.

5.6.2 Add New Block Device (AL = 01H)

Mit dieser Unterfunktion läßt sich ein blockorientierter Device-Treiber installieren.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 08H (DRIVER.SYS)           º
º AL: 01H Add new Block Device   º
º DS:DI Device Driver Header     º
û--------------------------------À
º      RETURN                    º
Û--------------------------------ì
```

Im Registerpaar DS:DI ist die Adresse der Drive-Data-Table zu übergeben. Die Funktion durchsucht die installierte Treiberliste, modifiziert das Wort ab Offset 29H und hängt die neuen Daten an die Treiberkette an. Die Struktur der Drive-Data-Table wird bei der Subfunktion 03H beschrieben.

5.6.3 Execute Device Driver Request (AL = 02H)

Mit dieser Unterfunktion läßt sich der Header eines blockorientierten Einheitentreibers modifizieren.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 08H (DRIVER.SYS)           º
º AL: 02H Execute Device Request º
º ES:BX Device Driver Header     º
û--------------------------------À
º      RETURN                    º
Û--------------------------------ì
```

Im Registerpaar ES:BX ist die Adresse des *Device Treiber Request Headers* zu übergeben. Dieser Header besitzt folgendes Format:

```
Ö-------------------------------------------------------Ì
º      Format des Device Driver Request Header:         º
º                                                       º
º Offs. Byte  Beschreibung                              º
º                                                       º
º 00H     1  Länge des Request Headers                  º
º 01H     1  Subunit im Device Driver                   º
º 02H     1  Kommando Code                              º
º          00H INIT                                     º
º          01H MEDIA CHECK (Block Devices)              º
º          02H BUILD BPB (Block Devices)                º
º          03H IOCTL INPUT                              º
º          04H INPUT                                    º
º          05H NONDESTRUCTIVE INPUT, NO WAIT (char dev) º
º          06H INPUT STATUS                             º
º          07H INPUT FLUSH   (char dev)                 º
º          08H OUTPUT                                   º
º          09H OUTPUT WITH VERIFY                       º
º          0AH OUTPUT STATUS  (char dev)                º
º          0BH OUTPUT FLUSH   (char dev)                º
º          0CH IOCTL OUTPUT                             º
º          0DH (DOS 3.x) DEVICE OPEN                    º
º          0EH (DOS 3.x) DEVICE CLOSE                   º
º          0FH (DOS 3.x) REMOVABLE MEDIA (block devices)º
º          10H (DOS 3.x) OUTPUT UNTIL BUSY  (char dev)  º
º          11H - 12H reserviert                         º
º          13H (ab DOS 3.2) GENERIC IOCTL               º
º          14H - 16H reserviert                         º
º          17H (DOS 3.2+) GET LOGICAL DEVICE            º
º          18H (DOS 3.2+) SET LOGICAL DEVICE            º
º          19H (DOS 5.0+) CHECK GENERIC IOCTL SUPPORT   º
º          80H - 88H CD-ROM-Support                     º
º 03H     2  Status Wort                                º
º          Bit 15   : Error                             º
º              10-14: reserviert                        º
º                  9: Busy                              º
º                  8: Done                              º
º                0- 7: Fehlercode falls Bit 15=1        º
º 05H     8  reserviert                                 º
º --- Kommandocode 00H ---                             º
º 0DH     1  Zahl der Einheiten (vom Treiber gesetzt)   º
º 0EH     4  Adresse 1. freies Byte hinter dem Trei-    º
º            ber (vom Treiber gesetzt)                  º
º 12H     4  Zeiger auf BPB Array (wird nur bei         º
º             Blocktreibern gesetzt)                    º
º 16H     4  Laufwerksnummer der 1. Unit bei Block-     º
º            treibern (0=A:, ab DOS 3.0)                º
º --- Kommandocode 01H ---                             º
º 0DH     1  Media Descriptor Byte                      º
º 0EH     1  zurückgegebener Status                     º
º               00H Medium unbekannt                    º
º               01H Medium nicht gewechselt             º
º               FFH Medium gewechselt                   º
º 0FH     4  Zeiger auf vorhergehendes Volume ID falls  º
º            das OPEN/CLOSE/RM-Bit im Treiberkopf ge-   º
º          setzt ist und das Medium gewechselt wurde    º
º               (wird vom Treiber gesetzt)              º
º --- Kommandocode 02H ---                             º
º 0DH     1  Media Descriptor Byte                      º
º 0EH     4  Transferadresse (scratch Sektor im         º
º            NON-IBM-Format                             º
º            FAT-Sektor sonst)                          º
º 12H     4  Zeiger auf BPB (wird vom Treiber gesetzt)  º
º --- Kommandocode 03H, 0CH ---                        º
º 0DH     1  Media Descriptor Byte                      º
º 0EH     4  Transferadresse                            º
º 12H     2  Bytezähler (character device)             º
º            Sektorzähler (block device)                º
º 14H     2  Nummer Startsektor (block device)         º
º 16H     4  Zeiger auf Volume Label, falls der Error-  º
º            code 0FH vom Treiber gesetzt (DOS 3.0+)    º
º --- Kommandocode 04H, 08H, 09H ---                   º
º 0DH     1  Media Descriptor Byte                      º
º            (nur block devices)                        º
º 0EH     4  Transferadresse                            º
º 12H     2  Bytezähler (character device)             º
º             Sektorzähler (block device)               º
º 14H     2  Nummer Startsektor (block device)         º
º 16H     4  Zeiger auf Volume Label, falls der Error-  º
º            code 0FH vom Treiber gesetzt (DOS 3.0+)    º
```

```
º 1AH     4  32-Bit-Startsektor (ab DOS 4.0) falls Bit 1º
º          im Device-Attribut gesetzt ist               º
º --- Kommandocode 05H ---                             º
º 0DH     1  von Einheit gelesenes Byte, falls das      º
º          BUSY-Bit bei Rückkehr gelöscht ist           º
º --- Kommandocode 06H,07H,0AH,0BH,0DH,0EH,0FH,85H,88H--º
º --- unbelegt                                         º
º --- Kommandocode 10H ---                            º
º 0DH     1  unbelegt                                  º
º 0EH     4  Transferadresse                           º
º 12H     2  Bytezähler                               º
º --- Kommandocode 13H, 19H ---                       º
º 0DH     1  Category Code                             º
º             00 unbekannt                             º
º             01 COMn:                                 º
º             03 CON                                   º
º             05 LPTn:                                 º
º             08 Disk                                  º
º 0EH     1  Funktionscode                             º
º 0FH     2  Kopie des DS beim IOCTL-Aufruf            º
º 11H     2  Offset Treiberheader                      º
º 13H     4  Zeiger auf Parameterblock INT 21 Aufruf   º
º            440CH, 440DH                              º
Û-----------------------------------------------------ì
```

Es werden noch die Funktionscodes 80H-88H unterstützt. Weitere Informationen über das Format des Header für diese Codes finden sich beim INT 2FH, Funktion 1510H. Der Aufruf erlaubt zum Beispiel die Einbindung von CD-ROM's.

5.6.4 Get Drive Data Table (AL = 03H)

Mit dieser Unterfunktion läßt sich die Adresse einer Tabelle mit den Daten des Laufwerkes ermitteln.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 08H (DRIVER.SYS)            º
º AL: 03H Get Data Table          º
û---------------------------------À
º      RETURN                     º
º DS:DI  resse Tabelle            º
Û---------------------------------ì
```

Im Registerpaar DS:DI wird die Adresse der Tabelle zurückgegeben. Diese Tabelle besitzt in Abhängigkeit von der DOS-Version folgenden Aufbau:

```
Ö-----------------------------------------------------------Ì
º Offs. Byte  Beschreibung                                  º
º --- DOS 3.30 Drive Data Table                            º
º 00H     4  Zeiger auf Folgetabelle                        º
º 04H     1  physikalische Unit Nummer (INT 13)             º
º 05H     1  logische Drive Nummer                          º
º 06H    19  BIOS Parameter Block                          º
º         2  Byte pro Sektor                                º
º         1  Sektor pro Cluster (FFH unbekannt)            º
º         2  Zahl der reserv. Sektoren                     º
º         1  Zahl der FAT's                                 º
º         2  Zahl der Stammverzeichniseinträge             º
º         2  Gesamtzahl der Sektoren                       º
º         1  Media Descriptor (00H unbekannt)              º
º         2  Sektoren pro FAT                              º
º         2  Sektoren pro Track                            º
º         2  Zahl der Köpfe                                º
º         2  Zahl der hidden Sektoren                      º
º 19H     1  Flags (Bit 6: 1 = 16-Bit-FAT                 º
º                        0 = 12-Bit-FAT                   º
º 1AH     2  Zahl der DEVICE OPEN-Calls ohne              º
º            DEVICE CLOSE-Calls                           º
º 1CH    11  Volume Label oder "NO NAME   "               º
º 27H     1  00H                                          º
º 28H     1  Device Typ (siehe INT 21,440DH)              º
º 29H     2  Flags:                                       º
º        Bit 0: Fixed Media                               º
º            1: Door Lock Support                         º
º            2: ---                                       º
º            3: Sektoren der Spur haben gleiche Größe     º
º            4: physikal. Einheit ist in mehrere log.     º
º               Einheiten aufgeteilt                      º
º            5: aktuelle log. Einheit als phys. Einheit   º
º          6-8: ---                                       º
º 2BH     2  Zahl der Zylinder                            º
º 2DH    19  BPB für höchste unterstützte Kapazität       º
º 40H     3  ---                                          º
º 43H     9  ---                                          º
º 4CH     1  LSB letzter aktiver Zylinder                 º
º 4DH     4  -- removable media ---                       º
º            Zeit letzter Zugriff in Clock Ticks          º
º            (FFFFFFFFH = nie zugegriffen)                º
º 4DH        -- fixed media ---                           º
º         2  Partition (FFFFH = primär, 0001 = extended)  º
º         2  absolute Zylinder Nummer Beginn der          º
º            Partition im physikal. Laufwerk (immer       º
º            0001H bei primären Partitionen)              º
º-----------------------------------------------------------º
º --- COMPAC DOS 3.31 Drive Data Table                     º
º 00H     4  Zeiger auf Folgetabelle                        º
º 04H     1  physikalische Unit Nummer (INT 13)             º
º 05H     1  logische Drive Nummer                          º
º 046    25  BIOS Parameter Block (wie DOS 4.01)           º
º 1FH     6  -----                                         º
º 25H     1  Flags (Bit 6: 1 = 16-Bit-FAT                 º
º                        0 = 12-Bit-FAT                   º
º 26H     2  -----                                         º
º 28H    11   Volume Label oder "NO NAME   "              º
º 33H     1  00H Abschluß Name                            º
º 34H     1  Device Typ (siehe INT 21,440DH)              º
º 35H     2  Flags:                                       º
º        Bit 0: Fixed Media                               º
º            1: Door Lock Support                         º
º            2: ---                                       º
º            3: Sektoren der Spur haben gleiche Größe     º
º            4: physikal. Einheit ist in mehrere log.     º
º               Einheiten aufgeteilt                      º
º            5: aktuelle log. Einheit als phys. Einheit   º
º          6-8: ---                                       º
º 37H     2  Zahl der Zylinder                            º
º 39H    19  BPB für höchste unterstützte Kapazität       º
º 52H     6  ---                                          º
º 58H     1  LSB letzter aktiver Zylinder                 º
º 59H     4  -- removable media ---                       º
º            Zeit letzter Zugriff in Clock Ticks          º
º            (FFFFFFFFH = nie zugegriffen)                º
º 59H        -- fixed media ---                           º
º         2  Partition (FFFFH = primär, 0001 = extended)  º
º         2  absolute Zylinder Nummer Beginn der          º
º            Partition im physikal. Laufwerk (immer       º
```

```
º            0001H bei primären Partitionen)            º
º-------------------------------------------------------º
º --- DOS 4.0 - 6.0 Drive Data Table                     º
º 00H     4  Zeiger auf Folgetabelle                     º
º 04H     1  physikalische Unit Nummer (INT 13)          º
º 05H     1  logische Drive Nummer                       º
º 06H    25  BIOS Parameter Block                        º
º         2  Byte pro Sektor                            º
º         1  Sektor pro Cluster (FFH unbekannt)         º
º         2  Zahl der reserv. Sektoren                  º
º         1  Zahl der FAT's                             º
º         2  Zahl der Stammverzeichniseinträge          º
º         2  Gesamtzahl Sektoren (siehe letztes DWORD)  º
º         1  Media Descriptor (00H unbekannt)           º
º         2  Sektoren pro FAT                           º
º         2  Sektoren pro Track                         º
º         2  Zahl der Köpfe                             º
º         4  Zahl der hidden Sektoren                   º
º         4  Zahl Sektoren, falls WORD ab Offset 08 = 0)º
º 1FH     1  Flags (Bit 6: 1 = 16-Bit-FAT               º
º                        0 = 12-Bit-FAT                º
º 20H     2  ---                                        º
º 22H     1  Device Typ (siehe INT 21,440DH)           º
º 23H     2  Flags:                                     º
º        Bit 0: Fixed Media                            º
º            1: Door Lock Support                      º
º            2: ---                                    º
º            3: Sektoren der Spur haben gleiche Größe  º
º            4: physikal. Einheit ist in mehrere log.  º
º               Einheiten aufgeteilt                   º
º            5: aktuelle log. Einheit als phys. Einheit º
º          6-8: ---                                    º
º 25H     2  Zahl der Zylinder                         º
º 27H    25  BPB für höchste unterstützte Kapazität    º
º 40H     7  ---                                       º
º 47H     4  -- removable media ---                    º
º            Zeit letzter Zugriff in Clock Ticks       º
º            (FFFFFFFFH = nie zugegriffen)             º
º 47H        -- fixed media DOS 4.x ---                º
º         2  Partition (FFFFH = primär, 0001 = extended)º
º         2  absolute Zylinder Nummer Beginn der       º
º            Partition im physikal. Laufwerk (immer    º
º            FFFFH bei primären Partitionen)           º
º 47H        -- fixed media DOS 5.0 ---                º
º         2  immer 0001                                º
º         2  absolute Zylinder Nummer Beginn der       º
º            Partition im physikal. Laufwerk           º
º -------------------------------------------------------º
º 4BH    11  Volume Label oder "NO NAME    "           º
º 56H     1  ---                                       º
º 57H     4  Seriennummer Medium                       º
º 5BH     8  Typ Filesystem "FAT 12    "  "FAT 16   "  º
º 63H     1  ---                                       º
Û-------------------------------------------------------ì
```

Durch die unterschiedliche Größe der unterstützten Speichermedien mußte die Tabelle in den verschiedenen DOS-Versionen mehrfach modifiziert werden. Der Funktionsaufruf wird erst ab DOS 3.3 unterstützt.

5.7 Kommunikation with SHARE (DOS 3.0 - 5.0, AH = 10H)

Diese Aufrufe erlauben eine Kommunikation mit dem Programm SHARE. Einige Aufrufe stehen nicht in allen DOS-Versionen zur Verfügung.

5.7.1 SHARE Installation Check (AX = 1000H)

Dieser Aufruf steht in allen DOS-Versionen oberhalb 3.0 zur Verfügung.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 10H (Share)                º
º AL: 00H Get Installationsstatus º
û--------------------------------À
º      RETURN                    º
º AL: Statusbyte                 º
Û--------------------------------ì
```

Mit dieser Funktion läßt sich der residente Teil von SHARE auf den Installationsstatus abfragen.

Es ist nur die Unterfunktion AL = 00 (Get Installationsstatus) implementiert. Aufrufe mit AL > 00H führen in DOS 3.x wegen eines Fehlers zu einer Endlosschleife. Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Treibers.

- Falls AL = 0 ist, dann ist der residente Teil von SHARE nicht vorhanden und kann installiert werden.
- Falls AL = 1 ist, dann ist der residente Teil von SHARE nicht vorhanden, er läßt sich aber auch nicht installieren.
- Falls AL = FFH ist, dann ist der residente Teil von SHARE installiert.

SHARE erlaubt innerhalb von Netzwerkanwendungen den Zugriff mehrerer Prozesse auf einzelne Dateien.

5.7.2 SHARE Turn ON File Sharing Checks (AX = 1080H)

Dieser Aufruf steht nur in DOS 4.x zur Verfügung. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 10H (Share)                º
º AL: 80H Sharing Check ON       º
û--------------------------------À
º      RETURN                    º
º AL: Statusbyte                 º
Û--------------------------------ì
```

Mit dieser Funktion läßt sich der SHARE-Funktionalität einschalten.

In DOS 4.x hat SHARE zwei Funktionen. Einmal werden FCB's für Medien größer als 32 Mbyte unterstützt. Weiterhin übernimmt SHARE die Prüfung gemeinsam geöffneter Dateien. Mittels des undokumentierten Kommandozeilenparameters /NC kann die Prüfung abgeschaltet werden.

In AL wird nach dem Rücksprung ein Statuscode übergeben:

```
 AL = F0H SHARING erfolgreich eingeschaltet.
      FFH SHARING-Check war bereits eingeschaltet.
```

Die Prüfung läßt sich mit der folgenden Funktion auch wieder abschalten.

5.7.3 SHARE Turn OFF File Sharing Checks (AX = 1081H)

Dieser Aufruf steht nur in DOS 4.x zur Verfügung. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 10H (Share)                 º
º AL: 81H Sharing Check OFF       º
û---------------------------------À
º      RETURN                     º
º AL: Statusbyte                  º
Û---------------------------------ì
```

Mit dieser Funktion läßt sich der SHARE-Funktionalität abschalten. In AL wird nach dem Rücksprung ein Statuscode übergeben:

```
 AL = F0H SHARING erfolgreich eingeschaltet.
      FFH SHARING-Check war bereits eingeschaltet.
```

Die Funktion AX = 1040H wird von SHARE ebenfalls benutzt, genaue Informationen über die Schnittstelle sind aber nicht bekannt. Die Funktionen AX=10xxH werden nur intern von DOS benutzt und dürften in der Regel für Anwendungsprogramme nicht von Interesse sein.

5.8 Kommunikation with Network Redirector (AH = 11H, DOS 3.1 - 5.0)

Mit dieser Funktion des INT 2F wird die Netzwerkumleitung in DOS unterstützt. Weiterhin dürfte DOS bereits einige Grundfunktionen eines installierbaren Filesystems zu beinhalten. Die Kommunikation mit diesen Routinen erfolgt über die Funktion AH=11H des INT 2F. In DOS 3.1 bis 3.3 sind die Aufruf des INT 2F im Kern implementiert. In DOS 4.x muß vorher das Programm IFSFUNC.EXE geladen werden, da die Funktionen dort erst installiert werden. Ab DOS 5.0 finden sich die Funktionen wieder im Kern. Es sind folgende Unterfunktionen implementiert.

5.8.1 Get Installation Status (AL = 00H)

Die Unterfunktion erlaubt eine Überprüfung, ob das Programm zur Netzwerkumleitung vorhanden ist.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 11H (Network Redirector)    º
º AL: 00H Get Installationsstatus º
û---------------------------------À
º      RETURN                     º
º AL: Statusbyte                  º
Û---------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Treibers.

- Falls AL = 0 ist, dann ist der Redirector nicht vorhanden und kann installiert werden.
- Falls AL = 1 ist, dann ist der Redirector nicht vorhanden, er läßt sich aber auch nicht installieren.
- Falls AL = FFH ist, dann ist der Redirector vorhanden.

5.8.2 Unterfunktionen Network Redirector

Die nachfolgende Tabelle gibt einen groben Überblick über die weiteren Funktionen des INT 2F mit AX = 11xxH. Für Anwendungsprogramme dürfte die praktische Verwertbarkeit der Aufrufe allerdings sehr begrenzt sein.

AH = 11H (Netzwerk Umleitung)
AL = 00H Installation check
AL = 01H Remove Remote Directory
AL = 02H Remove Remote Directory (nur DOS 4.X)
AL = 03H Make Remote Directory
AL = 04H Make Remote Directory (nur DOS 4.X)
AL = 05H CHDIR
AL = 06H Close Remote File
AL = 07H Commit Remote File
AL = 08H Read from Remote File
AL = 09H Write to Remote File
AL = 0AH Lock Region of File
AL = 0BH Unlock Region of File (nur DOS 3.X)
AL = 0CH Get Disk Space (ab DOS 3.1)
AL = 0DH unbekannt
AL = 0EH Set Remote File Attribut
AL = 0FH Get Remote File Attribut
AL = 10H unbekannt
AL = 11H Rename Remote File
AL = 12H unbekannt
AL = 13H Delete Remote File
AL = 14H unbekannt
AL = 15H unbekannt
AL = 16H Open Existing Remote File
AL = 17H Create/Trancate Remote File
AL = 18H Create/Truncate File Without CDS
AL = 19H Find First File Without CDS
AL = 1AH unbekannt
AL = 1BH FindFirst
AL = 1CH FindNext
AL = 1DH Close All Remote Files for Process
AL = 1EH Do Redirection (siehe INT 21, AX=5Fxx)
AL = 1FH Set Printer Setup (siehe INT 21, AX=5Exx)
AL = 20H Flush All DIsk Buffers
AL = 21H Seek From End Of Remote File
AL = 22H Process Termination Hook
AL = 23H Qualify Remote Filename
AL = 24H unbekannt
AL = 25H Redirected Printer Mode
AL = 26H unbekannt
AL = 27H unbenutzt
AL = 28H unbenutzt
AL = 29H unbenutzt
AL = 2AH unbekannt
AL = 2BH unbekannt
AL = 2CH unbekannt
AL = 2DH unkannt
AL = 2EH Extended Open/Create File (DOS 4.x+)
AL = 2FH unbekannt
AL = 30H unbekannt

Die genaue Aufrufschnittstelle (soweit bekannt) läßt sich /3,5/ entnehmen.

5.9 Kommunikation with DOS 3.x internal Funktions (AH = 12H)

Über diese Funktion des INT 2F wickelt DOS ab der Version 3.x viele seiner internen Funktionsaufrufe ab. Nachfolgend werden die wichtigsten Unterfunktionen beschrieben.

5.9.1 Get Installationsstatus (AL = 00H)

Mit dieser Unterfunktion läßt sich der Installationsstatus abfragen.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 12H (DOS 3.x)              º
º AL: 00H Get Installationsstatus º
û--------------------------------À
º      RETURN                    º
º AL: Statusbyte                 º
Û--------------------------------ì
```

Die Funktion gibt den Wert FFH zurück, um mit den Konventionen des INT 2F kompatibel zu bleiben.

Es sind weitere Unterfunktionen mit verschiedenen Codes belegt. Die folgende Tabelle gibt einen groben Überblick über die wichtigsten Funktionsaufrufe. Viele dieser Aufruf stehen nur intern für DOS zur Verfügung.

```
Ö-----------------------------------------------------------Ì
º            AH = 12H (DOS 3.x Funktionen)                 º
Û-----------------------------------------------------------ì
Ö-----------------------------------------------------------Ì
º AL = 00H Installation Check                              º
û-----------------------------------------------------------À
º AL = 01H  Close Current File                             º
û-----------------------------------------------------------À
º AL = 02H Get Interrupt Address                           º
º     Stack:  Wort mit der Vektor Nummer                   º
º                                                          º
º     Return: ES:BX Zeiger auf den Interrupt Vektor        º
º             Stack unchanged                              º
Û-----------------------------------------------------------ì
Ö-----------------------------------------------------------Ì
º AL = 03H Get DOS Data Segment                            º
º     Return: DS Segment of IBMDOS,MSDOS                   º
û-----------------------------------------------------------À
º AL = 04H Normalize Path Separator                        º
º     Stack:  Wort mit Zeichen zum Normalisieren           º
º     Return: AL = normalisiertes Zeichen                  º
º             (forward slash wird zu backslash)            º
º             Stack unchanged                              º
û-----------------------------------------------------------À
º AL = 05H Output Character                                º
º     Stack:  Wort mit auszugebendem Zeichen               º
º     Return: Stack unchanged (nur intern in DOS benutzbar)º
û-----------------------------------------------------------À
º AL = 06H Invoke Critical Error                           º
º     DI: Fehlercode                                       º
º     DS:DI Adresse Device Driver Header                   º
º     SS: DOS Datensegment                                 º
º     STACK: Wort welches in AX zum INT 24 übergeben wird  º
º     Return: AL = 0-3 für Abort, Retry, Ignore, Fail      º
º     (Stack unchanged)                                    º
û-----------------------------------------------------------À
º AL = 07H Make Disk Buffer most-recently used            º
º      DS:DI -> Disk Buffer                                º
º      Return: --                                          º
º Der angegebene Buffer wird an das Ende der Pufferliste   º
º verschoben. Der »least-recently used buffer**Fehler! Verweisquelle konnte nicht
gefunden werden.**º Anfang der Pufferliste. (nur intern in DOS benutzbar)    º
û-----------------------------------------------------------À
º AL = 08H Decrement SFT Reference Count                    º
º      ES:DI -> Adresse SFT                                º
º      Return: AX = neuer Wert des Wortes                  º
º      Erniedrigt die Zahl der Handles, (FFFFH = leer,     º
º      0000H = SFT unbenutzt)                              º
û-----------------------------------------------------------À
º AL = 09H Flush and Free Disk Buffer                      º
º      DS:DI -> Disk Buffer                                º
º      Return: ---                                         º
º Der Inhalt des Puffers wird auf die Disk ausgelagert und º
º der Puffer wird als unbenutzt markiert. (nur intern in   º
º DOS benutzbar)                                           º
û-----------------------------------------------------------À
º AL = 0AH Perform Critical Error Interrupt                º
º      DS = SS = DOS DS                                    º
º      STACK: Extended Error Code                          º
º      Return: AL: User Response (0=ignore,                º
º      1=retry & CF = 0, 2=abort, 3=fail)                  º
º (nur intern in DOS benutzbar)                            º
û-----------------------------------------------------------À
º AL = 0BH Signal Sharing Violation to User                 º
º (nur intern in DOS benutzbar)                            º
û-----------------------------------------------------------À
º AL = 0CH unbekannt                                       º
û-----------------------------------------------------------À
º AL = 0DH Get Date and Time                               º
º      SS = DOS DS                                         º
º      Return: AX = aktuelles Datum in gepackter Form      º
º              DX = aktuelle Zeit in gepackter Form        º
º Das Zeit- und Datumsformat entspricht dem Eintrag in dem º
º Directory (siehe auch INT 21, AX=5700H) (nur intern in   º
º DOS benutzbar)                                           º
û-----------------------------------------------------------À
º AL = 0EH Free all Disk Buffers                           º
º      Return: DS:DI -> Adresse des ersten Puffers         º
º (nur intern in DOS benutzbar)                            º
û-----------------------------------------------------------À
```

```
º AL = 0FH Make Buffer most-recently used                 º
º      DS:DI -> Adresse Disk Puffer                        º
º      Return: DS:DI -> Adresse nächster Puffer in der     º
º              Pufferliste. (nur intern in DOS benutzbar)  º
û-----------------------------------------------------------À
º AL = 10H Find free Disk Buffer                           º
º      DS:DI -> Adresse 1. zu prüfender Disk Buffer        º
º      Return: DS:DI -> 1. verfügbarer Disk Buffer         º
º      ZF = 0 freier Buffer gefunden                       º
º           1 kein freier Buffer gefunden                  º
Û-----------------------------------------------------------ì
Ö-----------------------------------------------------------Ì
º AL = 11H Normalize ASCIIZ Filename                       º
º       DS:SI -> ASCIIZ-Filename zum Normalisieren         º
º       ES:DI -> Buffer für den normalisierten Filenamen   º
º       Return: --                                         º
º Im Zielpuffer wird der Filename in Großbuchstaben ge-    º
º speichert. Slash wird in Backslash konvertiert.          º
û-----------------------------------------------------------À
º AL = 12H Get Length of ASCIIZ-String                     º
º       ES:DI -> Adresse des ASCIIZ-String                 º
º       Return: CX = Länge des Strings                     º
û-----------------------------------------------------------À
º AL = 13H Uppercase Character                             º
º      STACK: Wort mit dem zu konvertierenden Zeichen      º
º      Return: AL = Zeichen als Großbuchstabe              º
º      Stack unchanged                                     º
û-----------------------------------------------------------À
º AL = 14H Compare FAR Pointers                            º
º       DS:SI = erster Zeiger                              º
º       ES:DI = zweiter Zeiger                             º
º       Return: ZF = 1  Zeiger sind gleich                 º
º                    0  Zeiger sind ungleich               º
û-----------------------------------------------------------À
º AL = 15H Flush Buffer                                    º
º      DS:DI -> Adresse Disk Buffer                        º
º      STACK: Wort mit der Drive Nummer für den der Buffer º
º      Return: ---                                         º
º      Stack unchanged                                     º
º                                                          º
º Auf dem Stack wird ein Wort mit den Drive Nummern über-  º
º geben, deren Buffer übersprungen werden soll. Der Puffer º
º wird ignoriert, falls die Drive Nummer zu einem anderen  º
º Puffer gehört. (nur intern in DOS benutzbar)             º
û-----------------------------------------------------------À
º AL = 16H Get Address of System File Table (SFT)          º
º       BX = System File Table Entry Nummer                º
º       Return: CF: 0 -> ok                                º
º               ES:DI -> Adresse System File Table Entry   º
º               CF: 1 -> Wert in BX war größer als in      º
º               FILES= erlaubt                             º
û-----------------------------------------------------------À
º AL = 17H Get Current Directory Structure                 º
º      STACK: Wort mit Drive (0 = A:, 1 = B:, etc)         º
º      Return: CF = 1 Fehler (Drive > Lastdrive            º
º                 = 0 o.k.                                 º
º     DS:SI -> Directory Struktur für das spezifierte Driveº
º      Stack unchanged  (nur intern in DOS benutzbar)      º
û-----------------------------------------------------------À
º AL = 18H Get Caller's Registers                          º
º      Return: DS:SI -> Stackadresse in dem die Register   º
º      AX,BX,CX,DX,SI,DI,BP,DS,ES gesichert wurden.        º
º      (nur gültig bei Aufrufen innerhalb DOS)             º
û-----------------------------------------------------------À
º AL = 19H unbekannt                                       º
û-----------------------------------------------------------À
º AL = 1AH Get Files Drive                                 º
º      DS:SI -> Adresse Filename                           º
º      Return: AL = Drive (0 = default, 1 = A:, etc,       º
º              FFh = invalid)                              º
û-----------------------------------------------------------À
º AL = 1BH Set Year/Length of February                     º
º      CL = Jahr - 1980                                    º
º      Return: AL = Tage im Februar                        º
º DS muß auf das DOS-Codesegment gesetzt sein.             º
Û-----------------------------------------------------------ì
Ö-----------------------------------------------------------Ì
º AL = 1CH Checksum Memory                                 º
º      DS:SI -> Startadresse Checksumberechnung            º
º      CX = Zahl der Bytes                                 º
```

```
°      DX = Checksumregister mit Startwert                 °
°      Return: DX = Checksumme (nur intern in DOS nutzbar) °
û-----------------------------------------------------------À
° AL = 1DH Sum Memory                                      °
°      DS:SI -> zu summierende Speicheradresse             °
°      CX = 0000H                                          °
°      DX = Limit                                          °
°      Return: AL = Byte welches Limit überschreitet       °
°              CX = Bytes vor Erreichen des Limits         °
°              DX = Rest nach Addition des 1. Bytes in CX  °
û-----------------------------------------------------------À
° AL = 1EH Compare Filenames                               °
°      DS:SI -> first ASCIIZ-Filename                      °
°      ES:DI -> second ASCIIZ-Filename                     °
°      Return: ZF = 1  Filenamen equivalent                °
°                  0  Filenamen ungleich                   °
û-----------------------------------------------------------À
° AL = 1FH Build Current Directory Strukture               °
°      STACK: Wort mit dem Drive Buchstaben                °
°      Return: ES:DI -> Adresse Drive-Info-Block           °
°      Stack unchanged  (nur intern in DOS benutzbar)      °
û-----------------------------------------------------------À
° AL = 20H Get Job File Table Number Entry                 °
°      BX = File Handle                                    °
°      Return: CF = 1 Fehler                               °
°              AL = 6 (invalid File Handle)                °
°              CF = 0 o.k.                                 °
°      BYTE ES:DI = Job-File-Table-Entry-Number für        °
°      den übergebenen File Handle (FFH unbenutzt)         °
û-----------------------------------------------------------À
° AL = 21H Canonicalize Filen Name                         °
°      DS:SI File Name zum expandieren                     °
°      ES:DI 128-Byte Ergebnisbuffer                       °
°      Return: ----                                        °
°      identisch zur Funktion 60H des INT 21               °
û-----------------------------------------------------------À
° AL = 22H Set Extended Error Info                         °
°      nur intern durch DOS nutzbar                        °
û-----------------------------------------------------------À
° AL = 23H Check if Character Device                       °
°      nur intern durch DOS nutzbar                        °
û-----------------------------------------------------------À
° AL = 24H Delay                                           °
°      Verzögerungsschleife, intern in DOS nutzbar         °
û-----------------------------------------------------------À
° AL = 25H Get Length of ASCIIZ-String                     °
°      DS:SI -> Adresse ASCIIZ-String                      °
°      Return: CX = Stringlänge                            °
û-----------------------------------------------------------À
° AL = 26H Open File (ab DOS 3.3 intern)                   °
° AL = 27H Close File (ab DOS 3.3 intern)                  °
° AL = 28H Move File Pointer (ab DOS 3.3 intern)           °
° AL = 29H Read From File (ab DOS 3.3 intern)              °
û-----------------------------------------------------------À
° AL = 2AH Set Fastopen Entry Point                        °
°      BX = (DOS 4.x) Entry Point to set (0001H or 0002H)  °
°      DS:SI -> FASTOPEN Entry Point                       °
°      Return: CF = 1 Entry Point bereits gesetzt          °
û-----------------------------------------------------------À
° AL = 2BH IOCTL (ab DOS 3.3 intern)                       °
° wirkt wie INT 21, Funktion 44H, läßt sich aber nur       °
° intern in DOS aufrufen.                                  °
û-----------------------------------------------------------À
° AL = 2CH Get Device Chain                                °
°      Return: BX:AX -> Header des 2. Device Driver in der °
°      Treiber Kette (NUL ist der 1. Treiber)              °
û-----------------------------------------------------------À
° AL = 2DH Get Extended Error Code (ab DOS 3.3)            °
°      Return: AX -> Extended Error Code                   °
û-----------------------------------------------------------À
° AL = 2EH Get/Set Error Table Adresses (ab DOS 4.0)       °
°      DL: Subfunktion                                     °
°        00H Get DOS Error Table                           °
°        01H Set DOS Error Table                           °
°        02H Get Parameter Error Table                     °
°        03H Set Parameter Error Table                     °
°        04H Get Critical Error Table                      °
°        05H Set Critical Error Table                      °
°        06H Get unknown Error Table                       °
°        07H Set unknown Error Table                       °
```

```
º        08H Get Error Meesage Retriever                  º
º        09H Set unknown Error Table                      º
º      Die genaue Belegung ist allerdings nicht bekannt   º
û-----------------------------------------------------------À
º AL = 2FH Set DOS Version Number to Return (ab DOS 4.0)   º
º      DX: DOS Versions Nummer (0000H = Original DOS-      º
º          Version zurückgeben)                            º
Û-----------------------------------------------------------ì
```

Die Unterfunktionen 2EH und 2FH sind erst ab DOS 4.x vorhanden. Mit der Funktion 2FH läßt sich offenbar bereits ab DOS 4.0 festlegen, welche Versionsnummer das Betriebssystem zurückgibt.

5.10 Set Disk Interrupt Handler (AH = 13H, ab DOS 3.3)

Das Programm IO.SYS fängt den INT 13 des BIOS ab und setzt einige Filterfunktionen ein (z.B. Überwachung auf Diskettenwechsel, Aufzeichnung der Formataufrufe, BIOS-Bugfixes, etc.).

Mit dieser Unterfunktion läßt sich der Vektor des INT 13 abfangen. In DS:DX ist die Adresse des neuen Handlers für den INT 13 zu übergeben. In ES:BX wird die Adresse des Vektors übergeben, die das System beim Warmstart wieder einsetzen muß.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 13H (Set DISK Int-Handler   º
º DS:DX Adresse Handler           º
º ES:BX Adresse Buffer            º
û---------------------------------À
º      RETURN                     º
º ----                            º
Û---------------------------------ì
```

Nach dem Aufruf enthalten DS:DX und ES:BX die Werte, die beim vorherigen Aufruf gesetzt wurden. Dies ist eine recht gefährliche Sicherheitslücke in DOS, da einige Viren sich dadurch die Original ROM-Einsprungpunkte des INT 13 abrufen können und dadurch Schutzsoftware umgehen.

5.11 Kommunikation with NLSFUNC.COM (AH = 14H)

Mit dem Aufruf läßt sich mit dem Programm NLSFUNC.COM ab DOS 3.3 kommunizieren.

5.11.1 Get Installation Status (AL = 00H)

Mit dieser Unterfunktion läßt sich feststellen, ob NLSFUNC resident geladen ist.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 14H (NLSFUNC)              º
º AL: 00H Get Installationsstatus º
û--------------------------------À
º      RETURN                    º
º AL: Statusbyte                 º
Û--------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Programmes.

- Falls AL = 0 ist, dann ist der residente Teil von NLSFUNC nicht vorhanden und kann installiert werden.
- Falls AL = 1 ist, dann ist der residente Teil von NLSFUNC nicht vorhanden, er läßt sich aber auch nicht installieren.
- Falls AL = FFH ist, dann ist der residente Teil von NLSFUNC installiert.

NLSFUNC übernimmt ab DOS 3.3 die länderspezifische Anpassung. Die folgende Tabelle gibt einen groben Überblick über die wichtigsten Funktionsaufrufe. Viele dieser Aufruf stehen nur intern für DOS zur Verfügung.

```
Ö-----------------------------------------------------------Ì
º            AH = 14H (NLSFUNC.COM)                         º
Û-----------------------------------------------------------ì
Ö-----------------------------------------------------------Ì
º AL = 00H Installation Check                               º
û-----------------------------------------------------------À
º AL = 01H Change Code Page                                 º
º AL = 02H Get Country Info                                 º
º AL = 03H Set Country Info                                 º
º AL = 04H Get Country Info                                 º
º   BX: Code Page                                          º
º   DX: Country Code                                       º
º   DS:SI Adresse Code Page Struktur                       º
º   ES:DI Adresse User Puffer (nur AL = 02,04)              º
º   BP:   Subcode wie INT 21, AH=65H (nur AL=02)           º
º     Return: AL: Status (00 = ok, sonst Errorcode)         º
Û-----------------------------------------------------------ì
```

Die Struktur des Code-Page-Puffers ist versionsabhängig und wird hier nicht aufgeführt, da die Funktion für Anwendungsprogramme nicht relevant ist.

5.12 Kommunikation with CD-ROM/Graphics (AH = 15H)

Über diese INT 2F-Funktion wird die Kommunikation mit dem CD-ROM-Treiber und in DOS 4.0 mit dem Programm GRAHICS.COM abgewickelt.

5.12.1 GRAPHICS Get Installation Status (AL = 00H)

Bis DOS 3.31 besaß das Programm GRAPHICS keine Überprüfung des Installationsstatus. Deshalb konnte der Code mehrfach geladen werden. In DOS 4.0 wurde der Installationscheck für GRAPHICS.COM auf den INT 2F mit der Funktion 1500H gelegt. Damit wird diese Unterfunktion doppelt benutzt, da auch die CD-ROM-Extension mit dieser Funktion arbeitet (siehe unten). Die Installationsabfrage erfolgt mit folgenden Parametern.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 15H (GRAPHICS)              º
º AL: 00H Get Installationsstatus º
û---------------------------------À
º      RETURN                     º
º AX: FFFFH                       º
Û---------------------------------ì
```

Der Aufruf gerät mit dem Installationscheck der CD-ROM-Erweiterung in Konflikt. Deshalb wurde die Statusabfrage des GRAPHICS-Treibers ab der DOS-Version 4.01 auf die Funktion AC00H des INT 2F verlegt.

5.12.2 Kommunikation with CD-ROM (AL = 00H)

Über diese Gruppe von Funktionen wird ebenfalls die Kommunikation mit dem Microsoft-CD-ROM-Treiber abgewickelt. Für den Installationscheck besitzt der Treiber folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 15H (CD-ROM)                º
º AL: 00H Get Installationsstatus º
º BX: 0000H                       º
û---------------------------------À
º      RETURN                     º
º BX: Zahl der CD-ROM Drives      º
º CX: Starting Driveletter        º
Û---------------------------------ì
```

Der Aufruf folgt nicht den üblichen DOS-INT-2F-Konventionen. Im Register BX wird die Zahl der benutzten CD-ROM-Laufwerke zurückgegeben. CX hält das Zeichen für das Laufwerk, von dem gestartet wurde (0 = A:).

Die CD-ROM-Treiber bieten weitere Unterfunktionen, die kurz in folgender Tabelle aufgeführt sind.

```
Ö-----------------------------------------------------------Ì
º             AH = 15H  CD-ROM-Treiber                      º
Û-----------------------------------------------------------ì
Ö-----------------------------------------------------------Ì
º AL = 0 Get Installation Status                            º
û-----------------------------------------------------------À
º AL = 01H CD-ROM Get Drive-Device-List                      º
º      ES:BX -> Buffer für die Drive Letter Liste           º
º      Return: Buffer mit den Daten                         º
º Für jedes Laufwerk sind 5 Byte in der Liste erforder-     º
º lich (1 Byte : Subunit Nummer im Treiber, 1 DWORD         º
º Adresse Device Driver Header)                             º
û-----------------------------------------------------------À
º AL = 02H CD-ROM Get Copyright File Name                    º
º      ES:BX -> 38-Byte Buffer (Name Copyright File)        º
º      CX = Drive Nummer (0=A:)                             º
º      Return: CF = 1 Drive kein CR-ROM                     º
º              AX = 15 (invalid Drive)                      º
û-----------------------------------------------------------À
º AL = 03H CD-ROM Get abstract File Name                     º
º      ES:BX -> 38-Byte Buffer (abstract File Name)         º
º      CX = Drive Nummer (0=A:)                             º
º      Return: CF = 1 Drive kein CR-ROM                     º
º              AX = 15 (invalid Drive)                      º
û-----------------------------------------------------------À
º AL = 04H CD-ROM Get bibliographic Doc File Name           º
º      ES:BX -> 38-Byte Buffer (bibl. Doc File Name)        º
º      CX = Drive Nummer (0=A:)                             º
º      Return: CF = 1 Drive kein CR-ROM                     º
º              AX = 15 (invalid Drive)                      º
û-----------------------------------------------------------À
º AL = 05H CD-ROM Read VTOC                                 º
º      ES:BX -> 2048-Byte Buffer                            º
º      CX = Drive Nummer (0=A:)                             º
º      DX = Sektor Index (0 first vol descriptor, 1=..)     º
º      Return: CF = 1 Drive kein CR-ROM                     º
º              AX = 15 (invalid Drive)                      º
º              AX = 21 (Drive not ready)                    º
º              CF = 0  ok                                   º
º              AX = Volume descriptor (1=Standard,          º
º              FFH=Standard,00=other)                       º
û-----------------------------------------------------------À
º AL = 06H CD-ROM Turn Debugging ON                         º
º      BX = Debug enable                                    º
º (reserviert für Entwicklungszwecke)                       º
û-----------------------------------------------------------À
º AL = 07H CD-ROM Turn Debugging OFF                        º
º      BX = Debug disable                                   º
º (reserviert für Entwicklungszwecke)                       º
û-----------------------------------------------------------À
º AL = 08H CD-ROM Absolute Disk Read                        º
º      ES:BX -> Buffer                                      º
º      CX = Drive Nummer (0=A:)                             º
º      SI:DI = Start Sector Nummer                          º
º      DX = Sektor Zahl zu lesen                            º
º      Return: CF = 1 Fehler                                º
º              AX = 15 (invalid Drive)                      º
º              AX = 21 (Drive not ready)                    º
Û-----------------------------------------------------------ì
Ö-----------------------------------------------------------Ì
º AL = 09H CD-ROM Absolute Disk Write                       º
º      ES:BX -> Buffer                                      º
º      CX = Drive Nummer (0=A:)                             º
º      SI:DI = Start Sector Nummer                          º
º      DX = Sektor Zahl zu schreiben                        º
º      Return: CF = 1 Fehler                                º
º              AX = 15 (invalid Drive)                      º
º              AX = 21 (Drive not ready)                    º
û-----------------------------------------------------------À
º AL = 0AH reserviert                                       º
û-----------------------------------------------------------À
º AL = 0BH CD-ROM (2.0) Drive Check                         º
º      CX = Drive Nummer (0=A:)                             º
º      Return: BX = ADADH falls MSCDEX.EXE geladen          º
º              AX = 0000H Drive nicht supported             º
º              AX = <> 0  Drive supported                   º
û-----------------------------------------------------------À
º AL = 0CH CD-ROM (2.0) MSCDEX Version                      º
º      Return: BH = Hauptversion                            º
º              BL = Unterversion                            º
```

```
° MSCDEX Versionen vor 1.02 geben BX=0 zurück           °
û-------------------------------------------------------À
° AL = 0DH CD-ROM (2.0) Get Drive Letters               °
°      ES:BX -> Buffer für Drive Letter List            °
°      Return: ---                                      °
° In der Liste wird für jedes Drive ein Byte eingetra-  °
° gen (0 = A:, etc.)                                    °
û-------------------------------------------------------À
° AL = 0EH CD-ROM Get/Set Volume Descriptor Preference  °
°      CX = Drive Nummer       (2.0)                    °
°      BX = Subfunktion                                 °
°        00H Get Preference                             °
°        DX = 00H                                       °
°      Return: DX = Preference Settings                 °
°        01H Set Preference                             °
°        DH =    Volume Descriptor Preference           °
°          01H = Primary Volume Descriptor              °
°          02H = Supplementary Volume Descriptor        °
°        DL = Supplementary Volume Descript. Preference °
°          01H = shift-Kanji                            °
°          01H = Primary Volume Descriptor              °
°      Return: CF = 1 Fehler                            °
°          AX = (15=invalid Drive,1=invalid Function)   °
û-------------------------------------------------------À
° AL = 0FH CD-ROM (2.0) Directory Entry                 °
°      CX = Drive Number (0=A:)                         °
°      ES:BX -> ASCIIZ-Path-Name                        °
°      SI:DI -> 255-Byte Buffer for Directory Entry     °
°      Return: CF = 1 Fehler                            °
°              AX = Fehlercode                          °
°              CF = 0 ok                                °
°              AX = Disk Format                         °
°              (0=High Sierra, 1=ISO 9669)              °
û - - - - - - -- - - - - - - - - - - - - - - - - - - - - À
°      Format der Directory Einträge:                   °
°                                                       °
° Offs. Byte    Bedeutung                               °
° 00H    1   Länge des Directory Eintrags              °
° 01H    1   Länge des XAR in LBN's (?)                °
° 02H    4   LBN of data, Intel (little-endian) Format °
° 06H    4   LBN of data, Motorola (big-endian) Format °
° 0AH    4   Filelänge im Intel Format                 °
° 0EH    4   Filelänge im Motorola Format             °
°                                                       °
Û-------------------------------------------------------ì
Ö-------------------------------------------------------Ì
°              ---High Sierra---                        °
°                                                       °
° 12H    6   Datum und Zeit                           °
° 18H    1   Bit Flags                                °
° 19H    1   reserviert                               °
°                                                       °
°               ---ISO 9660---                          °
°                                                       °
° 12H    7   Datum und Zeit                           °
° 19H    1   Bit Flags                                °
°                                                       °
°              ---beide Formate---                      °
°                                                       °
° 1AH    1   Interleave Faktor                        °
° 1BH    1   Interleave skip Faktor                   °
° 1CH    2   Volume set sequence number, Intel Format °
° 1EH    2   Volume set sequence number, Motorola Format°
° 20H    1   Länge des Filenamens                     °
° 21H    N   Bytes Filename                           °
°        1   (optional) padding if Filename odd Length °
°        N   Bytes System Data                        °
°                                                       °
û-------------------------------------------------------À
° AL = 10H CD-ROM (2.0) Send Device Driver Request      °
°      CX = CD-ROM Drive Letter (0 = A, 1 = B, etc)     °
°      ES:BX -> CD-ROM Device Driver Request Header     °
°      Return : ---                                     °
°                                                       °
°      Format des Device Driver Request Header:         °
°                                                       °
° Offs. Byte  Beschreibung                              °
°                                                       °
° 00H     1  Länge des Request Headers                  °
° 01H     1  Subunit within Device Driver               °
```

```
º 02H     1  Kommando Code                                  º
º           00H INIT                                        º
º           01H MEDIA CHECK (Block Devices)                 º
º           02H BUILD BPB (Block Devices)                   º
º           03H IOCTL INPUT                                 º
º           04H INPUT                                       º
º           05H NONDESTRUCTIVE INPUT, NO WAIT               º
º           06H INPUT STATUS                                º
º           07H INPUT FLUSH                                 º
º           08H OUTPUT                                      º
º           09H OUTPUT WITH VERIFY                          º
º           0AH OUTPUT STATUS                               º
º           0BH OUTPUT FLUSH                                º
º           0CH IOCTL OUTPUT                                º
º           0DH (DOS 3.x) DEVICE OPEN                       º
º           0EH (DOS 3.x) DEVICE CLOSE                      º
º           0FH (DOS 3.x) REMOVABLE MEDIA (block devices)   º
º           10H (DOS 3.x) OUTPUT UNTIL BUSY                 º
º           11H - 12H reserviert                            º
º           13H (ab DOS 3.2) GENERIC IOCTL                  º
º           14H - 16H reserviert                            º
º           17H (DOS 3.2+) GET LOGICAL DEVICE               º
º           18H (DOS 3.2+) SET LOGICAL DEVICE               º
º           80H (CD-ROM) READ LONG                          º
º           81H (CD-ROM) reserved                           º
º           82H (CD-ROM) READ LONG PREFETCH                 º
º           83H (CD-ROM) SEEK                               º
º           84H (CD-ROM) PLAY AUDIO                         º
º           85H (CD-ROM) STOP AUDIO                         º
Û-----------------------------------------------------------ì
Ö-----------------------------------------------------------Ì
º           86H (CD-ROM) WRITE LONG                         º
º           87H (CD-ROM) WRITE LONG VERIFY                  º
º           88H (CD-ROM) RESUME AUDIO                       º
º                                                           º
º 03H     1  Status (gesetzt durch Device Driver)          º
º              Bit 15: Error                                º
º              Bit 9: Busy                                  º
º              Bit 8: Done                                  º
º              Bits 7-0: Error Code falls Bit 15 = 1        º
º                00H Write-Protect violation                º
º                01H unbekannte Unit                        º
º                02H Drive not ready                        º
º                03H unbekanntes Kommando                   º
º                04H CRC-Error                              º
º                05H bad Drive Request Struktur Länge       º
º                06H Seek-Error                             º
º                07H unbekanntes Media                      º
º                08H Sektor nicht gefunden                  º
º                09H Printer out of Paper                   º
º                0AH Write fault                            º
º                0BH Read fault                             º
º                0CH General failure                        º
º                0DH reserved                               º
º                0EH reserved                               º
º                0FH invalid Disk change                    º
º 05H     8  reserviert                                    º
º 0DH        Zusatzinformationen beginnen hier            º
Û-----------------------------------------------------------ì
```

5.13 Kommunikation with WINDOWS, DOS, DPMI (AH = 16H)

Die Funktion 16xxH des INT 2F wird von mehreren Programmen belegt. Einmal wickelt WINDOWS einige Kommunikationsfunktionen über diese Multiplexerfunktion ab. Weiterhin belegt das DOS-Protected-Mode-Interface (DPMI) Subfunktionen des INT 2F mit dem Code 16XXH. Zuletzt wurde mit DOS 5.0 die Kommunikation mit dem TaskSwitcher auf diese Funktion gelegt. Nachfolgend möchte ich auf die einzelnen Schnittstellen eingehen.

5.13.1 WINDOWS Enhanced Mode Install Check (AX = 1600H)

Über diesen Funktionsaufruf läßt sich in Windows von DOS-Applikationen prüfen, welcher Mode eingeschaltet ist. Es gilt dabei folgende Schnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 2F              °
°                                 °
° AH: 16H (WINDOWS)               °
° AL: 00H Get Mode                °
û---------------------------------À
°      RETURN                     °
° AL: Statusbyte                  °
Û---------------------------------ì
```

Nach dem Aufruf enthält das Register AL einen Statuscode, der den Windows-Mode spezifiziert. Hierfür gilt folgende Kodierung:

```
AL = 00H WINDOWS 3.x Enhanced Mode oder Windows/386 läuft nicht
AL = 80H WINDOWS 3.x Enhanced Mode oder Windows/386 läuft nicht
AL = 01H WINDOWS/386 2.x läuft
AL = FFH WINDOWS/386 2.x läuft
AL = sonst ->
     AL WINDOWS Hauptversion (>= 3)
     AH WINDOWS Unterversion
```

Mit diesem API-Aufruf kann aus einem DOS-Programm heraus ermittelt werden, in welchem Mode WINDOWS arbeitet. Der Aufruf ist auch im WINDOWS 286-DOS-Extender implementiert.

5.13.2 WINDOWS/386 2.x Get API Entry Point (AX = 1602H)

Über diesen Funktionsaufruf läßt sich in Windows/386 2.x der Eintrittspunkt für das API ermitteln. Es gilt dabei folgende Schnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 2F              °
°                                 °
° AH: 16H (WINDOWS)               °
° AL: 02H Get API Entry           °
û---------------------------------À
°      RETURN                     °
° ES:DI Entry Point               °
Û---------------------------------ì
```

Nach dem Aufruf enthält das Registerpaar ES:DI die Adresse der API-Eintrittsroutine. Der Aufruf wird in Windows 3.x aus Kompatibilitätsgründen noch unterstützt. Wird zu der Adresse in ES:DI mit AX = 0000 und ES:DI = RETURN-Adresse per JMP FAR verzweigt, enthält BX nach dem Aufruf die ID-Nummer der aktuellen virtuellen Maschine.

5.13.3 WINDOWS Enhanced Mode Broadcast (AX = 1605H)

Über diesen Funktionsaufruf wird beim Eintritt in den WINDOWS 3.0 Enhanced Mode oder beim Start des Microsoft DOS 286-Extenders ein Broadcast an alle TSR-Programme und DOS-Treiber abgesetzt. Es gilt dabei folgende Schnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 16H (WINDOWS)              º
º AL: 05H (Broadcast)            º
º CX: 0000H                      º
º DX: Flags                      º
º ES:BX 0000:0000                º
º DS:SI 0000:0000                º
º DI: Versionsnummer             º
û--------------------------------À
º      RETURN                    º
º CX: 0 ok to load               º
º    <> 0 not load               º
Û--------------------------------ì
```

Vor dem Aufruf sind die Register ES:BX, DS:SI und CX mit den Werten 0000H zu laden. In DI wird die Versionsnummer (Hauptversion im oberen Byte, Unterversion im unteren Byte) übergeben. Das Register DX fungiert als Flag, welches signalisiert, ob der Aufruf durch WINDOWS oder den DOS-Extender ausgelöst wurde. DX besitzt dabei folgende Kodierung:

```
DX Bit 0: 1 MS-286-DOS-Extender Initialisierung
          0 MS-WINDOWS Enhanced-Mode Initialisierung
   Bit 1-15: reserviert
```

Der WINDOWS Enhanced-Mode-Lader, sowie der MS-DOS-286-Extender aktivieren bei der Initialisierung die Funktion 1605H des INT 2F. Auch der DOS 5.0 TaskSwitcher führt einen solchen Aufruf aus. DOS-Einheitentreiber oder TSR-Programme können den Funktionsaufruf überwachen und gegebenenfalls geeignete Werte an den rufenden Prozess zurückgeben. Das Register CX dient dabei als Flag. Wird ein Wert CX <> 0 zurückgegeben, signalisiert dies dem rufenden Prozess, daß WINDOWS nicht im Enhanced Mode geladen werden kann. Es ist dann Aufgabe des jeweiligen Treibers eine geeignete Fehlermeldung auszugeben, die erklärt, warum WINDOWS nicht geladen werden kann. Wird dagegen CX=0 zurückgegeben, kann sich Windows installieren. Jeder Handler in der Kette der INT 2F Funktion 1605H muß nach dem Aufruf sofort die Kontrolle an den nächsten Handler in der Kette übergeben. Die Register dürfen dabei nicht verändert werden. Erst nachdem der Handler die Kontrolle zurückerlangt, kann er Werte an den rufenden Prozess zurückgeben. Benötigt der Handler lokale Daten auf einer VM-Basis, muß dies dem rufenden Prozess mitgeteilt werden. Hierzu dienen die beiden Zeiger ES:BX und DS:SI. In ES:BX wird die Adresse einer Startup-Info-Struktur geführt. Diese Struktur besitzt folgenden Aufbau.

```
Offset Bytes Bemerkung
  00H    2   Haupt- und Unterversion der Info-Struktur
  02H    4   Zeiger auf Next-Startup-Info-Struktur
             oder der Wert 0000:0000
  06H    4   Zeiger auf den ASCIIZ-Namen des Virtual Device
             File oder 0000:0000
  0AH    4   Adresse Virtual-Device-Reference Data
  0EH    4   Zeiger auf den Instance-Data-Record oder
             0000:0000
```

Für jeden Eintrag im *Instance Data Record* ist folgende Struktur vorzusehen:

```
Offset Bytes Bemerkung
  00     4   Zeiger auf die eigentlichen Instance Daten (oder 0:0)
  04     2   Länge des Instance Datenbereiches
```

Benötigt ein Handler lokale Daten, muß er den vom vorhergehenden Handler in ES:BX übergebenen Vektor in der eigenen *Startup Info Struktur* im Feld *Next-Startup-Info-*

Strukture ablegen. Dann wird in ES:BX die Adresse der eigenen Startup-Info-Struktur eingetragen.

In DS:SI kann die Adresse der Virtual86-Mode enable/disable Callback-Funktion hinterlegt werden. Ein residentes Programm kann den V86-Mode enable/disable Vektor in DS:SI mit einem Wert belegen. Dann ist in CX ein Wert <> 0 zurückzugeben, damit die Installation von WINDOWS oder des Extenders abgebrochen wird.

Die V86-Mode enable/disable Prozedur wird mit folgenden Parametern aufgerufen:

```
 AX= 0000H disable V86-Mode
     0001H enable V86-Mode
```

Weiterhin ist beim Aufruf das Interruptsystem gesperrt. Bei einem Fehler ist nach dem Aufruf das Carry-Flag gesetzt.

5.13.4 WINDOWS Enhanced Mode und 286-DOS-Extender Exit Broadcast (AX = 1606H)

Über diesen Funktionsaufruf wird beim Eintritt in den WINDOWS 3.0 Enhanced Mode oder beim Start des Microsoft DOS-286-Extenders ein Broadcast beendet, falls der Aufruf 1605H mit CX <> 0 quittiert wird.

```
Ö---------------------------------Ì
°      CALL:  INT 2F              °
°                                 °
° AH: 16H (WINDOWS)               °
° AL: 06H (Exit Broadcast)        °
° DX: Flags                       °
û---------------------------------À
°      RETURN                     °
Û---------------------------------ì
```

Für das Flag in DX gilt die gleiche Kodierung wie bei der Funktion 1605H.

Dieser Aufruf wird noch im Real-Mode ausgeführt. Das System schaltet dann nicht in den Enhanced- oder Extendermode um.

5.13.5 WINDOWS Virtual Device Call out API (AX = 1607H)

Dieser Funktionsaufruf bietet einen Mechanismus für virtuelle Geräte im WINDOWS Enhanced Mode mit DOS Einheitentreibern zu kommunizieren.

```
Ö---------------------------------Ì
°      CALL:  INT 2F              °
°                                 °
° AH: 16H (WINDOWS)               °
° AL: 07H (VD Call out API)       °
° BX: Virtual Device ID           °
û---------------------------------À
°      RETURN                     °
Û---------------------------------ì
```

Beim Aufruf ist in BX die ID-Nummer der virtuellen Einheit zu übergeben (siehe Funktion 1684H).

5.13.6 WINDOWS Enhanced Mode Init complete Broadcast (AX = 1608H)

Dieser Funktionsaufruf wird aktiviert, nachdem alle installierbaren Einheiten initialisiert wurden.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 16H (WINDOWS)               º
º AL: 08H (Complete Braoadcast)   º
û---------------------------------À
º      RETURN                     º
Û---------------------------------ì
```

Real-Mode Software kann nur zwischen dem Aufruf AX = 1605H und dem Aufruf AX = 1608H aktiviert werden. Treiber müssen bei der Initialisierung den Aufruf AX = 1608H überwachen.

5.13.7 WINDOWS Enhanced Mode Begin Exit Broadcast (AX = 1609H)

Dieser Funktionsaufruf wird aktiviert, sobald der WINDOWS Enhanced Mode verlassen werden soll.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 16H (WINDOWS)               º
º AL: 09H (Exit Braoadcast)       º
û---------------------------------À
º      RETURN                     º
Û---------------------------------ì
```

Der Aufruf wird nur beim Beginn einer normalen Exit-Sequenz aktiviert. Bei einem Fehler-Exit unterbleibt dieser Aufruf.

5.13.8 Idle Call / Release Current VM Time Slice (AX = 1680H)

Dieser Funktionsaufruf wird von WINDOWS 3.x, von DOS 5.0, von DPMI 1.0 und von OS/2 2.0 unterstützt. Hauptaufgabe ist es, dem Betriebssystem mitzuteilen, daß der aktuelle Prozeß keine Rechenzeit mehr benötigt. Hierbei gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 16H (Idle Call)             º
º AL: 80H                         º
û---------------------------------À
º      RETURN                     º
Û---------------------------------ì
```

Ab MS-DOS 5.0 wird der Aufruf beim Warten auf Zeicheneingaben permanent ausgelöst. Dadurch können TSR-Programme die Funktion abfangen und so feststellen, wann DOS nicht beschäftigt ist. Microsoft empfiehlt, diesen Funktionaufruf als Ersatz für den INT 28 zu nutzen.

Unter WINDOWS 3.x kann ein Programm diesen Aufruf benutzen, um die Kontrolle wieder an das Betriebssystem zurückzugeben. Ein Aufruf blockiert dabei das Programm

nicht. Vielmehr wird der Rest der Zeitscheibe an das Betriebssystem zurückgegeben und kann anderen Prozessen zugeteilt werden.

5.13.9 WINDOWS 3.x Begin Critical Section (AX = 1681H)

Dieser Funktionsaufruf wird von WINDOWS 3.x unterstützt. Hauptaufgabe ist es, dem Betriebssystem mitzuteilen, daß der aktuelle Prozeß in eine kritische Phase eintritt und nicht unterbrochen werden kann. Hierbei gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 16H (WINDOWS)              º
º AL: 81H (Begin Critical Section)º
û--------------------------------À
º      RETURN                    º
Û--------------------------------ì
```

Nach dieser Funktion sollte so schnell als möglich die Funktion AX = 1602H abgesetzt werden, da sonst das Betriebssystem blockiert werden kann.

Die Funktion erhöht vermutlich das InDOS-Flag um 1 und wird erst ab WINDOWS 3.0 unterstützt.

5.13.10 WINDOWS 3.x End Critical Section (AX = 1682H)

Dieser Funktionsaufruf wird von WINDOWS 3.x unterstützt. Hauptaufgabe ist es, dem Betriebssystem mitzuteilen, daß der aktuelle Prozeß die kritische Phase beendet hat und wieder unterbrochen werden kann. Hierbei gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 16H (WINDOWS)              º
º AL: 82H (End Critical Section)  º
û--------------------------------À
º      RETURN                    º
Û--------------------------------ì
```

Mit dieser Funktion wird vermutlich das InDOS-Flag um 1 erniedrigt. Der Aufruf wird erst ab WINDOWS 3.0 unterstützt.

5.13.11 WINDOWS 3.x Get Current Virtual Maschine ID (AX = 1683H)

Dieser Funktionsaufruf wird von WINDOWS 3.x unterstützt. Hauptaufgabe ist es, die ID-Nummer der aktuellen virtuellen Maschine abzufragen. Hierbei gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 16H (WINDOWS)               º
º AL: 83H (Get VM ID)             º
û---------------------------------À
º      RETURN                     º
º BX: VM ID                       º
Û---------------------------------ì
```

WINDOWS selbst läuft in der VM 1. Eine VM 0 wird nicht zurückgegeben. Die Bedeutung der Nummern für die virtuellen Maschinen ist der folgenden Funktion zu entnehmen.

5.13.12 WINDOWS 3.x Get Device API Entry Point (AX = 1684H)

Dieser Funktionsaufruf wird von WINDOWS 3.x unterstützt. Hauptaufgabe ist es, die Einsprungpunkte der aktuellen virtuellen Maschine abzufragen. Hierbei gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 16H (WINDOWS)               º
º AL: 84H (Get VM Entry Point)    º
º BX: VM ID                       º
º ES:DI Entry Point               º
û---------------------------------À
º      RETURN                     º
Û---------------------------------ì
```

Nach dem Aufruf findet sich im Registerpaar ES:DI die Adresse der entsprechenden Routine für die Einheit. Der Wert 0000:0000 signalisiert, daß die betreffende Einheit kein API unterstützt. Nur einige der WINDOWS Enhanced Mode Virtual Devices unterstützen solche API's.

Für die Virtual Device ID (VxD) gelten folgende Codes:

```
Code VxD Einheit
 01H VMM         Virtual Machine Manager
 02H Debug
 03H VPICD       Virtual Prog. Interrupt Controller
                 Device
 04H VDMAD       Virtual DMA Device
 05H VTD         Virtual Timer Device
 06H V86MMGR     Virtual 8086 Mode Device
 07H PAGESWAP    Paging Device
 08H Parity
 09H Reboot
 0AH VDD         Virtual Display Device (Grabber)
 0BH VSD         Virtual Sound Device
 0CH VMD         Virtual Mouse Device
 0DH VKD         Virtual Keyboard Device
 0EH VCD         Virtual COMM Device
 0FH VPD         Virtual Printer Device
 10H VHD         Virtual Hard Disk Device
 11H VMCPD
 12H EBIOS       Erweitertes BIOS (PS/2)
 13H BIOSXLAT    Map ROM BIOS API (V86 -> Prot. Mode)
 14H VNETBIOS    Virtual NetBIOS Device
 15H DOSMMGR
 16H WINLOAD
 17H SHELL
 18H VMPoll
 19H VPROD
 1AH DOSNET
 1BH VFD        Virtual Floppy Device
 1CH VDD2       2. Display Adapter
 1DH WINDEBUG
 1EH TSRLOAD
```

Die VxD ID-Codes sind dabei nach dem folgenden Schema angelegt: Das oberste Bit ist reserviert. Die folgenden 10 Bit bilden die (von Microsoft vergebenen) OEM-Nummer. Lediglich die unteren 5 Bit erlauben 32 unterschiedliche Einheiten zu adressieren.

5.13.13 WINDOWS Switch VM and Callback (AX = 1685H)

Dieser Funktionsaufruf wird von WINDOWS unterstützt um einzelnen DOS-Einheiten den Aufruf von Funktionen in einzelnen VM-Einheiten zu ermöglichen. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 16H (WINDOWS)               º
º AL: 85H (Switch VM an Callback) º
º BX: VM ID                       º
º CX: Flags                       º
º DX:SI Priorität                 º
º ES:DI FAR Callback Proc.        º
û---------------------------------À
º      RETURN                     º
º CY: 1 -> Fehler                 º
º AX: Fehlercode                  º
º CF: 0 -> ok                     º
Û---------------------------------ì
```

Einige DOS-Einheiten, z.B. Netzwerktreiber, müssen Funktionen in spezifischen virtuellen Maschinen aufrufen. Dies erfolgt über diese Funktion den INT 2FH.

Beim Aufruf ist in BX die ID-Nummer der virtuellen Maschine zu übergeben, zu der geschaltet werden soll. In CX wird ein Flag mit folgender Kodierung übergeben:

```
CX : Bit 0 : Wait until Interrupts enabled
         1 : Wait until critical Section unowned
       2-15 : reserviert
```

Im Registerpaar DX:SI ist die Priorität (priority boost) zu übergeben. ES:DI enthält einen Zeiger auf die FAR Callback Prozedur.

Ist nach dem Aufruf das Carry-Flag gesetzt, liegt ein Fehler vor. AX enthält folgende Fehlercodes:

```
AX = 01H invalid VM ID
     02H invalid priority boost
     03H invalid Flags
```

Bei gelöschtem Carry-Flag wurde der Aufruf erfolgreich ausgeführt. Die Call-Back-Funktion muß alle Register sichern und mit IRET beendet werden.

5.13.14 DOS-DPMI Detect Mode (AX = 1686H)

Dieser Funktionsaufruf wird vom DOS-Protected-Mode-Interface (DPMI) zur Abfrage des Modes zur Verfügung gestellt. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 16H (DPMI)                  º
º AL: 86H (Detect Mode)           º
û---------------------------------À
º       RETURN                    º
º AX: Mode                        º
Û---------------------------------ì
```

Ein Programm kann die Funktion aufrufen. Nach dem Aufruf enthält das Register AX den Status des Prozessors:

```
AX = 0000H die CPU arbeitet im Protected Mode und
           die DPMI-Funktionen stehen unter dem
           INT 31 zur Verfügung.
   <> 0    die CPU befindet sich im REAL- oder im
           V86-Mode, b.z.w. der INT 31 ist nicht
           verfügbar.
```

Ergänzend ist die nachfolgend beschriebene Funktion AX = 1687H zur Abfrage des DPMI-Status zu nutzen.

5.13.15 DOS-DPMI Installation Check (AX = 1687H)

Dieser Funktionsaufruf wird vom DOS-Protected-Mode-Interface (DPMI) zur Abfrage des Installationstatus zur Verfügung gestellt. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 16H (DPMI)                 º
º AL: 87H (Installation Check)   º
û--------------------------------À
º      RETURN                    º
º AX: Status                     º
º BX: Flags                      º
º CL: Prozessor                  º
º DH: DPMI-Hauptversion          º
º DL: DPMI-Unterversion          º
º SI: Datengröße                 º
º ES:DI DPMI Entry Point         º
Û--------------------------------ì
```

Ein Programm kann über die Funktion prüfen, ob der DPMI-Treiber installiert wurde. Nur falls AX = 0000H zurückgegeben wird, ist das DPMI aktiv. Dann enthalten die restlichen Register zusätzliche Informationen.

In BX wird ein Flag zurückgegeben, welches signalisiert, ob das DPMI 32-Bit-Mode Programme unterstützt.

```
BX: Bit 0: 1 = 32-Bit-Mode Programme werden unterstützt
```

Das Register CL enthält einen Wert, der den Typ des Prozessors decodiert:

```
CL: 02H = 80286
    03H = 80386
    04H = 80486
```

Das Register DX enthält die Versionsnummer des DPMI-Treibers. Zur Zeit existieren nur die Versionen 0.9 und 1.0. In DL wird die Unterversion mit 2 Ziffern angegeben. SI enthält die Zahl der Paragraphen (16-Byte-Blöcke), die der DOS-Extender als privater Datenbereich reserviert hat. Im Registerpaar ES:DI wird die Einsprungadresse für den DPMI Mode-Switch übergeben.

Mit einem Aufruf dieser Routine läßt sich der Prozessor in den Protected Mode schalten. Hierfür gilt folgende Aufrufschnittstelle:

```
Ö-----------------------------------------------------Ì
º  AX = Flags                                          º
º         Bit 0 : 1 -> 32-Bit-Programm                 º
º  ES = Real Mode Segment in dem der Puffer für die     º
º       privaten Daten des DPMI abgelegt werden. Wird  º
º       ignoriert, falls SI = 0 war.                   º
º                                                      º
º RETURN                                               º
º                                                      º
º CY : 1 -> Error, Programm noch im Real-Mode          º
º  AX: Fehlercode (DPMI 1.0)                           º
º      8011H Zuweisung der Descriptoren nicht möglich  º
º      8021H 32-Bit-Programm angegeben, aber der DPMI- º
º      Server unterstützt nur 16 Bit                   º
º CY : 0 -> ok, Programm im Protected Mode             º
º   CS = 16-Bit-Selector des Real-Mode CS              º
º   SS = Selector des Real-Mode SS                     º
º   DS = Selector des Real-Mode DS                     º
º   ES = Selector zum PSP (100H Byte Limit)            º
º   FS = GS = 0, ESP High Word = 0 -> 32-Bit-Programm  º
Û-----------------------------------------------------ì
```

Nach einem erfolgreichen Aufruf ist die CPU im Protected Mode. Der Aufruf ist jeweils nur einmal erforderlich.

5.13.16 DOS-DPMI Get Vendor specific API Entry Point (AX = 168AH)

Dieser Funktionsaufruf wird vom DOS-Protected-Mode-Interface (DPMI) ab der Version 0.9 unterstützt und ist nur im Protected Mode verfügbar. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
°      CALL:  INT 2F             °
°                                °
° AH: 16H (DPMI)                 °
° AL: 8AH (Get Entry Point)      °
° DS:ESI Adresse Name            °
û--------------------------------À
°      RETURN                    °
° AL: Status                     °
°   00H -> OK                    °
°   DS:ESI Entry Point           °
°   8AH -> Fehler                °
Û--------------------------------ì
```

Beim Aufruf ist in DS:SI (oder im 32-Bit-Mode in ESI) die Adresse auf den ASCIIZ-String mit dem Namen des Herstellers anzugeben. Nach dem Aufruf findet sich in AL der Statuscode. Mit dem Wert 8AH wird ein Fehler beim Aufruf signalisiert.

Mit AL = 0 war der Aufruf erfolgreich und in ES:DI (oder EDI im 32-Bit-Mode) findet sich die Adresse des Einsprungpunktes zu den erweiterten API-Routinen. Über den Namen läßt sich der Einsprungpunkt selektieren. Dieser Aufruf ist ab DPMI 0.9 vorhanden, wurde aber nicht dokumentiert, da herstellerspezifische Erweiterungen des DPMI-Handlers nicht offengelegt werden. Eine ausführliche Beschreibung der DPMI-Schnittstelle findet sich im Kapitel über die DOS-Speicherverwaltung.

5.14 WINDOWS WinOldAp-Schnittstelle (AH=17H)

Diese Gruppe von Funktionen wurde unter Windows 3.0 eingeführt um Applikationen zu unterstützen, die nicht für die Version 3.0 geschrieben wurden. WinOldAp unterstützt alte zeichenorientierte Applikationen, die DDE und Clipboard benutzen. Es existieren verschiedene Subfunktionen, die nachfolgend beschrieben werden.

5.14.1 WinOldAp Identify WinOldAp Version (AX = 1700H)

Mit diesem Aufruf läßt sich prüfen, ob die Funktion unter dem INT 2F zur Verfügung steht. Die Schnittstelle entspricht jedoch nicht den Konventionen des INT 2F.

```
Ö--------------------------------Ì
°      CALL:  INT 2F             °
°                                °
° AH: 17H (WinOldAp)             °
° AL: 00H (Get Version)          °
û--------------------------------À
°      RETURN                    °
° AX: Status                     °
Û--------------------------------ì
```

Das Register AX enthält nach dem Aufruf den Status. Wird als Wert AX = 1700H zurückgegeben, steht WinOldAP nicht zur Verfügung.

Enthält AX einen Wert ungleich 1700H ist dieser als Versionsnummer zu interpretieren:

```
AL: WinOldAP Hauptversion (z.B. 02)
AH: WinOldAP Unterversion (z.B. 00)
```

Falls diese Funktion vorhanden ist, lassen sich auch die folgenden Subfunktionen aus einer Applikation heraus aufrufen.

5.14.2 WinOldAp Open Clipboard (AX = 1701H)

Mit diesem Aufruf läßt das Clipboard öffnen.

```
Ö----------------------------------Ì
º      CALL:  INT 2F               º
º                                  º
º AH: 17H (WinOldAp)               º
º AL: 01H (Open Clipboard)         º
û----------------------------------À
º      RETURN                      º
º AX: Status                       º
Û----------------------------------ì
```

Das Register AX enthält nach dem Aufruf den Status. Wird als Wert AX = 0000H zurückgegeben, ist das Clipboard bereits offen. Alle Werte ungleich 0 signalisieren, daß der Aufruf erfolgreich war.

5.14.3 WinOldAp Empty Clipboard (AX = 1702H)

Mit diesem Aufruf läßt der Inhalt des Clipboards leeren.

```
Ö----------------------------------Ì
º      CALL:  INT 2F               º
º                                  º
º AH: 17H (WinOldAp)               º
º AL: 02H (Empty  Clipboard)       º
û----------------------------------À
º      RETURN                      º
º AX: Status                       º
Û----------------------------------ì
```

Das Register AX enthält nach dem Aufruf den Status. Wird als Wert AX = 0000H zurückgegeben, ist ein Fehler aufgetreten und das Clipboard wurde nicht gelöscht. Alle Werte ungleich 0 signalisieren, daß der Aufruf erfolgreich war und die Daten im Clipboard gelöscht wurden.

5.14.4 WinOldAp Set Clipboard Data (AX = 1703H)

Diese Funktion übergibt Daten von der Applikation an das Clipboard. Es gilt folgende Schnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 17H (WinOldAp)             º
º AL: 03H (Empty  Clipboard)     º
º DX: Format                     º
º ES:BX Datenpuffer              º
º SI:CX Datengröße               º
û--------------------------------À
º      RETURN                    º
º AX: Status                     º
Û--------------------------------ì
```

Im Register DX ist ein Wert zwischen 1 und 7 einzugeben, der die Clipboard-Kodierung der Daten spezifiziert. Für diesen Wert gilt:

```
Kodierung des Clipboard-Formates

DX = 01H: Text
     02H: Bitmap
     03H: Metafile Picture
     04H: SYLK
     05H: DIF
     06H: TIFF
     07H: OEM-Text
```

Bei diesen Formaten handelt es sich um die Standard Clipboard-Formate von Windows. Der Aufbau der Formate ist in /6,7/ beschrieben.

Die Daten müssen von der Applikation in der gesetzten Kodierung in einem Puffer abgelegt werden. Die Adresse des Datenbereiches ist im Registerpaar ES:BX beim Aufruf zu übergeben. Das Registerpaar SI:CX spezifiziert dabei die Länge des Datenbereiches in Byte.

Das Register AX enthält nach dem Aufruf den Status. Wird als Wert AX = 0000H zurückgegeben, ist ein Fehler aufgetreten und die Daten wurden nicht übertragen. Bei einem Wert von AX <> 0 wurden die Daten aus dem Puffer in das Clipboard kopiert.

5.14.5 WinOldAp Get Clipboard Data Size (AX = 1704H)

Mit dieser Funktion kann eine Applikation ermitteln, ob Daten im Clipboard vorliegen.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AH: 17H (WinOldAp)             º
º AL: 04H (Data Size)            º
º DX: Clipboard Format           º
û--------------------------------À
º      RETURN                    º
º DX:AX Datengröße               º
Û--------------------------------ì
```

Im Register DX ist ein Wert zwischen 1 und 7 einzugeben, der die Clipboard-Kodierung der Daten spezifiziert. Für diesen Wert gilt die bei AX = 1703H besprochene Kodierung.

Im Registerpaar DX:AX findet sich nach dem Aufruf die Länge des Datenbereiches des Clipboards in Byte. Diese Länge umfaßt sowohl den Header als auch die Nutzdaten. Ist DX:AX = 0, dann liegen keine Daten im angegebenen Format im Clipboard vor.

5.14.6 WinOldAp Get Clipboard Data (AX = 1705H)

Mit dieser Funktion kann eine Applikation Daten aus dem Clipboard abrufen.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 17H (WinOldAp)              º
º AL: 05H (Get Data)              º
º DX: Clipboard Format            º
º ES:BX Datenpuffer               º
û---------------------------------À
º      RETURN                     º
º AX: Status                      º
Û---------------------------------ì
```

Im Register DX ist ein Wert zwischen 1 und 7 einzugeben, der die Clipboard-Kodierung der Daten spezifiziert. Für diesen Wert gilt die bei AX = 1703H besprochene Kodierung.

Im Registerpaar ES:BX findet sich die Adresse des Puffers, in den die Funktion die Daten des Clipboards kopieren soll. Dieser Puffer muß vom Anwenderprogramm angelegt werden. Die Puffergröße läßt sich mit der Funktion 1704H ermitteln.

In AX wird nach dem Aufruf der Statuscode zurückgegeben:

```
AX = 0: Fehler beim Aufruf, keine Daten
   <>0: Aufruf erfolgreich
```

Ist der Wert von AX ungleich 0, dann liegen im Puffer die Daten in der Formatierung des Clipboards vor.

5.14.7 WinOldAp Close Clipboard (AX = 1708H)

Mit dieser Funktion kann eine Applikation das Clipboard schließen.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 17H (WinOldAp)              º
º AL: 08H (Close Clipboard)       º
û---------------------------------À
º      RETURN                     º
º AX: Status                      º
Û---------------------------------ì
```

In AX wird nach dem Aufruf der Statuscode zurückgegeben:

```
AX = 0: Fehler beim Aufruf, keine Daten
   <>0: Clipboard geschlossen
```

Ist der Wert von AX ungleich 0, dann wurde das Clipboard geschlossen.

5.14.8 WinOldAp Compact Clipboard (AX = 1709H)

Mit dieser Funktion kann eine Applikation Daten aus dem Clipboard verdichten.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 17H (WinOldAp)              º
º AL: 09H (Compact Data)          º
º SI:CX Datenlänge                º
û---------------------------------À
º      RETURN                     º
º DX:AX Länge                     º
Û---------------------------------ì
```

Im Register SI:CX ist anzugeben, auf welche Größe das Clipboard zu verkleinern ist. Nach dem Aufruf findet sich im Registerpaar DX:AX die Zahl der Bytes des größten freien Speicherblocks.

5.14.9 WinOldAp Get Device Capabilities (AX = 170AH)

Mit dieser Funktion kann eine Applikation Daten über das Graphic Device Interface (GDI) abfragen.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 17H (WinOldAp)              º
º AL: 0AH (Get Info)              º
º DX: GDI-Index                   º
û---------------------------------À
º      RETURN                     º
º AX: Flag                        º
Û---------------------------------ì
```

Im Register DX ist ein Index für die gewünschte GDI-Information zu übergeben. Hierfüg gilt folgende Kodierung:

```
DX = GDI-Index
 00H Device Driver Version
 02H Device Classification
 04H Width in mm
 06H Height in mm
 08H Width in Pixel
 0AH Height in Pixel
 0CH Bits per Pixel
 0EH Number of Bitplanes
 10H Number of Brushes supported by Device
 12H Number of Pens supported by Device
 14H Number of Markers supported by Device
 16H Number of Fonts supported by Device
 18H Number of Colors
 1AH Size required for device descriptor
 1CH Curve Capabilities
 1EH Line Capabilities
 20H Polygon Capabilities
 22H Text Capabilities
 24H Clipping Capabilities
 26H BitBld Capabilities
 28H X Aspect
 2AH Y Aspect
 2CH Length of Hypotenouse of Aspect
 58H Logical Pixel per Inch of Widht
 5AH Logical Pixel per Inch of Height
```

Über den GDI-Index ermittelt die Funktion dann die Möglichkeiten und gibt den betreffenden Wert im Register AX zurück. Der Wert in AX wird dabei teilweise als Bitflag interpretiert. Es gilt folgende Kodierung:

```
AX bei Device Classification
   00H Vector Plotter
   01H Raster Display
   02H Raster Printer
   03H Raster Kamera
   04H Character Stream
   05H Metafile (VDM)
   06H Display File
AX bei Curve
  Bit 0: Circles
      1: Pie
      2: Arcs
      3: Ellipses
      4: Wide Lines
      5: Styled Lines
      6: Wide Styled Lines
      7: Interiors
AX bei Line
  Bit 1: Polylines
      2: Markers
      3: Polymarkers
      4: Wide Lines
      5: Styled Lines
      6: Wide Styled Lines
      7: Interiors
AX bei Polygon
  Bit 0: Polygons
      1: Rectangles
      2: Trapezoides
      3: Scanlines
      4: Wide Boarders
      5: Styled Boarders
      6: Wide Styled Boarders
      7: Interiors
AX bei Text
  Bit 0: Output Precision Character
      1: Output Precision Stroke
      2: Clipping Precision Stroke
      3: 90-Degree Char. Rotation
      4: Arbitrary Char. Rotation
      5: Indep. X and Y Scaling
      6: Double Size
      7: Integer Scaling
      8: Continouous Scaling
      9: Bold
     10: Italic
     11: Underline
     12: Strikeout
     13: Raster Font
     14: Vector Font
     15: reserviert
AX bei Clipping
     00H: kein Clipping
     01H: Clipping to Rectangle
AX bei Raster
  Bit 0: Simple BitBLT
      1: Device Requires Banding Support
      2: Device Requires Scaling Support
      3: Supports Bitmap > 64 Kbyte
```

Damit kann die Applikation auf die entsprechenden Möglichkeiten der Einheit reagieren.

5.15 Kommunikation with SHELL (AH = 19H)

Diese Funktion ist nur in DOS 4.X vorhanden und dient der Kommunikation mit der DOS-Shell. Es sind folgende Aufrufe bekannt:

5.15.1 Get Installation Status (AL = 00H)

Mit dieser Unterfunktion läßt sich feststellen, ob die Funktion SHELLB.COM resident geladen ist.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                 º
º AH: 19H (SHELLB.COM)           º
º AL: 00H Get Installationsstatus º
û--------------------------------À
º      RETURN                    º
º AL: Statusbyte                 º
Û--------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Programmes.

- Falls AL = 0 ist, dann ist der residente Teil von SHELLB nicht vorhanden und kann installiert werden.
- Falls AL = FFH ist, dann ist der residente Teil von SHELLB installiert.

SHELLB.COM ist ein Bestandteil der DOS 4.x Shell.

5.15.2 SHELLB.COM - SHELLC.EXE Interface (AL = 01H)

Auch diese Unterfunktion ist nur in DOS 4.x vorhanden. Es gilt folgende Aufrufschnittstelle.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                 º
º AH: 19H (SHELLB.COM)           º
º AL: 01H SHELLB-SHELLC Interface º
º BL: 00H falls SHELLC transient  º
º     01H falls SHELLC resident   º
º DS:DX FAR CALL Adresse SHELLC   º
û--------------------------------À
º      RETURN                    º
º ES:DI Arbeitsbereich           º
Û--------------------------------ì
```

SHELLC.EXE läßt sich resident oder transient laden. Im Register BL ist ein entsprechender Code zu übergeben, der zwischen den beiden Varianten unterscheidet. Im Registerpaar DS:DX erwartet die Unterfunktion die Adresse des Eintrittspunktes (als FAR CALL) für das Programm SHELLC.EXE. Nach dem Aufruf enthält das Registerpaar ES:DI einen Zeiger auf den gemeinsamen Arbeitsbereich zwischen SHELLB.COM und SHELLC.EXE.

5.15.3 SHELLB.COM - COMMAND.COM Interface (AL = 02H)

Es gilt folgende Aufrufschnittstelle.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                 º
º AH: 19H (SHELLB.COM)           º
º AL: 02H SHELLB-COMMAND Interfaceº
º ES:DI ASCIIZ-Batchfilename     º
º DS:DX Buffer für Ergebnisse    º
û--------------------------------À
º      RETURN                    º
º AL = 00 Fehler                 º
º      FF ok                     º
Û--------------------------------ì
```

SHELLC ist die neue Benutzeroberfläche von DOS 4.x, die über SHELLB mit dem Kommandoprozessor COMMAND.COM kommuniziert. SHELLC ersetzt dann den Teil

des Userinterfaces von COMMAND.COM. Zur Aktivierung von SHELLC legt COMMAND.COM intern eine Pseudo-Batchdatei an, die die entsprechenden Befehle enthält.

Diese Unterfunktion wird dann von COMMAND.COM aufgerufen, um mit SHELLB zu kommunizieren. In ES:DI ist der volle Name des eröffneten aktuellen Batchfiles als ASCIIZ-String zu übergeben. Das Registerpaar DS:DX enthält einen Zeiger auf einen Puffer, in dem SHELLB Programm-Start-Kommandos (PSC) zurückgeben kann. COMMAND.COM liest diese Kommandos (PSC) und führt diese wie Befehle aus einer Batchdatei aus. Nach Ausführung eines Kommandos wird solange in der aktuellen Zeile der Batchdatei gewartet, bis SHELLB keine Kommandos (PSC) mehr im SHELLB-Arbeitsbereich zurückgibt. Die Kommandos (PSC) werden von SHELLC (dem Benutzerinterface) in den Arbeitsbereich von SHELLB geschrieben. Der Befehl GOTO COMMON schließt eine Kommandosequenz (PSC) ab. Dies führt dann dazu, daß COMMAND.COM wieder an den Anfang des Batchfiles geht und mit der Abarbeitung des Befehls beginnt. Da hier aber ein Aufruf von SHELLC steht, wird die SHELL erneut aktiviert. Die Prüfung des Batchfilenamens erlaubt den PSC's den Aufruf (CALL) von geschachtelten Batchdateien, so daß noch nicht ausgeführte PSC's für eine spätere Bearbeitung gestapelt werden.

Nach dem Aufruf gibt der Wert in AL Hinweise, ob der Aufruf korrekt ausgeführt wurde. Ist AL = 00, dann trat ein Fehler bei der Auswertung der Übergabeparameter auf. Dies ist zum Beispiel der Fall, wenn der in ES:DI übergebene Filename nicht mit der von SHELLB erwarteten Schreibweise übereinstimmt. Der Wert 00H wird auch zurückgegeben, falls keine Programm-Start-Commands mehr vorhanden sind.

Bei erfolgreichem Aufruf findet sich in AL der Wert FFH. In dem durch das Registerpaar DS:DX adressierten Puffer finden sich dann die Programm-Start-Commands (PSC's). Der Puffer besitzt folgende Struktur:

Adr	Byte	Feld
DX+1	1	Zahl der Bytes des folgenden PSC
DX+2	n	Programm Start Command Text mit 0DH abgeschlossen

Von der DOS-SHELLC werden also die DOS-Kommandos in den Arbeitsbereich von SHELLB abgelegt. Diese PSC's haben das gleiche Textformat wie Anweisungen in Batchdateien. COMMAND.COM liest über den obigen Aufruf diese Kommandos und behandelt diese wie Batchdateien.

5.15.4 SHELLB.COM - COMMAND.COM Interface II (AL = 03H)

Es gilt folgende Aufrufschnittstelle.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 19H (SHELLB.COM)            º
º AL: 03H SHELLB-COMMAND Interfaceº
º ES:DI ASCIIZ-Batchfilename      º
û---------------------------------À
º      RETURN                     º
º AL = 00 Fehler                  º
º      FF ok                      º
Û---------------------------------ì
```

Bei AL = 00 stimmt der durch ES:DI angegebene Batchfilename nicht mit dem letzten in SHELLB gespeicherten Parameter überein.

5.15.5 SHELLB.COM - SHELLB.COM transient TSR Interface (AL = 04H)

Diese Funktion erlaubt die Kommunikation zwischen dem transienten und residenten Teil der SHELLB. Es gilt folgende Aufrufschnittstelle.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 19H (SHELLB.COM)            º
º AL: 04H SHELLB-TSR Interface    º
û---------------------------------À
º      RETURN                     º
º ES:DI ASCIIZ-Batchfilename      º
Û---------------------------------ì
```

Im Register ES:DI wird der Name des aktuellen SHELL-Batchfiles zurückgegeben. Dabei gilt folgendes Format:

```
1  WORD  Zahl der Folgebytes
n  BYTEs Name des Shell Batchfiles in Großbuchstaben
```

Der Name darf maximal 8 Zeichen lang sein.

5.16 Kommunikation with DOS-ANSI.SYS (AH = 1AH)

Mit diesem Aufruf läßt sich ab DOS 4.0 mit dem Programm ANSI.SYS kommunizieren.

5.16.1 Get Installation Status (AX = 1A00H)

Der Aufruf wurde ab DOS 5.0 offiziell dokumentiert. Die Unterfunktion erlaubt die Prüfung, ob ANSI.SYS installiert ist.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 1AH (ANSI.SYS)              º
º AL: 00H Get Installationsstatus º
û---------------------------------À
º      RETURN                     º
º AL: Statusbyte                  º
Û---------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Programmes.

- Falls AL = 0 ist, dann ist ANSI.SYS nicht vorhanden und kann installiert werden.
- Falls AL = FFH ist, dann ist ANSI.SYS installiert.

5.16.2 DOS 4.x ANSI.SYS Get/Set Display Informations (AX = 1A01H)

Mit dieser Unterfunktion werden die Display-Informationen des INT 21-Aufrufes 440CH mit CX = 037FH und 035FH gesetzt oder gelesen. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 1AH (ANSI.SYS)              º
º AL: 01H Get/Set Display Info    º
º CL: 7FH für Get Info            º
º     5FH für Set Info            º
º DS:DX Aresse Parameterblock     º
û---------------------------------À
º      RETURN                     º
º CF: 1 Fehler                    º
º AX: Fehlercodes                 º
Û---------------------------------ì
```

Die Unterfunktion bildet das Interface zwischen dem DOS-INT 21-Aufruf 44H (IOCTL) und dem ANSI.SYS-Treiber. Die Register DS:DX enthalten die Adresse eines Parameterblocks mit den Steuerzeichen. Dieser wird beim INT 21, Funktion 440CH aufgebaut.

5.16.3 DOS 4.x ANSI.SYS-Misc Requests (AX = 1A02H)

Der Aufruf ist erst ab DOS 4.0 vorhanden und es gelten folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 1AH (ANSI.SYS)              º
º AL: 02H Misc Requests           º
º DS:DX Adresse Parameterblock    º
û---------------------------------À
º      RETURN                     º
º --                              º
Û---------------------------------ì
```

Die Register DS:DX enthalten die Adresse eines Parameterblocks mit den Steuerzeichen. Dieser Parameterblock besitzt folgenden Aufbau:

```
Ö-------Ú-----------------------------------------Ì
º  Offs º  Funktion                                 º
û-------é-----------------------------------------À
º   00H º 00H Set / Reset Interlock                 º
º       º 01H Get /L Flag                         º
º   01H º Interlock Status                        º
º       º 00H: Reset  01H: Set                    º
º   02H º 00: /L aktiv  01: /L aus                 º
Û-------Ù-----------------------------------------
```

Der Set-Interlock-Befehl verhindert, daß ANSI.SYS die Ausgaben über den INT 10 nicht mehr bearbeitet. Vor dem Aufruf muß das 1. Byte mit dem Code 00H oder 01H besetzt werden. Die beiden folgenden Bytes werden dann von ANSI.SYS zurückgegeben. DOS 5.0 verzweigt zum vorhergehenden Treiber der Kette, falls AL > 02H beim Aufruf ist.

5.17 DOS 4.0 Kommunikation with XMA2EMS.SYS (AH = 1BH)

Dies ist das Interface zum DOS-4.x-EMS-Treiber, der bestimmte Funktionen unterstützt.

5.17.1 Get Installation Status (AL = 00H)

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AH: 1BH (XMA2EMS.SYS)           º
º AL: 00H Get Installationsstatus º
û---------------------------------À
º            RETURN               º
º AL:  Statusbyte                 º
Û---------------------------------ì
```

Die Unterfunktion erlaubt die Prüfung, ob XMA2EMS.SYS installiert ist.

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Programmes.

- Falls AL = 0 ist, dann ist der Treiber nicht vorhanden und kann installiert werden.
- Falls AL = FFH ist, dann ist der Treiber installiert.

Der XMA2EMS.SYS wird nur installiert, falls das System mit EMS ausgestattet ist und DOS freie *page frames* im EMS erhält.

5.17.2 XMA2EMS.SYS Get hidden frame Info (AL = 01H)

Dieser Aufruf ist ab DOS 4.0 vorhanden.

```
Ö---------------------------------Ì
º      CALL: INT 2F               º
º                                 º
º AH: 1BH (XMA2EMS.SYS)           º
º AL: 01H Get hidden frame Info   º
º DI: hidden page number          º
û---------------------------------À
º      RETURN                     º
º AX: FFFFH no hidden page        º
º     0000H ok                    º
º ES: Segment page frame          º
º DI: physikal. page number       º
Û---------------------------------ì
```

Diese Funktion liest Daten über den INT 67, Funktion 58H. Nach dem Aufruf zeigt der Inhalt des Registers AX, ob eine *hidden page frame* vorhanden ist. In ES steht die Segmentadresse, in der das EMS-Fenster eingeblendet wird. DI enthält die physikalische Seitennummer.

5.18 OS/2 Compatibility-Box (AH = 40H)

Diese Funktion wird von der OS/2-Kompatibilitästsbox benutzt um DOS in den Vorder- und Hintergrund zu stellen.

5.18.1 Switch DOS to Background (AL = 01H)

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 40H (OS/2)                  º
º AL: 01H Switch DOS to Backgroundº
û---------------------------------À
º       RETURN                    º
º ---                             º
Û---------------------------------ì
```

Dieser Aufruf wird von OS/2 verwendet um die DOS-Kompatibilitätsbox in den Hintergrund zu schalten. Der Aufruf steht in DOS nicht zur Verfügung.

5.18.2 Switch DOS to Foreground (AL = 02H)

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 40H (OS/2)                  º
º AL: 02H Switch DOS to Foregroundº
û---------------------------------À
º       RETURN                    º
º ---                             º
Û---------------------------------ì
```

Dieser Aufruf wird von OS/2 verwendet um die DOS-Kompatibilitätsbox in den Vordergrund zu schalten. Der Aufruf steht in DOS nicht zur Verfügung.

Der Funktionsaufrufe AX = 41xxH und AX = 42xxH werden vom Microsoft LAN-Manager 2.0 benutzt. Die Aufrufschnittstelle ist aber nicht genau bekannt.

5.19 Kommunikation with XMS (AH = 43H)

Ab DOS 4.01 unterstützt Microsoft das Extended Memory mit einem neuen Treiber. Die genaue Funktionsweise ist im Kapitel über die Speichererweiterungen (Expanded Memory, Extended Memory) zu finden. Über den Interrupt 2FH läßt sich der XMS-Treiber ansprechen.

5.19.1 XMS Get Installation Status (AL = 00H)

Mit diesem Aufruf läßt sich prüfen, ob der Treiber installiert ist.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 43H (XMS)                   º
º AL: 00H Get Installation Status º
û---------------------------------À
º      RETURN                     º
º AL: 80H Driver installed        º
Û---------------------------------ì
```

Der XMS-Treiber hält sich nicht an die üblichen Konventionen des Multiplexer Interrupts. Wird nach dem Aufruf der Wert 80H zurückgegeben, dann ist der Treiber installiert. Alle anderen Werte sind so zu interpretieren, daß der Treiber nicht vorhanden ist.

5.19.2 XMS Get Driver Address (AL = 10H)

Mit diesem Aufruf läßt sich die Adresse des Funktionsdispatchers ermitteln. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
°       CALL:  INT 2F             °
°                                 °
° AH: 43H (XMS)                   °
° AL: Get Dispatcher Address      °
û---------------------------------À
°       RETURN                    °
° ES:BX Adresse                   °
Û---------------------------------ì
```

Im Register AL ist der Unterfunktionscode 10H zu übergeben. Nach dem Aufruf findet sich im Registerpaar ES:BX die Adresse des XMS-Dispatchers. Die Funktionen des XMS-Treibers lassen sich dann über FAR CALL's zu dieser Adresse ansprechen. Eine Beschreibung der Funktionen und deren Schnittstellen ist im Kapitel über die Speichererweiterung zu finden.

5.20 DOS 5.0 Kommunikation (AH = 46H)

Ab der Version 5.0 des Betriebssystems wird die Funktion AX=46xxH durch den DOS-Kern und die Shell belegt.

Der DOS-Kern benutzt die Subfunktionen AX=4601H und 4602H, die Belegung und Bedeutung ist aber bisher nicht genau bekannt.

5.20.1 WINDOWS 3.0 Installation Check (AX = 4680H)

Diese Subfunktion ist nicht offiziell dokumentiert, wird aber zukünftig durch Microsoft unterstützt.

```
Ö---------------------------------Ì
°       CALL:  INT 2F             °
°                                 °
° AH: 46H (WINDOWS 3.x)           °
° AL: 80H                         °
û---------------------------------À
°       RETURN                    °
° AX: Status                      °
Û---------------------------------ì
```

Nach dem Aufruf findet sich im Register AX der Status, der folgende Kodierung aufweist:

- AX = 0000H WINDOWS 3.0 arbeitet im Real- (/R) oder Standardmode (/S).
- AX <> 0000H WINDOWS 3.0 ist nicht installiert, oder es arbeitet im Enhanced Mode /3), oder es ist eine Version kleiner 3.0 installiert.

Der Funktionsaufruf wird unter anderem von der DOS 5.0/6.0-Shell aktiviert.

5.21 DOSKEY Funktionen (AX = 48, DOS 5.0-6.0)

Diese Funktionen stehen erst ab DOS 5.0 zur Verfügung und werden durch das Programm DOSKEY benutzt.

5.21.1 DOSKEY Installation Check (AX = 4800H)

Mit diesem Aufruf prüft DOSKEY, ob der residente Teil bereits installiert wurde.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 48H (DOSKEY)                º
º AL: 00H (Installation Check)    º
û---------------------------------À
º       RETURN                    º
º AX: Status                      º
Û---------------------------------ì
```

Nach dem Aufruf findet sich im Register AX der Status des Programmes. Der Wert AX <> 0 signalisiert, daß der residente Teil von DOSKEY bereits installiert ist.

5.21.2 DOSKEY Read Input Line from Console (AX = 4810H)

Über diese Funktion läßt sich der Kommandozeileneditor von DOSKEY aufrufen. Die Funktion wird ab DOS 5.0 durch COMMAND.COM benutzt.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 48H (DOSKEY)                º
º AL: 10H (Read Input Line)       º
º DS:DX Adresse Zeilenpuffer      º
û---------------------------------À
º       RETURN                    º
º AX: Status                      º
Û---------------------------------ì
```

Die Funktion erwartet in DS:DX die Adresse eines Puffers, der genau die Struktur wie beim Aufruf der INT 21-Funktion 0AH haben muß:

```
Ö------Ú-----Ú----------------------------Ì
ºOffsetº Byteº Bedeutung                  º
û------é-----é----------------------------À
º 00   º 1   º Länge (muß immer 80H sein) º
º 01   º 1   º Zahl der gelesenen Byte    º
º 02   º n   º Datenbytes                 º
Û------Ù-----Ù----------------------------ì
```

Wichtig ist, daß die Länge des Puffers auf 80H gesetzt wird, da sonst DOSKEY zur nächsten Funktion in der Kette des INT 2F verzweigt. Ist nach dem Aufruf der Wert AX = 0000H vorhanden, dann wurde die Funktion erfolgreich ausgeführt. Wird vom Benutzer ein Macroname eingegeben, legt die Funktion keinen Text im Puffer ab. Dann muß die Funktion ein weiteres Mal aktiviert werden, um den expandierten Macrotext abzufragen.

5.22 DOS 5.0/6.0 HMA-SPACE (AH = 4AH)

Ab DOS 5.0 wird der HMA-Bereich durch das Betriebssystem verwaltet. Die Funktion AX = 4AxxH des INT 2F dient hier zur Kommunikation mit dem HMA-Treiber.

5.22.1 DOS 5.0/6.0 Query Free HMA-Space (AX = 4A01H)

Mit dieser Unterfunktion läßt sich prüfen, ob DOS HMA-Speicher benutzt. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 4AH (DOS 5.0/6.0)           º
º AL: 01H (Query Free HMA-Space)  º
û---------------------------------À
º       RETURN                    º
º BX: freier HMA-Speicher         º
º ES:DI Anfang HMA-Bereich        º
Û---------------------------------ì
```

Die Funktion gibt in ES:DI die Anfangsadresse des HMA-Bereiches an. Steht in diesem Registerpaar der Wert FFFF:FFFFH, dann wird kein HMA-Speicher benutzt. Das Register BX enthält nach dem Aufruf die Zahl der freien Byte im HMA-Bereich. Der Wert 0000H signalisiert, daß der HMA-Bereich unbenutzt ist.

5.22.2 DOS 5.0/6.0 Allocate HMA-Space (AX = 4A02H)

Mit dieser Unterfunktion läßt sich unter DOS ein freier HMA-Speicherbereich reservieren. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 4AH (DOS 5.0/6.0)           º
º AL: 02H (Allocate HMA-Space)    º
º BX: Zahl der Bytes              º
û---------------------------------À
º       RETURN                    º
º ES:DI Anfang HMA-Block          º
Û---------------------------------ì
```

Die Funktion erwartet in BX die Größe des zu reservierenden Blockes in Byte. Nach dem Aufruf ist der Inhalt des Registers BX zerstört. In ES:DI steht nach dem Aufruf die Anfangsadresse des reservierten Speicherblockes im HMA-Bereich. Der Aufruf dieser Subfunktion ist nur gültig, falls Teile des DOS-Kerns im HMA-Bereich geladen sind.

Normale Anwenderprogramme sollten diesen Aufruf zur Zuweisung von HMA-Speicher nicht benutzen. Hierfür stehen die Funktionen des XMA-Managers (siehe entsprechendes Kapitel) zur Verfügung.

Die Unterfunktion AX = 4A05H ist vermutlich durch die DOSSHELL belegt. Genaue Informationen über die Aufrufschnittstelle liegen allerdings nicht vor.

5.23 DOS 5.0/6.0 TaskSwitcher Interface (AH = 4BH)

Ab DOS 5.0 wird die Unterfunktion 4BH des INT 2F durch den TaskSwitcher belegt. Unter diesem Funktionscode wird ein API-Interface (API = Application Programm Interface) für zukünftige DOS- und WINDOWS-Versionen definiert. Bestimmte Programme (z.B. 3270 Emulatoren, Netzwerktreiber, etc.) dürfen nicht ohne weiteres durch den TaskSwitcher unterbrochen werden. Bei einer 3270 Emulation geht zum Beispiel bei einer Unterbrechung die Verbindung zum Hostrechner verloren. Die Funktionen des INT 2F mit dem Code 4BH ermöglichen nun eine Kommunikation zwischen TaskSwitcher und den aktiven Tasks. Bei Bedarf kann dann ein Programm, welches die Konventionen des API einhält, einen Taskwechsel verhindern oder verzögern. Laut Microsoft sollen zukünftige Versionen von WINDOWS dieses API im Real- und Standardmode unterstützen. WINDOWS 3.0 besitzt dagegegen im 386-Mode eine eigene API für die Prozeßkoordination.

Zum Verständnis der nachfolgenden Beschreibung möchte ich kurz die Abläufe beim Taskwechsel skizzieren. Initiiert der Benutzer über die Tastatur einen Taskwechsel, löst der Switcher über den INT 2F eine globale Botschaft zur Taskunterbrechung aus. Jeder geladene Prozess kann nun auf diese Botschaft auf folgende Arten reagieren:

- Er signalisiert dem Switcher, daß eine Unterbrechung nicht zulässig ist. Dann bricht der Switcher den Taskwechsel ab.
- Er gibt die Kontrolle sofort an den Switcher zurück und signalisiert so, daß ein Wechsel möglich ist.
- Er bringt sich in einen sicheren Zustand und gibt dann die Kontrolle an den Switcher zurück.

Die per Switcher geladenen Tasks werden durch eine eindeutige *Task ID Nummer* unterschieden. Über diese Nummer lassen sich auch Informationen über den Status eines Tasks abfragen.

In DOS löst der Switcher folgende Aktionen aus, sobald ein Taskwechsel ansteht:

1) Aufbau der Call-Out chain per INT 2F, AX=4B01H
2) Aufruf der Query Supend Funktion
3) Sperre der Interrupts
4) Aufruf der Suspend Session Funktion
5) Sicherung der aktuellen Interruptvektor-Tabelle und der Instance Daten des aktuellen Prozesses
6) Restaurierung der Interruptvektor-Tabelle und der Instance Daten des Vorgängerprozesses
7) Freigabe der Interrupts
8) Erzeugung die lokalen Daten der Zieltask
9) Sperre der Interrupts
10) Sichere die globale Interruptvektor-Tabelle und die Instance Daten.
11) Restauriere die Interruptvektor-Tabelle der Applikation und die Instance Daten

12) Aufruf der Resume Session Funktion

13) Freigabe der Interrupts

14) Aufruf der Session Active Funktion

15) Ausführung der neuen Applikation

Programme können nun diese API-Schnittstelle unterstützen. Hierzu müssen sie zwei Einsprungpunkte aufweisen:

Den Handler für den INT 2F-Service

Den Handler für die Call-Out-API-Funktion

Jedes Programm muß sich in den INT 2F einhängen, um bestimmte Aufrufe des Switcher zu erkennen. Der Switcher löst zum Beispiel den INT 2F mit der Funktion AX = 4B01H aus. Erhält der Handler die Kontrolle vom INT 2F, muß er das AX-Register auswerten, ohne die anderen Register zu verändern. Handelt es sich nicht um einen Switcher-API-Aufruf, ist die Kontrolle sofort per JMP FAR an die nächste Routine des INT 2F zu übergeben. Die Adresse muß der Handler bei der Installation ermitteln (alter INT 2F-Vektor) und speichern. Falls in AX ein gültiger Code des Switchers steht, ist ein PUSHF auszuführen und die nächste Routine des INT 2F per CALL FAR aufzurufen. Dadurch werden erst die Programme in der INT 2F-Kette ausgeführt. Anschließend erhält der Prozess die Kontrolle wieder zurück. Dabei steht in ES:BX ein Zeiger, welcher auf eine Datenstruktur des vorhergehenden Prozesses zeigt. Diese als *Switcher Callback Info Structure* (SCBI) bezeichnete Datenstruktur enthält Informationen über den Prozess und ist über Zeiger verkettet (siehe Funktion 4B01H). Ein Wert von 0:0 in ES:BX signalisiert, daß kein Folgehandler existiert, der den TaskSwitcher; unterstüzt. Der Handler muß nun selbst eine SCBI-Datenstruktur anlegen und den Zeiger auf den SCBI des Vorgängers aus ES:BX in diese Datenstruktur übertragen. Anschließend wird die Adresse auf den eigenen SCBI in ES:BX eingetragen. So erhält der folgende Handler die Adresse und es wird eine verkettete Liste aufgebaut. Der Handler gibt die Kontrolle per IRET-Anweisung an den folgenden Prozess weiter.

Als weiteres muß der Handler einen Einsprungpunkt für die *Call-Out-API-Funktionen* des Switchers aufweisen. Die Adresse dieses Einsprungpunktes wird in der SCBI-Datenstruktur vermerkt. Der Switcher benutzt zum Beispiel diese Call-Out-Aufrufe um Nachrichten an die einzelnen Prozesse abzusetzen. Jeder Handler muß einen entsprechenden Vorrat an API-Funktionen bereitstellen (siehe Funktion 4B01H).

Der Switcher seinerseits besitzt ebenfalls eine Reihe von API-Funktionen (*Call-In-Funktionen*), die über eine eigene Einsprungadresse zu aktivieren sind. Die Adresse läßt sich zum Beispiel per INT 2F, Funktion 4B02H ermitteln. (Beachten Sie, daß sich die Adresse des Einsprungpunktes während der Laufzeit ändern kann).

Für die Kommunikation mit dem DOS-Switcher gibt es ab DOS 5.0 folgenden Funktionen.

5.23.1 Build Call Out Chain (AX = 4B01H)

Mit dieser Unterfunktion baut der Switcher eine Kommunikationskette zwischen den Prozessen auf. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 4BH (DOS 5.0 TaskSwitcher)  º
º AL: 01H (Build Call Out Chain)  º
º CX:DX Switcher Entry Point      º
º ES:BX 0000:0000H                º
û---------------------------------À
º       RETURN                    º
º ES:BX Call Back Infostruktur    º
Û---------------------------------ì
```

Die Funktion wird durch den TaskSwitcher aufgerufen. Anwendungsprozesse (Clients) können sich in die INT 2F-Aufrufkette einhängen. Dadurch erhalten sie Informationen über die Aktivitäten des TaskSwitchers. In CX:DX wird die Eintrittadresse (Call-In-API-Einspungpunkt) des TaskSwitchers übergeben. Beim Aufruf des INT 2F-Handlers muß das Programm zuerst die Kontrolle an folgenden Prozess in den INT 2F-Kette weitergegebenen. Dabei dürfen keine Register verändert werden. Sobald der Client die Kontrolle wieder zurückerhält, muß er eine *Callback Info Structur* im Speicher aufbauen. Der vom vorhergehenden Prozess in ES:BX zurückgegebene Adresswert ist dann in das *next field* der SCBI-Struktur zu kopieren. Der aktive Prozess trägt nun seinerseits die Adresse auf die eigene Callback Info Structur in ES:BX ein und beendet den Aufruf mit IRET.

Ein Client muß sich selbst bei der Installation in die INT 2F-Kette einhängen. Vor der Beendigung muß der Client vom TaskSwitcher mit dem INT 2F-Aufruf AX=4B02H und der Call-In-Unterfunktion AX = 0005H abmelden. Die bei der Aktivierung der Unterfunktion AX = 4B01H in CX:DX vom Switcher übergebene Einsprungadresse für Call-In-Funktionen sollte von den Programmen nicht gespeichert werden, da sich die Adresse durchaus ändern kann. Die Adresse läßt sich über AX = 4B02H jederzeit neu ermitteln.

Die Callback Info Structur besitzt folgende Struktur:

```
Offset Bytes  Bemerkung
 00H   4      next-field-Zeiger auf die Adresse
              der nächsten Callback Info Struktur
 04H          Zeiger auf die Adresse der Behandlungsroutine (Call Out API)
 08H   4      reserviert
 0CH   4      Adresse API Info Struktur
```

Der Zeiger auf das Call-Out-API des Handlers dient dem Switcher zur Kommunikation mit den Tasks. Deshalb muß der Handler hier die Adresse der eigenen API-Routinen eintragen.

Die *API-Info-Struktur* ist eine Liste, die mit 00H abgeschlossen wird (siehe AX = 4B02H).

Der Switcher übergibt beim Aufruf des INT 2F in CX:DX die Adresse des Call-In-API.

Die Call-Out-API-Funktionen

Jeder Handler muß einen Set an Call-Out-Funktionen zur Verfügung stellen. Die Call-Out-Funktionen werden vom Switcher per CALL FAR aktiviert. Der Wert in AX steuert dabei die Art der Funktionen. Hierbei gelten folgende Registerbelegungen:

INIT-Switcher (AX=0000H)

Jede Applikation, die das API unterstützt, kann bei der Initialisierung über die Adressen in der SCBI-Struktur die Call-Out-Funktion der geladenen Prozesse aktivieren. Hierbei gilt folgende Übergabeschnittstelle:

```
CALL FAR xxx

AX = 0000H Init Switcher
ES:DI Switcher Call-In Adresse
Interrupts enabled
DOS Aufrufe enabled

RETURN

AX  = 0 ok
AX <> 0 Fehler
```

Mit AX = 0000H wird die Funktion *switcher initialisation* aktiviert. Das aktivierte Programm kann dann prüfen, ob es eventuell zusätzliche Aktivitäten ausführen muß, um zum Beispiel mit dem Switcher zu kommunizieren. Die DOS 5.0/6.0-Shell führt zum Beispiel bei der Installation diesen Aufruf aus. Enthält AX nach dem Aufruf den Wert 0000H, dann darf der Switcher/ Prozeß geladen werden. Andernfalls sollte der Prozeß beendet werden. Alle geladenen Prozesse werden dann vom Switcher mit der Funktion *Switcher Exit* aktiviert. Der Aufruf kann aber ignoriert werden.

Query Suspend (AX = 0001H)

Mit AX = 0001H wird die Funktion *query suspend* ausgeführt.

```
Ö-----------------------------------Ì
º CALL FAR xxx                     º
º                                  º
º AX = 0001H Query Suspend         º
º BX = Current Session ID          º
º ES:DI Switcher Call-In Adresse   º
º Interrupts enabled               º
º DOS Aufrufe enabled              º
º                                  º
º RETURN                           º
º                                  º
º AX  = 0 ok suspend               º
º AX <> 0 no suspend               º
Û-----------------------------------ì
```

In BX ist die *session ID* mit zu übergeben. Enthält AX nach dem Aufruf den Wert 0000H, dann darf zum nächsten Prozess umgeschaltet werden. Mit AX = 0001H signalisiert ein Task in der Kette, daß keine Umschaltung erlaubt ist. Der Switcher löst dann einen Aufruf *Session Active* aus. Gegebenenfalls kann der aktive Task per *Test Memory Region* prüfen, ob eigene Code- und Datenbereiche durch die Umschaltung betroffen sind.

Suspend Session (AX = 0002H)

Falls beim Aufruf der Call-Out-Funktion *Query Suspend* kein Prozess den Wert AX<>0 zurückgibt, darf der Switcher einen Taskwechsel ausführen. Hierzu wird mit dem Wert AX = 0002H die Funktion *suspend sessio* aktiviert. Vorher müssen alle Interrupts gesperrt und in BX die *session ID* übergeben werden.

```
Ö----------------------------------Ì
º CALL FAR xxx                     º
º                                  º
º AX = 0002H Suspend Session       º
º BX = Current Session ID          º
º ES:DI Switcher Call-In Adresse   º
º Interrupts disabled              º
º DOS Aufrufe disabled             º
º                                  º
º RETURN                           º
º                                  º
º AX  = 0 ok suspend noe           º
º AX <> 0 cannot suspend now       º
Û----------------------------------ì
```

Nach dem Aufruf enthält AX den Statuscode. Enthält AX nach dem Aufruf den Wert 0000H, dann darf zum nächsten Prozess umgeschaltet werden. Bei der Umschaltung wird die komplette Interruptvektor-Tabelle gesichert, b.z.w. durch die Vektoren vor Aktivierung des Tasks ersetzt. Deshalb muß auch das Interruptsystem gesperrt werden. Durch die Restaurierung werden alle lokal vom aktiven Task installierten Interrupt-Handler suspendiert.

Mit AX = 0001H ist keine Umschaltung erlaubt, d.h. ein Prozess verhindet die Umschaltung. Dann löst der Switcher für alle aktiven Prozesse die Funktion *Session Active* aus.

Resume Session (AX = 0003H)

Mit dem Wert AX = 0003H wird die Funktion *activate session* aktiviert. Vorher muß in BX die *session ID* übergeben werden. In CX ist das *session status flag* zu setzen. In diesem Flag ist nur Bit 0 belegt, alle anderen Bits sind reserviert und zu 0 zu setzen. Bit 0 = 1 bedeutet, die Session wird zum ersten mal aktiviert. Weiterhin muß vor dem Aufruf das Interruptsystem gesperrt werden.

```
Ö----------------------------------------Ì
º CALL FAR xxx                           º
º                                        º
º AX = 0003H Resume Session              º
º BX = Current Session ID                º
º ES:DI Switcher Call-In Adresse         º
º CX = Flags                             º
º Interrupts disabled                    º
º DOS Aufrufe disabled                   º
º                                        º
º RETURN                                 º
º                                        º
º AX = 0 (für zukünftige Kompatibilität) º
Û----------------------------------------ì
```

Nach dem Aufruf enthält AX den Wert 0000H. Wird ein neu aktivierter Task zum ersten mal unterbrochen, dann geht ein Create-Session-Aufruf voran.

Session Active (AX = 0004H)

Mit dem Wert AX = 0004H wird die Funktion *session active* aktiviert. Dies ist der Fall, nachdem ein vorher unterbrochener Prozeß wieder geladen wurde. Beim Aufruf muß in BX die *session ID* übergeben werden. In CX ist das *session status flag* zu setzen. In diesem Flag ist nur Bit 0 belegt, alle anderen Bits sind reserviert und zu 0 zu setzen. Bit 0 = 1 bedeutet, die Session wird zum ersten mal aktiviert.

```
Ö-----------------------------------------Ì
º CALL FAR xxx                            º
º                                         º
º AX = 0004H Session Active               º
º BX = Current Session ID                 º
º ES:DI Switcher Call-In Adresse          º
º CX = Flags                              º
º Interrupts enabled                      º
º DOS Aufrufe enabled                     º
º                                         º
º RETURN                                  º
º                                         º
º AX  = 0 (für zukünftige Kompatibilität) º
Û-----------------------------------------ì
```

Nach dem Aufruf enthält AX den Wert 0000H. Die Funktion wird auch aktiviert, falls beim Query Suspend oder bei Suspend Session ein Prozeß einen Taskwechsel blockiert. Hierdurch kann ein aktiver Task eine solche Nachricht erhalten, die dann aber zu ignorieren ist.

Create Session (AX = 0005H)

Mit dem Wert AX = 0005H wird die Funktion *create session* aktiviert. Vorher muß in BX die *session ID* übergeben werden.

```
Ö-------------------------------------Ì
º CALL FAR xxx                        º
º                                     º
º AX = 0005H Create Session           º
º BX = Session ID created Task        º
º ES:DI Switcher Call-In Adresse      º
º Interrupts enabled                  º
º DOS Aufrufe enabled                 º
º                                     º
º RETURN                              º
º                                     º
º AX  = 0 ok create session           º
º    <> 0 no create sessuib           º
Û-------------------------------------ì
```

Enthält AX nach dem Aufruf den Wert 0000H, dann darf die Session erzeugt werden. Mit AX = 0001H ist keine Erzeugung erlaubt. Dann wird vom Switcher die Funktion Destry Session aktiviert.

Destroy Session (AX = 0006H)

Mit dem Wert AX = 0006H wird die Funktion *destroy session* aktiviert. Dies ist zum Beispiel der Fall, falls eine Applikation beendet wird. Vorher muß in BX die *session ID* übergeben werden. Dann beendet der Switcher die entsprechende Session.

```
Ö-------------------------------------Ì
º CALL FAR xxx                        º
º                                     º
º AX = 0006H Destroy Session          º
º BX = Session ID to destroy          º
º ES:DI Switcher Call-In Adresse      º
º Interrupts enabled                  º
º DOS Aufrufe enabled                 º
º                                     º
º RETURN                              º
º                                     º
º AX  = 0 (zukünftige Kompatibilität) º
Û-------------------------------------ì
```

Nach dem Aufruf enthält AX den Wert 0000H. Der Aufruf wird auch ausgelöst, falls ein Create- Session-Aufruf nicht akzeptiert wird.

Switcher Exit (AX = 0007H)

Sobald der TaskSwitcher beendet wird, ist dies den globalen Handlern mitzuteilen. Mit dem Wert AX = 0007H wird dann die Funktion *switcher exit* aktiviert. Vorher muß in BX das Flagregister übergeben werden. Dieses benutzt nur Bit 0, während die restlichen Bits reserviert und auf 0 zu setzen sind. Ist Bit 0 gesetzt, daß ist nur der Switcher geladen.

```
Ö---------------------------------------Ì
° CALL FAR xxx                          °
°                                       °
° AX = 0007H Switcher Exit              °
° BX = Flags                            °
° ES:DI Switcher Call-In Adresse        °
° Interrupts enabled                    °
° DOS Aufrufe enabled                   °
°                                       °
° RETURN                                °
°                                       °
° AX  = 0 (zukünftige Kompatibilität)   °
Û---------------------------------------ì
```

Nach dem Aufruf enthält AX den Wert 0000H. Die Funktion wird von dem Programm aufgerufen, welches den Switcher installiert hat (z.B. DOS-Shell). Dann darf die Call-In-Adresse in ES:DI auch 0:0 sein.

Allgemeine Bemerkungen

Die Call-Out-Funktion 0000H wird von dem Programm aufgerufen, welches den Switcher überwacht oder aktiviert. Der Aufruf erfolgt nicht durch den Taskswitcher selbst. Die in ES:DI übergebene Adresse ist nicht notwendigerweise die Einsprungadresse des TaskSwitchers und kann durchaus 0000:0000H sein. Sobald ein Client signalisiert, daß ein Ladevorgang nicht möglich ist, wird dies über die Funktion 0007H an alle Clients weitergegeben. Die Handler in der Call-Out-Kette dürfen, mit Ausnahme der Funktionen 0002H und 0003H, mit freigegebenem Interruptsystem die Funktionen aufrufen. Weiterhin dürfen alle INT 21-Funktionen aufgerufen werden.

5.23.2 Installation Check (AX = 4B02H)

Mit dieser Unterfunktion des INT 2F läßt sich von einem Programm aus prüfen, ob der TaskSwitcher installiert ist. Weiterhin wird der Aufruf zur Ermittlung der Call-In-Adresse des Switchers verwendet. Es gelten folgende Aufrufkonventionen:

```
Ö---------------------------------Ì
°       CALL:  INT 2F             °
°                                 °
° AH: 4BH (DOS 5.0/6.0 TaskSwitch)°
° AL: 02H (Installation Check)    °
° BX: 0000H                       °
° ES:DI 0000:0000H                °
û---------------------------------À
°       RETURN                    °
° ES:DI Switcher Entry Point      °
Û---------------------------------ì
```

Die Funktion muß mit BX = 0000H und ES:DI = 0000:0000H aufgerufen werden. Ist kein TaskSwitcher installiert, enthält das Registerpaar ES:DI nach dem Aufruf unverändert den Wert 0000:0000H. Bei geladenem Switcher findet sich in ES:DI die Einsprungadresse in den TaskSwitcher (Call-In-API).

Die Call In-Funktionen des TaskSwitchers

Der TaskSwitcher stellt für die Clients eine Reihe von Funktionen (Call In Funktionen) zur Verfügung. Der Einsprungpunkt für die Funktionen wird zum Beispiel bei obiger Funktion

in ES:DI zurückgegeben. Die Funktionen des TaskSwitcher API sind per CALL FAR mit verschiedenen Parametern in AX aufzurufen. Alle Aufrufe werden mit gelöschten Carry-Flag beendet. Ist dies nicht der Fall, wird das API nicht unterstützt. Der Switcher bietet dabei folgende Funktionen:

Get Version (AX = 0000H)

Mit AX = 0000H wird die Funktion *get version* aktiviert. Dies kann zur Identifikation des Switchers genutzt werden. Es gilt folgende Aufrufstruktur:

```
Ö--------------------------------------Ì
º CALL FAR xxxx                        º
º                                      º
º AX = 0000H Get Version               º
º Interrupt disabled                   º
º DOS disabled                         º
º                                      º
º EXIT                                 º
º CY 0 -> Aufruf unterstützt           º
º ES:BX Adresse Switcher Datenblock    º
Û--------------------------------------ì
```

Ein gesetzes Carry-Flag nach der Rückkehr signalisiert: der Aufruf wird nicht unterstützt. Nach einem erfolgreichen Aufruf ist das Carry-Flag gelöscht. AX ist unverändert, aber in ES:BX wird die Adresse auf einen Puffer mit den Versionsdaten zurückgegeben. Dieser Puffer besitzt folgende Struktur:

```
Offset Byte Bedeutung
 00H   2    Hauptversion des Protokolls
 02H   2    Unterversion des Protokolls
 04H   2    Hauptversion TaskSwitcher
 06H   2    Unterversion TaskSwitcher
 08H   2    TaskSwitcher ID
 0AH   2    Operation Flag:
            Bit 0: 1 = TaskSwitcher aus
            Bit 1-15:  ----
 0CH   4    Zeiger auf den Namen des Switchers
 10H   4    Zeiger auf den Einsprungpunkt des
            vorhergehenden TaskSwitchers
```

Als Name des Switchers wird zur Zeit der Text *MS-DOS Shell Task Switcher* zurückgegeben. Der Zeiger ab Offset 10H enthält die Adresse des Einsprungpunktes in den voher geladenen TaskSwitcher. War kein Switcher geladen, ist die Adresse = 0000:0000H.

Test Memory Region (AX = 0001H)

Mit AX = 0001H wird die Call-In-Funktion *test memory region* ausgeführt. In CX ist die Größe des zu testenden Bereiches zu übergeben. ES:DI enthält die Adresse des ersten zu testenden Bytes.

```
Ö-----------------------------------------Ì
º CALL FAR xxxx                           º
º                                         º
º AX = 0001H Test Memory Region           º
º CX = Puffergröße (0-64 Kbyte)           º
º ES:DI Startadresse Puffer               º
º Interrupt disabled                      º
º DOS disabled                            º
º                                         º
º EXIT                                    º
º CY 0 -> Aufruf unterstützt              º
º AX = 0 Puffer ist im globalen Speicher  º
º    1 Puffer ist teilweise lokal         º
º    2 Puffer ist komplett lokal          º
Û-----------------------------------------ì
```

Nach dem Aufruf zeigt ein gesetztes Carry-Flag an, daß die Funktion nicht unterstützt wird. Bei gelöschtem Carry-Flag enthält AX eine Information über die Lage des des getesteten Bereiches.

Damit läßt sich von einem Prozeß prüfen, ob ein Taskwechsel Daten aus dem lokalen Speicher beeinflußt. Dann kann der Prozeß geeignet reagieren.

Suspend Switcher (AX = 0002H)

Mit AX = 0002H wird die Funktion *suspend switcher* ausgeführt. Hierdurch läßt sich ein neuer TaskSwitcher installieren. In ES:DI ist die Adresse des neuen TaskSwitcher Eintrittspunktes anzugeben.

```
Ö-----------------------------------------------Ì
º CALL FAR xxxx                                 º
º                                               º
º AX = 0002H Suspend Switcher                   º
º ES:DI neue Switcher Adresse                   º
º Interrupt enabled                             º
º DOS enabled                                   º
º                                               º
º EXIT                                          º
º CY 0 -> Aufruf unterstützt                    º
º   AX = 0000H Switcher suspendiert             º
º        0001H Switcher nicht suspendiert,      º
º              neuen Switcher beenden           º
º        0002H Switch nicht suspendiert,        º
º              neuer Switcher trotzdem aktiv    º
Û-----------------------------------------------ì
```

Nach dem Aufruf zeigt ein gesetztes Carry-Flag, daß die Funktion nicht unterstützt wird. Bei gelöschtem Carry-Flag enthält AX eine Information über den Status des Switchers. Mit diesem Aufruf läßt sich die Einsprungadresse für das API auf neue Routinen umleiten.

Resume Switcher (AX = 0003H)

Mit dem Wert AX = 0003H wird die Funktion *resume switcher* aktiviert. Vorher muß in ES:DI die Adresse des neuen Switcher entry points übergeben werden.

```
Ö------------------------------------------------Ì
º CALL FAR xxxx                                  º
º                                                º
º AX = 0003H Resume Switcher                     º
º ES:DI Call In Adresse                          º
º Interrupt enabled                              º
º DOS enabled                                    º
º                                                º
º EXIT                                           º
º CY 0 -> Aufruf unterstützt                     º
º   AX = 0000H (für zukünftige Erweiterungen)    º
Û------------------------------------------------ì
```

Bei glöschtem Carry-Flag war der Aufruf erfolgreich, andernfalls konnte er nicht ausgeführt werden. In ES:DI ist die Call-In-Adresse des zu beendenden Switchers zu übergeben.

Hook Notification Chain (AX = 0004H)

Mit dem Wert AX = 0004H wird die Funktion *hook notification chain* aktiviert. Damit läßt sich von einem laufenden Prozeß eine neue *callback info structure* in die Kette einsetzen. Die Adresse dieser Struktur ist beim Aufruf in ES:DI zu übergeben.

```
Ö------------------------------------------------Ì
º CALL FAR xxxx                                  º
º                                                º
º AX = 0004H Hook Call Out Chain                 º
º ES:DI SCBI-Adresse                             º
º Interrupt enabled                              º
º DOS enabled                                    º
º                                                º
º EXIT                                           º
º CY 0 -> Aufruf unterstützt                     º
º   AX = 0000H (für zukünftige Erweiterungen)    º
Û------------------------------------------------ì
```

Ein gelöschtes Carry-Flag zeigt nach dem Aufruf den Erfolg der Aktion an. Nach dem Aufruf enthält AX den Wert 0000H.

Unhook Notification Chain (AX = 0005H)

Mit dem Wert AX = 0005H wird die Funktion *unhook notification chain* aktiviert. Diese trägt den Handler des rufenden Prozesses aus der Call-Out-Kette aus. Hierzu wird die in ES:DI übergebene Adresse der *callback info structure* aus der Kette ausgetragen.

```
Ö------------------------------------------------Ì
º CALL FAR xxxx                                  º
º                                                º
º AX = 0005H Unhook Call Out Chain               º
º ES:DI SCBI-Adresse                             º
º Interrupt enabled                              º
º DOS enabled                                    º
º                                                º
º EXIT                                           º
º CY 0 -> Aufruf unterstützt                     º
º   AX = 0000H (für zukünftige Erweiterungen)    º
Û------------------------------------------------ì
```

Ein gelöschtes Carry-Flag zeigt nach dem Aufruf den Erfolg an.

Query API Support (AX = 0006H)

Mit dem Wert AX = 0006H wird die Funktion *query API support* aktiviert. Wird eine Funktion (z.B. ein Netzwerktreiber) von zwei Handlern mit unterschiedlichen Versionsständen abgewickelt, läßt sich der API-Level abfragen. Vorher muß in BX eine Identifikationsnummer für asynchrone API-Aufrufe übergeben werden.

```
Ö-----------------------------------------------Ì
°CALL FAR xxxx                                  °
°                                               °
° AX = 0006H Query API Support                  °
° BX = API Code                                 °
° Interrupt disabled                            °
° DOS enabled                                   °
°                                               °
° EXIT                                          °
° CY 0 -> Aufruf unterstützt                    °
°   AX = 0000H (für zukünftige Erweiterungen)   °
°   ES:BX Adresse der API-Info-Struktur         °
Û-----------------------------------------------ì
```

Ein gelöschtes Carry-Flag zeugt von einem erfolgreichen Aufruf und in ES:BX findet sich die Adresse der API-Infostruktur. Diese besitzt folgenden Aufbau:

```
Offset Byte Bemerkung
  00H  2    Länge der Struktur (0AH)
  02H  2    API Identifikationscode
               0001H NetBIOS
               0002H 802.2
               0003H TCP/IP
               0004H LAN Manager named pipes
               0005H Novell Netware IPX
  04H  2    Hauptversionsnummer
  06H  2    Unterversionsnummer
  08H  2    Support Level
               0001H minimale Unterstützung
               0002H API-Level Support
               0003H Switcher Kompatibilität
               0004H volle Kompatibilität
```

Die genaue Bedeutung dieser Aufruf ist mir allerdings nicht klar.

5.23.3 Allocate Switcher ID (AX = 4B03H)

Diese Unterfunktion wird ausschließlich durch den TaskSwitcher ausgeführt, um eine Identifikationsnummer zuweisen.

```
Ö---------------------------------Ì
°       CALL:  INT 2F             °
°                                 °
° AH: 4BH (DOS 5.0/6.0 TaskSwitch)°
° AL: 03H (Allocate Switcher ID)  °
° BX: 0000H                       °
° ES:DI Entry Point               °
û---------------------------------À
°      RETURN                     °
° BX: Switcher ID                 °
Û---------------------------------ì
```

Die Funktion muß vom ersten TaskSwitcher aufgerufen werden, damit ihm eine Identifikationsnummer zugewiesen wird. In ES:DI steht die Adresse des TaskSwitcher Entry Points. Nach dem Aufruf enthält AX = 0000H und in BX findet sich die Switcher ID. Diese setzt sich aus 4 Bit Switcher ID und einer 12-Bit-Tasknummer zusammen. Es dürfen 15 Switcher mit je 4096-Tasks geladen werden. (Anmerkung: die DOS 5.0-Shell unterstützt jedoch nur 16 Tasks). Der Wert BX = 0000H bedeutet, daß keine ID-Nummer mehr frei ist. Nummern für die Switcher ID liegen zwischen 0 und 15. Genauere Informationen über diesen Aufruf sind mir allerdings nicht bekannt.

5.23.4 Free Switcher ID (AX = 4B04H)

Diese Unterfunktion ist aufzurufen, falls der Switcher terminiert.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 4BH (DOS 5.0/6.0 TaskSwitch)º
º AL: 04H (Free Switcher)         º
º BX: Switcher ID                 º
º ES:DI Entry Point               º
û---------------------------------À
º       RETURN                    º
º BX: Status                      º
Û---------------------------------ì
```

Die Funktion muß mit der freizugebenden ID-Nummer in BX und dem TaskSwitcher-Call-In-Entry-Point in ES:DI aufgerufen werden. Nach dem Aufruf ist AX = 0000H und BX enthält einen Status (0000H Aufruf erfolgreich, sonst war die ID ungültig oder nicht erlaubt).

5.23.5 Identify Instance Data (AX = 4B05H)

Diese Unterfunktion wird durch den TaskSwitcher selbst aufgerufen. Es gilt folgende Schnittstelle:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 4BH (DOS 5.0/6.0 TaskSwitch)º
º AL: 05H (Instance Data)         º
º ES:BX 0000:0000H                º
º CX:DX Call In Entry Point       º
û---------------------------------À
º       RETURN                    º
º ES:BX Startup Info Structure    º
Û---------------------------------ì
```

In CX:DX wird die Adresse des Switcher-Call-InEntry-Points übergeben. Nach dem Aufruf steht in ES:BX die Adresse auf eine *WIN 386 Startup Info Structure* mit folgendem Aufbau:

```
Offset Byte Bedeutung
00H    2    Version Info Struktur
02H    4    Zeiger auf nächste Startup Info Struktur
08H    4    0000:00000
0AH    4    reserviert
0EH    4    Zeiger auf den Instance Data Record

Offset Byte Bedeutung Instance Data Record
00H    4    Adresse   Instance Data
04H    2    Länge     Bereich Instance Data
```

Client-Programme mit Instance Daten müssen diesen Aufruf abfangen und ohne Veränderung der Register an den vorhergehenden Treiber weiterleiten. Nach der Rückkehr muß der Client obige Startup-Info-Struktur anlegen, die in ES:BX übergebene Adresse in der Struktur ab Offset 02H speichern und die Adresse der eigenen Struktur in ES:BX zurückgeben. Eine genauere Bedeutung des Aufrufes ist allerdings nicht bekannt.

Damit möchte ich die Beschreibung des TaskSwitcher API beenden.

Die Unterfunktion 5453H wird durch das TesSeRact-Interface belegt. Hierbei handelt es sich um eine Vereinbarung zur Koordinierung residenter Programme. Einzelheiten finden sich im Anhang.

5.24 DOS 5.0/6.0 COMMAND.COM Interface (AX = 5500H)

Diese Unterfunktion existiert erst ab DOS 5.0 und dient intern zur Kommunikation zwischen transientem und residentem Teil von COMMAND.COM. Die Schnittstelle wird benötigt, falls der DOS-Kern im HMA-Bereich liegt.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 55H (DOS 5.0/6.0)           º
º AL: 00H (COMMAND.COM)           º
û---------------------------------À
º       RETURN                    º
º AX: 0000H                       º
º DS:SI Einspungadresse           º
Û---------------------------------ì
```

Die Funktion wird vom residenten Teil von COMMAND.COM aktiviert, welche den gemeinsam nutzbaren (shareable) Teil von COMMAND nutzen möchte. Die genaue Aufrufstruktur zur Aktivierung von DOS-Funktionen über die via DS:SI zurückgegebene Adresse ist aber nicht bekannt.

Die Funktion AX = 62XXH des INT 2F wird durch PC_Tools V 7.0 belegt. Genaue Aufrufkonventionen sind aber nicht bekannt.

5.25 Novell NetWare IPX Installation Check (AH = 7AH)

Mit diesem Aufruf läßt sich prüfen, ob der residente Teil der Novell NetWare (Low Level IPX API) installiert ist. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: 7AH (NetWare)               º
º AL: 00H Get Installation Status º
û---------------------------------À
º       RETURN                    º
º AL: Installation Status         º
º ES:DI FAR CALL Entry Point      º
Û---------------------------------ì
```

Die Funktion gibt nach dem Aufruf im Register AL den Installationsstatus zurück. Mit AL = 00H ist NetWare nicht installiert und kann installiert werden. Mit AL = FFH ist der Treiber resident geladen. Dann enthält das Registerpaar ES:DI die Adresse des CALL-FAR-Einsprungpunktes für den Funktionsdispatcher. Es besteht aber auch die Möglichkeit, den Treiber über die INT 2A-Funktionen anzusprechen.

Weiterhin benutzt die NetWare-Shell (3.01d) noch die Funktionen AX = 7A8xH und AX = 7FxxH des INT 2F. Die genaue Aufrufschnittstelle ist aber nicht bekannt.

5.26 ERGO-DOS-Extender Installation Check (AH = A1H)

Die ERGO-DOS-Extender benutzen den Aufruf zur Installationsprüfung. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: A1H (ERGO Extender)         º
º AL: 00H Extender Code           º
º BX: 0081H                       º
º ES:DI 16-Byte-Puffer            º
û---------------------------------À
º       RETURN                    º
º ---                             º
Û---------------------------------ì
```

Im Register AX ist vor dem Aufruf eine Kennung des Extendertyps zu übergeben:

```
 AL = FEH  OS=A/286 oder OS/386
      FFH  HummingBoard DOS Extender
```

In ES:DI wird eine Adresse auf einen 16 Byte langen Puffer zurückgegeben, der bei installiertem Extender die Zeichen "IABH" enthält.

5.27 GRAPHICS.COM Installation Check (AX = AC00H)

Dieser Aufruf existiert erst ab DOS 4.01 und wurde eingeführt, damit GRAPHICS.COM nicht den gleichen Code wie die CD-ROM zur Installationsprüfung benutzt. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: C0H (GRAPHICS.COM)          º
º AL: 00H Installation Status     º
û---------------------------------À
º       RETURN                    º
º AX: Installation Status         º
Û---------------------------------ì
```

Das Register AX enthält nach dem Aufruf eine Kennung. Mit AX = FFFFH ist der Treiber installiert.

5.28 Kommunikation with DISPLAY.SYS (AH = ADH)

Ab DOS 3.3 wird der Treiber DISPLAY.SYS zur Ausgabe verschiedener Zeichenfonts verwendet. Der INT 2F bietet über die Funktion ADH die Möglichkeit zur Kommunikation mit dem Treiber.

5.28.1 Get Installation Status (AL = 00H)

Mit diesem Aufruf läßt sich prüfen, ob der Treiber resident installiert ist.

```
Ö--------------------------------Ì
º       CALL:  INT 2F            º
º                                º
º AH: ADH (DISPLAY)              º
º AL: 00H Get Installation Status º
û--------------------------------À
º       RETURN                   º
º AL: Installation Status        º
Û--------------------------------ì
```

Die Funktion gibt nach dem Aufruf im Register AL den Installationsstatus zurück. Mit AL = 00H ist DISPLAY nicht installiert und kann installiert werden. Mit AL = FFH ist der Treiber resident geladen.

Der Treiber belegt die Unterfunktionen AL = 00H, 01H, 02H, 03H und ab DOS 4.0 die Codes 10H, 40H. Die Aufrufschnittstelle ist allerdings nicht bekannt.

5.29 DOS 3.3 -Kommunikation mit KEYB.COM (AX = AD8XH)

Der Treiber KEYB.COM benutzt ebenfalls einige Unterfunktion des INT 2F mit dem Code AH = ADH.

5.29.1 KEYB.COM Get Installation Status (AX = AD80H)

Als weitere Besonderheit wird ab DOS 3.3 über den INT 2F mit AH = ADH der Installationsstatus des Treibers KEYB.COM geprüft. In AL ist der Wert 80H zu übergeben.

```
Ö--------------------------------Ì
º       CALL:  INT 2F            º
º                                º
º AH: ADH (KEYB.COM)             º
º AL: 80H Get Installation Status º
û--------------------------------À
º       RETURN                   º
º AL: Installation Status        º
º BX: Versionsnummer             º
º ES:DI interne Daten            º
Û--------------------------------ì
```

Die Funktion gibt nach dem Aufruf im Register AL den Installationsstatus zurück. Mit AL = 00H ist KEYB nicht installiert und kann installiert werden. Mit AL = FFH ist der Treiber resident geladen. Dann findet sich in ES:DI ein Zeiger auf einen internen Datenbereich. Die Belegung ist allerdings nicht vollständig bekannt. Die in BX zurückgegebene Versionsnummer ist bis zur Version DOS 5.0/6.0 immer 1.0.

5.29.2 KEYB.COM Set Keyboard Code Page (AX = AD81H)

Mit diesem Aufruf läßt sich ab DOS 3.3 die Codepage für die Tastatur setzen.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: ADH (KEYB.COM)              º
º AL: 81H Set Code Page           º
º BX: Code Page                   º
û---------------------------------À
º       RETURN                    º
º CF: 1 Fehler                    º
º AX: 0001H (Code Page fehlt)     º
º CF: 0 ok.                       º
Û---------------------------------ì
```

Der Aufruf wurde erst ab DOS 5.0 dokumentiert, steht aber bereits ab DOS 3.3 zur Verfügung. Die Funktion wird durch DISPLAY.SYS aufgerufen.

5.29.3 KEYB.COM Set Keyboard Mapping (AX = AD82H)

Mit diesem Aufruf läßt sich ab DOS 3.3 die Einstellung der Tastatur verändern.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: ADH (KEYB.COM)              º
º AL: 82H Set Keyboard Mapping    º
º BL: Mode                        º
û---------------------------------À
º        RETURN                   º
º CF: 1 Fehler                    º
º CF: 0 ok.                       º
Û---------------------------------ì
```

Der Aufruf wurde erst ab DOS 5.0 dokumentiert, steht aber bereits ab DOS 3.3 zur Verfügung. In BL wird der neue Tastaturmode übergeben. Hierbei gilt:

```
BL = 00: US-Keyboard
     FF: Foreign Keyboard
```

Mit FFH läßt sich damit auf die landessprachliche Tastatur umschalten. Auf der Tastatur kann die Funktion über Ctrl+Alt+F1 und Ctrl+Alt+F2 ebenfalls erreicht werden.

5.29.4 KEYB.COM Get Keyboard Mapping (AX = AD83H)

Mit diesem Aufruf läßt sich ab DOS 3.3 die Einstellung der Tastatur abfragen.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: ADH (KEYB.COM)              º
º AL: 83H Get Keyboard Mapping    º
û---------------------------------À
º       RETURN                    º
º BL: Mode                        º
Û---------------------------------ì
```

Der Aufruf wurde erst ab DOS 5.0 eingeführt und gibt in BL den eingestellten Tastaturmode zurück. Hierbei gilt:

```
BL = 00:  US-Keyboard
     FF:  Foreign Keyboard
```

Über die Funktion AX = AD82H läßt sich der Mode umschalten.

5.30 DOS Installable Command (AH = AEH)

Ab DOS 3.3 besteht die Möglichkeit zur nachträglichen Installation residenter Kommandos im Kommandointerpreter. Diese müssen lediglich die Funktion AEH des INT 2F abfangen. Bei jeder unbekannten Kommandoeingabe aktiviert COMMAND.COM dann die Routinen in dieser Funktion. Dadurch lassen sich nachträglich neue DOS-Befehle installieren.

5.30.1 Installable Command Install Check (AX = AE00H)

Dieser Aufruf wird durch COMMAND.COM aktiviert, sobald ein Kommando in DOS eingegeben wurde. Dabei werden folgende Parameter übergeben:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: AEH (Installable Command)   º
º AL: 00H Install Check           º
º DX: FFFFH                       º
º DS:BX Kommandozeile             º
û---------------------------------À
º       RETURN                    º
º AL: Status                      º
Û---------------------------------ì
```

Der Aufruf steht erst ab DOS 3.3 zur Verfügung. In DS:BX wird von COMMAND.COM die Adresse auf den Puffer mit der Kommandozeile übergeben. Ein eventuell resident installiertes Programm kann dann die Kommandozeile auswerten. Im Register AL muß das Programm dann einen Status an COMMANDO.COM zurückgeben:

```
AL = 00:  Kommandozeile wie normal ausführen
     FF:  Kommando ist eine Erweiterung von
          DOS und wird durch TSR-Programm ausgeführt.
```

Das Programm APPEND benutzt diesen Aufruf um sich als internes Kommando zu installieren. Die Kommandozeile ist in einem Buffer gespeichert, der folgende Struktur besitzt:

```
Byte  Bemerkung
 1    Länge Puffer
 1    Zeichenzahl im Puffer
 n    Zeichen im Puffer
```

Das aktivierte Programm kann dann den Text im Puffer analysieren und gegebenenfalls als Kommando ausführen. Dann ist der Wert FFH in AL zurückzugeben, damit COMMAND.COM das Kommando nicht selbst ausführt.

5.30.2 Installable Command Execute Command (AX = AE01H)

Dieser Aufruf wird durch COMMAND.COM aktiviert, sobald ein Kommando in DOS eingegeben wurde und bekannt ist, daß eine residente Erweiterung vorliegt. Dabei werden folgende Parameter übergeben:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: AEH (Installable Command)   º
º AL: 01H Execute Command         º
º DX: FFFFH                       º
º DS:SI Puffer                    º
û---------------------------------À
º       RETURN                    º
º DS:SI Puffer                    º
Û---------------------------------ì
```

Der Aufruf steht erst ab DOS 3.3 zur Verfügung. Die Funktion wird ausgeführt, falls beim Aufruf von AX = AE00H das TSR-Programm den Wert AL = FFH zurückgegeben hat. In DS:SI wird die Adresse auf den Puffer übergeben. Nach dem Aufruf steht in Puffer im ersten Byte die Länge des Kommandos. Dahinter folgt das Kommando in Großbuchstaben (falls Länge <> 0). Falls das gerufende Programm den String nicht löscht, versucht COMMAND.COM dann das Kommando als internen Befehl auszuführen. APPEND nutzt z.B. diesen Mechanismus.

5.31 DOS 3.3 GRAFTABL.COM INSTALLATION CHECK (AH = B0H)

Die Unterfunktion ist ab DOS 5.0 dokumentiert und erlaubt ab DOS 3.3 die Prüfung, ob GRAFTABL.COM installiert ist.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: B0H (GRAFTABL)              º
º AL: 00H Get Installationsstatus º
û---------------------------------À
º       RETURN                    º
º AL: Statusbyte                  º
Û---------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Programmes.

- Falls AL = 0 ist, dann ist der Treiber nicht vorhanden und kann installiert werden.
- Mit AL = 1 ist der Treiber nicht vorhanden, kann aber auch nicht neu installiert werden.
- Falls AL = FFH ist, dann ist der Treiber installiert.

GRAFTABL nutzt weiterhin den Code 01H als Unterfunktion um eine Tabelle mit den Graphik Fonts zu laden. Die Adresse der Tabelle ist in DS:BX zu übergeben. Die Bedeutung der Aufrufes ist allerdings nicht klar.

Die INT 2F-Funktion B4xxH wird durch den IBM 3270-Emulator belegt. Genaue Spezifikationen liegen jedoch nicht vor.

5.32 Kommunikation with APPEND (AH = B7H)

Der Aufruf steht ab DOS 3.3 zur Verfügung.

```
Ö--------------------------------Ì
º       CALL:  INT 2F            º
º                                º
º AH: B7H (APPEND)               º
º AL: Unterfunktionscode         º
û--------------------------------À
º       RETURN                   º
º AL: Statusbyte                 º
Û--------------------------------ì
```

Mit diesem Aufruf läßt sich mit dem residenten Programm APPEND kommunizieren. Es gelten folgende Übergabeparameter:

Im Register AL wird der Code der Unterfunktion angegeben.

5.32.1 Get Installationsstatus (AX = B700H)

In DOS 3.3 ist nur die Unterfunktion AL = 00 (Get Installationsstatus) implementiert. Mit diesem Aufruf läßt sich der Status des residenten Programmes APPEND prüfen. Es gelten folgende Übergabeparameter:

```
Ö--------------------------------Ì
º       CALL:  INT 2F            º
º                                º
º AH: B7H (APPEND)               º
º AL: 00H Get Installationsstatus º
û--------------------------------À
º   RETURN                       º
º AL: Statusbyte                 º
Û--------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Treibers.

- Falls AL = 0 ist, dann ist der residente Teil von APPEND nicht vorhanden und kann installiert werden.
- Falls AL = FFH ist, dann ist der residente Teil von APPEND installiert.

Mit dem APPEND-Kommando lassen sich in DOS Suchpfade für verschiedene Verzeichnisse angeben. Diese Einheiten werden durchsucht, falls eine Datei nicht im Standardverzeichnis gefunden wird. Der Aufruf ist in DOS 3.x und DOS 4.x implementiert.

Der Aufruf AX = B701H ist durch APPEND belegt, aber die Schnittstelle ist nicht bekannt.

5.32.2 Get APPEND Version (AX = B702H)

Dieser Aufruf ist erst ab DOS 4.0 implementiert und ermöglicht die Abfrage der APPEND-Version. Er gelten folgende Übergabeparameter:

```
Ö--------------------------------Ì
º       CALL:  INT 2F            º
º                                º
º AH: B7H (APPEND)               º
º AL: 02H Get Version            º
û--------------------------------À
º       RETURN                   º
º AX: Version                    º
Û--------------------------------ì
```

Der Aufruf ermöglicht eine Prüfung, ob die Netzwerkversion oder die DOS-Version des APPEND-Befehls geladen ist.

Enthält das Register AX nach dem Aufruf den Wert FFFFH, dann ist weder die DOS-4.0 noch die 5.0 Version des Befehls APPEND installiert.

Bei DOS 4.0-6.0 enthält AL die Hauptversionsnummer und AH die Unterversionsnummer.

5.32.3 APPEND Hook INT 21 (AX = B703H)

Dieser Aufruf steht in DOS 3.3 und 5.0 zur Verfügung. Er fängt den INT 21-Interrupt ab.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: B7H (APPEND)                º
º AL: 03H Hook INT 21             º
º ES:DI Adresse                   º
û---------------------------------À
º   RETURN                        º
º ES:DI Adresse                   º
Û---------------------------------ì
```

Mit diesem Aufruf hängt sich APPEND in den INT 21-Vektor ein. Bei den folgenden Aufrufen wird über ein Flag entschieden, ob der APPEND-Handler, oder der INT 21-Handler aktiviert wird.

5.32.4 Get APPEND Path Pointer (AL = 04H)

Auch dieser Aufruf ist erst ab DOS 4.0 implementiert und ermöglicht die Abfrage des APPEND-Zeigers auf den APPEND-Pfad. Es gelten folgende Übergabeparameter:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: B7H (APPEND)                º
º AL: 04H Get APPEND Path Pointer º
û---------------------------------À
º   RETURN                        º
º ES:DI Zeiger auf Path           º
Û---------------------------------ì
```

Der Aufruf gibt im Registerpaar ES:DI einen 32-Bit-Zeiger auf den gerade aktiven APPEND-Pfad zurück. Dieser Aufruf ist allerdings nur bei der DOS-Version 4.0 des APPEND-Befehls implementiert.

5.32.5 Get APPEND Function State (AL = 06H)

Dieser Aufruf ist nur in der DOS-4.0-Version von APPEND implementiert und ermöglicht die Abfrage des Funktionsstatus. Es gelten folgende Aufrufparameter:

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: B7H (APPEND)                º
º AL: 06H Get Function State      º
û---------------------------------À
º       RETURN                    º
º BX: Function State              º
Û---------------------------------ì
```

Im Register BX wird ein 16-Bit-Statuswort zurückgegeben, welches Aufschluß über den aktuellen Zustand von APPEND gibt. Bei gesetzten Bits gelten folgende Bedingungen:

```
Bit  0 :  APPEND ist aktiv
    12 :  Directory Search (DOS 5.0)
    13 :  /PATH Option ist gesetzt
    14 :  /E Option ist gesetzt
    15 :  /X Option ist gesetzt
```

Diese Optionen lassen sich in der DOS-4.0-Version von APPEND setzen oder löschen. Die vorliegende Funktion erlaubt nur die Abfrage der gesetzten Optionen.

5.32.6 Set APPEND Function State (AL = 07H)

Dieser Aufruf ist ab der DOS-4.0-Version von APPEND implementiert und ermöglicht die Modifizierung des Funktionsstatus. Es gelten folgende Aufrufparameter:

```
Ö--------------------------------Ì
º       CALL:  INT 2F            º
º                                º
º AH: B7H (APPEND)               º
º AL: 07H Set Append State       º
º BX: Append State               º
û--------------------------------À
º   RETURN                       º
º  ---                           º
Û--------------------------------ì
```

Im Register BX wird ein 16-Bit-Statuswort übergeben, welches den Status der APPEND-Funktion modifiziert. Dabei gilt die Kodierung der einzelnen Bits gemäß Funktion AX = B706H.

Ein gesetztes Bit (1) aktiviert die entsprechende Option, während ein gelöschtes Bit die Option deaktiviert. Diese Optionen lassen sich nur in der DOS-4.0-Version von APPEND setzen oder löschen.

5.32.7 Get Version Info (AL = 10H)

Der Aufruf ist erst ab DOS 3.3 verfügbar und wird mit AX = B710H aktiviert. Nach dem Aufruf enthält DL die Hauptversionsnummer und DH die Unterversionsnummer.

5.32.8 Set Return Found Name (AL = 11H)

Dieser Aufruf ist erst ab der DOS-4.0-Version von APPEND implementiert und setzt das Return-Found-Name-Flag. Es werden keine Parameter übergeben. Ist dieses Flag gesetzt, gibt APPEND beim nächsten INT 21-Funktionsaufruf 3DH (Open Handle), 43H (Get/Set File Attribut) oder 6CH (Extended Open/Create) den Namen des Zugriffspfades in den ASCIIZ-Buffer des Anwenderprozesses zurück. Es ist darauf zu achten, daß in diesem Buffer genügend Platz zur Aufnahme des ASCIIZ-Strings mit dem Pfadnamen vorhanden ist. Nach Ausführung des INT 21 wird das Return-Found-Name-Flag durch APPEND wieder zurückgesetzt. Mit diesem Funktionsaufruf läßt sich demnach der Name des Zugriffspfades für Daten ermitteln.

5.33 Kommunikation with Network (AH = B8H)

Über diesen Aufruf des Interrupt 2F lassen sich verschiedene Netzwerkfunktionen ansprechen.

5.33.1 Netzwerk Installation Check (AL = 00H)

Die Unterfunktion prüft, ob die Kommunikation mit den Netzwerkfunktionen möglich ist.

```
Ö---------------------------------Ì
º       CALL:  INT 2F             º
º                                 º
º AH: B8H (Netzwerk)              º
º AL: 00H Installation Check      º
û---------------------------------À
º   RETURN                        º
º AL: 0 not installed             º
º     > 0 installed               º
º BX: Komponent Flag              º
Û---------------------------------ì
```

Die Funktion gibt im Register AL den Installationsstatus zurück. Alle Werte ungleich 0 signalisieren, daß der Treiber installiert ist. Im Register BX finden sich dann einige Flags, die auf die Art des Netzwerkknotens hinweisen.

```
BX = installierte Komponenten
Bit 6   server
Bit 2   messenger
Bit 7   receiver
Bit 3   redirector
```

Die Funktion benutzt noch die Unterfunktion 03H, um die Power-On-Restart-Adresse zu ermitteln. Diese wird in ES:BX zurückgegeben. Mit der Unterfunktion 04H läßt sich in ES:BX die Adresse einer neuen POST-Routine übergeben. Mit AX = B807H wird die NetBIOS-Adresse der Maschine abgefragt und in CH zurückgegeben. Der Code AX = B808H ist benutzt, die Belegung ist aber nicht bekannt.

Die Funktion AX = B9xxH wird ebenfalls vom Netzwerk-Redirector RECEIVER.COM belegt. Die Funktionen:

```
AL = 00H: Installationsprüfung
     01H: GET RECEIVER.COM INT 2F Handler Adr.
     03H: Get RECEIVER.COM POST Adr.
     04H: Set RECEIVER.COM POST Adr.
     05H: Get Filename
     06H: Set Filename
     08H: Unlink Keyboard Handler
```

Da RECEIVER.COM nur selten benutzt wird, möchte ich auf die Wiedergabe der Aufrufschnittstelle an dieser Stelle verzichten.

5.34 DOS/WINDOWS EGA.SYS Check (AH = BCH)

Dieser Aufruf ist ab DOS 5.0 und ab WINDOWS 3.0 implementiert, sobald der Treiber EGA.SYS installiert wurde. Dieser Treiber soll den Zustand der EGA-Register für externe Programme speichern.

5.34.1 EGA.SYS Installation Check (AX = BC00H)

Mit diesem Aufruf läßt sich prüfen, ob der Treiber installiert wurde.

```
Ö--------------------------------Ì
º       CALL:  INT 2F            º
º                                º
º AH: BCH (EGA.SYS)              º
º AL: 00H Installation Check     º
û--------------------------------À
º       RETURN                   º
º  AL: Status                    º
Û--------------------------------ì
```

Der zurückgegebene Wert im Register AL gibt Hinweise über den Status des installierten Treibers.

- Falls AL = 0 ist, dann ist der residente Teil von EGA.SYS vorhanden und kann installiert werden.
- Falls AL = 1 ist, dann ist der residente Teil von EGA.SYS nicht vorhanden, er läßt sich aber auch nicht installieren.
- Falls AL = FFH ist, dann ist EGA.SYS installiert.

In BX wird bei installiertem Treiber der Wert 5456H (TV) zurückgegeben.

5.34.2 EGA.SYS Get Version Info (AX = BC06H)

Mit diesem Aufruf läßt sich die Versionnummer abfragen.

```
Ö--------------------------------Ì
º       CALL:  INT 2F            º
º                                º
º AH: BCH (EGA.SYS)              º
º AL: 06H Get Version Info       º
û--------------------------------À
º       RETURN                   º
º BX: 5456H ("TV")               º
º CX: Version                    º
º DL: Revision                   º
Û--------------------------------ì
```

Der zurückgegebene Wert im Register CX gibt die Version (CH = Hauptversionsnummer) des Treibers an. In DL findet sich eine Revisionsnummer.

5.35 REDIRIFS.EXE Interface (AX = BFXXH)

Der INT 2F mit dem Code BFH ist für die Kommunikation mit dem PC-LAN-Programm REDIRIFS.EXE reserviert. Mit AL = 00H läßt sich der Installationsstatus abfragen. Mit der Unterfunktion BF80H läßt sich die Adresse des internen Arbeitsspeichers abfragen. In ES:DI ist vor dem Aufruf der FAR Entry Point des IFS Handlers in REDIRIFS zu übergeben. Nach dem Aufruf zeigt ES:DI auf einen internen Arbeitsspeicher. Die genaue Bedeutung des Aufrufes ist mir allerdings nicht bekannt.

Der Code AX = C000H ist durch den *Novell ODI Link Support Layer* belegt, der hier den Installationscheck abwickelt. Bei installiertem Treiber wird AL = FFH zurückgegeben.

Die Funktion AX = D2xxH wird durch Quarterdecks QEMM 5.0 belegt, der hier die Installationsprüfung abwickelt. (BX = QD, CX = ME und DX = M0). Nach dem Aufruf ist AL =FFH falls der Treiber vorhanden ist. DESKVIEW belegt die Funktion AX = DExxH.

PharLap belegt mit dem DOS-Extender den Code AX = ED00H zur Installationsprüfung. In BL ist beim Aufruf die Versionsnummer des Extenders zu übergeben. Der Status wird in AL zurückgegeben. Borland benutzt dagegen für ihren DPMI-Lader und den DOS-Extender den Code FBXXH. Auf eine detaillierte Beschreibung wird an dieser Stelle verzichtet, da die Schnittstellen teilweise nicht bekannt sind und der Stoff den Umfang des Buches sprengen würde.

Damit ist die Beschreibung des INT 2FH abgeschlossen.

6 Die Fehlerbehandlung in DOS

Beim Aufruf der DOS-Funktionen und innerhalb der Interrupt-Service-Routinen können Fehler auftreten. MS-DOS kennt nun verschiedene Möglichkeiten, diese Fehler zu behandeln. Diese verschiedenen Modi sind etwas abhängig von der Version des Betriebssystems und sollen nachfolgend beschrieben werden.

6.1 Funktionen ohne Fehlermeldung

Es existieren einige DOS-Funktionsaufrufe, wo Fehler nicht bemerkt werden können. In diesen Fällen werden überhaupt keine Fehlermeldungen zurückgegeben. Nachfolgende Tabelle enthält eine Aufstellung dieser Funktionen.

Code	Funktion
00	Terminate Programm
01	Read Keyboard And Echo
02	Display Character
03	Auxiliary Input
04	Auxiliary Output
05	Print Character
06	Direct Console I/o
07	Direct Console Input
08	Read Keyboard
09	Display String

Tabelle 6.1: Funktionen ohne Fehlermeldung

Code	Funktion
0a	Buffered Keyboard Input
0b	Check Keyboard Status
0d	Reset Disk
0e	Select Disk
19	Get Current Disk
1a	Set Disk Transfer Address
1b	Allocation Table Information
1c	Allocation Table Information For Spezific Device
24	Set Relative Record Field
25	Set Interrupt Vektor
26	Create New Psp
2a	Get Date
2c	Get Time
2e	Set/reset Verify Switch
2f	Get Disk Transfer Address
30	Get Dos Version Number
31	Terminate Process And Remain Resident
35	Set Interrupt Vektor
36	Get Disk Free Space
4c	Terminate A Process
54	Get Verify State
62	Get Psp Address

Tabelle 6.1: Funktionen ohne Fehlermeldung (Ende)

Nun kann es aber vorkommen, daß eine dieser Funktionen versucht, auf eine nicht existierende Einheit zuzugreifen. In diesen Fällen tritt ein Fehler auf, der durch das System abzufangen ist. Wie diese Fehler behandelt werden, wird nachfolgend beschrieben.

6.2 Critical-Error-Handler

Wie bereits oben beschrieben, gibt es in DOS Fehlersituationen, die eine sofortige Benachrichtigung des Benutzers verlangen. Hierzu gehören insbesondere Zugriffe auf gestörte oder nicht vorhandene Einheiten, wie zum Beispiel:

- Zugriff auf ein Diskettenlaufwerk, welches keine Diskette enthält
- Zugriff auf einen Drucker, der gestört ist (Offline)
- Zugriff auf eine gesperrte Datei innerhalb eines Netzwerkes

In diesen oder ähnlichen Fällen wird ein INT 24 ausgelöst, der mit einer Fehlermeldung auf der Konsole reagiert. Meist wird dann die Abfrage:

```
Abort, Retry, or Ignore?
```

erscheinen. Durch eine entsprechende Antwort läßt sich die Reaktion auf diese Fehlermeldung steuern. Falls sich der Fehler nicht beheben läßt, wird die Funktion abgebrochen, um den DOS-Kommandointerpreter zu aktivieren.

6.3 Die FCB-orientierten Fehlermeldungen

Aus Gründen der Aufwärtskompatibilität zu den Versionen 1.x fallen einige ältere Funktionsaufrufe bei der Fehlerbehandlung aus der Reihe. Sie benutzen das AL-Register zur Fehler- und Statusanzeige. Beim Aufruf der CP/M-orientierten Funktionen wird noch mit File Control Blocks (FCB) gearbeitet. Die folgende Tabelle enthält eine Zusammenfassung der betroffenen Funktionen:

Code	Funktion
0f	Open File
10	Close File
11	Search For First Entry
12	Search For Next Entry
13	Delete File
16	Create File
17	Rename File
22	File Size
2b	Set Date (Nicht Über Fcb)
2d	Set Time (Nicht Über Fcb)
33	Control-c Check (Nicht Über Fcb)

Tabelle 6.2: Funktionen mit Fehlermeldungen im AL-Register

Tritt ein Fehler auf, wird im Register AL der Wert 0FFH zurückgegeben. Andernfalls ist das Register AL gelöscht.

6.4 Die handleorientierten Fehlermeldungen

Ab DOS 2.x wurde eine einheitliche Fehlerbehandlung und Signalisierung für die Funktionsaufrufe eingeführt. Sie betrifft im wesentlichen die ab DOS 2.x neu eingeführten handleorientierten Aufrufe. Ausgenommen bleiben die bereits oben aufgeführten Funktionen, die aus Kompatibilitätsgründen eine eigene Fehlerbehandlung durchführen. Tritt bei den neuen Funktionen ein Fehler auf, wird dies durch ein gesetztes Carry-Flag signalisiert. Im Register AX findet sich dann der entsprechende Fehlercode, der Hinweise

auf die Fehlerursache gibt. Die nachfolgende Tabelle gibt die in DOS implementierten Fehlercodes wieder. Die Fehlercodes sind als Dezimalwerte aufgeführt.

Code	Fehlerart
1	Ungültige Funktionsnummer
2	Datei nicht gefunden
3	Pfad nicht gefunden
4	Zu viele offene Dateien (keine freien Handles)
5	Zugriff abgewiesen
6	Ungültiger Handle
7	Memory Control Blocks zerstört
8	Nicht genügend Speicher frei
9	Ungültige Speicherblockadresse (Segment)
10	Ungültiges Environment
11	Ungültiges Format der Parameterblocks
12	Ungültiger Zugriffscode
13	Ungültige Daten
14	Reserviert
15	Ungültige Laufwerksangabe (nicht vorhanden)
16	Versuch, das aktuelle Inhaltsverzeichnis zu löschen
17	Nicht die gleiche Einheit (Rename)
18	Keine weiteren Dateien
---	Ab Dos 3.0 ---
19	Schreibversuch auf eine schreibgeschützte Diskette
20	Unbekannte Einheit
21	Laufwerk/gerät nicht bereit
22	Unbekanntes Kommando
23	Datenfehler (Crc-fehler)
24	Fehlerhafte Länge der Aufrufstruktur (Sektorlänge)
25	Seek-fehler (Spur nicht gefunden)
26	Unbekannter Typ des Speichermediums
27	Sektor nicht gefunden
28	Papierendemeldung vom Drucker
29	Schreibfehler
30	Lesefehler
31	Allgemeiner Fehler
32	Verletzung des Zugriffsrechts bei File Sharing (File Lock)
33	Verletzung des Zugriffsrechts bei File Lock (Record Lock)
34	Wechsel des Laufwerks nicht zulässig
35	Keine Fcb Verfügbar
36	Überlauf des Sharing-puffers
37	Code Page Mismatch (ab DOS 4.0)
38	Dateiende erreicht (ab DOS 4.0)
39	Speichermedium voll (ab DOS 4.0)
40	Reserviert
.	.
49	.
50	Funktion wird durch den Netzwerkdienst nicht unterstützt
51	Netzwerkknoten (Remote) empfängt keine Nachrichten

Tabelle 6.3: MS-DOS-Fehlercodes

Code	Fehlerart
52	Name innerhalb des Netzwerkes mehrfach vorhanden
53	Name innerhalb des Netzwerkes nicht gefunden
54	Netzwerk belegt
55	Netzwerkeinheit existiert nicht mehr
56	BIOS-Aufruf Parameterzahl im Netzwerk überschritten
57	Hardwarefehler im Netzwerkadapter
58	Unkorrekte Antwort des Netzwerkes
59	Unbekannter Netzwerkfehler
60	Unkompatibler (remote) Adapter im Netzwerk
61	Ausgabequeue Printer voll
62	Nicht genügend Speicher für die Print-Datei
63	Print-Datei ist gelöscht/abgebrochen
64	Netzwerkname ist gelöscht
65	Zugriff abgewiesen
66	Type der Netzwerkeinheit ungültig
67	Netzwerkname nicht gefunden
68	Netzwerkname zu lang
69	BIOS-Grenzwert für die Sessionzahl überschritten
70	SHARE: Einheit befindet sich temporär im Pause-Mode
71	Netzwerkaufruf nicht akzeptiert
72	Printer- oder Diskumleitung im Pause-Mode
73	reserviert
.	.
79	.
80	Datei existiert bereits
81	FCB existiert bereits
82	Eintrag im Inhaltsverzeichnis nicht möglich
83	Fehler innerhalb des INT 24
84	Zu viele Umleitungen (ab DOS 3.3)
85	Mehrfachumleitungen auf diese Einheit (ab DOS 3.3)
86	Ungültiges Paßwort (ab DOS 3.3)
87	Ungültiger Parameter (ab DOS 3.3)
88	Netzwerk-Schreibfehler (ab DOS 3.3)
89	Funktion wird nicht im Netzwerk unterstützt (DOS 4.0)
90	Benötigte Systemkomponente nicht installiert (DOS 4.0)

Tabelle 6.3: MS-DOS-Fehlercodes (Ende)

Die DOS-Version 2.x unterstützt nur die Fehlercodes 1 bis 18, während handleorientierte Aufrufe innerhalb des INT 24-Handlers nur die Codes 1 bis 12 liefern. Um die Codes in der Tabelle zu identifizieren, ist der Wert 18 zu addieren, da der INT 24 die Fehlercodes 1 bis 12 auf die DOS-Fehlercodes 19 bis 31 abbildet. Ab DOS 3.x sind dann alle die oben aufgeführten Codes implementiert. Im Rahmen der Beschreibung der INT 21-Funktionsaufrufe ist meist auch angegeben, welche Fehlercodes zurückgegeben werden. Allerdings sind diese Angaben nicht immer vollständig, so daß in der Praxis bei unterschiedlichen DOS-Versionen weitere Fehlercodes auftreten können.

6.5 Erweiterte Fehlercodes

Ab Version 3.x bietet DOS die Möglichkeit, erweiterte Fehlercodes mittels der INT 21-Funktion 59H (Get Extended Error Code) abzufragen. Sobald das Carry-Flag gesetzt ist, bietet diese Funktion weitere Informationen hinsichtlich der Fehlerart. Die erweiterten Fehlercodes wurden eingeführt, um eine einheitliche Fehlerbehandlung zu ermöglichen.

Die erweiterten Fehlercodes lassen sich nur einmalig abfragen. In jeder neuen DOS-Version werden weitere Fehlercodes implementiert. Ein bereits bestehendes Programm kann auch mit neueren DOS-Versionen arbeiten, da die alten Fehlermeldungen erhalten bleiben. Um eine zukünftige Erweiterung der Funktionalität zu ermöglichen, wird beim Aufruf im Register BX ein Versionscode übergeben. In den DOS-Versionen 3.0 bis 3.3 ist der Wert immer null.

In den Registern AX, BX und CX wird dann ein erweiterter Fehlercode zurückgegeben, der genauere Hinweise auf die Fehlerursache und deren Beseitigung gibt.

Im Register AX findet sich ein erweiterter Fehlercode. Die Funktion versucht hier die Fehlermeldungen der handleorientierten DOS-Versionen abzubilden. Der übergebene Wert entspricht den üblicherweise im Register AL übergebenen Fehlercodes. Falls dieser Code nicht mehr existiert, wird die am besten passende Fehlernummer zugeordnet.

Im Register BH findet sich ein Code, der die Fehlerklasse beschreibt. Es gilt folgende Nomenklatur:

```
 Code ° Fehlerklasse
------é-------------------------------------------------------------------
  01  ° Fehlende Resourcen (z.B. kein Speicher mehr frei, oder alle Kanäle belegt)
  02  ° Hierbei handelt sich sich nicht um einen Fehler, sondern um eine temporäre
      ° Situation, die einen einwandfreien Ablauf verhindert (z.B. Zugriffsversuch
      ° auf einen Locked-Bereich einer Netzwerkdatei)
  03  ° Fehlende oder falsche Autorisierung (z.B. kein Create Access,
      ° um eine Datei zu löschen)
  04  ° Fehlerursache in der internen Systemsoftware (Plausibilität verletzt)
  05  ° Hardwarefehler
  06  ° Fehler in der Systemsoftware, der nicht durch den laufenden Prozeß
      ° bewirkt wurde (z.B. fehlende oder falsche Konfigurationsdateien)
  07  ° Fehler im Anwendungsprogramm
  08  ° Datei oder Einheit nicht gefunden
  09  ° Datei oder Einheit besitzen ein falsches Format oder Typ
  10  ° Datei oder Einheit blockiert (locked)
  11  ° Falsche Diskette im Laufwerk, oder die Diskette ist beschädigt, oder
      ° sonstige Probleme mit dem Speichermedium
  12  ° Spezifizierter Name existiert bereits (z.B. Versuch, einen Namen innerhalb
      ° eines Netzwerkes mehrfach zu vergeben)
  13  ° Fehlerursache kann nicht entschlüsselt werden
  14  ° Operation läßt sich nicht ausführen (ab DOS 5.0)
  15  ° Time Out (ab DOS 5.0)
```

Diese Hinweise lassen sich dann in eine entsprechende Fehlermeldung umsetzen. Als weitere Hilfestellung liefert die Funktion einen Code im Register BL, der Hinweise zur weiteren Vorgehensweise enthält. Es gilt folgende Kodierung:

```
 Code ° Aktion
------é-------------------------------------------------------------------------
  01  ° Wiederhole Versuch und gebe dann einen Hinweis an den Benutzer
  02  ° Wiederhole den Versuch nach einer Pause
  03  ° Falls der Anwender Datei-, Pfad- oder Laufwerksnamen eingegeben hat,
      ° wiederhole die Abfrage
  04  ° Setze die benutzten Resourcen zurück und beende das Programm
      ° (z.B. Dateien schließen)
  05  ° Das System ist so stark gestört, daß das Programm sofort zu beenden ist
      ° (Dateien sollten nicht mehr geschlossen werden)
  06  ° Der gemeldete Fehler dient nur zur Information
  07  ° Fehlermeldung mit Abfrage an den Benutzer (z.B. Disketten einlegen),
      ° dann wiederhole den Versuch noch einmal
```

Als weitere Information findet sich im Register CH ein Hinweis zur Lokalisierung des Fehlers. Es gelten folgende Konventionen:

```
 Code ° Bemerkung
------é-------------------------------------------------------------------------
  01  ° unbekannt
  02  ° Der Fehler trat bei einem wahlfreien Zugriff auf eine blockorientierte
      ° Einheit (z.B. Diskette) auf
  03  ° Der Fehler trat im Netzwerk auf
  04  ° Der Fehler trat bei einem sequentiellen Zugriff auf eine zeichenorientierte
      ° Einheit auf (z.B. Printer nicht bereit)
  05  ° Der Fehler trat bei einem Zugriff auf den RAM-Speicher auf
```

Die erweiterten Fehlercodes sollten nur innerhalb folgender Konstellationen benutzt werden:

- Innerhalb eines INT 24-Handlers, um auch Netzwerkfehler abzufangen.
- Bei Verwendung von INT 21-Funktionsaufrufen, falls das Carry-Flag gesetzt ist und der Fehlercode im Register AX zurückgegeben wird.
- Bei FCB-Funktionsaufrufen, die als Fehler den Wert FFH im AL-Register zurückgeben.

An Hand der zurückgegebenen Werte, insbesondere im BL-Register, läßt sich dann die weitere Vorgehensweise bestimmen.

6.6 Funktionen mit abweichenden Fehlercodes

Bei einigen INT 21-Funktionen weichen die zurückgegebenen Fehlercodes von der allgemeinen Fehlerdefinition ab. Die nachfolgende Tabelle enthält eine Aufstellung dieser Funktionen:

```
Ö------Ú-------------------------------------------------------Ì
° Code ° Funktion                                              °
û------é-------------------------------------------------------À
°  14  ° Sequential Read                                       °
û------é-------------------------------------------------------À
°  15  ° Sequential Write                                      °
û------é-------------------------------------------------------À
°  21  ° Random Read                                           °
û------é-------------------------------------------------------À
°  22  ° Random Write                                          °
û------é-------------------------------------------------------À
°  27  ° Random Block Read                                     °
û------é-------------------------------------------------------À
°  28  ° Random Block Write                                    °
û------é-------------------------------------------------------À
°  29  ° Parse Filename                                        °
Û------Ù-------------------------------------------------------ì
```

Tabelle 6.4: Funktionen mit abweichenden Fehlercodes

Nähere Informationen sind den jeweiligen Funktionsbeschreibungen zu entnehmen.

7 DOS-Memory-Manager und DOS-Datenstrukturen

Im Kapitel über die MS-DOS-Speicheraufteilung wurde das Speicherabbild des 640-Kbyte-RAM-Bereiches vorgestellt. Die relative Lage einzelner Bereiche (Puffer, Einheitentreiber, Kommandointerpreter, Anwenderprogramme) zueinander ist zwar bekannt, nicht aber ihre absoluten Anfangsadressen. DOS nimmt erst zur Laufzeit die Aufteilung des RAM-Bereiches vor. Dies hat natürlich Konsequenzen hinsichtlich Speicherverwaltung und Programmablauf. Da zum Beispiel der residente Teil des Kommandointerpreters (COMMAND.COM) oberhalb des DOS-Pufferbereiches liegt, sind Lage und Größe der Puffer bereits beim Systemstart festzulegen. Dies erfolgt mittels der Steuerdatei CONFIG.SYS, die einmalig durch das Programm IO.SYS (IBMBIO.COM) ausgeführt wird.

Nun muß DOS aber auch für eigene Bedürfnisse in der Lage sein, die Adressen einzelner Bereiche festzustellen. Für diesen Zweck besitzt MS-DOS einen eigenen Speichermanager (Memory-Manager) zur dynamischen Verwaltung. Prozesse können ihren aktuellen Speicherbedarf dem Memory-Manager über folgende INT 21-Aufrufe mitteilen:

```
48h  Allocate Memory
49h  Free Allocated Memory
4ah  Modify Allocated Memory
```

Bei anderen Systemaufrufen (z.B. 4BH Load and Execute a Programm) werden die Speicheranforderungen implizit durch die Funktion aktiviert. Der Memory-Manager versucht die jeweilige Anforderung zu befriedigen, wobei der Speicher in Blöcke zu je 16 Byte (Paragraphen) unterteilt wird, d.h. es kann immer nur ein Block, oder ein Vielfaches davon, angefordert oder freigegeben werden.

7.1 Die Speicherverwaltungsstrategie des Memory-Managers

Werden von Programmen und Treibern dynamische Speicherblöcke angefordert und wieder freigegeben (z.B. Laden von Subprozessen und residenten Programmen, Freigabe des PSP und Environmentsegments etc.), dann führt dies mit der Zeit zu einer Fragmentierung des Speicherbereiches (Bild 7.1). Die freien Blöcke sind in BIld 7.1 mit Zahlen belegt, während der Rest bereits verfügt ist.

```
Ö-------Ú---Ú-Ú--------Ú--------Ú-Ú------Ú-Ú-------Ú--------
°   10  °###°1°########°   10   °#°  10  °#°   50  °#####...
Û-------Ù---Ù-Ù--------Ù--------Ù-Ù------Ù-Ù-------Ù--------
```

Bild 7.1: Fragmentierung des DOS-Speicherbereiches

Nun kann es vorkommen, daß eine Speicheranforderung von FFFH Blöcken (64 Kbyte) zurückgewiesen wird, obwohl vielleicht insgesamt noch 96 Kbyte Speicher frei sind. Dies liegt daran, daß einem Prozeß nur zusammenhängende Blöcke zugewiesen werden können.

Falls nun der größte zusammenhängende Block 7FFH Paragraphen (32 Kbyte) enthält, wird die Anforderung zurückgewiesen (Bild 7.2)

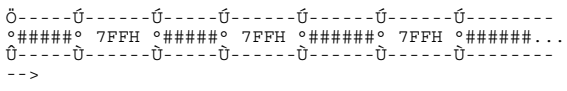


Bild 7.2: Abweisung einer Speicheranforderung

Die einzige Lösung bestände darin, die belegten Speicherbereiche zusammenzuschieben, um so einen größeren freien Block zu erhalten. Dies wird aber durch DOS nicht unterstützt. Es sind aus der Theorie aber verschiedene Strategien (first fit, best fit, last fit) zur optimaleren Freispeicherverwaltung bekannt. Diese sollen nachfolgend kurz beschrieben werden.

first fit

Es wird angenommen, daß ein fragmentierter Speicher mit folgender Struktur vorliegt:

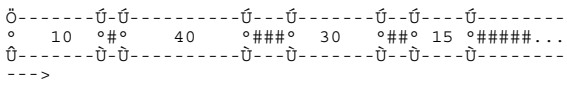


Bild 7.3: Speicherabbild vor einer Speicheranforderung (first fit)

Nun wird ein Speicherbereich von 15 Blöcken angefordert. Der Memory-Manager durchsucht den Speicher nach dem ersten zusammenhängenden Bereich, der mindestens der angeforderten Blockgröße entspricht. Der erste freie Bereich (10 Blöcke) ist zu klein, aber der nächste Freispeicher (40 Blöcke) reicht aus. Es ergibt sich anschließend folgendes Bild:

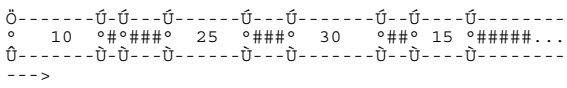


Bild 7.4: Speicherabbild nach einer Speicheranforderung (first fit)

Von den 40 freien Blöcken wurden 15 Blöcke neu belegt. Eine weitere Anforderung von 40 Blöcken kann nun nicht mehr befriedigt werden, obwohl insgesamt noch 80 Blöcke frei sind. Offensichtlich hat in diesem Fall die Strategie versagt, da der Speicher bei Verwendung des letzten freien Bereiches (15 Blöcke) besser ausgenutzt würde. Die *First-fit-Methode* besitzt den Vorteil, daß ein Speicherblock im Mittel recht schnell zugeordnet wird, da nicht der gesamte Speicherraum durchsucht werden muß. Die Methode empfiehlt sich, wenn die freien Speicherbereiche annähernd gleiche Größen besitzen.

best fit

Bei oben diskutierter Strategie trat der Fall auf, daß große freie Speicherbereiche durch die Anforderungen nur teilweise belegt werden. Dies führte dazu, daß letztlich zwar in der Summe ein großer freier Speicherbereich vorlag, dieser aber in viele kleine Teile fraktioniert wurde. Damit ist das System nach einer gewissen Zeit nicht mehr in der Lage, größere Speicherplatzanforderungen zu erfüllen. Die Anfragen müssen also abgewiesen werden, obwohl die Summe der freien Bereiche die angeforderte Größe vielleicht übersteigt. Naheliegend ist daher die Überlegung, den angeforderten Bereich nicht in das erste passende Segment zu legen, sondern ein optimales freies Segment auszuwählen. Bei der *Best-fit-Strategie* wird dieser Ansatz benutzt. Es sei zum Beispiel folgende Konfigurierung gegeben.

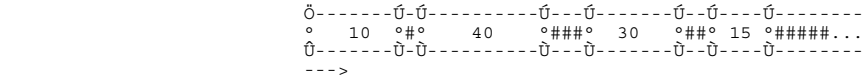


Bild 7.5: Speicherabbild vor einer Speicheranforderung (best fit)

Nun wird ein Speicherbereich von 14 Blöcken angefordert. Anschließend liegt folgende Verteilung vor.

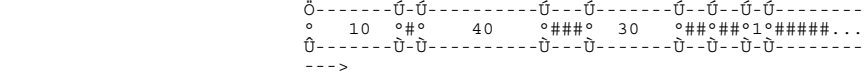


Bild 7.6: Speicherabbild nach einer Speicheranforderung (best fit)

Die Darstellung zeigt, daß bei der Anforderung der am besten passende Bereich belegt wurde. Es wird lediglich 1 Block »verschenkt« Die Best-fit-Strategie stellt also sicher, daß keine großen zusammenhängenden Bereiche fragmentiert werden, solange noch passende kleinere Lücken vorliegen. Die .i.Speicherausnutzung; ist also ökonomischer. Dies wird allerdings mit dem Nachteil erkauft,»daß der gesamte Speicher nach freien Lücken durchsucht werden muß. Nur wenn eine Lücke gefunden wird, deren Größe mit der angeforderten Speichergröße übereinstimmt, kann die Suche vorher abgebrochen werden, da dann das Optimum bereits erreicht wurde. Weiterhin sind die freibleibenden Restsegmente sehr klein, so daß eine Belegung durch andere Programme kaum möglich wird. Dies kann bei bestimmten Applikationen durchaus unerwünscht sein, so daß eine andere Strategie günstiger ist.

last fit

Als letzte Strategie soll die *Last-fit-Methode* besprochen werden. Hier wird versucht, den letzten passenden Speicherbereich zu belegen. Es sei zum Beispiel folgende Konfiguration gegeben.

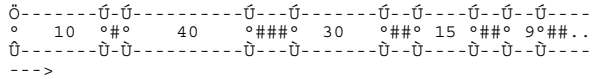


Bild 7.7: Speicherabbild vor einer Speicheranforderung (last fit)

Eine Speicheranforderung von 10 Blöcken ergibt dann folgendes Bild:

Bild 7.8: Speicherabbild nach einer Speicheranforderung (last fit)

In diesem Fall wurde zwar nicht die optimale Belegung gefunden, da ja der erste freie Bereich genau 10 Blöcke enthält. Aber das Ziel dieser Strategie ist es ja, zuerst den oberen Speicherbereich zu belegen. Dies kann zum Beispiel erwünscht sein, wenn Anwenderprogramme geladen werden, die sich oberhalb des residenten Teils des DOS-Kommandointerpreters (COMMAND.COM) befinden sollen, während die Systemprogramme und Treiber im unteren Speicherbereich liegen. Als Nachteil wirkt sich einmal die Tatsache aus, daß der gesamte freie Speicher durchsucht werden muß, es sei denn, es liegt eine doppelt verkettete Liste der freien Blöcke vor. Dann kann die Analyse mit dem letzten Block beginnen. MS-DOS unterstützt diese doppelt verketteten Listen allerdings nicht, so daß die Suche nur von der unteren Speichergrenze beginnt. Weiterhin ist wie bei der First-fit-Methode keine optimale Speicherverwaltung gewährleistet.

Ab DOS 5.0 sind weitere Strategien zur Zuweisung des Speichers implementiert. Auf 80386-Systemen kann der Bereich zwischen 640 Kbyte und 1 Mbyte mit Adaptern und RAM-Speicher belegt werden. Diese als Upper Memory Blocks bezeichneten Bereiche lassen sich für Programme nutzen. Die Strategien wurden deshalb ab DOS 5.0 auf die Suche in diesen Blöcken ausgeweitet (first fit UMB first, best fit UMB first, last fit UMB first). Erst wenn kein freier Speicher im UMB gefunden wird, sucht DOS den 640-Kbyte-Bereich durch.

Im DOS-Memory-Manager sind alle beschriebenen Freispeicherverwaltungsmodelle implementiert. Standardmäßig wird die First-fit-Methode eingestellt. Mittels der undokumentierten INT 21-Funktion 5800H (Get-Allocation-Strategie) läßt sich der eingestellte Mode abfragen. Im AX-Register wird die Kodierung:

```
0  First fit
1  Best fit
2  Last fit
```

zurückgegeben. Der Funktionsaufruf 5801H (Set-Allocation-Strategie) erlaubt die Modifizierung dieser Einstellung über einen entsprechenden Codewert im BX-Register. Der Systemaufruf wird genauer im Kapitel über die INT 21-Funktionen besprochen.

7.2 Die Memory-Control-Blocks (MCB)

Da der interne Arbeitsspeicher von DOS in Blöcke zu je 16 Byte aufgeteilt wird, muß eine Verwaltung dieses Speichers durch den DOS-Memory-Manager erfolgen. Woran erkennt dieser nun, welche Blöcke belegt sind und welche Informationen dort abgelegt wurden. Die einfachste Möglichkeit bestände darin, eine Tabelle mit je einem Eintrag für jeden Paragraphen (16 Byte) im Speicher anzulegen. Bei einem Speicher von 1 Mbyte bedeutet dies aber, daß alleine die Tabelle bereits 8 Kbyte (1 Bit pro Paragraph) oder mindestens 64 Kbyte (1 Byte pro Paragraph) belegt. Dort sind dann noch keine Informationen über die Art der Daten im belegten Block vorhanden.

DOS geht daher einen anderen Weg, indem es über Zeigerstrukturen und verkettete Listen die jeweiligen Blöcke markiert. Hierzu wird vor jedem belegten oder freien Speicherbereich eine 16 Byte großen Datenstruktur (1 Paragraph) generiert. Diese wird als Memory-Control-Block (MCB) bezeichnet und enthält Informationen über die Zahl der belegten Paragraphen, der Lage des folgenden MCB, dem Eigentümer des Blocks, etc. Der Aufbau des MCB ist folgender Tabelle zu entnehmen (erst ab DOS 5.0 dokumentiert).

Offset	Bytes	Feld
00	1	Identifikationszeichen ('M'oder 'Z')
01	2	Segmentadresse des nächsten PSP
03	2	Zahl der belegten Paragraphen
05	11	reserviert für künftige Erweiterungen

Tabelle 7.1: Aufbau des Memory-Control-Blocks (MCB).

Im ersten Byte finden sich die Zeichen 'M' (4DH) und 'Z' (5AH). Sie dienen zur Prüfung, ob es sich wirklich um einen MCB handelt. Beginnend vom ersten MCB bis zum vorletzten MCB wird das Zeichen 'M' eingetragen. Das Zeichen 'Z' markiert den letzten MCB in der Kette. Die Buchstaben entsprechen den Initialen des DOS-Entwicklers Mark Zbikovski, der sich hier ein heimliches Denkmal gesetzt hat.

Das übernächste Wort (Offset 03H) enthält die Zahl der durch den Block belegten Paragraphen. Hierzu zählt auch der Paragraph (16 Byte) des MCB. Wird zu diesem Wert eine 1 addiert, ist dies die Segmentadresse des folgenden MCB. Da die MCB's immer an Segmentgrenzen liegen, ist der Offset gleich null.

Im ersten Wort (Offset 01H) findet sich ein Zeiger (Segmentadresse) auf das zugeordnete Program-Segment-Prefix (PSP). Dieser Speicherbereich bildet den Vorspann zu einem ausführbaren Programm und wird auf den nachfolgenden Seiten näher beschrieben. Es kann vorkommen, daß mehrere MCB's einen Zeiger auf das gleiche PSP enthalten. Dies ist erklärbar, da die DOS-EXEC-Funktion beim Erzeugen eines Subprozesses sowohl einen Programmlaufbereich mit vorgeschaltetem PSP reserviert als auch einen Environmentbereich. Jedem dieser Bereiche wird ein MCB vorangestellt, der einen Zeiger auf dieses PSP enthält.

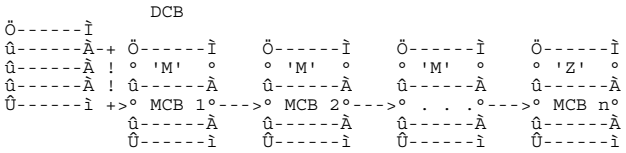


Bild 7.9: Die DOS-Memory-Control-Block-Kette

Zur Verwaltung des Speichers bietet DOS die drei INT 21-Funktionen:

```
48H  Allocate Memory
49H  Free Allocated Memory
4AH  Set Block
```

Alle diese Aufrufe verändern den Speicher in seiner Struktur. Wird zum Beispiel ein Bereich mit 48H reserviert, legt DOS einen MCB an, der diesen Block markiert. Wird ein bestehender Block durch die Funktion *Set Block* (4AH) verkleinert, dann legt DOS einen zweiten MCB für den freien Block an. Die Informationen im MCB des belegten Blocks werden entsprechend justiert. Freie und belegte MCB's werden ab Offset 01H mit der PSP-Adresse markiert. Wird ein Block durch den Memory-Manager freigegeben, setzt er diesen Eintrag auf den Wert 0000H zurück. Die Länge des freien Bereiches (in Paragraphen) wird ab Offset 03H vermerkt. Die Umrechnung in Byte erfolgt mit der Formel:

```
Länge in Byte = (Länge in Paragraphen -1) * 16
```

Alle Einträge ungleich 0000H ab Offset 01H signalisieren, daß der Block belegt ist. Bei einzelnen MCB's findet sich dort aber der Wert 0008H. Es ist aber unwahrscheinlich, daß auf Adresse 0008:0000 ein MCB liegt, da dieser Bereich für die Interruptvektor-Tabelle reserviert wurde. Der Wert 0008H ab Offset 01H signalisiert deshalb, daß der MCB durch das System belegt wurde. Dieses benötigt ja auch Speicherplatz für seine internen Tabellen, Buffer und Treiber etc. Alle diese Bereiche besitzen aber kein PSP-Segment, so daß DOS die Blöcke mit 0008H markiert.

Die restlichen Byte ab Offset 05H bis Offset 0FH im Kopf des MCB bleiben bis DOS 3.3 unbelegt und sind für zukünftige Erweiterungen reserviert.

Ab DOS 3.1 enthält der erste MCB das DOS-Datensegment mit den Treibern, Puffern, etc. Der erste MCB wird in allen DOS-Versionen über einen Zeiger aus dem DOS-Control-Block (DCB) adressiert (siehe folgender Abschnitt).

Änderungen ab DOS 4.0

Ab der Version 4.0 wurden einige nicht dokumentierte Änderungen an der Struktur der MCB vorgenommen. Der Kopf besteht nach wie vor aus einer 16 Byte (1 Paragraph) langen Tabelle mit folgender Struktur:

Offset	Bytes	Feld
00	1	Identifikationszeichen ('M'oder 'Z')
01	2	Segmentadresse des nächsten PSP
03	2	Zahl der belegten Paragraphen
05	3	reserviert für künftige Erweiterungen
08	8	ASCIIZ-String mit dem Namen des Owners

Tabelle 7.2: Aufbau des Memory-Control-Blocks (MCB) ab DOS 4.0

Die Einträge ab Offset 0 bis Offset 07H bleiben gegenüber den älteren Versionen gleich. Die MCB-Kette wird nach wie vor mit den Zeichen 'M' und 'Z' signiert. Die drei Byte zwischen Offset 05H und 07H bleiben unbelegt.

Ab Offset 08H legt DOS den Namen des zugehörigen Prozesses als ASCIIZ-String ab. Dieser Name wird aus den ersten 8 Buchstaben des Filenamens gebildet, aus dem der Prozeß geladen wurde. Falls der Name kürzer als 8 Zeichen ist, endet er mit einem Nullbyte (ASCII-Zero-String).

Eine weitere Änderung ergibt sich ab DOS 4.0 bei der Belegung der System-MCB-Bereiche. Bis zur Version 3.3 legte DOS nur einen MCB für das System an und kombinierte alle Treiber, Puffer und Tabellen zusammen in diesem Block. Ab der Version 4.0 wird der reservierte MCB-Bereich nochmals in einzelne Unterblöcke unterteilt, in denen dann jeweils ein Treiber, ein Buffer, ein Stack, etc. untergebracht wird. Das Feld ab Offset 01H im MCB wird dann mit dem Wert 0008H signiert (Systemblock). An den MCB schließt sich dann eine 16 Byte lange Datenstruktur (Sub-Control-Block) an, die den folgenden Unterblock beschreibt. Jeder Teilblock ist mit einem Sub-Control-Block SBC markiert. Damit ergibt sich innerhalb des System-MCB eine Struktur gemäß folgendem Bild.

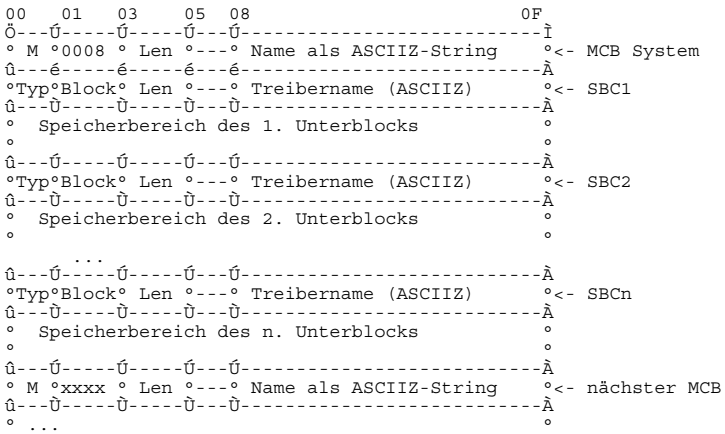


Bild 7.10: Struktur eines System-Memory-Control-Blocks mit Sub-Control-Blocks ab DOS 4.0.

Mit dem MCB wird vom System ein genügend großer Speicherbereich reserviert. Dann teilt DOS diesen Bereich durch die SBC's in einzelne Teile, die jeweils die Treiber, Puffer,

Stacks etc. aufnehmen. Jeder SBC beginnt mit einem eigenen Kennbuchstaben, der den Typ des Unterblocks festlegt. Die folgende Tabelle enthält die Kodierung der einzelnen Typen.

Offset	Bytes	Feld
00	1	Typ des SBC's
		42H = B Buffer für DISK-I/O (BUFFERS=XX)
		43H = C BUFFERS EMS-Arbeitsbereich
		44H = D Gerätetreiber (DEVICE=XX)
		45H = E Device Treiber Anhang
		46H = F Handle Tabelle (FILES=XX)
		49H = I IFS-Treiber
		4CH = L Laufwerksumleitung (SUBST/JOIN)
		4DH = M nächster MCB -> Ende SBC
		53H = S Stack (STACKS=XX)
		54H = T INSTALL = transienter Codebereich
		58H = X FCB-Tabelle (FCBS=XX)
		5AH = Z letzter MCB -> Ende SBC
01	2	Zeiger auf Segment Unterblock (Owner)
03	2	Länge des SBC in Paragraphen
05	3	reserviert
08	8	Treibername des Unterblocks für D oder I

Tabelle 7.3 Struktur eines SBC-Blocks (DOS 4.x)

Die Struktur lehnt sich stark an den Aufbau der MCB's an. Das erste Byte enthält einen Buchstaben, der den Typ des Sub-Control-Blocks spezifiziert. 'B' steht dabei zum Beispiel für die DOS-Disk-Buffer, über die die Daten-Ein-/Ausgabe abgewickelt wird. Die Größe des Bufferbereiches läßt sich durch die Anweisung:

```
BUFFERS;=xx
```

in der Datei CONFIG.SYS einstellen. Der Aufbau des Bufferbereichs wird in einem eigenen Abschnitt beschrieben. Mit dem Buchstaben 'D' wird ein Codebereich mit einem DOS-Treiber markiert. Diese Treiber lassen sich in CONFIG.SYS mit der Anweisung:

```
DEVICE=...
```

installieren. Bei Gerätetreibern wird gleichzeitig der Dateiname ab Offset 08H im SBC abgelegt. Fehlende Zeichen werden durch Nullbytes (00H) ersetzt. Die Handle-Tabelle läßt sich in ihrer Größe durch die Anweisung:

```
FILES=xx
```

in CONFIG.SYS beeinflussen. Der dazugehörige SBC enthält den Buchstaben 'F' im Typfeld. Die Tabelle mit den Laufwerksumleitungen wird durch ein 'L' im SBC markiert. Die Einträge werden durch die Befehle SUBST und JOIN verwaltet. DOS legt für die Bearbeitung der Hardwareinterrupts eigene Stackbereiche an. Deren Zahl läßt sich in CONFIG.SYS durch die Anweisung:

```
STACKS=xx
```

einstellen. Im SBC erscheint der Kennbuchstabe 'S' für den Unterblock. Das Zeichen 'X' markiert den Bereich mit den File-Control-Blocks (FCB). Die Tabellen werden aber nur angelegt, falls in CONFIG.SYS die Anweisung:

```
FCBS=xx
```

steht. 'I' steht als Markierung für Daten eines installierbaren Dateisystems (IFS). Mit 'T' wird ein Daten-/Codebereich markiert, in dem transiente Teile eines Treibers mit INSTALL = geladen wurden. In DOS 4.x lassen sich Daten für den EMS-Puffer mit 'C' markieren.

Das Ende des Systembereiches mit den SBC's wird durch den nächsten regulären MCB markiert. Im Feld mit dem Typ des Blocks tauchen dann die Zeichen 'M'oder 'Z' auf.

Ab Offset 01H enthält der SBC einen 16-Bit-Zeiger auf das Anfangssegment des zugehörigen Unterblocks. Hierbei handelt es sich in der Regel immer um den folgenden Unterblock, der im nächsten Paragraphen beginnt.

Der 16 Bit Wert ab Offset 03H gibt die Länge des SBC's in Paragraphen an. Hier sind auch die 16 Byte des SBC-Kopfes mit einzurechnen. Der Bereich ab Offset 05H bis einschließlich 07H ist für zukünftige Erweiterungen reserviert. Das gleiche gilt für die 8 Byte ab Offset 08H, falls dort kein Treibername eingetragen ist.

Änderungen ab DOS 5.0

In DOS 5.0 wurde die Struktur für die Systemblöcke aus DOS 4.x übernommen. Trotzdem ergeben sich einige Änderungen.

Gehört ein MCB zu DOS, wird im MCB in den Bytes ab Offset 08H eine Signatur an Stelle des Treibernamens eingetragen. SC steht dabei für Bereiche mit Systemcode oder belegte UMB's (z.B. ROM). SD markiert Systemdaten oder Einheitentreiber.

```
Weiterhin kann der UMB-Adressbereich durch DOS verwaltet werden. Hier nutzt DOS
ebenfalls MCB's zur Markierung. Der erste Block kann auf Adresse A000:0000H liegen
(EGA-Video-RAM-Bereich). Soll der UMB mit genutzt werden, wird im letzten MCB im
konventionellen 640-Kbyte-Bereich der Buchstabe »Z»Fehler! Verweisquelle konnte nicht
gefunden werden.MÖ--------Ű-------Ű-----------------------------------------------Ì
º Offset º Bytes º Feld                                                 º
û--------é-------é-----------------------------------------------------À
º   00   º   1   º Identifikationszeichen ('M'oder 'Z')                  º
º   01   º   2   º Segment PSP (0000 frei, 0008 DOS Block)               º
º   03   º   2   º Zahl der belegten Paragraphen                         º
º   05   º   3   º reserviert für künftige Erweiterungen                 º
º   08   º   8   º Text 'UMB' 1. UMB-Block, 'SM' letzter UMB-Blckº
Û--------Ù-------Ù-----------------------------------------------------ì
```

Tabelle 7.4: Aufbau des UMB-Memory-Control-Blocks (MCB) ab DOS 5.0

Die Belegung des UMB erfolgt jedoch nur, falls in CONFIG.SYS die Anweisung DOS=UMB vorhanden ist. Der Zugriff auf den 1. MCB im Speicher erfolgt über die nachfolgend beschriebene Speicherstruktur.

7.3 Struktur und Aufbau des DOS-Control-Blocks (DCB)

Der DOS-Memory-Manager verwaltet einen internen Datenbereich mit Zeigern auf die MCB-Kette und andere Listen und kann damit alle Informationen über den internen Speicheraufbau rekonstruieren. Da jede Liste einen Eingangszeiger auf den Listenkopf benötigt, besitzt MS-DOS eine undokumentierte interne Tabelle, in der die Zeiger auf die jeweiligen Listenköpfe abgespeichert sind. Diese Datenstruktur wird nachfolgend als *DOS-Control-Block* (DCB) bezeichnet. Eine andere häufig gebrauchte Bezeichnung lautet *List of Lists*. Über den undokumentierten INT 21-Funktionsaufruf 52H läßt sich die Lage des DCB ermitteln (Beschreibung im Kapitel über die INT 21-Funktionen). Im Registerpaar ES:BX wird ein Zeiger auf diesen internen Datenbereich zurückgegeben. Der Aufbau dieser Datenstruktur ist ebenfalls undokumentiert und variiert in Abhängigkeit von der DOS-Version. Für die DOS-Versionen ab 2.x gilt folgende Belegung:

Offset	Bytes	Feld
-02H	2	Zeiger auf ersten Memory-Control-Block (MCB)
00H	4	Zeiger auf den Disk-Parameter-Blocks (DPB)
04H	4	Zeiger auf die Open-File-Tabelle
08H	4	Zeiger auf den Kopf des CLOCK$-Treibers
0CH	4	Zeiger auf den Kopf des Consol-Treibers
10H	1	Zahl der logischen Drives im System
11H	2	Bytes pro Sektor (Maximalwert)
13H	4	Zeiger auf den ersten DOS-Puffer

Tabelle 7.5: Aufbau des DOS-Control-Blocks (DCB) in DOS 2.x

Die Tabelle enthält mehrere 4-Byte-Zeiger auf verschiedene DOS-Strukturen. Ab Offset -02H ist zum Beispiel der Zeiger auf den ersten MCB-Block abgelegt. Ab Offset 00H ein 32-Bit-Zeiger (Segment:Offset) auf den DOS-Disk-Parameter-Block (DPB) gespeichert. Daran schließt sich der Zeiger auf die Liste mit den offenen Files an. Die Struktur der *Open File-Tabelle* ist in einem eigenen Abschnitt beschrieben. Ab Offset 8 finden sich zwei 4-Byte-Zeiger auf die CLOCK$- und CON-Treiber. Im Byte ab Offset 10H steht die Zahl der logischen Laufwerke, die durch das System unterstützt werden. Ab Offset 11H steht die maximal mögliche Sektorgröße (in Byte) des Systems als 16-Bit-Wert. Daran schließt sich ein 4-Byte-Zeiger auf den ersten logischen DOS-Disk-Buffer an. Diese Buffer lassen sich durch den Befehl:

```
BUFFERS=xx
```

in der Datei CONFIG.SYS zur Bootzeit installieren. Der DCB endet in DOS 2.x ab Offset 17H, da sich ab dieser Adresse der Code des ersten (NUL) Device-Treibers anschließt.

In verschiedenen DOS-Versionen wurde der Aufbau der Tabelle stark variiert, lediglich der oben beschriebene Kern blieb erhalten. Nachfolgende Tabelle enthält den Aufbau des DOS-Control-Blocks der Versionen 2.0 bis 6.0.

Offset	Bytes	Feld
-0CH	2	SHARE-Wiederholungszähler (DOS 3.1 - DOS 3.3) läßt per INT 21-Funktion 440B setzen
-0AH	2	SHARE-Verzögerungszähler (DOS 3.1 - DOS 3.3) läßt per INT 21-Funktion 440B setzen
-08H	4	Zeiger auf aktuellen Diskpuffer erst ab DOS 3.0 definiert
-04H	2	(ab DOS 3.0) Index auf den Puffer mit den ungelesenen Eingaben der CON-Einheit der Wert 0000 bedeutet keine Eingaben
-02H	2	Zeiger auf ersten Memory-Control-Block (MCB)
00H	4	Zeiger auf den DOS-Disk-Parameter-Block (DPB)
04H	4	Adresse der DOS System File Table
08H	4	Zeiger auf den Kopf des CLOCK$-Treibers
0CH	4	Zeiger auf den Kopf des Consol-Treibers
---	-	DOS 2.x Datenstruktur
10H	1	Anzahl log. Laufwerke im System
11H	2	max. Byte/Block der Block Treiber
13H	4	Zeiger auf den ersten Sektorpuffer
17H	18	Begin des NUL-Device Headers
---	-	DOS 3.0 Datenstruktur
10H	1	Zahl der Blockeinheiten
11H	2	max. Blockgröße für alle Blocktreiber
13H	4	Zeiger auf den ersten Diskpuffer
17H	4	Zeiger auf ein Feld mit den aktuellen Directory Strukturen (CDS)
1BH	1	LASTDRIVE Wert (meist 5)
1CH	4	Zeiger auf den STRING= Arbeitsbereich
20H	2	Wert x von STRING = X aus CONFIG.SYS
22H	4	Zeiger auf die FCB Tabelle
26H	2	Wert y aus FCBS = x,y in CONFIG.SYS
28H	18	Kopf des NUL-Einheitentreibers
---	-	DOS 3.1 - 3.3 Datenstruktur
10H	2	max. Blockgröße (Byte / Sektor) für alle Blocktreiber
12H	4	Zeiger auf den ersten Diskpuffer in der Kette
16H	4	Zeiger auf ein Feld mit den aktuellen Directory Strukturen (CDS)
1AH	4	Zeiger auf die FCB Tabellen
1EH	2	Anzahl der geschützten FCB's (FCBS=x,y)
20H	1	Anzahl der installierten Bockeinheiten
21H	1	Anzahl der Laufwerksbuchstaben
22H	18	Kopf des NUL-Einheitentreibers

---	-	DOS 4.x Datenstruktur
10H	2	max. Byte / Sektor der Blocktreiber
12H	4	Zeiger auf den Diskpuffer Info Record
16H	4	Zeiger auf die ein Feld mit den aktuellen Directory Strukturen
1AH	4	Zeiger auf die FCB Tabellen
1EH	2	Anzahl der geschützten FCB's
20H	1	Anzahl der installierten Blockeinheiten
21H	1	LASTDRIVE Wert (meist 5)
22H	18	Kopf des NUL-Einheitentreibers
34H	1	Zahl der mit JOIN umgeleiteten Drives
35H	2	Zeiger auf das IBMDOS Codesegment mit der Liste der Programmnamen/ -versionen
37H	4	Zeiger auf eine FAR-Routine mit dem residenten IFS-Funktionen, werden von IFS-Funktionen aktiviert, die die Funktionen 20H, 24H-28H nicht unterstützen.
3BH	4	Zeiger auf die Kette der IFS-Treiber
3FH	2	x-Wert von BUFFERS x,y wird auf ein mehrfaches von 30 gerundet, falls der Buffer im EMS liegt.
41H	2	y-Wert von BUFFERS x,y
43H	1	Boot Laufwerk (1=A:)
44H	1	CPU-Flag (01H 80386+)
45H	2	EMS-Größe in Kbyte
---	-	DOS 5.0/6.0 Datenstruktur
10H	2	max. Byte/Sektor der Blocktreiber
12H	4	Zeiger auf den Diskpuffer Info Record
16H	4	Zeiger auf die ein Feld mit den aktuellen Directory Strukturen
1AH	4	Zeiger auf die FCB Tabelle
1EH	2	Anzahl der geschützten FCB's
20H	1	Anzahl der installierten Blockeinheiten
21H	1	LASTDRIVE Wert (meist 5)
22H	18	Kopf des NUL-Einheitentreibers
34H	1	Zahl der mit JOIN umgeleiteten Drives
35H	2	0000H
37H	4	Zeiger auf die SETVER-Tabelle
3BH	2	unbekannt
3DH	2	unbekannt
3FH	2	x-Wert von BUFFERS x,y
41H	2	y-Wert von BUFFERS x,y
43H	1	Boot Laufwerk (1=A:)
44H	1	CPU-Flag (01H 80386+)

```
°  45H   °   2   ° EMS-Größe in Kbyte                              °
Û--------Ù-------Ù-------------------------------------------------ì
```

Tabelle 7.6: Aufbau des DOS-Control-Blocks

Vor der Tabelle steht immer ab Offset ES:BX-2 die Segmentadresse des ersten Memory-Control-Blocks (MCB). Ab Offset ES:BX-4 findet sich ab DOS 3.0 ein Zeiger auf das DOS-Codesegment mit dem ungelesenen CON-Input. Der Zeiger ist nur dann gültig, wenn die Eingaben per Handle gelesen werden. Ein Wert 0000H signalisiert, daß keine ungelesenen Daten vorliegen. Ab DOS 3.0 wurde die Struktur des DOS-Control-Blocks (DCB) erweitert. Die Einträge im Bereich ES:BX-4 bis ES:BX+0FH sind bei allen DOS-Versionen von 2.0 bis 4.01 gleich. Ab Offset 10H ergibt sich aber ab DOS 3.0 eine Änderung. Hier wurden Informationen über die internen DOS-Puffer eingefügt.

Einschränkend gilt auch, daß das Tabellenende ab Offset 12H bei PC-DOS erst ab Version 3.2 und bei MS-DOS ab Version 3.1 definiert ist.

Ab Offset 04H steht ein 4-Byte-Zeiger auf die interne DOS-Tabelle mit den offenen Dateien. Die Tabellenstruktur wird in einem eigenen Abschnitt behandelt.

Die Adressen der CLOCK$- und CON-Treiber finden sich wieder ab Offset 08H und 0CH.

Die Zahl der maximal unterstützten logischen Laufwerke wurde gegenüber der Version 2.0 auf Offset 20H verschoben. Daran schließt sich ab Offset 21H ein Byte an, welches die Zahl der aktuell eingestellten logischen Laufwerke enthält. Dieser Wert läßt sich in CONFIG.SYS durch den Befehl:

```
LASTDRIVE=xx
```

beeinflussen. Standardmäßig trägt DOS hier den Wert 5 ein.

Im Bereich zwischen Offset 10H und 20H befinden sich einige Informationen über die DOS-Disk-I/O-Operationen. Ab Offset 10H steht ab DOS 3.0 die maximale Größe eines durch das System unterstützten logischen Disksektors. Hierbei handelt es sich um einen 16-Bit-Wert, der durch DOS standardmäßig auf 512 Byte gesetzt wird. Einige Hersteller (z.B. COMPAC) haben diesen Wert jedoch erhöht, um in DOS 3.x mehr als 32 Mbyte pro Platte verwalten zu können. Der 4-Byte-Zeiger ab Offset 12H enthält die Adresse des ersten logischen DOS-Sektor-Buffers für den Zugriff auf logische Diskeinheiten. Die Zahl der Buffer läßt sich beim Systemstart durch den Befehl:

```
BUFFERS=xx
```

in der Datei CONFIG.SYS festlegen. Der Aufbau dieser Buffer ist in einem eigenen Abschnitt beschrieben.

Der nächste 4-Byte-Zeiger ab Offset 16H bezieht sich auf eine interne DOS-Tabelle mit den Informationen bezüglich der Zugriffspfade und SEEK-Operationen. Hier werden zum Beispiel die DOS-I/O-Umleitungen gespeichert.

Der folgende 4-Byte-Vektor (Offset 1AH) zeigt auf den internen DOS-Datenbereich mit den FCB-Einträgen. Der Bereich ist jedoch nur belegt, falls in der Datei CONFIG.SYS die Anweisung:

```
FCBS=xx
```

steht. In diesem Fall findet sich im DCB ab Offset 1EH ein 16-Bit-Wert mit der Größe dieser FCBS-Tabelle.

Der DCB endet ab DOS 3.x bei der Offsetadresse 21H, da sich ab Offset 22H der erste (Nul) Device-Treiber anschließt.

Einige DOS-Versionen verwalten weitere Vektoren in der Umgebung des DCB. Die genaue Belegung der Funktionen ist allerdings undokumentiert.

7.4 Die Disk-Parameter-Blocks (DPB)

Die INT 21-Funktion 52H gibt die Adresse des DOS-Control-Blocks zurück. In dieser Datenstruktur findet sich ab Offset 00H ein 4-Byte-Zeiger auf die Anfangsadresse des ersten Disk-Parameter-Blocks (DPB). Hierbei handelt es sich um eine (undokumentierte) Tabelle, die vom Treiber zur Verwaltung der jeweiligen blockorientierten Einheit angelegt wird. Hier finden sich zum Beispiel Informationen über den Aufbau des Speichermediums. Der Inhalt läßt alternativ über die INT 21-Funktionsaufrufe 1FH und 32H abfragen. Der DCB besitzt folgende Struktur:

Offset	Bytes	Funktion
00	1	Drive-Nummer (Nummer des Treibers)
01	1	Einheiten-Nummer (Subunit im DPB)
02	2	Bytes pro Sektor
04	1	Sektor pro Cluster - 1
05	1	Cluster to Sektor shift
06	2	Zahl der reservierten Bootsektoren
08	1	Zahl der FAT's
09	2	Zahl der Einträge im Hauptverzeichnis
0B	2	Sektornummer des ersten Datenclusters
0D	2	letzter belegter Sektor
----	---	--- Aufbau für DOS 2.x ---
0F	1	Zahl der Sektoren für eine FAT
10	2	Start-Sektor des Root Directory

Offset	Bytes	Funktion
12	4	Zeiger auf den Device Header des Laufwerks
16	1	Media Descriptor Byte
17	1	Zugriffsbyte (0 = Disk Access, FFH rebuild Block)
18	4	Zeiger auf den folgenden DPB (FFFFH = last Block)
1C	2	Start Cluster current Directory (0 = Root)
1E	64	ASCIIZ-String aktueller Directory Pfad
----	---	--- Aufbau für DOS 3.x ---
0F	1	Zahl der Sektoren für eine FAT
10	2	Start Sektor des Root Directory
12	4	Zeiger auf den Device Header des Laufwerks
16	1	Media Descriptor Byte
17	1	Zugriffsbyte (0 = Disk Access, FFH rebuild Block)
18	4	Zeiger auf den folgenden DPB (FFFFH = last Block)
1C	2	Startcluster für die Suche nach »free space«
1E	2	Zahl der freien Cluster (FFFFH = unbekannt)
----	---	--- Aufbau für DOS 4.x - 6.0 ---
0F	2	Zahl der Sektoren für die FAT's
11	2	Start Sektor des Root Directory
13	4	Zeiger auf den Device Header
17	1	Media Descriptor Byte
18	1	Zugriffsbyte (0 = Disk Access, FFH rebuild Block)
19	4	Zeiger auf den folgenden DPB (FFFFH = last Block)
1D	2	letzter geschriebener Cluster
1F	2	Zahl der freien Cluster (FFFFH = unbekannt)

Tabelle 7.7: Der Aufbau des Disk-Parameter-Blocks (DPB)

Im ersten Byte steht die Nummer des Treibers, die mit der Laufwerksnummer übereinstimmt (0 = A:, 1 = B:, 2 = C: etc.). Diese Nummer wird bei der Installation (SYSINIT) eingetragen. Im zweiten Byte findet sich der Code der Untereinheit, falls der Treiber mehrere Einheiten unterstützt. Enthält der Eintrag ab Offset 00H den Wert FFFFH:FFFFH, dann ist das Ende der Liste erreicht.

Ab Offset 02 findet sich die Zahl der Byte pro Sektor. MS-DOS speichert standardmäßig 512 Byte (200H) pro Sektor ab. Die Verwaltung des Speichers erfolgt in Clustern (Segmenten aus mehreren aufeinanderfolgenden Sektoren). Die Zahl der Sektoren pro Cluster findet sich ab Offset 04H. Es ist aber zu beachten, daß die Zählweise ab null beginnt, d.h. der Wert 3 bedeutet 4 Sektoren pro Cluster. Die Zahl der Sektoren pro Cluster muß zu 2 ** n (N=1,2,..) festgelegt werden.

Offset 06H enthält ein Wort, in dem die Zahl der reservierten Sektoren gespeichert ist. In diesen Sektoren finden sich z.B. die Boot-Programme.

Das folgende Byte (Offset 08H) spezifiziert die Zahl der File-Allocation-Tables (FAT). DOS trägt hier standardmäßig den Wert 2 ein.

Eine wichtige Größe ist auch die Zahl der Einträge im Hauptverzeichnis (Offset 09H). Bei 360-Kbyte-Disketten findet sich hier der Wert 112 (70H), während er bei 20-Mbyte-Platten bis auf die Zahl 512 reichen kann.

Die Sektornummern des ersten und letzten Datenclusters finden sich in zwei Worten ab Offset 0BH. Daran schließt sich ein Byte an, in dem die Zahl der Sektoren für die FAT steht. Offset 10H spezifiziert den Startsektor für das Hauptinhaltsverzeichnis. Das Wort ab Offset 0DH enthält die Nummer des letzten Clusters+1 des Datenbereiches. Ist der Wert kleiner als 0FF6H, benutzt DOS eine 12-Bit-FAT für das Medium. Andernfalls wird eine 16-Bit-FAT angelegt.

Interessant ist der 4-Byte-Zeiger ab Offset 12H. Er spezifiziert die Adresse des *Device Headers* des Einheitentreibers (s. entsprechendes Kapitel). Damit kann zumindest der Einstieg in die Treiberkette gefunden werden.

Das Media-Descriptor-Byte (Offset 16H) gibt an, um welchen Speichertyp es sich handelt (s. Kapitel über das DOS-Dateisystem). Das folgende Byte dient als Flag, mit dem ein Zugriff auf das Medium angezeigt wird. Ist der Wert des Flags=0, wurde auf das Medium zugegriffen. Wird das Flag auf -1 gesetzt (oder ist es -1), dann aktualisiert DOS die internen Datenstrukturen für das Laufwerk vor dem nächsten Zugriff auf das Medium neu.

Das letzte Feld (Offset 18H) enthält einen 4-Byte-Zeiger auf den nächsten Disk-Parameter-Block (DPB). Der letzte Eintrag der Liste wird mit dem Wert FFFF:FFFFH markiert.

Ab DOS 4.0 ergibt sich eine Erweiterung des DPB um ein Byte.

Offset	Bytes	Funktion
00	1	Drive-Nummer (Nummer des Treibers)
01	1	Einheiten-Nummer (Subunit im DPB)
02	2	Bytes pro Sektor
04	1	Sektor pro Cluster - 1
05	1	Cluster to Sektor shift
Offset	Bytes	Funktion
06	2	Zahl der reservierten Bootsektoren
08	1	Zahl der FAT's
09	2	Zahl der Einträge im Hauptverzeichnis
0B	2	Sektornummer des ersten Datenclusters
0D	2	letzter belegter Sektor
0F	2	Zahl der Sektoren für die FAT's
11	2	Start-Sektor des Root Directory
13	4	Zeiger auf den Device Header
17	1	Media Descriptor Byte
18	1	Zugriffsbyte (0 = Disk Access, FFH rebuild Block)
19	4	Zeiger auf den folgenden DPB (FFFFH = last Block)
1D	2	letzter geschriebener Cluster
1F	2	Zahl der freien Cluster (FFFFH = unbekannt)

Tabelle 7.8: Der Aufbau des Disk-Parameter-Blocks (DPB) ab DOS 4.0

Da DOS 4.x Festplatten mit mehr als 32 Mbyte Kapazität verwaltet, reicht ein Byte (Offset 0FH) nicht aus. Die Clusternummer wird deshalb auf 2 Byte erweitert. Die Länge des DPB erweitert sich von DOS 3.3 von 32 Byte ab DOS 4.0 auf 33 Byte.

Weiterhin unterscheiden sich die die DOS-Versionen auch in der Belegung freier Cluster. In DOS 2.0 beginnt die Suche nach einem freien Cluster immer vom Beginn des Datenträgers und der erste freie Bereich wird belegt. Ab DOS 3.0 merkt sich das Betriebssystem die Nummer des zuletzt geschriebenen Sektors. Die Belegung neuer Sektoren beginnt dann immer ab diesem Startpunkt. Nur beim Systemstart wird der Startpunkt auf 0 gesetzt. Durch diese Technik wird die Fraktionierung der Platte/Diskette zumindest verzögert.

7.5 Die Device-Treiberkette

Beim Systemstart lädt das Programm SYSINIT (aus IO.SYS) die Treiber aus der Datei CONFIG.SYS und aus dem residenten Teil von IO.SYS. Der erste geladene Treiber ist der NUL-Treiber. An diesen schließt sich eine Kette weiterer Treiber an.

Im Kopf eines jeden Treibers findet sich eine Datenstruktur, die als erstes einen 4-Byte-Vektor enthält. Dieser Vektor spezifiziert die Anfangsadresse des nachfolgenden Treibers. Der letzte Treiber in dieser Kette enthält den Wert -1,-1 (FFFF:FFFFH). Ab Offset 04H steht das Device-Attribut-Wort. Die Belegung ist in Tabelle 8.2 aufgeführt. Interessant ist Bit 4, da die offizielle Dokumentation angibt, daß das Bit = 0 gesetzt werden muß. Dieses Bit wird von speziellen Treibern genutzt, um über den undokumentierten INT 29 Zeichen auszugeben. Bei Block-Einheiten werden die Bits 8 und 9 vermutlich von DRIVER.SYS benutzt. Die weitere Struktur des Kopfes ist in Tabelle 8.1 aufgeführt. Bei CD-ROM's wird die Tabelle um 10 Byte erweitert. Das folgende Bild zeigt den Aufbau dieser einseitig verketteten Treiberliste.

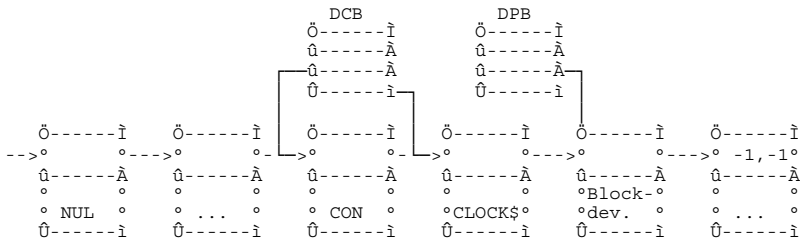


Bild 7.11: Der Aufbau der Treiberkette

Es stellt sich die Frage, wie man nun an die Adressen dieser Treiber kommt?

Sobald ein Adreßvektor bekannt ist, lassen sich die nachfolgenden Treiber über die Linkliste erreichen. Die Adresse der Standard-Ein-/Ausgabeeinheit (CON) findet sich ab Offset 0CH im DOS-Control-Block (DCB). Auch die Adresse der CLOCK$-Einheit läßt sich über den 4-Byte-Vektor ab Offset 08H im DOS-Control-Block erreichen.

Um die Adressen der blockorientierten Einheiten zu ermitteln, besteht einerseits die Möglichkeit, die Treiberkette nach einem entsprechenden Namen zu durchsuchen. Andererseits findet sich die Adresse ab Offset 12H im Disk-Parameter-Block (DPB).

Dies ist aber nicht ganz befriedigend, da gemäß obigem Bild die Kette mit dem NUL-Treiber beginnt. Sobald diese Adresse bekannt ist, läßt sich die gesamte Treiberkette erreichen. Aber wo ist dieser Vektor gespeichert?

DOS muß ja irgendwie diesen Anfang der Kette ermitteln. In DOS 1.x und 2.x wurde der Vektor nach einem INT 21-OPEN-Funktionsaufruf im reservierten Teil des FCB zurückgegeben. Diese undokumentierte Möglichkeit besteht aber ab DOS 3.x nicht mehr. Da IO.SYS die Treiber irgendwo im Speicher ab Adresse 0600:0000 speichert, besteht die Möglichkeit, im Speicher nach dem Text »NULlgetreibers ab Offset -0AH vor dem Namen. Mit dieser Methode läßt sich die Adresse des NUL-Device meist finden.

Während der Entstehung dieses Buches wurde jedoch eine wesentlich einfachere Lösung gefunden. Die Verwaltung des Speichers übernimmt während des Systemstarts das Programm SYSINIT (aus IO.SYS). Dieses installiert auch die MS-DOS-Anwendertreiber. Weiterhin baut es die internen DOS-Tabellen auf. Nun kommt eine Eigenart von DOS zum Tragen, daß die Systemtreiber (aus IO.SYS) zuletzt installiert werden. Dies bedeutet, daß andere Systemtreiber und Datenbereiche vor dem NUL-Treiber liegen. In der MS-DOS-Version 2.x und 3.x liegt das NUL-Device hinter dem DOS-Control-Block. Daher braucht

DOS auch keinen Zeiger auf diesen Treiber zu speichern, da sich die Adresse aus der Anfangsadresse des DCB ermitteln läßt. Die INT 21-Funktion 52H gibt den Vektor auf diesen Block zurück (s. Abschnitt Struktur und Aufbau des DCB). Zwischen DCB und Kopf des NUL-Device findet sich allerdings noch ein 4-Byte Zwischenraum.

Bei MS-DOS 2.x ist zur Anfangsadresse des DCB ein Offset von 17H zu addieren, um die Anfangsadresse des NUL-Device zu erhalten.

Ab MS-DOS (PC-DOS) 3.1 ist der Offsetwert auf 22H zu erhöhen. Zur Kontrolle kann der Name des Treibers geprüft werden. Bei Fehlern ist die Lage des Treibers experimentell zu bestimmen, da die beschriebenen Zusammenhänge bisher von Microsoft nicht dokumentiert wurden.

7.6 Die DOS-Puffer

DOS wickelt den Datentransfer zwischen den Anwenderprozessen und den physikalischen Einheiten über die Einheitentreiber ab. Bei blockorientierten Einheiten (Diskette/Platte) ist nur ein Zugriff auf die Daten eines kompletten Sektors möglich, d.h. es kann nur ein kompletter Sektor gelesen oder geschrieben werden. In vielen Fällen umfaßt die Transferanweisung des Anwenderprozesses aber weniger Bytes, womit DOS eine Zwischenspeicherung der Daten vornehmen muß. Möchte ein Prozeß z.B. 80 Byte aus einer Datei lesen, ist DOS gezwungen, den gesamten Sektor (512 Byte) in den Speicher zu transferieren. Dort werden die 80 Byte dann an den Prozeß weitergereicht. Das gleiche Bild ergibt sich, falls ein Prozeß Daten in einer Datei speichern möchte. Solange die Satzlänge kleiner als die Sektorlänge des Mediums ist, werden die Daten im Speicher gehalten. Erst eine Close-Anweisung oder ein Erreichen der Sektorgröße sorgt für den Transfer zum Medium. Weiterhin müssen die Informationen zur Dateiverwaltung (FAT) im Speicher gehalten werden. Hierfür reserviert DOS bestimmte Bereiche im Speicher für diesen Zweck, die als Puffer bezeichnet werden.

Standardmäßig werden beim Systemstart (IO.SYS) zwei Puffer angelegt, die zur Blockierung/Deblockierung der zu transferierenden Daten und zur Aufnahme der FAT dienen.

Da aber alle Schreib-/Leseaufrufe über Puffer abgewickelt werden, prüft DOS vor jedem Transfer, ob die Daten nicht bereits im Puffer stehen. In diesem Fall findet nur ein Transfer zwischen Puffer und dem Prozeß statt. Dies spart Zeit, da Zugriffe auf das Speichermedium vermieden werden. Allerdings tritt schnell die Situation auf, daß die zwei Puffer nicht ausreichen und immer durch andere Daten überschrieben werden. Damit geht der erreichbare Geschwindigkeitsgewinn wieder verloren. Aus diesem Grunde besteht ab DOS 2.0 die Möglichkeit, die Zahl der Puffer zwischen 1 und 99 einzustellen. Hierzu ist in der Datei CONFIG.SYS die Anweisung:

```
BUFFERS = xx
```

einzutragen. Beim Systemstart wird dann die angegebene Zahl von Puffern installiert. Jeder Puffer belegt einen Speicherbereich von 528 Byte (16-Byte-Kopf, 512-Byte-Datenbereich).

Die Zahl der Puffer, die zur optimalen Systemeinstellung notwendig ist, wird üblicherweise experimentell ermittelt. Systeme mit Festplatten sollten mindestens 3 Puffer enthalten. Datenbankanwendungen erfordern meist zwischen 10 und 20 Puffern. Eine möglichst

große Pufferzahl ist andererseits auch zu vermeiden, da erstens entsprechend Speicherplatz belegt wird. Andererseits muß DOS jeweils alle Puffer absuchen, um zu prüfen, ob die Daten nicht schon vorliegen. Bei vielen Puffern kann dies viel Zeit benötigen, so daß der Zeitvorteil gegenüber dem Diskzugriff verschwindet.

Der Aufbau eines DOS-Puffers wurde von Microsoft nicht dokumentiert, das Format variiert zwischen den verschiedenen Versionen. An den Kopf schließt sich der Datenbereich mit 512 Byte an. Der Kopf besitzt folgende Struktur:

Offset	Bytes	Funktion
---	--	--- DOS 2.x ---
00	4	Zeiger auf den nächsten Puffer (FFFFH = last)
04	1	Laufwerksnummer (0 = A, 1 = B, .., FF = unbelegt)
05	3	unbelegt (00 00 01)
08	2	logische Sektornummer
0A	1	Zahl der Schreibkopien (1 für nicht FAT-Sektoren)
OB	1	Sektor Offset bei mehreren Schreibkopien
0C	4	Zeiger zum DOS-Parameter-Block (DPB)
10	512	Puffer von 512 Byte
---	--	--- DOS 3.x ---
00	4	Zeiger auf den nächsten Puffer (FFFFH = last)
04	1	Laufwerksnummer (0 = A, 1 = B, .., FF = unbelegt)
05	1	Benutzungskennummer
06	2	logische Sektornummer
08	1	Zahl der Schreibkopien (1 für nicht FAT-Sektoren)
09	1	Sektor Offset bei mehreren Schreibkopien
0A	4	Zeiger zum DOS-Parameter-Block (DPB)
0E	2	unbenutzt
10	512	Puffer von 512 Byte

Tabelle 7.9: Der Aufbau der DOS-Puffer bis DOS 3.3

Der Kopf enthält einen 4-Byte-Vektor, der auf die Anfangsadresse des nächsten Puffers zeigt. Der letzte Puffer enthält als Endemarkierung den Wert FFFF:FFFFH im Vektor.

Weiterhin findet sich ab Offset 0AH ein 4-Byte-Vektor auf die Anfangsadresse des Disk-Parameter-Blocks (DPB) des zugeordneten Treibers, der den Puffer belegt hat.

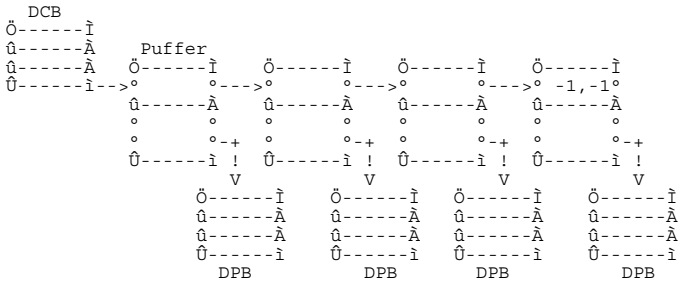


Bild 7.12: Die DOS-Pufferkette

Die Anfangsadresse dieser Pufferkette findet sich im DOS-Control-Block (DCB) ab Offset 12H. Ein Zugriff auf die Einzelpuffer kann dann über die verkettete Liste erfolgen. Bei einer großen Anzahl von Puffern ist es natürlich klar, daß die Zugriffe länger dauern, da ja im ungünstigsten Fall die gesamte Kette durchsucht werden muß.

Der eigentliche Puffer von 512 Byte beginnt ab Offset 10H. Im Feld ab Offset 04H findet sich ein Byte, in welchem die Kennung des aktuell zugeordneten Laufwerkes steht (1 = A:, 2 = B: etc.). Bei einem Wechsel des Laufwerkes wird dieser Code modifiziert. Wurde der Puffer noch nicht benutzt, enthält das Feld den Wert 0FFH.

Die Benutzungskennummer (Byte ab Offset 05H) gibt einen Hinweis auf den Typ der benutzten Daten. Hier geben verschiedene Bits an, mit welchen Daten der Puffer belegt ist.

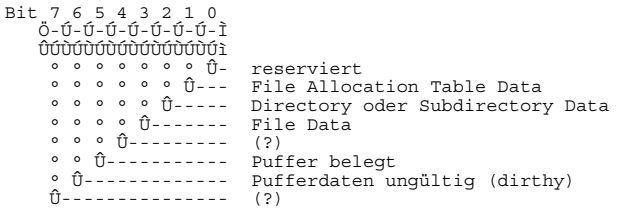


Bild 7.13: Kodierung der Benutzungskennummer

Ist Bit 1 gesetzt, befindet sich ein Sektor aus der FAT des Laufwerkes im Puffer. Mit Bit 2 markiert DOS alle Sektoren, die zwischen Puffer und Directory-Sektoren transferiert werden. Nur wenn Bit 3 gesetzt ist, handelt es sich um Daten aus dem Datenfilebereich.

Ab Offset 08H merkt sich DOS eine Zugriffsnummer auf den Puffer, deren genaue Bedeutung aber nicht bekannt ist. Das Wort ab Offset 0EH ist zur Zeit noch unbenutzt.

Änderungen in DOS 4.X

Ab DOS 4.0 wurde die Verwaltung der Diskpuffer komplett umgestellt. Hier lassen sich zum Beispiel einige Daten in den EMS-Bereich auslagern. Der Zeiger aus dem DOS-Control-Block zeigt auf die Anfangsadresse des Disk-Buffer-Infoblocks. Dieser enthält eine Reihe von globalen Informationen für die Verwaltung der Diskpuffer und besitzt folgenden Aufbau:

Offset	Bytes	Funktion
00H	4	Zeiger auf die Kette mit den Puffer Headern (NDBCH)
04H	2	Zahl der Diskpuffer (NDBCH) in der Kette
06H	4	Zeiger auf den »lookahead buffer«
0AH	2	Zahl der »lookahead buffer«
0CH	1	00H Buffer in EMS (/X-Option), FFH kein EMS benutzt
0DH	2	EMS-Handle für die Puffer, 0 -> kein EMS
0FH	2	EMS-physical-Page Number für die Puffer (meist FFH)
11H	2	unbelegt (0001H)
13H	2	Segment des EMS-physical Page-Frame
15H	2	unbelegt (0000H)
17H	8	4 Worte mit EMS-Informationen (?)
----	----	Struktur in DOS 4.01
00H	4	Zeiger auf die Kette mit den Puffer Headern (NDBCH)
04H	2	Zahl der Diskpuffer (NDBCH) in der Kette
06H	4	Zeiger auf den »lookahead buffer«
0AH	2	Zahl der »lookahead buffer«
0CH	1	01H Markierung für DOS 4.01 Buffer
0DH	9	EMS-Daten Belegung nicht bekannt
16H	2	Segment des Arbeitspuffers im Hauptspeicher (/XS/XD)
18H	2	EMS-Handle für die Puffer, 00H kein EMS
1AH	2	EMS physikal. Seitennummer des Puffers (meist FFH)
1CH	2	unbelegt (0000H)
1EH	2	Segment EMS Seite (page frame)
20H	2	unbelegt (000H)
22H	2	00H bei /XS, 01H bei /XD, FFH Puffer nicht im EMS

Tabelle 7.11: Der Aufbau des DOS-Disk-Puffer-Infoblocks (DOS 4.x)

Der erste Eintrag zeigt auf eine Kette mit den Köpfen des Diskpuffers (disk buffer hash chain), während die Zahl der Einträge in der Kette ein Byte später gespeichert ist. DOS 4.x besitzt die Möglichkeit die Informationen über FAT's und Unterverzeichnisse im Speicher zu halten. Deshalb läßt sich ein »lookahead«Buffer anlegen. Ab 06H findet sich der Zeiger auf diesen Bereich. Bei fehlendem Buffer ist der Wert auf 0 gesetzt. Ab 0AH steht die Zahl der Sektoren, die im lookahead-Buffer stehen. Fehlt der Buffer, ist das Byte mit 0 belegt, sonst nimmt es den Wert y »n, der sich mit der CONFIG.SYS-Anweisung:

```
BUFFERS=x,y
```

setzen läßt. Da DOS 4.x einen Teil der Puffer im EMS halten kann, verwaltet es die Informationen über den EMS-Bereich ebenfalls im Infoblock.

Leider wurde auf Grund zahlreicher Fehler der Aufbau des DOS-Disk-Puffer-Info-Blocks beim Übergang zur Version 4.01 verändert (siehe obige Tabelle). In Byte 0CH findet sich der Wert 01H als Markierung für die Version 4.01. Die Einträge ab dem Offset 0DH bis 15H beziehen sich auf EMS-Parameter. Die Belegung ist aber nicht bekannt.

Die Kette mit den DOS-Disk-Puffer-Headern ist in DOS 4.x gleich und besitzt folgendes Format:

Offset	Bytes	Funktion
00H	2	EMS logical Page Number (-1 not in EMS)
02H	4	Zeiger auf den »least recently used**Fehler! Verweisquelle konnte nicht gefunden werden.**
06H	2	reserviert (0000H)

Tabelle 7.12: Der Aufbau des DOS-Disk-Puffer-Headers (DOS 4.x)

Für jeden Puffer ist ein eigener Eintrag vorhanden. Das erste Wort enthält die logische EMS-Seitennummer, in der die Pufferkette resident gespeichert ist. Falls kein EMS-Bereich existiert, ist der Eintrag nicht belegt. Dann findet sich der Puffer im DOS-640-Kbyte-Bereich. Alle Puffer liegen im gleichen Segment. Im folgenden Feld findet sich ein Zeiger auf den Header des *least recently used*-Puffers. DOS merkt sich, welche Puffer benutzt werden und markiert den Puffer, auf den am wenigsten zugegriffen wird. Sind alle Puffer belegt, nutzt DOS den *least recently used*-Puffer für die Einlagerung neuer Daten. Ein gepufferter Disk-Sektor wird in den Puffer mit der Nummer:

```
Sektoradresse Modulo (NDBCH)
```

gespeichert (0 <= N <= NDCBH-1). NDCBH ist der N-te Disk-Chain-Block-Header. Jede Teilkette wird nach Möglichkeit resident in eine EMS-Page gelegt, falls EMS vorhanden ist.

An den Kopf schließt sich dann der eigentliche Puffer mit folgendem Format an.

Offset	Bytes	Funktion
00	2	Zeiger (Offset) next least-recently-used-buffer
		Zeiger in eine vorwärts verkettete Liste
02	2	Zeiger (Offset) auf den vorhergehenden Puffer
		Zeiger in eine rückwärts verkettete Liste
04	1	Drive (0 = A, 1 = B, ... FFH = unbelegt)
05	1	Benutzungskennummer
06	4	logische Sektornummer
0A	1	Zahl der Schreibkopien (1 für nicht FAT-Sektoren)
OB	2	Sektor Offset bei mehreren Schreibkopien
0D	4	Zeiger zum DOS-Parameter-Block (DPB)
11	2	Buffer Count (bei remote, siehe Benutzerkennummer)
13	1	unbelegt
14	512	Puffer von 512 Bytes

Tabelle 7.13 Der Aufbau der DOS-Puffer ab DOS 4.0

Ab DOS 4.0 ist die Kette mit den Puffern als doppelt verkettete Liste ausgeführt. Alle Puffer mit der gleichen Adresse modulo NDBCH finden sich in der gleichen verketteten Liste. Die Zeiger werden als Offset angegeben, da sich alle Angaben innerhalb einer Kette auf das gleiche Segment beziehen. Ab Offset 0AH steht die Zahl der zu schreibenden Kopien eines Buffers. Bei FAT-Puffern findet sich hier die Zahl der FAT's. Bei Daten wird der Puffer nur einmal geschrieben. Ab Offset 0BH findet sich dann der Offset zwischen zwei FAT's.

Die Informationen über den Pufferaufbau sind nicht durch Microsoft dokumentiert.

Änderungen ab DOS 5.0

Ab DOS 5.0 wurde die Verwaltung der Diskpuffer, insbesondere was die EMS-Verwaltung betrifft, wieder komplett umgestellt. Der Zeiger aus dem DOS-Control-Block zeigt auf die Anfangsadresse des Disk-Buffer-Infoblocks. Dieser enthält eine Reihe von globalen Informationen für die Verwaltung der Diskpuffer und besitzt folgenden Aufbau:

Offset	Bytes	Funktion
00H	4	Zeiger auf die Kette mit den Puffer Headern (NDBCH)
04H	2	0000H (DOS besitzt keine Hash Puffer)
06H	4	Zeiger auf den »lookahead bufferû
0AH	2	Zahl der »lookahead sektorenû
0CH	1	Buffer Location (00H Base Memory, no workspace) (01H HMA, workspace in Base Mem)
0DH	4	Zeiger auf 1 Segment Workspace Buffer im Base Mem
11H	3	unbelegt
14H	8	unbekannte Belegung
1CH	1	Bit 0 = 1, falls UMB MCB mit normalen MCB verkettet
1DH	2	unbekannt
1FH	2	Segment 1. MCB im UMB oder FFFFH.
21H	2	Startadresse (Paragraph) Suche MCB-Kette

Tabelle 7.14: Der Aufbau des DOS-Disk-Puffer-Infoblocks (DOS 5.0)

Der erste Eintrag zeigt auf die Adresse des zuletzt benutzten Puffers. DOS 5.0 benutzt keine Hash-Puffer, weshalb das folgende Wort unbenutzt ist. Weitere Einträge der Tabelle beziehen sich auf die Verwaltung des UMB. So findet sich die erste UMB-MCB meist ab Adresse 9FFF:0000H. Hier wird das Video-RAM von DOS als Block reserviert.

Der eigentliche Puffer besitzt ab DOS 5.0 folgendes Format.

Offset	Bytes	Funktion
00	2	Zeiger (Offset) next least-recently-used-buffer
		Zeiger in eine vorwärts verkettete Liste
02	2	Zeiger (Offset) auf den vorhergehenden Puffer
		Zeiger in eine rückwärts verkettete Liste
04	1	Drive (0 = A, 1 = B, ... FFH = unbelegt)
05	1	Benutzungskennummer Bit 7: remote Buffer Bit 6: Buffer dirthy Bit 5: Buffer benutzt (referenced) Bit 4: Search Data Buffer Bit 3: Sektor im Datenbereich Bit 2: Sektor in Directory Bit 1: Sektor in FAT Bit 0: reserviert
06	4	logische Sektornummer
0A	1	Zahl der Schreibkopien (1 für nicht FAT-Sektoren)
OB	2	Sektor Offset bei mehreren Schreibkopien
0D	4	Zeiger zum DOS-Drive-Parameter-Block (DPB)
11	2	Buffer Count (bei remote, siehe Benutzerkennummer)
13	1	unbelegt
14	512	Puffer von 512 Bytes

Tabelle 7.15 Der Aufbau der DOS-Puffer ab DOS 5.0

In DOS 5.0 ist die Kette mit den Puffern als doppelt verkettete Liste ausgeführt. Ab Offset 0AH steht die Zahl der zu schreibenden Kopien eines Buffers. Bei FAT-Puffern findet sich hier die Zahl der FAT's. Bei Daten wird der Puffer nur einmal geschrieben. Ab Offset 0BH findet sich dann der Offset zwischen zwei FAT's.

Die Informationen über den Pufferaufbau sind nicht offiziell durch Microsoft dokumentiert. Die Angaben über die DOS-5.x-Pufferverwaltung sind deshalb mit Vorsicht zu betrachten.

7.7 Die DOS-Stackverwaltung

Bis zur Version 3.1 besitzt DOS drei interne Stacks zur Abwicklung interner Interrupts. Das BIOS benutzt dagegen die Stacks der Anwenderprogramme. Ab DOS 3.2 wurde eine erweiterte Stackverwaltung implementiert, die beim Systemstart durch den Befehl:

```
STACKS= m,n
```

in CONFIG.SYS eingebunden wird. Der Wert m legt dabei die Zahl der Stacks fest, während n die Größe eines Stacks in Byte bestimmt. Die Werte dürfen dabei in folgenden Grenzen variieren:

```
m =  8 ..  64 Stacks
n = 32 .. 512 Byte
```

Standardmäßig reserviert DOS beim Start neun Stacks mit je 128 Byte Größe. Die Anweisung:

```
STACKS=0,0
```

schaltet die Stackverwaltung unter DOS jedoch ab.

Der Handler belegt einen Codebereich von 2064 Byte und wird beim Systemstart geladen. Bei PC-DOS setzt der Treiber dann die Vektoren der Interrupts 2, 8, 9 und 70 auf eine eigene Routine um, so daß nach einem Interrupt zuerst die Stackverwaltung aktiviert wird. Ein Interrupt veranlaßt dann folgende Sequenz:

- Es wird die Stackverwaltung über den INT x aktiviert.
- Der Stackmanager sucht einen freien Stack und sichert den Zustand der Register SS:SP in lokalen Variablen.
- Die Register SS:SP werden auf den neuen Stack gesetzt.
- Dann wird die Original-Interruptroutine aktiviert.

Nach der Abarbeitung der ISR durchläuft DOS die obige Sequenz in umgekehrter Reihenfolge. Problematisch wird der Ablauf immer dann, wenn der Stackmanager nach einer Aktivierung keinen freien Stack mehr findet. Dann sperrt er alle Interrupts und verzweigt zu einer Endlosschleife. Das System ist abgestürzt und muß neu gebootet werden.

Die Stackverwaltung benutzt einen internen Datenbereich (Stackverwaltungsrecord) mit folgender Struktur zur Ablage der Daten:

Bytes	Feld
2	Status des Stacks 0 -> frei, 1 -> belegt, 3 -> gesperrt
1	unbelegt
4	Speicherstelle in der SS:SP des unterbrochenen Programmes gesichert wird
2	Offset auf das oberste Wort des zugehörigen Stacks

Tabelle 7.16: Stack-Datenbereich

Im ersten Wort steht eine Markierung, ob der Stack frei oder belegt ist. Mit dem Wert 3 läßt sich ein Stack für weitere Zugriffe sperren. Dies ist zum Beispiel erforderlich, falls ein darüberliegender Stackbereich überläuft und Daten im nachfolgenden Stackbereich abgelegt werden.

Im nächsten Doppelwort speichert der Manager den Inhalt der Register SS:SP ab, bevor er den neuen DOS-Stack setzt. Durch das Statuswort läßt sich ein mehrfacher Zugriff auf ein belegtes Segment vermeiden.

Das letzte Wort enthält den Offset auf das oberste Wort des zugehörigen Stackbereiches. Die Segmentadresse des Stacks ist identisch mit dem Codesegment des Treibers, da Daten und Code innerhalb von 64 Kbyte liegen.

Für jeden definierten Stack legt DOS die obige Struktur an. Weiterhin existieren noch die m Stackbereiche mit einer Länge von n-Byte. Diese Bereiche liegen hintereinander im Speicher. Im obersten Wort eines jeden Stacks findet sich wieder der Offset auf den zugehörigen Stapelverwaltungsrecord. Der Manager nutzt diesen Eintrag zur Konsistenzprüfung des Stackbereiches. Falls der darüberliegende Stack überläuft, wird das oberste Wort des folgenden Stacks mit überschrieben. Der Offset ist damit verloren, und der Manager sperrt den Stack für weitere Zugriffe bis zum nächsten Systemstart. Bei einem Interrupt durchsucht der Manager die Stacks von »oben**Fehler! Verweisquelle konnte nicht gefunden werden.**unten**Fehler! Verweisquelle konnte nicht gefunden werden.**Der Treiber zur Stackverwaltung befindet sich hinter der DOS-Umleitungstabelle, deren Anfangsadresse per INT 21, Funktion 52H abfragbar ist. Pro Laufwerk sind dort 81 Byte reserviert. Alternativ läßt sich die Segmentadresse des Treibers über einen Interruptvektor (z.B. INT 2 oder INT 0E) abfragen. Dieser Vektor darf allerdings nicht durch ein anderes Programm umdefiniert worden sein. Der Treiber enthält ab Offset 0 eine Datenstruktur mit folgendem Aufbau:

Bytes	Feld
2	Kennung
2	Stackanzahl (x aus STACKS=x,y)
2	Größe Stack-Control-Block (8*x)
2	Größe jedes Einzelstacks (y aus STACKS=x,y)
4	Zeiger auf den Bereich mit den Stacks
2	unterster Rahmen
2	oberster Rahmen
2	nächster freien Rahmen

Tabelle 7.17: Aufbau Stack-Verwaltungsrecord

An den Kopf schließt sich dann der erste Handler für den ersten abgefangenen Interrupt an.

Bei PC-DOS verwaltet der Handler zusätzlich die Interrupts 0A, 0B, 0C, 0D, 0E, 72, 73, 74, 76 und 77, sofern die folgenden Bedingungen erfüllt sind:

- Der Vektor muß auf eine gültige ISR zeigen, d.h., der Vektor wird auf 0 überprüft. Außerdem darf er nicht auf eine IRET- Anweisung zeigen.
- Die ISR muß ab Offset 06H die Kennung »KB**Fehler! Verweisquelle konnte nicht gefunden werden.**· Fehlt die Kennung, muß das Segment ungleich F000H sein.
- ISR-Handler mit der Segmentadresse F000H werden nur dann nicht verwaltet, wenn ab Offset 1FFH der Offset der ISR steht.

Für jeden verwalteten Interrupt wird im Speicher ein eigener Handler mit zirka 90 Byte Speicherbedarf angelegt. Hinzu kommen noch die Datenbereiche für die Stacks.

7.8 Die Current Directory Structure (CDS)

Im DOS-Control-Block (DCB) findet sich ab Offset 16H (17H in DOS 3.0) ein 4-Byte-Vektor auf eine weitere undokumentierte interne Datentabelle (Current Directory Structure). Diese enthält alle Angaben über die Device Umleitungen. Hier werden zum Beispiel die Pfadangaben jedes Laufwerkes abgespeichert. Damit kann DOS feststellen, welcher Pfad gerade in diesem Laufwerk aktuell eingestellt ist. Für jedes Laufwerk ist eine Tabelle mit folgendem Aufbau vorhanden:

Offset	Bytes	Funktion
00H	65	Laufwerksbezeichnung (z.B. A:\) und Pfad als ASCIIZ-String mit einer Länge von maximal 64 Zeichen
40H	3	reserviert (00 00 00)
43H	2	Used Flag
45H	4	Zeiger auf den Disk-Parameter-Block (DPB)
---	---	--- lokale Laufwerke ---
49H	2	Startcluster aktuelles Directory (0000H = root) (FFFFH Cluster nicht benutzt)
4BH	4	Marke FFFFH (Zweck unbekannt)
---	---	--- Netzwerk Drives ---
49H	4	Zeiger auf den Redirector, oder auf den REDIRIFS Record, oder FFFF:FFFFH
4DH	2	User Daten vom INT 21/AX=5F03H (?)
---	---	--- beide Versionen ---
4FH	2	Offset in den aktuellen Pfad bezogen auf \
---	---	--- DOS 4.x - DOS 6.0 ---
51H	1	unbekannt
52H	4	Zeiger auf IFS-Treiber
56H	2	unbekannt

Tabelle 7.18: Aufbau der Current Directory Structure (CDS)

Die ersten drei Byte enthalten die Laufwerksbezeichnung (z.B. A:\). Die Bytes gehören zu dem Pfadnamen, der als ASCIIZ-String abgespeichert wird. Dieser Pfad spezifiziert den Zugriff auf die Daten. In DOS darf ein Pfadname deshalb lediglich 64 Byte umfassen, da das Nullbyte noch mit im Puffer zu speichern ist. Bei Netzwerken wird der Maschinenname (z.B. \\MASCHINE\PFAD) angegeben.

Das Flag ab Offset 43H definiert die Attribute des Laufwerkes und besitzt folgende Kodierung:

```
Bit 15:  Netzwerk Laufwerk
    14:  physikalisches Laufwerk
    13:  Laufwerk mit JOIN umgeleitet
    12:  Laufwerk mit SUBST umgeleitet
```

Ist weder Bit 14 noch Bit 15 gesetzt, existiert das Laufwerk nicht.

Ab Offset 45H findet sich ein 4-Byte-Zeiger auf dem Disk-Parameter-Block des Treibers, der dieses Laufwerk verwaltet.

Die Daten ab Offset 49H variieren etwas zwischen lokalen Laufwerken und Netzwerkdrives. Bei lokalen Laufwerken findet sich ab Offset 49H ein Wort mit dem Startcluster des aktuellen Verzeichnisses. Steht hier der Wert 0000H, wird das Hauptverzeichnis benutzt. Ein Wert von FFFFH markiert einen unbenutzten Startcluster. In Netzwerken verweist ein 32-Bit-Zeiger ab Offset 49H auf den Redirector oder den REDIRIFS-Record. Mit FFFF:FFFFH ist der Eintrag unbenutzt.

Die DOS-Programme JOIN und SUBST verändern ebenfalls diese Tabelle und tragen Laufwerksname und DPB des neuen Laufwerkes ein.

Mit dem Befehl:

```
LASTDRIVE = . . .
```

läßt sich in der Datei CONFIG.SYS die Zahl der Laufwerke festlegen. Ansonsten nimmt DOS immer die Laufwerke A bis E an. Ab DOS 4.0 wurde die Tabelle um 7 Byte erweitert, deren Belegung ist allerdings unbekannt.

7.9 Die DOS-Dateitabelle

In DOS bestehen zwei Möglichkeiten, die offenen Dateien zu verwalten. Einmal gibt es die älteren CP/M-orientierten Funktionsaufrufe, die alle Informationen in File-Control-Blocks ablegen. Weiterhin existieren die neuen Xenix-orientierten Aufrufe, die Dateien über Handlecodes verwalten. Diese Handles werden für jede offene Datei im PSP (in der Job File Table JFT) des jeweiligen Prozesses abgespeichert. DOS muß sich jedoch intern die Zuordnung zwischen Datei, Einheit und Handle merken. Aus diesem Grund wird eine interne Tabelle (System File Table SFT) angelegt, die diese Informationen enthält. Die Adresse findet sich im DOS-Control-Block ab Offset 1AH. Die Größe dieser Tabelle kann durch den Befehl:

```
FILES = xx
```

in der Datei CONFIG.SYS beeinflußt werden. Der genaue Aufbau der undokumentierten *System File Table* ist folgender Abbildung zu entnehmen.

Offset	Bytes	Funktion
---	--	--- DOS 2.x ---
00H	4	Zeiger auf die nächste File Tabelle
04H	2	Zahl der Files in dieser Tabelle
---	--	für jeden File folgen 28H Byte
00H	2	Zahl der zugeordneten Filehandles
01H	1	File Open Mode (siehe INT 21, Funktion 3DH)
02H	1	File Attribute
03H	1	Drive (0 = char. device, 1 = A, 2 = B etc.)
04H	11	Filename im FCB-Format
0FH	4	unbekannt
13H	4	Filegröße (?)
17H	2	Filedatum (gepacktes Format, siehe INT 21, 5700H)
19H	2	Filetime (gepacktes Format)
1BH	1	Attribut des Treibers (siehe INT 21, 4400H)
1CH	4	Zeiger zum Device Treiber (Character Device)
	2	Startcluster des Files (Block Device)
	2	letzter zu lesender Cluster (relativ) (Block Device)
20H	2	absolute Clusternummer aktueller Cluster
22H	2	unbekannt
24H	4	aktuelle Fileposition (?)
---	--	--- DOS 3.0 ---
00H	4	Zeiger auf die nächste File Tabelle
04H	2	Zahl der Files in dieser Tabelle
---	--	für jeden File folgen 38H Byte
00H	2	Zahl der zugeordneten Filehandles
02H	2	File Open Mode (siehe INT 21, Funktion 3DH)
		Bit 15 = 1: File mit FCB geöffnet
04H	1	File Attribut
05H	2	Device Info (siehe INT 21, AX = 4400H)
07H	4	Zeiger auf den Device Treiber Header (Char. Device)
		Zeiger zum DOS-Bios-Parameter-Block (Block Device)
0BH	2	Start Cluster der Datei
0DH	2	Filetime (gepacktes Format, s. INT 21, AX = 5700H)
0FH	2	Filedatum (gepacktes Format)
11H	4	Filegröße
15H	4	aktueller Offset in die Datei (SFT)
	2	Zähler letzer I/O für FCB (FCB)
17H	2	Zähler für letztes Open (FCB)
19H	2	relative Clusternummer letzter Lesezugriff
1BH	2	absolute Clusternummer letzter Lesezugriff
1DH	2	Sektornummer mit Directory-Einträgen
1FH	2	Byte Offset in Sektor mit Directory Eintrag

21H	11	Filename im FCB-Format
2CH	4	Zeiger auf vorhergehende Share-File-Tabelle
30H	2	unbekannt
32H	2	PSP Segment des zugehörigen Prozesses (file owner)
34H	2	Offset zum Segment des Sharing Records
36H	2	unbelegt
---	--	--- DOS 3.1 - 3.3 ---
00H	4	Zeiger auf die nächste File Tabelle
04H	2	Zahl der Files in dieser Tabelle
---	--	für jeden File folgen 35H Byte
00H	2	Zahl der zugeordneten Filehandles
02H	2	File Open Mode (siehe INT 21, Funktion 3DH) Bit 15 = 1: File mit FCB geöffnet
04H	1	File Attribut
05H	2	Device Info (siehe INT 21, AX = 4400H)
07H	4	Zeiger auf den Device Treiber Header (Char. Device) Zeiger zum DOS-Bios-Parameter-Block (Block Device)
0BH	2	Start Cluster der Datei
0DH	2	Filetime (gepacktes Format, s. INT 21, AX = 5700H)
0FH	2	Filedatum (gepacktes Format)
11H	4	Filegröße
15H	4	aktueller Offset in die Datei (SFT)
	2	Zähler letzer I/O für FCB (FCB)
17H	2	Zähler für letztes Open (FCB)
19H	2	relative Clusternummer letzter Lesezugriff
1BH	2	absolute Clusternummer letzter Lesezugriff
1DH	2	Sektornummer mit Directory-Einträgen
1FH	1	Zahl der Directory-Einträge im Sektor
20H	11	Filename im FCB-Format
2BH	4	Zeiger auf vorhergehende Share-File-Tabelle
2FH	2	SHARE Netzwerk Maschinennummer offene Datei
31H	2	PSP Segment des zugehörigen Prozesses (file owner)
33H	2	Offset zum Segment des Sharing Records
---	--	--- DOS 4.0 - 6.0 ---
00H	4	Zeiger auf die nächste File Tabelle
04H	2	Zahl der Files in dieser Tabelle
---	--	für jeden File folgen 3BH Byte
00H	2	Zahl der zugeordneten Filehandles
02H	2	File Open Mode (siehe INT 21, Funktion 3DH) Bit 15 = 1: File mit FCB geöffnet
04H	1	File Attribut
05H	2	Device Info (siehe INT 21, AX = 4400H) Bit 15: 1 = Remote File

		° Bit 14: 1 = no Date/Time Update bei File Close
		° Bit 13: 1 = Named Pipe
		° Bit 12: 1 = no Inherit
		° Bit 11: 1 = Netzwerkspooler
07H	4	° Zeiger auf den Device Treiber Header (Char. Device)
		° Zeiger zum DOS-Bios-Parameter-Block (Block Device)
0BH	2	° Start Cluster der Datei
0DH	2	° Filetime (gepacktes Format, s. INT 21, AX = 5700H)
0FH	2	° Filedatum (gepacktes Format)
11H	4	° Filegröße
15H	4	° aktueller Offset in die Datei
		° --- lokale Dateien ---
19H	2	° relative Clusternummer letzter Lesezugriff
1BH	4	° Sektornummer Directory Eintrag
1FH	1	° Zahl der Directory Einträge im Sektor
		° --- Netzwerk ---
19H	4	° Zeiger auf den REDIRIFS Record
1DH	3	° unbekannt
20H	11	° Filename im FCB-Format
2BH	4	° Zeiger auf vorhergehende Share-File-Tabelle
2FH	2	° SHARE Netzwerk Maschinennummer offene Datei
31H	2	° PSP Segment des zugehörigen Prozesses (file owner)
33H	2	° Offset zum Segment des Sharing Records
35H	2	° Nummer (absolute) des letzten benutzten Clusters
37H	4	° Zeiger auf IFS-Treiber (0) falls DOS

Tabelle 7.19: Aufbau der System-File-Tabelle

Die Tabelle nimmt sowohl die Informationen der per Handle geöffneten Dateien, als auch die per FCB geöffneten Dateien auf. Die Struktur variiert aber in den verschiedenen DOS-Versionen. Im Kopf der SFT steht ein Zeiger auf die Folgetabelle. Daran schließt sich der Zähler mit der Information über die Zahl der Einträge der aktuellen Tabelle an. Für jede Datei enthält die Tabelle dann einen Datenbereich in der ab Offset 00H die Zahl der Filehandles, die dieser Tabelle zugeordnet sind, steht. Durch den INT 21-Aufruf DUP-Filehandle lassen sich durchaus mehrere Handles einer Datei zuordnen.

Ab Offset xxH wird der File- oder Einheitenname gespeichert. Dieser umfaßt 8 Zeichen für den Namen und drei Zeichen für die Extension. Der Punkt als Separator entfällt in dieser Darstellung. Fehlende Buchstaben werden mit Leerzeichen aufgefüllt.

Ab Offset XXH steht die Segmentadresse des PSP, dem diese Handles zugeordnet sind. In dem betreffenden PSP-Bereich sind die Handles ja abgespeichert.

Das Programm SHARE.EXE belegt ab DOS 3.x die Bytes ab Offset 31H. Ab Offset 33H findet sich ein Zeiger (Offset) auf das Segment mit dem Share Record. Ein Eintrag 0000H signalisiert, daß dieser Record nicht existiert. Der Record besitzt folgendes Format:

Offset	Bytes	Funktion
00H	1	Flag (00H Block frei, 01H Block belegt, FFH Ende)
01H	2	Blockgröße
03H	1	Checksumme Pfadname inclusive 00H
04H	2	Offset in SHARE DS mit den Lock-Records
06H	4	Zeiger auf Anfang der SFT mit den Dateien
0AH	2	Unique Sequenz Nummer
0CH	x	ASCIIZ-String mit dem Pfadnamen

Tabelle 7.20: Aufbau des SHARE-Records

Das SHARE.EXE-Programm legt beim Start ein Datensegment mit lokalen Daten an. In diesem Datenbereich werden Informationen über Datei- und Satzsperren festgehalten. Die Datenstruktur besitzt folgenden Aufbau:

Offset	Bytes	Funktion
00H	2	Offset in SHARE Datensegm. nächste Lock Tabelle
02H	4	Offset in Datei Start der Lock Region
06H	4	Offset in Datei Ende der Lock Region
0AH	4	Zeiger in die SFT für die Datei
0EH	2	PSP Segmentadresse Besitzer der Datei

Tabelle 7.21: Aufbau des SHARE-Lock-Records

Die Adressen auf die SHARE-Daten finden sich in der DOS-Datentabelle (List of Lists).

7.10 Der Environmentbereich

DOS legt für jeden Prozeß ein Speichersegment an, in dem Informationen über die Umgebung des Programmes (zum Beispiel der Name des Kommandoprozessors, der Pfad etc.) eingetragen werden. Dieses Segment wird als Environment bezeichnet und besteht aus ASCIIZ-Strings mit dem Format:

```
NAME=Parameter
```

Das Ende des Environmentbereichs wird durch zwei Nullbytes markiert, wobei das erste Nullbyte noch zum letzten ASCIIZ-String gehört. Nachfolgend findet sich ein Beispiel über die Informationen innerhalb des Environments.

```
COMSPEC=C:/COMMAND.COM
PATH=MSDOS;.........
OS=MSDOS 3.2
```

Mit COMSPEC wird der Name und der Pfad des Kommandoprozessors spezifiziert. Mittels des SET-Kommandos lassen sich von der Benutzerebene heraus weitere Parameter setzen.

DOS hinterlegt automatisch die PATH- und PROMPT-Angaben im Environmentbereich. Der erste String innerhalb des Environments bezieht sich auf den Kommandointerpreter. Die Anweisung:

```
COMSPEC=parameter
```

legt den Zugriffspfad und den Namen des Kommandointerpreters fest. Diese Information wird zum Beispiel benötigt, um den transienten Teil von COMMAND.COM zu laden. Weiterhin lassen sich vom Benutzer zusätzliche Strings im Environment ablegen, die zwar keine Bedeutung für DOS besitzen, aber durch Anwenderprozesse ausgewertet werden können (z.B. Angaben über Dateiname und Pfad von Datenbereichen). Hierfür ist in der Datei AUTOEXEC.BAT die Anweisung:

```
SET name=string
```

einzutragen. Wird nur das SET-Kommando ohne weitere Parameter eingetragen, erscheint die aktuelle Belegung des Environments am Bildschirm. Die Eingabe:

```
SET name
```

löscht den Eintrag aus dem Environmentbereich.

Normalerweise begrenzt DOS die Größe des Environments auf 160 Byte. Wird ein residentes Programm geladen, darf in DOS 2.x das Environment die Größe von 127 Byte nicht überschreiten. Ab DOS 3.x läßt sich der Environmentbereich bis zu einer Größe von 32 Kbyte erweitern. Hierzu ist der Befehl:

```
SHELL=<pfc></e:xxxx></p>
```

in die Datei CONFIG.SYS einzutragen. Mit <pfc> ist das Laufwerk, der Pfad und der Name des Kommandointerpreters gemeint. Die Angabe </e:xxxx) spezifiziert die Größe des Environments in Byte. Der Schalter </p> installiert den Kommandoprozessor permanent im Speicher. Mit der Anweisung:

```
SHELL=C:\MSDOS\COMMAND.COM /e:2000 /p
```

wird eine permanente Kopie des Kommandoprozessors installiert und die Länge des Environmentblocks auf 2000 Byte erweitert.

Die Segmentadresse des Environmentbereichs wird im PSP des jeweiligen PSP auf Offset 2CH gespeichert.

An den Environmentblock schließen sich ab DOS 3.0 weitere Initialisierungsparameter an. Im ersten Wort steht ein Zähler, der die Parameteranzahl spezifiziert (normalerweise 0001H). Daran schließt sich ein ASCIIZ-String an, der Laufwerks-, Pfad- und Dateiname des aufgerufenen Programmes enthält (z.B. C:/MSDOS/DEBUG.EXE).

7.11 Der Aufbau des DOS-Program-Segment-Prefix (PSP)

DOS erzeugt beim Laden eines Programmes einen Vorspann von 256 Byte, in dem der Lader verschiedene Informationen über die Programmumgebung ablegt. Der Bereich dient z.B. zur Abspeicherung der Programm-Rücksprungadressen, zur Verwaltung der offenen Einheiten (Dateien) und ähnlichem. Dieser 100H Byte große Bereich wird als Program-Segment-Prefix (PSP) bezeichnet. Nachfolgendes Bild zeigt einen Speicherdump aus einem solchen PSP-Bereich:

```
-d 0 100
1275:0000  CD 20 00 80 00 9A EE FE-1D F0 F5 02 C8 0E 2E 03
1275:0010  C8 0E BD 02 C8 0E B6 0B-01 03 01 00 02 FF FF FF
1275:0020  FF FF FF FF FF FF FF FF-FF FF FF FF 8A 0E 4E 01
1275:0030  00 12 14 00 18 00 75 12-FF FF FF FF 00 00 00 00
1275:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
1275:0050  CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20
1275:0060  20 20 20 20 20 20 20 20-00 00 00 00 00 20 20 20
1275:0070  20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00
1275:0080  00 0D 77 72 69 74 65 2E-61 70 70 0D 6D 0D 53 50
1275:0090  4C 55 53 22 20 67 65 6D-0D 66 31 5D 20 67 65 64
1275:00A0  72 81 63 6B 74 20 68 61-62 65 6E 2C 20 6C 94 73
1275:00B0  63 68 65 6E 20 53 69 65-20 64 69 65 20 6C 65 74
1275:00C0  7A 74 65 20 5A 65 69 6C-65 2E 0D 00 00 00 00 00
1275:00D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
1275:00E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
1275:00F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Bild 7.14: Speicherdump eines PSP-Bereiches

Die Belegung des PSP-Bereiches wird von Microsoft leider nur teilweise veröffentlicht. Viele Bereiche werden als reserviert gekennzeichnet. Die nachfolgende Tabelle gibt den Aufbau des PSP (soweit bekannt) an. Die undokumentierten Bereiche sind durch das Zeichen '*' markiert:

Adr	Code	Bemerkungen
00	CD 20	INT 20 als Programm Ende
02	00 80	Zahl der allocierten Paragraphen
04	00	1 Byte (reserviert)
05	9A EE FE 1D F0	Long Call (5 Byte) to DOS Dispatcher
0A	F5 02	IP Terminate Address
0C	C8 0E	CS "
0E	2E 03	IP Control Break
10	C8 0E	CS "
12	BD 02	IP Critical Error Handler
14	C8 0E	CS "
* 16	B6 0B	Laderidentifikation (Segmentadresse, von der das Programm geladen wurde)
* 18	01 03 01 00 02	Tabelle für 20 Handleeinträge, die ersten 5 Einträge werden durch DOS initialisiert
2B	. . . . FF FF	FFH -> Eintrag nicht belegt
2C	8A 0E	Pointer auf das Environment Segment
* 2E	4E 01 00 12	Adresse Stackbereich
* 32	14 00	Größe der Handletabelle (14H = 20 Einträge)
* 34	18 00	Offsetadresse Handletabelle
* 36	75 12	Segmentadresse Handletabelle
* 38	FF FF FF FF	vorheriges Seg. von SHARE (immer -1)
* 3C	00 .. 00	20 x 00 (vermutl. für High Byte Handle res.)
50	CD 21	INT 21 DOS Function Dispatcher. Eintritt per Long Call auf Adr. 50H. In
52	CD	RETF AH muß der Funktionscode stehen.
* 53	00 .. 00	9 x 00 Ab Offset 55H wird die Tabelle für den erweiterten FCB1 benutzt
		File Control Block 1 (FCB 1)
5C	xx	Drive: 0 = Default, 1 = A: 2 = B: ....
5D	xx .. xx	8 Bytes mit dem Filenamen in ASCII
65	xx xx xx	3 Bytes File Extension in ASCII
68	xx xx	2 Bytes mit Current Block Zähler
6A	xx xx	2 Bytes Record Größe in Bytes (128)

Adr	Code	Bemerkungen
		File Control Block 2 (FCB 2)
6C	xx	Drive: 0 = default, 1 = A: 2 = B: ....
6D	xx .. xx	8 Bytes mit dem Filenamen in ASCII
75	xx xx xx	3 Bytes File Extension in ASCII
78	xx xx	2 Bytes mit Current Block Zähler
7A	xx xx	2 Bytes Record Größe in Bytes (128)
7C	00 00 00 00	4 x 00
		Disk Transfer Bereich
80	xx	Länge des belegten Bereiches in Bytes
81	xx .. xx	n Bytes (max. 127) Beim Start sind hier die Eingabeparameter der Kommandozeile abgelegt. (Ausnahme: bei I/O Umleitung)

Tabelle 7.22: Aufbau des PSP-Segments

Die ersten 2 Byte enthalten eine INT 20-Anweisung zum Beenden des laufenden Prozesses. Dieser Aufruf (s. auch INT 20) wurde aus Kompatibilitätsgründen erhalten, da ab DOS 2.0 der INT 21 die Funktion 4CH (Programm Terminate) enthält. Im nächsten Wort findet sich die Länge des reservierten Programmspeichers in Paragraphen (à 16 Byte). Es ist zu beachten, daß in diesem Wert 10H Paragraphen für das PSP enthalten sind.

Ab Offset 05H findet sich ein *LONG CALL* zum DOS-Funktionsdispatcher (INT 21-Handler). Dabei haben die DOS-Entwickler tief in die Trickkiste gegriffen. Ein *Intrasegment CALL* zu dieser Adresse aktiviert zwar den Dispatcher. Dabei umfaßt der Befehl insgesamt 5 Byte, wobei das erste Byte durch den Opcode belegt ist. Das folgende Doppelwort gibt nun die Zieladresse der Einsprungroutine des INT 21-Dispatchers wieder. Dieser Vektor wird so justiert, daß er gleichzeitig im Word ab Offset 06H die Zahl der verfügbaren Byte im Codesegment beschreibt (s. Kapitel über DOS-COM-Dateien). Diese Doppelbelegung wird durch eine geschickte Ausnutzung der Segment:Offset-Adreßdefinition möglich.

Die nächsten Adressen (Offset 0AH bis 14H) enthalten je 3 Vektoren à 4 Byte, in denen die Adressen folgender Interrupt-Service-Routinen abgelegt sind:

```
Terminate Adresse
Control-C (Break) Adresse
Critical Error Handler Adresse
```

Wenn DOS einen Subprozeß erzeugt, können innerhalb dieses Prozesses die Interruptvektoren für diese Routinen verändert werden (z.B. der Prozeß besitzt einen eigenen Error Handler). Terminiert nun dieser Subprozeß, ohne die Vektoren zu restaurieren, bleibt ein inkonsistenter Systemzustand zurück, da im *parent process* noch auf Routinen Bezug genommen wird, die im nicht existierten Subprozeß liegen. Um dies zu vermeiden, sichert DOS bei der Erzeugung des Subprozesses die drei Originalvektoren im neuen PSP. Wird dieser mit den INT 21-Funktionen 00H (Terminate Programm) oder 4CH (End Prozess) beendet, restauriert DOS automatisch die Interruptvektoren. Der INT 20 benutzt intern ebenfalls die INT 21-Funktion 00H zum Abbruch eines Prozesses. Dadurch wird der Zustand vor Aufruf des Subprozesses erzeugt.

Ab Offset 16 findet sich ein Wert, welcher die Segmentadresse des Programmladers enthält. Damit läßt sich z.B. feststellen, ob das Programm durch COMMAND.COM geladen wurde.

Die nächsten Bytes ab Offset 18H dienen zur Verwaltung der offenen Dateien. Mit den neueren XENIX-orientierten I/O-Funktionen werden einzelne Einheiten oder Dateien über Handlecodes identifiziert. Insgesamt lassen sich so bis zu 20 Handles verwalten. Die Bytes ab Offset 18H bis 2BH enthalten die jeweiligen (max. 20) Handlecodes. DOS vergibt standardmäßig die ersten 5 Handles für die Standard-Ein-/Ausgabeeinheiten. Es gilt folgende Kodierung:

```
 Code °  Handle
------é-------------------
  00  ° Input device
  01  ° Output device
  02  ° Error device
  03  ° Auxiliary device
  04  ° Printer device
```

Die ersten 5 von DOS vergebenen Handles eröffnen diese Standard-Ein-/Ausgabeeinheiten. Nicht benutzte Einträge werden mit der Signatur FFH versehen.

Die Verwaltung der Handletabelle erfolgt über weitere Einträge im PSP. So enthält das Wort ab Offset 32H die Größe der Handletabelle. Der Wert ist üblicherweise auf 20 Handles (0014H) begrenzt. Ab Offset 34 findet sich ein 4-Byte-Vektor, der die Adresse dieser Handletabelle spezifiziert. Durch Auslagern dieser Informationen in andere Speicherbereiche läßt sich die Handletabelle vergrößern. Auch hier arbeiten die Entwickler mit einigen Tricks. Die Handles sind normalerweise 16 Bit Werte, die aber bei maximal 20 Handles immer kleiner als 0FFh sind, so daß das obere Byte unbelegt bleibt. Die Handletabelle enthält also nur die 20 Einträge für die unteren Bytes der Handles. Ab Offset 3CH findet sich aber im PSP ein Bereich mit genau 20 Nullbyte. Dieser Bereich wurde vermutlich bei der Entwicklung für die oberen Bytes der Handletabelle vorgesehen.

Zu den dokumentierten Einträgen gehört die Segmentadresse des Environmentbereichs. Diese findet sich ab Offset 2CH und zeigt auf den aktuellen Prozeß zugeordneten Environmentbereich.

Ab Offset 2EH ist ein 4-Byte-Vektor mit der Adresse des Programm-Stackbereiches abgespeichert.

Die Offsetadressen 50H-52H enthalten das Interface für den DOS-Dispatcher. Dieser besteht aus einem INT 21, gefolgt von einer IRET-Anweisung. Damit besteht die Möglichkeit aus einem Programm heraus die INT 21-Funktionen per LONG CALL zum PSP (Offset 0050H) zu aktivieren. Der Eintrag wurde aus Kompatibilitätsgründen zu früheren DOS-Versionen erhalten.

Ab Offset 5CH ist der File-Control-Block 1 sowie ab Offset 6CH der FCB 2 gespeichert. Erweiterte FCB's benutzen die vorhergehenden und nachfolgenden Bytes.

DOS legt standardmäßig einen DISK-Transfer-Bereich von 127 Byte ab Offset 80H an. Das erste Byte spezifiziert dabei die Länge des belegten Bereichs.

7.12 Der Aufbau der File-Control-Blocks (FCB)

Zur Steuerung der Disk-I/O-Anforderungen benötigt DOS Datenbereiche, in denen bestimmte Kontrollparameter, wie Dateiname, -länge, Recordnummer etc. zwischengespeichert werden. Die älteren CP/M-orientierten INT 21-Funktionen benutzen hierfür sogenannte File-Control-Blocks (FCB). Hierbei handelt es sich um nichts anderes als Datenbereiche, die für jede Datei vom laufenden Prozeß einzurichten sind. Der Aufbau eines solchen FCB ist fest vorgegeben. Betrachten wir zuerst den normalen FCB, der folgenden Aufbau besitzt:

Bytes	Offset	Funktion
1	00	Laufwerknummer (0 = default, 1 = A:, 2 = B: etc.)
8	01	Dateiname in ASCII-Zeichen
3	09	Dateiextension in ASCII-Zeichen
2	0C	Current-Block
2	0E	Record-Größe in Bytes
4	10	Dateigröße in Bytes
2	14	Datum des letzten Schreibzugriffs
2	16	Zeit des letzten Schreibzugriffs
8	18	reserviert
1	20	Current Record
4	21	Relativ Record

Tabelle 7.23: Aufbau des File-Control-Blocks

Im ersten Byte wird der Code für die Laufwerksnummer übergeben. Hierbei gibt es noch eine Besonderheit bezüglich der Kodierung. Vor dem ersten Aufruf gilt die Zuordnung:

```
1 = A:
2 = B:
```

und so weiter. Falls der Wert 0 in diesem Byte übergeben wird, bezieht sich DOS beim ersten Aufruf auf das Default-Laufwerk. Anschließend trägt die Funktion die Nummer des selektierten Laufwerkes in das Feld ein. Generell beginnt die Zuordnung der Laufwerke beim zurückgegebenen Code mit dem Wert 0 (0 = A:, 1 = B: etc.), da ja kein Wert für die Default-Einheit vorgesehen werden muß. Dies ist bei der Software-Entwicklung zu beachten.

Der Dateiname besteht aus 8 ASCII-Zeichen, die linksbündig ab Offset 01H angeordnet und notfalls mit Blanks aufgefüllt werden. Falls mit dem Namen eine Einheit angesprochen wird, dies gilt zum Beispiel für Ausgaben an den Drucker, darf kein Doppelpunkt hinter dem Namen (PRN) eingetragen werden.

Die Extension besitzt 3 ASCII-Zeichen und ist ab Offset 09H nach den gleichen Gesichtspunkten anzulegen wie der Dateiname. Falls die Datei keine Extension besitzt, dürfen die 3 Byte mit Blanks aufgefüllt werden.

Ab Offset 0CH findet sich ein Wort mit dem *Current Block Field*, in welchem die aktuelle Blocknummer verwaltet wird. MS-DOS unterteilt eine Datei in logische Blöcke, wobei jeder Block aus insgesamt 128 logischen Sätzen besteht. Beim Open-File-Aufruf wird die Blocknummer auf den Wert 0 gesetzt, so daß dann der erste Block adressiert wird. Die Länge eines logischen Satzes (Record) wird im nächsten Feld spezifiziert. Über das *Current Block Field*, in Kombination mit dem *Current Record Field*, läßt sich der logische Satz innerhalb einer Datei genau adressieren.

Die Größe eines logischen Satzes (Record) wird ab Offset 0EH in einem eigenen Wort abgespeichert. Sie wird beim Open-File-Aufruf auf den Wert 80H (128 Byte) gesetzt, damit die gelesenen oder geschriebenen Sätze über die im PSP eingerichtete Disk-Transfer-

Area gepuffert werden können. Falls eine andere logische Satzlänge erforderlich wird, kann diese durch einen Eintrag in diesem Feld, allerdings erst nach dem Open-Aufruf, eingetragen werden. Es ist aber darauf zu achten, daß bei Werten über 128 Byte eine entsprechend große DTA durch den Prozeß eingerichtet wird. Falls die alte Einstellung beibehalten wird, überschreibt der Puffer den Programmcode, der sich hinter dem PSP anschließt.

DOS behandelt alle Dateien als gespeicherte »Bytestrings« ohne irgendwelche Voraussetzungen bezüglich der Satzstruktur und des Zugriffsmechanismus zu machen. Die oben beschriebene Aufteilung in logische Blöcke und Sätze erfolgt durch den Anwendungsprozeß und kann in beliebigen Formaten erfolgen. Ab Offset 10H »ird die Dateigröße in Byte als Doppelwort in der gewohnten Notation (Low Word auf der unteren Adresse) verwaltet.

Die folgenden 2 Byte (Offset 14H) enthalten das Datumsfeld der Datei. Um das aktuelle Jahr zu erhalten, ist die Zahl 1980 zum gespeicherten Wert zu addieren. Dies wurde erforderlich, da in den 7 Bits die Jahreszahlen ab 1980 nicht direkt darstellbar sind. Es gilt folgende Belegung dieser Bytes:

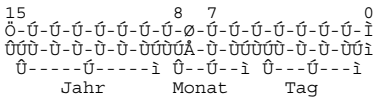


Bild 7.15: Kodierung des Datums im FCB

d.h. das Wort wird in drei Bitfelder (Tag, Monat, Jahr) unterteilt. Die in diesen Feldern enthaltenen Bits werden anschließend als Binärzahlen interpretiert. Es gibt dabei folgende Zuordnung:

```
Tag  :  1 -  31
Monat:  1 -  12
Jahr :  0 - 119  (1980-2099)
```

Das Feld wird bei der Erzeugung und bei jeder Modifikation der Datei aktualisiert.

Wird eine Datei geöffnet, speichert die INT 21-Funktion den Datumseintrag aus dem Inhaltsverzeichnis in dieses Feld. Bei Schreibzugriffen setzt die jeweilige Funktion automatisch das Datums- und Uhrzeitfeld im FCB auf den aktuellen Stand. Eine Abspeicherung innerhalb des Inhaltsverzeichnisses erfolgt aber erst beim CLOSE-FILE-Aufruf.

Ab Offset 16H wird die Uhrzeit (2 Byte) des letzten Schreibzugriffs auf die Datei geführt. Es gilt folgende Kodierung.

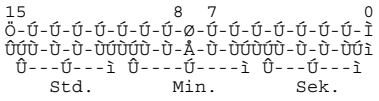


Bild 7.16: Kodierung der Zeit im FCB

Ähnlich wie beim Datumsfeld gelten folgende Wertebereiche:

```
Sek.  :  0 - 59
Min.  :  0 - 59
Std.  :  0 - 23
```

Die Daten werden bei Schreibzugriffen automatisch im FCB aktualisiert. Die Open- und Close-Funktionen lesen und schreiben die Daten innerhalb des Inhaltsverzeichnisses auf dem Speichermedium.

Die nachfolgenden 8 Byte (Offset 18H bis Offset 1FH) sind für interne Zwecke reserviert und werden durch MS-DOS benutzt.

Ab Offset 20H findet sich ein Byte, welches als *Current Record Field* bezeichnet wird. In diesem Feld wird ein Zeiger, der einen der 128 logischen Sätze (Records) adressiert, gespeichert. Zusammen mit dem *Current Block Field* läßt sich so ein logischer Satz innerhalb einer Datei bezeichnen. Dieses Feld wird beim Open-File-Aufruf nicht initialisiert und ist explizit vor einer sequentiellen Schreib-/Leseoperation mit einem Wert (0-27) zu besetzen.

Da DOS nicht nur sequentielle Dateizugriffe unterstützt, sondern auch direkte Schreib-/Leseoperationen auf Dateien erlaubt, wird ein Feld zur Adressierung des jeweiligen Satzes (Record) benötigt. Ab Offset 21H findet sich ein 4-Byte-Vektor, der einen logischen Satz innerhalb der Datei spezifiziert. Der Wert 0 bezieht sich dabei auf den Dateianfang (erster Satz). Zusammen mit dem Satzlängenfeld (Offset 0EH) läßt sich jedes einzelne Byte wahlfrei lesen oder schreiben. Das Feld wird beim Open-File-Aufruf nicht zu null gesetzt. Daher muß vor dem ersten direkten Dateizugriff der Wert initialisiert werden. Falls die Satzlänge (Recordsize) auf einen Wert kleiner als 64 Byte gesetzt ist, benutzen die DOS-Funktionsaufrufe alle 4 Byte des *Relativ Record Fields*. Bei längeren Sätzen werden nur die drei unteren Byte benutzt.

Achtung: Bei der Definition der FCB-Struktur ist den Entwicklern von DOS ein Fehler unterlaufen. Die FCB-Struktur wird im PSP ab Offset 5CH angelegt. Leider sind die FCB's 1 Byte zu lang, so daß das erste Byte (Offset 80H) des Disk Transfer Bereiches überschrieben wird. Bei der Benutzung des FCB darf also der Disk Transfer Bereich nicht verwendet werden. Weiterhin muß vorher der Inhalt des Byte ab Offset 80H gesichert werden, falls die Eingaben der DOS-Kommandozeile ausgewertet werden sollen.

7.13 Der erweiterte File Control Block (Extended FCB)

Mit dem oben besprochenen File-Control-Block lassen sich zwar normale Dateien bearbeiten. Andererseits existieren aber auch Dateien, denen bestimmte Attribute zugeordnet sind. Um Einträge mit bestimmten Dateiattributen im Inhaltsverzeichnis anzulegen oder zu suchen, werden erweiterte File-Control-Blocks benutzt. Diese bestehen aus normalen FCB's, denen weitere 7 Byte vorangestellt werden. Es ergibt sich somit folgende Belegung:

Bytes	Offset	Funktion
1	-07	Markierung Extended FCB = FFH (Flag Byte)
5	-06	reserviert
1	-01	Attributbyte
1	00	Laufwerknummer (0 = default, 1 = A:, 2 = B: etc.)
8	01	Dateiname in ASCII-Zeichen
3	09	Dateiextension in ASCII-Zeichen
2	0C	Current-Block
2	0E	Record-Größe in Bytes
4	10	Dateigröße in Bytes
2	14	Datum des letzten Schreibzugriffs
2	16	Zeit des letzten Schreibzugriffs
8	18	reserviert
1	20	Current-Record
4	21	Relativ-Record

Tabelle 7.24: Aufbau des erweiterten FCB

Das erste Byte (Offset -07H) wird immer mit dem Wert FFH belegt. Daran erkennen die INT 21-Funktionen, daß es sich um einen erweiterten FCB handelt.

Die folgenden 5 Byte (Offset -06H bis -02H) sind für das System reserviert.

Das siebte Byte (Offset -01H) enthält das Attributbyte. Mittels dieses Bytes läßt sich eine Datei mit bestimmten Attributen anlegen. Weiterhin kann die Suche innerhalb des Inhaltsverzeichnisses durch Setzen bestimmter Attributbits gesteuert werden. Für die Dateiattribute gelten folgende Kodierungen:

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙÚÙÚÙÚÙÚÙÚÙÚÙÚì
 ° ° ° ° ° ° ° Û-  Read Only File
 ° ° ° ° ° ° Û---  Hidden File
 ° ° ° ° ° Û-----  System File
 ° ° ° ° Û-------  Volume ID
 ° ° ° Û---------  Subdirectory
 ° ° Û-----------  Archive Bit
 ° Û-------------  ---
 Û---------------  ---
```

Bild 7.17: Kodierung der Dateiattribute

Abhängig von der jeweiligen Funktion lassen sich bestimmte Dateiattribute kombinieren. Beim Aufruf der INT 21-Funktion muß die Adresse des ersten Bytes (Offset -07H) des erweiterten FCB's übergeben werden und nicht die Adresse des Laufwerkcodes, wie dies bei normalen FCB's üblich ist.

Die File-Control-Blocks sind durch den jeweiligen Prozeß einzurichten. Dabei sind die Bytes 0-0FH und 21H-24H durch den Prozeß zu initialisieren, während die Bytes 10H bis 1FH durch DOS verwaltet werden. Solange ein FCB nicht geöffnet ist (unopened FCB), sind nur die vom Anwenderprozeß definierten Felder belegt. Nach der Open-Anweisung sind auch die restlichen Bytes mit Daten ausgefüllt. Bei Feldern, die mehrere Bytes umfassen, werden die niederwertigsten Bytes auf den unteren Adressen abgelegt.

DOS legt standardmäßig zwei FCB im Programmsegment-Prefix-Bereich (PSP) ab Offset 5CH und Offset 6CH an. Es ist aber zu beachten, daß es bei der Ausnutzung der gesamten Datenstruktur zu Überlappungen mit anderen Einträgen kommt. So überdeckt das letzte Byte des Relativ-Record Fields von FCB1 bereits das erste Byte (Offset 80H) der Disk-Transfer-Area (DTA). Weiterhin wird der FCB2 (Offset 6CH) komplett durch die Daten des ersten FCB überlagert. Auch bei Verwendung erweiterter FCB's treten ähnliche Probleme auf. Falls die Datenbereiche nicht in den Speicherbereich des laufenden Prozesses ausgelagert werden, ist dies bei der Softwareentwicklung zu berücksichtigen. Für jede zu öffnende Datei muß ein eigener FCB eingerichtet werden.

8 Die DOS-Einheitentreiber

DOS wickelt zwar aus Sicht der laufenden Prozesse alle Ein-/Ausgabefunktionen zu den jeweiligen Geräten ab. Intern bedient es sich aber für die Grundoperationen der BIOS-Funktionen. Der residente Teil des Programms IO.SYS (IBMBIO.COM) besteht ja im wesentlichen aus den Standard-Einheitentreibern (NUL, CLOCK etc.), die die Verbindung zur Hardware und zu den ROM-BIOS-Routinen übernehmen.

Nach einem Systemstart müssen mindestens 5 residente Einheitentreiber vorliegen. Diese werden normalerweise als Teil der Datei IO.SYS installiert. Ab DOS 2.0 besteht die Möglichkeit, weitere Treiber über die Datei CONFIG.SYS zu installieren. Hierzu ist ein Eintrag der Form:

```
Device = Treibername
```

in der Datei vorzusehen. Beim Systemstart wird die CONFIG.SYS-Datei ausgewertet und die Treiber werden installiert. Dadurch lassen sich auch bereits existierende residente Treiber durch neue installierbare Treiber ersetzen.

8.1 Die Verwaltung der Treiber

Die Anfangsadressen der einzelnen Treiber werden in DOS beim Systemstart ermittelt, da ja die Konfigurierung dynamisch (CONFIG.SYS) erfolgt. Um nun die Lage eines einzelnen Treibers später ermitteln zu können, legt DOS eine verkettete Liste im Speicher an. Jeder Treiber wird nun in diese Liste eingetragen. Hierzu enthält er im Programmkopf einen Zeiger (DWORD) zum nächsten Eintrag (Adresse des folgenden Treibers). Der letzte Eintrag in dieser Kette enthält den Wert -1,-1 (FFFFH) als Markierung.

8.2 Das Format der Einheitentreiber

Die Einheitentreiber werden in MS-DOS nicht wie normale Programme behandelt. Einmal wird kein PSP-Bereich angelegt. Vielmehr handelt es sich hier um *Image Files*, d.h. die Datei mit dem Code des Treibers liegt als getreues Speicherabbild vor. Die Datei kann dabei mit den Extensionen COM oder EXE abgespeichert werden.

Insbesondere bei COM-Dateien ist aber darauf zu achten, daß der Programmcode nicht mit der Assembler-ORG-100-Anweisung versehen wird. Falls dies zutrifft, ist das Programm nicht mehr lauffähig, da der Lader den Code ab Offset 0 plaziert. Alle direkten Sprünge innerhalb des Codesegments beziehen sich aber auf Adressen mit dem Offset 100H. Bei EXE-Dateien nimmt der Lader eine automatische »Entrelativierung**Fehler! Verweisquelle konnte nicht gefunden werden.**Zu Beginn muß ein Datenbereich (Deviceheader) eingerichtet werden. Dieser dient zur Aufnahme spezifischer Informationen über diese Einheit.

Bytes	Feld
4	Next Device Pointer (Zeiger auf den nächsten Treiber)
2	Attribute
2	Zeiger auf die Strategieroutine
2	Zeiger auf die Interruptroutine
8	Einheitenname

Tabelle 8.1: Struktur des Kopffeldes eines Einheitentreibers

Next Device Pointer

Die ersten vier Byte enthalten einen Zeiger (Low Word = Offset) auf die Anfangsdresse des nachfolgenden Treibers. Der Wert dieses Doppelwortes muß vor dem Aufruf bereits in der Datei auf den Wert -1,-1 (FFFFH,FFFFH) initialisiert werden. Dieser Wert signalisiert das Ende der Treiberkette. Wird ein neuer Treiber hinzugeladen, fügt DOS die Anfangsadresse in das Feld *Next Device Pointer* des vorhergehenden Treibers ein. Da das Feld Next-Device-Pointer des neuen Treibers bereits mit dem Wert -1,-1 initialisiert wurde, ist das Ende der Treiberkette automatisch markiert.

Das Attributfeld

Dieses Feld dient zur Identifikation des Treibers. Grundsätzlich werden zwei Treibertypen unterschieden:

- Zeichenorientierte Treiber, die eine serielle Ausgabe von Bytes an Einheiten wie CON, AUX, PRN erlauben. Den Treibern wird ein logischer Ein-/Ausgabekanal zugeordnet, der sich entweder über die FCB- oder über die Handlefunktionen ansprechen läßt (Open, Close etc.). Ein zeichenorientierter Treiber kann jeweils immer nur eine physikalische Einheit unterstützen, da diesem Treiber auch nur ein Name zugeordnet wird.
- Blockorientierte Treiber, zur Ansteuerung von Festplatten oder Diskettenlaufwerken. Diese Einheiten erlauben einen direkten Zugriff auf einzelne Daten auf den Medien. Diese Daten werden zu logischen Blöcken zusammengefaßt, die oft auf physikalische Bereiche (z.B. Sektoren) abgebildet werden. Im Gegensatz zu den zeichenorientierten Einheiten wird dem blockorientierten Treiber kein Name zugewiesen. Daher läßt sich eine solche Einheit auch nicht öffnen oder schließen (die Open-Anweisung bezieht sich ja auf eine Datei!). Die Zuordnung der logischen Einheit zu den jeweiligen Aufrufen erfolgt über die Laufwerkscodes (A, B, C, ...), wobei ein Treiber durchaus mehrere Laufwerke unterstützen kann. Die Zahl der unterstützten Laufwerke muß im Treiber angegeben sein. Welche logischen Laufwerke vom jeweiligen Treiber unterstützt werden, ist abhängig von der Lage innerhalb der Treiberkette. Die logischen Laufwerksbezeichnungen werden in aufsteigender Reihenfolge mit den Buchstaben des Alphabets bezeichnet (A, B, C, ...). Unterstützt ein Treiber zwei Einheiten und wird er als erste blockorientierte Einheit in die Kette geladen, sind automatisch die Laufwerke A und B zugeordnet. Der nächste blockorientierte Treiber

erhält die Laufwerke ab dem Buchstaben C. Insgesamt lassen sich so bis zu 26 Laufwerke (A bis Z) unterstützen.

Bei der Vergabe der Attribute sind diese zwei Einheitentypen zu unterscheiden. Das Attributwort besitzt folgende Kodierung:

```
     --- zeichenorientierte Treiber ---
   Bit 15 = 1 zeichenorientierte Einheit
   Bit 14 = 1 Der Treiber unterstützt IOCTL-Aufrufe
            0 Der Treiber unterstützt keine IOCTL-Aufrufe
   Bit 13 =   Bei zeichenorientierten Treibern gilt:
            1 Das »Output until busyFehler! Verweisquelle konnte nicht gefunden werden.
0 Kein »Output until busyFehler! Verweisquelle konnte nicht gefunden werden.   Bit 12 =
reserviert (muß zu 0 gesetzt werden)
              (oder Gerät wird lokal betrieben, ab DOS 3.1)
   Bit 11 = 1 Der Treiber unterstützt Open und Close
   Bit 10 =   reserviert (muß zu 0 gesetzt werden)
     .          .
   Bit  8 =     .
   Bit  7 =   Treiber unterstüützt IOCTL-Query (DOS 5.0)
   Bit  6 = 1 Der Treiber unterstützt Get/Set Logical Device und
              Generic IOCTL Aufrufe
            0 Der Treiber kennt keine Get/Set Logical Device
              oder Generic IOCTL Aufrufe
   Bit  5 =   definiert den Raw- oder Cooked-Mode
   Bit  4 =   Treiberausgabe per INT 29
   Bit  3 = 1 CLOCK$ Device Treiber
   Bit  2 = 1 NUL Device Treiber

   Bit  1 = 1 Standard Output (CON) Device Treiber
   Bit  0 = 1 Standard Input (CON) Device Treiber

     --- blockorientierte Treiber ---
   Bit 15 = 0 blockorientierte Einheit
   Bit 14 = 1 Der Treiber unterstützt IOCTL-Aufrufe
            0 Der Treiber unterstützt keine IOCTL-Aufrufe
   Bit 13 =   Bei blockorientierten Treibern gilt:
            1 Medium besitzt keine FAT ID
            0 Medium besitzt eine FAT ID
   Bit 12 =   reserviert (muß zu 0 gesetzt werden)
              Funktion 4409: Laufwerk lokal (DOS 3.1)
   Bit 11 = 1 Der Treiber unterstützt auswechselbare Medien
            0 Der Treiber unterstützt nur fixe Speichermedien
   Bit 10 =   reserviert (muß zu 0 gesetzt werden)
     .          .
   Bit  9 =     .
   Bit  8 =   Wird von DRIVER.SYS benutzt
   Bit  7 =   Funktion 4411H wird unterstützt
   Bit  6 = 1 Der Treiber unterstützt Get/Set Logical Device und
              Generic IOCTL Aufrufe
            0 Der Treiber kennt keine Get/Set Logical Device
              oder Generic IOCTL Aufrufe
   Bit  5 =   reserviert (muß zu 0 gesetzt werden)
     .          .
   Bit  2 =   reserviert (muß zu 0 gesetzt werden)
   Bit  1 = 1 Treiber unterstützt Sektornummern mit 32 Bit
   Bit  0 = 0 reserviert
```

Tabelle 8.2: Kodierung des Device-Attributfeldes

Nachfolgend wird die Bedeutung dieser Attribute kurz erklärt.

Bit 15

In diesem Bit wird der Einheitentyp selektiert. Es gilt die Zuordnung

0 = blockorientierte Einheit, 1 = zeichenorientierte Einheit.

Bit 14

Bit 14 signalisiert, ob der Treiber Kontrollzeichen verarbeiten kann.

Diese Kontrollzeichen werden durch die INT 21-Funktion 44H (IOCTL) mit den Untercodes AL = 2 bis AL = 5 übergeben. Das IOCTL-Bit gilt sowohl für blockorientierte, als auch für zeichenorientierte Treiber. Falls das Bit gelöscht (0) ist, unterstützt der Treiber keine IOCTL-Funktionen. Der Versuch, mittels der Funktion 44H solche Zeichen zu senden, führt zu einer Fehlermeldung. Ist das Bit gesetzt, lassen sich über den IOCTL-Funktionsaufruf Steuerzeichen (Control Characters) an die Einheit senden, die dann entsprechend interpretiert werden (z.B. Ctrl-Z als Dateiende).

Bit 13

Hier unterscheidet sich die Bedeutung in Abhängigkeit vom Einheitentyp. Bei blockorientierten Einheiten (siehe Bit 15) signalisiert das Bit, wie der Treiber den Typ des Speichermediums ermittelt. Ist das Bit = 1, dann wird die Information im BIOS-Parameter-Block (BPB) geführt. Falls die Einheit ein *Media Descriptor Byte* führt, um den Typ zu identifizieren, ist Bit 13 = 0 zu setzen.

Bei zeichenorientierten Einheiten deutet das Bit an, ob der Treiber den Status (output until busy) des Peripheriegerätes auswertet. Bei Druckern werden die Daten z.B. nur dann gesendet, wenn der Status auf »bereit**Fehler! Verweisquelle konnte nicht gefunden werden.**nicht bereit**Bit 12**

Diese Bit ist reserviert und muß immer zu Null gesetzt werden.

Bit 11

Falls das Bit = 1 ist, unterstützt der Treiber auswechselbare Speichermedien. Andernfalls ist das Bit zu löschen. Dieses Bit ist nur bei den DOS-Versionen 3.0 bis 3.3 gültig.

Bit 10-7

Diese Bits sind ebenfalls reserviert und müssen zu 0 gesetzt werden.

Bit 6

Hier handelt es sich um das Generic-IOCTL-Bit für beide Einheitentypen.

Dieses Bit signalisiert, ob der Treiber Generic-IOCTL-Funktionsaufrufe (INT 21-Funktion 44H, AL = 0CH und AL = 0DH) unterstützt. In diesem Fall ist das Bit zu setzen. Dieses Bit wird erst ab PC-DOS-Version 3.2 (MS-DOS-Version 3.0) unterstützt.

Bit 5-4

Diese Bits sind reserviert und müssen zu 0 gesetzt werden.

Bit 3

Dieses Bit ist für zeichenorientierte Einheiten vorgesehen und markiert die Clock Device. Falls dieses Bit gesetzt ist, ersetzt DOS den internen CLOCK$-Treiber durch das neu geladene Modul.

Bit 2

Auch dieses Bit bezieht sich auf zeichenorientierte Treiber und ist für die NUL-Einheit vorgesehen. Ist das Bit gesetzt, handelt es sich um die Nulleinheit, die aber nicht auf andere Einheiten umgeleitet werden kann. Ein gesetztes Bit 2 zeigt lediglich an, daß innerhalb von DOS eine NUL-Einheit benutzt wird.

Bit 1

Ist dieses Bit gesetzt (1) handelt es sich um die Standard-Ausgabeeinheit (CON). Wird ein neuer Treiber geladen, bei dem dieses Bit gesetzt ist, dann ersetzt DOS den internen Treiber durch die neue Einheit.

Bit 0

Hier handelt es sich um das Indikatorbit für die Standard-Eingabeeinheit (CON). Ein gesetztes Bit beim Laden des Treibers veranlaßt die Deaktivierung des internen StandardInput-Treibers. Damit ist der neu installierte Treiber aktiviert.

Bei Block-Device-Treibern ist ab DOS 4.0 eine Unterstützung von 32-Bit-Sektornummern erforderlich, da ab dieser Version Disks mit mehr als 32 Mbyte Kapazität unterstützt werden.

8.3 Zeiger zu den Strategie- und Interruptroutinen

Die Entwickler von MS-DOS planten bereits beim Entwurf der Einheitentreiber die Weiterentwicklung des Betriebssystems in Richtung Multi-Tasking-Fähigkeiten. In einer solchen Umgebung sind die I/O-Anforderungen asynchron zu den laufenden Prozessen zu bearbeiten. Nach einem Funktionsaufruf kann nicht gewartet werden, bis die Daten Ein-/Ausgabe abgewickelt ist, da ja während dieser Zeit kein anderer Prozeß Zugang zu dieser Funktion erhält. Aus diesem Grunde werden alle I/O-Anforderungen in einer Auftragsliste gespeichert und sequentiell abgearbeitet. Sobald ein Auftrag in dieser Liste übernommen wurde, kann der Prozeß die I/O-Funktion wieder freigeben.

Die MS-DOS Einheitentreiber sind für diesen Multi-Tasking-Betrieb bereits vorbereitet, wenn auch das Betriebssytem diesen Mode nicht unterstützt. Hierzu bietet jeder Treiber zwei Einsprungpunkte, eine Strategieroutine und eine Interruptroutine. Soll eine Daten-Ein-/Ausgabe erfolgen, wird ein Auftrag an die Strategieroutine abgesetzt. Dieser Auftrag wird dann durch die Interruptroutine bearbeitet. Sobald dieser Auftrag abgewickelt wurde, setzt die Interruptroutine eine Fertigmeldung (Done Flag) ab, die durch MS-DOS periodisch überprüft wird. Anschließend wird die Kontrolle wieder an den rufenden Prozeß zurückgegeben.

Im Datenkopf des Treibers finden sich nun zwei Zeiger (Word) auf die Strategie- und Interruptroutine. Da es sich nur um 16-Bit-Zeiger handelt, müssen die zwei Routinen im gleichen Codesegment wie der Datenkopf liegen.

Der Aufbau des Treibers sowie die Abarbeitung der I/O-Anforderungen werden auf den folgenden Seiten beschrieben.

Das Feld mit dem Einheitennamen

An den Zeiger mit dem Einsprungpunkt der Interruptroutine schließt sich ein Feld mit dem Namen des Treibers an. Hierbei handelt es sich um einen 8 Byte langen Bereich, der bei zeichenorientierten Treibern den Einheitennamen (CON, AUX, etc.) linksbündig angeordnet enthält. Falls der Name kürzer als 8 Zeichen ist, sind die restlichen Bytes mit Blanks aufzufüllen. Zusätzlich darf der Einheitenname nicht mit einem Doppelpunkt abgeschlossen werden. Eine Angabe »CON:**Fehler! Verweisquelle konnte nicht**

gefunden werden. Bei blockorientierten Treibern wird in diesem Feld die zugeordnete Einheit abgespeichert. Optional besteht die Möglichkeit, vor dem Aufruf im ersten Byte die Zahl der unterstützten Einheiten abzulegen. Anschließend ersetzt DOS diesen Wert mit dem Ergebnis des Treiberinitialisierungsaufrufes.

8.4 Installation eines Einheitentreibers

Die Installation eines Einheitentreibers läßt sich nach obigen Ausführungen relativ einfach ausführen. Hierzu sind folgende Schritte notwendig:

- Der Treiber muß als COM- oder EXE-Datei mit einem Code/Datenbereich ab Offsetadresse 0 vorliegen.
- Zu Beginn des Codesegments findet sich der Datenkopf des Treibers.
- Dieser Datenbereich muß entsprechend initialisiert werden. (Next Device Pointer = -1,-1, Attributbyte, Strategie- und Interruptpointer, Einheitenname).

Dann wird der Dateiname des Treibers in CONFIG.SYS eingetragen. Beim nächsten Systemstart installiert das Programm IO.SYS zuerst diese Treiber. Wird hier bereits der Name eines Standardtreibers (z.B. CON) gefunden, unterbleibt die Aktivierung des »residentenDer Aufruf eines Treibers durch DOS

DOS installiert die Treiber während des Systemstarts dynamisch innerhalb des Speichers. Die Aufrufe erfolgen später über FAR CALLS, wobei sich die Adressen an Hand der Treiberliste ermitteln lassen.

Wie aber bereits in den vorhergehenden Abschnitten beschrieben, besitzt jeder Treiber zwei Einsprungpunkte. Die Strategieroutine übernimmt die DOS-Ein-/Ausgabeaufträge, während die Interruptroutine diese Aufträge abwickelt. Zuerst ruft DOS deshalb die Strategieroutine auf und anschließend die Interruptroutine.

Da die Treiber im Hinblick auf einen Einsatz in einer Multi-Tasking-Umgebung entworfen wurden, ist ein besonderer Übergabemode für die Funktionsparameter erforderlich. Eine direkte Verwaltung über die Prozessorregister (wie bei den INT 21-Funktionsaufrufen) ist hier nicht möglich, da ja unter Umständen mehrere Anforderungen gleichzeitig abzuwickeln sind. Hier haben sich die Entwickler einen einfachen, aber sehr wirkungsvollen Mechanismus einfallen lassen. Falls DOS eine I/O-Funktion benötigt, legt es intern einen Datenbereich (Request Header) mit den Steuerparametern an. Anschließend wird die Strategieroutine des Treibers aufgerufen, die alle I/O-Aufträge in einer Liste zwischenspeichert. (Bei DOS besitzt diese Liste allerdings immer nur einen Eintrag). Beim Aufruf enthält das Registerpaar ES:BX einen Zeiger auf den internen DOS-Datenbereich (Request Header) mit den Steuerinformationen. Dieser Zeiger wird von der Strategieroutine in der Auftragsliste zwischengespeichert. Anschließend aktiviert DOS dann die Interruptroutine (ohne Parameter), um den Auftrag auszuführen.

Bei der Implementierung eines Treibers sind einige Punkte zu beachten. Die Aufrufe der Treiberroutinen erfolgen über FAR CALLS, so daß diese ebenfalls mit FAR RETURNS

abzuschließen sind. Weiterhin muß der Treiber beim Aufruf den Zustand des Prozessors (Register) sichern. DOS stellt auf dem Systemstack genügend Raum zur Verfügung, um zirka 20 Parameter zu sichern. Falls der Treiber für interne Verwaltungszwecke einen größeren Stackbereich benötigt, ist dieser Bereich durch den Treiber einzurichten. Vor Rückkehr zum DOS-Prozeß ist der alte Prozessorzustand wieder herzustellen.

Die Struktur der Aufrufparameter (Request Header)

DOS richtet vor Aufruf der Strategieroutine einen Datenbereich mit den Steuerinformationen ein und übergibt die Adresse im Registerpaar ES:BX. Dieser Datenbereich besitzt folgende Struktur:

Bytes	Feld
1	Länge des Parameterblocks in Bytes
1	Einheitencode
1	Kommandocode der Treiberfunktion
2	Statusword
8	reserviert
.	funktionsspezifischer Anhang mit den Funktionspara-
.	metern

Tabelle 8.3: Aufbau des »Request HeadersLängenfeld

Das erste Byte gibt die Länge des Datenbereichs in Bytes an. Der Parameterblock umfaßt den Headers sowie einen Anhang, dessen Aufbau und Länge von der aufgerufenen Treiberfunktion abhängt.

Das Einheitencode-Feld

Dieses Feld besitzt nur bei blockorientierten Einheitentreibern eine Bedeutung. Hier wird angegeben, auf welche Einheit sich die I/O-Anforderung bezieht. Dies ist erforderlich, da blockorientierte Treiber mehrere Einheiten unterstützten können. Sind zum Beispiel 4 Einheiten definiert, dürfen die Einheitencodes die Werte 0, 1, 2, 3 annehmen, wobei eine 1 sich auf die zweite Einheit bezieht. Welches logische Laufwerk dann angesprochen wird, hängt von der Lage des Treibers innerhalb der Treiberkette ab (siehe Beschreibung Attributfeld).

Das Kommandocode-Feld

Ein Treiber muß in der Lage sein, Funktionen wie:

- Initialisierung
- Überprüfung des Speichermediums
- Aufbau des BIOS-Parameter-Blocks (BPB)
- Ermittlung des Media-Descriptor-Bytes

- Daten-Ein-/Ausgabe
- Statusmeldungen
- Puffer leeren
- Open/Close-Funktionen

auf Anfrage auszuführen. Der Wert im Kommandocode Feld steuert die jeweilige Funktion des Treibers. Die folgende Tabelle gibt die implementierten Treiberfunktionen wieder:

Code	Funktion
0	Init (Initialisierungsaufruf bei der Installation)
1	Media Check (nur bei Blockdevices gültig)
2	Build BPB (nur bei Blockdevices gültig)
3	IOCTL Input Control String (nur falls IOCTL Bit = 1)
4	Input (Read)
5	Non Destructive Input no Wait (nur Zeichendevices)
6	Input Status (nur bei zeichenorientierten Einheiten)
7	Input Flush (nur bei zeichenorientierten Einheiten)
8	Output (Write)

Code	Funktion
9	Output with verify (Write)
10	Output Status (nur bei zeichenorientierten Treibern)
11	Output Flush (nur bei zeichenorientierten Treibern)
12	IOCTL Output Control String (nur falls IOCTL Bit = 1)
13	Device Open (nur falls OPEN/CLOSE/RM Bit = 1 ist)
14	Device Close (nur falls OPEN/CLOSE/RM Bit = 1 ist)
15	Removable Media (nur bei blockorientierten Treibern, sofern das OPEN/CLOSE/RM Bit = 1)
16	Output until busy (bei zeichenorientierten Treibern)
19	Generic IOCTL Request
23	Get Logical Device
24	Set Logical Device
25	IOCTL Query (DOS 5.0)

Tabelle 8.4: Kommandocodes für die Einheitentreiber (Ende)

Allerdings sind einige Codes nicht in allen DOS-Versionen implementiert. Die Open/Close-Aufrufe für *Removal Media* (Code 13, 14, 15) werden erst ab DOS 3.x unterstützt. Die Funktionen 19, 23, 24 sind nur in DOS 3.2 und DOS 3.3 implementiert.

Die Bedeutung dieser einzelnen Funktionen wird im folgenden Abschnitt besprochen.

Das Statusfeld

In diesem Feld findet sich der Status- und Fehlercode des Treibers. Das Feld ist in 16 Bit unterteilt, deren Bedeutung je nach Bitkombination wechselt. Es gilt folgende grobe Einteilung:

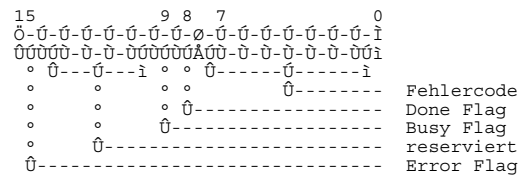


Bild 8.1: Kodierung des Treiber-Statusfeldes

Das Feld ist vor dem Aufruf der Strategieroutine des Treibers zu löschen. Nach dem Aufruf kann das Statuswort ausgewertet werden. Die Bits 10 bis 14 sind für zukünftige Erweiterungen vorgesehen und damit reserviert.

Falls das *Error Flag* (Bit 15) gesetzt ist, enthalten die unteren 8 Bit einen Fehlercode. Die folgende Tabelle enthält die entsprechenden Fehlercodes, mit den zugeordneten Meldungen.

Code	Fehlermeldung
00	Verletzung des Schreibschutzes
01	unbekannte Einheit
02	Einheit nicht bereit
03	unbekannter Kommandocode
04	CRC-Fehler
05	Länge der Aufrufstruktur des Laufwerkes falsch
06	SEEK-Fehler
07	unbekanntes Speichermedium
08	Sektor nicht gefunden
09	Drucker Papierende Meldung
0A	Schreibfehler
0B	Lesefehler
0C	Fehler ohne weitere Aufschlüsselung der Ursache
0D	reserviert
0E	reserviert
0F	Wechsel der Disk nicht erlaubt

Tabelle 8.5: Fehlercodes der Einheitentreiber

Bit 9 wird als *Busy Flag* genutzt und wird bei Statusabfragen gesetzt. Eine Abfrage, ob der Treiber auswechselbare Speichermedien (Removable Media) unterstützt, beeinflußt dieses Bit ebenfalls.

Das Bit 8 (*Done Flag*) dient zur Signalisierung, daß der Auftrag abgewickelt wurde (siehe auch Zeiger zu den Strategie- und Interruptroutinen). Das Bit wird durch die Interruptroutine auf den Wert 1 gesetzt, bevor die Kontrolle an DOS zurückgeht.

8.5 Die Funktionen der Einheitentreiber (DOS 2.0 - 6.0)

Nachdem DOS die Strategieroutine aufgerufen hat, die Adresse des *Request Headers* wurde im Registerpaar ES:BX übergeben, wird anschließend die Interruptroutine aktiviert. Diese übernimmt den Zeiger auf den Request-Header aus der Auftragsliste und wertet das Kommandofeld aus. Dann wird die entsprechende Treiberfunktion bearbeitet. Auf den nachfolgenden Seiten werden diese einzelnen Funktionen vorgestellt.

8.6 INIT (Code = 0, DOS 2.0 - 6.0)

Mit dem Wert 0 im Kommandofeld des Request-Headers führt der Treiber eine Initialisierung aus. Diese Routine wird nur einmal bei der Installation des Treibers aufgerufen. Der beim Aufruf der Strategieroutine im Registerpaar ES:BX übergebene Zeiger adressiert den Request-Header, an den aber weitere Felder angehängt sind. Die Länge des Datenbereiches (einschließlich Header) wird im ersten Byte angegeben. Beim INIT-Aufruf liegt daher folgende Struktur vor:

Bytes	Feld
1	Länge des Parameterblocks (17H Byte)
1	Einheitencode
1	Kommandocode = 0
2	Statusword
8	reserviert
1	Zahl der unterstützten Einheiten
4	Endadresse des residenten Treiberteils
4	Zeiger auf den BIOS-Parameter-Block
1	Laufwerksnummer (ab DOS 3.1)
2	CONFIG.SYS Error Message Control Flag (ab DOS 4.0)

Tabelle 8.6: Datenstruktur des Parameterblocks beim INIT-Aufruf

Number of Units

Im ersten Feld hinter dem Request-Header tragen blockorientierte Treiber die Zahl der unterstützten Einheiten ein. Anhand dieses Wertes weist DOS die Laufwerksbezeichnungen an den Treiber zu. Sind z.B. alle Laufwerke bis C bereits zugewiesen und ist das Feld mit dem Wert 3 belegt, ordnet DOS dem Treiber die logischen Laufwerke D, E und F zu (siehe auch »Das Attributfeld«. Bei zeichenorientierten Treibern bleibt das Feld unbelegt, da diese jeweils nur eine Einheit unterstützen.»End Address

Das nächste Feld enthält einen 4-Byte-Adreßzeiger (Low Word Offset) auf das Ende des residenten Treiberteils. Da die Initialisierung nur einmalig erforderlich ist, vertritt DOS die Strategie, nur einen residenten Kern im Speicher zu halten. Die Initialisierungsroutinen lassen sich dann durch den nachfolgenden Treiber, oder durch andere Prozesse überschreiben. Beim INIT-Aufruf muß deshalb der Treiber die Endadresse des residenten Teils ermitteln und in diesem Feld eintragen.

BIOS-Parameter-Block (BPB) Adreßfeld

```
Dieses Feld erhält eine Doppelfunktion in Verbindung mit »installierbarenDEVICE = MOUSE
<Param 1><Param 2><RET>
```

Dem Treiber MOUSE werden hier zwei Parameter übergeben (z.B. Schrittweite in x- und y-Richtung). Wie gelangt nun aber der Treiber an diese Informationen?

Hier haben die Entwickler zu einem Trick gegriffen. Die Datei CONFIG.SYS wird durch SYSINIT zeilenweise in den Speicher gelesen und ausgewertet. Das Zeilenende wird jeweils durch ein Return- oder Linefeed-Zeichen markiert. Beim Aufruf der INIT-Routine enthält das BIOS-Parameter-Block Feld die Speicheradresse der eingelesenen Kommandozeile (Offset auf den niederwertigsten Adressen) ab dem Gleichheitszeichen:

```
DEVICE = \Path\Devicename  /Parameter<RET>
         ^
      Position des Adreßzeigers
```

Damit kann der Treiber diesen String auslesen und auf eventuell vorhandene Parameter analysieren. Innerhalb dieses Treibers dürfen zu diesem Zeitpunkt nur die DOS-Funktionsaufrufe 01H-0CH und 30H benutzt werden.

Bei blockorientierten Einheiten legt der Treiber während des INIT-Aufrufes die Adresse einer Tabelle ab. Diese enthält für jede vom Treiber unterstützte Einheit einen Zeiger (nur Offset) auf den zugeordneten BIOS-Parameter-Block (BPB). Beim BPB handelt es sich um eine Datenstruktur, in der Informationen über den Aufbau der Einheit abgelegt sind. Mit diesen Informationen strukturiert DOS die gespeicherten Daten. Im Abschnitt über die Build BPB-Funktion finden sich weitere Hinweise über diesen BPB. Falls alle unterstützten Einheiten vom gleichen Typ sind, braucht nur ein BPB angelegt werden. Es ist allerdings für jede Einheit ein Zeiger auf diesen Block zu definieren. Bei der Implementierung des Treibers ist darauf zu achten, daß alle Datenbereiche im residenten Teil des Treibers liegen, da sie sonst durch andere Treiber überlagert werden.

Das Feld mit der Laufwerksnummer

Ab DOS 3.1 enthält der INIT-Parameterblock (ES:BX) im letzten Byte die Information über das erste zugeordnete Laufwerk innerhalb des Treibers. Es gilt die übliche Zuordung (0 = A:, 1 = B:, etc.). Die Initialisierungsfunktion eines Treibers muß nach dem Aufruf verschiedene Aktionen ausführen:

- Die Zahl der unterstützten Einheiten im entsprechenden Feld setzen (nur bei Block-Devices).
- Die Endadresse des residenten Treiberteils ermitteln und in das entsprechende Feld übertragen.
- Die Tabelle mit den Zeigern auf die BPBs aufbauen und die Tabellenadresse in das entsprechende Feld übertragen.
- Eventuell erforderliche Initialisierungscodes zu den physikalischen Einheiten senden.
- Den Status der Einheit ermitteln und in das Statuswort des Request-Header übertragen.
- Das *Media Descriptor Byte* ermitteln und abspeichern. Hierbei handelt es sich um ein Byte, welches intern verwaltet wird und Informationen über das verwaltete Medium gibt (siehe auch die Beschreibung der entsprechenden Funktion in diesem Kapitel).
- Den Kommandostring aus CONFIG.SYS analysieren und gegebenenfalls die Parameter übernehmen.

Es kann allerdings vorkommen, daß während der Initialisierung ein Fehler auftritt (z.B. die Einheit ist gestört oder nicht vorhanden). In diesem Fall sollte der Treiber die Installation abbrechen. Hierzu sind folgende Daten im Parameterblock zu setzen.

- Die Zahl der unterstützten Einheiten = 0 setzen
- Den Offsetwert im Feld *End-Adresse* auf 0 setzen
- Die Segmentadresse im Feld *End-Adresse* auf den Wert des aktuellen Codesegmentes einstellen.

Dadurch wird sichergestellt, daß kein Speicherplatz von diesem Treiber belegt wird.

Falls mehrere Treiber in einer Datei abgelegt sind, muß die Endadresse im Parameterblock eines jeden Treibers auf die Endadresse des letzten Treibers gesetzt werden. Falls die Implementierung dies nicht berücksichtigt, besteht die Gefahr, daß ein Teil des Codebereiches überschrieben wird. Daher müssen alle Treiber innerhalb dieser Datei beim INIT-Aufruf die letzte Codeadresse zurückgeben.

Ab DOS 4.0 wird der Parameterblock um ein Wort mit dem *CONFIG.SYS Error Message Flag* erweitert. Wird der Wert des Flags vor dem Aufruf ungleich 0 gesetzt, gibt das System folgende Meldung aus, falls bei der Initialisierung ein Fehler auftritt:

```
Error in CONFIG.SYS line xxx
```

8.7 Media Check (Code = 1);

Dieser Aufruf ist nur bei Block-Einheiten gültig. Wird im Request-Header das Feld mit dem Kommandocode auf den Wert = 1 gesetzt, führt die Funktion nach dem Aufruf das Kommando *Media Check* aus. Der im Register ES:BX adressierte Datenblock besitzt das in Tabelle 8.7 dargestellte Format:

Mit diesem Aufruf läßt sich prüfen, ob ein Speichermedium sich geändert hat (z.B. Diskettenwechsel von 360 Kbyte auf 1,2 Mbyte). Dies ist z.B. bei Open/Close, Rename- oder Delete-Funktionsaufrufen erforderlich.

Bytes	Feld
1	Länge des Parameterblocks (13H Bytes)
1	Einheitencode
1	Kommandocode = 1
2	Statusword
8	reserviert
1	DOS Media Descriptor Bytes
1	Returncode
4	Zeiger auf »Volume ID**Fehler! Verweisquelle konnte nicht gefunden werden.** (nur ab DOS 3.0)

Tabelle 8.7: Aufbau des »Request Header Fehler! Verweisquelle konnte nicht gefunden werden.**Das DOS-Media-Descriptor-Byte**

In diesem Feld legt DOS das intern gespeicherte *Media Descriptor Byte* ab. Damit kann der Treiber diesen Wert mit dem aktuell vorliegenden Code vergleichen.

Das Media-Descriptor-Byte ist gemäß folgender Notation kodiert:

```
Bit
0  1 = zweiseitig       0 = einseitig
1  1 = 8 Sektoren       0 = keine 8 Sektoren
2  1 = removable        0 = nicht removable
```

Bits 3-7 sind immer auf 1 zu setzen. Beim INT 21, AX = 3200H und in folgender Tabelle finden sich mögliche Codes zur Kennzeichnung des Mediums.

Type	Seiten	Sektoren/ Track	Media-Descriptor-Byte
Fixed Disk	--	--	F8H
5.25 Zoll	2	15	F9H
5.25 Zoll	1	9	FCH
5.25 Zoll	2	9	FDH
5.25 Zoll	1	8	FEH
5.25 Zoll	2	8	FFH
8 Zoll	1	26	FEH
8 Zoll	2	26	FDH
8 Zoll	2	8	FEH
3.5 Zoll	2	9	F9H
3.5 Zoll	2	18	F0H

Zur Bestimmung, ob ein Medium zweiseitig benutzbar ist, sollte ein Leseversuch auf der zweiten Seite unternommen werden. Tritt ein Fehler auf, ist das Medium nur einseitig benutzbar.

Returncode

Nach dem Aufruf prüft der Treiber an Hand der Einträge in der File-Allocation-Table (FAT) das Medium. Das Ergebnis wird als Code im Feld *Returncode* abgelegt. Es gilt folgende Zuordnung:

```
 1  Speichermedium wurde nicht gewechselt
 0  Nicht feststellbar, ob das Medium gewechselt wurde
-1  Speichermedium gewechselt
```

Der Treiber besitzt verschiedene Möglichkeiten, um festzustellen, ob das Speichermedium gewechselt wurde:

- Bei Festplatten wird sofort der Status »not changed**Fehler! Verweisquelle konnte nicht gefunden werden.**· Falls die Tür des Laufwerkes überwacht wird, kann ebenfalls sofort entschieden werden, ob ein Wechsel vorliegt.

· Falls weniger als 2 Sekunden seit dem letzten Zugriff vergangen sind, wird ebenfalls der Status »not changed Fehler! Verweisquelle konnte nicht gefunden werden.**· Die FAT des Mediums wird gelesen und die Media-Bytes oder die *Volume ID* werden verglichen. Bei unterschiedlichen Werten wird der Status »changed**Fehler! Verweisquelle konnte nicht gefunden werden.nicht feststellbar, ob das Medium gewechselt wurde**Fehler! Verweisquelle konnte nicht gefunden werden.Volume-ID-Zeiger**

Ab DOS 3.0 existiert ein weiteres Feld, welches zur Aufnahme eines Zeigers dient. Dieses Feld wird nur benutzt, falls im Attributfeld das Bit 11 (Removable Media) gesetzt ist und

der Treiber den Code *Media Changed* zurückgibt. In diesem Fall ist ein Zeiger auf den Speicherbereich mit dem alten *Volume ID Label* in diesem Feld abzulegen. Wenn im Treiber keine Identifikation des Volume-Labels möglich ist, sollte der Zeiger auf einen ASCIIZ-String mit dem Text »NO NAME**Fehler! Verweisquelle konnte nicht gefunden werden.**Falls nicht bekannt ist, ob das Medium gewechselt wurde, lagert DOS alle modifizierten Puffer auf das Medium aus und baut einen neuen BPB auf. Falls das Medium gewechselt wurde, löscht DOS alle Puffer und baut anschließend den BPB neu auf.

Im Media-Descriptor-Byte ist der Typ des Speichermediums kodiert:

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙ-Ù-Ù-ÙÚÙÙÙÚÙÚÙ¿
 Û---Ú---ì ° ° Û---  1   doppelseitiges Speichermedium
      °     ° °       0   einseitiges Speichermedium
      °     ° °
      °     ° Û-----  1   8 Sektoren pro Spur
      °     °         0   mehr als 8 Sektoren pro Spur
      °     °
      °     Û-------  1   Medium wechselbar
      °               0   Medium fest installiert.
      °
      Û-------------  diese Bits sind immer gesetzt (1)
```

Bild 8.2: Kodierung des Media-Descriptor-Bytes

Die Bits 3 bis 7 sind immer gesetzt (1). Die unteren 3 Bit dienen zur Kodierung des Speichertyps. Falls alle unteren 4 Bit gelöscht sind (0F0H), dann ist der Typ des Mediums nicht definiert.

8.8 Build BPB (Code = 2, DOS 2.0 - 6.0)

Falls eine Media-Change-Abfrage mit dem Ergebnis »changed**Fehler! Verweisquelle konnte nicht gefunden werden.**unbekanntDer Aufruf ist nur bei blockorientierten Einheiten zulässig. Im ersten Feld wird der Typ des Speichermediums, gemäß der vorgeschriebenen Kodierung, abgelegt. Diese Kodierung des Media-Descriptor-Byte wurde bereits im vorhergehenden Abschnitt beschrieben.

Das nächste Feld enthält einen 4-Byte-Adreßzeiger auf einen internen Puffer (Transfer-Puffer-Adresse). Dieser Puffer muß so groß bemessen sein, daß ein kompletter Sektor darin gespeichert werden kann. Er dient zur Aufnahme der Daten zur Identifizierung des Speichermediums.

Bytes	Feld
1	Länge des Parameterblocks (16H Byte)
1	Einheitencode
1	Kommandocode = 2
2	Statusword
8	reserviert
1	DOS Media Descriptor Byte
4	Transfer Puffer Adresse
4	Zeiger auf den BPB

Tabelle 8.8: Der Request Header beim Build BPB

Zum Aufbau des BIOS-Parameter-Blocks (BPB) muß der Treiber den Typ des Speichermediums identifizieren. MS-DOS speichert diesen Typ in Form des Media-Descriptor-Byte

Offset	Bytes	Bedeutung
00	3	In DOS 2.x finden sich hier 3 Byte (0E0H, xx, xx) mit einem Near JUMP. Ab DOS 3.X findet sich hier hier ein Short JUMP (0EBH, XX) und ein NOP (90H)
03	8	ASCII-Text mit dem OEM-Namen und der Version
0B	2	Bytes pro Sektor
0D	1	Sektoren pro Zuordnungseinheit als 2er Potenz
0E	2	Zahl der reservierten Sektoren (ab log. Sektor 0)
10	1	Zahl der File-Allocation-Tables
11	2	max. Zahl von Einträgen im Hauptinhaltsverzeichnis
13	2	Gesamtzahl der logischen Sektoren

Offset	Bytes	Bedeutung
15	1	Media Descriptor Byte
16	2	belegte Sektoren pro FAT
18	2	Sektoren pro Spur
1A	2	Zahl der Köpfe
1C	2	Zahl der »hidden**Fehler! Verweisquelle konnte nicht gefunden werden.**

Tabelle 8.8: Aufbau des Bootsektors

Die Codes zur Beschreibung des Speichermediums reichen von F8H bis FFH (siehe Abschnitt Volume ID-Zeiger). In neuen DOS-Versionen (ab 2.0) wird deshalb der Typ des Speichermediums durch eine neue Methode bestimmt.

Der Treiber liest den Bootsektor des Mediums in den Transfer-Puffer. Bei DOS 2.0 bis 3.3 wird der BIOS-Parameter-Block anschließend aus dem Bootsektor erstellt. In diesem ist nämlich eine Kopie des kompletten BPB abgelegt. Die Tabelle 8.9 enthält z.B. den Aufbau des Bootsektors.

Ab Offset 0BH finden sich die Informationen, die direkt in den BPB-Bereich kopiert werden können. Im Anschluß an diese Daten finden sich 3 Worte, die das Speichermedium spezifizieren. Diese Daten werden vom Treiber ausgewertet und dienen zur optimaleren Einstellung auf die Speichereinheit. So kann z.B. der gleiche Typ des Speichermedien für Platten mit unterschiedlicher Kopfzahl gelten. Dann kann der Treiber diese Information aus dem Bootsektor (Offset 1AH) lesen.

Bei älteren DOS-Versionen (1.x) muß das Media-Descriptor-Byte durch Lesen des FAT ID-Byte aus der Directory bestimmt werden. Bei DOS 1.x ist dies auch eindeutig möglich, da hier nur zwei Formate (FEH und FFH) unterstützt werden.

Der Inhalt des Attributbyte im Request Header bestimmt den Inhalt des Transferpuffers. Falls Bit 13 (No FAT ID) gesetzt (1) ist, sind eventuell im Puffer vorhandene Daten ungültig. Andernfalls wird in DOS 1.x der erste Sektor der File-Allocation-Tabelle (FAT) in diesen Puffer geladen. Das erste Byte im Puffer wird als *FAT ID* bezeichnet, da hier der Typ des Speichermediums kodiert ist.

Im nächsten Feld findet sich die Adresse des BIOS-Parameter-Blocks (BPB). Dieser Block wird durch den Treiber angelegt. Hierzu dienen entweder die Informationen des FAT ID-Bytes, oder der Block wird aus dem Bootsektor konstruiert. Der BPB besitzt den in Tabelle 8.9 beschriebenen Aufbau:

Offset	Bytes	Bedeutung
00	2	Bytes pro Sektor
02	1	Sektoren pro Zuordnungseinheit als 2er Potenz
03	2	Zahl der reserv. Sektoren (ab log. Sektor 0)
05	1	Zahl der File-Allocation-Tables
06	2	max. Zahl von Einträgen im Hauptinhaltsverz.
08	2	Gesamtzahl der logischen Sektoren
0A	1	Media Descriptor Byte
0B	2	belegte Sektoren pro FAT

Tabelle 8.9: Aufbau des BIOS-Parameter-Blocks (BPB)

Anschließend wird die Adresse dieser Tabelle im Aufrufparameterfeld (Request Header) abgespeichert. Falls der Treiber wechselbare Speichermedien unterstützt und das *Volume Label* lesen kann, wird das neue Label bestimmt. Andernfalls ist in diesem internen Feld ein ASCIIZ-String mit dem Text »NO NAME**Fehler! Verweisquelle konnte nicht gefunden werden.**Ab DOS 4.0 müssen Blocktreiber Festplatten größer als 32 Mbyte unterstützen. Deshalb definiert DOS ab der Version 4.0, in Abhängigkeit von der Größe des Mediums, eine erweiterte Struktur für den BPB. Bei Medien kleiner als 32 Mbyte kann die bisherige Struktur beibehalten werden. Andernfalls gilt folgende Struktur des BPB.

Bytes	Feld
2	Bytes pro Sektor
1	Sektoren pro Cluster
2	Zahl der reservierten Sektoren
1	Zahl der FATs (0 falls kein FAT vorhand.)
2	Max. Zahl d. Einträge im Hauptverzeichnis
2	Zahl der Sektoren des Mediums ist der Wert 0 -> extended BPB
1	Media Descriptor Byte
2	belegte Sektoren pro FAT
2	Sektoren pro Spur
2	Zahl der Köpfe
4	Zahl der »hidden**Fehler! Verweisquelle konnte nicht gefunden werden.**
4	Erweiterung für Platten > 32 Mbyte Gesamtzahl der logischen Sektoren als 32-Bit-Zahl

Die Struktur des erweiterten BPB entspricht im wesentlichen der bisherigen Struktur. Lediglich am Ende wird ein 4-Byte-Bereich mit der Zahl der logischen Sektoren (32 Bit) angehängt. Weiterhin muß im Feld mit der Zahl der Sektoren (Offset 8) der Wert 0 stehen. Der extended BPB ist immer dann zu benutzen, falls die Zahl der Sektoren nicht mit 16 Bit darstellbar ist. In allen anderen Fällen ist die Sektorenzahl ab Offset 8 abzulegen. In DOS 4.0 wird allerdings auch ein extended Bootrecord angelegt. Dieser besitzt folgenden Aufbau:

Bytes	Feld
2	2 Byte SHORT JMP (EBH) + NOP (90H)
8	ASCII-Text mit OEM-Namen und Version
25	extended BPB
1	physikalische Laufwerksnummer
1	reserviert
1	extended Bootrecord Signatur
4	Seriennummer des Mediums
11	Label des Speichermediums (ASCII)
8	reserviert

Unterstützt der Treiber 32 Bit Sektornummern, ist im Header des Devicetreibers das Bit 1 im Attributfeld auf 1 zu setzen.

Die physikalische Laufwerksnummer ist in DOS 4.0 immer 00H oder 80H. Die Signatur für den extended Bootrecord ist immer 29H.

8.9 Read/Write (Code 3, 4, 8, 9, 12 und 16, DOS 2.0 - 6.0)

Mit diesem Aufruf können Daten von der Einheit gelesen oder geschrieben werden. Dies betrifft sowohl den Transfer von Daten als auch den Austausch von Steuerzeichen (IOCTL Funktion 44H). Beim Aufruf der Strategieroutine wird ein Zeiger im Register ES:BX übergeben. Dieser adressiert den Datenblock mit den Steuerparametern (Request Header).

Bytes	Feld
1	Länge des Parameterblocks (1AH Byte)
1	Einheitencode
1	Kommandocode = 3, 4, 8, 9, 12, 16
2	Statusword
8	reserviert

Bytes	Feld
1	DOS Media Descriptor Byte (vom BPB)
4	Transfer Puffer Adresse
2	Bytes (Sektoren) Zähler
2	Startsektor
4	Zeiger auf den »Volume ID**Fehler! Verweisquelle konnte nicht gefunden werden.**

Tabelle 8.10: Request Header bei Read/Write

Den Kommandocodes sind folgende Read/Write-Funktionen zugeordnet:

Code	Funktion
3	IOCTL-Read
4	Read
8	Write
9	Write with Verify
12	IOCTL-Write
16	Output until Busy

Tabelle 8.11: Kommandocodes für die Read/Write-Funktionen

Die Funktion 16 (Output until Busy) ist nur bei zeichenorientierten Treibern zulässig. Die Kommandocodes sind im Parameterblock im Feld *Kommandocode* einzutragen.

Das Feld Media-Descriptor-Byte enthält den Code des Speichertyps, ermittelt aus dem BIOS-Parameter-Block (BPB).

Das nächste Feld enthält einen 4-Byte-Zeiger auf den Transferpuffer mit den Ein-/ Ausgabedaten. Dieser Zeiger wird durch MS-DOS gesetzt. Die Schreib-/Leseaufrufe

werden dann abhängig vom Treibertyp ausgeführt. Während Blockdevices nur ganze Sektoren bearbeiten, lesen oder schreiben zeichenorientierte Treiber einzelne Bytes.

Sobald eine I/O-Anforderung abgewickelt wurde, ist das Statuswort im Request Header zu setzen. Zusätzlich muß die Zahl der transferierten Bytes (zeichenorientierte Einheit) oder Sektoren (blockorientierte Einheit) im Parameterfeld gesetzt werden. Diese Zeichenzahl sollte auch gesetzt werden, falls der Transfer wegen eines Fehlers abgebrochen wurde. Bei Aufrufen der IOCTL-Funktionen führt der Treiber keine Fehlerprüfungen durch.

Ist der Funktionscode 9 gesetzt (Write with Verify), dann muß der Treiber diese Operation vornehmen. Dies ist bei der Implementierung zu beachten, da ja auch denkbar wäre, daß DOS mittels eines zweiten Read-Aufrufes die Verifizierung vornimmt.

Im Feld *Startsektor* wird vor dem Aufruf der Startsektor innerhalb des Speichermediums eingetragen. Damit ist der Treiber in der Lage, die Daten zu transferieren. Das Feld hat nur bei blockorientierten Einheiten Bedeutung. Bei einem Aufruf können maximal 64 Kbyte übertragen werden. Dies liegt an der MS-DOS-Architektur, die sich stark an die 64-Kbyte-Segmentierung der 8086-Prozessoren anlehnt. Bei der Definition des Datenpuffers ist auf die Lage im Segment zu achten. Falls der Puffer in der Mitte eines 64-Kbyte-Segments liegt, verursacht eine Transferanforderung von mehr als 32 Kbyte einen Adreßüberlauf und der Anfang des Segments wird als Fortsetzung des Datenpuffers gesehen. Der Treiber muß diesen Bereich aber nicht bearbeiten, so daß nicht alle Daten transferiert werden.

Ab DOS 3.0 existiert ein weiteres Feld, in dem der Treiber einen Zeiger auf den ASCIIZ-String mit dem *Volume ID* ablegt. Dieser Zeiger ist nur definiert, falls der Fehlercode 0FH (Invalid Disk Change) auftritt. In diesem Fall kann DOS den Benutzer auffordern, die korrekte Diskette einzulegen. Der ASCIIZ-String mit dem *Volume ID* wird beim Aufruf der Build-BPB-Funktion angelegt. Es besteht allerdings noch das Problem zu erkennen, ob ein Diskettenwechsel stattgefunden hat. Ein Vergleich des alten und neuen *Volume ID* Strings zeigt zwar einen Wechsel an. Dies kann aber durchaus legitim sein, falls keine offenen Dateien mehr vorliegen und eine neue Diskette eingelegt wurde. Der Treiber muß sich also intern den Zustand, abgeleitet aus den Open- und Closeaufrufen, des Dateisystems merken. Liegt ein Wechsel vor, obwohl noch Dateien geöffnet sind, ist der Fehlercode 0FH auszugeben.

Der Funktionsaufruf *Output until Busy* ist nur bei zeichenorientierten Treibern zulässig. Damit läßt sich der Treiber mit der Ausgabegeschwindigkeit besser an das Peripheriegerät anpassen. Der Treiber wertet das Busy-Statussignal des Gerätes aus. Ist dieser Eingang gesetzt, unterbricht der Treiber die Ausgabe. Dies ist z.B. bei Druckern von Vorteil, da diese oft interne Spooler besitzen. Der Treiber kann solange Zeichen senden, bis das Busy-Signal erkannt wird. Ohne dieses Signal muß die Zeichenausgabe an die Druckgeschwindigkeit angepaßt werden, da sonst Daten verloren gehen. Dies bedeutet natürlich eine erhebliche Optimierung bezüglich der Ausgabegeschwindigkeit, wenn ein Busy-Signal ausgewertet werden kann.

Vor Rückkehr nach DOS muß der Treiber das Statusbyte innerhalb des Parameterfeldes setzen.

An den Request-Header ist in DOS 4.x (s. Tabelle 8.10) bei den Funktionen 3,4,8,9 und 12 ein 4-Byte-Feld mit der 32-Bit-Startsektornummer anzuhängen. Gleichzeitig ist Bit 1-1 im Attribut des Device-Headers zu setzen.

8.10 Nondestructive Read no Wait (Code = 5, DOS 2.0 - 6.0)

Mit diesem Aufruf lassen sich Eingabepuffer auf Zeichen untersuchen, ohne daß diese Zeichen entfernt werden. Dies kann z.B. erforderlich sein, wenn überprüft werden soll, welches Zeichen als nächstes im Puffer gelesen wird. Vor dem Aufruf ist wieder ein Parameterblock anzulegen (Request Header), der folgenden Aufbau besitzt:

Bytes	Feld
1	Länge des Parameterblocks (0EH Bytes)
1	Einheitencode
1	Kommandocode = 5
2	Statusword
8	reserviert
1	gelesenes Byte

Tabelle 8.12: Request Header bei Nondestructive Read no Wait

Die Anfangsadresse dieses Parameterblocks wird im Registerpaar ES:BX übergeben. Im Kommandofeld ist der Wert 5 einzutragen. Bei einer zeichenorientierten Einheit wird das Busy-Bit überprüft. Falls dieses 0 ist, liegen Zeichen im Puffer vor. Dann wird das nächste Zeichen aus dem Eingabepuffer gelesen und im Parameterblock im letzten Feld zurückgegeben. Der Aufruf entfernt das Zeichen nicht aus dem Puffer und verändert diesen auch sonst nicht.

Vor der Rückkehr nach MS-DOS ist dann das Statuswort zu setzen.

8.11 Status (Code 6 und 10, DOS 2.0 - 6.0)

Mit dieser Funktion läßt sich prüfen, ob Daten zur Ein-/Ausgabe vorliegen. Hierzu ist nur der Parameterblock (Request Header) aufzubauen (Kommandocode 6 oder 10) und dessen Anfangsadresse im Registerpaar ES:BX zu übergeben.

Bytes	Feld
1	Länge des Parameterblocks (0DH Bytes)
1	Einheitencode
1	Kommandocode = 6, 10
2	Statusword
8	reserviert

Tabelle 8.13: Request Header bei Status

Der Treiber ermittelt den Status und gibt das Ergebnis vor Rückkehr nach DOS im entsprechenden Feld des Parameterblocks zurück.

Bei Ausgaben an zeichenorientierte Einheiten wird das Busy-Bit (Bit 9) beeinflußt. Falls der letzte Auftrag noch nicht abgewickelt wurde (z.B. das Zeichen kann nicht ausgegeben werden), ist nach dem Aufruf das Bit gesetzt. Dies bedeutet, daß DOS auf das Ende der Ausgabe warten muß. Ist Bit 9 = 0, läßt sich sofort ein neuer Auftrag bearbeiten (Device not Busy).

Bei zeichenorientierten Einheiten mit Eingabepuffern zeigt das Busy-Bit (Bit 9) den Zustand des Treibers an. Ist das Bit gesetzt (1), dann wurde gerade ein Leseaufruf an die Einheit abgesetzt und der Treiber wartet auf die Antwort. Falls Zeichen im Eingabepuffer vorliegen, ist das Bit nach dem Aufruf gelöscht (0). Bei Abfragen des Tastaturpuffers signalisiert Bit 9 = 0, daß der Benutzer Eingaben über die Tastatur vorgenommen hat. Falls die Einheit keinen Eingabepuffer unterstützt, muß der Treiber immer den Busy-Status = 0 zurückgeben, damit DOS nicht auf die Fertigmeldung des Treibers wartet und damit blockiert wird.

8.12 Flush (Code 7 und 11, DOS 2.0 - 6.0))

Mit diesem Aufruf wird der Treiber angewiesen, alle anstehenden Aufträge abzubrechen und zu löschen. Es ist ein normaler *Request Header* anzulegen, wobei die Anfangsadresse im Registerpaar ES:BX übergeben wird.

Bytes	Feld
1	Länge des Parameterblocks (0DH Bytes)
1	Einheitencode
1	Kommandocode = 7, 11
2	Statusword
8	reserviert

Tabelle 8.14: Request Header bei Flush

Der Aufruf dient hauptsächlich zur Löschung der Eingabepuffer bei zeichenorientierten Einheiten. Der Treiber muß also die geforderte Aktion ausführen, das Statuswort setzen und dann zu DOS zurückkehren.

8.13 Open/Close (Code 13 und 14, DOS 3.0 - 6.0)

Diese Aufrufe sind erst ab der DOS-Version 3.0 (bis 4.x) implementiert. Vor dem Aufruf ist ein normaler 13-Byte-Parameterblock (Request Header) aufzubauen.

Bytes	Feld
1	Länge des Parameterblocks (0DH Bytes)
1	Einheitencode
1	Kommandocode = 13, 14
2	Statusword
8	reserviert

Tabelle 8.15: Request Header bei Open/Close

Die Anfangsadresse wird im Registerpaar ES:BX übergeben. Die Funktion kann nur aufgerufen werden, falls im Attributwort des Treibers das Open/Close/RM-Bit gesetzt ist.

Bei blockorientierten Einheiten verwalten diese Aufrufe die internen Datenpuffer innerhalb des Treibers. Zusätzlich erhält der Treiber die Information, wieviele Dateien aktuell geöffnet sind. Dazu wird ein interner Zähler geführt, der durch jeden Open Call erhöht und durch einen CLOSE-Befehl erniedrigt wird. Ein Wert = 0 bedeutet, daß keine geöffneten Dateien mehr vorliegen. Dann können die internen Puffer gelöscht werden. Ein Wechsel des Speichermediums (z.B. Diskettenwechsel) ist dann erlaubt, d.h. es darf keine Fehlermeldung ausgelöst werden (siehe Read/Write). Allerdings können Probleme bei Benutzung der INT 21-FCB-Funktionen auftreten. Hier kann ein Prozeß durchaus mehrere Dateien öffnen, ohne sie später durch CLOSE-Aufrufe zu schließen. Damit ist keine wirksame Kontrolle mehr möglich (ein Grund mehr, die handleorientierten Aufrufe zu nutzen). Falls ein Build BPB-Aufruf erfolgt, ohne daß der interne Zähler auf dem Wert 0 steht, sollte dieser auf 0 gesetzt werden. Die Puffer sind aber dann nicht zu löschen.

Bei zeichenorientierten Treibern kann der Open-Aufruf benutzt werden, um Initialisierungssequenzen zu den Peripheriegeräten zu senden. Dies könnte z.B. eine Format-Einstellung für Drucker oder Plotter sein. Damit lassen sich auch neue Zeichenfonts leicht selektieren. Beim Close-Befehl kann ebenfalls eine Steuersequenz an die physikalische Einheit übertragen werden (z.B. Seitenwechsel). Bei der Open/Close-Funktion kann ein interner Zähler die Zahl der offenen Einheiten speichern. Hier ist aber zu beachten, daß die Standard-Einheiten (STDIN, STDOUT, STDERR, STDAUX und STDPRN) mit den Handlecodes 0ü4 immer geöffnet sind, da sie von allen Prozessen benutzt werden.

8.14 Removable MEDIA (Code 15, DOS 3.0 - 6.0)

Auch diese Funktion ist erst ab DOS 3.x implementiert und kann nur durch DOS ausgeführt werden, wenn im Attributfeld des Treibers das Bit 11 (Removable Media Support) gesetzt ist. Vor dem Aufruf ist ein Parameterblock (nur der Request Header) mit dem Kommandocode 15 anzulegen.

Bytes	Feld
1	Länge des Parameterblocks (0DH Bytes)
1	Einheitencode
1	Kommandocode = 15
2	Statusword
8	reserviert

Tabelle 8.16: Request Header bei Removable Media

Die Anfangsadresse wird im Registerpaar ES:BX übertragen. Über den INT 21-Funktionsaufruf 44H (IOCTL) kann ein Prozeß überprüfen, ob die Speichereinheit ein auswechselbares Medium besitzt (Floppy-Disk oder Festplatte). Das FORMAT-Programm benutzt z.B. diesen Aufruf, um abhängig vom Medium verschiedene Benutzermeldungen auszugeben. Der Aufruf ist nur bei blockorientierten Einheiten erlaubt. Das Ergebnis der Abfrage wird im Busy-Bit (Bit 9) des Statusfeldes (im Request Header) zurückgegeben. Falls dieses Bit gesetzt (1) ist, handelt es sich um ein fixes Medium (z.B. Festplatte). Andernfalls kann das Medium gewechselt werden (z.B. Diskette). Der Aufruf gibt keine Fehlermeldungen zurück.

8.15 Generic IOCTL Request (Code 19, DOS 3.2 - 6.0)

Diese Funktion wird nur bei Treibern von DOS 3.2 bis DOS 4.x implementiert. DOS legt vor dem Aufruf einen Parameterblock (Request Header) an, dessen Anfangsadresse beim Aufruf im Registerpaar ES:BX übergeben wird.

Bytes	Feld
1	Länge des Parameterblocks (17H Byte)
1	Einheitencode
1	Kommandocode = 19
2	Statusword
8	reserviert

Tabelle 8.17: Request Header bei Generic IOCTL Request

Bytes	Feld
1	Major Function
1	Minor Function
2	SI Registerinhalt
2	DI Registerinhalt
4	Zeiger auf die Generic IOCTL Sequenz

Tabelle 8.17: Request Header bei Generic IOCTL Request (Ende)

Damit lassen sich die INT 21-Generic-IOCTL-Request-Funktionen (440CH und 440DH) abwickeln.

Der INT 21-Aufruf 440CH bezieht sich nur auf zeichenorientierte Einheiten und ist erst ab DOS 3.3 implementiert. Er erlaubt es, daß Peripheriegerät mit neuen Zeichensätzen (Code Pages) auszustatten, oder aus der Menge der geladenen Fonts einen auszuwählen. An die 13 Byte des Request-Headers schließen sich weitere Aufrufparameter an, die durch den Treiber ausgewertet werden.

Major Function Field

In diesem Byte wird die Information übergeben, um welche Einheit es sich handelt.

Code	Einheit
00	unbekannt
01	COM
03	CON
05	LPT

Minor Function Field

Das zweite Byte enthält die Aktionen, die vom Treiber auszuführen sind.

Code	Aktion
4CH	Prepare Start
4DH	Prepare End
4AH	Select
6AH	Query selected
6BH	Query prepare List

Inhalt der SI- und DI-Register

Die nächsten beiden Werte dienen zur Aufnahme der Register SI (Source Index) und DI (Destination Index).

Zeiger auf das IOCTL-Request Paket

Im letzten Feld findet sich ein 4-Byte-Zeiger auf einen Datenbereich, in dem ein Parameterblock mit den Code-Page-Informationen abgespeichert ist.

Der genaue Aufbau der Parameterblöcke, sowie der Umfang der auszuführenden Funktionen wird im Kapitel über den INT 21-Aufruf 440CH beschrieben. Sobald sich der Aufruf auf zeichenorientierte Einheiten bezieht, ist die Funktion 440CH auszuführen. Vor Rückkehr nach DOS sind die Einträge im Statusfeld zu setzen.

Bei blockorientierten Einheiten ist die Funktion 440DH auszuführen. Diese ist nur bei den DOS-Versionen 3.2 und 3.3 in den Treibern implementiert. Der Aufruf erlaubt es, die Parameter der blockorientierten Einheit zu lesen oder zu setzen, sowie Spuren auf dem Speichermedium zu formatieren oder zu verifizieren.

Major Function Field

Dieses Feld enthält beim Aufruf 440DH immer den Wert 08H, so daß auch hier eine Unterscheidung zum Aufruf 440CH (Major Codes 00-05) möglich ist.

Minor Function Field

Dieses Feld enthält den entsprechenden Steuercode für die gewünschte Funktion:

Code	Funktion
40	Set Device Parameter
60	Get Device Parameter
41	Write Track
61	Read Track
42	Format and Verify Track
62	Verify Track

Alle diese Funktionen beziehen sich auf logische Einheiten und sind durch den Treiber zu implementieren.

Inhalte der SI- und DI-Register

Die nächsten beiden Werte enthalten die Inhalte der Register SI und DI.

Zeiger auf das IOCTL-Request-Paket

Das letzte Feld enthält einen 4-Byte-Zeiger auf einen Datenbereich mit den Steuerparametern. Der Aufbau dieses Datenfeldes ist abhängig von der jeweiligen Unterfunktion und wird im Rahmen der INT 21-Funktion 440DH beschrieben.

8.16 Get Logical Device (Code 23, DOS 3.2 - 6.0)

Mit diesem Aufruf läßt sich prüfen, ob eine logische Einheit einem blockorientierten Gerät zugeordnet ist. DOS erstellt einen Parameterblock und übergibt die Anfangsadresse im Registerpaar ES:BX. Dieser Block besitzt folgenden Aufbau.

Byte	Feld
1	Länge des Parameterblocks
1	Einheitencode
1	Kommandocode = 23
2	Statusword
8	reserviert

Tabelle 8.18: Request Header bei Get Logical Device (Ende)

Im Feld Einheitencode wird das entsprechende Laufwerk angegeben. Die Felder *Command Code* und *Status* entsprechen den Feldern innerhalb der ersten 13 Byte (Request Header). Nach dem Aufruf findet sich im Feld Einheitencode die Nummer des letzten zugeordneten logischen Laufwerkes. Weitere Informationen finden sich bei der Beschreibung der INT 21-Funktion 440EH.

8.17 Set Logical Device (Code 24, DOS 3.2 - 6.0)

Dieser Aufruf erlaubt es, einer blockorientierten Einheit bestimmte logische Laufwerksnamen zuzuordnen. DOS erstellt einen Parameterblock und übergibt die Anfangsadresse im Registerpaar ES:BX. Dieser Block besitzt den in Tabelle 8.19 dargestellten Aufbau.

Im Feld Einheitencode wird das entsprechende Laufwerk angegeben. Die Felder *Command Code* und *Status* entsprechen den Feldern innerhalb der ersten 13 Byte (Request Header). Nach dem Aufruf findet sich im Feld Einheitencode die Nummer des letzten zugeordneten logischen Laufwerkes.

Diese Funktion ist erforderlich, wenn einem physikalischen Laufwerk mehrere logische Einheiten zugeordnet sind. Dies kann z.B. bei einem Diskettenlaufwerk der Fall sein, dem die Buchstaben A: und B: zugewiesen sind, da kein zweites Laufwerk existiert. Bei dem Befehl Diskcopy wird das Laufwerk abwechselnd mit den Namen A: und B: angesprochen. Falls gerade der Name A: selektiert ist, schaltet der Treiber nach dem Aufruf auf das Laufwerk B: um. Ein zweiter Aufruf stellt wieder den alten Zustand her. Weitere Informationen finden sich bei der Beschreibung der INT 21-Funktion 440FH.

Byte	Feld
1	Länge des Parameterblocks
1	Einheitencode
1	Kommandocode = 24
2	Statusword
8	reserviert

Tabelle 8.19: Request Header bei Set Logical Device

8.18 IOCTL Query (Code 25, DOS 6.0)

Dieser Aufruf erlaubt die Prüfung, ob der Treiber ein bestimmte Kombination der Funktion 19 (13H) Generic IOCTL unterstützt. DOS erstellt einen Parameterblock und übergibt die Anfangsadresse im Registerpaar ES:BX. Dieser Block besitzt den in Tabelle 8.20 dargestellten Aufbau. Weitere Informationen finden sich bei der Beschreibung der INT 21-Funktion 4410H und 4411H.

Byte	Feld
1	Länge des Parameterblocks
1	Gerätekennziffer
1	Kommandocode = 25
2	Statusword
8	reserviert
1	Gerätelkategorie (Ergebnis)
1	Subcode
2	reserviert
2	reserviert
4	unbenutzt

Tabelle 8.20: Query IOCTL

Die reservierten Felder sind für zukünftige Erweiterungen reserviert und sollten nicht benutzt werden.

9 Die DOS-Dateiverwaltung

Die Daten auf Disketten/Festplatten werden durch die blockorientierten DOS-Treiber verwaltet. Grundsätzlich besitzt DOS keine Vorgaben bezüglich der anzulegenden Formate, vielmehr werden die Daten als Bytestrom beliebiger Länge betrachtet.

Andererseits wird aus Effizienzgründen die Aufzeichnung auf dem physikalischen Medium in Blöcken vorgenommen. Ein Schreib-/Lesezugriff bezieht sich immer auf einen gesamten Block und nicht auf einzelne Bytes. Die Daten werden dabei in Spuren gehalten, die bei Disketten/Festplatten als konzentrische Kreise ausgebildet sind. Eine solche Spur wird zusätzlich in einzelne Abschnitte, im folgenden Segmente genannt, aufgeteilt. Die Zahl der Spuren und der Segmente pro Spur ist abhängig vom Speichermedium und wird teilweise beim Formatieren festgelegt. Weiterhin kann das Medium mehrere Speicheroberflächen besitzen, auf denen jeweils ein Schreib-/Lesekopf die Daten bearbeitet.

Aus Sicht der Anwenderprogramme liegen die Daten ebenfalls in strukturierter Form vor. Alle Daten werden erst einmal in logischen Einheiten (Dateien) zusammengefaßt und verwaltet. Innerhalb dieser Dateien lassen sich die Einzelbytes dann noch satzweise betrachten. Hier sind verschiedene logische Formate denkbar und auch in Anwendung.

Die Einheitentreiber sowie die DOS-Dateiverwaltung sind für die Abbildung der logischen Strukturen aus Anwendersicht auf das physikalische Speichermedium zuständig. DOS legt sich (im Gegensatz zu CP/M) hierzu Informationen (Media-Descriptor-Byte) auf dem Medium mit ab. Dadurch lassen sich verschiedene Formate automatisch erkennen und verwalten. In diesem Kapitel werden Aufbau und Funktion der DOS-Speichermedien und der Dateiverwaltung beschrieben.

9.1 Die Struktur einer Diskette/Festplatte

Um verschiedene Speichermedien zu verwalten, organisiert DOS alle diese Medien nach dem gleichen Schema. Die physikalische Aufteilung in Spuren und Sektoren wurde bereits erwähnt. Gemäß dieser Aufteilung werden bestimmte Sektoren für verschiedene Aufgaben fest reserviert. Das folgende Bild gibt die Grobstruktur dieser Anordnung wieder:

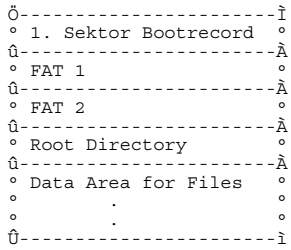


Bild 9.1: Strukturierung der Bereiche auf einem Speichermedium

Aus dieser Struktur ist erkennbar, daß die Anfangssektoren eines jeden Speichermediums für die Verwaltung der Diskette reserviert sind. Bevor wir uns jedoch näher mit den einzelnen Bereichen der Diskette beschäftigen, soll noch die Numerierung der einzelnen Sektoren besprochen werden.

9.2 Die Numerierung der Sektoren

In DOS gibt es verschiedene Verfahren, die einzelnen Sektoren zu numerieren. Das erste Verfahren numeriert alle Sektoren einer Diskette fortlaufend durch. Diese Numerierung der logischen Sektoren ist abhängig von der DOS-Version.

9.2.1 Die logische Sektornumerierung

Als MS-DOS entwickelt wurde, gab es im wesentlichen nur einseitig beschreibbare 5,25-Zoll-Disketten mit 40 Spuren und je 8 Sektoren pro Spur. Mit dieser Konstellation sind insgesamt 320 Sektoren möglich. DOS 1.x numeriert daher diese Sektoren in aufsteigender Reihenfolge durch. Während dieser Entwicklung kamen Laufwerke mit 2 Schreib-/ Leseköpfen auf den Markt. Damit läßt sich die Vorder- und Rückseite einer Diskette benutzen. Die Rückseite ist einfach als Fortsetzung der 1. Seite mit 40 Spuren à 8 Sektoren zu sehen. DOS 1.x setzt daher auf der 2. Seite die Numerierung, beginnend mit dem Wert 320 für den ersten Sektor der ersten Spur, fort. Somit besitzt die erste Seite die logischen Sektoren 0-319, während der folgenden Seite die Sektoren 320-639 zugeordnet werden. So einleuchtend dieses Vorgehen auch sein mag, in der Praxis besitzt es erhebliche Nachteile. Die beiden Schreib-/Leseköpfe eines doppelseitigen Laufwerkes sind mechanisch fest miteinander verbunden, so daß sie immer auf den zwei gegenüberliegenden Spuren positioniert werden. Falls sich nun die Daten einer Datei auf beide Diskettenseiten verteilen, bedeutet dies eine erhebliche Anzahl von Kopfpositionierungen über die gesamte Diskette hinweg (siehe Bild 9.2).

Dies bedingt natürlich recht lange Zugriffszeiten, um bestimmte Dateien zu lesen. Da ab DOS 2.x auch Festplatten unterstützt werden, mußten sich die Entwickler etwas einfallen lassen.

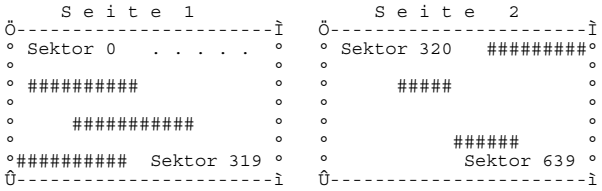


Bild 9.2: Anordnung der logischen Sektoren in DOS 1.x

Ab DOS 2.0 erfolgt deshalb die Numerierung der logischen Sektoren nach anderen Kriterien. Ziel ist es, die Schreib-/Leseköpfe möglichst wenig zu bewegen. Da ab DOS weiterhin Disketten mit 40 Spuren à 9 Sektoren und 80 Spuren à 9 Sektoren unterstützt werden, erfolgt die Numerierung spurweise. In Spur 1 auf der Vorderseite wird der erste Sektor mit dem Wert 0 belegt. Daran schließen sich die logischen Sektoren 1ü8 an, womit diese Spur aufgeteilt ist. Anstatt nun die zweite Spur auf dieser Diskettenseite zu numerieren, wird erst die erste Spur der Rückseite numeriert. Dieser Spur werden die logischen Sektoren 9-17 zugewiesen. Erst dann kommt Spur 2 auf der ersten Diskettenseite an die Reihe. Damit läßt sich die Kopfbewegung beim Datentransfer zu logisch aufeinanderfolgenden Sektoren minimieren. Dies ist insbesondere bei Festplatten wichtig, da hier die Zugriffsgeschwindigkeit signifikant durch die Kopfbewegungszeiten beeinflußt wird. Hier wird das Wort Spur allerdings nicht mehr verwendet, sondern man benutzt die Bezeichnung Zylinder. Bei einer Diskette ist eine Spur ja ein konzentrischer Kreis auf der Oberfläche des Mediums. Auf der Rückseite befindet sich ebenfalls eine Spur mit gleichem Durchmesser, welcher die logisch folgenden Sektoren zugeordnet werden. Besteht die Festplatte nun aus mehreren Oberflächen, die übereinander montiert sind, läßt sich pro Oberfläche ein Schreib-/Lesekopf installieren. Auch hier finden sich die Spuren auf den einzelnen Oberflächen. Betrachtet man nun alle Spuren gleichen Durchmessers als eine Einheit, können diese ohne Kopfbewegung gelesen oder beschrieben werden. Diese Zusammenfassung mehrerer Spuren wird bei Festplatten als Zylinder bezeichnet. Bei einer Platte mit 4 Köpfen besteht dann ein Zylinder aus 4 Spuren. DOS vergibt nun die Sektornummern in aufsteigender Form innerhalb des jeweiligen Zylinders. Erst dann wird der nächste Zylinder numeriert. Bei einer Platte mit 4 Köpfen ergibt sich dann zum Beispiel im Zylinder 1 (17 Sektoren/Spur) folgende Zuordnung:

Kopf	Sektoren
0	0-16
1	17-33
2	34-50
3	51-67

Bild 9.3: Anordnung der Sektoren pro Zylinder

Der zweite Zylinder beginnt dann beim ersten Kopf mit der logischen Sektornummer 68.

Daß es wegen der Aufteilung der Datenbereiche trotzdem zu Problemen kommen kann, wird auf den folgenden Seiten angesprochen.

9.2.2 Die relative Sektornumerierung

Nun soll aber noch die zweite Art der Numerierung der Einzelsektoren besprochen werden. Bei dieser wird erst ab dem 6. Sektor mit dem Zählen begonnen. Die ersten 5 Sektoren sind nicht numeriert, d.h. alle Angaben bezüglich einer Sektornummer beziehen sich relativ zum Sektor 6. Diese Bezeichnungsweise wurde im wesentlichen bei den DOS-1.x-Versionen verwendet.

9.3 Die Boot-Partition einer Festplatte

Im Gegensatz zu Disketten besitzen Festplatten die Möglichkeit, den gesamten Speicherbereich in 4 verschiedene Teile (Partitionen) aufzugliedern. Damit lassen sich dann z.B. mehrere Betriebssysteme auf der Platte halten. Eines dieser Betriebssysteme wird dann als »bootfähigestte enthält.

- Zahl der Partitionen (1-4)
- Größe der jeweiligen Partitionen

Weiterhin ist dort auch eingetragen, welche Partition aktiviert ist. Der *Master Boot Record* wird im ersten physikalischen Sektor der Platte abgelegt.

Die einzelnen Betriebssysteme erkennen anschließend nur die jeweils aktive Partition, d.h. für sie gibt es keinen Unterschied zwischen einer komplett verfügbaren Platte und einem solchen Teilbereich. Es ist allerdings möglich, über die direkte Adressierung einzelner physikalischer Plattensektoren auch auf andere Partitionen zuzugreifen. Dies sollte aber unter allen Umständen vermieden werden, da die Datenkonsistenz so nicht garantiert werden kann.

Da DOS die jeweils aktive Partition wie eine normale Platte verwendet, werden die pysikalischen Sektoren innerhalb dieses Bereiches mit einer logischen Numerierung versehen. Alle I/O-Zugriffe über das DOS-Dateisystem beziehen sich dann auf diese logischen Sektornummern. Daher ist es auch nicht weiter verwunderlich, wenn der erste logische Sektor einer Partition einen *Boot Record* enthält. Das DOS-Programm FORMAT bezieht sich ja auf eine einzelne Partition und legt mit dem /S-Schalter einen Boot Record sowie eine Kopie der DOS-Systemdateien (IO.SYS und MSDOS.SYS) ab.

Mit Hilfe des FDISK-Programms kann eine solche Partition als aktive bootbare Sektion selektiert werden. Beim Systemstart wird zuerst das Master-Boot-Programm aktiviert, welches dann den Boot Record der selektierten Partition anspricht.

Nun soll der Aufbau des Master-Boot-Records kurz besprochen werden. Wie breits erwähnt, findet sich dieser Record im ersten Sektor einer Festplatte oder eines logischen Laufwerkes (Festplatten über 32 Mbyte werden unter DOS meist in mehrere logische Laufwerke aufgeteilt). Zu Beginn findet sich ein kurzes Programmstück, welches beim Systemstart durch den BIOS-ROM-Lader im Hauptspeicher (per Adresse 0:700) installiert und gestartet wird. Dieses kurze Programm besitzt nun die Aufgabe, die Partitionstabelle im Boot Record auf eine aktive Partition zu untersuchen und von dieser dann das Boot-Programm zu installieren. Die Partitionstabelle beginnt ab Offset 1BEH im Master-Bootsektor und besitzt folgenden Aufbau:

Offset	Feld
1BEH	Partition 1 Boot Indikator
1BFH	Partition 1 Kopfnummer des 1. Sektors
1C0H	Partition 1 erster Sektor
1C1H	Partition 1 erster Zylinder
1C2H	Partition 1 System Indikator
1C3H	Partition 1 Kopfnummer des letzten Sektors
1C4H	Partition 1 letzter Sektor
1C5H	Partition 1 letzter Zylinder
1C6H	Partition 1 relativer Sektor Low Word
1C8H	Partition 1 relativer Sektor High Word
1CAH	Partition 1 Sektorzahl Low Word
1CCH	Partition 1 Sektorzahl High Word
1CEH	Partition 2 Boot Indikator
1CFH	Partition 2 Kopfnummer des 1. Sektors
1D0H	Partition 2 erster Sektor
1D1H	Partition 2 erster Zylinder
1D2H	Partition 2 System Indikator
1D3H	Partition 2 Kopfnummer des letzten Sektors
1D4H	Partition 2 letzter Sektor
1D5H	Partition 2 letzter Zylinder
1D6H	Partition 2 relativer Sektor Low Word

Tabelle 9.1: Aufbau der Partitionstabelle einer Festplatte

Offset	Feld
1D8H	Partition 2 relativer Sektor High Word
1DAH	Partition 2 Sektorzahl Low Word
1DEH	Partition 3 Boot Indikator
1DFH	Partition 3 Kopfnummer des 1. Sektors
1E0H	Partition 3 erster Sektor
1E1H	Partition 3 erster Zylinder
1E2H	Partition 3 System Indikator
1E3H	Partition 3 Kopfnummer des letzten Sektors
1E4H	Partition 3 letzter Sektor
1E5H	Partition 3 letzter Zylinder
1E6H	Partition 3 relativer Sektor Low Word
1E8H	Partition 3 relativer Sektor High Word
1EAH	Partition 3 Sektorzahl Low Word
1ECH	Partition 3 Sektorzahl High Word
1ECH	Partition 3 Sektorzahl High Word
1EEH	Partition 4 Boot Indikator
1EFH	Partition 4 Kopfnummer des 1. Sektors
1F0H	Partition 4 erster Sektor
1F1H	Partition 4 erster Zylinder
1F2H	Partition 4 System Indikator
1F3H	Partition 4 Kopfnummer des letzten Sektors
1F4H	Partition 4 letzter Sektor
1F5H	Partition 4 letzter Zylinder
1F6H	Partition 4 relativer Sektor Low Word
1F8H	Partition 4 relativer Sektor High Word
1FAH	Partition 4 Sektorzahl Low Word
1FCH	Partition 4 Sektorzahl High Word
1FEH	Signatur Word

Tabelle 9.1: Aufbau der Partitiontabelle einer Festplatte (Ende)

Die Tabelle besitzt 4mal die gleiche Datenstruktur (1mal für jede Partition)

Boot Indicator

Im ersten Byte dieser Partitiontabelle findet sich ein Feld mit der Boot-Markierung. Anhand dieser arkierung kann das Boot-Programm erkennen, welche der 4 Partitionen aktiv ist. Das Feld enthält dann den Wert 80H, der durch das DOS-Programm FDISK eingetragen wurd e. Die anderen 3 Felder (Offset 1CEH, 1DEH, 1EEH) besitzen den Wert 00H, da sie nicht gleichzeitig bootbar sind.

Kopf

Das zweite Feld innerhalb der Datenstruktur (Partitiontabelle) enthält jeweils die Nummer (1 Byte) des Schreib-/Lesekopfes (Head), mit dem die Partition beginnt. In der ersten Partition befindet sich der logische Sektor 1 auf der zweiten Oberfläche (Kopf 1). Die Spur unter Kopf 0 dient ja zur Aufnahme des Master-Boot-Records und ist deshalb reserviert. Bei den drei anderen Partitionen kann auch die erste Spur (Kopf 0) belegt werden.

Zylinder und Sektor

Die beiden Feld er Zylinder und Sektor werden zur Betrachtung zusammengefaßt. Hierbei gilt folgende Str uktur:

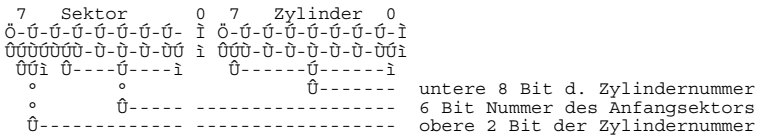


Bild 9.4: Kodieru ng der Zylinder- und Sektornummer im »Master-Boot-Record«

Die Felder sind in dieser merkwürdigen Struktur angeordnet, um die Daten leicht für den BIOS-INT 13 (Dis k I/O) laden zu können. Die erste Partition beginnt bei Sektor 1 des ersten Zylinders, allerdings erst auf Spur 1, d.h. der Kopf 1 bearbeitet die Daten des ersten Sektors. Dies i st erforderlich, da in Spur 0 (Kopf 0) dieses Zylinders der Master-Boot-Record gespeich ert ist. Weiterhin finden sich hier Informationen bezüglich der Konfiguration der Festplatte. Die drei restlichen Partitionen beginnen bei einem Zylinder auf Spur 0 (Ko pf 0) und Sektor 1, da ja hier kein Platz für den Master-Boot-Record**Fehler! Verweisquelle konnte nicht gefunden werden.**Die *-Zeichen s ignalisieren die Bootbereiche, während mit # die Sektoren einer Spur markiert werden. Bei einer Festplatte mit 17 Sektoren pro Spur und insgesamt 4 Köpfen (4 Spuren/Zylinde r) enthält ein Zylinder dann 68 Sektoren. Die einzelne Partition belegt dann mehrere aufeinanderfolgende Zylinder. Dieser Bereich wird von DOS wie eine komplette Platte betrachtet, mit Boot Record, FATs etc.

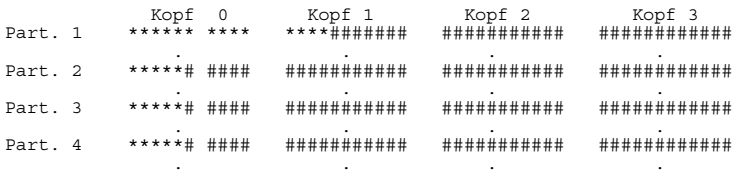


Bild 9.5: Aufteilung einer Festplatte

System Indicat or

Ab Offset 1X2H wird das Ende der Partition markiert. Das Feld enthält eine Markierung für das Betrieb ssystem, welches diesen Bereich benutzt. Anhand der Kodierung ergeben sich verschiedene Hinweise hinsichtlich dieser Partition:

Code	Bedeutu ng
00	unbekannter Typ
01	primärer DOS-Bereich mit 12 Bit FAT
02	---
03	---
04	primärer DOS-Bereich mit 16 Bit FAT
05	extended DOS-Partition

Tabelle 9.2: Kodierung des System-Indikatorfeldes

Beim Wert 05 handelt es sich um eine DOS-Version (3.3), die erweiterte Partitionen benutzt. Deren Aufbau wird auf den folgenden Seiten besprochen. Die Codes 01 und 04 geben an, wie die FATs aufgebaut sind (siehe folgende Seiten).

In den nächsten 3 Byte finden sich wieder Einträge für die Kopfnummer, den Sektor und den Zylinder. Diese Einträge spezifizieren das Ende der Partition innerhalb der Platte. Die Kodierung erfolgt analog der oben beschriebenen Form (Spezifizierung des Partitionsanfangs).

Partition-Relativ-Sektor

Ab Offset 1X6H findet sich ein 4-Byte-Vektor (Low Word first), in dem die relative Sektornummer des ersten Sektors der Partition gespeichert ist, d.h. die Zahl der Sektoren der vorhergehenden Zylinder wird hier abgelegt. Der Zähler beginnt bei Zylinder 0, Kopf 0, Sektor 1. Bei 19 Sektoren pro Spur und 4 Köpfen ergibt sich z.B. bei Zylinder 10 ein relativer Wert von (17 * 4 * 10) = 680. Falls die zweite Partition bei Zylinder 10 beginnt, wird dieser Wert dann als *Relativ-Sektor* eingetragen.

Sektorenzahl

Ab Offset 1XAH findet sich ein weiterer 4-Byte-Vektor, in dem die Zahl der Sektoren der jeweiligen Partition gespeichert wird (Low Word first).

Diese Datenstruktur findet sich insgesamt 4mal in der Partitionstabelle. Die Tabelle wird mit zwei Signatur-Bytes (Offset 1FEH) abgeschlossen. Diese markiert, ob die Partition gebootet werden kann. In diesem Fall müssen sowohl der Master-Boot-Record als auch der *Boot Record* der Partition die Signatur 55AAH besitzen. Weiterhin muß diese Partition im Boot-Feld (Offset 1XEH) mit dem Wert 80H markiert sein. Falls beim Systemstart diese Signaturen nicht gefunden werden, bricht das System den Start ab. Beim IBM-PC wird dann das ROM-Basic aktiviert.

Findet das System die korrekten Daten, wird die Partitionstabelle in den Speicher geladen. Die Anfangsadresse der Tabelle findet sich während der Abarbeitung des Boot-Programms im DS:SI-Register des Prozessors.

9.4 Die Extended-DOS-Partition

Mit den bisher besprochenen Konzepten ordnet DOS einer logischen Einheit jeweils nur eine Platte (oder Partitionen) zu. Die maximale Größe pro Einheit ist (bis DOS 3.3) auf 32 Mbyte beschränkt. Dies hängt im wesentlichen mit der Größe der FAT zusammen.

Nachdem mittlerweile größere Platten verfügbar sind, kann es erforderlich werden, mehrere Platten oder Partitionen einer physikalischen Einheit zusammenzufassen. Um mit zukünftigen DOS-Versionen diese technischen Entwicklungen besser unterstützen zu können, existiert ab DOS 3.3 der neue Typ der *Extended DOS Partition*.

Bei der Extended-DOS-Partition handelt es sich um eine Zusammenfassung mehrerer *Extended Volumes* zu einer logischen Einheit. In der Partitionstabelle des jeweiligen Boot-Records wird eine Extended-DOS-Partition mit dem Eintrag 05H im *System Indicator Field* markiert. Der entsprechende Record ist dann nicht bootbar. Falls dies doch versucht wird, terminiert das Programm mit einer entsprechenden Fehlermeldung. Programme wie FDISK dürfen deshalb den Master-Boot-Record**Fehler! Verweisquelle konnte nicht gefunden werden.**Bei dem *Extended Volume* handelt es sich schlicht um einen Speicherbereich auf einer Platte. Dieser darf minimal einen Zylinder bis maximal 32 Mbyte umfassen. Der Bereich muß weiterhin an den Zylindergrenzen beginnen und enden.

Das Extended-Volume besteht dabei einmal aus dem *Extended Boot Record* sowie der jeweiligen Platte/Partition. Bei DOS-Einheiten schließt sich an den Extended-Boot-Record ein normaler DOS-Bootsektor an (Systemindikator 01H, 04H).

Die einzelnen Extended-Volumes werden dann über Zeiger innerhalb des Extended-Boot-Records zu der Extended-DOS-Partition zusammengefaßt. Es ergibt sich also folgendes Bild:

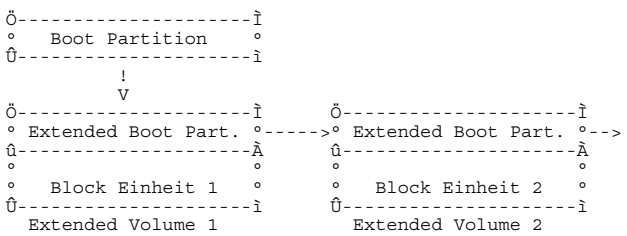


Bild 9.6: Der Aufbau einer Extended-DOS-Partition

Eine Extended-DOS-Partition ist zusammen mit einer *Primary Boot Partition* anzulegen. Hierbei handelt es sich um eine Partition, in deren Boot-Tabelle die Systemindikatoren 01H oder 04H eingetragen sind. Weiterhin muß die Partition auf einem bootfähigen Laufwerk liegen. Nur falls das Laufwerk nicht bootbar ist, kann auf die Primary-Boot-Partition verzichtet werden.

An diesen *Extended Boot Record* des Extended-Volumes schließt sich der normale DOS-Boot-Record mit dem Systemindikator 01H oder 04H an. Dieser Record beginnt allerdings auf einer neuen Spur innerhalb des gleichen Zylinders. Falls der Extended-Boot-Record auf Spur 0 liegt, beginnt der *DOS Boot Record* des Extended-Volume bei Spur 1.

Der Extended-Boot-Record findet sich jeweils im ersten Sektor des physikalischen Laufwerkes. Im Signatur Word (letztes Feld der Partitionstabelle) findet sich der Wert 55AAH. Damit wird die Kompatibilität zu den normalen Boot-Records sichergestellt.

Der Aufbau des *Extended Boot Record* entspricht im wesentlichen dem *Master Boot Record*. Der Hauptunterschied besteht darin, daß die Partitionstabelle durch eine Laufwerkstabelle ersetzt wurde. Hier lassen sich insgesamt 2 Laufwerke eintragen. Weiterhin ist der Wert 05H im Feld *System-Indikator* eingetragen, wobei er allerdings das nächste Laufwerk signalisiert. Die folgende Tabelle enthält die Belegung dieser Laufwerkstabelle und besteht aus 4 Bereichen à 16 Byte:

Offset	Feld	
1BEH	Partition 1	Boot Indikator
1BFH	Partition 1	Kopfnummer des 1. Sektors Drive
1C0H	Partition 1	erster Sektor/Drives
1C1H	Partition 1	erster Zylinder/Drives
1C2H	Partition 1	System Indikator

Tabelle 9.3: Aufbau der Extended-Partitionstabelle einer Festplatte

Offset	Feld
1C3H	Partition 1 Kopfnummer letzter Sektor/Drive
1C4H	Partition 1 letzter Sektor/Drive
1C5H	Partition 1 letzter Zylinder/Drive
1C6H	Partition 1 relativer Sektor Low Word
1C8H	Partition 1 relativer Sektor High Word
1CAH	Partition 1 Sektorzahl Low Word
1CCH	Partition 1 Sektorzahl High Word
1CEH	Partition 2 Boot Indikator
1CFH	Partition 2 Kopfnummer des 1. Sektors
1D0H	Partition 2 erster Sektor
1D1H	Partition 2 erster Zylinder
1D2H	Partition 2 System Indikator
1D3H	Partition 2 Kopfnummer des letzten Sektors
1D4H	Partition 2 letzter Sektor
1D5H	Partition 2 letzter Zylinder
1D6H	Partition 2 relativer Sektor Low Word
1D8H	Partition 2 relativer Sektor High Word
1DAH	Partition 2 Sektorzahl Low Word
1DEH	Partition 3 Boot Indikator
1DFH	Partition 3 Kopfnummer des 1. Sektors
1E0H	Partition 3 erster Sektor
1E1H	Partition 3 erster Zylinder
1E2H	Partition 3 System Indikator
1E3H	Partition 3 Kopfnummer des letzten Sektors
1E4H	Partition 3 letzter Sektor
1E5H	Partition 3 letzter Zylinder
1E6H	Partition 3 relativer Sektor Low Word
1E8H	Partition 3 relativer Sektor High Word
1EAH	Partition 3 Sektorzahl Low Word
1ECH	Partition 3 Sektorzahl High Word

Tabelle 9.3: Aufbau der Extended-Partition-Tabelle einer Festplatte (Fortsetzung)

Offset	Feld
1ECH	Partition 3 Sektorzahl High Word
1EEH	Partition 4 Boot Indikator
1EFH	Partition 4 Kopfnummer des 1. Sektors
1F0H	Partition 4 erster Sektor
1F1H	Partition 4 erster Zylinder
1F2H	Partition 4 System Indikator
1F3H	Partition 4 Kopfnummer des letzten Sektors
1F4H	Partition 4 letzter Sektor
1F5H	Partition 4 letzter Zylinder
1F6H	Partition 4 relativer Sektor Low Word
1F8H	Partition 4 relativer Sektor High Word
1FAH	Partition 4 Sektorzahl Low Word
1FCH	Partition 4 Sektorzahl High Word
1FEH	Signatur Word

Tabelle 9.3: Aufbau der Extended-Partitionstabelle einer Festplatte (Ende)

Die jeweils ersten 4 Byte einer Teiltabelle sollten aus Kompatibilitätsgründen initialisiert werden. Das erste Byte enthält dabei jeweils den System-Indikator mit folgender Bedeutung:

Code	Bedeutung
00	kein Speicher belegt
01	DOS-Partition mit 12 Bit FAT
02	---
03	---
04	DOS-Partition mit 16 Bit FAT
05	zeigt ein weiteres Laufwerk an
06	reserviert (DOS 4.0 Extended Partition > 32 Mbyte)

Alle 4 Felder müssen mit einem dieser Werte besetzt werden. Falls eine 0 auftritt, sind alle 16 Byte zu löschen.

Start und End

Die Belegung der Felder Zylinder, Head und Sektor hängt von der Markierung des ersten Bytes ab. Wird als System-Indikator der Wert 01H oder 04H eingetragen, beschreibt die Tabelle die logische Einheit des Extended-Volume. Dann geben die Felder die Grenzen des Plattenbereiches, relativ zum Boot Record an.

Wird der System-Indikator auf den Wert 05H gesetzt, enthalten die Felder den Zeiger auf das nächste Extended-Volume. Die Angaben beziehen sich dann relativ auf den Anfang der Platte.

Im Extended-Boot-Record sind deshalb immer nur diese 2 Einträge vorhanden. Welche der 4 Teiltabellen belegt werden, ist frei wählbar.

Damit soll die Beschreibung des Aufbaus der Partition einer Platte abgeschlossen werden. Weitere Informationen sind den Unterlagen der Hersteller zu entnehmen.

9.5 Der Boot-Record einer Einheit

Unabhängig vom Speichermedium (Diskette oder Platte) findet sich im ersten logischen Sektor der DOS-Boot-Record. Er wird vom Programm FORMAT abgelegt und enthält unter anderem Informationen über den logischen Aufbau des Mediums. Die nachfolgende Tabelle gibt den Anfang dieses Boot Records wieder.

Offset	Bytes	Bedeutung
00	3	In DOS 2.x finden sich hier 3 Byte (0E0H, xx, xx) mit einem Near JUMP. Ab DOS 3.X findet sich hier ein Short JUMP (0EBH, XX) und ein NOP (90H)
03	8	ASCII-Text mit dem OEM-Namen und der Version
0B	2	Bytes pro Sektor
0D	1	Sektoren pro Zuordnungseinheit als 2er Potenz
0E	2	Zahl der reservierten Sektoren (ab log. Sektor 0)
10	1	Zahl der File-Allocation-Tables
11	2	max. Zahl von Einträgen im Hauptinhaltsverzeichnis
13	2	Gesamtzahl der logischen Sektoren
15	1	Media Descriptor Byte
16	2	belegte Sektoren pro FAT
18	2	Sektoren pro Spur
1A	2	Zahl der Köpfe
1C	2	Zahl der "hidden" Sektoren

Tabelle 9.4: Aufbau des DOS Boot Records

Die Daten werden vom Treiber ausgewertet und dienen zum Beispiel zum Aufbau des BIOS-Parameter-Blocks (BPB). Dieser Sektor findet sich bei Disketten auf Spur 0, Kopf 0, Sektor 1. Bei Platten findet er sich zu Beginn einer Partition.

Sobald dieses Boot-Programm geladen und gestartet wurde, sucht es die Programme IO.SYS und MSDOS.SYS auf dem Medium. Diese müssen in einer bestimmten Reihenfolge an bestimmten Stellen auf dem Speichermedium abgelegt sein. Ist dies nicht der Fall, wird eine Fehlermeldung ausgegeben und der Start abgebrochen. Lediglich beim IBM-PC wird in diesem Fall das ROM-Basic aktiviert. Sind die DOS-Dateien vorhanden, werden diese in den Speicher geladen und das Programm IO.SYS wird gestartet. Dieses installiert dann das MS-DOS-Betriebssystem.

Ab DOS 4.0 müssen Blocktreiber Festplatten größer als 32 Mbyte unterstützen. Deshalb definiert DOS ab der Version 4.0, in Abhängigkeit von der Größe des Mediums, eine erweiterte Struktur für den Bootblock und den zugehörigen BIOS-Parameter-Block (BPB).

Bei Medien kleiner 32 Mbyte kann die bisherige Struktur beibehalten werden. Andernfalls gilt folgende Belegung des Bootrecords:

Bytes	Feld (Bootrecord ab DOS 4.0)
2	2 Byte SHORT JMP (EBH) + NOP (90H)
8	ASCII-Text mit OEM-Namen und Version
25	extended BPB
1	physikalische Laufwerksnummer
1	reserviert
1	extended Bootrecord Signatur
4	Seriennummer des Mediums
11	Label des Speichermediums (ASCII)
8	reserviert

Die physikalische Laufwerksnummer im Bootblock ist in DOS 4.0 immer 00H oder 80H. Die extended Bootrecord Signatur ist immer 29H. Für die 25 Byte des extended BPB gilt dann folgende Kodierung:

Bytes	Feld (extended BPB in DOS 4.0)
2	Bytes pro Sektor
1	Sektoren pro Cluster
2	Zahl der reservierten Sektoren
1	Zahl der FAT's (0 falls kein FAT vorhand.)
2	Max. Zahl d. Einträge im Hauptverzeichnis
2	Zahl der Sektoren des Mediums
	ist der Wert 0 -> extended BPB

Bytes	Feld (extended BPB in DOS 4.0)
1	Media Descriptor Byte
2	belegte Sektoren pro FAT
2	Sektoren pro Spur
2	Zahl der Köpfe
4	Zahl der »hidden**Fehler! Verweisquelle konnte nicht gefunden werden.**
4	Erweiterung für Platten > 32 Mbyte
	Gesamtzahl der logischen Sektoren
	als 32-Bit-Zahl

Die Struktur des erweiterten BPB entspricht im wesentlichen der bisherigen Struktur. Lediglich am Ende wird ein 4-Byte-Bereich mit der Zahl der logischen Sektoren (32 Bit) angehängt. Weiterhin muß im Feld mit der Zahl der Sektoren (Offset 8) der Wert 0 stehen. Der extended BPB ist immer dann zu benutzen, falls die Zahl der Sektoren nicht mit 16 Bit darstellbar ist. In allen anderen Fällen ist die Sektorenzahl ab Offset 8 abzulegen.

9.6 Die File-Allocation-Tabelle (FAT)

An den Boot Sektor schließt sich ein Bereich an, den DOS zur Verwaltung des Mediums nutzt. In diesem Bereich findet sich eine Tabelle (FAT), die die Zuordnung der einzelnen Sektoren zu den Dateien ermöglicht. Bevor wir zur Beschreibung des Aufbaus kommen, soll noch kurz auf die Bedeutung dieser Tabelle eingegangen werden. Die Einteilung einer Diskette oder Platte in Spuren, Sektoren oder Zylinder wurde bereits besprochen. Es ist nun denkbar, daß DOS die Dateien jeweils zusammenhängend sequentiell auf die Platte oder Diskette ablegt.

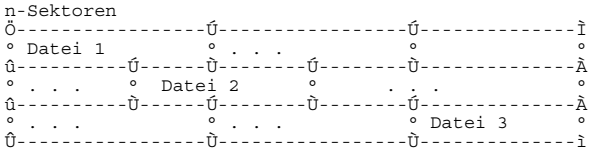


Bild 9.7: Verteilung der Dateien zu Beginn der Speicherung

Damit wird eigentlich nur der Anfangssektor der Datei und deren Länge gebraucht, um diese zu bearbeiten. Leider hat diese Verwaltung einen gravierenden Nachteil. Wird die Datei beim Bearbeiten kürzer, bleiben zwischen dem neuen Dateiende und dem Beginn der folgenden Datei einige Sektoren frei. Wird die Datei gelöscht, werden alle Sektoren freigegeben. Dies wäre ja noch akzeptierbar. Aber falls eine bestehende Datei vergrößert werden soll, muß sie in einen freien Bereich (z.B. hinter der letzten Datei) auf der Diskette/Platte umkopiert werden, der die erforderliche Zahl an freien Sektoren besitzt. Die alten Sektoren werden freigegeben. In der Praxis ist dieses Verfahren nicht anwendbar, da sehr schnell die Situation auftritt, daß z.B. eine Datei von 10 Sektoren Länge um einen weiteren Sektor zu verlängern ist. Auf dem Medium seien noch 7 Sektoren frei. Da die Datei aber einen zusammenhängenden Bereich von 11 Sektoren benötigt, kann die Erweiterung nicht erfolgen. Aus Sicht der Speicherbelegung ist dies ein sehr unökonomisches Verfahren. Weiterhin kostet das Umkopieren der Daten ebenfalls viel Zeit. Dies führt dazu, daß in der Praxis andere Verwaltungsverfahren eingesetzt werden. DOS benutzt ein solches Verfahren zur Verwaltung der Daten. Das Prinzip beruht darauf, daß eine Datei nicht mehr zwangsweise in aufeinanderfolgende Sektoren gespeichert werden muß, vielmehr kann sie in Teilen über den gesamten Speicherbereich verstreut werden.

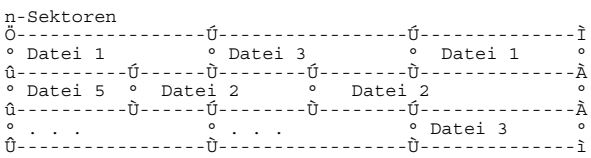


Bild 9.8: Verteilung der Dateien auf eine Diskette nach einigen Schreiboperationen

Auf einer neu formatierten Diskette/Platte werden die Dateien sicher zu Beginn in aufeinanderfolgenden Sektoren gespeichert. Sobald aber eine Datei in ihrer Länge verändert wird, tritt oben aufgezeigte Fraktionierung auf. Zur Verwaltung dieser Dateien muß DOS dann alle von einer Datei belegten Bereiche speichern. Am einfachsten ist es, die

Platte in logische Bereiche (z.B. Sektoren) aufzuteilen. Dann merkt sich DOS den Anfangssektor der Datei im Inhaltsverzeichnis (Directory) der Platte/Diskette. Weiterhin wird eine Tabelle (FAT) auf dem Medium angelegt, welche einen Eintrag für jeden Bereich besitzt, der die folgenden Sektoren der Datei identifiziert.

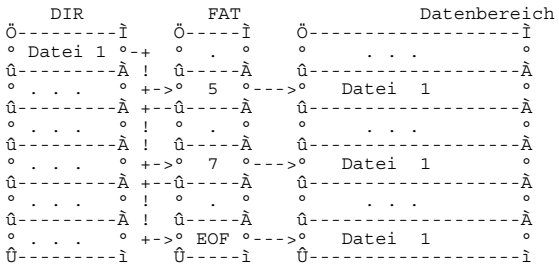


Bild 9.9: Verwaltung der belegten Sektoren über FAT`s

In obigem Beispiel ist die Datei 1 in den Sektoren 3, 5, 7 und 9 gespeichert. Im Inhaltsverzeichnis findet sich neben dem Namen »Datei 1lgenden Sektoren der Datei zu identifizieren. Hierzu wird die Nummer des Folgesektors (hier 5) in der FAT eingetragen, d.h. der nächste von der Datei belegte Sektor ist der Sektor 5. Gleichzeitig findet sich im 5. Eintrag der FAT wieder ein Zeiger auf den nächsten belegten Sektor. Diese Kette wird solange fortgeführt, bis die Datei komplett gespeichert ist. Falls kein Folgesektor mehr existiert, wird der entsprechende Eintrag innerhalb der FAT mit einer Endekennung versehen. Freie Sektoren innerhalb der FAT werden ebenfalls markiert. So kann DOS relativ einfach seine Dateien auf der Platte verwalten, ohne größere Umlagerungen vornehmen zu müssen. Der Speicherplatz wird ebenfalls sehr gut genutzt.

Ein Nachteil soll aber nicht verschwiegen werden. Während des normalen Betriebes werden Dateien immer wieder gelöscht, neu angelegt oder in ihrer Größe verändert. DOS arbeitet nun aber nach der Strategie, die Dateien möglichst in den unteren Sektoren abzulegen, um am Ende des Mediums möglichst freien Speicherplatz zu halten. Dies führt dazu, daß eine Datei in viele Teile zergliedert und auf der Platte/Diskette verstreut gespeichert wird. Soll diese Datei nun gelesen oder beschrieben werden, muß der jeweilige Schreib-/ Lesekopf sehr viel bewegt werden. Obwohl die Entwickler bei der Numerierung der logischen Sektoren die Zahl der Kopfbewegungen minimierten (siehe Abschnitt »Die Numerierung der Sektoren«, bewirkt die .i.Fraktionierung der Dateien; eine deutliche Verschlechterung der Zugriffsgeschwindigkeit. Dies macht sich insbesondere bei größeren Festplatten erheblich bemerkbar. Mittlerweile gibt es Programme, die die Diskette/Platte neu organisieren»und die Dateien in aufeinanderfolgenden Sektoren speichern. Wer diese Programme nicht besitzt, kann eine andere Optimierungsmöglichkeit benutzen. Hierzu sind z.B. alle Dateien einer Platte auf andere Medien auszulagern (per COPY-Befehl). Nachdem die Platte komplett gelöscht wurde, können die Daten zurückkopiert werden. DOS legt die Dateien dann in aufeinanderfolgenden Sektoren ab. Bei beiden Optimierungen ist aber darauf zu achten, daß kopiergeschützte Software vorher deinstalliert wird, da eventuell die Sicherungsmarkierung auf der Platte gelöscht oder überschrieben wird (z.B. beim Formatieren).

Ein weiteres Problem soll auch nicht verschwiegen werden. Bisher sind wir davon ausgegangen, daß DOS die Dateien auf einzelne Sektoren aufteilt. Damit muß für jeden

Sektor ein Eintrag in einer File-Allocation-Tabelle vorgesehen werden. Bei einer Platte mit sehr vielen Sektoren ist dies sehr unökonomisch, da einmal diese Tabelle sehr groß wird. Andererseits sind sehr viele Kopfpositionierungen erforderlich, um die Einzelsektoren einer fraktionierten Diskette zu lesen. Weiterhin besteht das Problem, daß mit einer 12-Bit-FAT nur 4095 Sektoren verwaltet werden können. Eine 16-Bit-FAT dehnt den Bereich zwar auf 65 536 Sektoren aus, womit der maximale Speicherplatz (bei 512 Byte pro Sektor) auf 32 Mbyte begrenzt wird.

Falls aber nur 256 Byte pro Sektor gespeichert sind, können diese 32 Mbyte nicht mehr erreicht werden. Es müßte also eine längere FAT-Kodierung verwendet werden. DOS geht hier einen anderen Weg, indem es mehrere Sektoren zu einem logischen Bereich (Cluster) zusammenfaßt. Die kleinste benutzte Einheit ist damit nicht mehr ein Sektor, sondern ein Cluster. Die Länge einer Datei beträgt daher immer ein ganzzahliges Vielfaches der Clustergröße. Wie viele Sektoren ein Cluster umfaßt, wird beim Formatieren des Mediums festgelegt. Bei einseitigen Disketten existiert zwar meist noch die Zuordnung 1 Cluster = 1 Sektor, aber bei zweiseitigen Disketten umfaßt ein Cluster bereits 2 Sektoren. Festplatten werden meist in Clustern zu je 4 Sektoren aufgeteilt. Diese Information findet sich im Boot Record und wird durch den Treiber der Einheit ausgewertet. Alle Einträge innerhalb einer FAT beziehen sich also auf Cluster und nicht auf Sektoren. Bei 4 Sektoren pro Cluster können also immer 4 aufeinanderfolgende Sektoren gelesen werden, bevor der Kopf neu positioniert wird. Als Nachteil ist aber die Tatsache zu sehen, daß eine Datei der Größe 1 Byte auf der Platte 4 komplette Sektoren belegt, d.h. die Nettokapazität der Platte nimmt bei vielen kleinen Dateien ab. In der Praxis läßt sich aber mit der Einteilung in Cluster gut leben.

DOS teilt also über die File-Allocation-Tabelle einer Datei den entsprechenden Raum in Form von Clustern zu. Dabei wird zwischen 12-Bit- und 16-Bit-FATs unterschieden.

12-Bit-FATs werden bei Disketten und kleineren Festplatten eingesetzt. Sobald mehr als 4085 Cluster (20 740 Sektoren) vorliegen, benutzt DOS die 16-Bit-FATs. Gleichzeitig variiert damit auch die Größe der FAT. Bei 12-Bit-FATs werden 3 Nibbles (Hexzeichen) pro Eintrag gespeichert, während bei 16-Bit-FATs 4 Nibbles vorliegen. Beide Typen besitzen den gleichen Aufbau:

Eintrag	Bedeutung
(0)000	Dieser Cluster ist frei
(F)FFO	Dieser Cluster enthält einen »bad block**Fehler! Verweisquelle konnte nicht gefunden werden.**
.	. wird durch MS-DOS nicht benutzt.
(F)FF7	
(F)FF8	Markierung für den letzten Cluster innerhalb
.	einer Kette. Damit wird das Ende einer Datei
(F)FFF	markiert.

Tabelle 9.5: Einträge in der File-Allocation-Tabelle (FAT)

Die Ziffer in Klammern () wird nur bei der 16-Bit-FAT benutzt. Alle anderen Hexzeichen werden als die Nummer des folgenden Clusters betrachtet. Die Nummer des ersten Clusters dieser Datei findet sich im Inhaltsverzeichnis (Directory). Die beiden FAT-Einträge 000 und 001 sind reserviert. So findet sich im ersten Byte der FAT-ID-Code, der von manchen

DOS-Treibern zur Identifizierung des Mediums und zum Aufbau des BIOS-Parameter-Blocks (BPB) benutzt wird. Die folgende Tabelle enthält eine Aufstellung gebräuchlicher FAT-ID-Bytes.

FAT ID	Medium
FF	Dual sided, 8 Sektoren pro Spur
FE	Single sided,8 Sektoren pro Spur
FD	Dual sided, 9 Sektoren pro Spur
FC	Single sided,9 Sektoren pro Spur
F9	Dual sided, 15 Sektoren pro Spur (5 1/4 Zoll 1,2 Mbyte)
F9	Dual sided, 9 Sektoren pro Spur (3 1/4 Zoll, 720 Kbyte)
F8	Fixed Disk
FO	andere Formate

Tabelle 9.6: Belegung des FAT-ID-Bytes

Hier wird übrigens deutlich, daß das Medium nicht mehr eindeutig durch die FAT-ID beschrieben wird. Die restlichen 2 (3 Byte bei 16-Bit-FAT) enthalten den Wert FFH.

Erst beim dritten Eintrag der Tabelle beginnt die Zuweisung der Dateien mit der Cluster-Nummer 002. Nachfolgendes Bild zeigt die Zuordnung einer Datei über Cluster:

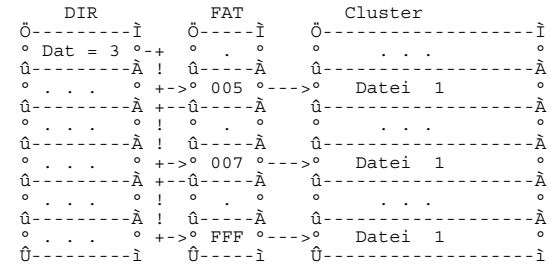


Bild 9.10: Zuordnung der Cluster über die FAT

Die Nummer des Anfangsclusters der Datei wird im Inhaltsverzeichnis gespeichert. Anschließend findet sich eine Kette der belegten Cluster für diese Datei in der FAT, da ja jeder FAT-Eintrag e inen Zeiger zum nächsten Cluster enthält.

Beschädigte Cluste r werden von DOS nicht benutzt und durch den FAT-Eintrag (F)FF0 bis (F)FF7 gesperrt. De r letzte Cluster einer Kette enthält den Wert (F)FF8 bis (F)FFF.

Die File-Allocati on-Tabelle (FAT) belegt einen oder mehrere Sektoren auf dem Speichermedium. Der Anfangssektor der FAT schließt sich an den Bootsektor an.

Sobald nun aber ein Fehler in der File-Allocation-Tabelle auftritt, z.B. die Daten lassen sich nicht mehr l esen, kann bei einer FAT die Datei nicht mehr rekonstruiert werden. Die Struktur von DOS erlaubt deshalb einen Bereich mit mehreren FATs einzurichten. In der Regel werden durc h DOS aber nur zwei FATs pro Einheit verwaltet. Die zweite FAT kann dann benutzt werden , wenn in der ersten FAT ein Fehler auftritt.

DOS liest diese FAT bei jedem modifizierenden I/O-Zugriff (Open, Create etc.) in einen der Puffer im Hauptspeicher. Dieser Puffer wird mit einer hohen Priorität versehen und

immer im Speicher belassen. Dies ist wichtig bei Systemen mit nur zwei Puffern. Sind beide belegt, wird bei jedem neuen Datentransfer der alte Pufferinhalt überschrieben. DOS sorgt dafür, daß der Puffer mit dem FAT-Inhalt möglichst lange im Speicher bleibt. Nur der Aufruf einer Open-Funktion liest eine neue FAT in den Puffer.

Wird eine neue Datei angelegt oder soll ein neuer Cluster belegt werden, durchsucht DOS die File-Allocation-Tabelle auf den ersten freien Eintrag. Dieser wird dann der Datei zugeordnet und in die Kette der Cluster eingefügt, indem die Nummer des freien Clusters in den vorhergehenden Tabellenplatz der FAT eingetragen wird. Die aktuelle Position in der FAT erhält einen Wert (F)FF8 bis (F)FFF, um den letzten Cluster der Kette zu markieren.

Um den Eintrag einer Datei zu finden, ist folgendermaßen vorzugehen:

- Der Name der Datei ist im Inhaltsverzeichnis zu suchen. Dort findet sich die Nummer des ersten Clusters der Datei.
- Dieser Wert ist mit 1,5 (bei 12-Bit-FATs) oder mit 2 (bei 16-Bit-FATs) zu multiplizieren, um den Byteoffset in die FAT zu erhalten.
- Der Eintrag in der FAT gibt den nächsten Cluster an. Die neue Position in der Tabelle ist analog dem vorherigen Schritt zu ermitteln.
- Falls ein Wert (F)FF8 bis (F)FFFH in der FAT gefunden wird, sind keine weiteren Cluster für diese Datei mehr belegt.
- Um die aktuelle Sektornummer zu berechnen, an dem ein Cluster beginnt, ist von der Clusternummer der Wert 2 zu subtrahieren. Dann wird das Ergebnis mit der Zahl der Sektoren pro Cluster multipliziert. Anschließend ist der Offsetwert vom Beginn des Datenbereiches zu addieren. Damit ist der Sektor gefunden.

Normalerweise braucht sich der Anwender um diese Berechnung nicht zu kümmern, da dies durch DOS erledigt wird. Nur falls z.B. die Interrupts 25H und 26H benutzt werden, ist dies notwendig.

Bis DOS 3.3 wurden nur Platten mit maximal 32 Mbyte unterstützt. Die Größe der FAT war deshalb mit maximal 16.384 Einträgen à 16 Bit begrenzt. Ab DOS 4.0 darf die 16-Bit-FAT bis zu 65.536 Einträge (entspricht 128 Kbyte auf der Platte) belegen. Weiterhin übernimmt DOS ab der Version 3.0 eine Optimierung in Hinblick auf die Fragmentierung der Dateien vor. DOS merkt sich den letzten geschriebenen Cluster und beginnt bei dieser Nummer mit der Suche nach freien Clustern für weitere Schreibzugriffe. Erst beim Systemstart wird die Nummer für den ersten freien Cluster auf 0 zurückgesetzt. So belegen DOS-Dateien auf Festplatten häufig doch noch zusammenhängende Bereiche.

9.7 Das DOS-Inhaltsverzeichnis

Die FAT-Tabellen dienen zwar zur Verwaltung einzelner Dateien. Sie besitzen aber keine Informationen über den Dateinamen, den Anfangscluster oder den Zeitpunkt des letzten Schreibzugriffs. Andererseits lassen sich z.B. die Dateinamen mit dem DIR-Kommando auflisten. Es muß also auf der Platte/Diskette einen Bereich geben, wo diese Informationen abgelegt sind. Das Programm FORMAT baut diesen als Hauptinhaltsverzeichnis

(Rootdirectory) bezeichneten Bereich auf. Lage und Größe sind abhängig vom verwendeten Speichermedium und können durch den Treiber abgefragt werden.

Das Hauptinhaltsverzeichnis besteht aus einer Tabelle mit jeweils 32 Byte langen Records. Pro Eintrag kann eine Datei, ein Unterverzeichnis oder ein Volume Label angelegt werden. Die Tabellengröße reicht von 64 Einträgen bei $5^1/4$-Zoll-Disketten (einfache Dichte, einseitig beschrieben) bis zu mehreren hundert Positionen bei Festplatten.

Der Aufbau dieser Einträge entspricht folgender Struktur:

Bytes 00-07H

In den ersten 8 Byte wird der Dateiname als ASCII-Zeichen abgelegt. Dabei dürfen die gültigen DOS-Filenamen benutzt werden. Zusätzlich wird das erste Zeichen als Statusanzeige benutzt. Es können folgende Werte auftreten:

00H Der Eintrag wurde bisher nicht benutzt. DOS legt die Einträge in aufeinanderfolgenden Plätzen ab und markiert den letzten Platz mit dem Wert 00. Sobald der Wert 00H gefunden wird, ist der letzte Eintrag erreicht und DOS bricht die Suche ab. Programme dürfen das erste Zeichen eines Dateinamens also nicht zu 00H setzen, da eventuell nachfolgende Einträge dann nicht mehr gefunden werden.

05H Das erste Zeichen innerhalb des Dateinamens sollte eigentlich den Wert E5H enthalten, d.h. die Datei wurde gelöscht.

E5H Der Eintrag wurde zwar benutzt, aber die zugehörige Datei ist gelöscht.

2EH Mit diesem Eintrag wird ein Unterverzeichnis spezifiziert. Ist das folgende Byte ebenfalls mit 2EH belegt, enthält das Feld mit der Cluster-Nummer die Adresse des *Parent Directory*. Falls sich das Parent Directory auf das Hauptverzeichnis (Root Directory) bezieht, wird der Wert (0)000H eingetragen.

Alle anderen Werte im ersten Byte werden als Teil eines Dateinamens interpretiert. Der Name ist linksbündig in diesem 8-Byte-Feld einzutragen. Nicht benutzte Bytes sind mit einem Blank (20H) aufzufüllen.

Bytes 08-0AH

In diesem Feld wird die Dateiextension eingetragen. Diese umfaßt in der Regel drei ASCII-Zeichen, die linksbündig einzutragen sind. Unbenutzte Bytes sind mit Blanks (20H) aufzufüllen. Der Punkt zwischen Dateiname und Extension wird nicht im Directory eingetragen.

Byte 0BH

Hier handelt es sich um ein Feld, in dem die Eigenschaften (Attribute) der Datei abgelegt werden. Dies wirkt sich einmal auf die Schreib-/Lesezugriffe aus (z.B. Read Only Files). Weiterhin wird die Suche nach Dateien innerhalb eines Verzeichnisses über die Attribute gesteuert. Das DIR-Kommando in DOS gibt nur Dateinamen mit »normalen00H Die Datei besitzt keine Attribute, kann also gelesen und

beschrieben werden. Weiterhin wird sie bei der Directory-Suche gefunden und angezeigt.

01H Diese Datei läßt sich nur lesen, da sie als *Read Only* markiert ist. Der Aufruf einer DOS-INT 21-Funktion zum Beschreiben der Datei wird mit einer Fehlermeldung zurückgewiesen. Solche Aufrufe sind z.B. 3DH (Open File) mit einem Write-Access-Code, Delete File (13H, 41H) oder ein Create-Aufruf (3CH) auf eine bereits bestehende Datei. Die Datei wird beim normalen Directory-Suchaufruf angezeigt. Das Read-Only-Attribut läßt sich mit anderen Attributen kombinieren.

02H Die Datei wird mit dem Hidden-File-Attribut belegt. Dies bedeutet, daß sie beim Suchaufruf ausgeschlossen wird. Das DIR-Kommando zeigt z.B. keine *Hidden Files* an.

04H Ist dieses Bit gesetzt, handelt es sich um eine Systemdatei. Die Files IO.SYS (IBMBIO.COM) und MSDOS.SYS (IBMDOS.COM) sind zum Beispiel mit diesem Attribut belegt. Dateien mit dem Systemattribut werden bei der normalen Suche innerhalb des Inhaltsverzeichnisses überlesen.

08H Hier handelt es sich um das Volume-Attribut. In den ersten 11 Byte steht kein Dateiname, sondern ein ASCII-Text mit dem Volume Label des Speichermediums. Alle anderen Felder bleiben bis auf die Ausnahme des Datums und Uhrzeitfeldes unbelegt. Das Volume-Attribut darf nur innerhalb des Hauptverzeichnisses gesetzt werden. Das Bit kann durch den CHMOD-Aufruf nicht gelöscht werden.

10H Mit diesem Attributbit wird ein Unterverzeichnis markiert, d.h. es liegt keine normale Datei vor. Der Eintrag wird bei den Directory-Suchoperationen übergangen. Das Bit läßt sich durch den CHMOD-Aufruf nicht verändern.

20H Hier handelt es sich eigentlich nicht um ein Attribut, sondern um eine Markierung für die DOS-Restore- und Backup-Programme. Das Bit wird bei jeder Datei nach einem Write/Close-Aufruf gesetzt. Wird ein Backup angefertigt, erkennen diese Programme dieses Bit und archivieren die modifizierten Dateien. Anschließend wird das Bit gelöscht.

Die Attribute einer Datei lassen sich durch den INT 21-Aufruf 43H (Get/ Set File Attributes) modifizieren. Das Volume- und Subdirectory-Bit läßt sich aber nicht ändern.

Die Systemdateien IO.SYS (IBMBIO.COM) und MSDOS.SYS (IBMDOS.COM) werden durch FORMAT mit der /S-Option an den Anfang des Inhaltsverzeichnisses eingetragen. Sie besitzen das Read-only-, Hidden- und System-Attribut. Das Hidden-Attribut läßt sich bereits beim Create-Aufruf setzen. Die restlichen Bits des Attributbytes sind reserviert und enthalten den Wert 0.

Bytes 0CH-15H

Diese Bytes sind für DOS reserviert.

Bytes 16H-17H

Hier wird die Zeit des letzten Schreibzugriffes abgelegt. Es gilt die in Bild 9.11 dargestellte Kodierung. Das niederwertigste Byte der Uhrzeit findet sich auf der unteren Adresse (Byte 16H).

```
15              8  7              0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-ø-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙ-Ù-Ù-Ù-ÙÚÙÚÙ-Ù-Å-Ù-ÙÚÙÙÚÙ-Ù-Ù-ÙÚì
 Û---Ú---ì Û----Ú----ì Û---Ú---ì
    Std.       Min.       Sek.
   Sek.  :  0-59
   Min.  :  0-59
   Std.  :  0-23
```

Bild 9.11: Kodierung der Zeit im FCB

Bytes 18H-19H

Das Datum der letzten Änderung an der Datei findet sich in diesen beiden Bytes, wobei folgende Kodierung zugrunde liegt:

```
15              8  7              0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-ø-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙ-Ù-Ù-Ù-Ù-Ù-ÙÚÙÚÅ-ÙÚÙÚÙ-Ù-Ù-Ù-ÙÚì
 Û-----Ú-----ì Û-Ú-ì Û----Ú----ì
     Jahr      Monat     Tag
   Tag   :  1- 31
   Monat:  1- 12
   Jahr :  0-119  (1980-2099)
```

Bild 9.12: Kodierung des Datums im FCB

Das *Low Byte* des Datums wird dabei zuerst gespeichert (Byte 18H).

Bytes 1AH-1BH

In diesem Feld wird die Nummer des ersten Clusters der Datei abgelegt. Dabei wird das Low-Byte der Clusternummer zuerst (Byte 1AH) gespeichert. Der niedrigste Wert ist der Cluster 0002, ab dem der Datenbereich beginnt. Das Feld erlaubt die Speicherung von 12-Bit- und 16-Bit-Clusternummern. Im ersten Fall sind die oberen Bits zu 0 gesetzt.

Bytes 1CH-1FH

Hier wird die Dateigröße in Bytes geführt, wobei die untersten Bytes auch zuerst gespeichert werden.

Die Daten innerhalb des Inhaltsverzeichnisses werden durch die INT 21-Close-Funktion beschrieben. Zusätzlich gibt es weitere Funktionen, die einzelne Felder modifizieren.

- Create File setzt z.B. bei bestehenden Dateien die Länge zu 0.
- Mit dem Set-Attribut-Aufruf lassen sich die Attribut-Bytes löschen.
- Die Delete-Funktion setzt das erste Byte auf den Wert E5H.

Da das Hauptverzeichnis nur eine feste Zahl von Einträgen besitzt, wäre normalerweise die Zahl der Dateien pro Medium begrenzt. DOS besitzt aber die Möglichkeit, Unterverzeichnisse anzulegen. Diese Unterverzeichnisse werden aus Sicht der *Root Directory* als normale Dateien behandelt. Innerhalb einer solchen »Unterverzeichnisdateiird bei MS-DOS (PS-DOS) also nur durch den Speicherplatz bestimmt. Allerdings ist bei vielen kleinen Dateien die »Nettobelegung**Fehler! Verweisquelle konnte nicht gefunden werden.**Ab DOS 4.0 werden die folgenden Formate für 5,25- und 3,5-Zoll-Disketten unterstützt.

Seiten	Sektoren/ Spur	FAT Sektoren	DIR Sektoren	DIR Einträge	Sektoren/ Cluster
1	8	1	4	64	1
2	8	1	7	112	2
1	9	2	4	64	1
2	9	2	7	112	2
2	15	7	14	224	1
2	9	3	7	112	2
2	18	9	14	224	1

Die ersten 5 Formate werden bei 5,25-Zoll-Disketten benutzt, während sich die restlichen Formate auf 3,5-Zoll-Disketten beziehen.

9.8 Der DOS-Datenbereich auf der Platte/Diskette

An das Hauptinhaltsverzeichnis schließt sich der Bereich an, in dem DOS die Dateien ablegt. Der Speicher wird immer in Form von Clustern zugewiesen oder zurückgegeben. Diese bestehen ja aus mehreren aufeinanderfolgenden Sektoren. Die Zahl dieser Sektoren pro Cluster hängt vom Medium ab. Bei Platten wird er durch FORMAT bestimmt und hängt im wesentlichen von der Größe der Partition ab. Die Cluster sind so arrangiert, daß die Kopfbewegungen minimiert werden.

Die Dateien werden nicht unbedingt sequentiell im Datenbereich abgelegt, sondern können über das ganze Medium verstreut sein. Die Verwaltung erfolgt über die File-Allocation-Tabelle. Wird ein neuer Cluster angefordert, sucht DOS die FAT auf freie Cluster ab. Der erste gefundene Cluster wird zugewiesen, unabhängig von der Lage der Datei. So kann es sein, daß Dateien am Ende einer Platte erweitert werden sollen. Falls vorher Einträge am Beginn des Datenbereiches freigegeben wurden, legt DOS den angeforderten Cluster in diesen Bereich. Dies führt mit der Zeit zu einer Fraktionierung der Dateien auf der Platte. Dies bedeutet vermehrte Kopfpositionierungen mit schlechteren Zugriffszeiten.

Weitere Informationen über die Verwaltung des Datenbereichs finden sich im Abschnitt über die File-Allocation-Tabelle.

10 Die DOS-Dateiformate

In den vorhergehenden Kapiteln wurde beschrieben, wie DOS-Daten auf Platten oder Disketten abspeichert werden. Aus Sicht der BIOS- und DOS-Dateiverwaltung existieren nur Dateien, die in geeigneter Form zu verwalten sind. Aber der DOS-Lader sowie die Anwenderprozesse unterscheiden bereits zwischen verschiedenen Dateiformaten.

Wie die einzelnen Formate aufgebaut sind und wie sie durch die DOS-Programme behandelt werden, ist Thema dieses Kapitels.

10.1 Die DOS-Textdateien

DOS besitzt die Möglichkeit, Daten als zeichenorientierte Datei abzulegen. Hierbei existiert eigentlich kein besonderes Format, sondern alle Zeichen werden als Bytes interpretiert und sequentiell in der Datei abgelegt. Damit kann nicht automatisch unterschieden werden, ob die Daten zu einer Textdatei gehören, da ja alle Werte zwischen 00H und 255H vorkommen. Lediglich bei der Ausgabe mit TYPE oder PRINT vermag die menschliche Intelligenz einen Text zu identifizieren. Die nachfolgende Abbildung enthält einen Speicherdump einer kleinen Textdatei.

```
-d
0D 0A 20 56 6F 6C 75 6D-65 20 69 6E 20 4C 61 75
 .  .     V o   l  u  m  e     i  n     L  a  u
66 77 65 72 6B 20 41 20-68 61 74 20 6B 65 69 6E
 f  w  e  r  k     A     h  a  t     k  e  i  n
                  .
                  .
20 20 20 35 20 44 61 1A-65 69 28 65 6E 29 20 20
          5     D  a  .  e  i  (  e  n  )
20 20 31 35 32 35 37 36-20 42 79 74 65 73 20 66
       1  5  2  5  7  6     B  y  t  e  s     f
72 65 69 0D 0A
 r  e  i  .  .
```

Bild 10.1: Speicherdump einer ASCII-Datei

Die Länge der Datei ist im Inhaltsverzeichnis des Speichermediums abgelegt. Die Zeichen 0DH und 0AH bilden den Abschluß des Textes (Return und Linefeed).

Finden sich nun aber Steuerzeichen im Text, werden diese durch die entsprechenden DOS-Programme oder durch die Einheitentreiber interpretiert. In obige Datei wurde zum Beispiel ab Offset 217H das Zeichen 1AH (<Ctrl><Z>) eingebracht. Dieses Control-Z wird von vielen DOS-Programmen als Dateiende betrachtet. Eine Ausgabe mit PRINT, COPY oder TYPE bricht dann an dieser Stelle ab, obwohl noch Zeichen in der Datei vorhanden sind. Dies ist zu beachten, falls ASCII-Dateien bearbeitet werden sollen.

10.2 Die DOS-COM-Dateien

Ein häufig verwendeter Dateityp ist das COM-Format. Eine solche Datei besteht ebenfalls aus einer Folge von Bytes, die den entsprechenden Programmcode repräsentieren. Nachfolgendes Bild zeigt einen Speicherdump einer solchen Datei.

```
-d
112C:0100  E9 79 2C 90 90 CD AB 43-6F 70 79 72 69 67 68 74
112C:0110  20 28 43 29 20 31 39 38-35 20 42 4F 52 4C 41 4E
112C:0170  07 07 70 0F 07 07 70 0E-07 07 4F 2E 8A 27 0A E4
```

Bild 10.2: Speicherdump einer COM-Datei

Die COM-Datei enthält ein getreues Abbild eines Codebereiches aus dem Hauptspeicher. Die Zahl der gespeicherten Bytes ist nur im Inhaltsverzeichnis der Datei abgelegt. Programme wie DEBUG enthalten die Länge der Datei nach dem Laden in den Registern des Prozessors.

Für COM-Files gelten bestimmte Randbedingungen, falls sie lauffähig sein sollen. So darf die Länge einer COM-Datei den Bereich von 64 Kbyte nicht überschreiten. Der Programmcode muß ab dem Offset 100H beginnen, da unterhalb dieser Adresse der PSP zur Laufzeit im Speicher liegt. Beim Assembler ist die ORG-100H-Anweisung zu verwenden, falls eine COM-Datei erzeugt werden soll.

Beim Laden von COM-Dateien nimmt die DOS-EXEC-Funktion bestimmte Einstellungen vor:

- Der Programmcode wird ab Offset 100H in den reservierten Programmspeicher geladen. Der PSP liegt unterhalb dieses Programmbereiches.
- Der gesamte verfügbare Speicherbereich wird vom Lader für das Anwenderprogramm reserviert.
- Vor der Übergabe der Kontrolle an das geladene Programm werden alle vier Segmentregister (CS, DS, SS, ES) auf die Anfangsadresse des reservierten Speicherblocks gesetzt.
- Der Stackpointer wird auf die oberste Adresse des 64-Kbyte-Bereiches gesetzt. Im PSP wird ab Offset 06H die Zahl der Bytes im Programmsegment gehalten. Dieser Wert wird um 100H erniedrigt, um den entsprechenden Stackbereich zu reservieren. Die obersten 2 Byte auf dem Stack werden mit den Werten 00H initialisiert.

Durch diese Einstellungen ergeben sich für COM-Programme einige Einschränkungen zur Laufzeit. Einmal ist der Speicherbereich für Code, Daten und Stack auf 64 Kbyte begrenzt. Diese Größe rührt noch aus der CP/M-Zeit, wo der gesamte Speicher nicht größer als 64 Kbyte war. Heute reicht dieser Wert für viele Anwendungen nicht mehr aus. Andererseits reserviert der Lader den gesamten Speicherbereich für den Prozeß. Soll ein zweiter Subprozeß gestartet werden (Funktion 4B00H), ist vorher erst der nicht benötigte Speicher zurückzugeben.

Weiterhin müssen sich alle Sprünge innerhalb des Codesegments auf Adressen mit dem Offset 100H beziehen, da ja der Code um 256 Byte im Segment verschoben ist.

Der Stackbereich findet sich im oberen Bereich des 64-Kbyte-Segments. Die Entwickler haben bei der Definition des Stacks tief in die Trickkiste gegriffen. Auf der obersten Adresse findet sich der Wert 0000H. Bereinigt nun ein laufender Prozeß den Stack bis auf diese beiden Einträge, kann er mit einer einfachen RETURN-Anweisung beendet werden. Der Prozessor liest dann die letzten beiden Bytes vom Stack (0000H) und interpretiert diese als Rückkehradresse. Normalerweise würde dies zu einem Programmabsturz führen, da ja der Prozessor zur Adresse 0000H des aktuellen 64-Kbyte-Codesegments verzweigt. Bei COM-Dateien findet sich aber hier das PSP, welches ab Offset 00H eine INT 20-Anweisung (Terminate Process) abgespeichert hat. Damit wird der Prozeß korrekt beendet.

Durch Verändern der Daten- und Stacksegmentregister kann ein Prozeß auch den verfügbaren Speicherbereich vergrößern. Allerdings wird diese Operation nicht durch das Betriebssystem unterstützt, so daß diese Möglichkeit nicht genutzt werden sollte.

COM-Dateien lassen sich aus EXE-Dateien mittels des Programmes EXE2BIN.COM erzeugen. Die Dateigröße ist dabei geringer als bei den ursprünglichen EXE-Files, da alle Adreßinformationen bei der Konvertierung entfernt werden.

Für neuere Entwicklungen sind EXE-Dateien vorzuziehen, die die obigen Nachteile vermeiden.

10.3 Die DOS-EXE-Dateien

Die Sprachübersetzer erzeugen normalerweise nur sogenannte Objektdateien, in denen das reine Programmskelett abgelegt ist. Diese Dateien lassen sich durch den DOS-LINKER zu einem ausführbaren Modul zusammenbinden. Dabei können mehrere Module kombiniert und durch Bibliotheken ergänzt werden. Der Linker legt das Ergebnis in Form einer EXE-Datei ab. Diese enthält neben dem Programmcode Listen zur Steuerung des Ladevorganges und zur Berechnung der Adressen im endgültigen Speicherabbild. Alle relativen Sprünge und Adreßinformationen werden erst zur Laufzeit endgültig berechnet. Damit ist die Datei in verschiedenen Speicherbereichen lauffähig. Das EXE-Format unterscheidet sich deshalb auch sehr stark von den bisher besprochenen Formaten. Die Datei beginnt mit einem Kopf (Header), der Informationen über den Aufbau der Datei enthält. Daran schließen sich die Relokationstabellen an. Erst dann kommt der eigentliche Codebereich, der das Programm in Rohform enthält. Die nachfolgende Tabelle zeigt den Aufbau des Headers der EXE-Datei:

Offset	Bytes	Feld
00H	2	Die zwei ersten Bytes enthalten die Markierung 4DH, 5AH für die EXE-Datei. Dies sind die Initialien 'M' 'Z' eines der DOS Entwickler
02H	2	Länge des belegten Speichers mod 200H. Hier steht der Rest der Datei, der nicht in einen 512 Byte Block paßt.
04H	2	Länge der Datei in 512 Byte (Pages) Blöcken. Der Wert umfaßt auch den EXE-Kopfbereich
06H	2	Zahl der Einträge in der Relokationstabelle.
08H	2	Größe des EXE-Headers in 16 Byte (Paragr.) Blöcken. Damit läßt sich der Anfang des Code- bereiches im File bestimmen.
0AH	2	Minimale Zahl von Paragraphen, die oberhalb d. Programmcodes im Speicher zu reservieren sind.
0CH	2	Maximale Zahl von Paragraphen, die oberhalb d. Programmcodes im Speicher zu reservieren sind.
0EH	2	Displacement des Stacksegments im Speicher, angegeben in Paragraphen ab dem Beginn des Ladesegments.
10H	2	Offsetwert, der im SP-Register abgelegt wird, bevor das Programm gestartet wird.
12H	2	Checksumme (Summe) aller Worte in der Datei mod 0FFFFH und negiert.
14H	2	Offsetwert der im IP-Register abgelegt wird, bevor das Programm gestartet wird.
16H	2	Displacement in Paragraphen für das Codeseg- ment relativ zum Beginn des Ladebereiches
18H	2	Displacement in Bytes des ersten zu relokie- renden Eintrages in der Datei.
1AH	2	Overlaynummer des Programms. Bei 00 handelt es sich um ein residentes Programm.

Tabelle 10.1: Aufbau des EXE-File-Headers

Im Gegensatz zur COM-Datei finden sich hier verschiedene Informationen für den Lader, wie groß der zu reservierende Speicherbereich ist (Offset 0AH und 0CH). Weiterhin kann angegeben werden, wo das Code- und das Stacksegment beginnt. Damit läßt sich das Programm an jede beliebige Speicherstelle laden. Die anderen Einträge beziehen sich überwiegend auf die Relokationstabellen.

Der Aufbau der Relokationstabelle

Die EXE-Datei enthält eine Tabelle mit allen Adressen, die sich bei einer Verschiebung des Programmanfangs verändern. Die Länge dieser Tabelle ist variabel und steht ab Offset 06H. Im Kopf des EXE-Files steht ab Offset 18H die Adresse des ersten Eintrages in dieser Tabelle. Jeder Eintrag in dieser Tabelle besteht aus 4 Byte, einem 2-Byte-Offset und einem 2-Byte-Displacement in den Codebereich der EXE-Datei. Beim Laden des Moduls werden alle durch die Relokationstabelle markierten Adressen modifiziert. Alle Adressen beziehen sich nämlich auf die Programmladeadresse 0. Wird zum Beispiel das Programm ab Adresse 100H geladen, ist zu jeder markierten Adresse dieser Offset zu addieren.

Bevor der Prozeß gestartet wird, sind die Register SS und SP auf die Werte im EXE-Header zu initialisieren. Die Werte von ES und DS werden auf den Wert der Segmentadresse des PSP gesetzt. Die Startadresse des Codesegments wird in CS zum Startsegment addiert. Der Eintrag des EXE-Headers in IP bestimmt dann die Programmstartadresse.

Diese Operationen übernimmt automatisch der DOS-Programmlader, so daß sich der Anwender nicht um diese Sachverhalte kümmern muß.

10.4 Die Font-Dateistruktur

Ab DOS 3.3 besteht die Möglichkeit, verschiedene Zeichenfonts auf den Ausgabeeinheiten zu installieren. Damit lassen sich dann leicht unterschiedliche Schriftarten darstellen. Diese als *Code Page* bezeichneten Zeichenfonts werden vom jeweiligen Treiber aus bestimmten Dateien geladen. Alle Dateien mit der Extension CPI (Code Page Image) enthalten solche Zeichenfonts. Bei DOS werden Fonts für die Bildschirme und die Drucker angeboten. Alle Dateien besitzen einen gemeinsamen Aufbau:

Bytes	Bedeutung
---	Header des Fontfiles
08	0FFH,"font " 7 Zeichen mit der Fontfilekennung
08	reserviert (00..00)
02	Zahl der Zeiger im Header (DOS 3.3 = 1)
01	Typ des Info Zeigers (DOS 3.3 = 1)
02	Displacement ab Fileanfang
02	00H als Endekennung

Tabelle 10.2: Aufbau eines Fontfiles für ein Display

Bytes	Bedeutung
---	Infoteil
02	Zahl der Code-Page-Einträge
---	Code-Page-Kopfteil
02	Größe des Kopfteils in Bytes
04	Zeiger auf den Header des nächsten Code Page (0,0 für den letzten Eintrag)
02	Einheitentype 1= Display 2 = Drucker
08	Name des Zeichenfonts (z.B. "EGA ")
02	Code Page ID Code = 999
06	reserviert (auf 00 zu setzen)
04	Zeiger zum Font (2 Worte, Displacement = 0)
---	Header des Datenbereichs
02	reserviert (muß auf 1 gesetzt sein)
02	Zahl der Zeichenfonts (03)
02	Länge des folgenden Fontdatenbereiches
---	Font-Daten
01	Zeilengröße eines Zeichens (meist 16 Punkte)
01	Spalten eines Zeichens (meist 8 Punkte)
02	unbenutzt und auf 0,0 gesetzt
02	Zahl der Zeichen im Fontfile (256)
xx	n Datenbytes mit der Beschreibung der Pixels aller 256 Zeichen im Font

Tabelle 10.2: Aufbau eines Fontfiles für ein Display (Ende)

Für verschiedene Drucker existieren eigene Fontdateien, die sich im wesentlichen im Datenbereich unterscheiden. Für weitere Informationen sei auf die Unterlagen der Hersteller verwiesen.

11 Sonderfragen zu DOS

Neben der Beschreibung der DOS-Systemschnittstellen ergeben sich immer wieder Fragen hinsichtlich bestimmter Abläufe. Das Kapitel über die Speicherverwaltung ist ein Beispiel, welches bereits ausgiebig besprochen wurde. Andere Fragen sind aber noch offen. Wie erkennt das System zum Beispiel zusätzliche Adapterkarten, die eigene BIOS-Routinen enthalten? Warum sollten die handleorientierten Funktionsaufrufe den FCB-orientierten Aufrufen vorgezogen werden?

Diese und ähnliche Fragestellungen sollen in diesem Kapitel kurz angerissen werden.

11.1 Der Programmablauf nach einem temstart

Nach einem Systemrestart beginnt der Prozessor mit der Bearbeitung des Programmes ab Adresse FFFF:0000. Die oberen 16 Byte des Adreßbereiches sind für die Restartroutine reserviert. Hier findet sich meist ein Sprungbefehl in die BIOS-Startroutine. Diese führt dann einen Power-On-Self-Test (POST) durch, wobei die Hardware überprüft und die Konfiguration festgestellt wird. Das Ergebnis wird dann im BIOS-Datenbereich abgelegt. Die Hardwarekonfiguration wird durch Abfrage der Kodierungsschalter ermittelt. Bei Systemen mit CMOS-CLOCK-Bausteinen findet sich die Konfiguration im RAM-Bereich dieses Bausteins.

Leider bleibt die Entwicklung der Hardware nicht stehen. Im Laufe der Zeit kamen Festplatten, EGA-Karten etc. als Erweiterungen hinzu. Alle diese Karten enthalten üblicherweise eine BIOS-Erweiterung auf der Platine. So definiert die EGA-Karte den INT 10 auf eigene Routinen um. Auch der INT 13 wird durch das BIOS der Festplatte verändert. Es stellt sich die Frage, wie das BIOS die fremden Karten erkennt? Vielfach konnten die BIOS-Entwickler nicht absehen, welche Erweiterungskarten im Laufe der Zeit in die Systeme eingebaut werden würden. Sie haben deshalb einen bestimmten Übergabemechanismus definiert, der eine einfache Erweiterung des Systems erlaubt, ohne daß das BIOS technische Kenntnisse über die neue Karte besitzen muß.

Der Speicherbereich oberhalb von C0000H bis EFFFFH ist für solche ROM-Erweiterungen reserviert. Beim Systemstart wird der Bereich dann in 2-Kbyte-Schritten auf ROMs untersucht. Ein ROM muß ab Offset 00H bestimmte Kennungen besitzen, damit es erkannt wird (Tabelle 11.1).

Offset	Bedeutung
00	55H Signatur 1 für Erweiterungs-ROMs
01	AAH Signatur 2 für Erweiterungs-ROMs
02	ROM Länge in 512-Byte-Blöcken
03	Adresse der Initialisierungsroutine

Tabelle 11.1: Aufbau der Signatur für Erweiterungs-ROMs

Wird die Signatur 55AAH gefunden, vermutet das BIOS eine Erweiterungsroutine. Zur Sicherheit wird noch die Checksumme über den ROM-Bereich (Addition aller Bytes

modulo 100H) berechnet. Ist das Ergebnis 00, dann geht die Kontrolle per LONG CALL an die Initialisierungsroutine dieser Karte. Die Transferadresse findet sich hinter der ROM-Signatur (Offset 03H). Diese kann dann die entsprechenden Interruptvektoren manipulieren. Anschließend erhält das BIOS die Kontrolle zurück. Nachdem alle Karten so installiert sind, wird nach einem bootfähigen Medium gesucht. Fehlt ein bootfähiges Medium, bricht das BIOS-Programm den Systemstart mit einer Fehlermeldung ab. Falls eine Diskette oder Platte gefunden wird, lädt das BIOS den ersten Sektor (Bootrecord) des Mediums in den Speicher. Hier findet sich ein kleines Programm, welches auf dem Medium die DOS-Dateien IO.SYS (IBMBIO.COM) und MSDOS.SYS (IBMDOS.COM) sucht und gegebenenfalls lädt. Falls diese nicht an bestimmten Stellen auf dem Medium stehen, wird der Systemstart mit einer Fehlermeldung abgebrochen. Falls die Module geladen wurden, aktiviert der Lader dann den Initialisierungsteil von IO.SYS. Dieser baut den BIOS-Datenbereich weiter auf und installiert die DOS-Treiber. Dann wird der DOS-Teil initialisiert und der Kommandoprozessor geladen. Nähere Einzelheiten enthält das Kapitel über die BIOS-Routinen.

11.2 Der Aufbau des Konfigurations-RAM im CMOS-Clock-Baustein

Alle Systeme mit einer CMOS-Uhr enthalten die Informationen über die Hardwarekonfiguration im RAM-Bereich des Uhrenbausteins. Da diese Informationen manchmal von Interesse sein können, soll der Aufbau hier kurz vorgestellt werden.

Bytes	Bedeutung
01	Daten der REAL-Time-Clock (Sekundenalarm)
02	Daten der REAL-Time-Clock (Minuten)
03	Daten der REAL-Time-Clock (Minutenalarm)
04	Daten der REAL-Time-Clock (Stunden)
05	Daten der REAL-Time-Clock (Stundenalarm)
06	Daten der REAL-Time-Clock (Sekunden)
07	Daten der REAL-Time-Clock (Wochentag)
08	Daten der REAL-Time-Clock (Monat)
09	Daten der REAL-Time-Clock (Jahr)
0A	Statusregister der Real-Time-Clock
.	
0D	
0E	Diagnose Statusbyte
0F	Shutdown Statusbyte
10	Typ des Diskettenlaufwerkes
11	reserviert
12	Typ der Festplatte
13	reserviert
14	Equipment Byte
15	Low Base Memory Byte
16	High Base Memory Byte
17	Low Expansion Memory Byte
18	High Expansion Memory Byte
19-2D	reserviert
2E-2F	2 Byte Checksum
30	Low Expansion Memory Byte
31	High Expansion Memory Byte
32	Datum Jahrhundert Byte
33	Info Flags
34-3F	reserviert

Tabelle 11.2: Der Aufbau des Konfigurations-RAM

Die Bytes 0-0FH und 30H-33H werden nicht in die Berechnung der Checksumme einbezogen. Falls die Checksumme vom gespeicherten Wert abweicht, bringt das System beim Start eine Fehlermeldung. Dann ist das Konfigurations-RAM neu zu definieren. Andernfalls werden die Daten in den BIOS-Datenbereich übernommen.

Für die anderen Bytes gelten folgende Belegungen:

Status-Byte (0EH)

Hier finden sich Informationen über den Systemstatus. Es gilt folgende Belegung:

Bit 7 = 1Zeigt an, daß die Uhr einen Spannungseinbruch hatte, womit die Konfiguration und die Uhrzeit neu zu setzen sind.

Bit 6 = 1Die Checksumme über den RAM-Bereich ist ungültig.

Bit 5 = 1Die Konfiguration des Systems weicht von den Einstellungen im RAM ab.

Bit 4 = 1Die ermittelte Speichergröße weicht von der Eintragung im RAM ab.

Bit 3 = 1Die Adapter für Disketten/Festplatte haben einen Fehler.

Bit 2 = 1Die Uhrzeit im Baustein ist nicht mehr gültig.

Die unteren 2 Bits sind reserviert.

Diskettentyp (Byte 10H)

Hier finden sich Informationen über die Diskettenlaufwerke des Systems. Es gilt folgende Kodierung:

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙ-Ù-ÙÚÙÚÙ-Ù-ÙÚì
 Û--Ú--ì Û--Ú--ì  erstes Diskettenlaufwerk
    °       Û---  0000 nicht vorhanden
    °             0001 Doppelseitige Diskette (48 TPI)
    °             0010 High-Density-Diskette (96 TPI)
    °
    °             zweites Diskettenlaufwerk
    Û-----------  0000 nicht vorhanden
                  0001 Doppelseitige Diskette (48 TPI)
                  0010 High-Density-Diskette (96 TPI)
```

Bild 11.1: Belegung des Diskettentyps (Byte 10H)

Die restlichen Codes sind reserviert.

Festplattentyp (Byte 12H)

Hier finden sich Informationen über die Festplattenlaufwerke des Systems. Es gilt folgende Kodierung:

```
 7 6 5 4 3 2 1 0
Ö-Ú-Ú-Ú-Ú-Ú-Ú-Ú-Ì
ÛÚÙ-Ù-ÙÚÙÚÙ-Ù-ÙÚì
 Û--Ú--ì Û--Ú--ì  erstes Plattenlaufwerk
    °       Û---  0000 nicht vorhanden
    °             xxxx Code des Plattentyps
    °
    °             zweites Plattenlaufwerk
    Û-----------  0000 nicht vorhanden
                  xxxx Code des Plattentyps
```

Bild 11.2: Belegung des Plattentyps

Die Codes reichen von 0001 bis 1110 für die verschiedenen Laufwerke. Die Zuordnung der Laufwerke zu den Codes ist folgender Tabelle zu entnehmen. Der Wert 1111 ist reserviert.

Type	Zylinder	Köpfe	Landezone
01	306	4	305
02	615	4	615
03	615	6	615
04	940	8	940
05	940	6	940
06	615	4	615
07	462	8	511
08	733	5	733
09	900	15	901
0A	820	3	820
0B	855	5	855
0C	855	7	855
0D	306	8	319
0E	733	7	733
0F	--	--	---

Tabelle 11.3: Kodierung des Disk-Typs

Equipment-Byte (Bytes 14H)

Das System legt in diesem Byte die Konfigurierung der Hardware ab. Es gilt folgende Belegung:

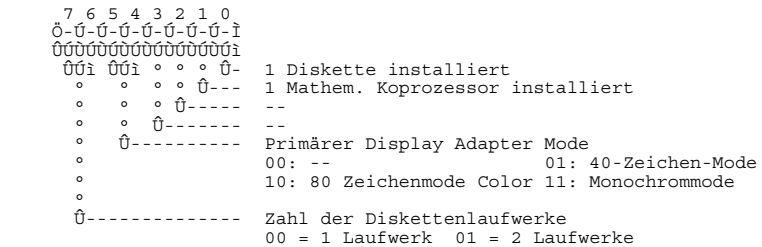


Bild 11.3: Belegung des Equipmentbytes

Speichergröße

In den Bytes 15H und 16H findet sich die Größe des Hauptspeichers in Kbyte-Blöcken. Ein Wert von 200H gibt zum Beispiel eine Größe von 512 Kbyte an. Ab Offset 17H und 30H findet sich die Größe des Expanded Memory in Kbyte. Der Wert 100H spezifiziert, daß das System mit 256-Kbyte-Erweiterungsspeicher ausgestattet ist. Der größte Wert ist 3C00H, was einer 15-Mbyte-Speichererweiterung entspricht.

Die restlichen Bytes sind für die Uhr und die BIOS-Statusflags reserviert. Um eine Adresse im CMOS-RAM zu beschreiben, ist die Offsetadresse (z.B. 12H) mit einem OUT-Befehl auf den Port 70H zu schreiben. Anschließend folgt ein OUT-Befehl mit dem zu setzenden Wert auf den Port 71H. Ein Lesevorgang erfordert ebenfalls die Ausgabe der Adresse auf dem Port 70H. Anschließend kann das Ergebnis durch einen IN-Befehl vom Port 71H

gelesen werden. Genauere Informationen sind den entsprechenden Herstellerunterlagen zu entnehmen.

11.3 Die Dateibehandlung in DOS

DOS kennt intern die Möglichkeit, eine Datei über File-Control-Blocks oder über Handles zu verwalten. Hierbei gelten folgende Bedingungen:

Die FCB-Funktionsaufrufe

Die FCB-Aufrufe wurden aus der DOS-1.x-Version übernommen. Sie setzen voraus, daß sich die Dateien im aktuellen Unterverzeichnis befinden. Files in Unterverzeichnissen werden nicht unterstützt. Falls eine Datei über File-Control-Blocks geöffnet wird, müssen auch alle anderen FCB-Funktionsaufrufe benutzt werden. Vor dem OPEN-Aufruf ist vom aktiven Prozeß ein Datenbereich mit einem ungeöffneten FCB einzurichten. Hierunter wird ein FCB verstanden, in dem lediglich der Dateiname eingetragen ist. Beim Open-Aufruf setzt DOS die restlichen Felder des FCBs. Pro geöffneter Datei ist ein eigener FCB zu verwalten.

Die Benutzung dieser Aufrufe in Anwenderprogrammen kann zu erheblichen Problemen führen. So kann eine Datei mit einer Close-Anweisung geschlossen werden. Anschließend sind erneute Schreibzugriffe des Prozesses auf diese Datei möglich. Falls kein zweiter Close-Befehl folgt, werden die Daten auf die Diskette geschrieben, ohne die Informationen im Inhaltsverzeichnis zu modifizieren. Damit befindet sich die Datei in einem inkonsistenten Zustand.

Weiterhin treten Probleme innerhalb von Netzwerkanwendungen auf, da die FCB-Aufrufe nur den *Compatibility Mode* erlauben. Es ist auch möglich, eine Datei mit einem FCB zu öffnen. Falls dieser FCB irrtümlicherweise ein weiteres Mal benutzt wird, um eine andere Datei zu bearbeiten, tritt keine Fehlermeldung auf.

Die neue Datei läßt sich auch korrekt bearbeiten. Ein Wechsel auf die erste Datei verursacht dann Probleme, da die DOS-Funktion zwar den Filenamen im FCB erkennt. DOS legt aber in den reservierten Bereichen des FCB Informationen über das Filesystem ab, die durch den zweiten Open-Aufruf überschrieben werden. Operationen auf der ersten Datei sind damit nicht mehr möglich. Es ist also für jede Datei ein eigener FCB anzulegen.

Eine Verwendung in neueren Programmen sollte deshalb aus obigen Gründen vermieden werden.

Die Handle-Funktionsaufrufe

Ab Version 2.x bietet DOS neue Möglichkeiten zur Dateiverwaltung. Dabei werden die Informationen intern im Dateisystem verwaltet, während dem externen Anwenderprogramm ein Refer enzcode (Handle) auf die entsprechende Einheit zugeteilt wird. Da die Information ü ber eine Datei zentral durch DOS (in einer Tabelle) verwaltet wird, kann dieser Handle durchaus von mehreren Prozessen verwendet werden. Die Zuteilung des Handlecode s beim DOS-Open-Aufruf verhindert weiterhin, daß zwei Dateien mit einem Handleco de belegt werden. Zugriffe auf nicht existierende Handles

werden abgewiesen. Mit den handleorientierten Funktionsaufrufen lassen sich Dateien in allen Einheiten und Unte rverzeichnissen ansprechen. Der Prozeß muß lediglich eine Handlenummer intern verwalten. Weiterhin werden insbesondere die Netzwerkfunktionen durch die Funktionen unterstützt. Dabei kommt der Möglichkeit, bestimmte Zugriffsrechte zu definieren (Compatibility, Deny read, Deny write, Deny both etc.), eine erhöhte Bedeutung zu.

Vor einen Open-Aufruf muß das Programm einen ASCIIZ-String mit dem Dateinamen erzeugen. Dieser kann Laufwerks- und Pfadbezeichnungen enthalten. Sobald ein Handle zugeteilt wurde, kann dieser ASCIIZ-String gelöscht werden. Die Handle-Funktionsaufrufe sind jedoch nicht auf Dateien beschränkt, sondern erlauben ebenfalls die Interaktion mit anderen Standard-I/O-Einheiten (Printer, Tastatur, Bildschirm etc.). DOS öffnet bereits beim Systemstart bestimmte Standardeinheiten.

Handle	Einheit
0000	Standard-Eingabe-Einheit
0001	Standard-Ausgabe-Einheit
0002	Standard-Error-Einheit
0003	Standard-Auxiliary-Einheit
0004	Standard-Printer-Einheit

Tabelle 11.4: Handles der Standard-Einheiten

Die Tabelle 11.4 enthält eine Aufstellung dieser vordefinierten 16-Bit-Handlecodes. Diese lassen sich durch jeden Prozeß benutzen und müssen nicht explizit eröffnet werden.

Von der Standard-Eingabeeinheit kann nur gelesen werden, während für die Standard-Ausgabe-, Printer- und Error-Einheiten nur Schreibzugriffe zulässig sind.

DOS bietet mit den Handles eine sehr elegante Möglichkeit der Eingabe-/Ausgabe-umleitung. Bei jedem Aufruf zum Datentransfer wird immer der Handlecode mit übergeben. Eine Variation des Handlecodes dirigiert die Ein-/Ausgabe auf eine andere Einheit um. Falls einer Datei z.B. der Handlecode 0010H zugewiesen wurde, beziehen sich alle Write-Aufrufe auf diese Datei. Wird aber beim Aufruf der Handlecode 0004H übergeben, erfolgt die Ausgabe auf dem Drucker. Analog kann bei der Eingabe verfahren werden. Benutzt ein Prozeß die Standard-Eingabe, läßt sich diese Eingabe leicht auf andere Einheiten umsetzen (z.B. Dateien).

Um diese Möglichkeiten zu demonstrieren, wurde nachfolgendes kleine Demonstrationsprogramm entwickelt. Es erlaubt die Ausgabe eines ASCII-Textes an verschiedene Einheiten (Bildschirm, Drucker, Datei). Das Programm gliedert sich in verschiedene Module, die hier kurz vorgestellt werden.

fehler_mod

Dieses Modul wird im Fehlerfall aufgerufen und dient zum Abbruch des laufenden Prozesses. Beim Aufruf ist ein Fehlercode zu übergeben. Dieser Fehlercode wird auf dem Bildschirm ausgegeben. Dann wird der Extended-DOS-Fehlercode über die Funktion 59H des INT 21 abgefragt und auf dem Bildschirm angezeigt. Zum Abschluß wird noch der

Abbruch eines laufenden Prozesses über die INT 21-Funktion 4CH (Terminate a Process) demonstriert.

write_f

Diese Prozedur definiert das Interface zu der INT 21-Funktion 40H (Write to a Device or File). Der übergebene Text wird mit dem Handlecode an die DOS-Funktion übergeben. Durch Variation des Handlecodes läßt sich die Ausgabe an verschiedene Einheiten dirigieren. Im Fehlerfall wird der Prozeß über die Prozedur *fehler_mod* beendet.

create_f

Diese Prozedure erzeugt eine Datei auf dem Speichermedium. Dabei ist einmal der Dateiname mit Laufwerksname und Zugriffspfad anzugeben. Weiterhin wird noch das Dateiattribut übergeben. Die Prozedur demonstriert intern die Verwendung der INT 21-Funktion 3CH (Create a File). Falls die Datei noch nicht vorhanden ist, wird ein entsprechender Eintrag im Inhaltsverzeichnis des Mediums vorgenommen. Bestehende Dateien werden lediglich auf die Länge 0 gesetzt. Nach einem fehlerfreien Aufruf enthält das Speichermedium eine leere Datei mit dem bezeichneten Namen. Im Fehlerfall wird der Prozeß über die Prozedur *fehler_mod* beendet.

open_f

Diese Prozedur erlaubt es, eine bestehende Datei zu eröffnen. Beim Open-Aufruf wird auch das Zugriffsrecht auf die jeweilige Datei festgelegt. Intern wird der DOS-INT 21-Funktionsaufruf 3DH (Open File) benutzt. DOS ordnet dem übergebenen Dateinamen dann einen Handlecode zu. Dieser Handlecode wird von der Prozedur als Parameter zurückgegeben. Im Fehlerfall wird der Prozeß über die Prozedur *fehler_mod* beendet.

close_f

Diese Prozedur erlaubt es, eine geöffnete Datei/Einheit zu schließen. Intern wird die INT 21-Funktion 3EH (Close Device) verwendet. Als Parameter ist der jeweilige Handlecode zu übergeben. Nach einem Close-Aufruf ist der Handlecode nicht mehr definiert. Der Versuch eines weiteren Zugriffs auf die Einheit unter Verwendung des Handles resultiert in einer Fehlermeldung.

Hauptmodul

Im Hauptmodul wird die I/O-Umleitung über Handles demonstriert. Nachdem der Ausgabetext sowie der Dateiname als ASCIIZ-Strings definiert wurden, erfolgt die Ausgabe an verschiedene Einheiten. Dies wird durch Aufruf der Prozedur write_f vorgenommen, wobei jeweils der Wert des Handlecodes verändert wird. Während die Werte für die Standardeinheiten (Screen = 1, Printer = 4) als Konstanten festgelegt sind, ermittelt der *open_f*-Aufruf den Filehandle. Die wenigen Zeilen des Hauptmoduls zeigen, wie mächtig die handleorientierten DOS-Funktionen sind. Bei Verwendung von FCB-Aufrufen wäre für jede Ausgabe die komplette Verwaltung eines File-Control-Blocks mit Open, Write und Close erforderlich gewesen. Zusätzlich lassen sich Zugriffe auf Unterverzeichnisse nicht ausführen.

```
program handle_demo;

{*****************************************************************}
{* Filename:    .i.HANDLE.PAS;                                   *}
{* Autor:       Born G.                                          *}
{* Prog. Spr.:  Quick Pascal/Turbo Pascal                         *}
{* Betr. Sys.:  MSDOS 3.x                                        *}
{*                                                               *}
{* Das Programm demonstriert die Benutzung der Handles und den  *}
{* Umgang mit den MS-DOS-INT 21-Funktionen. Es gibt einen Text  *}
{* auf dem Bildschirm, dem Drucker und in eine Datei aus.       *}
{* Weiterhin wird der Umgang mit den Extended Errorcodes in DOS *}
{* gezeigt sowie der Programmabbruch mit Terminate a Process.   *}
{*                                                               *}
{* Die Turbo 4.0 spezifischen Anweisungen sind mit ### markiert *}
{*****************************************************************}

{ ********** Declaration Modul ***************** }

{$R-}    {### Range checking off}
{$B+}    {### Boolean complete evaluation on}
{$S+}    {### Stack checking on}
{$I+}    {### I/O checking on}
{$N-}    {### No numeric coprocessor}
{$M 65500,16384,655360} {### Turbo 3 default stack and heap}


Uses Dos;   { ###  DOS Unit für Turbo Pascal 4 }

type registers = record
            AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS: WORD; {###}
     end;

     F_string = string[80];         {* Typ Textstring           *}

const Screen_handle = 1;            {* Handle für Bildschirm    *}
      Printer_handle = 4;           {* Handle für Drucker       *}
      F_attribut     = 0;           {* Dateiattribut normal     *}
      F_access       = 2;           {* Dateizugriff read/write  *}


var
    reg          : registers;
    text         : F_String;
    File_name    : F_string;
    File_handle  : Integer;

{ ***********   Hilfs Module    ***************** }

procedure fehler_mod (error: integer);

{ ***       Fehlermodul                                    *** }
{ *                                                          * }
{ * Es wird eine Fehlermeldung mit dem INT 21-Fehler-       * }
{ * code auf dem Bildschirm ausgegeben. Anschließend        * }
{ * wird der erweiterte DOS-Fehlercode über einen           * }
{ * INT 21-Aufruf ermittelt und ausgegeben. Dann wird       * }
{ * das Programm beendet.                                   * }
{ ********************************************************** }

begin
 writeln ('DOS - Fehler (AX) = ', error);
 with reg do
  begin                             {* ermittle den Extended    *}
   ax := $5900;                     {* Error Code               *}
   bx := 0;
   msdos (Dos.Registers(reg));      {* Aufruf INT 21-Funktion   *}

   writeln ('Extended Error Codes :');
   writeln ('Fehler        [AX] :', ax);
   writeln ('Fehlerklasse  [BH] :', bx div $FF);
   writeln ('Aktion        [BL] :', bx mod $FF);
   writeln ('Fehlerhinweis [CH] :', CX div $FF);

   ax := $4C00;                     {* Code Terminate a process  *}
   msdos (Dos.Registers(reg));      {* Aufruf INT 21-Terminat    *}
  end;
end; {* fehler_mod *}

procedure write_f (handle: integer; var text: F_string);
```

```
{ *** INT 21-Funktion 40H (Write to a Device or File) *** }
{ *                                                     * }
{ * Es wird der in "text" übergebene String an die mit  * }
{ * dem Handle spezifizierte Einheit ausgegeben.        * }
{ *                                                     * }
{ *  Parameter der INT 21-Funktion 40H                  * }
{ *  AX     --> Funktionscode 40H                       * }
{ *  DS:DX  --> Zeiger auf Stringadresse                * }
{ *  CX     --> Stringlänge                             * }
{ *  BX     --> Handle                                  * }
{ *                                                     * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag     * }
{ *  gesetzt und in AX findet sich ein Fehlercode.      * }
{ ******************************************************* }

begin
 with reg do
  begin
   ds := Seg(Text);                 {* Setze Adresse des Text-  *}
   dx := Ofs(Text)+1;               {* strings in DS:DX         *}
   ax := $4000;                     {* Write to a Device Code   *}
   bx := handle;                    {* setze Handle             *}
   cx := length(Text);              {* Länge String             *}
   MSDOS (Dos.Registers(reg));      {* Aufruf INT 21-Funktion   *}
   if (flags and 01) > 0 then
    Fehler_mod (ax);                {* Fehlerausgang            *}
  end;
end; {* write_f *}

procedure create_f (f_attribut: integer; var f_name: F_string);

{ *** INT 21-Funktion 3CH (Create a File)            *** }
{ *                                                     * }
{ * Es wird die in "F-name" übergebene Datei mit dem    * }
{ * spezifizierten Attribut angelegt.                   * }
{ *                                                     * }
{ *  Parameter der INT 21-Funktion 3CH                  * }
{ *  AX     --> Funktionscode 3CH                       * }
{ *  DS:DX  --> Zeiger auf Stringadresse (Filename)     * }
{ *  CX     --> Attribut                                * }
{ *                                                     * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag     * }
{ *  gesetzt und in AX findet sich ein Fehlercode.      * }
{ ******************************************************* }

begin
 with reg do
  begin
    ds := Seg(F_name);               {* Setze Adresse DS:DX      *}
    dx := Ofs(F_name)+1;             {* auf String mit Filename  *}
    ax := $3C00;                     {* Create File Code         *}
    cx := f_attribut;                {* Fileattribut setzen      *}
    MSDOS (Dos.Registers(reg));      {* Aufruf INT 21-Funktion   *}

    if (flags and 01) > 0 then
     Fehler_mod (ax);               {* Fehlerausgang            *}
  end;
end; {* create_f *}


procedure open_f (f_access: integer; var f_name: F_string;
                  var handle: integer);

{ *** INT 21-Funktion 3DH (Open File)                *** }
{ *                                                     * }
{ * Es wird die in "F-name" übergebene Datei mit dem    * }
{ * spezifizierten Accesscode geöffnet.                 * }
{ *                                                     * }
{ *  Parameter der INT 21-Funktion 3DH                  * }
{ *  AH     --> Funktionscode 3DH                       * }
{ *  DS:DX  --> Zeiger auf Stringadresse                * }
{ *  AL     --> Accesscode                              * }
{ *                                                     * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag     * }
{ *  gesetzt und in AX findet sich ein Fehlercode.      * }
{ ******************************************************* }

begin
 with reg do
```

```
  begin
    ds := Seg(F_name);                {* Setze Adresse DS:DX      *}
    dx := Ofs(F_name)+1;              {* auf String mit Filename  *}
    ax := $3D00 + F_access;           {* Open + Accesscode        *}
    MSDOS (Dos.Registers(reg));       {* Aufruf INT 21-Funktion   *}

    handle := ax;                     {* merke Handle             *}
    if (flags and 01) > 0 then
     Fehler_mod (ax);                 {* Fehlerexit               *}
  end;
end; {* open_f *}


procedure close_f (handle: integer);

{ *** INT 21-Funktion 3EH (Close Device)              *** }
{ *                                                     * }
{ * Es wird die in Handle spezifizierte Einheit         * }
{ * geschlossen.                                       * }
{ *                                                     * }
{ *  Parameter der INT 21-Funktion 3EH                  * }
{ *  AX     --> Funktionscode 3EH                       * }
{ *  BX     --> Handle                                  * }
{ *                                                     * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag     * }
{ *  gesetzt und in AX findet sich ein Fehlercode.      * }
{ ******************************************************* }

begin
 with reg do
  begin
    ax := $3E00;                      {* Close  File Code         *}
    bx := handle;                     {* Handle                   *}
    MSDOS (Dos.Registers(reg));       {* Aufruf INT 21-Funktion   *}

    if (flags and 01) > 0 then
     Fehler_mod (ax);                 {* Fehlerexit               *}
  end;
end; {* close_f *}


{ ***********           Root  Modul               ********** }

begin
{*       Vorbereitung der Textstrings           *}

 Text := 'Teststring zur Ausgabe an verschiedene Einheiten';
 Text := Text + chr($0D) + chr($0A) + chr($00);
 File_name := 'TEST1.DAT' + chr(00);

 writeln ('Ausgabe auf den Bildschirm');  {* sende Text an Screen *}
 write_f (Screen_handle, text);

 writeln ('Ausgabe auf den Drucker');     {* sende Text an Printer*}
 write_f (Printer_handle, text);

 writeln ('Ausgabe in eine Datei');       {* sende Text in Datei  *}

 create_f (F_attribut, File_name);        {* create Datei         *}
 open_f (F_access, File_name, File_handle); {* open Datei         *}

 write_f (File_handle, text);             {* sende an Datei       *}

 close_f (File_handle);                   {* close Datei          *}

end
```

11.4 Das DOS-20-File-Problem

Die DOS-Versionen 2.x bis 3.2 erlauben nur einen Betrieb mit insgesamt 20 gleichzeitig geöffneten Dateien/Einheiten. In CONFIG.SYS läßt sich zwar der Befehl

```
FILES = n
```

auch mit Werten für n größer 20 eintragen. Es können aber trotzdem nur maximal 20 Einheiten gleichzeitig verwaltet werden. Dies liegt daran, daß DOS die Informationen über die geöffneten Einheiten eines Prozesses in dessen Program-Segment-Prefix verwaltet. Dort ist jedoch nur Platz für 20 Einträge zu je 8 Bit vorgesehen. Die Tabelle findet sich im undokumentierten Teil des PSP ab Offset 18H.

Die ersten fünf Einträge werden bereits bei der Installation des Prozesses standardmäßig belegt. Hier öffnet DOS die Einheiten für die Zeichen-Ein-/Ausgabe. Dabei werden folgende Handlecodes in die Tabelle eingetragen:

```
Code ° Einheit
-----é-----------------
0000 ° Input device
0001 ° Output device
0002 ° Error device
0003 ° Auxiliary device
0004 ° Printer device
```

Unbelegte Einträge werden mit dem Wert FFH markiert.

Nun soll aber noch auf eine der Merkwürdigkeiten von DOS in bezug auf die Handles eingegangen werden. DOS belegt nur die ersten 5 Byte für die Standardeinheiten. Andererseits gibt die Funktion 3DH (Open Handle) einen 16-Bit-Wert zurück. Dies ist scheinbar ein Widerspruch, da ja dann 10 Byte belegt sein sollten. Die Lösung gehört zu den (undokumentierten) Mysterien von DOS. Mit der Funktion 3DH wird kein DOS-Handle zurückgegeben, auch wenn die Dokumentation dies behauptet. Der durch die Funktion 3DH im Register BX zurückgegebene Wert wird deshalb nachfolgend als »virtueller Handle**Fehler! Verweisquelle konnte nicht gefunden werden.**realen Handlejeden Eintrag werden intern mindestens 39 Byte belegt. Damit ist ein 16-Bit-Offset erforderlich, um die Tabelle zu adressieren. Die Funktion 3DH legt also bei Dateien einen 16-Bit-Handlecode an.

Der »virtuelle Handlecodebelle eines neu installierten Prozesses:

```
CS:0018  01 01 01 00 02 FF FF FF FF FF FF FF FF FF FF FF
```

Der vierte Eintrag enthält den Wert 00, was dem realen DOS-Handle für die Auxiliary-Einheit entspricht. DOS reserviert die ersten drei realen Handlecodes gemäß folgender Aufstellung:

```
Handle ° Einheit
-------é-----------------
00     ° Auxiliary Einheit
01     ° Console Einheit
02     ° Printer Einheit
```

Ein Blick auf den obigen Speicherdump offenbart, daß sich alle Ein-/Ausgaben sowie die Fehlermeldungen auf die Console beziehen. Lediglich Printer- und Auxiliary-Einheit besitzen andere Handlecodes.

Alle Aufrufe der Funktion 3DH (Open Handle) geben die virtuellen Handlecodes zurück und tragen die DOS-Handles in die Tabelle ein. Bezieht sich ein Aufruf auf zeichenorientierte Einheiten (LPT, COM1 etc.) wird in der Tabelle nur 1 Byte belegt. Damit lassen sich mit der Tabelle maximal 20 Einheiten verwalten.

Bei Dateien wurde zumindest bei DOS 3.x festgestellt, daß pro Aufruf der zurückgegebene Handle immer um 2 erhöht wird. Es werden also 2 Byte in der Handletabelle belegt. Der Offsetwert (virtueller Handle) zeigt auf das obere Byte des 16 Bit (real) Handles. Damit lassen sich innerhalb des Prozesses maximal sieben Dateien gleichzeitig öffnen. Dieser Sachverhalt ist in keiner Dokumentation beschrieben.

Wie läßt sich das Problem der Begrenzung auf 20 Handles umgehen? Hierfür existieren zwei Lösungen, die nachfolgend kurz skizziert werden.

Um in DOS mehr als 20 Dateien gleichzeitig zu verwalten, ist der Befehl

```
FILES = xx
```

in der Datei CONFIG.SYS einzutragen. Wobei xx auf die Zahl der zu öffnenden Dateien zu setzen ist. Der Eintrag ist wichtig, da DOS beim Systemstart die Größe der internen Dateiverwaltungstabellen einstellt. Pro Datei vergrößert sich diese Tabelle um mindestens 39 Byte.

Nun besteht noch das Problem, daß ein laufender Prozeß maximal 20 Handles im PSP verwalten kann. Von diesen 20 Einträgen sind fünf bereits durch die Standard-Ein-/Ausgabe-einheiten belegt. Alle nicht benutzten Einträge sind mit dem Wert FFH belegt.

Eine Möglichkeit besteht darin, alle Handles zu belegen und dann den letzten Eintrag der Tabelle (Offset 13H) in einer Variablen zu sichern. Wird die Tabelle dann mit dem Wert FFH belegt, kann ein weiterer OPEN-Aufruf erfolgen, da DOS den Eintrag als frei erkennt. Nach dem Aufruf findet sich ab dem Offset nun ein neuer *Real Handle*, während als virtueller Handle die Zahl 13H zurückgegeben wird. So können mehrere Handles erzeugt und gesichert werden. Vor einem I/O-Funktionsaufruf ist dann der entsprechende Real-Handle in der Tabelle zu restaurieren. Die I/O-Funktion wird dann mit dem Wert 13H (virtuelles Handle) im Register BX aufgerufen.

Die zweite Möglichkeit besteht darin, eine entsprechend große neue Handletabelle durch den Prozeß anlegen zu lassen. Dann müssen alle bereits eingetragenen DOS-Handlescodes aus dem PSP-Bereich in die neue Tabelle kopiert werden. Nicht belegte Einträge in der neuen Tabelle sind mit dem Wert FFH zu initialisieren. Nun muß DOS nur noch darüber informiert werden, daß die neue Handletabelle zu benutzen ist. Die Informationen über die Lage und Größe der Tabelle werden im undokumentierten PSP-Bereich verwaltet. Die entsprechenden Adressen sind in Tabelle 11.5 aufgeführt.

Offset	Bemerkung
18	Handletabelle mit 20 Byte
32	Größe der Handletabelle (def. 20)
34	Offsetadresse Handletabelle
36	Segmentadresse Handletabelle

Tabelle 11.5: Adressen der Handletabelle im PSP

Ab Offset 32 (Word) ist die Größe der Handletabelle in Bytes abgelegt. Normalerweise findet sich hier der Wert 0014H, was insgesamt 20 Handleeinträgen entspricht. In dieses Wort ist nun die Größe der neuen Handletabelle einzutragen. Daran schließt sich ein

DWORD (Offset 34H) mit der Adresse der Handletabelle an. Hier findet sich die 4-Byte-Adresse der alten Handletabelle. Der laufende Prozeß muß den Adreßvektor der neuen Handletabelle ab Offset 34H eintragen. Damit greift DOS anschließend auf die neue Handletabelle zu. Anschließend lassen sich mehr als 20 Handles verwalten, sofern die Tabelle genügend groß ist. Es ist aber daran zu denken, daß pro eröffneter Datei jeweils 2 Byte belegt werden.

Ab DOS 3.3 existiert eine neue Funktion 67H (Set Handles Count) des INT 21, die es erlaubt, bis zu 65 534 Handles zu verwalten. DOS legt, ähnlich wie oben beschrieben, eine erweiterte Handletabelle im freien Speicherbereich an. Falls die spezifizierte Handlezahl kleiner als die Zahl der bereits geöffneten Handles ist, bleibt der Aufruf unwirksam.

Das nachfolgende Listing zeigt, wie eine neue Handletabelle angelegt wird.

Die Funktionen *fehler_mod*, *create_f* und *open_f* wurden bereits im Abschnitt »Die Handle-Aufrufe**Fehler! Verweisquelle konnte nicht gefunden werden.**Das Hauptprogramm initialisiert erst die neue Handletabelle mit dem Wert FFH und setzt anschließend die 20 Einträge der alten Tabelle um. Dann wird die neue Adresse sowie die Tabellengröße im PSP-Bereich eingetragen.

Der Prozeß eröffnet zuerst die Einheiten »LPT1**Fehler! Verweisquelle konnte nicht gefunden werden.**COM1« Beim OPEN-Aufruf dürfen die Einheitennamen nicht mit einem Doppelpunkt (:) im Text abgeschlossen werden.»In einer Schleife werden dann 30 Dateien angelegt, eröffnet und mit einem Teststring beschrieben.

Einzelheiten sind dem nachfolgenden Listing zu entnehmen.

```
program table_demo;

{*************************************************************}
{* Filename:    TABLE.PAS                                    *}
{* Autor:       Born G.                                      *}
{* Prog. Spr.:  Quick Pascal/Turbo Pascal                     *}
{* Betr. Sys.:  MSDOS 3.x                                    *}
{*                                                           *}
{* Das Programm demonstriert die Benutzung von mehr als 20   *}
{* Handles pro Prozeß in MS-DOS. Es legt eine neue Handle-   *}
{* tabelle an und eröffnet 30 Dateien auf dem Medium. Jede   *}
{* eröffnete Ein-/Ausgabeeinheit belegt einen 8-Bit-Handle,  *}
{* während pro eröffneter Datei zwei Byte belegt werden.     *}
{* Im File CONFIG.SYS ist die Anweisung:                     *}
{*                                                           *}
{*          FILES = 70                                       *}
{*                                                           *}
{* einzutragen. Die Turbo Pascal 4.0 spezifischen Anweisungen*}
{* sind mit den Zeichen ### markiert.                        *}
{*************************************************************}

{ *********** Declaration Modul ***************** }

{$R-}    {### Range checking off}
{$B+}    {### Boolean complete evaluation on}
{$S+}    {### Stack checking on}
{$I+}    {### I/O checking on}
{$N-}    {### No numeric coprocessor}
{$M 65500,16384,655360} {### Turbo 3 default stack and heap}


Uses Dos;   {### }

type registers = record          { ### }
           AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS: WORD;
     end;

     F_string = string[80];         {* Typ Textstring           *}

const F_attribut     = 0;           {* Dateiattribut normal     *}
      F_access       = 2;           {* Dateizugriff read/write  *}
      Table_size    = 70;           {* Größe Handletabelle      *}
      Max_files     = 30;           {* max. offene Files        *}

var
    reg          : registers;
    adr          : word;            {* ### *}
    text         : F_string;
    tmp          : String[2];
    handle_table : array [1 .. Table_size] of Byte;
    File_name    : F_string;
    File_handle  : array [1 .. Table_size] of Integer;
    Com_handle   : Integer;
    Lpt_handle   : Integer;
    i            : Integer;

{ ***********   Hilfs Module    *************** }

procedure fehler_mod (error: integer);

{ ***       Fehlermodul                                  *** }
{ *                                                        * }
{ * Es wird eine Fehlermeldung mit dem INT 21-Fehler-    * }
{ * code auf dem Bildschirm ausgegeben. Anschließend     * }
{ * wird der erweiterte DOS-Fehlercode über einen        * }
{ * INT 21-Aufruf ermittelt und ausgegeben. Dann wird    * }
{ * das Programm beendet.                                * }
{ ******************************************************** }

begin
 writeln ('DOS - Fehler (AX) = ', error);
 with reg do
  begin                             {* ermittle den Extended    *}
   ax := $5900;                     {* Error Code               *}
   bx := 0;
   msdos (Dos.Registers(reg));      {* Aufruf INT 21-Funktion   *}

   writeln ('Extended Error Codes :');
   writeln ('Fehler        [AX] :', ax);
   writeln ('Fehlerklasse  [BH] :', bx div $FF);
```

```
   writeln ('Aktion        [BL] :', bx mod $FF);
   writeln ('Fehlerhinweis [CH] :', CX div $FF);

   ax := $4C00;                     {* Code Terminate a process  *}
   msdos (Dos.Registers(reg));      {* Aufruf INT 21-Terminat    *}
  end;
end; {* fehler_mod *}

procedure write_f (handle: integer; var text: F_string);

{ *** INT 21-Funktion 40H (Write to a Device or File) *** }
{ *                                                      * }
{ * Es wird der in "text" übergebene String an die mit  * }
{ * dem Handle spezifizierte Einheit ausgegeben.        * }
{ *                                                      * }
{ *  Parameter der INT 21-Funktion 40H                   * }
{ *  AX     --> Funktionscode 40H                        * }
{ *  DS:DX  --> Zeiger auf Stringadresse                 * }
{ *  CX     --> Stringlänge                              * }
{ *  BX     --> Handle                                   * }
{ *                                                      * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag      * }
{ *  gesetzt und in AX findet sich ein Fehlercode.       * }
{ ****************************************************** }

begin
 with reg do
  begin
   ds := Seg(Text);                 {* Setze Adresse des Text-  *}
   dx := Ofs(Text)+1;               {* strings in DS:DX         *}
   ax := $4000;                     {* Write to a Device Code   *}
   bx := handle;                    {* setze Handle             *}
   cx := length(Text);              {* Länge String             *}
   MSDOS (Dos.Registers(reg));      {* Aufruf INT 21-Funktion   *}
   if (flags and 01) > 0 then
    Fehler_mod (ax);                {* Fehlerausgang            *}
  end;
end; {* write_f *}

procedure create_f (f_attribut: integer; var f_name: F_string);

{ *** INT 21-Funktion 3CH (Create a File)             *** }
{ *                                                      * }
{ * Es wird die in "F-name" übergebene Datei mit dem    * }
{ * spezifizierten Attribut angelegt.                   * }
{ *                                                      * }
{ *  Parameter der INT 21-Funktion 3CH                   * }
{ *  AX     --> Funktionscode 3CH                        * }
{ *  DS:DX  --> Zeiger auf Stringadresse (Filename)      * }
{ *  CX     --> Attribut                                 * }
{ *                                                      * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag      * }
{ *  gesetzt und in AX findet sich ein Fehlercode.       * }
{ ****************************************************** }

begin
 with reg do
  begin
    ds := Seg(F_name);              {* Setze Adresse DS:DX      *}
    dx := Ofs(F_name)+1;            {* auf String mit Filename  *}
    ax := $3C00;                    {* Create File Code         *}
    cx := f_attribut;               {* Fileattribut setzen      *}
    MSDOS (Dos.Registers(reg));     {* Aufruf INT 21-Funktion   *}

    if (flags and 01) > 0 then
     Fehler_mod (ax);               {* Fehlerausgang            *}
  end;
end; {* create_f *}


procedure open_f (f_access: integer; var f_name: F_string;
                  var handle: integer);

{ *** INT 21-Funktion 3DH (Open File)                 *** }
{ *                                                      * }
{ * Es wird die in "F-name" übergebene Datei mit dem    * }
{ * spezifizierten Accesscode geöffnet.                 * }
{ *                                                      * }
{ *  Parameter der INT 21-Funktion 3DH                   * }
{ *  AH     --> Funktionscode 3DH                        * }
```

```
{ *  DS:DX  --> Zeiger auf Stringadresse               * }
{ *  AL     --> Accesscode                            * }
{ *                                                   * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag   * }
{ *  gesetzt und in AX findet sich ein Fehlercode.    * }
{ ************************************************** }

begin
 with reg do
  begin
    ds := Seg(F_name);                {* Setze Adresse DS:DX      *}
    dx := Ofs(F_name)+1;              {* auf String mit Filename  *}
    ax := $3D00 + F_access;           {* Open + Accesscode        *}
    MSDOS (Dos.Registers(reg));       {* Aufruf INT 21-Funktion   *}

    handle := ax;                     {* merke Handle             *}
    if (flags and 01) > 0 then
     Fehler_mod (ax);                 {* Fehlerexit               *}
  end;
end; {* open_f *}


procedure close_f (handle: integer);

{ *** INT 21-Funktion 3EH (Close Device)            *** }
{ *                                                   * }
{ * Es wird die in Handle spezifizierte Einheit      * }
{ * geschlossen.                                     * }
{ *                                                   * }
{ *  Parameter der INT 21-Funktion 3EH                * }
{ *  AX     --> Funktionscode 3EH                     * }
{ *  BX     --> Handle                                * }
{ *                                                   * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag   * }
{ *  gesetzt und in AX findet sich ein Fehlercode.    * }
{ ************************************************** }

begin
 with reg do
  begin
    ax := $3E00;                      {* Close  File Code         *}
    bx := handle;                     {* Handle                   *}
    MSDOS (Dos.Registers(reg));       {* Aufruf INT 21-Funktion   *}

    if (flags and 01) > 0 then
     Fehler_mod (ax);                 {* Fehlerexit               *}
  end;
end; {* close_f *}


{ ***********          Root  Modul              ********** }

begin

{*       Vorbereitung des Teststrings              *}

 Text := 'Teststring zur Ausgabe an verschiedene Einheiten';
 Text := Text + chr($0D) + chr($0A) + chr($00);

{*       Vorbereitung der Handletabelle            *}

 for i := 1 to table_size do               {* init Handle Table  *}
  handle_table[i] := $FF;

 with reg do                               {* ermittle Adr. PSP  *}
  begin
    ax := $6200;                      {* Get PSP Adresse (DOS 3.x *}
    MSDOS (Dos.Registers(reg));       {* Aufruf INT 21-Funktion   *}
    adr := bx;                        {* Segmentadresse PSP       *}
  end;

 for i := 1 to 20 do
  handle_table[i] := Mem [adr:$17+i];      {* copy old Handles    *}

 Memw[adr:$32] := Table_size;              {* set Handlecount     *}
 Memw[adr:$34] := Ofs(Handle_table[1]);    {* Offset new Table    *}
 Memw[adr:$36] := Seg(Handle_table[1]);    {* Segment new Table   *}

 writeln;
 writeln ('Eröffnung der 32 Handles');     {* sende Text an Screen *}
```

```
 writeln;

 File_name := 'COM1' + chr(00);            {* eröffne COM1         *}
 open_f(F_access, File_name,Com_handle);

 File_name := 'LPT1' + chr(00);            {* eröffne LPT1         *}
 open_f(F_access,File_name,Lpt_handle);

 for i := 1 to Max_files do                {* eröffne Dateien      *}
  begin
   STR(i,tmp);                             {* Filenummer in ASCII  *}
   File_name := 'TEST' + tmp + '.DAT' + CHR (00);
   writeln('Eröffne : ',File_name);
   create_f (F_attribut, File_name);       {*  create Datei        *}
   open_f (F_access, File_name, File_handle[i]); {* open Datei     *}
   write_f (File_handle[i], text);         {*  sende an Datei      *}
  end;

 writeln ('Nun sind alle Dateien geöffnet');

 for i := 1 to Max_files do
  close_f (File_handle[i]);                {* close Datei          *}
 close_f (Com_handle);
 close_f (Lpt_handle);

end.
```

11.5 Commit File

Ab DOS 3.3 existiert der INT 21-Aufruf 68H (Commit File). Dieser Aufruf erzwingt eine Auslagerung aller Zwischenpuffer auf die Speichermedien. DOS schreibt ja alle Ausgabedaten nur in einen Ausgabepuffer. Dieser wird erst auf die Platte ausgelagert, wenn er voll ist. Weiterhin existiert ein zweiter Puffer, in dem die Directory-Daten abgelegt werden. Ein OPEN-File-Aufruf liest die Daten der FAT und das Directory in den Puffer ein. Modifikationen innerhalb der Datei (Vergrößerung der Länge etc.) werden dann in diesem Puffer vermerkt. Dieser Puffer wird erst beim CLOSE-File-Aufruf auf das Speichermedium ausgelagert.

Wird aus irgend einem Grund kein CLOSE-File-Aufruf durchgeführt, dann sind zwar die modifizierten Daten auf dem Medium vorhanden. Falls sich die Dateigröße aber geändert hat, tritt ein inkonsistenter Zustand auf. Da der Puffer mit den geänderten Directory- und FAT-Einträgen nicht ausgelagert wurde, besitzt Directory noch den alten Zustand. Bei einer Vergrößerung der Datei gehen dann die neu belegten Cluster verloren, während bei einer verkleinerten Datei die gelöschten Werte weiter vorhanden sind, da noch die alte Dateilänge in der FAT steht.

Bei vielen Applikationen (Textverarbeitungsprogramme, Datenbanken etc.) ist es jedoch erwünscht, die Daten periodisch zu sichern. Aus obigen Ausführungen wird klar, daß hierfür auch die Einträge innerhalb der Directories und der FAT modifiziert werden müssen.

In älteren DOS-Versionen wird dies jedoch nur beim CLOSE-Aufruf veranlaßt. Falls eine Applikation die Pufferinhalte sichern will (oder muß), bleibt nur die Möglichkeit, eine CLOSE-File-Anweisung auszuführen. Anschließend muß die Einheit mit einem weiteren OPEN-Aufruf wieder geöffnet werden.

Unglücklicherweise benötigt die OPEN-File-Funktion recht lange Ausführungszeiten, was sich in der Laufzeit der Programme bemerkbar macht.

Ein weiteres Problem tritt in Netzwerken auf, falls zwei Prozesse um den Zugriff zu einer Datei konkurrieren. Nehmen wir an, ein Prozeß besitzt das ausschließliche Zugriffsrecht auf diese Datei, während ein zweiter Prozeß versucht, dieses Recht zu erlangen. Der zweite Prozeß erhält aber kein Zugriffsrecht, da die Datei mit einem File-Lock gesperrt ist. Falls nun der erste Prozeß die Puffer mit der CLOSE-OPEN-Sequenz auslagert, wird die Zugriffssperre nach dem CLOSE-File-Befehl aufgehoben. Der wartende Prozeß erlangt nun Zugriff auf die gerade geschlossene Datei und sperrt diese seinerseits für Zugriffe anderer Prozesse. Sobald der erste Prozeß nun den OPEN-Befehl absetzt, wird dieser abgewiesen, da ja die Datei bereits belegt ist. Damit muß dieser Prozeß warten, bis der zweite Prozeß den Zugriff auf die Datei wieder freigibt. Dies kann in der Praxis zu erheblichen Problemen führen.

DOS bietet aber eine Hintertür, um die COMMIT-File-Funktion der Version 3.3 auch in älteren Versionen zu emulieren. Ab DOS 2.0 existiert der INT 21-Aufruf 45H (Duplicate Handle). Diese Funktion erzeugt einen zusätzlichen Handle für eine Datei, die bereits eröffnet wurde, also ein Handle besitzt. Aus der Handletabelle wird dann mindestens ein Eintrag belegt. Der zweite Handle läßt sich nun verwenden, um die Datei zu schließen. Damit werden auch die Puffer zwangsweise auf das Medium ausgelagert. Da aber noch der erste Handle existiert, kann der Prozeß die Datei ohne einen zusätzlichen OPEN-Aufruf weiter bearbeiten. Das nachfolgende Programm demonstriert die Verwendung dieser Funktion.

```
program commit_demo;

{*****************************************************************}
{* Filename:    COMMIT.PAS                                       *}
{* Autor:       Born G.                                          *}
{* Prog. Spr.:  Quick Pascal/Turbo Pascal                         *}
{* Betr. Sys.:  MSDOS 3.x                                        *}
{*                                                               *}
{* Das Programm demonstriert die Emulation der Commit-File-      *}
{* Funktion über die DUP-Funktion (45H). Damit kann die Aus-     *}
{* lagerung der DOS-Puffer auf das Medium erzwungen werden.      *}
{*                                                               *}
{* Die Turbo Pascal 4.0 spezifischen Anweisungen wurden mit      *}
{* den Zeichen ### markiert.                                     *}
{*****************************************************************}

{ ********** Declaration Modul ***************** }

{$R-}    {### Range checking off}
{$B+}    {### Boolean complete evaluation on}
{$S+}    {### Stack checking on}
{$I+}    {### I/O checking on}
{$N-}    {### No numeric coprocessor}
{$M 65500,16384,655360} {### Turbo 3 default stack and heap}


Uses Dos;  { ### }

type registers = record            { ### }
           AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS: WORD;
     end;

      F_string = string[80];       {* Typ Textstring          *}

const F_attribut     = 0;          {* Dateiattribut normal    *}
      F_access       = 2;          {* Dateizugriff read/write *}

var
    reg          : registers;
    text         : F_string;
    File_name    : F_string;
    File_handle1 : Integer;
    File_handle2 : Integer;

{ ***********   Hilfs Module    ***************** }

procedure fehler_mod (error: integer);

{ ***       Fehlermodul                                  *** }
{ *                                                        * }
{ * Es wird eine Fehlermeldung mit dem INT 21-Fehler-    * }
{ * code auf dem Bildschirm ausgegeben. Anschließend     * }
{ * wird der erweiterte DOS Fehlercode über einen        * }
{ * INT 21-Aufruf ermittelt und ausgegeben. Dann wird    * }
{ * das Programm beendet.                                * }
{ ******************************************************** }

begin
 writeln ('DOS - Fehler (AX) = ', error);
 with reg do
  begin                            {* ermittle den Extended    *}
   ax := $5900;                    {* Error Code               *}
   bx := 0;
   msdos (Dos.Registers(reg));     {* Aufruf INT 21-Funktion   *}

   writeln ('Extended Error Codes :');
   writeln ('Fehler        [AX] :', ax);
   writeln ('Fehlerklasse  [BH] :', bx div $FF);
   writeln ('Aktion        [BL] :', bx mod $FF);
   writeln ('Fehlerhinweis [CH] :', CX div $FF);

   ax := $4C00;                    {* Code Terminate a process  *}
   msdos (Dos.Registers(reg));     {* Aufruf INT 21-Terminate   *}
  end;
end; {* fehler_mod *}

procedure write_f (handle: integer; var text: F_string);

{ *** INT 21-Funktion 40H (Write to a Device or File) *** }
{ *                                                      * }
```

```
{ * Es wird der in "text" übergebene String an die mit  * }
{ * dem Handle spezifizierte Einheit ausgegeben.        * }
{ *                                                     * }
{ *  Parameter der INT 21-Funktion 40H                  * }
{ *  AX     --> Funktionscode 40H                       * }
{ *  DS:DX  --> Zeiger auf Stringadresse                * }
{ *  CX     --> Stringlänge                             * }
{ *  BX     --> Handle                                  * }
{ *                                                     * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag     * }
{ *  gesetzt und in AX findet sich ein Fehlercode.      * }
{ ******************************************************* }

begin
 with reg do
  begin
   ds := Seg(Text);                 {* Setze Adresse des Text-  *}
   dx := Ofs(Text)+1;               {* strings in DS:DX         *}
   ax := $4000;                     {* Write to a Device Code   *}
   bx := handle;                    {* setze Handle             *}
   cx := length(Text);              {* Länge String             *}
   MSDOS (Dos.Registers(reg));      {* Aufruf INT 21-Funktion   *}
   if (flags and 01) > 0 then
    Fehler_mod (ax);                {* Fehlerausgang            *}
  end;
end; {* write_f *}

procedure create_f (f_attribut: integer; var f_name: F_string);

{ *** INT 21-Funktion 3CH (Create a File)             *** }
{ *                                                     * }
{ * Es wird die in "F-name" übergebene Datei mit dem    * }
{ * spezifizierten Attribut angelegt.                   * }
{ *                                                     * }
{ *  Parameter der INT 21-Funktion 3CH                  * }
{ *  AX     --> Funktionscode 3CH                       * }
{ *  DS:DX  --> Zeiger auf Stringadresse (Filename)     * }
{ *  CX     --> Attribut                                * }
{ *                                                     * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag     * }
{ *  gesetzt und in AX findet sich ein Fehlercode.      * }
{ ******************************************************* }

begin
 with reg do
  begin
    ds := Seg(F_name);              {* Setze Adresse DS:DX      *}
    dx := Ofs(F_name)+1;            {* auf String mit Filename  *}
    ax := $3C00;                    {* Create File Code         *}
    cx := f_attribut;               {* Fileattribut setzen      *}
    MSDOS (Dos.Registers(reg));     {* Aufruf INT 21-Funktion   *}

    if (flags and 01) > 0 then
     Fehler_mod (ax);               {* Fehlerausgang            *}
  end;
end; {* create_f *}


procedure open_f (f_access: integer; var f_name: F_string;
                  var handle: integer);

{ *** INT 21-Funktion 3DH (Open File)                 *** }
{ *                                                     * }
{ * Es wird die in "F-name" übergebene Datei mit dem    * }
{ * spezifizierten Accesscode geöffnet.                 * }
{ *                                                     * }
{ *  Parameter der INT 21-Funktion 3DH                  * }
{ *  AH     --> Funktionscode 3DH                       * }
{ *  DS:DX  --> Zeiger auf Stringadresse                * }
{ *  AL     --> Accesscode                              * }
{ *                                                     * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag     * }
{ *  gesetzt und in AX findet sich ein Fehlercode.      * }
{ ******************************************************* }

begin
 with reg do
  begin
    ds := Seg(F_name);              {* Setze Adresse DS:DX      *}
    dx := Ofs(F_name)+1;            {* auf String mit Filename  *}
```

```
    ax := $3D00 + F_access;           {* Open + Accesscode        *}
    MSDOS (Dos.Registers(reg));       {* Aufruf INT 21-Funktion   *}

    handle := ax;                     {* merke Handle             *}
    if (flags and 01) > 0 then
     Fehler_mod (ax);                 {* Fehlerexit               *}
  end;
end; {* open_f *}

procedure dup_handle (var Handle_1, Handle_2: integer);

{ *** INT 21-Funktion 45H (Duplicate Handle)          *** }
{ *                                                      * }
{ * Es wird der in Handle_1 übergebene Handle dupli-    * }
{ * ziert und in der Variablen Handle_2 zurückgegeben.  * }
{ *                                                      * }
{ *  Parameter der INT 21-Funktion 45H                   * }
{ *  AH     --> Funktionscode 45H                        * }
{ *  BX     --> Handle_1                                 * }
{ *                                                      * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag      * }
{ *  gesetzt und in AX findet sich ein Fehlercode.       * }
{ *  Andernfalls findet sich in AX der 2. Handle.        * }
{ ****************************************************** }

begin
 with reg do
  begin
    ax := $4500 + Handle_1;           {* Dup  + Handle_1 Code     *}
    MSDOS (Dos.Registers(reg));       {* Aufruf INT 21-Funktion   *}

    Handle_2 := ax;                   {* merke Handle_2           *}
    if (flags and 01) > 0 then
     Fehler_mod (ax);                 {* Fehlerexit               *}
  end;
end; {* dup_handle *}


procedure close_f (handle: integer);

{ *** INT 21-Funktion 3EH (Close Device)              *** }
{ *                                                      * }
{ * Es wird die in Handle spezifizierte Einheit        * }
{ * geschlossen.                                       * }
{ *                                                      * }
{ *  Parameter der INT 21-Funktion 3EH                   * }
{ *  AX     --> Funktionscode 3EH                        * }
{ *  BX     --> Handle                                   * }
{ *                                                      * }
{ *  Bei Fehlern ist nach dem Aufruf das Carry Flag      * }
{ *  gesetzt und in AX findet sich ein Fehlercode.       * }
{ ****************************************************** }

begin
 with reg do
  begin
    ax := $3E00;                      {* Close  File Code         *}
    bx := handle;                     {* Handle                   *}
    MSDOS (Dos.Registers(reg));       {* Aufruf INT 21-Funktion   *}

    if (flags and 01) > 0 then
     Fehler_mod (ax);                 {* Fehlerexit               *}
  end;
end; {* close_f *}


{ ***********           Root  Modul               *************** }

begin
{*                    Teststring                   *}

 Text := 'Teststring zur Ausgabe in die Datei Einheiten';
 Text := Text + chr($0D) + chr($0A) + chr($00);

 writeln;
 writeln ('Eröffnung der Datei');           {* sende Text an Screen *}
 writeln;

 File_name := 'TEST1.DAT' + CHR (00);
```

```
 create_f (F_attribut, File_name);          {* create Datei         *}
 open_f (F_access, File_name, File_handle1);  {* open Datei         *}

 write_f (File_handle1, text);              {* Schreibe in Datei    *}

 dup_handle (File_handle1,File_handle2);    {* dupliziere Handle    *}

 write_f (File_handle1, text);              {* schreibe in Datei    *}

 close_f (File_handle2);                    {* Commit Datei (close) *}

 write_f (File_handle1, text);              {* schreibe in Datei    *}

 close_f (File_handle1);                    {* close Datei          *}
end.
```

11.6 Tochterprozesse in DOS (die EXEC-Funktion)

MS-DOS bietet die Möglichkeit, daß ein laufender Prozeß weitere Unterprozesse erzeugt. Der erzeugende Prozeß wird als *Parent Process* und der kreierte Prozeß als *Child Process* bezeichnet.

Da in DOS immer nur ein Prozeß aktiv ist, geht bei der Erzeugung des Subprozesses die Kontrolle an diesen über. Der Parent-Process wird bis zur Beendigung des Tochterprozesses suspendiert.

Ab DOS 2.0 existiert ein INT 21-Funktionsaufruf (4BH Load or Execute a Programm) um Subprozesse zu erzeugen. Die DOS-EXEC-Funktion des Kommandoprozessors (COMMAND.COM) übernimmt dabei folgende Aufgaben:

- Erzeugung eines Environmentsegments mit allen Einstellungen des Masterenvironments
- Erzeugung eines Programm Segment Prefix
- Laden des Programmcodes in den Speicher
- Initialisierung der Segmentregister, des Stackpointers und des Instructionpointers
- Start des neuen Prozesses

Das Tochterenvironment ist eine Kopie des Masterenvironments (Environment des Parent Processes). Die Handletabelle des Tochterprozesses enthält eine Kopie aller Handlecodes des Vaterprozesses, deren Inherit Bit gesetzt ist. Damit werden alle Informationen, einschließlich der Dateizugriffsrechte, an den neuen Prozeß vererbt. Lediglich die Einträge für die Terminate- und Control-C-Exit-Adressen im PSP des neuen Prozesses enthalten die Adresse des Vaterprozesses. Dies ist erforderlich, da bei einem Abbruch des laufenden Prozesses die Kontrolle nicht an DOS, sondern an den Vaterprozeß zurückgeht.

Normalerweise braucht der Benutzer sich um die oben beschriebenen Einzelheiten nicht zu kümmern, da dies automatisch durch die EXEC-Funktion erfolgt. Allerdings enthält die DOS-INT 21-Funktion 4BH einige Fußangeln, so daß die Kenntnis der Zusammenhänge recht nützlich ist. Im Verlaufe dieses Abschnitts wird exemplarisch die Benutzung dieser Funktion beschrieben. Weitere Informationen finden sich im Kapitel über die INT 21-Funktionen. Das Programm demonstriert den Aufruf von Subprozessen aus einem laufenden Prozeß heraus. Die Funktion der einzelnen Module soll nun kurz beschrieben werden.

release_mem

Das Demoprogramm liegt als COM-File vor. DOS besitzt nun aber die Eigenart, daß beim Laden eines COM-Programmes diesem der gesamte Speicher zugeordnet wird. Das bedeutet, daß vor der Erzeugung eines Subprozesses erst der nicht benötigte Speicherplatz an die DOS-Speicherverwaltung zurückzugeben ist. Dies erfolgt durch die Prozedur *release_mem*, die intern die DOS-INT-21-Funktion 4AH (Set Block) benutzt.

get_environment

Diese Prozedur analysiert das Environmentsegment des Vaterprozesses. Ziel ist es, die Lage des Kommandoprozessors zu ermitteln. Diese wird in MS-DOS durch den String

```
COMSPEC=/Pfad/COMMAND.COM<CR><00>
```

angezeigt. Falls kein solcher String gefunden wird, bricht der Prozeß mit einer Fehlermeldung ab. Andernfalls wird der String in die Variable *Command_Pfad* kopiert. Die Information über die Lage des Kommandointerpreters ist beim EXEC-Aufruf von Bedeutung.

EXEC

Diese Prozedur bildet das Interface zum Aufrufen der DOS-EXEC-Funktion. Intern wird der INT 21-Aufruf 4BH benutzt. Dieser besitzt folgende Aufrufparameter:

```
Ö----------Ú--------------------------------------Ì
º Register º Belegung                             º
û----------é--------------------------------------À
º   AX     º 4B00H (Funktionscode)                 º
º  DS:DX   º Zeiger auf einen ASCIIZ-String        º
º          º mit dem Programmnamen                 º
º  ES:BX   º Zeiger auf den Parameterblock         º
Û----------Ù--------------------------------------ì
```

Im Registerpaar DS:DX findet sich ein 4-Byte-Adreßvektor auf einen ASIIZ-String. In diesem String sind Laufwerk, Pfad und der Name des auszuführenden Programmes abgelegt.

Nun ist es naheliegend, in diesem String direkt den Namen des auszuführenden Programmes anzugeben. Dann ist aber der rufende Prozeß dafür verantwortlich, daß der richtige Zugriffspfad eingestellt ist, daß keine EXE-Dateien zu bearbeiten sind etc. Dies sind alles Aufgaben, die normalerweise vom DOS-Kommandoprozessor übernommen werden. Es bietet sich also an, diesen zu benutzen, um das gewünschte Programm zu laden.

DOS bietet die Möglichkeit, eine zweite Kopie des Kommandoprozessors (COMMAND.COM) im Speicher zu installieren. Dies erfolgt mit dem Aufruf:

```
<Laufwerk><Pfad>COMMAND.COM /P<CR>
```

Falls der Schalter /P gesetzt ist, wird die Kopie des Kommandoprozessors permanent im Speicher installiert. Andernfalls wird der Speicher nach dem Aufruf wieder zurückgegeben. Dieses Verfahren wird bei der EXEC-Funktion angewendet. Mittels der Prozedur *get_environment* wird der Name und der Pfad des Kommandoprozessors ermittelt und in der Variablen *Command_Pfad* gespeichert. Die Adresse dieser Variablen wird dann im

Register DS:DX übergeben. Es ist aber darauf zu achten, daß der Stringname mit einem Nullbyte abgeschlossen wird.

Weiterhin ist noch ein Parameterblock mit folgender Struktur aufzubauen:

Bytes	Belegung
2	Segmentadresse Environment String
4	Zeiger auf einen Kommandostring
4	Zeiger auf den FCB1
4	Zeiger auf den FCB2

Der genaue Aufbau findet sich in Tabelle 4.28. Diese Parameterstruktur wird in Pascal als Record mit folgender Struktur:

```
Param_Typ : Env_seg
            Kommando_Ofs
            Kommando_Seg
            FCB_Ofs1
            FCB_Seg1
            FCB_Ofs2
            FCB_Seg2
```

vereinbart und initialisiert. Im Feld *Env_seg* wird der Wert 0 eingetragen, was bedeutet, daß ein neues Environmentsegment zu generieren ist. Andernfalls ist hier die Segmentadresse des Environmentbereiches anzugeben. Es besteht damit die Möglichkeit, hier die Segmentadresse eines bereits angelegten Environments einzutragen.

In dem folgenden 4-Byte-Vektor ist die Adresse eines Kommandostrings zu plazieren. Dieser Kommandostring kann dann durch das geladene Programm ausgewertet werden, da der String im PSP des neuen Prozesses ab Offset 80H abgelegt wird. Da der Aufruf den Kommandointerpreter aktiviert, muß der Name des eigentlichen Programmes spezifiziert werden. Wird dem Kommandoprozessor ein Kommandostring übergeben, muß dieser folgendes Format besitzen:

```
<Länge>C/<Kommandostring><CR>
```

Die Länge bezieht sich auf den String ohne das <CR>-Zeichen am Ende der Zeile. Viele DOS-Versionen verkraften es aber auch, wenn im Feld *Länge* die Zahl aller Bytes im String steht. Als Kommandostring kann jeder gültige DOS-Befehl verwendet werden. In unserem Beispiel ist der String mit TREE.COM belegt.

In den beiden folgenden Feldern finden sich Adreßvektoren auf die File-Control-Blocks. Hier werden die Adressen der aktuellen FCBs aus dem PSP angegeben.

Damit ist im Grunde die Parameterversorgung der EXEC-Funktion (4B00H) geklärt. Nun kann die Funktion aufgerufen werden. Leider tritt das Problem auf, daß bei Aufruf die Inhalte des Stacksegments (SS) und des Stackpointers (SP) verändert werden. In manchen DOS-Versionen wird der Inhalt vorher gerettet, während andere Versionen undefinierte Werte zurückgeben. Die Dokumentation der Hersteller enthält in den wenigsten Fällen Informationen über diese Problematik. Um unabhängig von der jeweiligen DOS-Version zu sein, sind die Register vor Aufruf der Funktion 4BH in lokalen Variablen zu sichern und nach dem Funktionsaufruf zu restaurieren.

Als zweite Maßnahme ist das Direction-Flag zu löschen, da sonst manche DOS-Versionen unkorrekt arbeiten. Vermutlich wurden bei der Implementierung die

Stringbehandlungsinstruktionen des 8086 benutzt, ohne vorher das Direction-Flag zu initialisieren.

Das nachfolgende Listing zeigt alle Einzelheiten der Implementierung.

```
program exec_demo;

{*****************************************************************}
{* Filename:    EXEC.PAS                                         *}
{* Autor:       Born G.                                          *}
{* Prog. Spr.:  Turbo Pascal 3.0                                *}
{* Betr. Sys.:  MSDOS 3.x                                        *}
{*                                                               *}
{* Das Programm demonstriert die Benutzung der INT 21-EXEC -     *}
{* Funktion (4B00H Load and Execute a Programm). Weiterhin       *}
{* wird die Funktion 4AH benutzt, um die Speichergröße aus       *}
{* dem laufenden Prozeß heraus zu variieren.                     *}
{*                                                               *}
{* Bei der Übersetzung sind die Optionen:                        *}
{*                                                               *}
{*        COM File                                               *}
{*        Min free dynamic memory = stack_size                   *}
{*        Max free dynamic memory = stack_size                   *}
{*                                                               *}
{* des Compilers gesetzt werden. Stack_size ist eine Pro-        *}
{* grammkonstante und wird hier auf 200 gesetzt. Eine Umsetzung *}
{* auf Quick/Turbo Pascal 4.0 ist nicht sinnvoll, da der Compiler den *}
{* Aufruf von Tochterprozessen durch die EXECUTE Funktion im-    *}
{* plizit unterstützt.                                           *}
{*****************************************************************}

{ *********** Declaration Modul *****************}

type Long_string  = STRING[80];

     reg8086 = record
            AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS: INTEGER;
     end;


const Stack_size  = $200;           {* Größe des HEAP und Stack *}

var
    Command_pfad : Long_string;     {* Environment String       *}
    prg_name     : Long_string;     {* Name child process       *}

{ *********** Hilfsroutinen     *****************}

procedure Exec(ProgrammName: Long_string);
{
  Aufruf einer Kopie des MS-DOS-Kommandprozessors über die
  Funktion Int 21  AX = $4B00. Durch Übergabe eines neuen
  Programmnamens wird ein Tochterprozeß aktiviert und ausge-
  führt. Das Modul versorgt die Register vor Eintritt in MS-DOS,
  insbesondere werden SP und SS gesichert, da nicht alle DOS
  Versionen diese Register retten.
}
type                        {* definiere den Parameterblock als Record *}
    Param_Typ = RECORD
                  Env_seg, Kommando_Ofs, Kommando_Seg,
                  Fcb_Ofs1,Fcb_Seg1,Fcb_Ofs2,Fcb_Seg2 : integer;
                  END;
const
    ParameterBlock :              {* initialisiere den Parameterblock *}
      Param_Typ =
           (Env_seg     : 0;      {* übernehme Master Environment     *}
            Kommando_Ofs: 0;      {* Adresse Kommandostring, der ab   *}
            Kommando_Seg: 0;      {* 80H im neuen PSP abgelegt wird   *}
            Fcb_Ofs1    : $5C;    {* Adressen der zwei FCBs           *}
            Fcb_Seg1    : 0;
            Fcb_Ofs2    : $6C;
            Fcb_Seg2    : 0);

   Pfad_name      : Long_String ='';     {* clear Pfadname            *}
   SS_save        : integer = 0;         {* Speicher für Stacksegm.   *}
   SP_save        : integer = 0;         {* und Stackpointer          *}
   AX_R           : integer = 0;         {* Hilfsvariablen für die    *}
   Flag_R         : integer = 0;         {* 8086 Flags und AX         *}

var
  Parameter  : Long_string;             {* lokaler Str. mit Kommando *}

begin
  Pfad_name := Command_Pfad + chr(0);        {* Pfadangabe COMMAND    *}
  Parameter := '/c'+ ProgrammName + chr(13); {* Name des child proc.  *}
```

```
  Parameter[0] := chr(ord(Parameter[0])-1);  {* Kommandolänge ohne CR *}
  with ParameterBlock do
   begin
    Kommando_Seg := SEG(Parameter);          {* setze Adr. Environm.  *}
    Kommando_Ofs := OFS(Parameter);
    Fcb_seg1     := Cseg;                    {* setze FCB Adressen    *}
    Fcb_seg2     := Cseg;                    {* aktuelles Codesegment *}
   end;

 INLINE                       {****   Aufruf der EXEC Funktion     ****}
  ($1E/                       (* push ds        Register ds und bp    *)
   $55/                       (* push bp        auf Stack retten,     *)
   $2E/$89/$26/SP_save/       (* cs: mov StackZeiger,sp  dann SP und  *)
   $8C/$D0/                   (* mov ax,ss      SS sichern            *)
   $2E/$A3/SS_save/           (* cs: mov StackSegment,ax              *)
   $0E/                       (* push cs                              *)
   $0E/                       (* push cs                              *)
   $1F/                       (* pop ds         ds und es auf cs      *)
   $07/                       (* pop es         setzen                *)
   $BA/Pfad_name+1/           (* mov dx, Pfad_name+1                  *)
   $BB/ParameterBlock/        (* mov bx, ParameterBlock               *)
   $B8/$00/$4B/               (* mov ax, 4B00H                        *)
   $FA/                       (* cli            disable Interrupt     *)
   $FC/                       (* cld            clear Direction Flag  *)
   $CD/$21/                   (* int 21         EXEC  CALL            *)
   $FA/                       (* cli                                  *)
   $2E/$8B/$1E/SS_save/       (* cs: mov bx, StackSegment             *)
   $8E/$D3/                   (* mov ss, bx                           *)
   $2E/$8B/$26/SP_save/       (* cs: mov sp, StackZeiger              *)
   $FB/                       (* sti                                  *)
   $5D/                       (* pop bp       restauriere die Re-     *)
   $1F/                       (* pop ds       gister                  *)
   $2E/$A3/AX_R/              (* cs: mov AX_R, ax                     *)
   $9C/                       (* pushf                                *)
   $58/                       (* pop ax                               *)
   $2E/$A3/Flag_R);           (* mov Flag_R, ax                       *)

   if (Flag_R and 1) <> 0 then
    begin                                {* Fehlerflag gesetzt ? *}
     if not (Flag_R in [1,2,3,7,8,10,11]) then
      writeln ('Wert Register AX illegal:',AX_R)
     else
      begin                              {* Fehlercode ausgeben  *}
       writeln;
       case AX_R of
        1 : writeln('Fehler im Funktionsaufruf AL <> 0 oder 3');
        2 : writeln('Programm- oder Pfadname ungültig');
        3 : writeln('Pfadname ungültig');
        7 : writeln('Speicherkontrollblock zerstört');
        8 : writeln('Speicherkapazität zu gering.');
       10 : writeln('Falsches Environment.');
       11 : writeln('Fehler im .EXE File');
       end;
      end;
      halt;   { Programmabbruch }
    end;
end;  { Exec }

procedure get_environment;
{
  Diese Procedure analysiert das Environment des Vater-
  Prozesses nach der Spezifikation

       COMSPEC=/pfad/COMMAND.COM

  Die Segmentadresse des Vaterenvironment befindet sich an
  der Adresse 2CH des PSP. Jedes Set Command im Environment
  ist durch den Wert 0 abgeschlossen. Das Ende des Environ-
  mentsegments ist durch eine weitere 0 markiert.
}
var
   ofs,seg : integer;
   ch_ptr  : ^char;

begin
  ofs := 0;
  seg := memw[CSeg : $2C];               {* lese seg_adr. Env. *}
  ch_ptr := ptr (seg,ofs);               {* Zeiger auf Env.    *}

  REPEAT                                 {* scan Environment   *}
```

```
    Command_pfad := '';                    {* clear Pfadstring   *}
    WHILE ch_ptr^ <> #0 DO                 {* scan Set Command   *}
     BEGIN
       Command_pfad := Command_pfad + ch_ptr^;
       ofs := SUCC(ofs);
       ch_ptr := ptr(seg,ofs);
     END;
    ofs := SUCC(ofs);                      {* next char adress   *}
    ch_ptr := ptr (seg,ofs);
  UNTIL (COPY(Command_pfad,1,8) = 'COMSPEC=') OR
        (ch_ptr^ = #0);
  IF COPY(Command_pfad,1,8) <> 'COMSPEC=' THEN
   BEGIN                                   {* Pfad nicht gefunden *}
     WRITELN;
     WRITELN('Environment-Bereich fehlerhaft.');
     WRITELN('Programminstallation abgebrochen !');
     HALT;
   END;
  Command_pfad := COPY(Command_pfad,9,LENGTH(Command_pfad));
END; { get_environment }

procedure release_mem;
{
  Freigabe des nicht benötigten Speichers an MS-DOS
  mittels der MS-DOS-Funktion Set Block.
  Parameter der INT 21-Funktion 4AH

   AX    ---> 4A00H
   BX    ---> Zahl der benötigten Paragraphen
   ES    ---> Segmentadresse des Speicherblocks

  Bei Fehlern ist anschließend das Carry Flag gesetzt und
  im Register AX findet sich ein Fehlercode.
}

 var register  : reg8086;

 begin
  {
     In MS-DOS besitzt Turbo Pascal folgenden Speicheraufbau:

          Codesegment
          Datensegment
          Stacksegment

     Der verfügbare Speicher läßt sich damit zu

         Mem := Stacksegment - Codesegment + Stackgröße

     bestimmen. Die Funktion $4A00 erwartet die Speichergröße
     in Anzahl 16 Bit Paragraphen. Die Stackgröße in Byte wird
     beim Programmstart durch SP angezeigt.
 }
  with register do
   begin                                   {* Register setzen        *}
    BX := (Sseg - Cseg) + Stack_size;      {* Zahl der Paragraphen   *}
    ES := Cseg;                            {* Seg. Adr. Codebereich  *}
    AX := $4A00;                           {* MSDOS Funct. free Mem. *}

    MSDOS (register);                      {* Int 21 -> free Memory  *}
{
  Prüfung, ob beim Systemaufruf Fehler aufgetreten sind. In
  diesem Fall ist das Carry Flag gesetzt. Das Carry Flag ist
  das niederwertigste Bit des Flagregisters. Das Register AL ent-
  hält den Fehlercode.

  Achtung:  In MS-DOS 3.2 liegt vermutlich ein Bug vor, der bei
            Aufruf aus Subdirectories auch bei erfolgreicher
            Speicherfreigabe das Carrybit setzt. AL enthält damit
            undefinierte Werte.
}

    if (FLAGS and 1) <> 0 then
     begin
      if not (AX in [7,8,9]) then    { Fehler in MS-DOS 3.2  }
        writeln ('Wert Register AX illegal:',AX)
      else
       begin
        case AX of
         7 : writeln('Speicher-Kontroll-Block zerstört');
```

```
         8 : writeln('Speicherkapazität zu gering ',BX,
             ' Paragraphen frei');
         9 : writeln('Unzulässige Segmentzuordnung in ES');
        end; { case }
        writeln('Abbruch bei Speicherfreigabe');
        halt;
       end;
     end;
   end;
 end; { release_mem }

{ ***********   Root  Modul     *************** }

begin
 release_mem;                          {* unbel. Speicher freigeben *}
 get_environment;                      {* lese Environment d. Progr.*}
 prg_name := 'tree.com';               {* Name Tochterprozeß       *}
 writeln('Aufruf der EXEC Funktion');
 Exec(prg_name);
 writeln ('EXEC Aufruf erfolgreich beendet');
end.
```

11.7 Aufruf interner DOS-Kommandos durch den INT 2EH

Neben der INT 21-Funktion 4BH besitzt DOS eine undokumentierte zweite Möglichkeit, die EXEC-Routine des Kommandoprozessors zu aktivieren. Dies geschieht über den reservierten Interrupt 2EH. DOS benutzt den Einsprung, um selbst interne Programme (DIR, DEL, COPY etc.) des Kommandoprozessors zu aktivieren. Der Vorteil besteht darin, daß bei diesem Aufruf keine Kopie des Kommandoprozessors installiert wird, da ja die residenten und transienten Teile von COMMAND.COM die Befehle enthalten. Es ist zwar möglich, mit diesem Funktionsaufruf auch externe Programme zu laden und zu starten. Die Funktion übernimmt allerdings kein Memory Management. Falls bei größeren Programmen Teile von COMMAND.COM überschrieben werden, endet dies meist in einem Systemabsturz.

Wo bringt die Verwendung dieses EXEC-Aufrufes Vorteile?

Solange interne DOS-Kommandos aktiviert werden sollen, bietet der Aufruf hierzu eine elegante Möglichkeit. Interessant ist vor allem, daß keine Kopie des Kommandoprozessors geladen wird.

Ein weiterer Anwendungsfall tritt auf, falls ein laufender Prozeß den Inhalt des Environmentsegments verändern möchte. Normalerweise wird das Environment durch das interne DOS-SET-Kommando beeinflußt. Ein Prozeß könnte nun über die INT 21-Funktion 4BH versuchen, dieses Kommando zu aktivieren. Hier tritt aber das Problem auf, daß eine Kopie von COMMAND.COM installiert wird, die auch ein eigenes Environment enthält. Alle SET-Kommandos wirken sich dann auf das Tochterenvironment aus und haben keinen Einfluß auf das Masterenvironment. Es besteht zwar noch die Möglichkeit, im Parameterblock der 4BH-Funktion die Segmentadresse des Masterenvironments anzugeben. Aber insgesamt ist das Verfahren aufwendig und es ist nicht immer klar, ob DOS nicht in diesem Fall automatisch Änderungen am Masterenvironment vornimmt. Alternativ besteht die Möglichkeit, die Segmentadresse des Environments ab Offset 2CH

im aktuellen PSP zu lesen und dann die Einstellung direkt im Speicher zu modifizieren. Dies verlangt aber, im Prinzip alle Funktionen des SET-Befehls im Programm nachzubilden.

Wesentlich einfacher läßt sich das Problem mit dem INT 2E lösen. Das nachfolgende Programmbeispiel zeigt, wie das Masterenvironment unter Verwendung dieser Funktion verändert werden kann.

Der INT 2E erwartet im Registerpaar DS:DI einen Zeiger auf den String mit dem ausführbaren Kommando. Dieses Kommando ist in der Form:

```
< SPACE >< Kommandostring ><CR>
```

abzuspeichern.

Vor dem Aufruf der EXEC-Funktion ist der nicht mehr benötigte Speicher freizugeben. Hierfür dient die Prozedur *release_mem*. Anschließend wird das Kommando als Textstring angelegt. Die Aktivierung der EXEC-Funktion per INT 2E erfolgt über die Prozedur *EXEC*. Als Parameter wird lediglich der Kommandostring übergeben. Das Modul EXEC ergänzt diesen String dann um ein führendes Blank und schließt den Text mit einem CR ab.

Der eigentliche Aufruf ist in Assemblercode formuliert. Wichtig ist es, vor dem Aufruf die Register zu retten, da der INT 2E dies nicht übernimmt. Die Werte der SS- und SP-Register sind dann in lokalen Variablen zu sichern. Da der verwendete Compiler globale Variablen im Datensegment ablegt, muß nur noch die Offsetadresse von *Cmd_Str* in das SI-Register geladen werden. Nach dem Aufruf sind die Register wieder zu restaurieren. Dies gilt insbesondere für die Herstellung des Stackpointers.

Falls Fehler auftreten, ist nach dem Aufruf das Carry-Flag gesetzt und im Register AX findet sich ein Fehlercode.

Der EXEC-Aufruf über den INT 2E ist bei allen DOS-Versionen ab 2.0 möglich. Weitere Einzelheiten sind dem Listing zu entnehmen.

```
program int2e_demo;

{*****************************************************************}
{* Filename:    INT2E.PAS                                        *}
{* Autor:       Born G.                                          *}
{* Prog. Spr.:  Turbo Pascal V 3.0                               *}
{* Betr. Sys.:  MSDOS 3.x                                        *}
{*                                                               *}
{* Das Programm demonstriert die Benutzung der INT 2E EXEC-      *}
{* Funktion. Mit dieser Funktion lassen sich interne DOS-        *}
{* Programme aufrufen. Weiterhin wird die Funktion 4AH benutzt,  *}
{* um die Speichergröße aus aus dem laufenden Prozeß heraus zu   *}
{* variieren. Im Beispiel wird die Einstellung des Master-       *}
{* Environments durch das SET Kommando verändert.                *}
{*                                                               *}
{* Bei der Übersetzung sind die Optionen:                        *}
{*                                                               *}
{*        COM File                                               *}
{*        Min free dynamic memory = stack_size                   *}
{*        Max free dynamic memory = stack_size                   *}
{*                                                               *}
{* des Compilers gesetzt werden. Stack_size ist eine Pro-        *}
{* grammkonstante und wird hier auf 200 gesetzt.                 *}
{*                                                               *}
{* In Turbo Pascal 4.0 kann die EXECUTE-Funktion des Compilers   *}
{* direkt verwendet werden. Wegen der geänderten Lage der Seg-   *}
{* mente zur Laufzeit ist eine direkte Übertragung nicht mögl.   *}
{*****************************************************************}

{ *********** Declaration Modul *****************}

type Long_string  = STRING[80];

     reg8086 = record
            AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS: INTEGER;
     end;


const Stack_size  = $200;           {* Größe des HEAP und Stack *}

var
    Kommando     : Long_string;     {* Name child process       *}

    Cmd_str      : Long_String;     {* Kommandstring            *}


{ *********** Hilfsroutinen     *****************}

procedure Exec(ProgrammName: Long_string);

{*****************************************************************}
{*  Aktivierung des MS-DOS Kommandprozessors über den           *}
{*  Interrupt 2E. Dieser führt dann das interne Kommando aus.   *}
{*  Externe Kommandos sollten nicht aufgerufen werden, da die   *}
{*  Funktion kein Memory Management übernimmt. Das Modul ver-   *}
{*  sorgt die Register vor Eintritt in MS-DOS, insbesondere     *}
{*  werden SP und SS gesichert.                                 *}
{*****************************************************************}

const

   SS_save        : integer = 0;         {* Speicher für Stacksegm.   *}
   SP_save        : integer = 0;         {* und Stackpointer          *}
   AX_R           : integer = 0;         {* Hilfsvariablen für die    *}
   Flag_R         : integer = 0;         {* 8086 Flags und AX         *}

begin
 {**  Cmd_str ist als globale Variable deklariert und liegt demnach im  **}
 {**  Datensegment, das DS Register muß also nicht mehr gesetzt werden. **}

 Cmd_str := ' ' + ProgrammName + chr(13);    {* Name des child proc.  *}

 INLINE                     {****    Aufruf der EXEC Funktion     ****}
  ($0E/                     (* push cs        Register retten       *)
   $1E/                     (* push ds                              *)
   $55/                     (* push bp                              *)
   $06/                     (* push es                              *)
   $56/                     (* push si                              *)
   $57/                     (* push di                              *)
```

```
   $2E/                      (* cs:            rette SP und SS in    *)
   $89/$26/SP_save/          (* mov SP_save,sp lokalen Variablen     *)
   $8C/$D0/                  (* mov ax,ss              "             *)
   $2E/                      (* cs:                    "             *)
   $A3/SS_save/              (* mov SS_save,ax                       *)

   $BE/Cmd_str+1/            (* mov si,Cmd_str hole Adr. Cmd_str     *)
   $FA/                      (* cli            disable Interrupt     *)
   $CD/$2E/                  (* int 2E         EXEC  CALL            *)
   $FA/                      (* cli                                  *)
   $2E/$8B/$1E/SS_save/      (* cs: mov bx, SS_save                  *)
   $8E/$D3/                  (* mov ss, bx                           *)
   $2E/$8B/$26/SP_save/      (* cs: mov sp, SP_save                  *)
   $FB/                      (* sti                                  *)
   $5F/                      (* pop di         restauriere die Re-   *)
   $5E/                      (* pop si         gister                *)
   $07/                      (* pop es                               *)
   $5D/                      (* pop bp                               *)
   $1F/                      (* pop ds                               *)
   $0F/                      (* pop cs                               *)
   $2E/$A3/AX_R/             (* cs: mov AX_R, ax                     *)
   $9C/                      (* pushf                                *)
   $58/                      (* pop ax                               *)
   $2E/$A3/Flag_R);          (* mov Flag_R, ax                       *)

   if (Flag_R and 1) <> 0 then
    begin                                {* Fehlerflag gesetzt ? *}
     if not (Flag_R in [1,2,3,7,8,10,11]) then
      writeln ('Wert Register AX illegal:',AX_R)
     else
      begin                              {* Fehlercode ausgeben  *}
       writeln;
       case AX_R of
        1 : writeln('Fehler im Funktionsaufruf AL <> 0 oder 3');
        2 : writeln('Programm- oder Pfadname ungültig');
        3 : writeln('Pfadname ungültig');
        7 : writeln('Speicherkontrollblock zerstört');
        8 : writeln('Speicherkapazität zu gering.');
       10 : writeln('Falsches Environment.');
       11 : writeln('Fehler im .EXE File');
       end;
      end;
      halt;   { Programmabbruch }
    end;
end;  { Exec }

procedure release_mem;
{
  Freigabe des nicht benötigten Speichers an MS-DOS
  mittels der MS-DOS-Funktion Set Block.
  Parameter der INT 21-Funktion 4AH

   AX    ---> 4A00H
   BX    ---> Zahl der benötigten Paragraphen
   ES    ---> Segmentadresse des Speicherblocks

  Bei Fehlern ist anschließend das Carry Flag gesetzt und
  im Register AX findet sich ein Fehlercode.
}

 var register  : reg8086;

 begin
  {
     In MS-DOS besitzt Turbo Pascal folgenden Speicheraufbau:

          Codesegment
          Datensegment
          Stacksegment

     Der verfügbare Speicher läßt sich damit zu

         Mem := Stacksegment - Codesegment + Stackgröße

     bestimmen. Die Funktion $4A00 erwartet die Speichergröße
     in Anzahl 16 Bit Paragraphen. Die Stackgröße in Byte wird
     beim Programmstart durch SP angezeigt.
 }
  with register do
   begin                               {* Register setzen       *}
```

```
    BX := (Sseg - Cseg) + Stack_size;     {* Zahl der Paragraphen   *}
    ES := Cseg;                           {* Seg. Adr. Codebereich  *}
    AX := $4A00;                          {* MSDOS Funct. free Mem. *}

    MSDOS (register);                     {* Int 21 -> free Memory  *}
{
  Prüfung, ob beim Systemaufruf Fehler aufgetreten sind. In
  diesem Fall ist das Carry Flag gesetzt. Das Carry Flag ist
  das niederwertigste Bit des Flagregisters. Das Register AL ent-
  hält den Fehlercode.

  Achtung:  In MS-DOS 3.2 liegt vermutlich ein Bug vor, der bei
            Aufruf aus Subdirectories auch bei erfolgreicher
            Speicherfreigabe das Carrybit setzt. AL enthält damit
            undefinierte Werte.
}

    if (FLAGS and 1) <> 0 then
     begin
      if not (AX in [7,8,9]) then    { Fehler in MS-DOS 3.2  }
        writeln ('Wert Register AX illegal:',AX)
      else
       begin
        case AX of
         7 : writeln('Speicher-Kontroll-Block zerstört');
         8 : writeln('Speicherkapazität zu gering ',BX,
             ' Paragraphen frei');
         9 : writeln('Unzulässige Segmentzuordnung in ES');
        end; { case }
        writeln('Abbruch bei Speicherfreigabe');
        halt;
       end;
     end;
   end;
 end; { release_mem }


{ ***********   Root  Modul     ********************* }

begin

 release_mem;                             {* unbel. Speicher freigeben *}

 Kommando := 'SET VER=3.0';               {* Name Tochterprozeß       *}

 writeln('Aufruf der EXEC Funktion');

 Exec(Kommando);

 writeln ('EXEC Aufruf erfolgreich beendet');

end.
```

11.8 Residente Programme (TSR)

In DOS werden Prozesse üblicherweise als Programmcode in den Speicher geladen und gestartet (INT 21-Funktion 4BH). Nach Erfüllung der Aufgaben terminiert der Prozeß per INT 20 oder über die INT 21-Aufrufe 00H oder 4CH. Diese Funktionen aktivieren dann den Vaterprozeß (meist COMMAND.COM) und geben den belegten Speicher wieder frei.

Nun gibt es aber eine ganze Reihe von Programmen, die quasi im Hintergrund ihren Dienst versehen. Das bedeutet, daß sie zwar zeitweise suspendiert sind, aber weiterhin im Speicher verbleiben. Der DOS-Druckerspooler PRINT.COM oder das Programm SideKick sind Vertreter aus der Klasse der residenten Programme. Sie werden als normale Prozesse durch DOS installiert, verbleiben aber nach der Beendigung im Speicher. Damit läßt sich ein solcher Prozeß jederzeit wieder aktivieren. Allerdings bietet die offizielle Microsoft-Dokumentation kaum Informationen über das Thema TSR-Programme (Terminate But Stay Resident). Der nachfolgende Abschnitt bietet deshalb eine Zusammenfassung

verschiedener Informationen zu diesem Komplex und schließt mit einem kleinen Programmbeispiel ab. Detailliertere Informationen über das Thema finden sich in /1/.

Erzeugung eines TSR-Prozesses

Um einen residenten Prozeß zu erzeugen, ist der Programmcode wie gewohnt zu laden. Dies kann von der DOS-Ebene oder durch die Funktion 4BH erfolgen. Der laufende Prozeß darf allerdings anschließend nicht durch den INT 20- oder die INT 21-Aufrufe 00H und 4CH beendet werden. Vielmehr bietet DOS für die Installation residenter Prozesse folgende Schnittstellen:

- Terminierung über den INT 27 (Terminate But Stay Resident)
- Terminierung über den INT 21-Aufruf 31H (Keep Process)

Die Aufrufkonventionen sind in den jeweiligen Kapiteln dokumentiert. Bei Neuentwicklungen ist der INT 21-Aufruf 31H zu bevorzugen. Er bietet insbesondere die Möglichkeit, die Größe des resident belegten Speicherbereiches zu variieren. Somit kann ein residentes Programm aus einem Installations- und einem residenten Aktionsteil bestehen. Der Installationsteil wird nur einmalig benötigt und kann anschließend über den INT 21, AH = 49H freigegeben werden. Dadurch wird nur der Aktionsteil resident installiert.

Nach dem Aufruf der DOS-EXIT-Funktion (INT 27 oder INT 21) wird der laufende Prozeß suspendiert, wobei der belegte Speicherbereich aber reserviert bleibt. Dabei werden auch die folgenden Adressvektoren im PSP auf den alten Zustand restauriert:

```
INT 22 Programm Terminate Adresse
INT 23 Control-C Handler Adresse
INT 24 Critical Error Handler Adresse
```

Anschließend geht die Kontrolle an den Vaterprozeß zurück. Falls es sich hier um COMMAND.COM handelt, erscheint der gewohnte DOS-Prompt. Nun lassen sich beliebige weitere Aktionen in DOS ausführen, ohne das dies den residenten Prozeß tangiert.

Hinweis: Aus einem Anwenderprogramm heraus sollten keine residenten Programme geladen werden, da es sonst zu einer Fraktionierung des Hauptspeichers kommt.

Aktivierung residenter Prozesse

Zur Aktivierung der suspendierten Prozesse existieren nun verschiedene Möglichkeiten unter DOS. Allen Variationen ist gemeinsam, daß der Prozeß durch ein externes Ereignis aufgerufen wird.

Periodische Aktivierung per Timerfolgeinterrupt

Zur periodischen Aktivierung eignen sich die Timerinterrupts. Der Interruptvektor läßt sich dann so initialisieren, daß er auf den Codebereich des residenten Prozesses zeigt. Bei jedem Timer-Interrupt wird dann das residente Programm aktiv.

Im PC sind in der Regel mehrere Timer vorhanden, die auf den Interrupt-Controller aufgelegt sind. Die Benutzung des Timerinterrupts erfordert allerdings einige Informationen über die Systemhardware, deren Beschreibung den Rahmen dieses Buches sprengen würde.

In der Regel ist es leichter einen bestehenden Timerinterrupt für die periodische Aktivierung des residenten Prozesses zu verwenden. Hierfür stehen bei jedem MS-DOS-System zwei Interrupts zur Verfügung:

- der Timerinterrupt INT 8
- der Timerfolgeinterrupt INT 1C

Der INT 1C wird zyklisch durch den INT 8 aktiviert. Es empfiehlt sich daher, den INT 1C-Vektor auf die Serviceroutine des residenten Prozesses umzulegen. Dann wird dieser Prozess ca. 18.2 mal pro Sekunde aktiviert. Nachfolgendes Programmfragment zeigt die Einbindung des INT 1C in ein Programm.

```
;---------------------------------------------------------
; Hier beginnt der residente Teil des Programmes
;---------------------------------------------------------
START:  PUSH  AX             ; rette alle benutzten Register
         .                   ; hier werden die Aktionen des
         .                   ; Programmes ausgeführt
         .                   ;
        POP   AX             ; restauriere Register
        IRET                 ; Ende Handler
ENDE:
        END
```

Der Prozess muß nur dafür sorgen, daß alle benutzten Register vorher auf dem Stack gesichert werden. Nach der Ausführung des Codebereiches sind die Register zu restaurieren und der Prozess terminiert mit einem einfachen IRET-Befehl.

Normalerweise initialisiert DOS den INT 1C so, daß er auf eine IRET-Anweisung im Speicher zeigt. Das TSR-Programm muß aber sicherstellen, daß gegebenfalls bereits unter diesem Interrupt installierte Programme aufgerufen werden. Es ist auch zu beachten, daß bei Verwendung der Timerinterrupts die Laufzeit der Prozesse nicht die Taktrate übersteigt, da sonst Zeittakte verloren gehen.

Aktivierung über die Tastaturinterrupts

Residente Programme sollen oft über eine Tastatureingabe aufrufbar sein. Dabei werden bestimmte Tastenkombinationen (z.B. Ctrl + ALT + 1) als sogenannte *Hot Keys* definiert. Sobald diese Kombination vom residenten Prozess erkannt wird, beginnt dieser mit der Abarbeitung der internen Funktionen. Andernfalls geht die Kontrolle an die Tastaturtreiber zurück. DOS bietet nun verschiedene Möglichkeiten zur Auswertung der Tastatureingaben.

Der INT 9 des Tastaturkontrollers Die Tastatur wird durch einen eigenen Single Chip Prozessor (Intel 8042) überwacht. Sobald eine Taste gedrückt wird, decodiert der Kontroller das Signal und löst über den 8259 Interruptkontroller einen INT 9 aus. Dieser Interrupt wird dann über eine entsprechende BIOS-Routine, entweder im ROM oder aus dem Programm IO.SYS, bearbeitet. Die TSR-Routine kann den Vektor abfangen und direkt mit dem Tastaturkontroller über die IO-Adressen 60H bis 6FH kommunizieren. Hierzu sind aber gute Systemkenntnisse erforderlich.

Die BIOS-Routinen legen die gelesenen Zeichen in einem internen Ringpuffer ab. Dieser Puffer findet sich im BIOS-Datenbereich ab der Adresse 0000:0417H (siehe BIOS-Datenbereich in Kapitel 1). Weiterhin finden sich im ALT-Keypad-Flag (ebenfalls im BIOS-Datenbereich) die Statusinformationen, ob bestimmte Kontrolltasten (Alt, Shift, Num Lock, etc.) gedrückt wurden. Die Bits lassen sich durch den residenten Prozess abfragen. Weiterhin kann über den INT 16 der Tastaturpuffer auf entsprechende Zeichen untersucht werden. Allerdings ist das nachfolgend beschriebene Reentrance Problem zu beachten. Weiterhin dürfen eingetragene Zeichen nicht aus dem Puffer entfernt werden, da sie sonst für den Anwenderprozess nicht mehr verfügbar sind. Der Vollständigkeit halber sei noch eine weitere Möglichkeit vorgestellt. Im BIOS-INT 9 werden die Tastatureingaben auf die Zeichen Ctrl-Break untersucht. Sobald diese Kombination auftritt, wird eine Ctrl-C Unterbrechung (INT 1B) ausgelöst. Dieser Vektor ist in DOS meist mit einer IRET-Anweisung abgeschlossen. Falls Anwenderprogramme den Vektor neu definieren, kann er zur Aktivierung residenter Prozesse genutzt werden.

Aktivierung per INT 10

Der INT 10 eignet sich zur Aktivierung residenter Programme, die Bildschirmausgaben beeinflussen sollen. Soll eine residente Applikation in diesen Datenstrom eingreifen, kann der Prozess den Interruptvektor belegen. Erst nachdem die Zeichen gefiltert wurden, ist dann die eigentliche BIOS-Routine zu aktivieren. Ein Nachteil sei allerdings auch nicht verschwiegen: Es gibt viele Programme, die direkt in den Bildschirmspeicher schreiben und sobald der INT 10 nicht mehr aufgerufen wird, läßt sich natürlich der Prozess auch nicht mehr aktivieren.

Aktivierung durch einen beliebigen Softwareinterrupt

Natürlich lassen sich residente Prozesse durch jeden anderen Softwareinterrupt aktivieren. Hierzu ist lediglich die Einsprungadresse unter dem betreffenden Vektor zu speichern. So kann die Ausgabe an COMx: oder LPT: abgefangen werden. Auch eine Erweiterung der DOS-INT 21-Funktionen ist denkbar. Um dann den residenten Prozess zu aktivieren, muß im rufenden Programm lediglich ein Softwareinterrupt ausgelöst werden.

Wird ein residentes Programm in den INT 28 eingehängt, aktiviert DOS den Prozess jedesmal, wenn Tastatureingaben erfolgen.

Eine Möglichkeit besteht zum Beispiel auch in der Nutzung des DOS-Multiplexerinterrupts INT 2F.

Aktivierung per CALL oder JMP

Als letzte Alternative besteht die Möglichkeit, ein residentes Programm mit einem CALL- oder JMP-Befehl zu aktivieren. Das Problem liegt jedoch darin, die Startadresse des residenten Programmes zu ermitteln. Diese wird ja durch DOS beim Laden des Codes festgelegt. Eine denkbare Möglichkeit besteht darin, daß das residente Programm die Startadresse im BIOS-Datenbereich ab Adresse 0000:04F0 bis 0000:04FF ablegt. Dann läßt sich der Aufruf über einen indirekten FAR CALL abwickeln.

Die Möglichkeiten zur Aktivierung eines suspendierten Prozesses sind also vielfältig. Nach diesen Informationen, die teilweise noch in der offiziellen DOS-Dokumentation enthalten sind, sollten TSR-Programme leicht zu erstellen sein. Solange das Programm nur einige Assemblerbefehle enthält, trifft dies auch zu. Sobald aber DOS- oder BIOS-Funktionen benutzt werden, steckt der Teufel im Detail. Ohne bestimmte Vorkehrungen sind Systemabstürze vorprogrammiert. Leider bietet die offizielle Dokumentation der Hersteller kaum Informationen über die internen Abläufe in DOS, so daß nachfolgend einige Ausführungen zu diesem Thema zusammengefaßt werden.

Abfrage des Installationsstatus

In der Praxis kommt es vor, daß ein residentes Programm bereits installiert ist und der Benutzer versucht, dieses Programm ein weiteres Mal zu installieren. Ein residentes Programm muß also in der Installationsphase die Möglichkeit zur Überprüfung des Installationsstatus besitzen. Ist bereits ein Prozess mit dem Namen installiert, muß die Installation abgebrochen werden. Im folgenden Abschnitt werden einige Techniken vorgestellt.

Abfrage per INT 2F

Bei der Diskussion des *Multiplexer Interrupts* (INT 2F) wurde die Möglichkeit zur Installationsabfrage vorgestellt. Hierfür ist beim Aufruf der Code für die INT 2F-Funktion in AH und AL der Wert 00H zu übergeben. Der Code im Register AH dient zur Identifikation des gewünschten Prozesses. Mit der Unterfunktion *get installation status* (Wert AL = 0) wird geprüft ob ein Prozess bereits vorhanden ist. Ein Programm, welches sich an die Konventionen des INT 2F hält, gibt dann in AL den Installationsstatus zurück. Dabei gilt folgende Kodierung:

```
AL = 0  Treiber nicht installiert
AL = 1  Treiber nicht installierbar
AL = FF Treiber installiert
```

Ein TSR-Programm kann sich in den INT 2F einhängen und einen unbenutzten Code belegen. Bei der Installation kann dann jedes Programm über den INT 2F feststellen, ob bereits ein Treiber aktiviert ist. Hierzu muß der residente Prozeß nur obige Konventionen erfüllen. Allerdings ist es mit etwas Aufwand verbunden, so daß folgende Techniken vermutlich vorgezogen werden.

Signatur eines Interruptvektors

Eine einfache Möglichkeit besteht darin, unter einem unbenutzten Interruptvektor eine Signatur (z.B. 55FF:FF55) abzulegen. Dann muß nur diese Signatur bei der Installation

überprüft werden. Sofern aber ein Prozeß diesen Interrupt aufruft, kommt es zu einem Systemabsturz.

Deshalb sollte folgende Alternative erwogen werden. Das betreffende Programm fängt bei der Installation einen unbenutzten Interruptvektor ab und hinterlegt eine Adresse auf den eigenen Codebereich. Hier steht lediglich ein IRET-Befehl, der für einen sauberen Abschluß sorgt. Hinter dem einen Codebyte für den IRET-Befehl folgt dann eine Signatur in Form einer Textkonstante (z.B. TSR-Handler 4711). Nach der Überprüfung auf einen gültigen Vektor muß an der angegebenen Adresse der Code für die IRET-Anweisung stehen. Dahinter folgt dann die Signatur, die sich eindeutig auswerten läßt.

Ein Nachteil soll aber nicht verschwiegen werden. Benutzen mehrere residente Programme den gleichen Vektor, funktioniert die Abfrage der Signatur durch ein externes Programm nur, falls der gesuchte Treiber als letzter installiert wurde. In allen anderen Fällen definieren die anschließend geladenen Programme den Vektor neu und die Folgeadresse auf die nächste Routine wird intern im Codebereich des Treibers gespeichert. Es besteht in der Regel keine Möglichkeit, diesen Code nach dem gesuchten Vektor zu inspizieren. Hier muß nach einem Ausweg gesucht werden. Zum Beispiel könnte das Programm unter dem Interrupt-Handler eine Routine installieren, die in einem Register einen bestimmten Wert erwartet und einen weiteren Wert zurückgibt. Dann muß das zu installierende Programm lediglich einen Interrupt mit den vereinbarten Aufrufkonventionen ausführen.

Signatur des RAM's

Bei der Verwendung von Hochsprachen treten bei den oben beschriebenen Verfahren immer wieder Probleme auf. Dies liegt oft daran, daß kein direkter Zugriff auf übergebene Register möglich ist. Also muß jedesmal ein Stück Assemblercode mit eingebunden werden. Hier setzt ein etwas anders Verfahren an. Die Signatur wird einfach in den RAM-Speicher geschrieben. Ist die Adresse der Signatur bekannt, läßt sich diese leicht von Hochspracheprogrammen auslesen. Es ist nur erforderlich, daß in der Sprache ein Zugriff auf absolute Speicheradressen möglich ist.

Ein geschützter Bereich im RAM stellt der BIOS-Datenbereich dar. Er liegt einmal auf einer absoluten Adresse (0000:0400) und wird bei jedem Systemstart initialisiert. Für unsere Zwecke interessiert nur der Kommunikationsbereich für Zwischenanwendungen zwischen 0000:04F0 und 0000:04FF. Auf einer dieser Adressen läßt sich die Signatur (z.B. 4742H) von dem installierten Programm ablegen. Ein anderer Prozess kann dann durch Abfrage dieser Speicherstelle leicht feststellen, ob die Installation noch möglich ist. Problematisch ist es aber, falls mehrere residente Programme diese Speicherstelle benutzen. Hier sollte vor jedem Schreibzugriff auf die Adressen geprüft werden, ob nicht vielleicht bereits eine Signatur vorliegt. Aber für eigene Zwecke reicht das Verfahren aus.

Abfrage der Memory-Control-Blocks

Als letztes Verfahren soll der Vollständigkeit halber noch die Möglichkeit zur Analyse der Speicherbelegung angesprochen werden. Ab DOS 3.0 hinterlegt das Betriebssystem im MCB den Namen des zugehörigen Programmes. Dieser Name kann zur Installationsprüfung ausgewertet werden. Durch Abfrage der MCB-Kette und Analyse der eventuell vorhanden ASCIIZ-Strings läßt sich natürlich herausfinden, ob ein residentes

Programm geladen wurde. Dies sagt allerdings nichts über den Installationsstatus aus. In der Praxis wird diese Methode deshalb kaum eingesetzt.

Kommunikation mit residenten Programmen

Ein weiteres Thema ist der Datenaustausch zwischen einem externen Programm und dem residenten Prozess. Wie übergibt der aktive Prozess seine Befehle und Daten zum residenten Prozess und wie erhält er eventuell Ergebnisse zurück? Auch hier gibt es mehrere Techniken, um Parameter zu übergeben.

- Benutzung des BIOS-Datenbereiches 04F0:0000 zum Austausch von Daten. Gegebenenfalls kann hier auch die Adresse auf einen Datenbereich des TSR-Prozesses abgelegt werden. Über diesen Datenbereich können Parameter ausgetauscht werden.
- Die Parameter lassen sich per Register beim Aufruf eines Interrupts übergeben (z.B. INT 21). Längere Datenbereiche sind im rufenden Programm in einem Puffer zu hinterlegen. Dann wird die Pufferadresse in einem Doppelregister übergeben.

Es bieten sich also vielfältige Möglichkeiten zur Datenübergabe an den residenten Prozess.

Das DOS-Reentrance-Problem

Nach diesen Vorüberlegungen sollte sich ein residentes Programm installieren und einfach aufrufen lassen. In der Praxis wird es dann aber schnell zu einem Systemabsturz kommen, sobald DOS-Funktionen benutzt werden. Der Grund für die Probleme beim Aufruf von Systemfunktionen aus einem TSR-Programm liegt darin, daß alle DOS-Systemaufrufe und auch ein Teil der BIOS-Funktionen (z.B. INT 13) nicht reentrant sind. Das bedeutet, daß, solange ein Systemaufruf nicht abgewickelt wurde, kein zweiter Aufruf erfolgen darf. Warum dies so ist, hängt mit einigen Eigenschaften von DOS zusammen. Die Entwickler von DOS wußten seinerzeit nicht im voraus, wie groß der Stackbereich der verschiedenen Anwenderprozesse sein würde. Andererseits führten sie aber auch interne Stackbereiche für die DOS-Systemfunktionen ein. So kommt es, daß heutige DOS-Versionen neben den Anwenderstacks mindestens noch drei weitere Systemstacks verwalten. Sobald nun ein Systemaufruf aus einem Anwenderprozeß erfolgt, schaltet DOS auf einen der internen Systemstacks um. Dies ist an sich keine große Affäre (viele Betriebssysteme tun dies), wenn nicht eine Eigenart der Intel 80x8-Prozessoren hier Probleme bereiten würde. Die CPUs besitzen nur ein internes Stackregister, so daß dieses jeweils auf die neue Adresse des Stackbereichs zu schalten ist. Hier sei angemerkt, daß viele andere Prozessoren mindestens einen User-Stackpointer und einen System-Stackpointer besitzen, wodurch solche Kontexwechsel wesentlich einfacher werden. Bei den Intel-Prozessoren muß die Umschaltung durch ein Stück Code explizit erfolgen. Da DOS nun ein Single-Tasking-Betriebssystem ist, haben die Entwickler auf reentrante DOS-Funktionsaufrufe verzichtet. Die Umschaltung zwischen Userstack und Systemstack wurde meist analog folgender Sequenz implementiert:

```
MOV     AX,CS               ; merke Adresse Codesegment
CLI                         ; sperre externe Interrupts
MOV     CS:USER_SS,SS       ; rette Stackpointer und
MOV     CS:USER_SP,SP       ; -segment in lokalen Variablen
MOV     SS,AX               ; Stacksegment = Codesegment
MOV     SP,SYSTEM_SP        ; setze neuen Stackpointer
STI                         ; externe Interrupts zulassen
```

Bild 11.4: Umschaltung auf den Systemstack

Der Systemstack liegt im Codesegment der jeweiligen Systemfunktion. Die Adresse des Anwenderstacks wird dann in den lokalen DOS-Variablen *USER_SS* und *USER_SP* gesichert. Anschließend werden die Register auf den Systemstack umgeschaltet. Vor der Rückkehr in den Anwenderprozeß wird eine ähnliche Sequenz durchlaufen, die die Stackregister wieder auf die alte Adresse des Anwenderstacks zurücksetzt. Dies funktioniert im normalen DOS-Betrieb auch recht gut. Probleme treten aber bei Interrupt-Service-Routinen und TSR-Programmen auf. Nehmen wir an, ein aktiver Prozeß setzt einen Systemaufruf ab, der in obiger Weise bearbeitet wird. Die Adresse des Anwenderstacks findet sich nach dem Systemaufruf in den lokalen Variablen *USER_SP* und *USER_SS*. Der Stackpointer zeigt auf eine Adresse innerhalb des internen Systemstacks, die (meist) nicht mehr mit dem Initialisierungswert in *SYSTEM_SP* übereinstimmt. Nun unterbricht ein externer Interrupt die Abwicklung des Systemaufrufes. Dies ist undramatisch, da der Systemstack durchaus Daten in gewissem Umfang aufnehmen kann. Falls aber innerhalb dieser Routine auch Systemaufrufe benutzt werden, tritt folgende Situation auf:

- Die lokalen Variablen *USER_SS* und *USER_SP* werden mit den aktuellen Registerwerten überschrieben.
- Der Stackpointer wird auf den Initialisierungswert von *SYSTEM_SP* gesetzt.

Damit ist einmal die Adresse des alten Anwenderstacks zerstört und weiterhin werden alle alten Einträge des ersten Systemaufrufes auf dem Stack überschrieben. Der zweite Systemaufruf wird zwar noch korrekt abgewickelt, aber anschließend stürzt das System ab, da es nicht mehr in den Anwenderprozeß zurückkehren kann.

Da das System mehrere interne Stacks besitzt, ist die gleichzeitige Bearbeitung mehrerer Systemaufrufe erlaubt, sofern diese unterschiedliche Stacks benutzen. Nachfolgende Tabelle zeigt eine Übersicht der Funktionen und der zugeordneten Stacks.

Stack	Systemfunktion
1	INT 25, INT 26, INT 21-Funktionsaufruf 00H und alle Aufrufe ab 0DH
2	INT 21-Funktionsaufrufe 01H bis 0CH
3	intern von den Systemfunktionen benutzt

Tabelle 11.6: Zuordnung der Systemstacks

Die BIOS-Funktionen sind (Ausnahme: INT 13-Disk-I/O) in der Regel reentrant, da sie den Anwenderstack benutzen. Die DOS-INT 25 und INT 26 benutzen den internen Stack Nr. 1. Weiterhin gilt dies auch für die INT 21-Funktionsaufrufe 00H und alle weiteren ab

0DH. Für die Zeichen-Ein-/Ausgabefunktionen schaltet der INT 21 auf einen eigenen Stack um, damit keine Störungen mit anderen Systemfunktionen auftreten. Die Dokumentation gibt deshalb an, daß die INT 21-Aufrufe 01H bis 0CH in Interruptroutinen benutzt werden dürfen. Da dies zu Problemen führen kann, wird hier nun deutlich. Der dritte interne Stack wird vermutlich von Funktionen benutzt, die sich gegenseitig rekursiv aufrufen.

Es ist also höchst heikel, aus einem Interrupt-Handler eine DOS-Funktion aufzurufen. Andererseits gibt es Programme wie PRINT.COM, die genau dies tun. Wie kann ein Prozeß aber feststellen, ob DOS gerade einen Systemaufruf bearbeitet?

Hier bietet DOS mehrere undokumentierte Möglichkeiten, die nachfolgend skizziert werden.

Abfrage des DOS-Critical-Region-Flags

DOS führt intern ein Flag (Byte), in dem die Zahl der in Bearbeitung befindlichen Systemaufrufe vermerkt wird. Bei jedem Aufruf wird der Wert um 1 erhöht und nach der Bearbeitung wieder erniedrigt. Bei rekursiven Aufrufen können Werte zwischen 1 und 255 auftreten. Beim Wert 0 ist aber kein Systemaufruf in Bearbeitung, so daß Funktionsaufrufe erfolgen dürfen. Mittels der INT 21-Funktion 34H läßt sich die Adresse dieses Bytes (Registerpaar ES:BX) ermitteln. Es ist aber zu beachten, daß auch die Funktion 34H nicht reentrant ist, ein TSR-Prozeß muß also die Adresse bereits in der Installationsphase ermitteln und in lokalen Variablen speichern. Zur Laufzeit ist ein Aufruf nicht mehr möglich. Über diese Adresse läßt sich dann der Zustand des Flags jederzeit überprüfen.

Der INT-28-Aufruf

Leider kommt es in der Praxis teilweise zu Problemen, falls ein TSR-Prozeß ein gesetztes Flag vorfindet und sich wieder suspendieren muß. Es besteht zwar die Möglichkeit einer zyklischen Reaktivierung über einen Timerinterrupt (z.B. 1CH). Aber oft ist es erwünscht, einen TSR-Prozeß nur dann zu aktivieren, falls sich nicht gerade ein DOS-Prozeß im kritischen Bereich befindet. Dies ist zum Beispiel der Fall, wenn DOS auf Tastatureingaben wartet. Hier liegt zwar ein unbearbeiteter Systemaufruf (0AH) vor, aber Systemaufrufe oberhalb 0CH benutzen einen eigenen Stack, können also problemlos benutzt werden. Um nun einen solchen Zustand anzuzeigen, löst DOS in diesen Fällen den undokumentierten INT 28 aus. Dieser ist normalerweise mit einer IRET-Anweisung abgeschlossen. Hier kann sich dann ein entsprechendes Programm einhängen. Vor dem Aufruf des INT 28 wird ein Flag auf dem Stack gesichert, dessen unteres Byte signalisiert, ob der Aufruf einer DOS-INT 21-Funktion oberhalb 0CH erlaubt ist. Dies ist der Fall, wenn der Wert = 1 ist. Solange das DOS-Critical-Region-Flag den Wert 0 oder 1 besitzt und das Byte auf dem Stack = 1 ist, können die Systemfunktionen benutzt werden.

DOS 5.0/6.0 Idle Call (INT 2F, AX = 1680);

DDiese Möglichkeit besteht erst seit DOS 5.0 und soll die INT 28-Abfrage ablösen. Sobald DOS auf Eingaben wartet, aktiviert es neben dem INT 28 ebenfalls die Funktion 1680H des INT 2F. Ein Anwenderprogramm kann dann den Wert des DOS-Critical-Region-Flags analog dem INT 28 auswerten. Sofern der Wert 0 ist, können DOS-Funktionen aufgerufen werden.

Abfangen des INT-21-Aufrufes

Manchmal möchte man aber mehr Kontrolle über das System besitzen. Hier ist es möglich, den INT 21-Vektor auf eine eigene Routine umzuleiten. Dann lassen sich Art und Zahl der in Bearbeitung befindlichen Systemaufrufe leicht ermitteln. Die Aufrufe werden anschließend an den eigentlichen DOS-Dispatcher (INT 21) weitergeleitet. Der Vorteil der Methode besteht darin, daß einmal bestimmte Systemaufrufe verzögert werden können, um vorher den TSR-Prozeß zu starten. Andererseits gelangt bei entsprechender Programmierung die Kontrolle nach dem Systemaufruf wieder an die Routine zurück. Diese kann nun entscheiden, ob sie den Anwenderprozeß oder den TSR-Prozeß aktiviert. Eine Verwaltung der INT 25- und INT 26-Aufrufe ist ebenfalls mit dieser Technik möglich. Dadurch erlangt der TSR-Prozeß eine sehr starke Kontrolle über die Abläufe innerhalb des Betriebssystems.

Sicherung des DOS-Datenbereiches

Eine recht effektive Möglichkeit nutzen einige DOS-Systemprogramme. Werden sie aktiviert, retten sie einnfach den kompletten DOS-Datenbereich. Zur Ermittlung der Adresse lassen sich die undokumentierten INT 21-Funktionen 5D06H und 5D0AH nutzen. Sobald der Bereich gesichert wurde, kann ein DOS-Aufruf erfolgen. Näheres hierzu findet sich in /3/.

Sonderfragen zu TSR-Programmen

Neben der korrekten Behandlung des Reentrance Problems müssen weitere Klippen umschifft werden.

Aktivierung der korrekten Segmente

Durch die Segmentierung des Speichers der 80x86 Prozessoren tritt ein Problem in den Vordergrund, welches sonst häufig übersehen wird. Jede Adresse ist mit Segment- und Offsetwert anzugeben. Dies bedeutet, daß vor einem Zugriff auf Code, Stack oder Daten erst die korrekte Segmenteinstellung vorgenommen werden muß. Dies ist durch geeignete Konstruktionen im Code des TSR-Programmes sicherzustellen.

Bei COM-Dateien genügt folgende kleine Codesequenz:

```
;--------------------------------------
; setze Datensegment auf eigenen Bereich
;--------------------------------------
     PUSH   DS          ; merke alte Einstellung
     PUSH   AX          ; sichere Register
     MOV    AX,CS       ; lade Adresse CS
     MOV    DS,AX       ; setze Adresse DS
      .
      .
     POP    AX          ; restauriere Register
     POP    DS          ; altes Datensegment
     IRET               ; Ende Prozess
```

Bei EXE-Dateien muß die Lage des Datensegmentes explizit abgefragt werden. In Assembler läßt sich dies mit folgender Sequenz erreichen.

```
;--------------------------------------
; setze Datensegment einstellen
;--------------------------------------
     PUSH   DS          ; merke alte Einstellung
     PUSH   AX          ; sichere Register
     MOV    AX,@DATA    ; lade Adresse CS
     MOV    DS,AX       ; setze Adresse DS
      .
      .
     POP    AX          ; restauriere Register
     POP    DS          ; setze altes DS
     IRET               ; Ende Prozess
```

Mit der Anweisung @DATA wird eine Konstante mit der Segmentadresse des Datenbereiches geladen. Diese Konstante wird durch den Linker im Code abgelegt.

Für den Stack gelten ähnliche Bedingungen. Sofern nicht allzuviele Parameter dort gespeichert werden, kann der Stack des unterbrochenen Prozesses mit benutzt werden. Sicherer ist es aber, den Stackpointer auf einen eigenen Bereich umzuschalten.

Aktivierung des Program-Segment-Prefix (PSP)

Neben der korrekten Definition des Code-, Daten- und Stacksegmentes benötigt ein Prozess weitere Informationen. Benutzt der residente Prozess zum Beispiel die DOS-Ein-/Ausgabefunktionen, droht ohne spezielle Maßnahmen einiges an Unheil. DOS verwaltet die Zugriffe auf Ein-/Ausgabekanäle (COM, CON, etc.) und Dateien über die Handles des PSP. Wird nun ein residenter Prozess gestartet, wird der PSP-Bereich des gerade unterbrochenen Prozesses benutzt. Öffnet ein aktiver Prozess eine neue Datei, wird im aktiven PSP ein neues virtuelles Handle angelegt. Ähnliches gilt für einen Zugriff auf bereits offene Dateien. Bei residenten Prozesses ist DOS aber nicht über den Contextwechsel informiert. In seinem internen Datenbereich hält es nach wie vor den Zeiger, der auf das PSP des gerade unterbrochenen Anwenderprozesses zeigt. Benutzt der residente Prozess nun eine der INT 21-Funktionsaufrufe zum File-I/O per Handles, kann er aber nur die eigenen Handles angeben. DOS übernimmt diese als Offset in die Handletabelle des (unterbrochenen) Anwenderprozesses und greift damit auf die Dateien dieses Prozesses zu. Es ist wohl leicht einsichtig, daß dies zu erheblichen Problemen führt.

Daher muß der residente Prozess selbst einen Contextwechsel in DOS veranlassen. Die Lage des eigenen PSP kann der residente Prozess ermitteln. Ab der Version 2.0 bietet DOS den undokumentierten Funktionsaufruf 51H zur Abfrage des aktiven PSP. Wird nun ein residenter Prozess per Interrupt aktiviert, muß er dafür sorgen, daß das korrekte PSP von DOS benutzt wird. Hierzu ist per INT 21, Funktion 50 das eigene PSP zu setzen. Zuerst ist allerdings die alte Einstellung zu sichern, da sie ja vor Beendigung des Aufrufes wieder herzustellen ist.

Achtung: Wird ein Prozess durch den INT 28 aktiviert, ist eine Verwendung der INT 21-Funktion 51H nicht möglich. Die Ursache liegt in einem internen DOS-Bug, der bei einem Aufruf der Funktion aus dem INT 28 den Stack in einem unkonsistenten Zustand hinterläßt.

Eine Veränderung des aktiven PSP ist bei allen File-I/O-Zugriffen und bei Operationen deren Vektor im PSP liegt (Environment, Terminate Prozess, Critical Error, etc.) erforderlich.

Beachten Sie auch, daß jedes TSR-Programm bereits bei der Installation 5 Handles (I/O-Geräte und Fehlerhandler) zugewiesen bekommt. Für jeden Handle wird in der System File

Table ein Eintrag angelegt. Bei mehreren TSR-Programmen werden die entsprechenden Handles blockiert, d.h. den Anwendungsprogrammen stehen diese nicht mehr zur Verfügung. Das DOS-20-File-Problem wird dann akut, obwohl die Anwendung vielleicht nur 10 Files benutzt. Bei TSR-Programmen sollten diese die Standardhandles bei der Installation freigeben, da diese in der Regel nicht weiter benötigt werden. Gegebenfalls können Sie ja beim Aufruf neu aktiviert werden.

Wer Sie sich ausführlich über die Thematik informieren möchte, sei auf /1/ und /3/ verwiesen.

Ein Beispielprogramm

Nach diesen Vorüberlegungen soll nun ein kleines Programmbeispiel vorgestellt werden. Hierbei werden allerdings nicht alle Techniken benutzt. Es handelt sich um ein residentes Programm, welches prüft, ob die Tasten Shift links, Shift rechts, Alt und Ctrl gleichzeitig gedrückt wurden. Trifft dies zu, dann wird der gerade laufende Prozeß beendet und die Kontrolle geht an DOS zurück. Dies ist hilfreich, falls ein Prozeß »hängt Texten aus, sind alle Änderungen verloren. Mit dem kleinen TSR-Programm lassen sich fast alle Programme abbrechen. Der Speicherinhalt kann anschließend mit DEBUG inspiziert und gesichert werden. Bei einigen Programmen (z.B. PCTOOLS) funktioniert dies scheinbar nicht, da kein DOS-Prompt nach der Betätigung der Tasten erscheint. Dies liegt daran, daß diese Programme verschiedene Bildschirmseiten benutzen, während DOS in die Seite 0 schreibt. Mit der Eingabe

```
MODE BW80
```

läßt sich nach dem Aufruf die Seite 0 selektieren und der DOS-Prompt erscheint wieder.

Das eigentliche Programm besteht aus einem Initialisierungs- und einem Aktionsteil, welche nachfolgend besprochen werden.

Das Hauptmodul

Das Hauptmodul dient zur Initialisierung und wurde in Pascal implementiert. Es soll die TSR-Routine in den INT 8 (Timer) einhängen, da dieser Teil die eigentliche Aufgabe übernimmt. Dies bedingt natürlich einige Aktionen, bevor das Programm installiert ist. Zuerst wird der nicht belegte Speicher mittels der Prozedur *release_mem* an DOS zurückgegeben. Dies ist erforderlich, da die EXEC-Funktion den gesamten freien Speicher für eine COM-Datei reserviert.

Nach der Eröffnungsmeldung beginnt die eigentliche Installation. Zuerst ist die Adresse des DOS-Critical-Region-Flags zu ermitteln und in lokalen Variablen zu speichern. Dies erfolgt mit dem INT 21-Funktionsaufruf 34H. Ein Aufruf in der TSR-Routine ist nicht möglich, da ja auch die Funktion 34H nicht reentrant ist. Da die lokalen Variablen im Codesegment des TSR-Prozesses liegen müssen, werden zwei Konstanten deklariert (Word1 und Word2). Die Variablen *Dos_flg1* und *Dos_flg2* liegen genau auf diesen beiden Konstanten. Diese etwas merkwürdige Konstruktion wird durch die Implementierung von Pascal erzwungen.

Als nächstes ist die TSR-Routine in einen Interrupt »einzuhängen« damit sie aktiviert werden kann. Hier wird der Timer (INT 8) benutzt, der pro Sekunde zirka 18mal aktiviert

wird. Es bietet sich auch die Möglichkeit, den INT 1C (Timerfolgeinterrupt) zu benutzen. Probleme treten aber auf, wenn der residente Prozeß den »erminate-process-Aufruf benutzt. In diesem Fall ist, abhängig von der BIOS-Implementierung, der PIC (*Peripheral Interrupt Controller*) für weitere Interrupts blockiert. Damit ist das System praktisch »tot« da auch keine Tastatureingaben mehr bearbeitet werden. Wegen der Hardwareabhängigkeit sollte auch darauf verzichtet werden, den PIC selbst zurückzusetzen. Die Benutzung des INT 8 vermeidet diese Probleme, da nach Bearbeitung der TSR-Routine die eigentli»he Service Routine des INT 8 aktiviert wird. Diese setzt dann auch den PIC zurück. Weiterhin ist es unerheblich, ob der INT 8-Vektor durch andere Prozesse modifiziert wurde. Mit dem INT 21-Aufruf 35H (Get-Vektor) liest das Modul den Original-Vektor des INT 8 und speichert das Ergebnis direkt mit dem Aufruf 25H (Set-Vektor) unter dem INT 68. Dieser Interrupt ist für Anwenderprozesse frei. Es können aber auch andere freie Interrupts belegt werden. Eine zweite Möglichkeit besteht darin, den Vektor lokal im Codebereich zu speichern und per CALL-FAR-Aufruf die Routinen zu aktivieren. Dies ist aber nur mit Tricks in Pascal möglich, weshalb die erste Lösung gewählt wurde.

Der zweite Aufruf der Funktion 25H trägt dann die Adresse des TSR-Moduls unter dem INT 8-Vektor ein. Damit wird dieses Modul bei jedem Timerinterrupt aktiviert. Die Addition von 7 Byte zur Offsetadresse des TSR-Moduls hängt mit der ursprünglichen Implementierung von Turbo Pascal zusammen. Der Compiler erzeugt bei jedem Prozedurkopf einen Vorspann, der im TSR-Modul aber nicht gebraucht wird.

Um den Code resident zu installieren, wird der Initialisierungsteil mit dem INT 21-Aufruf 31H (Keep Process) verlassen. Im Register DX steht die Zahl der belegten Paragraphen. Dieser könnte auf die Größe des TSR-Moduls begrenzt werden, was aber hier nicht ausgenutzt wurde.

Der int_handler

Diese Prozedur enthält den eigentlichen residenten Codeteil, der über den INT 8 periodisch aktiviert wird. Da der Timer alle 1/18.2 Sekunden einen Interrupt auslöst, darf die Bearbeitung des Services nicht länger als diese Zeitspanne dauern, da sonst Interrupts verlorengehen (Uhr geht dann nach). Die Implementierung ist prinzipiell in Pascal möglich, aus Effizienzgründen wurde aber Assembler verwendet. Zuerst werden die benutzten Register auf dem Stack gesichert. Dann wird der Zustand des *Keyboard Flags* (Adresse 0:417) überprüft. Falls alle Tasten gedrückt sind, enthält das Byte den Wert 0FH. Ist dies erfüllt, prüft das Modul anschließend den Zustand des DOS-Flags. Nur wenn keine Systemaufrufe in Bearbeitung sind (Flag = 0), kann der Anwenderprozeß terminiert werden. Zuerst ist dann der Stack zu bereinigen. Die Anweisung ADD SP,4 entfernt die Rückkehradresse des unterbrochenen Prozesses vom Stack. Mit POPF wird der Zustand der Flags vor Eintritt in die Interruptroutine restauriert. (Dies ist notwendig, da bei einem Interrupt die Flags und anschließend die CS- und IP-Register auf den Stack gespeichert werden.) Dann wird noch die Original-INT 8-Service-Routine per INT 68 aktiviert. Anschließend wird dann der Anwenderprozeß mittels des Funktionsaufrufes 4CH terminiert. Dabei nutzt der residente Prozeß die Eigenschaft aus, daß die Funktion 4CH intern die Adresse des aktiven PSP speichert, den zugehörigen Prozeß terminiert und den Speicher freigibt. Da aber die Adresse auf den PSP des »hängendenEine Deinstallation des TSR-Moduls wäre durch die Restaurierung des INT 8-Vektors möglich. Mit dem Aufruf 4AH (Set Block) könnte dann der Speicher freigegeben werden.

Falls keine Tasten gedrückt waren oder das DOS-Flag gesetzt ist, verzweigt der Prozeß in den EXIT-Teil. Dieser restauriert die benutzten und vorher gesicherten Register, aktiviert über den INT 68 die Original-INT 8-Service-Routine und beendet die Ausführung mit einer IRET-Anweisung.

Weitere Einzelheiten sind dem Listing zu entnehmen. Auf die Demonstration aller oben besprochenen Möglichkeiten wurde aus Aufwandsgründen verzichtet.

```
program tsr_demo;
{$T+}
{$D+}
{*****************************************************************}
{* Filename:    TSR.PAS                                          *}
{* Autor:       Born G.                                          *}
{* Prog. Spr.:  Quick/Turbo Pascal                        *}
{* Betr. Sys.:  MSDOS 3.x                                        *}
{*                                                               *}
{* Das Programm demonstriert den Umgang mit den TSR-Programmen.  *}
{* Es wird eine Routine installiert, die die Tastatur auf die    *}
{* Tasten Shift + Ctrl + ALT überwacht. In diesem Fall wird      *}
{* ein Systemreset ausgeführt.                                   *}
{*                                                               *}
{* Bei der Übersetzung in Turbo Pascal 3.0 müssen die Optionen:  *}
{*                                                               *}
{*        COM File                                               *}
{*        Min free dynamic memory = stack_size                   *}
{*        Max free dynamic memory = stack_size                   *}
{*                                                               *}
{* des Compilers gesetzt werden. Stack_size ist eine Pro-        *}
{* grammkonstante und wird hier auf 1000 gesetzt. Das vorlie-    *}
{* gende Programm ist in Turbo Pascal 4.0 implementiert. Die     *}
{* Befehle sind mit den Zeichen ### markiert. Der Code der       *}
{* Prozedur Int_handler beginnt ab Ofs()+10, da der Compiler     *}
{* einen recht merkwürdigen Vorspann erzeugt, der in der Doku-   *}
{* mentation nicht beschrieben ist. Ob ein Compilerfehler vor-   *}
{* liegt, konnte nicht geklärt werden. In Turbo Pascal 3.0 ist   *}
{* Ofs()+7 einzutragen. Beachte weiterhin, daß in V3.0 die typi- *}
{* sierten Konstanten im Codesegment liegen ! Nicht alle Pro-    *}
{* gramme lassen sich mit TSR abbrechen. Es gibt weiterhin Unter-*}
{* schiede in der TSR Version mit Turbo Pascal 3.0 die offenbar  *}
{* stabiler arbeitet und z.B. PCTOOLS abbricht, während dies bei *}
{* der vorliegenden Version in Turbo Pascal 4.0 nicht funkt.!!   *}
{*****************************************************************}

{ ********** Declaration Modul *****************}

{$R-}    {### Range checking off}
{$B+}    {### Boolean complete evaluation on}
{$S+}    {### Stack checking on}
{$I+}    {### I/O checking on}
{$N-}    {### No numeric coprocessor}
{$M $1000,200,1000} {### Turbo 3 stack and heap}


Uses Dos;   { ### }

type Long_string  = STRING[80];

     registers = record   { ### }
            AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS: WORD;
     end;

const Stack_size  = $1000;          {* Größe des HEAP und Stack *}
                                    {* für Turbo Pascal 3.0     *}

{ ###  Word1: Integer = 0;          {* Critical Flag            *}
{ ###  Word2: Integer = 0;          {*                          *}

var  reg1 : registers;
     Dos_flg1 : Integer  {absolute Word1 ###};
     Dos_flg2 : integer  {absolute Word2 ###};
     adr      : word;               {* Adresse PSP             *}
{ ********** Hilfsroutinen     *****************}

procedure release_mem;
{
  Freigabe des nicht benötigten Speichers an MS-DOS
  mittels der MS-DOS Funktion Set Block.
  Parameter der INT 21-Funktion 4AH

   AX    ---> 4A00H
   BX    ---> Zahl der benötigten Paragraphen
   ES    ---> Segmentadresse des Speicherblocks

  Bei Fehlern ist anschließend das Carry Flag gesetzt und
  im Register AX findet sich ein Fehlercode. Falls die Zahl
  der angeforderten Paragraphen = FFFFH ist, tritt zwar ein
  Fehlercode auf, aber im Register BX gibt DOS die Zahl der
```

```
  noch freien Paragraphen des größten Speicherblocks zurück.
}

 var register  : registers;

 begin
  with register do
   begin                                  {* Register setzen        *}
    BX := (Sseg - adr) + Stack_size;      {* Zahl der Paragraphen (V 3.0) }
    ES := adr;                            {* Seg. Adr. Codebereich  *}
    AX := $4A00;                          {* MS-DOS Funct. free Mem.*}

    MSDOS (Dos.Registers(register));      {* Int 21 -> free Memory  *}
{
  Prüfung, ob beim Systemaufruf Fehler aufgetreten sind. In
  diesem Fall ist das Carry Flag gesetzt. Das Register AL ent-
  hält den Fehlercode.

  Achtung:  In MS-DOS 3.2 liegt vermutlich ein Bug vor, der bei
            Aufruf aus Subdirectories auch bei erfolgreicher
            Speicherfreigabe das Carrybit setzt. AL enthält damit
            undefinierte Werte.
}

    if (FLAGS and 1) <> 0 then
     begin
      if not (AX in [7,8,9]) then    { Fehler in MS-DOS 3.2  }
        writeln ('Wert Register AX illegal:',AX)
      else
       begin
        case AX of
         7 : writeln('Speicher-Kontroll-Block zerstört');
         8 : writeln('Speicherkapazität zu gering ',BX,
             ' Paragraphen frei');
         9 : writeln('Unzulässige Segmentzuordnung in ES');
        end; { case }
        writeln('Abbruch bei Speicherfreigabe');
        halt;
       end;
     end;
   end;
 end; { release_mem }

 procedure int_handler;

{*********************************************************}
{*  Dies ist die Interrupt Service Procedure, die sich in *}
{*  den Timerfolgeinterrupt einhängt und den Tastatur-    *}
{*  status auf die Tasten abfragt.                        *}
{*********************************************************}

Begin
Inline (
 $50/            (* PUSH  AX        ; Save Register      *)
 $57/            (* PUSH  DI        ;                    *)
 $06/            (* PUSH  ES        ;                    *)
 $FB/            (* STI             ; Inter. freigeben   *)
 $B8/$00/$00/    (* MOV   AX,0000   ; set Adress Key-    *)
 $8E/$C0/        (* MOV   ES,AX     ; boardflag          *)
 $B8/$17/$04/    (* MOV   AX,0417   ;                    *)
 $97/            (* XCHG  DI,AX     ;                    *)
 $26/            (* ES:             ; get Keyboardflag   *)
 $8A/$05/        (* MOV   AL,[DI]   ;                    *)
 $25/$0F/$00/    (* AND   AX,000F   ; mask Bits          *)
 $3D/$0F/$00/    (* CMP   AX,000F   ; Bits = 0F ?        *)
 $75/$23/        (* JNZ   EXIT      ; EXIT Handler       *)
 $3E/            (* DS:             ; (V3.0 = CS: !!!)   *)
 $A1/Dos_flg1/   (* MOV   AX,[Dos_flg1] ; Read DOS Flag  *)
 $8E/$C0/        (* MOV   ES,AX     ;                    *)
 $3E/            (* DS:             ; (V3.0 = CS: !!!)   *)
 $A1/Dos_flg2/   (* MOV   AX,[Dos_flg2] ;                *)
 $97/            (* XCHG  DI,AX     ;                    *)
 $26/            (* ES:             ;                    *)
 $8A/$05/        (* MOV   AL,[DI]   ;                    *)
 $32/$E4/        (* XOR   AH,AH     ; Clear High Byte    *)
 $3D/$00/$00/    (* CMP   AX,0000   ; Flag = 0?          *)
 $75/$0E/        (* JNZ   EXIT      ; Exit Handler       *)
 $07/            (* POP   ES        ; restauriere Re-    *)
 $5F/            (* POP   DI        ; gister und be-     *)
 $58/            (* POP   AX        ; reinige Stack      *)
```

```
 $83/$C4/$04/    (* ADD   SP,04     ;                    *)
 $9D/            (* POPF            ;                    *)
 $CD/$68/        (* INT   68        ; old Int. Service   *)
 $B8/$00/$4C/    (* MOV   AX,4C00   ; terminate Process  *)
 $CD/$21/        (* INT   21        ;                    *)
                 (* EXIT:                                *)
 $07/            (* POP   ES        ; restauriere Re-    *)
 $5F/            (* POP   DI        ; gister             *)
 $58/            (* POP   AX        ;                    *)
 $CD/$68/        (* INT   68        ; old Int. Service   *)
 $CF);           (* IRET                                 *)

end;

{ ***********   Root  Modul     ***************** }

begin

 with reg1 do                              {* ermittle Adr. PSP  *}
  begin
    ax := $6200;                          {* Get PSP Adresse (DOS 3.x  *}
    MSDOS (Dos.Registers(reg1));          {* Aufruf INT 21-Funktion    *}
    adr := bx;                            {* Segmentadresse PSP        *}
  end;

 release_mem;                             {* unbel. Speicher freigeben *}

 writeln('Installation des residenten Programmes Reset');

 with reg1 do
  begin
   AX := $3400;                           {* get Adr. DOS Critical     *}
   msdos(Dos.Registers(reg1));            {* Flag                      *}
   Dos_flg1 := ES;                        {* save Flag Adress          *}
   Dos_flg2 := BX;

   AX := $3508;                           {* Get Timer Inter. Vektor   *}
   msdos(Dos.Registers(reg1));
   AX := $2568;                           {* Int Vektor auf INT 68H    *}
   DS := ES;                              {* umsetzen                  *}
   DX := BX;
   msdos(Dos.Registers(reg1));

   ax := $2508;                           {* Set Adresse INT 08H       *}
   DS := Seg(Int_handler);                {* to Int_handler            *}
   DX := Ofs(Int_handler)+10;             {* ### V3.0 -> Ofs()+7 !!!   *}
   msdos(Dos.Registers(reg1));

   writeln ('Ein Programm läßt sich nun durch drücken der Tasten:');
   writeln ('SHIFT links, SHIFT rechts, ALT, CTRL abbrechen');

   ax := $3100;                           {* TSR Code                  *}
   DX := (Sseg - adr) + Stack_size;       {* belegte Paragraphen       *}
   msdos (Dos.Registers(reg1));           {* terminate resident        *}
 end;
end.
```

11.9 BIOS-Aufrufe

Nachfolgend soll noch ein kleines Programm vorgestellt werden, welches die Benutzung einiger BIOS- und DOS-Funktionsaufrufe demonstriert. Zweck des Programmes ist es, über die Aufrufe die Hardwarekonfiguration des PC festzustellen und auf dem Bildschirm auszugeben. Dabei wird im wesentlichen der INT 11 (Get-Konfiguration) benutzt. Zur Ermittlung der Speichergröße existiert eine eigene Routine, da nicht alle BIOS-Implementierungen den korrekten Wert zurückgeben. Nachfolgend sollen die einzelnen Module kurz erläutert werden.

Write_hex

Dieses Hilfsmodul dient zur Ausgabe eines Wertes in hexadezimaler Notation. Dabei läßt sich über den Parameter *len* steuern, ob der Wert als Byte oder Word ausgegeben wird.

rom_check

Dieses Modul wird aufgerufen, falls der RAM-Test keine beschreibbare Zelle findet. Er prüft, ob sich in diesem Adreßraum ein ROM befindet und stellt gegebenenfalls die Größe des Bereiches fest.

Memo_check

Diese kleine Routine prüft, ob sich die angegebene Adresse im RAM-Bereich befindet, also beschreibbar ist.

memo_size

Das Modul ermittelt Lage und Größe des RAM-Bereiches sowie der BIOS-ROMs. Dies ist erforderlich, da nicht alle BIOS-Implementierungen über den INT 12 die korrekte RAM-Größe zurückgeben. Weiterhin findet diese Routine auch Lage und Größe des Bildschirmspeichers, was bei der BIOS-Abfrage nicht funktioniert.

bios_chk

Aufgabe dieser Routine ist es, die BIOS-Copyright-Meldung zu lesen. Bei PCs mit 16-Bit-Bus ist diese Information oft zweimal pro Wort vorhanden, wodurch die Buchstaben doppelt erscheinen.

dos_ver

Diese Prozedur ermittelt über den INT 21-Aufruf 30H die Versionsnummer des benutzten MS-DOS-Betriebssystems.

sys_config

Mit dieser Prozedur wird die Hardwarekonfiguration des PC ermittelt. Dazu dient einmal der BIOS-INT 11, der Auskunft über die Zahl der Diskettenlaufwerke, Schnittstellen und Grafikkarten gibt. Sind Festplatten im System, wird deren Kapazität sowie die Aufteilung in Sektoren und Cluster über die INT 21-Funktion 1CH ermittelt. Die INT 21-Funktion 36H gibt weiterhin die freie Kapazität auf der Platte an.

Im Hauptmodul werden die einzelnen Hilfsmodule aufgerufen, um die Konfiguration zu ermitteln und anzuzeigen. Die Einzelheiten sind nachfolgendem Listing zu entnehmen.

```
program hwinfo;

{*************************************************************}
{* Filename:    HWINFO.PAS                                   *}
{* Autor:       Born G.                                      *}
{* Prog. Spr.:  Quick/Turbo Pascal V 4.0  (V 3.0)                  *}
{* Betr. Sys.:  MSDOS 3.x                                    *}
{*                                                           *}
{* Das Programm demonstriert den Umgang mit den DOS- und BIOS-  *}
{* Systemaufrufen. Es stellt die Hardwarekonfiguration des PCs  *}
{* fest und gibt diese auf dem Bildschirm aus. Das Programm ist *}
{* für Turbo Pascal 4.0 geschrieben. Die Änderungen zu Turbo    *}
{* Pascal 3.0 sind mit ### markiert.                         *}
{*************************************************************}

{ ********** Declaration Modul *****************}

{$R-}    {Range checking off ###}
{$B+}    {Boolean complete evaluation on ###}
{$S+}    {Stack checking on ###}
{$I+}    {I/O checking on ###}
{$N-}    {No numeric coprocessor ###}
{$M 65500,16384,655360} {Turbo 3 default stack and heap ###}


{**$P256 *}                        {* DOS I/O erlauben ### *}


Uses Dos;                          {* ###  DOS Unit        *}

type registers = record            {* ###                  *}
        case integer of
          1: (AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS: WORD);
          2: (AL, AH, BL, BH, CL, CH, DL, DH           : BYTE);
     end;

const Byte_len = 1;
      Word_len = 2;

var
    register     : registers;      {* 8086 Registerfile        *}


{ ********** Hilfsroutinen     *****************}

procedure Write_hex (value, len : integer);
 {
   Ausgabe eines Wertes als Zahl auf dem Bildschirm.
   Durch Len wird festgelegt, ob ein Byte (Len = 1)
   oder Wort (Len = 2) ausgegeben werden soll.
  }
 const Hexzif : array [0..15] of char = '0123456789ABCDEF';

 VAR temp : integer;

 begin
{
               High  Low  Byte
   Format :    xxxx  xxxx      Hex -> xx = 4 Bit pro Ziffer
}
  if len = Word_len then
   begin
    temp := swap (value) and $0FF;      { high byte holen     }
    write (Hexzif[temp div 16]:1,Hexzif[temp mod 16]:1);
   end;
  temp := value and $0FF;               { low byte holen      }
  write (Hexzif[temp div 16]:1,Hexzif[temp mod 16]:1);
end;  { Write_hex }

function rom_check (adr : integer) : boolean;
{
  Dieses Modul wird aufgerufen, falls der Ram Test kein RAM
  zeigt. Prüfe, ob der Wert der Adresse <> FFH ist. In
  diesem Fall wird angenommen, daß ein BIOS Bereich vorliegt.
}
 var
  seg  : integer;
 begin
  seg := swap(adr);                 { xx00:0 = adresse    }
  if mem [seg:10] <> $FF then
```

```
   rom_check := true
  else
   rom_check := false;             { rom fail            }
 end;  { rom_check }

function memo_check (adr : integer) : boolean;
{
  Prüfe, ob sich die Adresse beschreiben läßt. Falls die Zelle
  während der Abfrage verändert wird, kann ein Systemabsturz
  auftreten. Daher wird der Interrupt gesperrt.
}
 var
  temp : integer;
  seg  : integer;
 begin
  seg := swap(adr);               { xx00:0 = adresse    }
  inline ($FA);                   { CLI                 }
  temp := mem [seg:1];            { save value          }
  mem [seg:1] := $55;             { test pattern write  }
  if mem [seg:1] = $55 then
   memo_check := true
  else
   memo_check := false;           { ram fail            }
  mem [seg:1] := temp;            { restore value       }
  inline ($FB);                   { STI                 }
 end;  { memo_check }

procedure memo_size;
{
   Da der BIOS INT 12 (get memory size) nicht bei allen
   PCs funktioniert, wird der Speicher (1 Mbyte) in
   4-Kbyte-Schritten getestet. Spiegelungen werden nicht
   berücksichtigt.
}
 const
  byte_len = 1;

 var
  size : integer;
  i    : integer;
  memo : integer;

 begin
  writeln;
  i := 0;
  repeat                           { loop 256 * 4 Kbyte }
   size := 0;                      { init variables  }
   memo := i;
   while memo_check(i) and (i < $100) do { test memory     }
    begin
     size := size + 4;             { add 4 Kbyte     }
     i := i + 1;                   { next segment    }
    end;
   if size > 0 then
    begin                          { RAM area found  }
     write ('RAM area from       : ');
     write_hex (memo,byte_len);    { RAM Startaddress }
     writeln ('00:0 len ',size,' kByte');
    end
    else
    begin                          { RAM fail -> ROM ? }
     while rom_check(i) and (i < $100) do { test memory     }
      begin
       size := size + 4;           { add 4 Kbyte     }
       i := i + 1;                 { next segment    }
      end;
     if size > 0 then
      begin                        { ROM area found  }
       write ('BIOS found from     : ');
       write_hex (memo,byte_len);  { RAM Startaddress }
       writeln ('00:0 len ',size,' kByte');
      end;
    end;
   i := i + 1;
  until i > $FF;
end;


procedure bios_chk;
{
```

```
  ROM Copyright lesen und ausgeben
}
  var rom_txt : string[60];
      i    : integer;
 begin
  rom_txt := ' ';
  for i := 0 to 50 do
   rom_txt := rom_txt + chr(mem [$F800:3+i]);
  writeln;
  writeln('Bios : ',rom_txt);
end; { Bios_chk}

procedure dos_vers;
{
  Lese DOS Versionsnummer über INT 21
  Code $30
}
 begin
  with register do
   begin
    AX := $3000;                      { get MS DOS Vers.  }
    msdos (Dos.Registers(register));
    writeln;
    if AL = 0 then
     writeln ('MS - DOS Version  1.x')
    else
     write ('MS - DOS Version ',AL:1,'.',AH:1);
   end;
  writeln;
 end;   { dos_vers }

procedure sys_config;
{
  Stelle über den Bios Int 11 die Hardwareconfiguration
  fest
}
 var drive : integer;              { Zahl der Floppys }
     h_disk: integer;              { Nr. Harddisk     }
     displ : integer;              { Display Typ      }
     temp  : integer;
     i     : integer;
     bytes : real;
     frei  : real;

begin

 intr ($11,Dos.Registers(register));  { Hardware test    }
 temp := register.AX;              { get configration }

 if (temp and 1) <> 0 then
  begin                            { disc drive found }
   drive := (temp and $C0) div 64; { count drives     }
   writeln;
   write ('Floppy Disks       ');
   for i := 0 to drive do
    write (chr($41+i),':  ');         { disc name       }
   writeln;
  end
 else
  begin
   drive := 0;
   writeln ('No disk drives');
  end;
{
  Prüfe ob Harddisks im System vorhanden sind
}
  writeln;
  h_disk := drive+1;
  if h_disk < 3 then  h_disk := 3;    { Disk ab C:       }
  for i :=  h_disk to h_disk + 2 do   { max 3. Harddisk  }
   with register do
    begin
     AX := $1C00;                     { get drive data     }
     DX := I;                         { drive Nr.          }
     msdos (Dos.Registers(register)); { MSDOS Call         }
     if (AX and $FF) <> $FF then
      begin                           { drive found        }
       bytes := (AX and $FF) * CX;    { calc. storage space}
       bytes := bytes * DX / 1.0E6;
       if mem [DS:BX] = $F8 then
```

```
        write ('Harddisk       ')
       else
        write ('Floppydisk     ');
        writeln ('    ',chr($40+i),': ',bytes:5:2,' MB');
        writeln ('Sectors per cluster : ',(AX and $FF));
        writeln ('Bytes per sector    : ',CX);
        writeln ('Clusters            : ',DX);

        AX := $3600;                  { get free disk space }
        DX := I;                      { drive Nr.           }
        MSDOS (Dos.Registers(register));

        frei := AX * CX;              { calc free space    }
        frei := frei * BX / 1.0E6;
        write ('Free space          : ',frei:5:2,' MB (');
       writeln ((frei * 100.0 /bytes):3:1,' %)');
      end;
    end;
{
  Anzeige der weiteren Information aus dem BIOS Aufruf
}

 displ := (temp and $30) div 16;
 writeln;
 case displ of
   0: writeln ('Illegal display mode');
   1: writeln ('Colorcard 40 x 25 monochrom');
   2: writeln ('Colorcard 80 x 25 monochrom');
   3: writeln ('Monochromcard 80 x 25');
  end;

 temp := swap(temp);
 writeln;
 writeln ('Serial   interfaces : ',(temp and $0E) div 2);
 writeln ('Parallel interfaces : ',(temp and $C0) div 64);

end;  { sys_config }


{ ***********   Root  Modul     *************** }


begin

 writeln;
 writeln ('Hardware  I N F O             (c) Born   Version 1.0');
 writeln;

 bios_chk;                              {* Bios Copyright lesen      *}

 dos_vers;                              {* Dos Versionsnummer        *}

 memo_size;                             {* Ram Bereiche ermitteln    *}

 sys_config;                            {* Schnittstellen bestimmen  *}

 writeln;
 writeln ('Ende Hardware I N F O');

end.
```

12 DOS-Speichererweiterungen

In MS-DOS (PC-DOS) wird nach wie vor der 1 Mbyte große Speicherbereich der 8088/8086-Prozessoren zu Grunde gelegt. Die Aufteilung in 640-Kbyte-RAM und den Bereich für die Grafikkarten, BIOS-ROMs, etc. wurde in Kapitel 1 (Bild 1.6) behandelt. Nun werden mittlerweile aber PCs mit 80286/80386- und 80486-Prozessoren ausgeliefert. Diese CPUs erlauben die Adressierung von mehr als 1 Mbyte Speicher und die Datenblätter neuerer PCs weisen häufig einen Speicherausbau von 1 bis 2 Mbyte und mehr auf. Der RAM-Bereich oberhalb der 640-Kbyte-Grenze wird in der Regel nur als Plattencache oder als virtuelle RAM-Disk genutzt. Offensichtlich gibt es aber Techniken, um auch unter DOS bei den 80286/80386-Prozessoren mehr als 1-Mbyte-RAM zu nutzen. Schlagworte wie:

»Expanded Memory« »HIMEM« »HMA« »XMS**Fehler! Verweisquelle konnte nicht gefunden werden.**Extended Memory« _

werden in diesem Zusammenhang häufig gehandelt. Da die Bedeutung dieser Begriffe nicht allgemein bekannt ist, sollen die Grundlagen nachfolgend etwas eingehender behandelt werden.

DOS-Speicherarten

Der Speicher eines PC läßt sich in verschiedene Bereiche aufteilen, die mit bestimmten Begriffen belegt werden. Die folgenden Abschnitte enthalten eine kurze Einführung in die unterschiedlichen Speicherarten.

Konventioneller Speicher

Dies ist der DOS-Grundspeicher von 640 Kbyte der ab Adresse 0 beginnt. Hier legt das Betriebssystem seinen Code und interne Daten ab. Weiterhin sind in diesen Bereich auch die Anwendungsprogramme zu laden.

Upper Memory Block (UMB)

Der Bereich oberhalb 640 Kbyte bis 1 Mbyte wird allgemein für die Adapterkarten (Graphikkarten, Plattencontroller) und die BIOS-Programme reserviert. In den meisten Systemen weist dieser 384 Kbyte große Adressbereich jedoch Lücken auf, in denen keine Adapter liegen. Diese Lücken lassen, insbesondere bei Rechnern auf 80386-Basis, mit verschiedenen Techniken nutzen. Eine dieser Lücken wird zum Beispiel durch den EMS-Manager genutzt, um die einzelnen Speicherseiten in einem 64-Kbyte-Fenster einzublenden. Auf 80386-Systemen besteht weiterhin die Möglichkeit in die Lücken zwischen den Adaptern RAM einzublenden (Shadow-RAM und Backfilling). Dann lassen sich zumindest in diesen Bereichen Treiber und residente Programme unterbringen. Die Verwaltung des Speicherbereiches erfolgt durch den HIMEM.SYS-Treiber. Der Speicherbereich selbst wird als Upper-Memory-Block (UMB) bezeichnet. DOS 5.0 und DOS 6.0 unterstützen die Benutzung dieses Speichers zur Auslagerung von Betriebssystemteilen. In älteren DOS-Versionen läßt sich der Bereich zum Beispiel durch

Programme wie QEMM der Firma Qarterdeck nutzen. Von diesem Programm wurde die UMB-Unterstützung durch DOS abgeleitet. Durch Nutzung dieses Bereichs kann der freie konventionelle 640-Kbyte-Speicherbereich vergrößert werden, d.h. den Applikationen steht dieser Speicher zur Verfügung.

Extended Memory (XMS)

Konnten die 8088/8086-Prozessoren der ersten PC's nur 1 Mbyte adressieren, änderte sich dies bei den Nachfolgern. Rechner auf 80286-, 80386- oder 80486-Basis sind in der Lage, Speicherbereiche von 16 Mbyte bis zu 4 Gbyte zu verwalten. Mittlerweile werden die meisten PC's auch mit mindestens 1 Mbyte, meist sogar mit 2 oder 4-Mbyte-Speicher ausgestattet. Der Speicher oberhalb der 1 Mbyte-Grenze ist durch DOS normalerweise jedoch nicht erreichbar und wird als *Extended Memory* bezeichnet. Es gibt aber Möglichkeiten und Tricks in diesem Bereich Daten und Programme abzulegen und zu bearbeiten. WINDOWS nutzt zum Beispiel diesen erweiterten Speicherbereich. Aber auch RAM-Disks und andere Hilfsprogramme können Teile dieses Speichers belegen. Zur Verwaltung dieses Bereiches wird der XMA-Manager benutzt. Die Programmschnittstelle wird von Microsoft durch den Treiber HIMEM.SYS realisiert. Dieses Programm ist ab DOS 4.0 und in WINDOWS im Lieferumfang vorhanden. Nur falls dieses Programm beim Systemstart in CONFIG.SYS eingebunden wird, können mehrere Programme auf den erweiterten Speicher unter DOS zugreifen.

Expanded Memory (EMS)

Extended Memory steht nur auf Maschinen ab dem 80286-Prozessor zur Verfügung. Weiterhin kann unter DOS nur mit Tricks auf Speicherbereiche oberhalb 1 Mbyte zugegriffen werden. Deshalb waren die Entwickler bereits frühzeitig bemüht, in diesem 1 Mbyte großen Adressbereich zusätzlichen Speicher bereitzustellen. Aus den Tagen des CP/M-Betriebssystemes wurde eine Lösung an DOS angepaßt. Die Idee beruht darauf, daß im 1-Mbyte-Speicherbereich ein Fenster mit einer Größe von 64 Kbyte eingeblendet wird. Dieses Fenster liegt meist im reservierten 384-Kbyte-Bereich in einer Lücke zwischen den Adaptern. Das Fenster selbst wird nochmals in Segmente zu 16 Kbyte unterteilt. Mit Hilfe einer geeigneten Hardware (z.B. EMS-Boards) lassen sich in diese Segmente dann unterschiedliche Speicherseiten aus einem maximal 32 Mbyte großen Speicherblock einblenden. Für DOS ist dann nur ein maximal 64-Kbyte-Bereich innerhalb des Fensters zugreifbar. Aber über ein geeignetes Verwaltungsprogramm lassen sich andere Speicherausschnitte in das Fenster einblenden. Das Verwaltungsprogramm wird als Expanded-Memory-Manager (EMS) bezeichnet. Mittlerweile unterstützen viele Programme den EMS-Speicher und lagern Programmteile oder Daten in diesem Bereich aus. Die Anforderung verschiedener Speicherbereiche über den EMS-Manager geht relativ schnell von statten, so daß sich die Methode durchgesetzt hat. Besitzer von 80386/80486-System haben meist über Emulationsprogramme (EMM386 Expanded Memory Emulator) die Möglichkeit, einen Teil des Extended Memory als Expanded Memory zur Verfügung zu stellen.

Der High-Memory-Bereich (HMA)

Die ersten 64 Kbyte innerhalb des Extended-Memory-Bereiches werden auch häufig als High-Memory-Area (HMA) bezeichnet. Die Verwaltung erfolgt ebenfalls über den XMS-Manager (HIMEM.SYS). Die herausragende Stellung dieses Speichersbereiches ergibt sich aus der Tatsache, daß sich dieser Bereich auch unter DOS ansprechen läßt, obwohl er oberhalb der 1-Mbyte-Grenze liegt. Voraussetzung ist mindestens ein 80286-Prozessor und der HIMEM.SYS-Treiber. Der Trick liegt darin, daß die 80x86-Prozessoren nicht ganz kompatibel zu den 8086-Vorgängern sind. Dadurch lassen sich mit der DOS-Adresseinstellung auch die ersten 64 Kbyte des Extended Memory erreichen. Genaueres hierzu findet sich in /1/. Mit DOS 5.0 und DOS 6.0, sowie Memory-Managern wie QEMM lassen sich ebenfalls Teile des Betriebssystemes in diesem HMA-Bereich auslagern. Weitere Ausführungen zu diesem Punkt sind auf den folgenden Seiten zu finden.

Nach dieser Kurzeinführung möchte ich nun die Möglichkeiten unter DOS zur Verwaltung dieser Speicherbereiche vorstellen.

12.1 Extended-Memory-Zugriffe (per INT 15)

Wie bereits erläutert, besitzen die 80x86-Prozessoren einen erheblich größeren Adreßraum als unter DOS verwaltet wird. Beim 80286 sind dies im Protected Mode immerhin 16 Mbyte und beim 80386 bis zu 4 Gbyte. Unter DOS ist der Adreßraum dann wie folgt belegt:

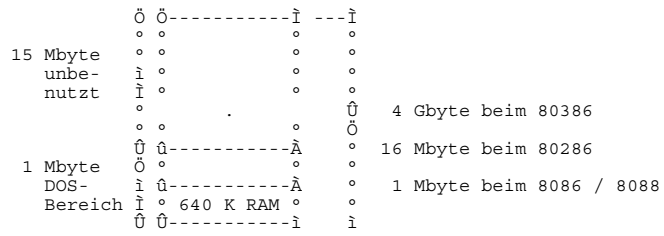


Bild 12.1: Aufteilung des 80x86-Adreßraumes unter DOS

Dies bedeutet, unter DOS werden alle Prozessoren im *Real Mode* mit 1-Mbyte-Adreßraum betrieben. Dieser Real Mode emuliert die 8086-CPU. Bei Maschinen mit 80286- oder 80386-Prozessor läßt sich aber ein größerer Speicherbereich durch den Chip adressieren. Viele Hersteller von PCs statten ihre Systeme bereits mit 1 oder 2 Mbyte aus, um auch Betriebssysteme (OS/2 und Unix) zu unterstützen. Gute Rechner lassen sich auf der Hauptplatine bis auf 8- oder 16-Mbyte-RAM hochrüsten. Leider müssen sich diese Systeme bei DOS den dort geltenden Speicherkonventionen anpassen. Der Bereich zwischen 0 und 640 Kbyte ist für das normale DOS-RAM reserviert. Ab 640 Kbyte bis 1 Mbyte finden sich die Grafikadapter, Netzwerkkarten und die BIOS-ROMs von Festplatten und dem System. Das bedeutet andererseits, daß die restlichen 384 Kbyte, bei Systemen mit 1-Mbyte-RAM, nicht im 1-Mbyte-Adreßraum abgebildet werden dürfen. In der Regel wird deshalb der restliche RAM-Bereich in den Adreßraum oberhalb 1 Mbyte gelegt.

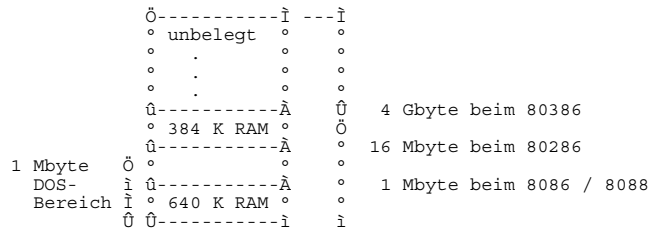


Bild 12.2: Belegung des restlichen 80x86-Adreßraumes unter DOS

Dieser Bereich oberhalb FFFF:000F wird bei den 80286/80386-Prozessoren als *Extended Memory* bezeichnet.

Eine Nutzung ist von DOS aber nicht vorgesehen, da alle Adressen im Real Mode auf 1 Mbyte begrenzt sind. In der Regel benutzen RAM-Disks (z.B. VDISK.SYS) diesen Speicher. Um den Protected Mode zu umgehen, greift das Programm allerdings zu einem Trick. Es müssen ja lediglich Daten zwischen den DOS-Puffern im 640-Kbyte-RAM-Bereich und der RAM-Disk im Extended Memory ausgetauscht werden. Hierfür läßt sich der DMA-Controller benutzen, der im 80286-System auf den gesamten 16-Mbyte-Adreßraum zugreift. VDISK programmiert den Controller so, daß er Daten zwischen den Speicherbereichen transferiert.

Eine andere Möglichkeit besteht in der Nutzung der INT 15-Funktion des BIOS, die einen Datentransfer zwischen DOS und dem Extended Memory erlaubt. Allerdings muß die CPU kurzzeitig in den Protected Mode geschaltet werden. In diesem Zustand sind die Interrupts blockiert, so daß die Systemuhr nicht mehr korrekt arbeitet. Auch der Datenempfang über die serielle Schnittstelle wird beeinflußt. Jeder Datentransfer benötigt eine gewisse Zeit, die durch die Zahl der Datenbytes beeinflußt wird. Deshalb sollte die Zahl der zu transferierenden Bytes bestimmte Grenzen nicht überschreiten, um obige Nachteile zu verhindern.

Bytes	Zeitdauer (AT 03) in ms
512	0.98 - 1.13
64 K	37 - 72

Tabelle 12.1: Zeitdauer eines MOVE-Befehls

Ein Transfer von 512 Byte zwischen DOS und dem Extended Memory mittels der INT 15-Funktion des BIOS benötigt bei einem AT 03 mindestens 0,98 Millisekunden. Bei einem Transfer zwischen geraden und ungeraden Adressen kann diese Zeit auf 1,04 Millisekunden steigen, während bei einem Transfer zwischen ungeraden Adressen 1,13 Millisekunden vergehen. Zum Vergleich sei darauf hingewiesen, daß bei einer Kommunikation mit 9600 Baud und 11 Datenbits alle 1,04 Millisekunden ein Zeichen empfangen wird und damit ein Interrupt auftritt. Bei Benutzung des MOVE-Block-Befehls des INT 15 treten dann bereits Probleme auf. Noch krasser wird das Verhältnis, wenn die Zahl der zu transferierenden Bytes gegen 64 Kbyte geht. Ein Blocktransfer sollte deshalb nicht zusammen mit einer laufenden Datenkommunikation betrieben werden.

Für den Zugriff auf das Extended Memory bietet der INT 15 des BIOS folgende Funktionen:

Move Block (AH = 87H)

Die Funktion 87H erlaubt den Datenaustausch zwischen MS-DOS und dem Extended-Memory-Bereich. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 15            º
º                               º
º AH: 87H (Move Block)          º
º CX: Zahl der Worte im Puffer  º
º ES:SI Adresse Descriptor Tab. º
û-------------------------------À
º      RETURN                   º
º CY: 0 kein Fehler             º
º CY: 1 Fehler                  º
º AH: Status                    º
Û-------------------------------ì
```

Im Register CX wird die Zahl der zu transferierenden Worte spezifiziert. Es läßt sich immer nur ein Bereich von maximal 64 Kbyte (32 Kworte) transferieren, wodurch der maximale Wert in CX auf 8000H begrenzt ist.

Das Registerpaar ES:DI zeigt auf eine Tabelle mit Steuerparametern für den Transfer. Da die Speicherbereiche oberhalb der 1-Mbyte-Grenze liegen, erfolgt der Transfer über 24 Bit Adressen im *Protected Mode* des 80286-Prozessors. Die Tabelle mit den Steuerparametern (Descriptor-Tabelle) enthält Einträge für die Quelle und das Ziel und wurde beim INT 15 in Kapitel 2 beschrieben.

Die Segmentgröße muß kleiner als 64 Kbyte sein. Im Feld »Zugriffsrecht**Fehler! Verweisquelle konnte nicht gefunden werden.**91H nur lesen
93H schreiben und lesen

abgespeichert.

Der Sourcedescriptor findet sich ab Offset 10H und besitzt den in Tabelle 2.10 beschriebenen Aufbau. Ab Offset 18H findet sich der Descriptor für die Zieladresse mit der gleichen Struktur.

Die restlichen zwei Descriptoren sind vom Anwender mit den Werten 00H zu füllen. Der erste dieser Descriptoren findet sich ab Offset 20H und dient zur Aufnahme eines »virtuellen**Fehler! Verweisquelle konnte nicht gefunden werden.**Nach dem Aufruf zeigt das Carry-Flag den Fehlerstatus an. Ist es gesetzt, findet sich im AH-Register ein Statuscode mit der Kodierung aus Tabelle 2.12.

Die Funktion sichert alle Registerinhalte bis auf das AX-Register. Es lassen sich also bestimmte Datensegmente im 16-Mbyte-Adreßbereich transferieren. Die Größe eines einzelnen Segments darf dabei 64 Kbyte nicht überschreiten. Quell- und Zielsegment müssen eine Mindestgröße von 2 * CX - 1 Byte aufweisen, da sonst Daten verlorengehen.

Während des Transfers ist das komplette Interruptsystem wegen der in Kapitel 1 beschriebenen Inkompatibilitäten mit den Intel Spezifikationen gesperrt. Die Interrupt-Descriptor-Table für den *Protected Mode* der 80286-CPU findet sich komplett im BIOS-ROM.

Es ist zu beachten, daß eventuell die Uhrzeit durch das gesperrte Interruptsystem nachgeht. Der Abgleich ist Aufgabe des Anwenderprogrammes.

Extended Memory Size (AH = 88H)

Dieser Aufruf des INT 15 erlaubt, die Größe eines erweiterten Speicherbereiches (Extended Memory) im Bereich oberhalb von 1 Mbyte festzustellen. Es gelten folgende Aufrufkonventionen:

```
Ö--------------------------------Ì
°      CALL:  INT 15             °
°                                °
° AH: 88H (Extend. Memory Size)  °
û--------------------------------À
°      RETURN                    °
° AX: Speichergröße in Kbyte     °
Û--------------------------------ì
```

Die Speichergröße oberhalb 1 Mbyte wird beim Systemstart ermittelt und ab Adresse 20H und 31H im CMOS-RAM des Uhrenbausteins abgelegt. Dieser Extended-Memory-Bereich kann aber erst genutzt werden, falls das Gerät mit mindestens 512 Kbyte im Grundbereich ausgerüstet ist.

Der Extended-Memory-Bereich läßt sich nur zur Speicherung von Daten nutzen, da DOS den Protected Mode nicht unterstützt. Weitere Informationen über die Funktion des INT 15 sind in Kapitel 2 zu finden.

Bei einigen 80386-Systemen (z.B. mit dem NEAT Chipsatz) gibt es Softwaretreiber, die den Extended-Memory-Bereich für DOS als Expanded-Memory zur Verfügung stellen.

12.2 High Memory Area (HMA)

Als Manko erweist sich die Tatsache, daß der Extended-Memory-Bereich über das BIOS anzusprechen ist. Damit sind Zugriffe von der BIOS-Implementierung abhängig. Daten müssen immer zwischen Extended-Memory und den normalen 640 Kbyte verschoben werden. Die Umschaltung der CPU von DOS in den Protected Mode und die anschließende Rückkehr in den Real Mode lassen sich nur mit Tricks erreichen. So setzt die CPU ein Reset-Flag im CMOS-Uhrenbaustein und veranlaßt den 8048-Tastaturprozessor, einen Reset-Befehl auszuführen. Dieser Vorgang ist recht zeitintensiv (ca. 6 ms), was bei häufigen Zugriffen auf den Extended-Memory-Bereich reichlich Probleme bringt.

Nun gibt es aber bei den 80286-Prozessoren und deren Nachfolgern einen Trick, das letzte Segment des Real-Mode-Adreßsbereiches mit der Segmentadresse FFFFH als RAM zu nutzen. Das RAM liegt auf den Adressen:

```
FFFF:0000H
  ..
FFFF:FFFFH
```

und befindet sich damit, abgesehen von den ersten 16 Byte, oberhalb des DOS-1-Mbyte-Bereiches. Der Speicher wird deshalb als High-Memory-Area (HMA) bezeichnet.

Bei den 8086-CPUs läßt sich dieser Adreßsbereich nicht ansprechen, da nur 20 Bit zur Verfügung stehen. Alle Adressen größer als FFFF:000FH führen zu einem Überlauf der

Register CS:IP, was zu einem Systemabsturz führt. Bei den 80286-CPUs sieht die Situation aber anders aus. Die Chips verfügen über das benötigte Bit 21, so daß sich Adressen zwischen FFFF:0010H bis FFFF:FFFFH als RAM ansprechen lassen. Allerdings muß die Adreßleitung A20 (Bit 21) freigegeben werden, da diese unter DOS blockiert ist (Kompatibilität zum Real Mode). Diese Situation wird durch Bild 12.3 verdeutlicht.

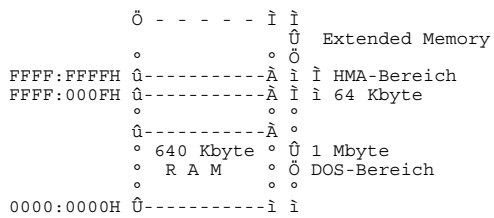


Bild 12.3: Lage des HMA-Bereiches

Dieses Bit A20 der Adreßleitung läßt sich in der Regel bei 80x86-Systemen durch den Tastaturprozessor (8042) blockieren oder zuschalten. Wird die Leitung freigegeben, stehen im Segment FFFFH weitere 65520 Byte (64 Kbyte-16 Byte) RAM zur Verfügung. 16 Byte sind abzuziehen, da diese noch im oberen ROM-Bereich des DOS-Speichers liegen.

Die Firmen Lotus, Intel, Microsoft und AST haben nun einen Standard definiert, der unter anderem den HMA-Bereich aus Sicht des Anwenderprogrammierers zugänglich macht und gleichzeitig die Verwaltung übernimmt. Dieser Standard wird als DOS-Extended-Memory-Specification (XMS) bezeichnet. Die Funktionen selbst werden durch den Extended-Memory-Manager (EMM) abgewickelt. Der ab DOS 4.01 mitgelieferte Treiber HIMEM.SYS unterstützt zum Beispiel die XMS-Funktionen. Auch in Windows ist der Treiber vorhanden.

12.3 Die XMS-Schnittstelle

Nachfolgend wird die Softwareschnittstelle des Treibers beschrieben.

Kommunikation with XMS (INT 2F, AH = 43H)

Der Funktionsdispatcher wird nicht über einen Interrupt, sondern über einen CALL-FAR-Befehl aktiviert. Allerdings muß der INT 2F zur Prüfung des Installationsstatus und zur Ermittlung der Dispatcheradresse benutzt werden. Die Funktion 43H des INT 2F bietet folgende Aufrufe:

Get Installation Status (AL = 00H)

Zuerst ist zu prüfen, ob der XMS-Treiber installiert ist. Hierzu gilt folgende Aufrufschnittstelle:

```
Ö------------------------------Ì
º      CALL:  INT 2F           º
º                              º
º AH: 43H (XMS Treiber)        º
º AL: 00H (Get Installation)   º
û------------------------------À
º      RETURN                  º
º AL: Status                   º
Û------------------------------ì
```

Der Treiber gibt im Register AL den Status zurück. Allerdings hält er sich nicht an die Konventionen des INT 2FH. Ist der Wert gleich 80H, ist der XMS-Treiber installiert. Alle anderen zurückgegebenen Werte signalisieren, daß XMS nicht vorhanden ist. Der Treiber wird in CONFIG.SYS durch die Anweisung

```
DEVICE = HIMEM.SYS
```

beim Systemstart aktiviert.

Get Driver Address (AL = 10H)

Mit diesem Aufruf läßt sich die Adresse des XMM-Dispatchers (XMM = Extended Memory Manager) ermitteln. Es gilt folgende Aufrufschnittstelle:

```
Ö------------------------------Ì
º      CALL:  INT 2F           º
º                              º
º AH: 43H (XMS Treiber)        º
º AL: 10H (Get Address)        º
û------------------------------À
º      RETURN                  º
º ES:BX XMM-Address            º
Û------------------------------ì
```

Der Treiber gibt im Registerpaar ES:BX die Adresse des Funktionsdispatchers zurück. Über diese Adresse lassen sich die folgenden Funktionen aktivieren.

12.3.1 Die XMS-Funktionen

Der XMS-Treiber wird über einen CALL-FAR-Aufruf aktiviert, wobei der Wert in AH die Funktion des Dispatchers selektiert. Die CALL-Adresse läßt sich über den INT 2F ermitteln. Der Treiber stellt folgende CALL FAR-Routinen zur Verfügung:

Get XMS-Version Number (AH = 00H)

Mit diesem Aufruf läßt sich die Version des XMS-Treibers abfragen.

```
Ö------------------------------Ì
º      CALL:  XMS              º
º                              º
º AH: 00H (XMS-Version)        º
û------------------------------À
º      RETURN                  º
º AX: XMS-Version              º
º BX: interne Revision Nummer  º
º DX: Status                   º
Û------------------------------ì
```

Im Register AX wird die Versionsnummer des XMS-Treibers als BCD-Zahl zurückgegeben. BX enthält eine interne Revisionsnummer. Der XMS-Treiber sollte eine

Version größer 2.06 besitzen, um eine einwandfreie Verwaltung des HMA zu gewährleisten. In DX steht der Status des Aufrufes:

```
DX = 0001H : HMA existiert ab 1 Mbyte
     0000H : kein HMA vorhanden
```

Die Statusmeldung 0000H kommt immer dann zurück, wenn zwar ein Treiber geladen ist, aber der PC keinen 64-Kbyte-RAM-Bereich oberhalb 1 Mbyte besitzt.

Request High Memory Area (AH = 01H)

Mit dieser Funktion läßt sich ein Speicherbereich in der HMA anfordern. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 01H (Request HMA)          º
º DX: Speicherbedarf in Byte     º
û--------------------------------À
º      RETURN                    º
º AX: Status                     º
º BL: Errorcode                  º
Û--------------------------------ì
```

Im Register DX wird die Größe des benötigten Speicherbereiches übergeben. Hier unterscheidet man allerdings zwischen Treibern und TSR-Programmen sowie Anwenderprogrammen. Für TSR-Programme und Treiber kann die benötigte Speichergröße in Byte übergeben werden.

Soll der Bereich für Anwenderprogramme unter DOS benutzt werden, ist der Wert FFFFH in DX zu übergeben. Dann wird das komplette HMA-RAM in einem Stück belegt. Der XMS-Treiber versucht den angeforderten Bereich im Adreßbereich von FFFF:000F bis FFFF:FFFFH (HMA) zu reservieren. Die angeforderte Blockgröße (DX) wird mit dem Parameter /HMAMIN des XMS-Treibers verglichen. Eine Freigabe erfolgt nur, falls:

```
DX > /HMAMIN
```

ist. Dadurch läßt sich der HMA-Bereich optimal durch TSR-Programme nutzen, da kleine Blöcke nicht allokiert werden. Nach dem Aufruf findet sich in AX ein Statuscode:

```
AX: 0001H  Aufruf erfolgreich
    0000H  Aufruf fehlerhaft
```

Wird der Wert 0001H zurückgegeben, konnte der XMM den Speicher reservieren. Ein Wert 0 in AX signalisiert, daß der Aufruf nicht korrekt abgewickelt wurde. Im Register BL steht dann ein Fehlercode. Die möglichen Fehlercodes (81H, 80H, 91H, 92H, 93H) sind in Tabelle 12.2 aufgeführt.

Release High Memory Area (AH = 02H)

Mit dieser Funktion läßt sich ein Speicherbereich aus der HMA freigeben. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  XMS               º
º                               º
º AH: 02H (Release HMA)         º
û-------------------------------À
º      RETURN                   º
º AX: Status                    º
º BL: Errorcode                 º
Û-------------------------------ì
```

Der Aufruf gibt den kompletten HMA-Speicherbereich wieder frei. Nach dem Aufruf findet sich in AX ein Statuscode:

```
AX = 0001H Speicher freigeben
     0000H Fehler beim Aufruf
```

Falls der Wert 0 zurückgegeben wird, findet sich in BL ein Fehlercode mit der Ursache (81H, 80H, 93H). Die Tabelle 12.2 gibt die möglichen Codes mit ihren Ursachen an.

Global Enable A20 (AH = 03H)

Dieser Aufruf sorgt dafür, daß die High Memory Area für Zugriffe freigegeben wird.

```
Ö-------------------------------Ì
º      CALL:  XMS               º
º                               º
º AH: 03H (Enable  HMA)         º
û-------------------------------À
º      RETURN                   º
º AX: Status                    º
º BL: Errorcode                 º
Û-------------------------------ì
```

Die Funktion veranlaßt über den Tastaturprozessor, daß das Adreßbit A20 geschaltet wird. Erst dann sind Zugriffe auf den HMA-Bereich möglich. Nach dem Aufruf findet sich in AX der Status:

```
AX: 0001H  Aufruf erfolgreich
    0000H  Aufruf fehlerhaft
```

Beim Wert AX = 0 liegt ein Fehler vor, dessen Ursache in BL zurückgegeben wird. Tabelle 12.2 enthält die entsprechenden Codes (81H, 82H). Die Funktion darf nur durch Programme aufgerufen werden, die einen HMA-Bereich reserviert haben.

Global Disable A20 (AH = 04H)

Dieser Aufruf sorgt dafür, daß die High Memory Area für Zugriffe gesperrt wird.

```
Ö-------------------------------Ì
º      CALL:  XMS               º
º                               º
º AH: 04H (Disable HMA)         º
û-------------------------------À
º      RETURN                   º
º AX: Status                    º
º BL: Errorcode                 º
Û-------------------------------ì
```

Die Funktion veranlaßt über den Tastaturprozessor, daß das Adreßbit A20 gesperrt (auf 0 setzen) wird. Dann sind Zugriffe auf den HMA-Bereich nicht mehr möglich. Nach dem Aufruf findet sich in AX der Status:

```
AX: 0001H  Aufruf erfolgreich
    0000H  Aufruf fehlerhaft
```

Beim Wert AX = 0 liegt ein Fehler vor, dessen Ursache in BL zurückgegeben wird (81H, 82H, 94H). Tabelle 12.2 enthält die entsprechenden Codes. Die Funktion sollte immer dann aufgerufen werden, wenn ein HMA-Programm terminiert.

Local Enable A20 (AH = 05H)

Dieser Aufruf sorgt dafür, daß die Adreßleitung A20 freigegeben wird.

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 05H (Enable  A20)          º
û--------------------------------À
º      RETURN                    º
º AX: Status                     º
º BL: Errorcode                  º
Û--------------------------------ì
```

Die Funktion veranlaßt über den Tastaturprozessor, daß das Adreßbit A20 freigegeben wird. Anschließend sind direkte Zugriffe auf den Extended-Memory-Bereich möglich. Dies ist zum Beispiel erforderlich, falls ein Programm im Protected Mode arbeitet. Nach dem Aufruf findet sich in AX der Status:

```
AX: 0001H  Aufruf erfolgreich
    0000H  Aufruf fehlerhaft
```

Beim Wert AX = 0 liegt ein Fehler vor, dessen Ursache in BL zurückgegeben wird (81H, 82H). Tabelle 12.2 enthält die entsprechenden Codes. Die Funktion wird von Programmen benutzt, die einen direkten Zugriff auf den Extended-Memory-Bereich benötigen. Dies muß nicht unbedingt ein Zugriff auf HMA sein. Local Enable erhöht bei jedem Aufruf einen internen *Lock Counter*. Nur wenn der Zähler = 0 ist, wird A20 wirklich freigegeben.

Local Disable A20 (AH = 06H)

Dieser Aufruf sorgt dafür, daß die Adreßleitung A20 gesperrt wird.

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 06H (Disable A20)          º
û--------------------------------À
º      RETURN                    º
º AX: Status                     º
º BL: Errorcode                  º
Û--------------------------------ì
```

Die Funktion veranlaßt über den Tastaturprozessor, daß das Adreßbit A20 gesperrt wird. Die Sperrung erfolgt allerdings nur, falls der interne Lock-Zähler den Wert 1 besitzt. Bei jedem Aufruf wird der Zähler um 1 dekrementiert. Die Zahl der Local-Enable- und Local-Disable-Aufrufe muß also immer übereinstimmen. Anschließend sind direkte Zugriffe auf den Extended-Memory-Bereich nicht mehr möglich. Nach dem Aufruf findet sich in AX der Status:

```
AX: 0001H  Aufruf erfolgreich
    0000H  Aufruf fehlerhaft
```

Beim Wert AX = 0 liegt ein Fehler vor, dessen Ursache in BL zurückgegeben wird (81H, 82H, 94H). Tabelle 12.2 enthält die entsprechenden Codes.

Query A20 State (AH = 07H)

Dieser Aufruf prüft, ob die Adreßleitung A20 freigegeben oder gesperrt ist. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 07H (Query   A20)          º
û--------------------------------À
º      RETURN                    º
º AX: Status                     º
º BL: Errorcode                  º
Û--------------------------------ì
```

Das Ergebnis findet sich nach dem Aufruf in AX:

```
AX: 0001H  A20 enabled
    0000H  A20 disabled
```

Im Register BL wird der Fehlerstatus zurückgegeben (81H). Nur falls dort der Wert 0 steht, wurde der Aufruf erfolgreich durchgeführt und der Wert in AX ist gültig. Andernfalls ist der Fehler gemäß Tabelle 12.2 zu dekodieren.

Query free Extended Memory (AH = 08H)

Dieser Aufruf ermittelt die Länge des größten zusammenhängenden freien Extended-Memory-Blocks und die Größe des gesamten Extended-Memory-Bereichs und gibt diese zurück:

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 08H (Query Size)           º
û--------------------------------À
º      RETURN                    º
º AX: Blockgröße                 º
º DX: Größe Ext. Memory          º
º BL: Errorcode                  º
Û--------------------------------ì
```

Im Register BL wird nach dem Aufruf ein Fehlercode gemäß Tabelle 12.2 zurückgegeben (81H, A0H). Nur falls der Wert 0 ist, sind die Parameter in den restlichen Registern gültig. In AX steht dann die Größe des größten freien Extended-Memory-Blocks in Kbyte. Das Register DX gibt die Größe des gesamten Extended-Memory-Bereiches in Kbyte an. Der Bereich des HMA-RAM (64 Kbyte) wird allerdings immer von dem Wert in DX abgezogen.

Allocate Extended Memory Block (AH = 09H)

Mit diesem Aufruf läßt sich ein Speicherbereich aus dem Extended-Memory-Bereich einem Programm zuweisen. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 09H (Allocate Memory)      º
º DX: Blockgröße                 º
û--------------------------------À
º      RETURN                    º
º AX: 0001H ok                   º
º DX: Handle                     º
º AH: 0000H Fehler               º
º BL: Errorcode                  º
Û--------------------------------ì
```

Im Register DX ist die Größe des benötigten Speicherbereiches in Kbyte anzugeben. Der .i.XMS-Manager; prüft, ob ein genügend großer Bereich vorliegt und ordnet diesen gegebenenfalls dem Programm zu. Im Register AX findet sich nach dem Aufruf der Status. Der Wert 0001H signalisiert, daß der Speicherblock zugewiesen wurde. Um die verschiedenen Blöcke zu verwalten, gibt der XMS-Manager in DX deshalb einen Handlecode zurück.

Mit diesem Handlecode kann dann das Anwenderprogramm auf den reservierten Bereich zugreifen.

Beim Wert AX = 0 liegt ein Fehler vor, dessen Ursache in BL zurückgegeben wird (81H, A0H, A1H). Tabelle 12.2 enthält die entsprechenden Codes.

Free allocated Extended Memory Block (AH = 0AH)

Mit diesem Aufruf läßt sich ein reservierter Speicherblock aus dem Extended-Memory-Bereich wieder freigeben. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 0AH (Free Allocated Block) º
º DX: Handle des Blocks          º
û--------------------------------À
º      RETURN                    º
º AX: 0001H ok                   º
º AH: 0000H Fehler               º
º BL: Errorcode                  º
Û--------------------------------ì
```

Im Register DX ist die Handlenummer des freizugebenden Speicherbereiches anzugeben. Diese Handlenummer muß durch den Aufruf 09H ermittelt worden sein. Der XMS-Manager prüft, ob ein entsprechender Block vorhanden ist und gibt ihn gegebenenfalls frei. Der Wert AX = 0001H signalisiert, daß der Speicherblock freigegeben wurde. Der Block darf allerdings vor der Freigabe nicht gesperrt worden sein (siehe Funktion 0CH).

Beim Wert AX = 0 liegt ein Fehler vor, dessen Ursache in BL zurückgegeben wird (81H, A2H, ABH). Tabelle 12.2 enthält die entsprechenden Codes.

Move Extended Memory Block (AH = 0BH)

Mit diesem Aufruf läßt sich der Inhalt eines Blockes innerhalb des Adreßraumes verschieben. Es gilt folgende Schnittstelle:

```
Ö-------------------------------Ì
°      CALL:  XMS               °
°                               °
° AH: 0BH (Copy Block)          °
° DS:SI EMM-Struktur            °
û-------------------------------À
°      RETURN                   °
° AX: 0001H ok                  °
°     0000H Fehler              °
° BL: Errorcode                 °
Û-------------------------------ì
```

Die Funktion erlaubt Speicherverschiebungen innerhalb des DOS-Bereiches, zwischen DOS und dem Extended Memory und innerhalb des Extended-Memory-Bereiches. Im Register DS:SI ist die Adresse eines EMM-Kontrollblockes zu übergeben. Dieser Block besitzt folgenden Aufbau:

```
Ö-------Ú---------------------------------Ì
° Bytes ° Feld                           °
û-------é---------------------------------À
°   4   ° Zahl der zu verschiebenden Bytes°
û-------é---------------------------------À
°   2   ° Handle Source Block            °
û-------é---------------------------------À
°   4   ° Offset in den Source Block     °
û-------é---------------------------------À
°   2   ° Handle Destination Block       °
û-------é---------------------------------À
°   4   ° Offset in den Destination Block °
Û-------Ù---------------------------------ì
```

Im ersten DWORD steht die Größe des zu verschiebenden Blocks in Byte. Dieser Wert muß immer gerade sein, da nur Worte verschoben werden. Der folgende Wert spezifiziert das Handle des Quellblocks, von dem die Daten kopiert werden sollen. Die folgende DWORD-Adresse gibt den Offset des Blockes an. Dieser wird nicht benötigt, falls sich die Angabe auf einen Bereich im Extended Memory bezieht. Hier kennt der Benutzer die Adresse nicht, sondern greift über den Handle zu. Nun gibt es aber die Situation, daß ein Block aus dem DOS-Bereich in den Extended-Memory-Bereich ausgelagert wird. Dann ist der Wert für das Quellhandle auf den Wert 0000H zu setzen. Das folgende DWORD wird dann als physikalische Adresse mit Segment:Offset interpretiert. Der Anwenderprozeß kann dort die absolute Adresse ablegen und der Block wird kopiert. Das gleiche gilt für die Felder mit dem Destination Handle und dem Destination-Block-Offset. Bei Adreßüberlappungen funktioniert der Aufruf nur, falls die Quelladresse kleiner als die Zieladresse ist. Die Adreßleitung A20 wird automatisch durch die Funktion aktiviert. Die Funktion stellt im Gegensatz zum INT 15 weiterhin sicher, daß genügend Zeit für Unterbrechungen vorhanden ist, so daß keine Interrupts verlorengehen.

Prinzipiell läßt sich mit dieser Technik auch auf absolute Adressen im Extended-Memory-Bereich zugreifen. Dann funktioniert aber die Verwaltung durch den EMM-Handler nicht mehr.

Im Register AX steht nach dem Aufruf der Fehlerstatus. Bei AX = 0 wurde ein Fehler festgestellt, dessen Code in BL steht (81H, 82H, A3H, A4H, A5H, A6H, A7H, A8H, A9H).

Lock Extended Memory Block (AH = 0CH)

Mit diesem Aufruf läßt sich ein Speicherblock im Extended Memory gegen weiteres Verschieben sperren. Normalerweise verschiebt XMM die Blöcke so, daß immer genügend freier Speicherraum für neue Anforderungen zur Verfügung steht. Dadurch lassen sich aber

die physikalischen Adressen der Speicherblöcke nicht mehr feststellen. Die Sperre sorgt dafür, daß die Blöcke fest an einer Stelle bleiben und damit ist die physikalische Adresse wieder abfragbar. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 0CH (Lock Block)           º
º DX: Handle                     º
û--------------------------------À
º      RETURN                    º
º AX: 0001H ok                   º
º DX:BX Blockadresse             º
º AX: 0000H Fehler               º
º BL: Errorcode                  º
Û--------------------------------ì
```

Vor dem Aufruf ist im Register DX der Handle des zu sperrenden Blockes zu übergeben. Nach dem Aufruf enthält AX den Fehlerstatus. Beim Wert AX = 0001H wurde der Block gesperrt und in DX:BX steht die absolute Adresse des gesperrten Blocks in der Segment:Offset-Notation. Der Block läßt sich allerdings dann nur noch freigeben (Funktion 0AH), falls vorher die Funktion 0DH aufgerufen wurde.

Mit dem Wert AX = 0 wird ein Fehler signalisiert, wobei der Fehlercode in BL steht (81H, A2H, ACH, ADH).

Unlock Extended Memory Block (AH = 0DH)

Mit diesem Aufruf läßt sich ein gesperrter Speicherblock im Extended Memory für weitere Verschiebungen freigeben. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 0DH (Unlock Block)         º
º DX: Handle                     º
û--------------------------------À
º      RETURN                    º
º AX: 0001H ok                   º
º     0000H Fehler               º
º BL: Errorcode                  º
Û--------------------------------ì
```

Vor dem Aufruf ist im Register DX der Handle des freizugebenden Blockes zu übergeben. Nach dem Aufruf enthält AX den Fehlerstatus. Beim Wert AX = 0001H wurde der Block freigegeben. Anschließend sind alle absoluten Adreßzeiger auf diesen Block ungültig. Mit dem Wert AX = 0 wird ein Fehler signalisiert, wobei der Fehlercode in BL steht (81H, A2H, AAH).

Get Handle Information (AH = 0EH)

Mit diesem Aufruf lassen sich einige Informationen bezüglich des zugehörigen Blockes abfragen.

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 0EH (Get Information)      º
º DX: Handle                     º
û--------------------------------À
º      RETURN                    º
º AX: 0001H ok                   º
º DX:BX Block                    º
º AX: 0000H Fehler               º
º BL: Errorcode                  º
Û--------------------------------ì
```

Vor dem Aufruf ist im Register DX der Handle des abzufragenden Blockes zu übergeben. Nach dem Aufruf enthält AX den Fehlerstatus. Beim Wert AX = 0001H finden sich in den Registern BX und DX Informationen über den Status:

```
BH: Lock Zähler
BL: Zahl der freien Handles
DX: Blockgröße in Kbyte
```

Die Daten in BH und BL beziehen sich auf den ganzen Extended-Memory-Bereich, während in DX die Größe des per Handle spezifizierten Blockes steht. Mit AX = 0000H trat ein Fehler auf und in BL steht ein Fehlercode (81H, A2H) gemäß Tabelle 12.2.

Reallocate Extended Memory Block (AH = 0FH)

Mit diesem Aufruf läßt sich die Größe eines Blockes im Extended Memory verändern. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  XMS                º
º                                º
º AH: 0FH (Reallocate Block)     º
º DX: Handle                     º
º BX: neue Blockgröße            º
û--------------------------------À
º      RETURN                    º
º AX: 0001H ok                   º
º     0000H Fehler               º
º BL: Errorcode                  º
Û--------------------------------ì
```

Vor dem Aufruf ist im Register DX der Handle des zu verändernden Blockes zu übergeben. In BX steht die neue Größe des Blockes in Kbyte. Nach dem Aufruf enthält AX den Fehlerstatus. Beim Wert AX = 0001H wurde die Blockgröße verändert. Mit dem Wert AX = 0 wird ein Fehler signalisiert, wobei der Fehlercode in BL steht (81H, A0H, A1H, A2H, ABH).

Request Upper Memory Block (AH = 10H)

Die Funktion reserviert einen Speicherblock oberhalb des 640-Kbyte-RAM-Bereiches, der aber nicht zum EMS-Bereich gehören darf. Hiermit lassen sich freie Bereiche zwischen 640 Kbyte und 1 Mbyte (Upper Memory zwischen den Adaptern) für RAM-Karten nutzen und verwalten. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  XMS               º
º                               º
º AH: 10H (Request Memory)      º
º DX: Blockgröße                º
û-------------------------------À
º      RETURN                   º
º AX: 0001H ok                  º
º DX: Blockgröße                º
º BX: Segmentadresse            º
º AX: 0000H Fehler              º
º BL: Errorcode                 º
º DX: größter Block             º
Û-------------------------------ì
```

In DX ist die Blockgröße in Paragraphen anzugeben. Der Treiber prüft nun, ob ein Speicherbereich frei ist und weist gegebenenfalls den Speicherblock zu. Mit AX = 0001H wurde der Block reserviert und in BX findet sich die Segmentadresse. Die aktuelle Blockgröße steht in DX. Ansonsten steht in diesem Register die Größe des noch freien UMB. Mit AX = 0 liegt ein Fehler vor, dessen Code in BL zurückgegeben wird (B0H, B1H). Dann findet sich in DX die Größe des größten freien Blocks.

Release Upper Memory Block (AH = 11H)

Mit diesem Aufruf wird ein durch die Funktion 10H reservierter Block wieder freigegeben. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  XMS               º
º                               º
º AH: 11H (Release Block)       º
º DX: Segmentadresse Block      º
û-------------------------------À
º      RETURN                   º
º AX: 0001H ok                  º
º     0000H Fehler              º
º BL: Errorcode                 º
Û-------------------------------ì
```

In DX ist die Segmentadresse des Upper-Memory-Blocks (UMB) zu übergeben. Dieser Block wird wieder freigegeben, so daß das Register den Wert 0001H nach dem Aufruf enthält. Mit AX = 0 liegt ein Fehler vor, dessen Code in BL steht (B2H).

Die vom XMS-Treiber zurückgegebenen Fehlercodes sind gemäß folgender Tabelle zu dekodieren:

Code	Fehler
80H	Funktion nicht implementiert
81H	VDISK wurde erkannt
82H	Ein A20-Error ist aufgetreten
8EH	genereller Treiberfehler
8FH	nicht korrigierbarer Treiberfehler
90H	HMA existiert nicht
91H	HMA wird bereits benutzt
92H	DX kleiner als der /HMAMIN = Parameter
93H	HMA nicht belegt (allocated)
94H	A20 Adressleitung noch enabled
A0H	Extended Memory bereits belegt (allocated)
A1H	alle verfügbaren Ext.Mem. Handles belegt
A2H	ungültiger Handle
A3H	Quell Handle ungültig
A4H	Quell Offset ungültig
A5H	Ziel Handle ungültig
A6H	Ziel Offset ungültig
A7H	Länge ungültig
A8H	Überlappung bei den Move Adressen
A9H	Parity Error
AAH	Block nicht gesperrt (locked)
ABH	Block ist gesperrt (locked)
ACH	Block Lock Count Überlauf
ADH	Lock failed
B0H	nur ein kleinerer UMB verfügbar
B1H	keine UMB (upper Memory Blocks) verfügbar
B2H	UMB Segment Number ungültig

Tabelle 12.2: Fehlercodes des XMS-Handlers

12.4 Das Expanded Memory

Neben den 80x86-Prozessoren sind noch viele 8086-CPUs in Systemen im Einsatz. Auch hier machte sich bald die Begrenzung des Adreßraumes bemerkbar. Anders als bei den 80286-Rechnern ist eine Erweiterung des Speichers durch Extended Memory aber nicht möglich. Adressen oberhalb 1 Mbyte lassen sich ja nicht ansprechen.

Bereits 1984 wurde deshalb von den Firmen Lotus, Intel und Microsoft ein Standard zur Erweiterung des 1-Mbyte-Bereiches definiert. Dieser Standard erhielt die Bezeichnung EMS (Expanded Memory Spezification). In Anlehnung an die DOS-Versionen gab es auch für EMS die Versionen 3.0, 3.1 und 3.2. Mit der Version 3.0 ließen sich zum Beispiel zum erstenmal unbenutzte Teile von Lotus-1-2-3-Tabellen in den EMS-Bereich auslagern. EMS-Version 3.2 unterstützte dann bereits Microsoft Windows. 1985 wurde dann von den Firmen Ashton Tate, AST und Quadram ein erweiterter Standard EEMS (Enhanced Expanded Memory Standard) definiert, der maximal 64 Seiten zu 16 Kbyte in einem 64-Kbyte-Fenster in den 1-Mbyte-Adreßraum einblendet. Allerdings war diesem Standard kein sonderlicher Erfolg beschieden. So entschloß sich Microsoft bereits 1985 mit den Herstellern der EMS- und EMMS-Komponenten einen neuen Standard zu kreieren. Im August 1987 wurde der zur Version 3.2 kompatible EMS-4.0-Standard veröffentlicht. Nachfolgende Informationen beziehen sich (sofern nicht anders spezifiziert) auf die Versionen
LIM 3.2 und LIM 4.0.

Beim EMS-Treiber wird in der Funktionalität zwischen den Versionen 3.2 und 4.0 unterschieden. In beiden Versionen wird ein freies Fenster innerhalb des 1-Mbyte-Adreßraumes benutzt, um Speicherbänke einzublenden. Bei EMS 3.2 umfaßt dieses

Fenster eine Größe von 64 Kbyte, welches auf die Adressen C000H, C800H, CC00H oder D000H gelegt werden kann. Über das 64-Kbyte-Fenster lassen sich dann bis zu 8-Mbyte-RAM ansprechen. Allerdings dürfen nur Daten in diesem Speicherbereich abgelegt werden.

LIM 4.0 überwindet viele dieser Beschränkungen. Ist ein Teil der oben angegebenen 64-Kbyte-Speicherbereiche belegt, kann LIM 3.2 nicht mehr eingesetzt werden. Deshalb besteht in LIM 4.0 die Möglichkeit, das 64-Kbyte-Fenster (Small Page Frame) in vier Blöcke zu je 16 Kbyte aufzuteilen. Diese Blöcke müssen nicht mehr physikalisch zusammenhängend im Adreßraum liegen. Vielmehr können die Teilfenster auf jeder (freien) Adresse innerhalb des 1-Mbyte-Bereiches plaziert werden. Eine Anwendung kann dann allerdings auch nur 16-Kbyte-Blöcke anfordern. Bei entsprechender Hardwarekonfiguration empfiehlt sich deshalb, die vier Blöcke zu einem physikalisch zusammenhängenden 64-Kbyte-Fenster (Page Frame) zu kombinieren. Weiterhin dürfen ab LIM 4.0 bis zu 64 Seiten a 16 Kbyte im 1-Mbyte-Adressraum eingeblendet werden (Large Page Fragme). Damit läßt sich theoretisch der komplette DOS-Speicher mit EMS-RAM belegen. Der Treiber EMS40.SYS wird in der Datei CONFIG.SYS eingebunden und beim Systemstart von DOS geladen. Er übernimmt die Verwaltung des EMS-Speicherbereiches. Bild 12.4 gibt die prinzipielle Arbeitsweise des Expanded Memory wieder.

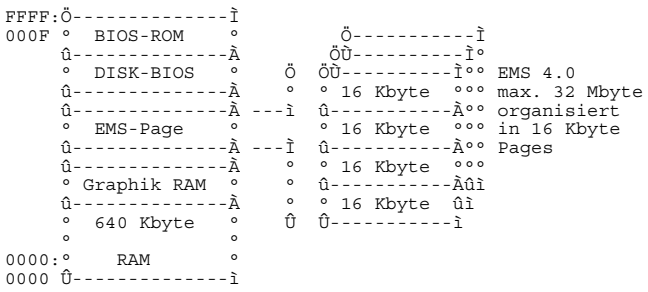


Bild 12.4: Organisation des Expanded Memory

Die Basisadresse der Fenster läßt sich in der Regel hardwaremäßig einstellen. Beim *Intel Above Board* sind die I/O- und Segmentadressen gemäß Tabelle 12.2 zulässig.

I/O - Adressen	Segmente
208H - 20FH	C400H, C800H
218H - 21FH	CC00H, D000H
258H - 25FH	D400H, DC00H
268H - 26FH	E000H

Tabelle 12.2: Adreßbereiche für EMS-Karten

In EMS 4.0 läßt sich der Expanded-Memory-Bereich bis auf 32 Mbyte ausbauen. Ein 64-Kbyte-Fenster wird durch den Treiber nochmals in 16-Kbyte-Teilblöcke geteilt. Im EMS-4.0-RAM dürfen sowohl Code als auch Daten abgelegt werden.

Durch Umschalten der Pages lassen sich beliebige physikalische 16-Kbyte-Blöcke aus dem 32-Mbyte-Bereich in das 64-Kbyte-Fenster logisch zusammenhängend einblenden. Bei

älteren EMS-Karten erfolgt die Verwaltung der Pageadressen und Look-up-Tabellen durch den Treiber im DOS-RAM. Moderne Boards besitzen eigene Pageregister zur Verwaltung des Speichers, was sich auch in der Zugriffsgeschwindigkeit bemerkbar macht. Die Verwaltung der Pages erfolgt, analog zum DOS-Filesystem, über logische »Handles« die der Treiber dem Anwenderprozeß zuweist. Hierbei handelt es sich aber um nichts anderes als um Offsetadressen auf eine Tabelle, in der Informationen über die Zuordnung der logischen Pages zu den physikalischen 16-Kbyte-Blöcken verwaltet wird (Bild 12»5).

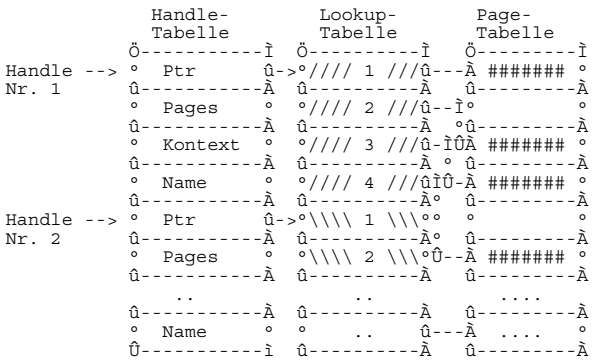


Bild 12.5: Interne Verwaltung der EMS-Tabellen

Benötigt eine Anwendung Speicher aus dem EMS-Bereich, dann muß sie diesen analog zu den INT 21-Funktionen des DOS-Memory-Managers über den INT 67H vom EMS-Memory-Manager anfordern. Dieser prüft zuerst, ob noch genügend freie 16-Kbyte-Blöcke vorliegen und weist gegebenenfalls mehrere physikalisch unabhängige 16-Kbyte-Seiten als logisch zusammenhängend dieser Anwendung zu. Die Zuordnung zwischen logischer und physikalischer Ebene erfolgt im EMS-Treiber über die in Bild 12.5 dargestellten Tabellen. Der Treiber legt für jeden Anwenderprozeß, der EMS-Speicher anfordert, einen Eintrag in der Handletabelle an. Der zurückgelieferte Handlecode ist dann der Offset vom Tabellenanfang auf das erste Element des zugehörigen Eintrages. Da der Treiber pro Prozeß einen Eintrag in der Tabelle angelegt und über die Daten den Speicher verwaltet, möchte ich die Einträge als Expanded-Memory-Control-Blöcke (EMCB) bezeichnen. Jeder EMCB besitzt folgenden Aufbau:

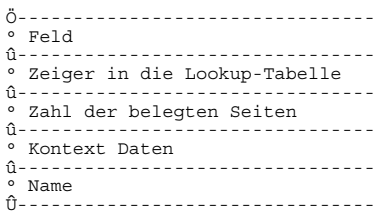


Tabelle 12.4: Aufbau des Expanded-Memory-Control-Blocks

Das erste Feld enthält einen Zeiger in die Lookup-Tabelle. In dieser Tabelle erfolgt die Zuordnung zwischen den logischen und physikalischen Seiten. Jeder logischen 16-Kbyte-

Page wird ein Eintrag für einen Zeiger auf die physikalische 16-Kbyte-Page reserviert. Der Treiber trägt ab der angegebenen Position die Blocknummer der physikalischen Pages in der Lookup-Tabelle ein. Die Zahl der belegten Blöcke wird im zweiten Feld der Handletabelle vermerkt. In der Lookup-Tabelle ist dann ab der Zeigerposition die entsprechende Anzahl von Einträgen vorhanden. Das folgende Feld enthält die Kontext-Daten zum betreffenden Handle. Die genaue Bedeutung ist allerdings nicht bekannt. Im Namensfeld legt der Treiber gegebenenfalls einen vom Benutzer definierten Namen des Handle ab.

Bei den bisher aufgeführten Informationen handelt es sich um weitgehend undokumentierte Fakten, da der Zugriff des Benutzers keinerlei Kenntnisse über die Funktionsweise des EMS-Treibers voraussetzt. Vielmehr wickelt der Treiber die komplette Verwaltung des Speichers ab.

12.4.1 Der Zugriff auf den EMS-Speicher

Alle Zugriffe eines System- oder Anwenderprozesses auf den EMS-Speicher erfolgen über den Treiber. Dieser wird beim Systemstart resident in DOS eingebunden. Der Treiber stellt den INT 67H zur Kommunikation mit anderen Prozessen zur Verfügung. Im Register AH sind (ähnlich wie beim INT 21) die entsprechenden Codes für die gewünschten Unterfunktionen zu übergeben. Die restlichen Register werden dann funktionsspezifisch belegt.

Für den Umgang mit EMS gelten einige Besonderheiten, die nachfolgend kurz zusammengefaßt werden.

- Vor einem ersten Zugriff auf den INT 67H muß sichergestellt sein, daß der Treiber installiert ist.
- Es muß sichergestellt werden, daß genügend freier Speicher für die Applikation zur Verfügung steht.
- Sobald die Page-Frame-Adresse bekannt ist, kann auf den Speicher wie auf konventionelle Bereiche zugegriffen werden.
- Belegte EMS-Seiten müssen nach Gebrauch an den Manager zurückgegeben werden, da sie sonst belegt bleiben.
- Stackbereiche sollten niemals in den EMS-Bereich gelegt werden. Bei Änderungen der eingeblendeten Seiten ist sonst ein Systemabsturz vorprogrammiert.
- Interrupt-Service-Routinen sollten ebenfalls nicht mit ihrem Einsprungpunkt im EMS-Bereich liegen. Der Vektor INT 67H ist nur für den EMS-Treiber vorgesehen, eine Verwendung für andere Zwecke ist nicht zulässig.

Der Umgang mit den einzelnen Funktionsaufrufen wird in den folgenden Abschnitten beschrieben.

12.4.2 Die Prüfung des Installationsstatus

Vor einem Zugriff auf den EMS-Treiber muß ein Prozeß sicherstellen, daß dieser Treiber wirklich geladen ist. Beim Multiplexerinterrupt INT 2FH sorgt DOS ja dafür, daß der

Vektor definiert ist. Leider unterstützt DOS zumindest bis zur Version 3.3 den INT 67H nicht. Ist der Treiber nicht installiert, führt ein Aufruf im ungünstigsten Fall zu einem Systemabsturz, falls der Vektor zum Beispiel den Wert 0000:0000H enthält. Ein Prozeß muß deshalb vor dem ersten INT 67**H**-Aufruf den Installationsstatus prüfen. Hierfür lassen sich folgende zwei Verfahren einsetzen:

Die erste Möglichkeit benutzt einen Test des INT 67-Vektors mit Codeinspektion. Dieser Test setzt voraus, daß der INT 67-Vektor vom rufenden Prozeß gemäß folgenden Schritten untersucht wird:

- Lese den INT 67**H**-Vektor über die INT 21-Funktion 35H (Get Interrupt Vektor) ein und prüfe den Wert. Enthält der Vektor den Wert 0000:0000H, ist der Treiber nicht installiert.
- Andernfalls ist der durch den Vektor adressierte Speicher auf die Werte CFH zu prüfen. Dieser Code entspricht der IRET-Anweisung. Findet sich das Codebyte an dieser Stelle, dann ist der Treiber nicht installiert.
- In allen anderen Fällen ist zumindest ein gültiger Prozeß unter dem Interruptvektor installiert. Treiber müssen in DOS aber bestimmten Vorschriften genügen. So steht ab Offset 0AH der Name des Treibers als ASCII-String im Kopf des Treibers. Für alle EMS-Versionen muß an dieser Stelle der Text: EMMXXXX0 stehen.

Nur wenn obiger Text gefunden wird, ist ein EMS-Treiber installiert. Welche Version vorliegt, ist durch weitere Aufrufe des INT 67 zu prüfen. Die Vorgehensweise ist immer dann erforderlich, wenn ein Zugriff auf die DOS-INT 21-Funktionen nicht möglich ist (z.B in Treibern).

Die zweite Möglichkeit besteht darin, den Treiberstatus über den INT 21 (Open Handle) zu prüfen. Dies ist die sauberste Möglichkeit für Applikationsprogramme. Hierfür sind folgende Schritte auszuführen:

- Über den INT 21 wird die Funktion 3DH (Open Handle) aufgerufen. Das Zugriffsrecht beschränkt sich dabei auf *read only*. Im Register DS:DX ist die Adresse eines ASCIIZ-Strings mit dem Laufwerk und dem Pfad oder der Einheit zu übergeben. Hier ist der Name des Treibers (EMMXXXX0) einzutragen.
- Meldet DOS einen *too many open files*-Fehler, muß das Programm alle anderen Dateien schließen und die Abfrage wiederholen.
- Gibt DOS die Meldung *File/Path not found* zurück, dann ist Manager nicht installiert.
- Bei fehlerfreiem Aufruf ist entweder der Treiber oder eine Datei mit dem Namen vorhanden. Über den INT 21 läßt sich die Funktion 4400H ansprechen, die Informationen über den Treiber liest.
- Fehler bei diesem Aufruf deuten darauf hin, daß kein Treiber vorhanden ist. Andernfalls ist Bit 7 zu prüfen. Falls es den Wert 0 besitzt, liegt eine Datei EMMXXXX0 vor. Andernfalls ist ein Treiber mit dem Namen installiert.
- Dann ist ein INT 21-Funktionsaufruf 4407H (get output status) auszuführen. Falls der Treiber korrekt installiert ist, gibt er den Wert 0FFH im Register AL zurück. Der Status AL = 0 bedeutet, daß der Treiber nicht bereit ist und installiert werden muß.

· Nach der Abfrage ist der offene Handle mit einem (Close File Handle) Kommando (INT 21, Funktion 3EH) zu schließen.

Egal wie der Installationsstatus festgestellt wird, fehlt der EMS-Treiber, muß das System neu gebootet werden. Der Treiber ist dann über die Datei CONFIG.SYS zu laden.

12.4.3 Die Funktionen des EMS-Treibers (INT 67)

Alle EMS-Zugriffe aus Anwenderprozessen erfolgen per INT 67H, wobei im Register AH der entsprechende Funktionscode zu übergeben ist. Die Registerbelegung wird dabei funktionsspezifisch vorgenommen. Nachfolgend werden die Funktionen der EMS-Versionen 3.2 und 4.0 besprochen.

Get Manager Status (AH = 40H, Version 3.2, 4.0)

Mit der Funktion läßt sich der Status des installierten Treibers abfragen. Der Aufruf benutzt folgende Register:

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 40H (Get Install Status)   º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
Û--------------------------------ì
```

Im Register AH gibt die Funktion den Status des Treibers zurück. Der Code bezieht sich dabei auf den Zustand der letzten ausgeführten Funktion. Die Funktion 40H darf beliebig oft zur Statuscodeabfrage aufgerufen werden, ohne daß sich der Statuscode ändert. Die Funktion gibt die Statuscodes: 00H, 80H, 81H und 84H zurück. Der Aufruf ist in EMS 3.2 und EMS 4.0 sowie in EEMS 3.2 implementiert.

Ein Aufruf anderer Funktionen beeinflußt allerdings den Statuscode. Für die zurückgegebenen Werte gilt die Kodierung gemäß Tabelle 12.5.

Code	Fehlerstatus
00H	Funktionsaufruf erfolgreich beendet
80H	interner Fehler im EMM-Treiber
81H	Hardware-Fehler
82H	Treiber »busy° 83H ° ungültiger Handle
84H	ungültiger Funktionscode
85H	keine Handles mehr frei
86H	Fehler im Save/Restore-Mapping-Context
87H	mehr Pages angefordert als existieren
88H	mehr Seiten angefordert als frei
89H	Handle für Seite 0 ungültig (frei EMS 4.0)
8AH	EMS-Seite liegt außerhalb des Handles
8BH	EMS-Seite liegt außerhalb des Fensters
8CH	Überlauf interne Konfigurationstabelle
8DH	Handle bereits belegt
8EH	Handle existiert nicht
8FH	Fehlerursache nicht identifizierbar
---	erweiterte Fehlermeldungen ab EMS 4.0
90H	Attribut Typ undefiniert
91H	Warm-Boot-Data-Save nicht implementiert
92H	Move: Adreßüberlappung
93H	Move/Exchange größer als belegter Bereich
94H	konvent./Expanded Bereich Überlappend
95H	log. Page Offset außerhalb der log. Page
96H	Bereich größer als 1 Mbyte
97H	Exchange: Source/Destination Überlappung
98H	Source/Destination undefiniert
99H	kein Status zugewiesen
9AH	Alternate-Map-Register-Set werden unterstützt, nicht aber der aktuelle Set
9BH	alle Alternate-Map&DMA-Register-Sets zugewiesen
9CH	Alternate-Map&DMA-Register-Sets werden nicht unterstützt
9DH	Alternate-Map-Register oder DMA-Set nicht definiert, oder belegt oder aktiv
9EH	dedizierte DMA-Kanäle nicht unterstützt
9FH	spezifizierter DMA-Kanal nicht unterstützt
0A0H	Handlenamen nicht gefunden
0A1H	Handlename existiert bereits
0A2H	Move/Exchange überschreitet 1 Mbyte
0A3H	Datenstruktur enthält defekte Daten
0A4H	Zugriff abgewiesen

Tabelle 12.5: EMS-Fehlercodes

Die Funktionscodes werden bei jedem Funktionsaufruf im Register AH zurückgegeben. Deshalb ist die Funktion 40H in der Regel überflüssig. Die Funktion ist abwärtskompatibel mit früheren EMS-Versionen und mit der EEMS-Version 3.2.

Get Page Frame Segment Adresse (AH = 41H, Version 3.2, 4.0)

Mit dieser Funktion läßt sich die Segmentadresse des 64-Kbyte-EMS-Fensters ermitteln. Es gelten folgende Aufrufparameter:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 41H (Get Page Frame Adr)  º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º BX: Segmentadresse            º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 41H zu übergeben. Nach dem Aufruf enthält das Register AH den Statuscode mit der Kodierung gemäß Tabelle 12.5. Es werden die Codes 0, 80H, 81H und 84H zurückgegeben. Ist AH = 0, dann enthält das Register BX die Segmentadresse des 64-Kbyte-Fensters mit dem eingeblendeten EMS-Speicherbereich.

Mögliche Werte sind C000H, C400H etc. Der Wert ändert sich während des Betriebs nicht, da er durch den Treiber bei der Installation festgelegt wird. Die Funktion ist auch in EEMS 3.2 vorhanden.

Get Unallocated Page Count (AH = 43H, Version 3.2, 4.0)

Diese Funktion liefert die Zahl der freien EMS-Seiten und die Gesamtzahl der verfügbaren EMS-Seiten.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 42H (Get unalloc. Pages)  º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º BX: Free Pages                º
º DX: Total Pages               º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 42H zu übergeben. Nach dem Aufruf enthält das Register AH den Statuscode mit der Kodierung gemäß Tabelle 12.5. Es werden die Codes 0, 80H, 81H und 84H zurückgegeben. Ist AH = 0, dann enthält das Register BX die Zahl der noch unbelegten 16-Kbyte-Pages. In DX gibt der Treiber gleichzeitig die Zahl der verfügbaren Pages der EMS-Karte zurück.

Eine Page entspricht immer einem physikalischen 16-Kbyte-Block. Die Funktion ist nicht 100 % kompatibel mit EEMS 3.2, da dieser Treiber keine Möglichkeit zur Abfrage der maximal vom Programm belegbaren Handles bietet. Ersatzweise läßt sich die EMS-4.0-Funktion 54H mit dem Code AL = 02H nutzen.

Get Handle and Allocate Memory (AH = 43H, Version 3.2, 4.0)

Diese Funktion reserviert die Zahl der angeforderten Speicherseiten und gibt einen Handle an den Prozeß zurück.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 43H (Allocate Memory)     º
º BX: Pages                     º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º DX: Handle                    º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 43H zu übergeben, während in BX die Zahl der zu reservierenden 16-Kbyte-Pages steht. Nach dem Aufruf enthält das Register AH den Statuscode mit der Kodierung gemäß Tabelle 12.5. Es werden die Codes 0, 80H, 81H, 84H, 85H, 87H, 88H und 89H zurückgegeben. Ist AH = 0, dann enthält das Register DX den Handlecode, mit dem der Prozeß auf die reservierten Seiten zugreifen kann.

Der Aufruf wird ebenfalls von EEMS 3.2 unterstützt. Aber erst ab EMS 4.0 darf die Zahl der angeforderten Seiten (BX = 0) auf 0 gesetzt werden. In früheren Versionen wird sonst der Fehlerstatus 89H zurückgegeben. Innerhalb des Treibers lassen sich insgesamt 255 Handles verwalten, d.h. das obere Byte in DX ist immer null.

Map/Unmap Memory (AH = 44H, Version 3.2, 4.0)

Dieser Funktionsaufruf blendet die angegebenen EMS-Seiten in das 64-Kbyte-Fenster ein. Es gelten folgende Aufrufkonventionen:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 44H (Map/Unmap Pages)     º
º AL: physikalische Seite       º
º BX: logische Seite            º
º DX: Handle                    º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 44H zu übergeben, während in DX der Handlecode steht. Dieser Code muß vorher durch die Funktion 43H zugewiesen worden sein. Im Register AL spezifiziert der rufende Prozeß, auf welche physikalische Seite innerhalb des 64-Kbyte-Fensters die logische 16-Kbyte-Seite einzublenden ist. Erlaubte Werte für die physikalische Page liegen zwischen 0 und 3. Die Zahl der logischen Blöcke wird bei der Reservierung des Speichers (Funktion 43H) festgelegt.

Im Register BX ist bei der Funktion 44H die Nummer des einzublendenden logischen Blocks anzugeben. Der Wert reicht von 0 bis zur maximalen Seitenzahl 1, die dem Handle zugewiesen wurde. Wird als logische Blocknummer in BX der Wert FFFFH übergeben, sperrt der EMS-Treiber die im Register AL angegebene physikalische Seite (0-3) für alle Schreib-Lese-Zugriffe. Diese Funktion wird aber nur ab EMS-4.0 unterstützt. Mit der EMS 4.0-Funktion 58H läßt sich die gesperrte Seite wieder freigeben. Nach dem Aufruf enthält das Register AH den Statuscode der Funktion. Es werden die Codes 0, 80H, 81H, 83H, 84H, 8AH und 8BH zurückgegeben. Ist AH = 0, dann wurde die Funktion korrekt abgewickelt. EEMS 3.2 unterstützt ebenfalls diese Funktion.

Release Handle and Memory (AH = 45H, Version 3.2, 4.0)

Die mit dem Aufruf 43H reservierten Speicherseiten werden mit der Funktion 45H wieder freigegeben. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 45H (Release Memory)      º
º DX: Handle                    º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 45H zu übergeben, während in DX der Handlecode steht. Dieser Code muß vorher durch die Funktion 43H zugewiesen worden sein. Der Treiber gibt anschließend die reservierten Seiten wieder frei. Ein eventuell dem Handle zugewiesener logischer Name wird dabei mit 00H Zeichen überschrieben.

Im Register AH wird nach dem Aufruf der Status zurückgegeben (Codes 00H, 80H, 81H, 83H, 84H und 86H). Ein Programm muß diese Funktion aufrufen, bevor es terminiert, da sonst die Speicherseiten bis zum Systemneustart belegt bleiben! Die Seiten lassen sich anschließend nicht mehr durch andere Prozesse nutzen. Die Funktion wird auch in EEMS 3.2 unterstützt.

Get EMM Version (AH = 46H, Version 3.2, 4.0)

Über diesen Funktionsaufruf gibt der Treiber die EMS-Version zurück.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 46H (Get Version)         º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º AL: Versionsnummer            º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 46H zu übergeben, während nach dem Aufruf dort der Statuscode steht. Es werden die Codes 00H, 80H, 81H und 84H zurückgegeben. Beim Statuscode AH = 0 findet sich im Register AL die Versionsnummer des EMM-Treibers. Diese wird als BCD-Zahl kodiert, so daß der Wert 40 der Version 4.0 entspricht. EEMS 3.2 unterstützt ebenfalls diesen Aufruf.

Save Mapping Context (AH = 47H, Version 3.2, 4.0)

Dieser Funktionsaufruf sichert die aktuelle Seitenkonfiguration (Page Mapping Register) innerhalb des Fensters.

```
Ö-------------------------------Ì
°      CALL:  INT 67            °
°                               °
° AH: 47H (Save Page Map)       °
° DX: Handle                    °
û-------------------------------À
°      RETURN                   °
° AH: Status                    °
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 47H zu übergeben, während in DX der Handlecode steht, unter dem die Konfigurierung gespeichert wird. Dabei sichert der Treiber intern die Nummern der aktuellen 16-Kbyte-Seiten, die gerade im 64-Kbyte-Fenster eingeblendet sind in einer internen Tabelle. Der Offset zu dieser Tabellenposition wird als Handle vom Treiber zurückgegeben. Die Funktion arbeitet komplementär zur Funktion *Restore Page Map*.

Damit läßt sich die gerade aktuelle Einstellung mit einem Befehl sichern und das Fenster kann neu konfiguriert werden. Ein Aufruf von Restore stellt später die alten Zustände wieder her. Die Technik ist insbesondere für die Erstellung residenter Programme interessant. Ein solches Programm kann durchaus den EMS-Speicher benutzen. Vor einem Zugriff auf die eigenen Seiten wird einfach die aktuelle Einstellung in einem Handle gesichert. Vor Beendigung des Programmes restauriert der Aufruf 48H dann den alten Zustand. Es läßt sich einem Handle allerdings nur ein Fenster zuweisen. Mehrere Aufrufe mit dem gleichen Handle führen zu einer Fehlermeldung des Treibers. Die Funktion sichert allerdings nur die Konfigurierung. Das Programm ist anschließend für die Einblendung der benötigten Seiten verantwortlich. Dies erfolgt mit der Funktion 44H. Die Funktion 47H gibt in AH die Statuscodes 00H, 80H, 81H, 83H, 84H, 8CH und 8DH zurück und wird auch in EEMS 3.2 unterstützt. Da nur die Kontext-Daten des Fensters gesichert werden, empfiehlt es sich, bei residenten Programmen die Funktion 4EH und 4FH zu nutzen.

Restore Page Map (AH = 48, Version 3.2, 4.0)

Dieser Funktionsaufruf restauriert eine mit der Funktion 47H gesicherte Seitenkonfiguration innerhalb des Fensters.

```
Ö-------------------------------Ì
°      CALL:  INT 67            °
°                               °
° AH: 48H (Restore Page Map)    °
° DX: Handle                    °
û-------------------------------À
°      RETURN                   °
° AH: Status                    °
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 48H zu übergeben, während in DX der Handlecode steht, unter dem die Konfigurierung gespeichert ist. Die Funktion setzt voraus, daß vorher einmal die Funktion 47H benutzt wurde. Der Aufruf sorgt dafür, daß die unter dem Handle gespeicherte Konfiguration wieder im 64-Kbyte-Fenster eingeblendet wird.

Dabei werden die Seiten ebenfalls korrekt eingeblendet. Es ist immer nur ein Restore per Handle möglich. Mehrere Aufrufe der Funktion 48H mit dem gleichen Handle und ohne zwischenliegendes Save führt zu einer Fehlermeldung. Im Register AH gibt der Treiber den Status der Operation (00H, 80H, 81H, 83H, 84H, 8EH) zurück. Die Funktion ist kompatibel zu EEMS 3.2. Die Funktion 48H sichert nur die Map-Register des 64-Kbyte-

Fensters. Sollen die Map-Register außerhalb des Fensters gesichert werden, lassen sich die Funktionen 4EH und 4FH benutzen. Dies gilt auch für .i.TSR-Programm;e und Treiber.

Funktion 49H (Version 3.0, reserviert)

Die Funktion 49H wurde in EMS 3.0 benutzt, um das I/O-Port Registerfeld zurückzugeben. In EMS 3.2 und EMS 4.0 ist die Funktion nicht benutzt und reserviert. Sie sollte deshalb nicht benutzt werden, da Seiteneffekte mit anderen Funktionen auftreten können.

Funktion 4AH (Version 3.0, reserviert)

Auch diese Funktion wurde in EMS 3.0 benutzt, ist aber ab EMS 3.2 nicht mehr belegt. Sie gibt in EMS 3.0 das Page Translation Array zurück. Eine Verwendung in neueren EMS-Versionen ist nicht möglich.

Get Number of EMM Handles (AH = 4BH, Version 3.2, 4.0)

Mit der Funktion 4BH läßt sich die Zahl der benutzten Handles abfragen.

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 4BH (Get Handle Count)     º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º BX: Handlecount                º
Û--------------------------------ì
```

Nach dem Aufruf enthält das Register AH den Status (00H, 80H, 81H, 84H). Beim Wert 0 findet sich im Register BX die Zahl der vom Treiber aktuell verwalteten Handles. Der Handle 00H wird dabei vom Treiber belegt. Die Zahl der benutzten Handles liegt deshalb zwischen 0 und 255. Die Funktion wird von EEMS 3.2 unterstützt. Der Aufruf ist interessant, falls ein Programm prüfen muß, ob noch freie Handles vorhanden sind.

Get Pages Owned by Handle (AH = 4CH, Version 3.2, 4.0)

Mit der Funktion 4CH läßt sich die Zahl der Seiten ermitteln, die dem entsprechenden Handle zugeordnet sind. Es gilt folgende Aufrufschnittstelle.

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 4CH (Save owned Page)      º
º DX: Handle                     º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º BX: Seitenzahl                 º
Û--------------------------------ì
```

Im Register AH ist der Funktionscode 4CH zu übergeben, während in DX der Handlecode steht, für die die Zahl der belegten Seiten ermittelt werden soll. Nach dem Aufruf enthält das Register AX den Fehlerstatus (00H, 80H, 81H, 83H, 84H). Beim Wert 0 findet sich im Register BX die Zahl der 16-Kbyte-Seiten aus dem EMS-Speicher, die dem betreffenden Handle zugewiesen wurden.

Die Zahl der logischen Seiten, die ein Handle belegen kann, bewegt sich bei EMS 4.0 zwischen 1 und 2048. Die Funktion wird auch in EEMS 3.2 unterstützt.

Get Pages for All Handles (AH = 4DH, Version 4.0)

Die Funktion ermittelt die Gesamtzahl der Seiten des EMS-Speichers, die bereits benutzten Handles zugewiesen wurden.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 4DH (Get Page Count)      º
º ES:DI Feldadresse             º
û-------------------------------Ä
º      RETURN                   º
º AH: Status                    º
º BX: Handlecount               º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 4DH zu übergeben. Nach dem Aufruf enthält das Register AX den Fehlerstatus (00H, 80H, 81H, 84H). Beim Wert 0 findet sich im Register BX die Zahl der vom Treiber aktuell verwalteten Handles. Vor dem Aufruf ist ein 1 Kbyte großer Datenbereich für die Ergebnisse einzurichten.

Das Programm muß dessen Anfangsadresse im Registerpaar ES:DI übergeben. Pro Handle sind in diesem Feld dann zwei Worte belegt.

```
Ö------------------Ì
º    Handlecode    º
û------------------Ä
º    Seitenzahl    º
û------------------Ä
º       ...        º
û------------------Ä
º    Handlecode    º
û------------------Ä
º    Seitenzahl    º
Û------------------ì
```

Im ersten Wort steht die Nummer des Handles, dem die Seiten zugeordnet sind. Diese Nummer stimmt mit den an den Anwenderprozeß zurückgegebenen Handles überein. Im folgenden Wort findet sich die Zahl der belegten 16-Kbyte-Seiten, die diesem Handle zugeordnet sind. Bei EMS sind insgesamt 255 Handles möglich, so daß die Tabelle 1 Kbyte lang sein muß. Die Funktion wird nur in EMS 4.0 unterstützt und ist mit früheren EMS- und EEMS-Versionen nicht kompatibel. Das Handle 0 wird durch das Betriebssystem (Treiber) belegt, so daß der Anwenderprozeß nur die Handles 1 bis 255 erhält. Der Wert in BX muß deshalb auch immer größer 0 sein.

Get or Set Page Map (AH = 4EH, Version 4.0)

Die Funktion 4EH besitzt verschiedene Unterfunktionen, die die interne Speicherorganisation beeinflussen.

Get Page Map (AL = 00H)

Mit der Unterfunktion 00H läßt sich die Aufteilung des Speichers abfragen:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 4EH                       º
º AL: 00H (Get Page Map)        º
º ES:DI Feldadresse Map         º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 4EH zu übergeben. Das Register AL selektiert mit dem Wert 00H die Unterfunktion Get Page Map. Nach dem Aufruf enthält das Register AX den Fehlerstatus (00H, 80H, 81H, 84H, 8FH). Vor dem Aufruf ist ein x Kbyte großer Datenbereich für die Ergebnisse einzurichten. Die Größe dieser Tabelle kann von System zu System variieren.

Sie läßt sich aber mit der Funktion 4E03H ermitteln. Die Anfangsadresse ist in ES:DI zu übergeben. Pro Handle sind in diesem Feld dann zwei Worte belegt. Das erste Wort enthält die Handlenummer, dem die Seiten zugeordnet sind. Diese Nummer stimmt mit der dem Anwenderprozeß übergebenen Handlenummer überein. Im folgenden Wort findet sich die Zahl der 16-Kbyte-Seiten, die diesem Handle zugeordnet sind. Die Funktion ist zu EMS 3.2 und EEMS 3.2 kompatibel und beinhaltet auch die Funktionalität des EEMS-Aufrufes 6AH.

Set Page Map (AL = 01H)

Mit der Unterfunktion 01H läßt sich die Aufteilung des Speichers pro Handle neu setzen.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 4EH                       º
º AL: 01H (Set Page Map)        º
º DS:SI Feldadresse Map         º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 4EH zu übergeben. Das Register AL selektiert mit dem Wert 01H die Unterfunktion Set Page Map. Nach dem Aufruf enthält das Register AX den Fehlerstatus (00H, 80H, 81H, 84H, 8FH, A3H). Vor dem Aufruf ist ein x Kbyte großer Datenbereich für die Ergebnisse einzurichten.

Die Anfangsadresse muß in DS:SI übergeben werden. Die Tabelle sollte vorher mit der Funktion 4E00H abgefragt werden. Ein Anwenderprozeß kann dann die Zahl der Seiten pro Handle in dieser Tabelle modifizieren. Anschließend übernimmt der Treiber die Einstellung aus der Tabelle.

Change Page Map (AL = 02H)

Mit der Unterfunktion 02H läßt sich die Aufteilung des Speichers ändern:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 4EH                       º
º AL: 02H (Change Page Map)     º
º DS:SI Feldadresse Quelle      º
º ES:DI Feldadresse Ziel        º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 4EH zu übergeben. Das Register AL selektiert mit dem Wert 02H die Unterfunktion Change Page Map. Nach dem Aufruf enthält das Register AX den Fehlerstatus (00H, 80H, 81H, 84H, 8FH, A3H). Der Treiber erwartet beim Aufruf im Registerpaar DS:SI die Anfangsadresse einer x Kbyte großen Tabelle mit der bei der Unterfunktion 00H beschriebenen Aufteilung.

Die Länge der Tabelle läßt sich über die Funktion 4E03H abfragen. In der Tabelle sind die neuen Einstellungen vom rufenden Prozeß abzulegen. Beim Aufruf übernimmt der Treiber die Werte aus der Tabelle und überträgt sie in die Page Map des EMS-Boards. Gleichzeitig speichert er die aktuellen Werte des Boards in die durch das Registerpaar ES:DI spezifizierte Tabelle zurück. Nach einem erfolgreichen Aufruf enthält die Tabelle dann den alten Zustand.

Get Page Map Table Size (AH = 03H)

Mit der Unterfunktion 03H läßt sich die Größe der Page-Map-Tabelle abfragen.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 4EH                       º
º AL: 03H (Get Table Size)      º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º AL: Size                      º
Û-------------------------------ì
```

Nach dem Aufruf enthält das Register AH den Fehlerstatus (00H, 80H, 81H, 84H, 8FH). Beim Wert AH = 00H findet sich in AL die Größe der Tabelle in Kbyte.

Die folgenden Funktionen werden nur durch EMS 4.0 unterstützt.

Get/Set Partial Page Map (Funktion 4FH, Version 4.0)

Mit diesen vier Unterfunktionen läßt sich die Page Map partiell bearbeiten. Dies ist zum Beispiel bei TSR-Programmen wichtig, falls der Context gewechselt werden muß. Die Aufrufe erweitern die Funktionen der EEMS-Aufrufe.

Get Partial Page Map (AL = 00H)

Die Unterfunktion liest einen Teil der Page Map zurück. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 4FH                       º
º AL: 00H (Get Page Map)        º
º DS:SI Tabellenadresse 1       º
º ES:DI Tabellenadresse 2       º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 4FH zu übergeben. Das Register AL selektiert mit dem Wert 00H die Unterfunktion. In DS:SI ist die Anfangsadresse der Tabelle mit der Liste aller zu sichernden Segmente zu übergeben. Die Tabelle besitzt folgenden Aufbau:

```
Ö-------Ú-------------------------------------Ì
º Bytes º Feld                                º
û-------é-------------------------------------À
º   2   º Anzahl der Einträge im nächsten Feld º
û-------é-------------------------------------À
º   n   º Feld mit den Adressen der sichernden º
º       º physikalischen Seite.               º
Û-------Ù-------------------------------------ì
```

Das Registerpaar ES:DI enthält die Adresse der Tabelle, in der die Page Map zu speichern ist. Die Größe der Tabelle ist mit der Funktion 4F02H abfragbar. Nach dem Aufruf enthält das Register AX den Fehlerstatus (00H, 80H, 81H, 84H, 8BH, 8FH, A3H).

Set Partial Page Map (AL = 01H)

Mit dieser Unterfunktion läßt sich der Teil der Page Map wieder setzen, der durch 4F00H gelesen wurde.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 4FH                       º
º AL: 01H (Set Page Map)        º
º DS:SI Feldadresse Quelle      º
º ES:DI Feldadresse Ziel        º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 4FH zu übergeben. Das Register AL selektiert mit dem Wert 01H die Unterfunktion. Nach dem Aufruf enthält das Register AX den Fehlerstatus (00H, 80H, 81H, 84H, 8FH, A3H). Der Treiber erwartet beim Aufruf im Registerpaar DS:SI die Anfangsadresse der Tabelle mit der gesicherten Teil-Page-Map. Die Tabelle ist vorher mit 4F00H zu erstellen.

Get Size of Partial Page Map Save Array (AL = 02H)

Mit dieser Funktion läßt sich die Größe der zu sichernden Page-Map-Register-Tabelle ermitteln.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 4FH                       º
º AL: 02H (Get Size of Map)     º
º BX: Zahl der Segmente         º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º AL: Zahl der Segmente         º
Û-------------------------------ì
```

Im Register AH ist der Funktionscode 4FH zu übergeben. Das Register AL selektiert mit dem Wert 02H die Unterfunktion. In BX steht vor dem Aufruf die Zahl der zu verändernden Segmente, die in der .i.Partial-Page-Map; zu verarbeiten sind. Nach dem Aufruf enthält das Register AH den Fehlerstatus (00H, 80H, 81H, 84H, 8BH, 8FH, A3H). In AL wird die Zahl der Einträge in der Partial-Map-Table zurückgegeben.

Map/Unmap Multiple Pages (Funktion 50H, Version 4.0)

Mit diesen zwei Unterfunktionen lassen sich mehrere logische Seiten in die physikalischen 16-Kbyte-Fenster ein- oder ausblenden. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 50H                       º
º AL: 00H (physikalische Page)  º
º AL: 01H (Segment Nummer)      º
º CX: Zahl der Einträge         º
º DX: Handle                    º
º DS:SI Adresse Tabelle         º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Der Wert in AL bestimmt, ob die Selektion über die physikalische Seitennummer AL = 00H oder über die Segmentnummer AL = 01H erfolgt. In DS:DI ist die Anfangsadresse der Tabelle mit den Definitionen zu übergeben. Die Tabelle besitzt für jeden Eintrag folgende Struktur:

```
Ö-------Ú---------------------------------------Ì
º Bytes º Feld                                  º
û-------é---------------------------------------À
º   2   º log. Seitennummer der einzublendendenº
º       º Seite                                 º
û-------é---------------------------------------À
º   2   º physikalische Seitennummer           º
Û-------Ù---------------------------------------ì
```

CX enthält die Zahl der Einträge in dieser Tabelle. Der Wert 4 bedeutet, daß die Tabelle 4 physikalische Seiten oder vier Segmente aufnehmen kann. In DX steht der zugehörige Handlecode. In AH wird nach dem Aufruf der Fehlerstatus (00H, 80H, 81H, 83H, 84H, 8AH, 8BH, 8FH) zurückgegeben.

Diese neue Funktion erlaubt mehrfache Zuweisungen zwischen logischen und physikalischen Seiten in einem Aufruf. Die Tabelle mit den Definitionen besteht aus n 2 Worteinträgen. Mit AL = 00H steht im ersten Wort jeweils die logische Seitennummer, während das zweite Wort die physikalische Seitennummer enthält. Die logische Seitennummer 0FFFFH bedeutet, daß keine Zuordnung zwischen logischer und physikalischer Seite besteht. Mit AL = 01H bezieht sich diese auf die logischen Segmente anstatt auf die Seiten. Eine Tabelle der verfügbaren physikalischen Seiten läßt sich durch

die Funktion 5800H abfragen. Mit einem Aufruf läßt sich sowohl die Zuordnung zwischen physikalischen und logischen Seiten einstellen (Mapping) als auch aufheben (Unmapping). Eine Zuweisung der logischen Seite 0 bleibt ohne Konsequenzen, da die Funktion den Aufruf ignoriert. Eine Zuordnung läßt sich auf zwei Arten vornehmen. Einmal kann eine logische Seite in eine physikalische Seite eingeblendet (gemappt) werden. Dies ist eine Erweiterung der Funktion *Map Handle Page* (44H).

Reallocate Pages (Funktion 51H, Version 4.0)

Mit diesem Aufruf läßt sich die Zahl der logischen Seiten, die einem Handle zugeordnet sind, verändern.

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 51H (Reallocate Pages)     º
º BX: Zahl der Seiten            º
º DX: Handle                     º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º BX: Zahl der Seiten            º
Û--------------------------------ì
```

In BX wird die Zahl der benötigten Speicherseiten übergeben. Der Treiber versucht dann die geforderte Speichergröße dem Handle zuzuordnen. Nach dem Aufruf enthält BX die Zahl der wirklich zugewiesenen Seiten. Vor dem Aufruf ist in DX der Handlecode zu setzen. In AH wird nach dem Aufruf der Fehlerstatus (00H, 80H, 81H, 83H, 84H, 87H, 88H) zurückgegeben.

Get/Set Handle Attributes (Funktion 52H, Version 4.0)

Mit dieser Gruppe von Unterfunktionen lassen sich die Attribute der Handles (löschbar/unlöschbar) beeinflussen.

Get Handle Attributes (AL = 00H)

Mit diesem Aufruf lassen sich die Handle-Attribute abfragen. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 52H                        º
º AL: 00H (Get Handle Attribut)  º
º DX: Handle                     º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º AL: Attribut                   º
Û--------------------------------ì
```

Vor dem Aufruf ist in AX der Funktionscode 5200H zu setzen. In DX muß die betreffende Handlenummer eingetragen werden. Nach dem Aufruf gibt das Register den Fehlerstatus (00H, 80H, 81H, 83H, 84H, 8FH, 91H) zurück. Ist der Wert von AH = 0, enthält das Register AL das Attributbyte mit folgender Kodierung:

```
AL = 00H  Das Handle ist löschbar (volatile)
     01H  Das Handle ist fest (nonvolatile)
```

Die Unterscheidung in volatile und nonvolatile ist beim Warmstart wichtig. Der Treiber löscht bei diesem Vorgang alle als löschbar markierten Handles und hebt damit die Zuordnung zwischen Handle und Seiten auf. Dies ist zum Beispiel sehr hilfreich, wenn ein System in den Warmboot-Mode gebracht wird und die Handles automatisch an DOS zurückgegeben werden sollen. Andererseits läßt sich auch festlegen, daß ein Handle immer spezielle Seiten belegt. Dann kann diese Einstellung auch beim Warmstart nicht aufgehoben werden.

Set Handle Attribut (AL = 01H)

Mit diesem Aufruf lassen sich die Handle-Attribute neu setzen. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 52H                        º
º AL: 01H (Set Handle Attribut)  º
º BL: neues Attribut             º
º DX: Handle                     º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
Û--------------------------------ì
```

Vor dem Aufruf ist in AX der Funktionscode 5201H zu setzen. In DX muß die betreffende Handlenummer eingetragen werden. In BL wird das neue Attributbyte des Handles übergeben. Es gilt folgende Kodierung:

```
BL = 00H  Handle löschbar (volatile)
     01H  Handle fest (nonvolatile)
```

Nach dem Aufruf gibt das Register den Fehlerstatus (00H, 80H, 81H, 83H, 84H, 8FH, 90H, 91H) zurück. Ist der Wert von AH = 0, wurde das Attribut dem Handle zugeordnet.

Die Unterscheidung in volatile und nonvolatile ist beim Warmstart wichtig. Der Treiber löscht bei diesem Vorgang alle als löschbar markierten Handles und hebt damit die Zuordnung zwischen Handle und Seiten auf.

Get Attribute Capability (AL = 02H)

Mit diesem Aufruf läßt sich prüfen, ob die Karte nicht löschbare Attribute unterstützt. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 52H                        º
º AL: 02H (Get Attribut Capab.)  º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º AL: Attribut                   º
Û--------------------------------ì
```

Vor dem Aufruf ist in AX der Funktionscode 5202H zu setzen. In DX muß die betreffende Handlenummer eingetragen werden. Nach dem Aufruf gibt das Register den Fehlerstatus

(00H, 80H, 81H, 84H, 8FH) zurück. Ist der Wert von AH = 0, enthält das Register AL das Byte mit den Möglichkeiten des Handlers zur Unterstützung von Attributen:

```
AL = 00H  Der Treiber unterstützt nur löschbare
          (volatile) Handles.
     01H  Der Treiber unterstützt beide Attribute
          (volatile/nonvolatile Handles).
```

Dies bedeutet, daß sich ein Attribut mit der Unterfunktion 01H verändern läßt.

Bei als nicht löschbar gekennzeichneten Handles bleibt die Einstellung beim Systemstart erhalten. Es ist aber zu beachten, daß viele PCs während der Startphase den RAM-Refresh abstellen. In diesem Fall gehen die Daten im EMS-RAM verloren. Das Attribut sollte deshalb nur gesetzt werden, falls sicher ist, daß die Daten erhalten bleiben. Bei den vorliegenden EMS-Boards wird zur Zeit nur das Attribut 01H unterstützt.

Handle Name Functions (AH = 53H, Version 4.0)

Einem EMS-Handle läßt sich ein Name zuweisen. Dies kann mit der Funktion 53H erfolgen.

Bei der Installation werden die Felder mit den Handlenamen alle mit 8 Nullbyte belegt. Der Aufruf erlaubt einen String mit acht Zeichen als Namen zu übergeben. Dabei dürfen alle Zeichen zwischen 00H und FFH benutzt werden. 8 Nullbyte deuten auf einen nicht initialisierten Namen hin. Da in jedem Byte alle Zeichen zulässig sind, lassen sich die 8 Byte auch als Nummer interpretieren, die 64 Byte umfaßt. Die Funktion 53H kann den Handlenamen sowohl mit einem Namen als auch mit einer Nummer belegen. Die Funktion **5**4H vergibt dagegen einen Namen an den Handle. Dabei muß allerdings mindestens ein Byte ungleich 00H sein.

Get Handle Name (AL = 00H)

Die Funktion liest den Handlenamen mit den acht Zeichen aus. Der Name wird bei jedem Systemstart, bei einer Neubelegung des Handles und bei einer Freigabe des Handles mit Nullbytes belegt. Für den Aufruf gilt folgende Schnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 53H                        º
º AL: 00H (Get Handle Name)      º
º DX: Handle                     º
º ES:DI Adresse Feld             º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
Û--------------------------------ì
```

Vor dem Aufruf ist in AX der Funktionscode 5300H zu setzen. In DX muß die betreffende Handlenummer eingetragen werden. Weiterhin ist ein Feld mit 8 Byte zu definieren, dessen Anfangsadresse in ES:DI übergeben wird. Nach dem Aufruf enthält das Register den Fehlerstatus (00H, 80H, 81H, 83H, 84H, 8FH) zurück. Ist der Wert von AH = 0, enthält die Tabelle den Namen des Handles.

Set Handle Name (AL = 01H)

Die Funktion besetzt den Handlenamen mit den acht Zeichen. Für den Aufruf gilt folgende Schnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 53H                       º
º AL: 01H (Set Handle Name)     º
º DX: Handle                    º
º DS:SI Adresse Feld            º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Vor dem Aufruf ist in AX der Funktionscode 5301H zu setzen. In DX muß die betreffende Handlenummer eingetragen werden. Weiterhin ist ein Feld mit 8 Byte zu definieren, dessen Anfangsadresse in DS:SI übergeben wird. In diesem Feld ist der Name abzulegen. Nicht benutzte Bytes sind mit dem Wert 00H zu belegen.

Nach dem Aufruf enthält das Register den Fehlerstatus (00H, 80H, 81H, 83H, 84H, 8FH, A1H) zurück. Ist der Wert von AH = 0, wurde der Name aus der Tabelle dem Handle zugewiesen. Der Name muß eindeutig sein, d.h., er darf nur einem Handle zugewiesen werden. Der Name wird beim Systemstart und beim Allocate/Deallocate-Aufruf gelöscht.

Handle Directory Functions (AH = 54H, Version 4.0)

Mit dieser Funktion lassen sich die Namen aller Handles manipulieren. Bei der Installation werden die Felder mit den Handlenamen alle mit 8 Nullbyte belegt.

Get Handle Directory (AL = 00H)

Die Funktion liest alle 255 Handlenamen des Treibers mit den acht Zeichen aus. Für den Aufruf gilt folgende Schnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 54H                       º
º AL: 00H (Get Name Directory)  º
º ES:DI Adresse Feld            º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Vor dem Aufruf ist in AX der Funktionscode 5400H zu setzen. Weiterhin ist ein Feld mit 2550 Byte zu definieren, dessen Anfangsadresse in ES:DI übergeben wird. Nach dem Aufruf enthält das Register den Fehlerstatus (00H, 80H, 81H, 84H, 8FH) zurück. Ist der Wert von AH = 0, enthält die Tabelle die Namen aller aktiven Handles.

Pro Handle werden 10 Byte in der Tabelle belegt. Im ersten Wort steht die Handlenummer, gefolgt von dem 8 Byte langen Namen. Nicht benutzte Handles besitzen den Wert 00H in allen Bytes. Ein Handle ohne Namen besitzt zumindest einen Wert ungleich null im ersten Wort mit der Handlenummer.

Search for named Handle (AL = 01H)

Die Funktion durchsucht die Liste mit den Handlenamen nach einem spezifizierten Namen und gibt den zugehörigen Handlecode zurück. Für den Aufruf gilt folgende Schnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 54H                        º
º AL: 01H (Search for Handle)    º
º DS:SI Adresse Feld             º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º DX: Handle                     º
Û--------------------------------ì
```

Vor dem Aufruf ist in AX der Funktionscode 5401H zu setzen. Weiterhin ist ein Feld mit 8 Byte zu definieren, dessen Anfangsadresse in DS:SI übergeben wird. In diesem Feld ist der zu suchende Name abzulegen. Nicht benutzte Bytes sind mit dem Wert 00H aufzufüllen. Nach dem Aufruf enthält das Register den Fehlerstatus (00H, 80H, 81H, 84H, 8FH, A0H, A1H) zurück. Ist der Wert von AH = 0, wurde der Name in der Tabelle gefunden, dann steht in DX die zugehörige Handlenummer.

Get Total Handles (AL = 02H)

Die Funktion liest die Zahl der Handles, die maximal durch EMS unterstützt werden. Der Wert schließt den Handle 0 ein, der durch das Betriebssystem belegt wird. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 54H                        º
º AL: 02H (Get Total Handle)     º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º BX: Handlecount                º
Û--------------------------------ì
```

Nach dem Aufruf enthält AH den Fehlerstatus (00H, 80H, 81H, 84H, 8FH). Ist der Wert 0, steht in BX die Zahl der maximal unterstützten Handles.

Alter Page Map and Jump (Funktion 55H, Version 4.0).

Die Funktion liest die Liste mit dem *Memory Mapping Context*, blendet die spezifizierte Seite ein und übergibt die Kontrolle zur spezifizierten Adresse (Cross Page Branch). Dies entspricht einem JMP-FAR-Befehl beim 8086. Es gilt folgende Aufruf-schnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 55H (Alter Page Map)      º
º AL: Subfunktion               º
º DX: Handle                    º
º DS:SI Adresse Tabelle         º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

In AH ist der Code 55H zu übergeben. Der Wert in AL bestimmt, welche Unterfunktion aufgerufen wird.

Es gilt die Kodierung:

```
AL = 00H Die physikalische Seitennummer wird
         durch das rufende Programm übergeben.
     01H Die Segmentadresse wird durch das
         rufende Programm übergeben.
```

In **DX** steht der Handle, auf den sich der Aufruf bezieht. In Registerpaar **DS:SI** wird die Tabellenadresse mit dem *Memory Mapping Context* übergeben. Die Tabelle besitzt folgende Struktur:

```
Ö-------Ú---------------------------------------Ì
º Bytes º Feld                                  º
û-------é---------------------------------------À
º   4   º Adresse (JMP FAR) Sprungziel           º
û-------é---------------------------------------À
º   1   º Anzahl der Einträge im Feld            º
û-------é---------------------------------------À
º   4   º Zeiger auf das Feld mit folgendem      º
º       º Aufbau:                               º
º       º 2 Byte: Nummer der logischen Seite,   º
º       º die einzublenden ist.                 º
º       º 2 Byte: Nummer oder Segmentadresse    º
º       º der physikalischen Seite, in die die  º
º       º logische Seite einzublenden ist.      º
Û-------Ù---------------------------------------ì
```

Die Funktion gibt im Register AH den Status (00H, 80H, 81H, 83H, 84H, 8AH, 8BH, 8FH) zurück. Die Flags und alle Register bis auf AX werden vor dem Sprung zu der in der Tabelle angegebenen Position gesichert. Der alte Zustand des Fensters geht durch den Aufruf verloren und sollte explizit gesichert werden. Es dürfen auch Nullseiten ein-/ ausgeblendet werden.

Alter Page Map and Call (Funktion 56H, Version 4.0)

Die Funktion sichert den aktuellen Memory-Mapping-Context, aktualisiert den spezifizierten *Memory Mapping Context*, d.h. blendet die spezifizierte Seite ein und übergibt die Kontrolle zu der in der Tabelle angegebenen Adresse über einen CALL-FAR-Aufruf (Cross Page Call):

```
Ö------------------------------Ì
º      CALL:  INT 67           º
º                              º
º AH: 56H                      º
º AL: Subfunktion              º
º DX: Handle                   º
º DS:SI Adresse Tabelle        º
û------------------------------À
º      RETURN                  º
º AH: Status                   º
Û------------------------------ì
```

In AH ist der Code 56H zu übergeben. Der Wert in AL bestimmt, welche Unterfunktion aufgerufen wird. Es gilt die Kodierung:

```
AL = 00H  Die physikalische Seitennummer wird
          durch das rufende Programm übergeben.
     01H  Die Segmentadresse wird durch das
          rufende Programm übergeben.
```

In DX steht der Handle, auf den sich der Aufruf bezieht. In Registerpaar DS:SI wird die Adresse der Tabelle übergeben, in der der Memory-Mapping-Context**Fehler! Verweisquelle konnte nicht gefunden werden.**

```
Ö-------Ú--------------------------------------Ì
º Bytes º Feld                                 º
û-------é--------------------------------------À
º   4   º Adresse (CALL FAR) der Routine        º
û-------é--------------------------------------À
º   1   º Anzahl der Einträge im Feld mit den   º
º       º einzublendenden Seiten               º
û-------é--------------------------------------À
º   4   º Zeiger auf dieses Feld                 º
û-------é--------------------------------------À
º   1   º Zahl der Einträge im Feld mit den     º
º       º nach RETF einzublendenden Seiten.    º
û-------é--------------------------------------À
º   4   º Zeiger auf dieses Feld                 º
û-------Ù--------------------------------------À
º --- Das Feld besitzt für jeden Eintrag ---   º
º     folgenden Aufbau :                       º
û-------Ú--------------------------------------À
º   2   º Nummer der logischen Seite            º
û-------é--------------------------------------À
º   2   º Nummer der Segmentadresse der         º
º       º physikalischen Seite.                ºº
Û-------Ù--------------------------------------ì
```

Nach dem Aufruf blendet die Funktion wieder die alten Zustände aus der Tabelle mit den »restauriertenDie Funktion gibt im Register AH den Status (00H, 80H, 81H, 83H, 84H, 8AH, 8BH, 8FH) zurück. Die Flags und alle unbenutzten Register werden vor dem CALL zu der in der Tabelle angegebenen Position gesichert.

Get Page Map Stack Space Size (AL = 02H)

Die Funktion 5601H benötigt zusätzlichen Stackraum zur Ausführung. Dieser Wert läßt sich über die Subfunktion 02H ermitteln. Es gilt folgende Aufrufschnittstelle:

```
Ö------------------------------Ì
º      CALL:  INT 67           º
º                              º
º AH: 56H                      º
º AL: 02 (Get Stack Size)      º
û------------------------------À
º      RETURN                  º
º AH: Status                   º
º BX: Stackbedarf (Byte)       º
Û------------------------------ì
```

In AL ist der Code 02H zu übergeben, um die Unterfunktion zu aktivieren. Nach dem Aufruf enthält das Register AH den Fehlerstatus (00H, 80H, 81H, 84H, 8FH). Bei erfolgreichem Aufruf wird in BX der zusätzliche Stackbedarf in Bytes zurückgegeben.

Move/Exchange Memory Region (Funktion 57H, Version 4.0)

Diese Gruppe von Unterfunktionen erlaubt das Verschieben/Austauschen von Speicherregionen.

Move Memory Region (AL = 00H)

Diese Funktion verschiebt Daten zwischen zwei angegebenen Speicherbereichen. Dies kann sich sowohl auf die Seiten des EMS-Boards als auch auf den DOS-RAM-Bereich beziehen. Es gilt folgende Aufrufschnittstelle:

```
Ö------------------------------Ì
º      CALL:  INT 67           º
º                              º
º AH: 57H                      º
º AL: 00H (Move Memory)        º
º DS:SI Adresse Steuerblock    º
û------------------------------À
º      RETURN                  º
º AH: Status                   º
Û------------------------------ì
```

Vor dem Aufruf ist das Register AX auf den Wert 5700H zu setzen. Im Registerpaar DS:SI findet sich die Adresse (Segment:Offset) auf den Steuerblock. Dieser Block enthält die Informationen über die zu verschiebenden Bereiche. Es gilt dabei folgender Aufbau für den Block:

```
Ö-------Ú-------------------------------------------------Ì
º Bytes º Feld                                            º
û-------é-------------------------------------------------À
º   4   º Länge des zu verschiebenden Blocks in Byte       º
º   1   º Steuerbyte (Quellblock)                         º
º       º  0 = Quellblock im DOS-Bereich                  º
º       º  1 = Quellblock im EMS-Bereich                  º
º   2   º Handle Quellblock                               º
º   2   º Offset Quellblock oder Seitenselektor           º
º   2   º logische Seitennummer des Quellblocks bei EMS   º
º       º oder Segmentadresse bei DOS-Speicherbereichen   º
º   1   º Steuerbyte (Zielblock)                          º
º       º  0 = Ziel im DOS-Bereich                        º
º       º  1 = Ziel im EMS-Bereich                        º
º   2   º Handle-Zielblock                                º
º   2   º Offset-Zielblock oder Seitenselektor            º
º   2   º logische Seitennummer des Zielblocks bei EMS    º
º       º oder Segmentadresse bei DOS-Speicherbereichen   º
Û-------Ù-------------------------------------------------ì
```

Wird ein Speicherblock aus dem DOS-Bereich in den EMS-Bereich verschoben, ist der Handle des Quellblocks ungültig, d.h. auf 0 zu setzen. Wird die Adresse des Quell- oder Zielblockes direkt angegeben, enthält das auf das Steuerbyte folgende Wort die

Offsetadresse. Im folgenden Wort steht dann die Segmentadresse. Sonst wird in diesem Feld die logische Seitennummer angegeben.

Der dem Handle zugewiesene Expanded-Memory-Bereich wird als ein lineares Feld, beginnend mit der logischen Seite 0 bis zur logischen Seite n betrachtet. Die angegebenen Speicherbereiche dürfen sich nicht überlappen. Der Context des aktuellen Fensters wird nach Ausführung der Funktion wiederhergestellt, d.h. eine eingeblendete und dann verschobene Seite befindet sich nach wie vor im EMS-Fenster.

In AH wird nach dem Aufruf der Status zurückgegeben (00H, 80H, 81H, 83H, 84H, 8AH, 8FH, 92H, 93H, 94H, 95H, 96H, 98H, A2H).

Exchange Memory Region (AH = 01H)

Mit dieser Unterfunktion lassen sich Daten zwischen zwei Datenbereichen austauschen. Dies kann sowohl zwischen dem DOS-Bereich und EMS als auch zwischen EMS-Seiten erfolgen.

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 57H                        º
º AL: 01H (Exchange Memory)      º
º DS:SI Adresse Steuerblock      º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
Û--------------------------------ì
```

Vor dem Aufruf ist das Register AX auf den Wert 5701H zu setzen. Im Registerpaar DS:SI findet sich die Adresse (Segment:Offset) auf den Steuerblock. Dieser Block enthält die Informationen über die zu verschiebenden Bereiche und besitzt den bei der Funktion 5700H beschriebenen Aufbau. Es gilt dabei das bei der Subfunktion 00H definierte Format. In AH wird nach dem Aufruf der Status zurückgegeben (00H, 80H, 81H, 83H, 84H, 8AH, 8FH, 93H, 94H, 95H, 96H, 97H, 98H, A2H).

Mappable Physical Address Array (Funktion 58H, Version 4.0)

Diese Funktion erlaubt die Abfrage der Tabelle mit dem physikalischen Speicheraufbau. Diese Tabelle ist unabhängig von den Vorgaben verschiedener Hersteller. Der Aufruf ist funktionell äquivalent mit der EEMS-Funktion 68H. Die EEMS-Funktion 60H enthält eine Untermenge der Funktion 68H.

Get Mappable Physical Address Array (AL = 00H)

Die Tabelle mit den Segmentadressen aller vorhandenen physikalischen 16-Kbyte-Seiten mit der zugehörigen Seitennummer wird gelesen.

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 58H                        º
º AL: 00H (Get Array)            º
º ES:DI Adresse Datenfeld        º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º CX: Zahl der Einträge          º
Û--------------------------------ì
```

Das Feld enthält dann die Übersetzung zwischen der physikalischen Seitennummer und der aktuellen Segmentadresse der übereinstimmenden Seiten. Mit dem Untercode 00H läßt sich das Feld mit den Segmentadressen und den physikalischen Seiten-nummern für jede physikalische EMS-Seite abfragen.

In AH steht nach dem Aufruf der Fehlercode (00H, 80H, 81H, 84H, 8FH). Beim Code AH = 0 findet sich in CX die Zahl der Einträge in der Tabelle. Diese Tabelle wird durch ES:DI adressiert und enthält für jeden Eintrag zwei Worte. Das erste Wort in der Tabelle ist der Selektor der physikalischen Seite. Das zweite Wort ist die .i.physikalische Seite;nnummer. Die Tabelle ist nach aufsteigenden Segmenten geordnet.

Get Physical Page Address Array Entries (AL = 01H)

Diese Funktion gibt ein Wort zurück, welches die Zahl der Tabelleneinträge der Funktion 5800H angibt. Dieser Wert entspricht der Zahl der physikalischen Seiten im System.

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 58H                        º
º AL: 01H (Get Adresse)          º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º CX: Zahl der Einträge          º
Û--------------------------------ì
```

In AH steht nach dem Aufruf der Status (00H, 80H, 81H, 84H, 8FH). In CX findet sich die Zahl der Einträge der Tabelle aus AX = 5800H. Wird der Wert mit 4 multipliziert, entspricht das Ergebnis der Tabellenlänge der Funktion 5800H in Bytes.

Get Expanded Memory Hardware Information (Funktion 59H, Version 4.0)

Diese Funktion gibt Informationen über die unterlagerte Hardware zurück. Der Aufruf sollte nur durch das Betriebssystem genutzt werden. Für das Betriebssystem besteht die Möglichkeit, die Funktion zu sperren (Funktion 5DH). Es gelten folgende Unterfunktionen:

Get EMS Hardware Info (AL = 00H)

Der Aufruf gibt ein Feld mit Informationen über die Hardwarekonfiguration des EMS-Boards zurück.

```
Ö------------------------------Ì
º      CALL:  INT 67           º
º                              º
º AH: 59H                      º
º AL: 00H (Get Hardware Info)  º
º ES:DI Adresse Infoblock      º
û------------------------------À
º      RETURN                  º
º AH: Status                   º
Û------------------------------ì
```

In ES:DI ist die Adresse eines 10 Byte langen Blockes mit folgender Struktur zu übergeben:

```
Ö-------Ú-----------------------------------------------Ì
º Bytes º Feld                                          º
û-------é-----------------------------------------------À
º   2   º "Roh"-Seiten Größe in Paragraphen (16 Byte)    º
º   2   º Zahl der "alternate register sets"            º
º   2   º Größe der Page Maps                           º
º   2   º Zahl der "alternate register sets" für DMA    º
º   2   º DMA-Operation                                 º
Û-------Ù-----------------------------------------------ì
```

Im ersten Wort wird die Größe der intern verwendeten Seiten (raw pages) in Paragraphen angegeben. Diese Größe muß nicht immer 16 Kbyte betragen. Das folgende Wort enthält die Zahl der zusätzlichen Registersätze zum Ein-/Ausblenden von Seiten. Wort 3 gibt die Größe des Speicherbereiches zur Aufnahme des *Mapping Context* an. Der Wert wird auch durch die Funktion 4E03H zurückgegeben. Das folgende Wort gibt die Zahl der Registersätze an, die sich direkt mit DMA-Kanälen koppeln lassen. Diese erlauben einen direkten DMA-Transfer zwischen dem DOS-Speicher und dem EMS-RAM. Das letzte Wort enthält Informationen über die Funktion der DMA-Registersätze. Der Wert 0 bedeutet, die DMA-Registersätze verhalten sich wie bei der Funktion 5BxxH beschrieben. Beim Wert 01 existiert nur ein DMA-Registersatz, mit dem alle Kanäle gekoppelt werden. Dann ist dieser Registersatz gegebenenfalls neu zu laden, wenn der Kanal gewechselt wird.

In AH wird der Fehlerstatus zurückgegeben (00H, 80H, 81H, 84H, 8FH, A4H). Die Funktion sollte nur durch das Betriebssystem benutzt werden. Das Betriebssystem kann die Funktion auch sperren.

Get Unallocated Raw Page Count (AL = 01H)

```
Der Aufruf gibt die Zahl der unbelegten »RohseitenFehler! Verweisquelle konnte nicht
gefunden werden.Ö------------------------------Ì
º      CALL:  INT 67           º
º                              º
º AH: 59H                      º
º AL: 01H (Get Raw Page Count) º
û------------------------------À
º      RETURN                  º
º AH: Status                   º
º BX: freie Rohseiten          º
º DX: Zahl der Rohseiten       º
Û------------------------------ì
```

Die EMS-Hardware darf auch Seiten kleiner 16 Kbyte (raw pages) verwenden. Nach außen sind aber immer 16 Kbyte sichtbar. Die fehlenden n Kbyte sind dann nicht ansprechbar. Das Betriebssystem kann dann die kleineren Seiten zum Ablegen von Programmen nutzen. Dies erlaubt eine ökonomischere Nutzung des EMS-Bereiches.

Eine EMS-Seite wird als Rohseite bezeichnet, falls sie eine Untermenge einer 16-Kbyte-Seite bildet.

Nach dem Aufruf steht in AH der Fehlercode (00H, 80H, 81H, 84H, 8FH).

Allocate Raw Pages (Funktion 5AH, Version 4.0)

Mit der Funktion läßt sich eine Rohseite durch das Betriebssystem belegen. Dabei wird der Seite vom Treiber ein EMS-Handle zugewiesen.

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 5AH                        º
º AL: 00H Untercode              º
º BX: Zahl der Seiten            º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º DX: Handle                     º
Û--------------------------------ì
```

In AL wird ein Code zur Auswahl der Unterfunktion übergeben:

```
AL = 00H  belege eine Standardseite
          von 16 Kbyte
     01H  belege eine Rohseite
          (kleiner 16 Kbyte)
```

Mit AL = 0 ist in BX die Zahl der benötigten 16-Kbyte-Seiten zu übergeben. Bei AL = 01H wird in BX die Zahl der benötigten Rohseiten kleiner 16 Kbyte spezifiziert. Nach dem Aufruf findet sich in AH der Fehlerstatus (00H, 80H, 81H, 84H, 85H, 87H, 88H). In DX findet sich nach dem Aufruf der EMM-Handlecode (1 bis 255) der zugewiesenen Seite. Die Funktion sollte nur durch das Betriebssystem aufgerufen werden. Der Treiber erkennt später am Handlecode, ob es sich um eine 16-Kbyte-Seite oder um eine Rohseite handelt. Die Seite 0 darf dabei benutzt werden.

Alternate Map Register Set-DMA Registers (Funktion 5BH, Version 4.0)

Diese Gruppe von Unterfunktionen spricht die DMA-Register des EMS-Boards an.

Get Alternate Map Register Set (AL = 00H)

Die Funktion liest die Daten des gesicherten Hardwarezustandes in eine Tabelle zurück:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5BH                       º
º AL: 00H Untercode             º
û-------------------------------À
º      RETURN                   º
º AH: Fehlercode                º
º BL: 0                         º
º ES:DI Adresse Tabelle         º
º BL: <> 0 Nummer alternate     º
º          Registersatz         º
Û-------------------------------ì
```

Findet sich nach dem Aufruf im Register BL der Wert 0, zeigt das Registerpaar ES:DI auf die Map-Register-Context-Sicherungstabelle. Alle Werte un-gleich 0 in BL geben die Nummer des *Alternate*«Registersatzes) an. In AH wird ein Fehlercode (00H, 80H, 84H, 81H, 8FH, A4H) zurückgegeben.»

Set Alternate Map Register Set (AL = 01H)

Die Funktion setzt die Daten des gesicherten Hardwarezustandes in eine Tabelle zurück:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5BH                       º
º AL: 01H Untercode             º
º BL: Nummer Registersatz       º
º ES:DI Adresse Tabelle         º
û-------------------------------À
º      RETURN                   º
º AH: Fehlercode                º
Û-------------------------------ì
```

In Register BL wird die Nummer des Alternate Registersatzes übergeben, während die Register ES:DI die Adresse der Tabelle mit den Daten enthalten. In AH wird ein Fehlercode (00H, 80H, 81H, 84H, 8FH, 9AH, 9CH, 9DH, A3H, A4H) zurückgegeben.

Get Alternate Map Save Array Size (AL = 02H)

Die Funktion liest die Größe der Sicherungstabelle:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5BH                       º
º AL: 02H Untercode             º
û-------------------------------À
º      RETURN                   º
º AH: Fehlercode                º
º DX: Größe der Tabelle         º
Û-------------------------------ì
```

Nach dem Aufruf enthält das Register DX die Größe der Tabelle in Byte. In AH wird ein Fehlercode (00H, 80H, 84H, 81H, 8FH, A4H) zurückgegeben.

Allocate Alternate Register Set (AL = 03H)

Die Funktion fordert einen alternativen Registersatz an.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5BH                       º
º AL: 03H Untercode             º
û-------------------------------À
º      RETURN                   º
º AH: Fehlercode                º
º BL: Nummer Registersatz       º
Û-------------------------------ì
```

Findet sich nach dem Aufruf im Register BL der Wert 0, werden keine weiteren Registersätze unterstützt. Dann sollte die Get-Alternate-Funktion aktiviert werden. Sonst enthält BL die Nummer des Registersatzes. In AH wird ein Fehlercode (00H, 80H, 81H, 84H, 8FH, 9BH, A4H) zurückgegeben.

Deallocate Alternate Map Register Set (AL = 04H)

Die Funktion gibt einen alternativen Registersatz wieder frei.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5BH                       º
º AL: 04H Untercode             º
º BL: Nummer Registersatz       º
û-------------------------------À
º      RETURN                   º
º AH: Fehlercode                º
Û-------------------------------ì
```

In BL ist die Nummer des freizugebenden Registersatzes zu übergeben. In AH wird ein Fehlercode (00H, 80H, 81H, 84H, 8FH, 9CH, 9DH, A4H) zurückgegeben.

Allocate DMA-Register Set (AL = 05H)

Die Funktion fordert einen DMA-Registersatz vom EMM an.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5BH                       º
º AL: 05H Untercode             º
û-------------------------------À
º      RETURN                   º
º AH: Fehlercode                º
º BL: DMA-Registersatz          º
Û-------------------------------ì
```

Mit diesem DMA-Registersatz sind Übertragungen in den Block möglich, ohne daß die betreffende Seite in das Fenster eingeblendet werden muß. Die Nummer des Registersatzes wird in BL zurückgegeben. In AH wird ein Fehlercode (00H, 80H, 81H, 84H, 8FH, 9BH, A4H) zurückgegeben.

Enable DMA on Alternate Map Register Set (AL = 06H)

Die Funktion gibt den DMA-Transfer über den alternativen Registersatz frei.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5BH                       º
º AL: 06H Untercode             º
º BL: DMA-Registersatz          º
º DL: DMA-Kanal                 º
û-------------------------------À
º      RETURN                   º
º AH: Fehlercode                º
Û-------------------------------ì
```

Mit diesem Aufruf wird der in BL übergebene Registersatz mit dem in DL übergebenen DMA-Kanal gekoppelt. In AH wird ein Fehlercode (00H, 80H, 81H, 84H, 8FH, 9AH, 9CH, 9DH,9EH, 9FH, A4H) zurückgegeben.

Disable DMA on Alternate Map Register Set (AL = 07H)

Die Funktion sperrt den DMA-Transfer über den alternativen Registersatz.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5BH                       º
º AL: 07H Untercode             º
º BL: DMA-Registersatz          º
û-------------------------------À
º      RETURN                   º
º AH: Fehlercode                º
Û-------------------------------ì
```

In AH wird ein Fehlercode (00H, 80H, 81H, 84H, 8FH, 9AH, 9CH, 9DH, 9EH, 9FH, A4H) zurückgegeben.

Deallocate DMA-Register Set (AL = 08H)

Die Funktion gibt den DMA-Registersatz wieder frei.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5BH                       º
º AL: 08H Untercode             º
º BL: DMA-Registersatz          º
û-------------------------------À
º      RETURN                   º
º AH: Fehlercode                º
Û-------------------------------ì
```

In AH wird ein Fehlercode (00H, 9CH, 9DH, A4H) zurückgegeben.

Die Funktion sollte nur durch das Betriebssystem benutzt werden und kann auch durch dieses gesperrt werden. Die Funktion erledigt die gleichen Aufgaben wieder EEMS-Aufruf 6AH.

Prepare EMS Hardware for Warm Boot (Funktion 5CH, Version 4.0)

Der Aufruf bereitet die EMS-Hardware für einen Warmstart vor.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5CH                       º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

In AH wird der Status (00H, 80H, 81H, 84H) zurückgegeben. Die Funktion nimmt an, daß die nächste Operation des Betriebssystems ein Warmstart ist. Falls eine Applikation Speicher in den Bereich unter 640 Kbyte einblendet, muß der Aufruf vor dem Warmstart erfolgen.

Enable/Disable OS Function Set Functions (Funktion 5DH, Version 4.0)

Die Funktion erlaubt dem Betriebssystem oder dem Treiber spezielle Aufrufe zu blockieren. Dadurch lassen sich durch das Betriebssystem bestimmte Aufrufe für Anwenderprogramme sperren.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 5DH                       º
º AL: 00H (enable OS Funktion)  º
º     01H (disable OS Funktion) º
º     02H (Get Access Key)      º
º BX: Zugriffsschlüssel         º
º CX: Zugriffsschlüssel         º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º BX: Zugriffsschlüssel         º
º CX: Zugriffsschlüssel         º
Û-------------------------------ì
```

In AL wird der Untercode übergeben. Mit AL = 00H werden die Betriebssystemfunktionen freigegeben und mit AL = 01H gesperrt. Die Unterfunktion 02H gibt den Zugriffscode zurück. Dabei wird die Speicherverwaltung zurückgesetzt. Der Schlüssel wird anschließend beim nächsten Aufruf zurückgegeben.

Zum Aufruf der Funktionen ist in BX und CX der Zugriffsschlüssel, der beim ersten Aufruf nach der Funktion 5D02H zurückgegeben wurde, zu übergeben. Der Schlüssel wird beim Aufruf der Funktionen 5D00H und 5D01H zurückgegeben. Als Fehlercodes sind in AH die Werte 00H, 80H, 81H, 84H, 8FH und A4H vorgesehen. Mit dem Aufruf lassen sich Zugriffe auf die folgenden EMS-Funktionen beeinflussen:

```
Funktion 59H:  Get Expanded Memory Hardware
               Information.
         5BH:  Alternate Map Register Sets.
         5DH:  Enable/Disable Operating System
               Functions.
```

Die Funktion **5EH** und **5FH** sind unter EMS 4.0 nicht definiert.

12.5 Die EEMS-Funktion 60H bis 6AH

Mit diesen Funktionscodes des INT 67H lassen sich die Funktionen des EEMS 3.2 Managers ansprechen. Die Funktionen sind unter EMS nicht definiert.

Get Physical Window Array (Funktion 60H, EEMS 3.2)

Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 60H (Get Window Array)    º
º ES:DI Adresse Puffer          º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º AL: Zahl der Einträge         º
Û-------------------------------ì
```

Es ist ein Puffer mit n Byte anzulegen, in dem die Funktion die Einträge zurückgibt. Die Zahl der Einträge findet sich nach dem Aufruf in AL, AH enthält den Statuscode.

Generic Accelerator Card Support (Funktion 61H, EEMS 3.2)

Mit dieser Funktion lassen sich Daten des Boards lesen. AST-Research hat hier eine Definition veröffentlicht, die allerdings nur für Hersteller von Accelerator-Karten von Interesse ist.

Get Addresses of All Page Frames in System (Funktion 68H, EEMS 3.2)

Die Funktion stimmt mit der EMS-Funktion 58H überein.

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 68H (Get Address)         º
º ES:DI Adresse Puffer          º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
º AL: Zahl der Einträge         º
Û-------------------------------ì
```

In ES:DI ist die Adresse eines Puffers zu übergeben. Nach dem Aufruf findet sich in AL die Zahl der Einträge in der Tabelle.

Map Page Into Frame (Funktion 69, EEMS 3.2)

Der Aufruf ist unter EMS nicht definiert. Es gilt folgende Aufrufschnittstelle:

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 69H (Map Page)            º
º AL: Frame Nummer              º
º BX: Seitennummer              º
º DX: Handle                    º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Die Funktion ist mit dem EMS-Aufruf 44H vergleichbar.

Page Mapping (Funktion 6AH, EEMS 3.2)

Der Aufruf ist unter EMS nicht definiert. Es gilt folgende Aufrufschnittstelle:

Save partial Page Map (AL = 00H)

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 6AH (Map Page)             º
º AL: 00 (Save partial Map)      º
º CH: first Page Frame           º
º CL: Zahl der Frames            º
º ES:DI Zeiger auf Puffer        º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º AL: Feldgröße in Byte          º
Û--------------------------------ì
```

Die Map wird in den durch ES:DI adressierten Buffer gespeichert.

Restore partial Page Map (AL = 01H)

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 6AH (Map Page)             º
º AL: 01 (Restore partial Map)   º
º CH: first Page Frame           º
º CL: Zahl der Frames            º
º DS:SI Zeiger auf Puffer        º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º AL: Feldgröße in Byte          º
Û--------------------------------ì
```

Save and Restore partial Page Map (AL = 02H)

```
Ö--------------------------------Ì
º      CALL:  INT 67             º
º                                º
º AH: 6AH (Map Page)             º
º AL: 02 (Save/Restore)          º
º CH: first Page Frame           º
º CL: Zahl der Frames            º
º ES:DI Zeiger auf Puffer        º
º DS:SI Zeiger auf Puffer        º
û--------------------------------À
º      RETURN                    º
º AH: Status                     º
º AL: Feldgröße in Byte          º
Û--------------------------------ì
```

In ES:DI findet sich die Adresse des Puffers zur Aufnahme der aktuellen Page Map. In DS:SI ist die Adresse des Puffers mit der neuen Page Map gespeichert.

Die Unterfunktionen 04H und 05H sind wie folgt belegt:

```
04h switch to standard map register setting
05h switch to alternate map register setting
```

Deallocate Pages mapped to Frames in Conventional Memory (AL = 06H)

```
Ö-------------------------------Ì
º      CALL:  INT 67            º
º                               º
º AH: 6AH (Map Page)            º
º AL: 06 (Deallocate)           º
º CH: first Page Frame          º
º CL: Zahl der Frames           º
û-------------------------------À
º      RETURN                   º
º AH: Status                    º
Û-------------------------------ì
```

Die Aufrufe sind vergleichbar mit der EMS-Funktion 4EH.

EMS-Zugriffe durch MS-DOS

Ab der DOS-Version 4.0 wird der EMS-Speicher benutzt. DOS greift über den Multiplexerinterrupt INT 2F auf den EMS-Treiber zu. Es sind nur die Funktionen AL = 00H (Installation Check) und AL = 01H (Get hidden Frame Info) implementiert. Näheres findet sich bei der Beschreibung des INT 2F.

12.6 Die VCPI-Schnittstelle

Die Intel Prozessoren der 80386-Serie (und aufwärts kompatible Typen) besitzen 3 Modi. Neben den Real-Mode existiert noch der Protected-Mode und der Virtuelle-86-Mode. Im Protected Mode kann von den Möglichkeiten zur direkten Adressierung des Extended Memory oberhalb 1 Mbyte Gebrauch gemacht werden. Insbesondere unter DOS reizte diese Möglichkeit die Entwickler, so daß bereits vor einigen Jahren Programme entstanden, die die Umschaltung von DOS in den Protected Mode erlauben. Diese Programme werden als DOS-Extender bezeichnet. Produkte wie LOTUS 1-2-3 Version 3.0 benutzen diese Extender um Code und Daten im Protected Mode zu speichern. Der Extender sorgt dann für die Speicherverwaltung sowie die Übergänge zu den DOS-Funktionen im Real-Mode.

Als Problem erwies sich aber bald die unterschiedliche Verwaltung des Extended Memory. Neben dem XMS-Manager existierte noch der INT 15 und es sind sicherlich weitere Möglichkeiten denkbar. Um eine einheitliche Technik zur Speicherverwaltung für DOS-Extender zu schaffen, definierten 1987 mehrere Firmen (PharLap, Quarterdeck, Qualitas, LOTUS, Autodesk und andere) einen gemeinsamen Standard. Dieser wurde als *Virtual Control Program Interface (VCPI)* bekannt. In der Spezifikation wird beschrieben, wie Programme, *EMS-Emulatoren* und *DOS-Extender* das Extended Memory verwalten müssen.

Die Programme, welche das VCPI benutzen, unterteilen sich in Clients und einen Server. Der Server ist das Programm, welches die Verwaltung des Speichers übernimmt. Die Clients sind Anwenderprogramm, die die Leistungen des Servers für ihre Zwecke nutzen.

Der WINDOWS-EMM386-Treiber unterstützt zum Beispiel die VCPI-Spezifikation und fungiert als Server. Zur Kommunikation mit dem Server existieren zwei Möglichkeiten:

- Aufruf über den INT 67 im Real-Mode des Prozessors.
- Aufruf über einen CALL-Aufruf im Protected-Mode des Prozessors.

Möchte eine Software mit dem VCPI-Server kommunizieren, muß sie zuerst prüfen, ob der Server vorhanden ist. Vor dem Aufruf des INT 67H ist ein Test erforderlich, ob der Vektor auch belegt ist. Dies kann mit der Technik, die beim EMS-Manager beschrieben ist, erfolgen. Zusätzlich ist zu prüfüen, ob als Prozessor der 80386/80486 vorhanden ist. Dann kann über einen INT 67-Aufruf der Server aktiviert werden. Für die Aufrufe im Protected Mode ist weiterhin der Einsprungpunkt zu ermitteln. Genauere Informationen zur VCPI-Version 1.0 finden sich auf den folgenden Seiten.

VCPI Installation Check (INT 67, AX = DE00H)

Das VCPI-Programm belegt einige Funktionen des INT 67. Mit der Unterfunktion AX = DE00H läßt sich prüfen, ob ein Server installiert ist.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE00H Install Check       º
û--------------------------------À
º         Return:                º
º AH = 00H VCPI ist vorhanden    º
º BH = Hauptversion              º
º BL = Unterversion              º
º AH <> 0 VCPI nicht vorhanden   º
Û--------------------------------ì
```

Nach dem Aufruf enthält das Register AH den Statuscode. Nur bei AH = 00H ist ein VCPI-Server geladen. Andernfalls wird der INT 67-Aufruf den Fehlercode AH = 84H zurückgeben (siehe EMS). Bei AH = 00H enthält BX die Versionsnummer des VCPI-Servers.

VCPI Get Protected Mode Interface (INT 67, AX = DE01H)

Für Aufrufe aus dem Protected Mode muß die Einsprungadresse in den Server bekannt sein, da diese per CALL und nicht per INT 67 erfolgen. Im Protected Mode stehen die DOS-Interrupts nicht mehr zur Verfügung. Die Funktion AX = DE01H dient zur Ermittlung der Einsprungadresse.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE01H Get Interface Adr.  º
º ES:DI Zeiger auf Puffer        º
º DS:SI Zeiger zu Descriptoren   º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º DI = Zeiger in Seitentabelle   º
º EBX= Offset Einspungpunkt      º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

In DS:SI wird ein Zeiger auf die drei Descriptor-Tabellen übergeben. Der erste der 3 Descriptor-Tabellen, die *Global Descriptor Table* (GDT) enthält nach dem Aufruf den

Descriptor für das VCPI-Codesegment und bildet zusammen mit dem EBX-Register die Einsprungadresse für Protected-Mode-Aufrufe des Servers. Diese Adresse kann dann mit einem CALL FAR angesprungen werden. Die beiden anderen Descriptor-Tabellen kann der Server beliebig belegen. Das GDT-Segment muß im Codebereich auf Seite 0 liegen.

Im VCPI wird der Speicher in 4 Kbyte großen Blöcken verwaltet. Dies ist die Blockgröße, die durch die Prozessoren der 80386-Reihe vorgegeben wird. In ES:DI ist beim Aufruf ein Zeiger auf die Seitentabelle (0. Page) des Clients für die 4-Kbyte-Seiten zu übergeben. Dort legt der Server die von ihm benutzten Seitennummern im 1-Mbyte-Bereich, sowie eventuell oberhalb von 1 Mbyte benutzte Speichernummern, ab. Die Tabelle darf der Client weder verschieben noch verändern. Lediglich die Bits 9-11 dürfen aus Kompatibilitätsgründen gelöscht werden.

Nach dem Aufruf enthält DI einen Zeiger auf den ersten nicht mehr initialisierten Eintrag in der Seitentabelle. EBX enthält die Offsetadresse für den Einsprung in die Protected Mode-Funktionen.

VCPI Get Maximum physical Memory (INT 67, AX = DE02H)

Mit diesem Aufruf läßt sich die Adresse der höchsten physikalischen 4-Kbyte-Speicherseite feststellen.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE02H Get Memory Adr.     º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º EDX= Memory Adresse            º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

Im Register AX ist der Wert DE02H beim Aufruf zu übergeben. Anschließend zeigt AL = 00H einen erfolgreichen Aufruf an. In EDX findet sich die physikalische Adresse der höchsten 4-Kbyte-Speicherseite. Aus Kompatibilitätsgründen müssen die 12 untersten Bits auf 0 gesetzt werden. Der Client kann mit diesem Aufruf gegebenenfalls seine Speicherstrukturen einrichten.

VCPI Get Number of free 4K Pages (INT 67, AX = DE03H)

Mit diesem Aufruf läßt sich die Größe des freien Speichers abfragen.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE03H Get free Memory     º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º EDX= free Memory               º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

In AX ist beim Aufruf der Code DE03H zu übergeben. Nach einem erfolgreichen Aufruf enthält AH den Wert 00H. In EDX findet sich dann die Zahl der freien 4-Kbyte-Pages. Dies ist die Zahl der freien Seiten, die im VCPI-Speicherpool allen Tasks zur Verfügung

stehen. Bedingt durch Systemrestriktionen kann es sein, daß für einen Task nur kleinere Blöcke reserviert werden dürfen.

Der Aufruf ist im Protected Mode ebenfalls verfügbar.

VCPI Allocate a 4K Page (INT 67, AX = DE04H)

Mit diesem Aufruf läßt sich eine 4-Kbyte-Seite allocieren.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE04H Allocate 4K Memory  º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º EDX= Adresse 4K Page           º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

In AX ist der Code DE04H beim Aufruf zu übergeben. Dann reserviert der Server eine 4-Kbyte-Seite für den Clienten. Bei erfolgreichem Aufruf ist AH = 00H und in EDX findet sich die physikalische Adresse der reservierten Seite. Bei Werten AH <> 0 konnte die Seite nicht reserviert werden, der Inhalt von EDX ist undefiniert. Aus Kompatibilitätsgründen müssen die unteren 12 Bits der physikalischen Adresse zu 0 maskiert werden.

Der Client muß später die reservierten Seiten explizit freigeben, da sie sonst bis zum Systemstart allociert bleiben. Die Funktion ist auch im Protected Mode verfügbar.

VCPI Free a 4K Page (INT 67, AX = DE05H)

Mit diesem Aufruf läßt sich eine 4-Kbyte-Seite freigeben.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE05H Free a 4K Page      º
º EDX= Adresse 4K Page           º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

In AX ist der Code DE05H beim Aufruf zu übergeben. Die Seite muß vorher durch die Funktion DE04H allociert worden sein. Die dabei zurückgegebene physikalische Adresse der Seite ist nun in EDX zu übergeben. Bei erfolgreichem Aufruf ist AH = 00H und die Seite wurde wieder freigegeben. Bei Werten AH <> 0 konnte die Seite nicht freigegeben werden. Aus Kompatibilitätsgründen müssen die unteren 12 Bits der physikalischen Adresse zu 0 maskiert werden.

Die Funktion ist auch im Protected Mode verfügbar.

VCPI Get physikal Adress of Page in first MB (INT 67, AX = DE06H)

Mit diesem Aufruf läßt sich die physikalische Adresse eine 4-Kbyte-Seite innerhalb des ersten Megabyte im Adressraum feststellen.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE06H Get 1. MB Adress    º
º CX = Seitennummer              º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º EDX= physikal. Seitennummer    º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

Der Aufruf wird hauptsächlich vom Clienten verwendet, um die Lage einer EMS-Seite im 1-Mbyte-Adressraum festzustellen. Falls dieser Speicher im Protected Mode benutzt werden soll, muß die physikalische Adresse vorliegen.

Im Register CX ist die Seitennummer der gewünschten Seite zu übergeben. Als Wert ist die Anfangsadresse der Seite, um 12 Bit nach rechts geshiftet, zu übergeben. Nach einem erfolgreichen Aufruf ist AH = 00H und in EDX steht die physikalische Speicheradresse der 4-Kbyte-Seite. Bei AH <> 0 liegt ein Fehler beim Aufruf vor. Aus Kompatibilitätsgründen sind die unteren 12 Bit der physikalischen Adresse zu 0 zu maskieren.

VCPI Read CR0 (INT 67, AX = DE07H)

Mit diesem Aufruf läßt sich der Inhalt des CR0-Registers im V86-Mode abfragen.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE07H Read CR0            º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º EBX= CR0                       º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

Nach einem erfolgreichen Aufruf (AH = 00H) findet sich in EBX der Wert des CR0-Registers.

VCPI Read Debug Register (INT 67, AX = DE08H)

Mit diesem Aufruf lassen sich im V86-Mode die Debug-Register des Prozessors abfragen.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE08H Read Debug Register º
º ES:DI Zeiger auf Puffer        º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

Im Register ES:DI ist ein Zeiger auf einen Puffer mit 8 Einträgen a 4 Byte zu reservieren. Nach einem erfolgreichen Aufruf ist AH = 0 und im Puffer stehen die Registerinhalte. Die Register sind in der Reihenfolge: DR0, DR1, ..., DR7 abgelegt. Die Register DR4 und DR5 sind dabei unbenutzt.

VCPI Set Debug Register (INT 67, AX = DE09H)

Mit diesem Aufruf lassen sich im V86-Mode die Debug-Register des Prozessors setzen.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE09H Set Debug Register  º
º ES:DI Zeiger auf Puffer        º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

Im Register ES:DI ist ein Zeiger auf einen Puffer mit 8 Einträgen a 4 Byte zu reservieren. Vor dem Aufruf ist der Puffer mit den gewünschten Werten für die Register zu initialisieren. Nach einem erfolgreichen Aufruf ist AH = 0 und die Register wurden geladen. Die Register sind in der Reihenfolge: DR0, DR1, ..., DR7 abzulegen. Die Register DR4 und DR5 sind dabei unbenutzt.

VCPI Get 8259 Interrupt Vector Mappings (INT 67, AX = DE0AH)

Mit diesem Aufruf läßt sich Abbildung der Interruptvektoren des 8259-Controllers ermitteln.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE0AH Get 8259 Mapping    º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º BX = 1. Vector Map             º
º CX = 2. Vector Map             º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

Der Server ermittelt die Interruptvektoren, die durch den 8259-Controller belegt werden. Nach dem erfolgreichen Aufruf findet sich in BX der Code für den IRQ0 und in CX der Code für den IRQ8. Falls kein Slave-Controller im System vorhanden ist, enthält CX undefinierte Werte.

VCPI Set 8259 Interrupt Vector Mappings (INT 67, AX = DE0BH)

Mit diesem Aufruf läßt sich Abbildung der Interruptvektoren des 8259-Controllers setzen.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE0AH Set 8259 Mapping    º
º BX = 1. Vector Map             º
º CX = 2. Vector Map             º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

Der Server speichert die in BX und CX übergebenen Interruptvektoren im 8259-Controller. Nach einem erfolgreichen Aufruf ist AX = 00H. Das Interruptsystem muß beim Aufruf gesperrt sein.

VCPI Switch to Protected Mode (INT 67, AX = DE0CH)

Dies ist der Aufruf um den Prozessor vom V86-Mode in den Protected Mode zu schalten.

```
Ö--------------------------------Ì
º         CALL: INT 67           º
º                                º
º AX = DE0CH Protected Mode      º
º ESI= Adresse Datenstruktur     º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

Vor dem Aufruf muß in ESI die lineare Adresse einer Datenstruktur übergeben werden. Diese Datenstruktur ist vom Client im 1-Mbyte-Bereich anzulegen. Die Datenstruktur besitzt folgenden Aufbau:

```
 Offset Bytes Bedeutung
  00H     4   neuer Wert für CR3
  04H     4   lineare Adresse im 1. Mbyte mit dem
              6 Byte Wert für das GDTR-Register
  08H     4   lineare Adresse im 1. Mbyte mit dem
              6 Byte Wert für das IDTR-Register
  0CH     2   Selector Wert für LDTR
  0EH     2   Selector Wert für TR
  10H     6   CS:EIP mit Einsprungadresse
```

Die Daten in der Tabelle sind entsprechend zu initialisieren. Vor dem Aufruf müssen die Interrupts gesperrt werden. Nach dem Aufruf bleibt das Interrupt-System gesperrt. Die Register: GDTR, IDTR, LDTR, TR wurden geladen und SS:ESP muß auf einen Stackbereich mit minmal 16 Byte zeigen. Die Register: EAX, ESI, DS, ES, FS, GS werden beim Aufruf zerstört.

Anschließend befindet sich die CPU im Protected Mode und führt das Programm ab der in CS:EIP angegebenen Stelle aus.

Aufruf der VCPI-Funktionen im Protected Mode

Im Protected Mode läßt sich der INT 67 nicht mehr zum Einsprung in die VCPI-Funktionen nutzen. Vielmehr muß der Server über einen CALL FAR-Aufruf aktiviert werden. Die Adresse des Einsprungpunktes läßt sich im V86-Mode mit der Funktion DE02H ermitteln.

Für die Aufrufe im Protected Mode gelten die gleichen Registerbelegungen wie im V86-Mode:

```
Ö--------------------------------Ì
º         CALL FAR xxxx          º
º                                º
º AX = DEXXH Funktionscode       º
º ....                           º
û--------------------------------À
º         Return:                º
º AH = 0 ok                      º
º AH <> 0 Fehler                 º
Û--------------------------------ì
```

Im Protected Mode sind folgende Funktionen des VCPI erreichbar:

```
AX = DE03H Get Number of Free 4K Pages
AX = DE04H Allocate a 4K Page
AX = DE05H Free a 4K Page
```

d.h. für XX sind die Codes 03H, 04H und 05H einzusetzen.

Switch from Protected Mode to V86-Mode

Mit diesem Funktionsaufruf läßt sich vom Protected Mode zum V86-Mode zurückschalten. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
°         CALL FAR xxxx          °
°                                °
° AX = DE0CH Switch to V86-Mode  °
° DS = Segment Selector          °
û--------------------------------À
°         Return:                °
° ----                           °
Û--------------------------------ì
```

Vor dem Aufruf muß der Stack (SS:ESP) in den linearen 1-Mbyte-Adressbereich gelegt werden. In DS ist ein Segment Selector zu übergeben, welcher den kompletten per Funktion DE01H ermittelten linearen Adressraum beinhaltet.

Beim Aufruf schaltet der Server die CPU wieder in den V86-Mode. Das Interruptsystem muß dabei gesperrt sein. Nach dem Aufruf sind GDTR, IDTR, LDTR und TR mit den Werten vom Server geladen. Die Kontrolle geht an die spezifizierte CALL FAR-Routine über. SS:ESP und die anderen Register werden mit den Werten vom Stack initialisiert. Dieser muß vor dem Aufruf folgende Daten enthalten:

```
Stack (jeweils DWORD Einträge)
28H GS
24H FS
20H DS
1CH ES
18H SS
14H ESP
10H reserviert
0CH CS
08H EIP
00H Return Adresse 8 Byte
```

Die ersten 8 Byte enthalten die 8-Byte-Rücksprungadresse des CALL FAR-Aufrufes des VCPI-Servers. Wichtig ist, daß der Aufruf der Funktion aus dem 1-Mbyte-Adressraum erfolgt, da ja nur dieser im V86-Mode erreicht wird. Nach dem Aufruf ist der Inhalt von EAX zerstört. Alle anderen Register bleiben unverändert erhalten.

12.7Das DOS-Protected-Mode-Interface (DPMI)

Der VCPI-Standard war zwar sehr erfolgreich, konnte aber nicht alle Aspekte des Zugriffs auf den Extended-Memory-Bereich abdecken. Insbesondere die Koexistenz verschiedener Speichermanager ist in VCPI nicht möglich. Weiterhin erlaubt VCPI einem Clienten auf der höchsten Prioritätsebene (Ring 0) zu laufen. Damit ist eine Virtualisierung des Clienten durch den VCPI-Server nicht mehr möglich. Microsoft entwickelte deshalb frühzeitig den Prototypen eines Speichermanagers für WINDOWS 3.0. Anfang 1990 wurde dann die DPMI-Spezifikation unter Mitarbeit der Firmen Microsoft und Intel in der Version 0.9

herausgegeben. Ab März 1991 existiert nun die Version 1.0. Nachfolgend werden beide Versionen beschrieben.

Allgemeine Anmerkungen

Das DOS Protected Mode Interface (DPMI) wurde definiert, damit DOS Programme den Extended-Memory-Bereich im PC adressieren können. Der Standard benutzt den INT 31 zur Kommunikation zwischen Client und DPMI-Server. Mit den Funktionen lassen sich Speicherbereiche verwalten, sowie Descriptoren verändern. Die wichtigste Eigenschaft ist jedoch, daß unter DPMI mehrere Protected-Mode-Programme in verschiedenen virtuellen Maschinen ausführbar sind. Wichtig ist allerdings, daß die Funktionen nur im Protected Mode benutzbar sind, im Real Mode stehen sie nicht zur Verfügung.

Generell ist festzuhalten, daß die INT 31-Aufrufe sowohl den Inhalt des Flagregisters, als auch den Inhalt von AX verändern. Die restlichen Register bleiben erhalten. Ungültige Funktionsaufruf setzen vor der Rückkehr das Carry-Flag. Alle DPMI-Funktionen sind reentrant programmiert.

Protected-Mode-Programme können Hard- und Software-Interrupts abfangen. Die Einstellung des Interrupt-Controllers wird so umprogrammiert, daß die Interrupt auch im Protected Mode zur Verfügung stehen. Viele der Software-Interrupts stehen aber nur im Real Mode zur Verfügung. Die DPMI-Server richtet jedoch folgende Interrupts auch im Protected Mode ein:

```
Interrupt
 1CH  BIOS Timerfolge Interrupt
 23H  DOS Control-C Interupt
 24H  DOS Critical Error Interrupt
```

Viele Implementierungen des DPMI unterstützen virtuelle Speichertechniken. Beim Betrieb von DOS-Programmen müssen daher die benutzen Speicherbereich gegebenen falls für eine Auslagerung gesperrt werden.

Jeder DPMI-Task benutzt 4 verschiedene Stacks zur Laufzeit:

- Protected Mode Application Stack (Ring 0)
- Locked Protected Mode Stack
- Real Mode Stack
- DPMI Host (Ring 0) Stack.

Der Protected-Mode-Stack wird vom Client bei der Umschaltung zwischen Real- und Protected Mode benutzt. Der Locked Protected Mode Stack wird vom DPMI-Server zur Simulation der Hardware-Interrupts benutzt. Der DPMI-Server richtet weiterhin einen 200H Byte großen Real-Mode-Stack für Real-Mode-Interrupts ein.

Die Real-Mode-DPMI-Funktionen.

Alle DPMI-Applikationen beginnen im Real Mode und müssen anschließend in den Protected Mode umschalten. Erst im Protected Mode stehen die INT 31-Aufruf mit den DPMI-Funktionen zur Verfügung. Allerdings gibt es einige Funktionen, die im Real Mode aufrufbar sind.

Release Virtual Maschine Time Slice (INT 2F, AX = 1680H, V 1.0)

Dieser bereits beschrieben Interrupt ist erst in der DPMI-Version 1.0 implementiert. Er erlaubt es, einem laufenden Programm im V86-Mode die Kontrolle an das Betriebssystem zurückzugeben. Hierzu ist das Register AX vor dem Aufruf auf den Wert 1680H zu setzen. Wird der Aufruf nicht unterstützt, dann kehrt der Aufruf mit AL = 80H zurück. Andernfalls geht die Kontrolle an das Betriebssystem zurück. Nach Ausführung der restlichen Task erhält die Applikation wieder Rechenzeit und AL ist auf den Wert 00H gesetzt. Weitere Informationen finden sich im Kapitel über den INT 2F.

Get CPU Mode (INT 2F, AX = 1686H, V 0.9)

Mit diesem Aufruf kann ein Programm im Real Mode feststellen, in welchem Mode die CPU arbeitet.

```
Ö--------------------------------Ì
º         CALL INT 2F            º
º                                º
º AX = 1686H Detect CPU Mode     º
û--------------------------------À
º         Return:                º
º AX = 00H Protected Mode        º
º   <> 00H V86 Mode               º
Û--------------------------------ì
```

Das Ergebnis der Prüfung wird in AL zurückgegeben. Ist der Wert AL <> 00H, arbeitet die CPU im virtuellen V86-Mode, oder im Real Mode. Der Aufruf ist jedoch nur auf 80386-Rechnern (und aufwärtskompatiblen) gültig, falls das DPMI-Interface installiert ist. Deshalb ist vorher zu prüfen, ob der DPMI-Server geladen ist.

Detect DPMI-Server (INT 2F, AX = 1687H, V 0.9)

Mit diesem Aufruf kann ein Programm prüfen, ob der DPMI-Server vorhanden ist.

```
Ö--------------------------------Ì
º         CALL INT 2F            º
º                                º
º AX = 1687H Detect DPMI         º
û--------------------------------À
º         Return:                º
º AX = 00H ok                    º
º BX = Flags                     º
º CL = Prozessor Typ             º
º DX = Versionsnummer            º
º SI = Memory Size DPMI Host     º
º ES:DI Einsprungpunkt DPMI      º
º AX = 16H DPMI nicht install.   º
Û--------------------------------ì
```

Falls der DPMI-Server vorhanden ist, muß der Inhalt von AX = 00H sein. Dann findet sich in BX ein Flag mit folgender Kodierung:

```
Bit 0 = 1 32-Bit-Programme werden unterstützt
```

Das Register CL enthält die Kodierung für den Prozessortyp:

```
CL = Processor-Typ
 02H = 80286
 03H = 80386
 04H = 80486
```

In DH findet sich die Hauptversion und in DL die Unterversion des DPMI-Servers. Das Register SI enthält eine Information, wieviel Speicher (in Paragraphen) für den Datenbereich vom DPMI-Server belegt wird. In ES:DI steht die Adresse der Prozedur zur Umschaltung in den Protected Mode. Bei Werten von AX <> 0 wird der DPMI-Standard nicht unterstützt.

Protected Mode Umschaltung

Nachdem per INT 2F die Anwesenheit des DPMI-Servers festgestellt wurde, läßt sich in den Protected Mode umschalten. Hierzu muß der Client einen CALL FAR-Aufruf zu der in ES:DI zurückgegebenen Adresse ausführen. Hierbei gilt folgende Aufrufschnittstelle:

```
Ö-----------------------------------------------Ì
º  CALL FAR                                     º
º                                               º
º AX = Flags                                    º
º  Bit 0 = 1 32-Bit-Applikation application     º
º ES =  Real Mode Segment der DPMI-Host-Daten   º
º       (Größe wird in SI zurückgegeben)        º
º                                               º
º  RETURN                                       º
º CY = 0 ok.                                    º
º CS, SS, ES, DS = 16-Bit-Selektor              º
Û-----------------------------------------------ì
```

Der Inhalt von ES wird ignoriert, falls die Größe des Datenbereiches 0 ist. Nach dem Aufruf zeigt ein gelöschtes Carry-Flag an, daß die CPU im Protected Mode arbeitet. FS und GS werden auf 80386-Prozessoren zu 0 gesetzt. Ist die Applikation ein 32-Bit-Programm, wird ESP = 0 zurückgegeben.

Bei gesetztem Carry-Flag wird das Programm weiter im Real Mode ausgeführt.

Sobald das Programm im Protected Mode läuft, können die DPMI-Funktionen per INT 31H aufgerufen werden. Um das Programm zu beenden, ist ein INT 21 mit AH = 4C auszuführen. Programme, die als 32-Bit-Applikation laufen sollen, werden nach der Umschaltung als 16-Bit-Programm ausgeführt. Die Programme können anschließend die erweiterten Segmentgrößen (32-Bit) nutzen. Das A20 Adressbit wird nicht automatisch im Protected Mode freigegeben, d.h. der Bereich oberhalb 1 Mbyte steht nicht zur Verfügung. Die Freigabe kann per XMS-Aufruf erfolgen. Beachten Sie aber, daß im Protected Mode die XMS-Funktionsaufrufe nicht mehr verfügbar sind. Dies ist jedoch nur von Bedeutung, falls Programme den HMA-Bereich benutzen möchten.

Get Vendor API Entry Point (INT 2F, AX = 168AH, V 1.0)

Dieser Aufruf ist erst ab der Version 1.0 dokumentiert (arbeitet aber ab Version 0.9) und erlaubt die Abfrage eines Einsprungpunktes für herstellspezifische Funktionen.

```
Ö--------------------------------Ì
º         CALL INT 2F            º
º                                 º
º AX = 168AH Vendor API Adresse  º
º DS:EDI Stringadresse           º
û--------------------------------À
º         Return:                º
º AX = 00H ok                    º
º ES:EDI Einsprungpunkt API      º
º AX <> 0 Fehler                 º
Û--------------------------------ì
```

In DS:ESI ist die Adresse (Selector:Offset) eines ASCIIZ-Strings mit der Hersteller-Identifikation zu übergeben.

Falls die Funktion nicht unterstützt wird, findet sich nach dem Aufruf in AL der Wert 8AH. Andernfalls steht in ES:EDI die Adresse für den API-Einsprungpunkt. Hier kann der Hersteller des DPMI-Servers eigene Funktionserweiterungen zur Verfügung stellen. Die Funktion ist nur im Protected Mode verfügbar.

12.8 Die Protected Mode DPMI-Funktionen

Die Hauptfunktionen des DPMI lassen sich nur im Protected Mode über den INT 31 aufrufen.

Allocate LDT Descriptors (INT 31, AX = 0000H, V 0.9)

Diese Funktion wird benutzt, um einen oder mehrere Descriptoren aus der lokalen *Task Descriptor Tabelle* (LDT) zu allocieren. Die allocierten Descriptoren müssen durch die Applikation über andere Funktionsaufrufe initialisiert werden.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                 º
º AX = 0000H Allocate LDT        º
º CX = Descriptor Nummern        º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º AX = Base Selektor             º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Beim Aufruf ist AX = 0000H zu übergeben. In CX steht die Nummer der zu allocierenden Descriptoren.

Bei gesetztem Carry-Flag nach dem Aufruf trat ein Fehler auf. In AX findet sich der Fehlercode:

```
AX = 8011H  Descriptor unavailable
```

Ein gelöschtes Carry-Flag zeigt einen erfolgreichen Aufruf an. In AX steht dann der Base Selector. Wird mehr als ein Descriptor allociert, gibt die Funktion den Base Selector zurück, der als erster in einem Feld mit Selektoren steht. Die Selektoren für weitere Descriptoren lassen sich aus dem per INT 31H, Funktion 0003H zurückgegebenen Wert bestimmen.

Der Descriptor wird auf die aktuellen Daten gesetzt, wobei *Base* und *Limit* den Wert 0 erhalten. Der Privileg Level des Descriptors entspricht dem des Codesegmentes der Applikation. Wird ein Selektor modifiziert, muß DPL auf den gleichen Privileg Ring wie das Codesegment des Programmes gesetzt werden. Zur Bestimmung des Privileg Levels eines Descriptors ist die LAR-Anweisung zu verwenden.

Free LDT Descriptor (INT 31, AX = 0001H, V 0.9)

Diese Funktion gibt einen durch die Funktion *Allocate LDT Descriptors* allocierten Descriptor wieder frei.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0001H Free LDT Descriptor º
º BX = Selector                  º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX ist der Selector für den freizugebenden Descriptor zu definieren. Ein Feld mit Descriptoren kann nur durch Einzelaufrufe aller Selektoren freigegeben werden. Descriptoren, die nicht per Funktion 0000H, 000AH oder 000DH allociert wurden, dürfen nicht durch obige Funktion freigegeben werden. Es ist aber möglich, die bei der Umschaltung in den Protected Mode automatisch allocierten Descriptoren für CS, DS und SS freizugeben.

Bei fehlerhaften Aufruf, ist das Carry-Flag gesetzt und AX enthält den Fehlercode:

```
AX = 8022H  Invalid Selector
```

Dann ist der angegebene Selektor ungültig.

Segment to Descriptor (INT 31, AX = 0002H, V 0.9)

Diese Funktion erlaubt die Konvertierung von Real-Mode-Segmenten in Descriptoren, die im Protected Mode adressierbar sind.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0002H Segment to Descript.º
º BX = Segmentadresse            º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º AX = Selector                  º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX ist die Segmentadresse für den Real Mode anzugeben. Bei fehlerhaftem Aufruf ist das Carry-Flag gesetzt und AX enthält einen Fehlercode:

```
AX = 8011H Descriptor unavailable
```

Bei gelöschtem Carry-Flag steht in AX der zugehörige Selector für das Real-Mode-Segment.

Die Descriptoren sind allerdings auf 64-Kbyte-Größe begrenzt. Mit Hilfe der Funktion kann der Client sehr einfach auf Real-Mode-Speicherbereiche (BIOS, VIDEO-RAM, etc.) aus dem Protected Mode zugreifen.

Descriptoren, die durch diesen Aufruf erzeugt werden, dürfen niemals verändert oder freigegeben werden.

Get Next Selector Increment Value (INT 31, AX = 0003H, V 0.9)

Einige Funktionen (z.B. LDT) erlauben es, mehrere Descriptoren pro Aufruf zu allocieren. Mit der Funktion 0003H läßt sich der Wert abfragen, der zum Basis Selector zu addieren ist, um den folgenden Selector zu erhalten.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0003H Get Select. Inc Val º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º AX = Selector Inc Value        º
Û--------------------------------ì
```

Nach dem Aufruf enthält AX (bei gelöschtem Carry-Flag) den Increment Wert. Der Wert zur Bestimmung des nächsten Selectors ist zu 2 ** x zu bestimmen.

Die Funktionen 0004H und 0005H sind reserviert und dürfen nicht benutzt werden.

Get Segment Base Address (INT 31, AX = 0006H, V 0.9)

Die Funktion gibt die 32-Bit (lineare) Basisadresse des spezifizierten Segmentes zurück.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0006H Get Segm. Base Adr. º
º BX = Selector                  º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX:DX 32-Bit-Adresse           º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX ist beim Aufruf der Selector für das betreffende Segment anzugeben. Bei gesetztem Carry-Flag enthält AX den Fehlercode:

```
AX = 8022H Invalid Selector
```

Bei Carry-Flag = 0 findet sich in CX:DX die lineare 32-Bit-Basisadresse des betreffenden Segmentes.

Die Funktion liefert keine gültigen Ergebenisse, falls der Selector in BX ungültig ist.

Set Segment Base Address (INT 31, AX = 0007H, V 0.9)

Diese Funktion setzt die lineare 32-Bit-Basisadresse für den spezifizierten Selector im LDT-Descriptor.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0007H Set Seg. Base Adr.  º
º BX = Selector                  º
º CX:DX 32-Bit-Adresse           º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX ist der Selector und in CX:DX die 32-Bit-Basisadresse zu übergeben. Bei gesetztem Carry-Flag ist ein Fehler aufgetreten. AX enhält folgende Fehlercodes:

```
AX = 8022H Invalid Selector
     8025H Invalid linear Address
```

Der Aufruf geht schief, falls der Wert in BX ungültig ist. Eine Anwendung sollte nur die per Funktion 0000H angeforderten Descriptoren verändern. Die höherwertigen 8 Bit (in CH) werden bei einer 16-Bit-Implementierung ignoriert.

Set Segment Limit (INT 31, AX = 0008H, V 0.9)

Diese Funktion setzt die Segmentgrenze für das spezifizierte Segement.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0008H Set Seg. Base Adr.  º
º BX = Selector                  º
º CX:DX 32-Bit-Segment-Limit     º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Bei erfolgreichem Aufruf ist das Carry-Flag nach der Rückkehr gelöscht und die Segmentgrenze wurde eingestellt. Bei einem Fehler während des Aufrufes wird das Carry-Flag gesetzt und in AX findet sich anschließend ein Fehlercode:

```
AX = 8021H Invalid Value (CX <> 0 auf 80286, oder
           Limit größer 1 Mbyte)
     8022H Invalid Selector
     8025H Invlid linear Address
```

Bei ungültigen Werten in BX geht der Funktionsaufruf schief. Bei 16-Bit-Implementationen darf die Segmentgrenze nicht über 64 Kbyte (0FFFFH) reichen, so daß CX = 0 zu setzen ist. Im 32-Bit-Mode müssen Segmentgrößen über 1 Mbyte auf Seitengrenzen (4 Kbyte) ausgerichtet werden. Dabei sind die unteren 12 Bit immer zu setzen. Mit der Funktion LSL (load segment limit) läßt sich die Segmentgröße ermitteln.

Set Descriptor Access Rights (INT 31, AX = 0009H, V 0.9)

Diese Funktion erlaubt es, die Zugriffsrechte und Typeninformationen eines Descriptors aus einem Protected-Mode-Programm heraus zu verändern.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0009H Set Access Rights   º
º BX = Selector                  º
º CL = Access Right              º
º CH = Typinformationen          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Der Funktionsaufruf geht schief, falls in BX ein ungültiger Wert steht. Es dürfen nur Descriptoren verändert werden, die mit LDT allociert wurden.

Bei einem fehlerhaften Aufruf wird das Carry-Flag gesetzt und in AX findet sich anschließend ein Fehlercode:

```
AX = 8021H  Invalid Value
     8022H  Invalid Selector
     8025H  Invalid linear Address
```

Um die Zugriffsrechte zu überprüfen, läßt sich die LAR-Anweisung (Load Access Rights) verwenden. Das in CL übergebene Byte mit den Zugriffsrechten besitzt folgende Kodierung:

```
Ö---Ú------Ú---Ú---Ú---Ú---Ú---Ì
º P º DPL  º 1 ºC/DºE/CºW/Rº A º
Û-Ú-Ù--Ú---Ù-Ú-Ù-Ú-Ù-Ú-Ù-Ú-Ù-Ú-ì
  º    º     º   º   º   º   Û- 0 Not Accessed
  º    º     º   º   º   º      1 Accessed
  º    º     º   º   º   Û-- Data: 0 Read, 1 R/W
  º    º     º   º   º       Code: 1 (readable)
  º    º     º   º   Û-- Data: 0 Exp-up, 1 Exp-dn
  º    º     º   º       Code: 0 (non-conform)
  º    º     º   Û-- 0 Data, 1 Code
  º    º     Û-- 1
  º    Û-- = caller's CPL
  Û- 0 Absent, 1 Present
```

Parameter, die obiger Kodierung nicht entsprechen sind ungültig. Bei 16-Bit-DPMI-Implementierungen wird der Wert in CH (Extended Access Rights) ignoriert. In 32-Bit-DPMI-Implementierungen wird der Wert in CH folgendermaßen interpretiert:

```
Ö---Ú---Ú---Ú---Ú---------------Ì
º G ºB/Dº 0 ºAvlº    Reserved   º
Û-Ú-Ù-Ú-Ù-Ú-Ù-Ú-Ù-------Ú-------ì
  º   º   º   º         Û---Ignorieren
  º   º   º   Û--  0 oder 1
  º   º   Û-- 0
  º   Û-- 0 Default 16-Bit, 1 Default 32-Bit
  Û- 0 Byte Auflösung, 1 Page Auflösung
```

Parameter, die nicht obiger Kodierung entsprechen, führen zu einer Fehlermeldung beim Aufruf.

Create Code Segment Alias Descriptor (INT 31, AX = 000AH, V 0.9)

Diese Funktion erzeugt einen Datendescriptor mit der gleichen Basis und Limit wie der angegebene Code-Segment-Descriptor.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 000AH Create CS Descript. º
º BX = CS Selector               º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º AX = neuer Daten Selector      º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Der Aufruf führt zu einem Fehler, falls der Wert in BX falsch ist. Der Descriptor ist mit der Funktion LDT freizugeben.

Get Descriptor (INT 31, AX = 000BH, V 0.9)

Diese Funktion kopiert einen LDT-Descriptor in einen Puffer.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 000BH Get Descriptor      º
º BX = Selector                  º
º ES:EDI Zeiger auf Puffer       º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Bei Aufruf ist in ES:EDI ein Zeiger auf einen 8-Byte-Puffer zu übergeben. Die Funktion legt den Descriptor in dem Puffer ab. Bei gesetztem Carry-Flag liegt ein Fehler vor und in AX steht ein Fehlercode:

```
AX = 8022H Invalid Selector
```

Der in BX übergebene Selector muß gültig sein. In 32-Bit-Programmen wird ES:EDI benutzt, bei 16-Bit-Programmen wird nur ES:DI als Zeiger verwendet.

Set Descriptor (INT 31, AX = 000CH, V 0.9)

Die Funktion kopiert den Inhalt eines 8-Byte-Puffers in den LDT Eintrag des spezifizierten Descriptors.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 000CH Set Descriptor      º
º BX = Selector                  º
º ES:EDI Zeiger auf Puffer       º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In ES:DI ist der Selector:Offset auf einen 8-Byte-Puffer mit dem Descriptor zu übergeben. BX enthält den Selector. Bei einem gesetzten Carry-Flag liegt ein Fehler beim Aufruf vor. In AX stehen folgende Fehlercodes:

```
AX = 8021H Invalid Value
     8022H Invalid Selector
     8022H Invalid linear Address
```

Der Selector in BX muß gültig sein. Es dürfen nur Descriptoren verändert werden, die per LDT allociert wurden. Das *Descriptor-Acces-Right-Byte* (Byte 5) besitzt die gleiche Kodierung und Bedeutung wie der bei der Funktion 0009H in CL übergebene Typ. Das gleiche gilt für Byte 6, welches die erweiterten Zugriffrechte definiert (32-Bit-Mode), wobei die unteren 4 Bit benutzt sind (Descriptor Limit).

Allocate Specific LDT Descriptor (INT 31, AX = 000DH, V 0.9)

Diese Funktion allociert einen spezifizierten LDT-Descriptor.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 000DH Allocate LDT Descr. º
º BX = Selector                  º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Beim Aufruf ist in BX der Selector des Descriptors zu übergeben. Ist nach dem Aufruf das Carry-Flag gesetzt, enthält AX den Fehlercode:

```
AX = 8011H  Descriptor unavailable
     8022H  Invalid Selector
```

Bei gelöschtem Carry-Flag wurde die Funktion erfolgreich ausgeführt und der LDT-Descriptor allociert. Bei fehlerhaften Werten in BX geht der Aufruf schief. Zur Freigabe des Descriptors ist die Funktion 0001H zu benutzten. Die ersten 16 Descriptoren sind reserviert und dürfen nicht benutzt werden.

Get Multiple Descriptors (INT 31, AX = 000EH, V 1.0)

Diese Funktion kopiert einen oder mehrere lokale Descriptoren aus der LDT in einen Puffer.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 000EH Get Descriptors     º
º CX = Zahl der Descriptoren     º
º ES:EDI Puffer                  º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX = Zahl der Descriptoren     º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Beim Aufruf ist in CX die Zahl der zu kopierenden Einträge der LDT anzugeben. In ES:EDI wird die Adresse des Puffers mit folgendem Aufbau erwartet:

```
Offset Bytes Bedeutung
 00H     2   Selector 1
 02H     8   Descriptor 1
 00H     2   Selector 2
 02H     8   Descriptor 2
  ..
```

Das Anwendungsprogramm (Client) muß den Puffer einrichten und die Selectoren vor dem Aufruf eintragen. Die Funktion kopiert dann die Descriptoren aus der lokalen Descriptor-Tabelle (LDT) in den Puffer. Die Größe der Tabelle wird durch die Zahl der zu kopierenden Descriptoren bestimmt. Diese Zahl ist in CX zu übergeben. Nach einem erfolgreichen Aufruf findet sich in CX die Zahl der wirklich kopierten Descriptoren.

Der Status des Aufrufs wird im Carry-Flag zurückgegeben. Ist nach dem Aufruf das Carry-Flag gesetzt, enthält AX den Fehlercode:

```
AX = 8021H  Invalid Value
     8022H  Invalid Selector
     8025H  Invalid linear Address
```

Bei gelöschtem Carry-Flag wurde die Funktion erfolgreich ausgeführt und der Inhalt der LDT-Descriptor-Tabelle in den Puffer kopiert. Bei 32-Bit-Programmen muß die Pufferadresse in ES:EDI übergeben werden, sonst reicht das Registerpaar ES:DI.

Set Multiple Descriptors (INT 31, AX = 000FH, V 1.0)

Diese Funktion kopiert einen oder mehrere Descriptoren aus dem Puffer in die LDT.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 000FH Set Descriptors     º
º CX = Zahl der Descriptoren     º
º ES:EDI Puffer                  º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX = Zahl der Descriptoren     º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Beim Aufruf ist in CX die Zahl der zu kopierenden Einträge anzugeben. In ES:EDI wird die Adresse des Puffers mit folgendem Aufbau erwartet:

```
Offset Bytes Bedeutung
 00H     2   Selector 1
 02H     8   Descriptor 1
 00H     2   Selector 2
 02H     8   Descriptor 2
  ..
```

Das Anwendungsprogramm (Client) muß den Puffer einrichten und die Einträge vor dem Aufruf initialisieren. Die Funktion kopiert dann die Descriptoren aus dem Puffer in die lokale Descriptor-Tabelle (LDT). Die Zahl der zu kopierenden Descriptoren ist in CX zu übergeben. Nach einem erfolgreichen Aufruf findet sich in CX die Zahl der wirklich kopierten Descriptoren.

Der Status des Aufrufs wird im Carry-Flag zurückgegeben. Ist nach dem Aufruf das Carry-Flag gesetzt, enthält AX den Fehlercode:

```
AX = 8021H Invalid Value
     8022H Invalid Selector
```

Bei gelöschtem Carry-Flag wurde die Funktion erfolgreich ausgeführt und der Inhalt der LDT-Descriptor-Tabelle in den Puffer kopiert. Bei 32-Bit-Programmen muß die Pufferadresse in ES:EDI übergeben werden, sonst reicht das Registerpaar ES:DI.

Die Access-Right-Bits besitzen die gleichen Formate wie bei der Funktion 0009H.

Allocate DOS Memory Block (INT 31, AX = 0100H, V 0.9)

Diese Funktion reserviert (allociert) einen Block aus dem freien DOS-Speicherpool.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0100H Alloc DOS Memory    º
º BX = Zahl der Paragraphen      º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º AX = Real Mode Seg-Adresse     º
º DX = Selector des Blocks       º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Die Funktion reserviert die Zahl der gewünschten Paragraphen (16 Byte) im freien DOS-Speicher. Bei erfolgreichem Aufruf ist das Carry-Flag gelöscht und in AX steht die Segmentadresse des allocierten Speicherblocks. DX enthält den Selector des allocierten Blocks.

Bei einem Fehler ist nach dem Aufruf das Carry-Flag gesetzt und AX enthält den Fehlercode:

```
AX = 0007H  MCB damaged
     0008H  Insufficient Memory
     8011H  Descriptor unavailable
```

Ist der angeforderte Speicherblock größer als der freie Speicher, enthält BX die Größe des größten freien Speicherblocks im DOS-Speicherbereich.

Falls der angeforderte Block größer als 64 Kbyte ist, werden im 16-Bit-Mode mehrere Descriptoren allociert. Die reservierten Blöcke müssen mit der Funktion 0101H wieder freigegeben werden.

Free DOS Memory Block (INT 31, AX = 0101H, V 0.9)

Diese Funktion gibt einen durch die Funktion 1000H allocierten DOS-Block wieder frei.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0101H Free DOS Memory     º
º DX = Selector des Blocks       º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Beim Aufruf ist in DX der Selector des freizugebenden Blockes zu definieren. Bei einem fehlerhaften Aufruf wird das Carry-Flag gesetzt und AX enthält einen Fehlercode:

```
AX = 0007H  MCB damaged
     0009H  Incorrect Memory Segment
     8022H  Invalid Selector
```

Beim Aufruf der Funktion werden alle allocierten Descriptoren des Speicherblocks freigegeben.

Resize DOS Memory Block (INT 31, AX = 0102, V 0.9)

Diese Funktion variiert die Größe eines durch die Funktion 1000H allocierten DOS-Blockes.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0102H Resize DOS Block    º
º BX = Blockgröße                º
º DX = Selector des Blocks       º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
º BX = Max. Blockgröße           º
Û--------------------------------ì
```

Beim Aufruf ist in DX der Selector des Blockes zu definieren. In BX wird die neue Größe des Blockes erwartet. Bei einem fehlerhaften Aufruf wird das Carry-Flag gesetzt und AX enthält einen Fehlercode:

```
AX = 0007H  MCB damaged
     0008H  Insufficient Memory
     0009H  Incorrect Memory Segment
     8011H  Descriptor unavailable
     8022H  Invalid Selector
```

In BX findet sich dann die Größe, auf die der Block maximal erweitert werden kann.

Get Real Mode Interrupt Vector (INT 31, AX = 0200H, V 0.9)

Die folgenden Funktionen erlauben den Zugriff auf Real-Mode-Interrupts aus dem Protected Mode.

Mit der Funktion 0200H läßt sich der Interruptvektor des aktuellen Tasks für die angegebene Interruptnummer ermitteln.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0200H Get INT Vector      º
º BL = Interrupt Nummer          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX:DX Interrupt Vektor         º
Û--------------------------------ì
```

Bei Aufruf ist in BL die Nummer des gewünschten Interrupts zu übergeben. Die Funktion ermittelt dann die Real-Mode-Adresse des betreffenden Interruptvektors und gibt diese in CX:DX zurück. Der Wert in diesem Registerpaar ist jedoch nur bei gelöschtem Carry-Flag gültig.

Set Real Mode Interrupt Vector (INT 31, AX = 0201H, V 0.9)

Mit der Funktion 0201H läßt sich der Interruptvektor des aktuellen Tasks für die angegebene Interruptnummer setzen.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0201H Set INT Vector      º
º BL = Interrupt Nummer          º
º CX:DX Interrupt Vektor         º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
Û--------------------------------ì
```

Bei Aufruf ist in BL die Nummer des gewünschten Interrupts zu übergeben. In CX:DX muß der Interruptvektor stehen. Die Funktion setzt dann die angegebene Real-Mode-Adresse in der Interrupttabelle ein. Ein Aufruf ist erfolgreich falls anschließend das Carry-Flag gelöscht ist. Falls Hardware-Interrupts abgefangen werden, muß das Segment des Interrupt-Handlers gesperrt (locked) werden. In CX:DX müssen Real-Mode-Adressen übergeben werden.

Get Processor Exception Handler Vector (INT 31, AX = 0202H, V 0.9)

Die Funktion gibt den Inhalt von CS:EIP für den aktuellen Protected Mode Exeption Handler zurück.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0202H Get Exeption Adr.   º
º BL = Exeption Nummer           º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX:EDX Selector:Offset Handler º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BL ist die Exeption/Fault-Nummer (00H-1FH) zu übergeben. Bei einem erfolgreichen Aufruf ist anschließend das Carry-Flag gelöscht und in CX:EDX findet sich die Adresse

(Selector:Offset) des Exeption Handlers. Bei einem Fehler wird das Carry-Flag gesetzt und AX enthält den Fehlercode:

```
AX = 8021H Invalid Value (BL nicht 00H - 1FH)
```

Bei 32-Bit-Programme ist das erweiterte Register EDX zu nutzen. In CX wird ein gültiger Protected-Mode-Selector zurückgegeben.

Set Processor Exception Handler Vector (INT 31, AX = 0203H, V 0.9)

Die Funktion setzt die Adresse für den aktuellen *Protected-Mode-Exeption-Handler*.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0202H Get Exeption Adr.   º
º BL = Exeption Nummer           º
º CX:EDX Selector:Offset Handler º
Û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BL ist die Exeption/Fault-Nummer (00H-1FH) zu übergeben. In CX:EDX ist der Selector und der Offset des neuen Exeption Handlers zu übergeben.

Bei einem erfolgreichen Aufruf ist anschließend das Carry-Flag gelöscht. Bei einem Fehler wird das Carry-Flag gesetzt und AX enthält den Fehlercode:

```
AX = 8021H  Invalid Value (BL nicht 00H - 1FH)
     8022H  Invalid Selector
```

Bei 32-Bit-Programme ist das erweiterte Register EDX zu nutzen. In CX wird ein gültiger Protected-Mode-Selector zurückgegeben. Ab DPMI 1.0 sollten die Funktionen 0212 und 0213 für diesen Zweck benutzt werden.

In CX muß ein gültiger Selector übergeben werden.

Der Exeption Handler sollte mit einem RETURN-FAR-Befehl beendet werden. Die Original Register SS:(E)SP, CS:(E)IP und die Flags werden gesichert.

Der Exeption Handler wird mit gesperrtem Stack und gesperrtem Interruptsystem aufgerufen. ESP, CS, und EIP werden auf den Stack gesichert.

Get Protected Mode Interrupt Vector (INT 31, AX = 0204H, V 0.9)

Mit der Funktion 0204H läßt sich der Interruptvektor des aktuellen Tasks für die angegebene Interruptnummer im Protected Mode ermitteln.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0204H Get INT Vector      º
º BL = Interrupt Nummer          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX:EDX Interrupt Vektor        º
Û--------------------------------ì
```

Bei Aufruf ist in BL die Nummer des gewünschten Interrupts zu übergeben. Die Funktion ermittelt dann die Protected-Mode-Adresse des betreffenden Interruptvektors und gibt diese in CX:DX zurück. Der Wert in diesem Registerpaar ist jedoch nur bei gelöschtem Carry-Flag gültig. In CX steht dabei ein Selector.

Set Protected Mode Interrupt Vector (INT 31, AX = 0205H, V 0.9)

Mit der Funktion 0205H läßt sich der Interruptvektor des aktuellen Tasks für die angegebene Interruptnummer im Protected Mode setzen.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0204H Get INT Vector      º
º BL = Interrupt Nummer          º
º CX:EDX Interrupt Vektor        º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Bei Aufruf ist in BL die Nummer des gewünschten Interrupts zu übergeben. Die Funktion erwartet dann die Protected-Mode-Adresse des betreffenden Interruptvektors in CX:DX. In CX steht dabei ein Selector. Der Wert wurde jedoch nur bei gelöschtem Carry-Flag übernommen.

Bei gesetztem Carry-Flag findet sich in AX der Fehlercode:

```
AX = 8022H  Invalid Selector
```

Get Processor Exception Handler Vector (INT 31, AX = 0210H, V 1.0)

Die Funktion gibt den Inhalt von CS:EIP für die spezifizierte Protected-Mode-Exeption zurück.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0210H Get Exeption Adr.   º
º BL = Exeption Nummer           º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX:EDX Selector:Offset Handler º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BL ist die Exeption/Fault-Nummer (00H-1FH) zu übergeben. Bei einem erfolgreichen Aufruf ist anschließend das Carry-Flag gelöscht und in CX:EDX findet sich die Adresse

(Selector:Offset) des *Protected Mode Exeption Handlers*. Bei einem Fehler wird das Carry-Flag gesetzt und AX enthält den Fehlercode:

```
AX = 8021H  Invalid Value (BL nicht 00H - 1FH)
```

Bei 32-Bit-Programme ist das erweiterte Register EDX zu nutzen. In CX wird ein gültiger Protected Mode Selector zurückgegeben.

Get Processor Exception Handler Vector (INT 31, AX = 0211H, V 1.0)

Die Funktion gibt den Inhalt von CS:EIP für die spezifizierte Real-Mode-Exeption zurück.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0211H Get Exeption Adr.   º
º BL = Exeption Nummer           º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX:EDX Selector:Offset Handler º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BL ist die Exeption/Fault-Nummer (00H-1FH) zu übergeben. Bei einem erfolgreichen Aufruf ist anschließend das Carry-Flag gelöscht und in CX:EDX findet sich die Adresse (Selector:Offset) des Protected Mode Exeption Handlers für die spezifizierten Real-Mode-Exeptions. Bei einem Fehler wird das Carry-Flag gesetzt und AX enthält den Fehlercode:

```
AX = 8021H  Invalid Value (BL nicht 00H - 1FH)
```

Bei 32-Bit-Programme ist das erweiterte Register EDX zu nutzen. In CX wird ein gültiger Protected Mode Selector zurückgegeben. Der Handler erhält die Kontrolle im Protected Mode, auch wenn der Fehler im Real Mode aufgetreten ist.

Simulate Real Mode Interrupt (INT 31, AX = 0300H, V 0.9)

Diese Funktion simuliert einen Real-Mode-Interrupt im Protected Mode.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0300H Simulate Interrupt  º
º BL = Interrupt Nummer          º
º BH = Flags                     º
º CX = Zahl der Worte            º
º ES:EDI Puffer                  º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX:EDX Puffer                  º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BH ist ein Flagwert zu übergeben, wobei zur Zeit aber alle Werte = 0 zu setzen sind. In BL steht die Interrupt Nummer. Das Register CX enthält die Zahl der Worte, die vom Protected-Mode-Stack auf den Real-Mode-Stack zu kopieren sind. In ES:EDI ist die Adresse (Selector:Offset) auf einen Datenpuffer mit folgender Struktur zu übergeben:

```
Offset Len Register
 00H    4    EDI
 04H    4    ESI
 08H    4    EBP
 0CH    4    Reserviert
 10H    4    EBX
 14H    4    EDX
 18H    4    ECX
 1CH    4    EAX
 20H    2    Flags
 22H    2    ES
 24H    2    DS
 26H    2    FS
 28H    2    GS
 2AH    2    IP
 2CH    2    CS
 2EH    2    SP
 30H    2    SS
```

In diesem Puffer werden Daten für die Übergabe an den Real-Mode-Stack übergeben. Bei einem erfolgreichen Aufruf ist anschließend das Carry-Flag gelöscht und im Puffer finden sich die Daten des Real-Mode-Stacks.

Bei einem Fehler setzt die Funktion das Carry-Flag und in AX findet sich ein Fehlercode:

```
AX = 8012H  Linear Memory unavailable
AX = 8013H  Physical Memory unvailable
AX = 8014H  Backing Store unavailable
AX = 8021H  Invalid Value (CX)
```

Der Handler muß den Stack vor der Rückkehr in den ursprünglichen Zustand bringen. Die Werte in den Segment-Registers sind als Real-Mode-Segmente anzugeben (keine Selectoren).

Call Real Mode Procedure RET FAR (INT 31, AX = 0301H, V 0.9)

Diese Funktion ruft eine Real-Mode-Funktion mit einem *FAR RETURN Frame* auf.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0301H Call Procedure      º
º BH = Flags                     º
º CX = Zahl der Worte            º
º ES:EDI Puffer                  º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX:EDX Puffer                  º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BH ist ein Flagwert zu übergeben, wobei zur Zeit aber alle Werte = 0 zu setzen sind. Das Register CX enthält die Zahl der Worte, die vom Protected-Mode-Stack auf den Real-Mode-Stack zu kopieren sind. In ES:EDI ist die Adresse (Selector:Offset) auf einen Datenpuffer mit folgender Struktur zu übergeben:

```
Offset Len Register
 00H    4    EDI
 04H    4    ESI
 08H    4    EBP
 0CH    4    Reserviert
 10H    4    EBX
 14H    4    EDX
 18H    4    ECX
 1CH    4    EAX
 20H    2    Flags
 22H    2    ES
 24H    2    DS
 26H    2    FS
 28H    2    GS
 2AH    2    IP
 2CH    2    CS
 2EH    2    SP
 30H    2    SS
```

In diesem Puffer werden Daten für die Übergabe an den Real-Mode-Stack übergeben. Bei einem erfolgreichen Aufruf ist anschließend das Carry-Flag gelöscht und im Puffer finden sich die Daten des Real-Mode-Stacks.

Bei einem Fehler setzt die Funktion das Carry-Flag und in AX findet sich ein Fehlercode:

```
AX = 8012H  Linear Memory unavailable
AX = 8013H  Physical Memory unvailable
AX = 8014H  Backing Store unavailable
AX = 8021H  Invalid Value (CX)
```

Die Prozedur muß den Stack vor der Rückkehr in den ursprünglichen Zustand bringen. Die Werte in den Segment-Registers sind als Real-Mode-Segmente anzugeben (keine Selectoren). Die Prozedur ist mit der Anweisung RETF zu beenden.

Call Real Mode Procedure IRET (INT 31, AX = 0302H, V 0.9)

Diese Funktion ruft eine Real-Mode-Funktion mit einem *IRET Frame* auf.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0302H Call Procedure      º
º BH = Flags                     º
º CX = Zahl der Worte            º
º ES:EDI Puffer                  º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º ES:EDI Puffer                  º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BH ist ein Flagwert zu übergeben, wobei zur Zeit aber alle Werte = 0 zu setzen sind. Das Register CX enthält die Zahl der Worte, die vom Protected-Mode-Stack auf den Real-Mode-Stack zu kopieren sind. In ES:EDI ist die Adresse (Selector:Offset) auf einen Datenpuffer mit folgender Struktur zu übergeben:

```
Offset Len Register
 00H    4    EDI
 04H    4    ESI
 08H    4    EBP
 0CH    4    Reserviert
 10H    4    EBX
 14H    4    EDX
 18H    4    ECX
 1CH    4    EAX
 20H    2    Flags
 22H    2    ES
 24H    2    DS
 26H    2    FS
 28H    2    GS
 2AH    2    IP
 2CH    2    CS
 2EH    2    SP
 30H    2    SS
```

In diesem Puffer werden Daten für die Übergabe an den Real-Mode-Stack übergeben. Bei einem erfolgreichen Aufruf ist anschließend das Carry-Flag gelöscht und im Puffer finden sich die Daten des Real-Mode-Stacks.

Bei einem Fehler setzt die Funktion das Carry-Flag und in AX findet sich ein Fehlercode:

```
AX = 8012H  Linear Memory unavailable
AX = 8013H  Physical Memory unvailable
AX = 8014H  Backing Store unavailable
AX = 8021H  Invalid Value (CX)
```

Die Prozedur muß den Stack vor der Rückkehr in den ursprünglichen Zustand bringen. Die Werte in den Segment-Registers sind als Real-Mode-Segmente anzugeben (keine Selectoren). Die Prozedur ist mit der Anweisung IRET zu beenden.

Allocate Real Mode Call-Back Address (INT 31, AX = 0303H, V 0.9)

Die Funktion wird benutzt, um die gleiche Real-Mode-Adresse (Segment:Offset) zu erhalten, die zur Umschaltung vom Real in den Protected Mode benutzt wird. Dies ist zum Beispiel erforderlich bei Treibern, die im Real Mode arbeiten (Maus, etc.). Protected-Mode-Software kann dadurch einen Real-Mode-Treiber benutzen um die Mausbewegung zu kontrollieren.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0303H Callback Address    º
º DS:ESI Adresse Procedur        º
º ES:EDI Adresse Puffer          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CX:DX Adresse Callback         º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In DS:ESI ist die Adresse (Selector:Offset) der aufzurufenden Protected-Mode-Prozedur zu übergeben. In ES:EDI muß ein Zeiger (Selector:Offset) auf einen 32H Byte langen Puffer übergeben werden. In diesem Puffer steht die Real-Mode-Register-Datenstruktur, die bei Aufruf der Callback Prozedur benutzt wird.

Bei einem fehlerhaften Aufruf wird das Carry-Flag gesetzt und in AX steht der Fehlercode:

```
AX = 8051H Callback unavailable
```

Bei erfolgreichem Aufruf wird das Carry-Flag gelöscht und in CX:DX findet sich die Adresse (Segment:Offset) der Real-Mode-Callback-Prozedur.

Beim Aufruf der Callback Prozedur ist das Interruptsystem gesperrt. Der Callback Prozedur werden im Puffer folgende Parameter übergeben:

```
DS:(E)SI = Selector:Offset of Real Mode SS:SP
ES:(E)DI = Selector:Offset Real Mode Call Struktur
SS:(E)SP = Locked Protected Mode API Stack
```

Alle anderen Register sind undefiniert. Die Callback Prozedur muß durch eine IRET-Anweisung beendet werden. Reentrante Aufruf sind innerhalb der Real-Mode-Prozedur zu vermeiden.

Ein DPMI-Server muß mindesten 16 solcher Callback Adressen unterstützen. Um eine reservierte Adresse freizugeben, ist die Funktion 0304H zu verwenden.

Free Real Mode Call-Back Address (INT 31, AX = 0304H, V 0.9)

Diese Funktion gibt eine reservierte Callback Adresse wieder frei.

```
Ö--------------------------------Ì
º          CALL INT 31           º
º                                º
º AX = 0304H Free Callback Adr.  º
º CX:DX freizugebende Adresse    º
û--------------------------------À
º          Return:               º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In CX:DX ist die Real-Mode-Adresse (Segment:Offset) der freizugebenden Callback Prozedur zu übergeben. Bei erfolgreichem Aufruf ist das Carry-Flag gelöscht. Bei Fehlern wird das Carry-Flag gesetzt und in AX findet sich ein Fehlercode:

```
AX = 8024H  Invalid Callback Address
```

Die Freigabe einer Callback Adresse sollte schnellst möglich erfolgen, da diese Resource limitiert ist.

Get State Save/Restore Addresses (INT 31, AX = 0305H, V 0.9)

Benutzt ein Programm den *raw mode switch* oder DOS-Calls von Hardware-Interrupts, dann muß es den Task Status vor der Modeumschaltung sichern. Der Aufruf der Funktion gibt die Adressen zweier Prozeduren zurück, die den Task Status sichern.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0305H Get State Adr.      º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º AX = Größe Puffer              º
º BX:CX Real Mode Adresse        º
º SI:EDI Protected Mode Adresse  º
Û--------------------------------ì
```

Bei erfolgreichem Aufruf wird das Carry-Flag gelöscht und in AX steht die Größe des Puffers (in Byte) zur Sicherung des Status. Die Register BX:CX enthalten die Real-Mode-Adresse (Segment:Offset) der Routine zur Sicherung/Restaurierung des Status. Die Routine wird nur im Real Mode aufgerufen und sichert den Status der Register im Protected Mode.

In SI:EDI wird die Adresse der Protected-Mode-Routine zur Sicherung/Restaurierung der Register im Real Mode zurückgegeben. Die aktuellen Register lassen sich auf dem Stack sichern. Eine Sicherung des Status ist bei Benutzung der Funktionen 0300H, 0301H und 0302H nicht erforderlich.

Die Prozeduren sind über einen CALL FAR-Aufruf zu aktivieren. In ES:EDI ist ein Zeiger auf den Puffer zu übergeben. In diesen Puffer werden die Register gesichert. Der Inhalt von AL bestimmt, ob der Status gesichert oder restauriert wird:

```
AL = 0  Save State
     1  Restore State
```

Die Größe des Puffers muß dem in AX beim Aufruf der Funktion 0305H zurückgegebenen Wert entsprechen.

Get Raw Mode Switch Addresses (INT 31, AX = 0306H, V 0.9)

Diese Funktion gibt die Einsprungadresse für *raw mode* Umschaltungen zurück. Es gelten folgende Aufrufparameter:

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0306H Get Raw Mode Adr.   º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º AX = Größe Puffer              º
º BX:CX Real Mode Adresse        º
º SI:EDI Protected Mode Adresse  º
Û--------------------------------ì
```

In BX:CX steht die Adresse der Real-Mode-Routine, mit dem sich in den Protected Mode schalten läßt. In SI:EDI steht die Adresse der Routine, mit der sich in den Real Mode schalten läßt. Die Werte sind allerdings nur gültig, falls das Carry-Flag gelöscht ist.

Die betreffende Umschaltroutine ist mit einer JMP FAR-Anweisung zu aktivieren. Dabei sind folgende Register zu setzen:

```
AX      = neues DS
CX      = neues ES
DX      = neues SS
(E)BX   = neues (E)SP
SI      = neues CS
(E)DI   = neues (E)IP
```

Mit dem Aufruf wird der Prozessor in den betreffenden Mode umgeschaltet. Die Adresse ist entweder mit Segment:Offset (Real Mode) oder mit Selector:Offset anzugeben. Es ist Sache des Anwendungsprogrammes den Taskstatus vor dem Aufruf zu sichern. Werden bei der Umschaltung in den Protected Mode ungültige Selectoren spezifiziert, löst der Prozessor ein Exeption Fault aus. Die Funktion sollte nach Möglichkeit durch die Aufruf 0300H, 0301H, 0302H und 0304H ersetzt werden.

Get Version (INT 31, AX = 0400H, V 0.9)

Die Funktion ermittelt die Versionsnummer des DPMI-Servers.

```
Ö--------------------------------Ì
º          CALL INT 31           º
º                                º
º AX = 0400H Get Version         º
û--------------------------------À
º          Return:               º
º CY = 0   ok                    º
º AX = Version                   º
º BX = Flags                     º
º CL = Prozessortyp              º
º DX = PIC Interrupt Maske       º
Û--------------------------------ì
```

Bei gelöschtem Carry-Flag enthalten verschiedene Register Informationen über den Server:

```
AH = Hauptversionsnummer
AL = Unterversionsnummer
```

Die Versionsnummer kann Hinweise über die vom Server unterstützten Funktionen geben. Das Register BX ist als Flagregister zu interpretieren.

```
BX = Flags
  Bit 0 = 1: 80386-DPMI-Implementierung
  Bit 1 = 1: Prozessor im Real Mode oder V86-Mode
  Bit 2 = 1: Virtueller Speicher wird unterstützt
  Bit 3    : reserviert
  Bit 4-15 : 0
```

In CL wird der Prozessortyp zurückgegeben:

```
CL = Processor-Typ
    02 = 80286
    03 = 80386
    04 = 80486
```

Das Register DX enthält die Information über den Wert des virtuellen PIC-Basis-Interrupt.

```
DH = virtual master PIC base interrupt
DL = virtual slave PIC base interrupt
```

Das Carry-Flag ist immer gelöscht, da der Aufruf keine Fehler zurückgibt.

Get DPMI Capabilities (INT 31, AX = 0401H, V 1.0)

Die Funktion ermittelt die Möglichkeiten, die der DPMI-Server unterstützt.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0401H Get Capabilities    º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º AX = Flags                     º
º CX = reserviert (0)            º
º DX = reserviert (0)            º
º ES:EDI Adresse Puffer          º
º CY = 1   Fehler                º
Û--------------------------------ì
```

Bei gelöschtem Carry-Flag enthält AX verschiedene Flagbits, die Aufschluß über die unterstützten Funktionen geben:

```
AX = Flags
  Bit 0 = 1: Page accessed/dirthy Funktion unterstützt
  Bit 1 = 1: Exeption Restartmöglichkeit unterstützt
  Bit 2 = 1: Device Mapping Funktion unterstützt
  Bit 3 = 1: Konventionelles Memory Mapping wird unterstützt
  Bit 4 = 1: Demand Zero Fill Funktion unterstützt
  Bit 5 = 1: Write Protect Client Funktion unterstützt
  Bit 6 = 1: Write Protect Server Funktion unterstützt
  Bit 7-15 : reserviert
```

Die Register CX und DX sind reserviert und werden zu 0 gesetzt. In ES:EDI wird die Adresse (Selector:Offset) auf einen 128-Byte-Puffer zurückgegeben. Dieser Puffer wird vom Server mit folgenden Informationen gefüllt:

```
Offset Byte Bedeutung
  00H    1  Host-Hauptversion (dezimal)
  01H    1  Host-Unterversion (dezimal)
  02H    x  ASCIIZ-String DPMI-Hersteller
```

Bei einem fehlerhaften Aufruf wird das Carry-Flag gesetzt. Dann sind die Daten in den Registern ungültig. Die Page Accessed/dirthy-Funktion ermöglicht die Verwaltung der Statusbits einer Seite. Bei einer Exeption-Restartability-Unterstützung kann der Server nach einer fehlerhaften Anweisung die Kontrolle vom Exeption Handler an den Kern zurückgeben. Die Versionsnummern im Ergebnispuffer sind herstellerspezifisch und haben nichts mit der DPMI-Version zu tun.

Get Free Memory Information (INT 31, AX = 0500H, V 0.9)

Die Funktion gibt Informationen über den verfügbaren (freien) physikalischen Speicher zurück.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0500H Get Free Memory     º
º ES:EDI Adresse Puffer          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
Û--------------------------------ì
```

In ES:EDI ist die Adresse (Selector:Offset) auf einen 30H Byte langen Puffer zu übergeben. Bei gelöschtem Carry-Flag enthält der Buffer nach dem Aufruf folgende Daten:

```
Offset Bytes Bedeutung
  00H    4   Größter freier Block in Byte
  04H    4   Anzahl freier unverriegelter Blocks in Seiten
  08H    4   Größter verriegelter Block in Seiten
  0CH    4   Linearer Adressraum Größe in Seiten
  10H    4   Zahl der unverriegelten Seiten
  14H    4   Zahl der freien Seiten
  18H    4   Zahl der physikalischen Seiten
  1CH    4   Freier linearer Adressraum in Seiten
  20H    4   Größe Paging File/Partition in Seiten
  24h-2Fh    reserviert
```

Bei fehlerhaften Aufruf ist das Carry-Flag gesetzt. Nur das erste Feld des Puffers wird bei fehlerfreiem Aufruf mit Sicherheit mit Daten gefüllt. Alle nicht belegten Felder werden dann mit -1 überschrieben.

Allocate Memory Block (INT 31, AX = 0501H, V 0.9)

Die Funktion reserviert lineare Speicherbereiche.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0501H Allocate Memory     º
º BX:CX Größe Block              º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º BX:CX Adresse Block            º
º SI:DI Handle Block             º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX ist die Größe des zu reservierenden Speichers in Byte anzugeben. Nach einem erfolgreichen Aufruf ist das Carry-Flag gelöscht. BX:CX enthält dann die lineare Adresse des reservierten Speicherblockes. In SI:DI steht ein Handle für diesen Block.

Die Funktion reserviert keinen Selektor für den Speicherblock. Dies muß die Anwendung selbst durchführen. Die Speicheranforderung erfolgt seitenweise (4 Kbyte). Bei Fehlern wird das Carry-Flag gesetzt und AX enthält die Fehlercodes:

```
AX = 8012H  Linear Memory unavailable
     8013H  Physical memory unavailable
     8014H  Backing Store unavailable
     8016H  Handle unavailable
     8021H  Invalid Value (BX:CX)
```

Free Memory Block (INT 31, AX = 0502H, V 0.9)

Die Funktion gibt einen Speicherblock wieder frei.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0502H Free Memory         º
º SI:DI Handle Block             º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In SI:DI ist der Handle auf den freizugebenden Block zu übergeben. Ein Fehler beim Aufruf setzt das Carry-Flag. In AX steht der Fehlercode:

```
AX = 8023H  Invalid Handle
```

Resize Memory Block (INT 31, AX = 0503H, V 0.9)

Die Funktion ändert die Größe eines Blockes.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0503H Resize Memory Block º
º BX:CX neue Blockgröße          º
º SI:DI Handle Block             º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º BX:CX Lineare Adresse Block    º
º SI:DI neuer Handle Block       º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

BX:CX enthält beim Aufruf die neue Größe des Speicherblocks. In SI:DI ist das Handle des betreffenden Speicherblockes zu übergeben. Bei einem erfolgreichen Aufruf wird der Speicher in der Größe angepaßt. Das Carry-Flag ist dann gelöscht und in BX:CX findet sich die lineare Adresse des Speicherblockes. In SI:DI wird der Handle zurückgegeben.

Bei Fehlern wird das Carry-Flag gesetzt und AX enthält die Fehlercodes:

```
AX = 8012H  Linear Memory unavailable
     8013H  Physical memory unavailable
     8014H  Backing Store unavailable
     8016H  Handle unavailable
     8021H  Invalid Value (BX:CX)
     8023H  Invalid Handle (SI:DI)
```

Allocate Linear Memory Block (INT 31, AX = 0504H, V 1.0)

Die Funktion reserviert einen Block mit linearen Adressen, die auf einer Seitengrenze ausgerichtet sind.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0504H Allocate lin. Block º
º EBX= Adresse Seite             º
º ECX= Größe BLock               º
º EDX= Flags                     º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º EBX= Adresse Memory Block      º
º ESI= Handle Memory Block       º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In EBX ist die gewünschte lineare Adresse des Blockes zu übergeben. Die Adresse muß auf einer Seitengrenze ausgerichtet sein. In ECX steht die Größe des Blockes in Byte. EDX funktiert als Flag:

```
EDX Bit 0: 1  create committed pages
           0  create uncommited pages
        1-31  reserviert
```

Nach dem Aufruf enthält EBX die lineare Adresse des Speicherblockes. ESI enthält den Handle zu diesem Block. Bei einem fehlerhaften Aufruf ist das Carry-Flag gesetzt und AX enthält einen Fehlercode:

```
AX = 8001H  Unsupported Function
     8012H  Linear Memory unavailable
     8013H  Physical memory unavailable
     8014H  Backing Store unavailable
     8016H  Handle unavailable
     8021H  Invalid Value (BX:CX)
     8025H  Invalid Linear Address
```

Die Funktion richtet reservierte Seiten immer an Seitengrenzen (4 Kbyte) aus.

Resize Linear Memory Block (INT 31, AX = 0505H, V 1.0)

Die Funktion verändert die Blockgröße eines linearen Speicherbereiches.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0505H Resize lin. Block   º
º ESI= Block Handle              º
º ECX= neue Größe Block          º
º EDX= Flags                     º
º ES:EBX Pufferadresse           º
º EDI= Selektoren im Puffer      º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º EBX= Adresse Memory Block      º
º ESI= Handle Memory Block       º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

ESI enthält den Handle zu diesem Block. In ECX steht die neue Größe des Blockes in Byte. EDX fungiert als Flag:

```
EDX Bit 0: 1  create committed pages
           0  create uncommited pages
    Bit 1: 1  Segment Descriptor update required
           0  Not update Segment Descriptors
    Bit 2-31  reserviert (0)
```

Ist Bit 1 in EDX gesetzt, wird die Adresse (Selector:Offset) eines Puffers in ES:EBX übergeben. In diesem Puffer sind für jeden Selector 16 Byte zu reservieren. In EDI wird die Zahl der Einträge im Puffer übergeben. Nach dem Aufruf enthält der Puffer linearen Adressen (Selectoren) der Speicherblöcke. In EBX wird die neue lineare Basisadresse des Blockes zurückgegeben, während in ESI das neue Handle für den Block steht. Bei einem fehlerhaften Aufruf ist das Carry-Flag gesetzt und AX enthält einen Fehlercode:

```
AX = 8001H  Unsupported Function
     8012H  Linear Memory unavailable
     8013H  Physical memory unavailable
     8014H  Backing Store unavailable
     8016H  Handle unavailable
     8021H  Invalid Value (BX:CX)
     8025H  Invalid Linear Address
```

Die Funktion richtet reservierte Seiten immer an Seitengrenzen (4 Kbyte) aus.

Get Page Attributes (INT 31, AX = 0506H, V 1.0)

Die Funktion gibt die Attribute einer oder mehrerer Seiten zurück, die vorher mit der Funktion AX = 0504H allociert wurden.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0506H Get Page Attribut   º
º ESI= Handle Memory Block       º
º EBX= Basisadresse Seite        º
º ECX= Zahl der Seiten           º
º ES:EDX Adresse Puffer          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In EBX ist die Basisadresse der gewünschten Seite oder des Blockes zu übergeben. In ESI steht die Handle Nummer. ECX enthält die Zahl der Seiten, deren Attribut zurückzugeben ist. ES:EDX enthält die Adresse (Selector:Offset) eines Puffers, in dem die Attribute einer Seite zurückgegeben werden. Der Puffer besitzt pro Seite einen 16-Bit-Eintrag. Jeder Eintrag ist mit folgendem Format kodiert:

```
Attributbit im Puffer

  Bit 0-2: Page Typ
           0 uncommited page
           1 committed page
           2 mapped page
        3: 0 read only page
           1 read/write page
        4: 0 accessed/dirthy bits not available
           1 accessed/dirthy bits available
        5: 0 page not accessed (if Bit 4 = 1)
           1 page accessed
        6: 0 page not modified (if Bit 4 = 1)
           1 page modified
     7-15: 0 reserviert
```

Die Werte im Puffer sind nur gültig falls nach dem Aufruf das Carry-Flag gelöscht ist. Ist das Carry-Flag gesetzt, findet sich in AX ein Fehlercode:

```
AX = 8001H  Unsupported Function
     8023H  Invalid Handle
     8025H  Invalid Linear Address
```

Die Funktion steht nur unter DPMI 1.0 zur Verfügung.

Set Page Attributes (INT 31, AX = 0507H, V 1.0)

Die Funktion setzt die Attribute einer oder mehrerer Seiten, die vorher mit der Funktion AX = 0504H allociert wurden.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0507H Set Page Attribut   º
º ESI= Handle Memory Block       º
º EBX= Offset Memory Block       º
º ECX= Zahl der Seiten           º
º ES:EDX Adresse Puffer          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
º ECX= Seitenzahl                º
Û--------------------------------ì
```

In EBX ist die Basisadresse (Offset) der gewünschten Seite oder des Blockes, dessen Attribute zu modifizieren sind, zu übergeben. In ESI steht die Handle Nummer. ECX enthält die Zahl der Seiten, deren Attribut zu setzen ist. ES:EDX enthält die Adresse (Selector:Offset) eines Puffers, in dem die Attribute einer Seite übergegeben werden. Der Puffer besitzt pro Seite einen 16-Bit-Eintrag mit der folgenden Kodierung.

```
Attributbit im Puffer
  Bit 0-2: Page Typ
           0 create uncommited page
           1 create committed page
           2 ----
           3 modify attributes/ Change not type
        3: 0 read only page
           1 read/write page
        4: 0 don't modify accessed/dirthy bits
           1 set accessed/dirthy bits
        5: 0 mark page not accessed (if Bit 4 = 1)
           1 mark page accessed (if Bit 4 = 1)
        6: 0 mark page not modified (if Bit 4 = 1)
           1 mark page modified (if Bit 4 = 1)
    7-15: 0 reserviert
```

Die Werte im Puffer werden nur übernommen, falls nach dem Aufruf das Carry-Flag gelöscht ist. Ist das Carry-Flag gesetzt, findet sich in AX ein Fehlercode:

```
AX = 8001H  Unsupported Function
     8002H  Invalid State
     8013H  Physical Memory unavailable
     8014H  Backing Store unavailable
     8021H  Invalid Value
     8023H  Invalid Handle
     8025H  Invalid Linear Address
```

In ECX steht dann die Zahl der Seiten, deren Attribute modifiziert wurden.

Map Device in Memory Block (INT 31, AX = 0508H, V 1.0)

Die Funktion setzt die physikalische Adresse die einem Geräte zugewiesen wurde in eine lineare Adresse eines Speicherblockes um. Der Block muß vorher mit der Funktion AX = 0504H allociert wurden.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0508H Map Device Block    º
º ESI= Handle Memory Block       º
º EBX= Basisadresse Seite        º
º ECX= Zahl der Seiten           º
º EDX= Adresse Gerät             º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In ESI ist der Handle des Speicherblocks zu übergeben. Dieser Handle wird von der Funktion 0504H zurückgegeben. EBX enthält den Offset des Memory Blockes, der dem Gerät zugewiesen werden soll. Der Block muß auf einer Seitengrenze beginnen. In ECX wird die Zahl der zuzuweisenden Seiten (4 Kbyte) übergeben. In EDX steht schließlich die physikalische Adresse des Gerätes (Device), die ebenfalls auf einer Seitengrenze liegen muß.

Die Zuweisung erfolgt nur, falls nach dem Aufruf das Carry-Flag gelöscht ist. Ist das Carry-Flag gesetzt, findet sich in AX ein Fehlercode:

```
 AX = 8001H  Unsupported Function
      8003H  System Integrity (invalid Device Address)
      8023H  Invalid Handle
      8025H  Invalid Linear Address
```

16-Bit-DPMI-Server unterstützen diese Funktion nicht. Die Unterstützung von 32-Bit-DPMI-Servern ist optional. Durch die Zuweisung physikalischer Geräteadresse zu logischen Speicherblöcken können die Geräte mit einem NEAR-Zeiger in 32-Bit-Programmen angesprochen werden. Die Funktion kann auch Adressen im 1 Mbyte zuweisen. So kann zum Beispiel ein Video Adapter direkt angesprochen werden.

Map Conventional Memory in Memory Block (INT 31, AX = 0509H, V 1.0)

Die Funktion setzt eine lineare Adresse im 1 Mbyte auf einen Speicherblock um, der vorher mit der Funktion AX = 0504H allociert wurde.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0509H Map Convent. Mem.   º
º ESI= Handle Memory Block       º
º EBX= Basisadresse Seite        º
º ECX= Zahl der Seiten           º
º EDX= Adresse Convent. Mem.     º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In ESI ist der Handle des Speicherblocks zu übergeben. Dieser Handle wird von der Funktion 0504H zurückgegeben. EBX enthält den Offset des Memory Blockes, dem der konventionelle Speicher zugewiesen werden soll. Der Block muß auf einer Seitengrenze beginnen. In ECX wird die Zahl der zuzuweisenden Seiten (4 Kbyte) übergeben. In EDX steht schließlich die lineare Adresse des 1-Mbyte-Speicherbereiches, die ebenfalls auf einer Seitengrenze liegen muß.

Die Zuweisung erfolgt nur, falls nach dem Aufruf das Carry-Flag gelöscht ist. Ist das Carry-Flag gesetzt, findet sich in AX ein Fehlercode:

```
 AX = 8001H  Unsupported Function
      8003H  System Integrity (invalid Device Address)
      8023H  Invalid Handle
      8025H  Invalid Linear Address
```

16-Bit-DPMI-Server unterstützen diese Funktion nicht. Die Unterstützung von 32-Bit-DPMI-Servern ist optional. Durch die Zuweisung physikalischer Speicheradressen zu logischen Speicherblöcken können die Bereiche mit einem NEAR-Zeiger in 32-Bit-Programmen angesprochen werden. Die Funktion kann auch Adressen im 1 Mbyte zuweisen.

Get Memory Block Size and Base (INT 31, AX = 050AH, V 1.0)

Die Funktion ermittelt die Größe und die Basisadresse eines Speicherblocks, der vorher mit der Funktion AX = 0501H oder 0504H allociert wurde.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 050AH Get Size and Base   º
º SI:DI Memory Block Handle      º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º SI:DI Größe Memory Block (Byte)º
º BX:CX Basisadresse Block       º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In SI:DI ist der Handle des Speicherblocks zu übergeben. Dieser Handle wird von der Reservierung des Speichers zurückgegeben. Bei einem fehlerfreien Aufruf ist anschließend das Carry-Flag gelöscht und SI:DI enthält die Größe des Speicherblockes in Byte. In BX:CX steht dann die Basisadresse des Speicherblockes. Ist das Carry-Flag gesetzt, findet sich in AX ein Fehlercode:

```
8023H  Invalid Handle
```

Get Memory Information (INT 31, AX = 050BAH, V 1.0)

Die Funktion ermittelt Informationen über physikalische und virtuelle Speicherbereiche.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 050BH Get Memory Info     º
º ES:EDI Adresse Puffer          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
Û--------------------------------ì
```

In ES:EDI ist die Adresse (Selector:Offset) eines 128-Byte-Puffers zu übergeben. Dieser besitzt folgenden Aufbau:

```
Offset Bytes Bedeutung
 00H     4   Total allocated bytes of physical
             memory controlled by DPMI host
 04H     4   Total allocated bytes of virtual
             memory controlled by DPMI host
 08H     4   Total available bytes of virtual
             memory controlled by DPMI host
 0CH     4   Total allocated bytes of virtual
             memory for this virtual maschine
 10H     4   Total available bytes of virtual
             memory for this virtual maschine
 14H     4   Total allocated bytes of virtual
             memory for this client
 18H     4   Total available bytes of virtual
             memory for this client
 1CH     4   Total locked bytes of memory for
             this client
 20H     4   Maximum locked bytes of memory for
             this client
 24H     4   Highest linear address avilable for
             this client
 28H     4   Size (bytes) of largest available
             free memory block
 2CH     4   Size of minimum allocation unit in
             bytes
 30H     4   Size of allocation unit in bytes
 34H    4C   reserviert
```

Bei DPMI 1.0 ist der Aufruf immer erfolgreich, während bei DPMI 0.9 die Funktion nicht unterstützt wird. Die Angabe *available bytes* gibt immer die Zahl der noch freien Bytes an.

Lock Linear Region (INT 31, AX = 0600H, V 0.9)

Die Funktion sperrt den angegebenen linearen Adressbereich. Dann kann dieser Speicher nicht mehr auf die Platte ausgelagert werden.

```
Ö---------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0600H Lock Linear Region  º
º BX:CX  Adresse Region          º
º SI:DI  Größe Region            º
û---------------------------------À
º          Return:               º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û---------------------------------ì
```

In BX:CX muß die Anfangsadresse des linearen Bereiches angegeben werden. In SI:DI erwartet der Server die Größe des zu sperrenden Bereiches in Byte. Ein gesetztes Carry-Flag zeigt einen Fehler beim Aufruf an. In AX findet sich dann der Fehlercode:

```
AX = 8013H  Physical Memory unavailable
     8017H  Lock count exeeded
     8025H  Invalid linear address
```

In diesem Fall wird die Speicherseite nicht gesperrt. Die Funktion kann durchaus mehrfach für eine gegebene Seite aufgerufen werden. Die Zahl der Aufrufe pro Seite wird in einem internen Zähler im Server gespeichert. Erst wenn der Zähler wieder auf 0 zurückgesetzt wurde, kann die Seite wieder ausgelagert werden.

Unlock Linear Region (INT 31, AX = 0601H, V 0.9)

Die Funktion gibt den angegebenen linearen Adressbereich wieder frei, d.h. dieser darf wieder auf Platte ausgelagert werden.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0601H Unlock Linear Regionº
º BX:CX  Adresse Region          º
º SI:DI  Größe Region            º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX muß die Anfangsadresse des linearen Bereiches angegeben werden. In SI:DI erwartet der Server die Größe des freizugebenden Bereiches in Byte. Ein gesetztes Carry-Flag zeigt einen Fehler beim Aufruf an. In AX findet sich dann der Fehlercode:

```
AX = 8002H  Invalid state
     8025H  Invalid linear address
```

In diesem Fall wird die Speicherseite nicht freigegeben. Bei jedem Aufruf der Funktion wird der Lockzähler für diesen Bereich decrementiert. Erst wenn der Zähler = 0 ist, wird die Seite freigegeben.

Mark Real Mode Region as Pageable (INT 31, AX = 0602H, V 0.9)

Signalisiert dem DPMI-Server, daß Speicher unterhalb 1-Mbyte seitenweise auf Platte ausgelagert werden darf.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0602H Mark Region         º
º BX:CX  Adresse Region          º
º SI:DI  Größe Region            º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX muß die Anfangsadresse des linearen Bereiches angegeben werden. In SI:DI erwartet der Server die Größe des zu markierenden Bereiches in Byte. Ein gesetztes Carry-Flag zeigt einen Fehler beim Aufruf an. In AX findet sich dann der Fehlercode:

```
AX = 8002H  Invalid state
     8025H Invalid linear address
```

In diesem Fall wird die Speicherseite nicht markiert. Falls der markierte Bereich in eine Seite hineinreicht, wird diese Seite nicht mehr markiert. Es sollte kein Speicher markiert werden, der nicht zur jeweiligen Applikation gehört. Speicher im Real Mode muß vor Beendigung eines Programmes mittels der Funktion 0603H gesperrt werden. Die Funktion 0602H sollte nicht mehrfach aufgerufen werden, da kein Lock-Zähler geführt wird.

Relock Real Mode Region (INT 31, AX = 0603H, V 0.9)

Die Funktion wird benutzt, um Speicherbereich wieder zu sperren, die per Funktion 0602H zur Auslagerung freigegeben wurden.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0603H Relock Region       º
º BX:CX  Adresse Region          º
º SI:DI  Größe Region            º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX muß die Anfangsadresse des linearen Bereiches angegeben werden. In SI:DI erwartet der Server die Größe des zu sperrenden Bereiches in Byte. Ein gesetztes Carry-Flag zeigt einen Fehler beim Aufruf an. In AX findet sich dann der Fehlercode:

```
AX = 8002H  Invalid state
     8013H  Physical memory unavailable
     8025H  Invalid linear address
```

In diesem Fall wird die Speicherseite nicht gesperrt. Falls der markierte Bereich in eine Seite hineinreicht, wird diese Seite nicht gesperrt.

Get Page Size (INT 31, AX = 0604H, V 0.9)

Diese Funktion gibt die Größe einer einzelnen Speicherseite in Byte zurück.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0604H Get Size            º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º BX:CX  Größe der Seite (Bytes) º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX wird bei gelöschtem Carry-Flag die Größe der Seite in Byte zurückgegeben. Ein gesetztes Carry-Flag zeigt einen Fehler beim Aufruf an. In AX findet sich dann der Fehlercode:

```
AX = 8001H  Unsupported function
```

Die Funktionen 0700H und 0701H sind reserviert.

Mark Page as Demand Paging Candidate (INT 31, AX = 0702H, V 0.9)

Die Funktion informiert das Betriebssystem, daß einige Seiten in die Liste der auszulagerenden Pages mit aufgenommen werden. Hierdurch kann das Betriebssystem die am wenigsten benutzten Seiten zuerst auslagern, was zur Optimierung der Systemleistung dient. Wenn ein Programm einen Speicherbereich längere Zeit nicht benötigt, sollte die Funktion aufgerufen werden.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0702H Mark Demand Page    º
º BX:CX  Adresse Region          º
º SI:DI  Größe Region            º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX muß die lineare Anfangsadresse des Bereiches angegeben werden. SI:DI enthält die Größe in Byte. Bei einem fehlerhaften Aufruf ist anschließend das Carry-Flag gesetzt und in AX steht der Fehlercode:

```
AX = 8025H  Invalid linear address
```

Nachdem eine Seite markiert wurde, kann das Betriebssystem diese bei Speicherplatzmangel auslagern.

Discard Page Contents (INT 31, AX = 0703H, V 0.9)

Diese Funktion verwirft einen angegebenen Speicherbereich. Der Inhalt dieses Bereiches ist dann undefiniert.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0703H Discard Page        º
º BX:CX  Adresse Region          º
º SI:DI  Größe Region            º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX muß die lineare Anfangsadresse des Bereiches angegeben werden. SI:DI enthält die Größe in Byte. Bei einem fehlerhaften Aufruf ist anschließend das Carry-Flag gesetzt und in AX steht der Fehlercode:

```
AX = 8025H  Invalid linear address
```

Umfaßt der Bereich nur Teile einer Seite, oder ist die Seite gesperrt, wird die Funktion nicht ausgeführt.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0703H Discard Page        º
º BX:CX  Adresse Region          º
º SI:DI  Größe Region            º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX muß die lineare Anfangsadresse des Bereiches angegeben werden. SI:DI enthält die Größe in Byte. Bei einem fehlerhaften Aufruf ist anschließend das Carry-Flag gesetzt und in AX steht der Fehlercode:

```
AX = 8025H  Invalid linear address
```

Umfaßt der Bereich nur Teile einer Seite, oder ist die Seite gesperrt, wird die Funktion nicht ausgeführt.

Physical Address Mapping (INT 31, AX = 0800H, V 0.9)

Die Funktion konvertiert einen physikalische Adresse in eine lineare Adresse. Damit lassen sich Treiberadressen, die oberhalb der 1-Mbyte-Grenze liegen, erfragen.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0800H Adress Mapping      º
º BX:CX  physical. Adresse       º
º SI:DI  Größe Region            º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º BX:CX  lineare   Adresse       º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX muß die physikalische Adresse des Bereiches angegeben werden. SI:DI enthält die Größe in Byte. Bei einem fehlerhaften Aufruf ist anschließend das Carry-Flag gesetzt und in AX steht der Fehlercode:

```
AX = 8003H  System integrity
     8021H  Invalid value
```

Bei gelöschtem Carry-Flag wird in BX:CX die konvertierte lineare Adresse zurückgegeben. Das Programm muß dann einen Descriptor für den Zugriff auf die Adresse anlegen und initialisieren. Der Aufruf wird nicht von allen Implementierungen unterstützt und sollte nur benutzt werden, falls ein direkter Zugriff erforderlich ist.

Free Physical Address Mapping (INT 31, AX = 0801H, V 1.0)

Die Funktion hebt eine per Funktion 8000H aktivierte Adresszuweisung wieder auf.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0801H Free Adress Mapping º
º BX:CX  physical. Adresse       º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX muß die von der Funktion 0800H zurückgegebene lineare Adresse übergeben werden. Bei einem fehlerhaften Aufruf ist anschließend das Carry-Flag gesetzt und in AX steht der Fehlercode:

```
8025H  Invalid linear address
```

Bei gelöschtem Carry-Flag wird die Adresszuweisung aufgehoben.

Get and Disable Virtual Interrupt State (INT 31, AX = 0900H, V 0.9)

Mit dieser Funktion wird das virtuelle Interrupt-Flag gesperrt und der vorherige Status des Flags zurückgegeben.

```
Ö--------------------------------Ì
°         CALL INT 31            °
°                                °
° AX = 0900H Disable Interrupt   °
û--------------------------------À
°         Return:                °
° CY = 0   ok                    °
° AL = Status Flag               °
Û--------------------------------ì
```

Nach dem Aufruf ist das Virtual-Interrupt-Flag disabled und in AL findet sich der Zustand des Flags vor dem Aufruf:

```
AL = 0  Virtual Interrupt war disabled
     1  Virtual Interrupt war enabled
```

Der Inhalt von AH wird nicht verändert.

Get and Enable Virtual Interrupt State (INT 31, AX = 0901H, V 0.9)

Mit dieser Funktion wird das virtuelle Interrupt-Flag gesperrt und der vorherige Status des Flags zurückgegeben.

```
Ö--------------------------------Ì
°         CALL INT 31            °
°                                °
° AX = 0901H Enable  Interrupt   °
û--------------------------------À
°         Return:                °
° CY = 0   ok                    °
° Al = Status Flag               °
Û--------------------------------ì
```

Nach dem Aufruf ist das Virtual-Interrupt-Flag enabled und in AL findet sich der Zustand des Flags vor dem Aufruf:

```
AL = 0  Virtual Interrupt war disabled
     1  Virtual Interrupt war enabled
```

Der Inhalt von AH wird nicht verändert.

Get Virtual Interrupt State (INT 31, AX = 0902H, V 0.9)

Mit dieser Funktion läßt sich der Status des Interrupt-Flags abfragen.

```
Ö--------------------------------Ì
°         CALL INT 31            °
°                                °
° AX = 0902H Get Interrupt State °
û--------------------------------À
°         Return:                °
° CY = 0   ok                    °
° Al = Status Flag               °
Û--------------------------------ì
```

Nach dem Aufruf steht in AL der Zustand des Virtual-Interrupt-Flag:

```
AL = 0  Virtual Interrupt war disabled
     1  Virtual Interrupt war enabled
```

Der Inhalt von AH wird nicht verändert.

Get Vendor spezific API Entry Point (INT 31, AX = 0A00H, V 0.9)

Einige DOS-Extender weisen Erweiterungen zum Standard DPMI auf. Mit der Funktion läßt sich der Einsprungpunkt für die erweiterten API-Funktionen ermitteln.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0A00H Vendor API Entry    º
º DS:ESI Adresse ASCIIZ-String   º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º ES:EDI Adresse Entry Point     º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In DE:ESI ist die Adresse (Selector:Offset) auf einen ASCIIZ-String mit der Identifikation für den DPMI-Host zu übergeben. Bei fehlerhaftem Aufruf ist anschließend das Carry-Flag gesetzt und in AX steht ein Fehlercode:

```
AX = 8001H  Unsupported function
```

Andernfalls steht in ES:EDI die Adresse (Selector:Offset) der API-Funktionen. Diese Adresse ist per CALL FAR aufzurufen. Die Registerbelegungen für die erweiterten API-Aufrufe sind herstellerspezifisch.

Set Debug Watchpoint (INT 31, AX = 0B00H, V 0.9)

Die Funktion setzt einen Überwachungspunkt für die angegebene lineare Adresse im Debugregister. Die Funktion ist erst ab dem 80386-Prozessor verfügbar.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0B00H Set Watchpoint      º
º BX:CX Adresse Watchpoint       º
º DL = Größe Watchpoint          º
º DH = Type des Watchpoint       º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º BX = Handle Watchpoint         º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX:CX ist die lineare Adresse des Überwachungspunktes anzugeben. DL nimmt die Größe (1, 2 oder 4 Byte) des Überwachungspunktes auf. Der Typ des Überwachungspunktes muß in DH übergeben werden. Hierbei gilt:

```
DH = 0  execute
     1  write
     2  read/write
```

Bei einem fehlerfreien Aufruf wird der Überwachungspunkt eingestellt und in BX steht anschließend ein Handle für den Punkt. Dieses Handle wird für weitere Funktionen

benötigt. Bei Fehlern setzt der Server das Carry-Flag und gibt in AX den Fehlercode zurück:

```
AX = 8016H  Too many breakpoints
     8021H  Invalid value
     8025H  Invalid linear address
```

Der Handle für den Überwachungspunkt wird ab der DPMI-Version 1.0 auf Werte zwischen 0 und 14 begrenzt. Der Handle korrespondiert mit den Bits im virtuellen Register DR6, welches im Execption Frame zurückgegeben wird.

Clear Debug Watchpoint (INT 31, AX = 0B01H, V 0.9)

Die Funktion löscht einen Überwachungspunkt für den angegebenen Handle. Die Funktion ist erst ab dem 80386-Prozessor verfügbar.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0B01H Clear Watchpoint    º
º BX = Watchpoint Handle         º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX ist der Handle des Überwachungspunktes anzugeben. Bei einem fehlerfreien Aufruf wird der Überwachungspunkt gelöscht und das Carry-Flag ist ebenfalls 0. Bei Fehlern, setzt der Server das Carry-Flag und gibt in AX den Fehlercode zurück:

```
AX = 8023H  Invalid handle
```

Get State of a Debug Watchpoint (INT 31, AX = 0B02H, V 0.9)

Die Funktion ermittelt den Status des Überwachungspunktes für den angegebenen Handle. Die Funktion ist erst ab dem 80386-Prozessor verfügbar.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0B02H Get Watchpt. Status º
º BX = Watchpoint Handle         º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º AX = Status                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX ist der Handle des Überwachungspunktes anzugeben. Bei einem fehlerfreien Aufruf wird der Status des Überwachungspunktes in AX zurückgegeben und das Carry-Flag ist gelöscht. Der Status besitzt folgende Kodierung:

```
AX = Status
Bit 0: 0 Watchpoint wurde erreicht (encountered)
       1 Watchpoint nicht erreicht
```

Die restlichen Bits sind unbelegt. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt und AX enthält den Fehlercode:

```
AX = 8023H  Invalid handle
```

Um den Status zurückzusetzen, ist die Funktion 0B03H zu verwenden.

Reset Debug Watchpoint (INT 31, AX = 0B03H, V 0.9)

Die Funktion setzt den Status eines Überwachungspunktes zurück.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0B03H Reset Status        º
º BX = Watchpoint Handle         º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In BX ist der Handle des Überwachungspunktes anzugeben. Bei einem fehlerfreien Aufruf wird der Status des Überwachungspunktes zurückgesetzt, der Handle bleibt erhalten und das Carry-Flag ist gelöscht. Bei einem Fehler ist das Carry-Flag nach dem Aufruf gesetzt und AX enthält den Fehlercode:

```
AX = 8023H Invalid handle
```

Install Resident Service Provider Callback (INT 31, AX = 0C00H, V 1.0)

Protected Mode TSR-Programme können Funktionen für 16- oder 32-Bit-Programme bereitstellen. Ein TSR-Programm kannt die Funktion 0C00H nutzen um Information über gestartete oder beendete Programme der gleichen virtuellen Maschine vom Host zu erhalten.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0C00H Install Callback    º
º ES:EDI Adresse Puffer          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In ES:EDI ist die Adresse (Selector:Offset) eines 40-Byte-Puffers zu übergeben. Dieser Puffer besitzt folgende Struktur:

```
Offset Bytes Bemerkung
  00H    8   Descriptor für 16-Bit-Datensegment
  08H    8   Descriptor für 16-Bit-Codesegment
  10H    2   Offset 16-Bit-Callback Prozedur
  12H    2   reserviert
  14H    8   Descriptor für 32-Bit-Datensegment
  1CH    8   Descriptor für 32-Bit-Codesegment
  24H    4   Offset 32-Bit-Callback Prozedur
```

Die Einträge im Puffer sind vor dem Aufruf der Funktion zu setzen. Bei fehlerfreiem Aufruf wird das Carry-Flag gelöscht. Bei gesetztem Carry-Flag enthält AX einen Fehlercode:

```
AX = 8015H  Callback unavailable
     8021H  Invalid value
     8025H  Invalid linear address
```

Ein DPMI-Client, der diese Funktion aufruft, signalisiert damit dem Server, daß er bestimmte residente Dienste erbringen möchte. Der Client muß sich dann resident installieren. TSR-Programme, die nur im Real Mode arbeiten, sollten diese Funktion nicht benutzen. Wird nur ein Mode (16/32-Bit) unterstützt, ist der betreffende Descriptor mit 0 zu initialisieren.

Terminate and Stay Resident (INT 31, AX = 0C01H, V 1.0)

Diese Funktion ist zu benutzen, um einen Clienten im Protected Mode resident zu installieren.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0C01H TSR Install         º
º BL = Return Code               º
º DX = Speichergröße (Paragraph) º
û--------------------------------À
º         Return:                º
º ---                            º
Û--------------------------------ì
```

In BL ist ein Return Code für DOS zu übergeben, der dann mit ERRORLEVEL abfragbar ist. In DX muß die Zahl der vom TSR-Programm belegten Paragraphen übergeben werden. Für das Programm wird dann der entsprechende Bereich im Speicher permanent reserviert. Das TSR-Programm ist dann resident installiert und die Kontrolle geht an DOS zurück.

Die Funktion darf nur von DPMI-Clienten benutzt werden, die Funktionen für ander DPMI-Programme erbringen. Ansonsten sind die DOS-TSR-Funktionen zu benutzen. Der Wert in DX spezifiziert nur den Speicherbedarf für den DOS-Teil des TSR-Programmes. Der vom Programm belegte Protected-Mode-Speicher bleibt dagegen solange reserviert, bis er explizit freigegeben wird. Ist DX = 0, wird die INT 21-Funktion 4CH ausgeführt, andernfalls wird die Funktion 31H benutzt. Falls das Programm die INT 31-Funktion 0C00H nicht aufgerufen hat, wird es sofort beendet.

Allocate Shared Memory (INT 31, AX = 0D00H, V 1.0)

Diese Funktion ist zu benutzen, um von einem Clienten Shared Memory zu reservieren.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0D00H Alloc. Shared Memoryº
º ES:EDI Adresse Puffer          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In ES:EDI ist die Adresse (Selector:Offset) auf einen Puffer mit der folgenden Datenstruktur zu übergeben:

```
Offset Bytes Bemerkung
  00H    4   Länge des benötigten Speichers (Byte)
  04H    4   Länge allocierter Speicher (Server)
  08H    4   Shared Memory Handle (Server)
  1CH    4   Lineare Adresse Shared Memory
             (Selector:Offset) (Server)
  10H    6   Adresse (Selector:Offset32) ASCIIZ-
             String mit dem Namen des Bereiches
  16H    2   reserviert
  18H    4   reserviert (0)
```

Der Eintrag für die Länge des benötigten Speichers ist vor dem Aufruf der Funktion zu setzen. Bei fehlerfreiem Aufruf wird das Carry-Flag gelöscht. Im Puffer werden dann vom Server die ermittelten Werte zurückgegeben. So wird der Speicherblock mit einem Namen versehen, dessen Adresse im Puffer zurückgegeben wird. Bei gesetztem Carry-Flag enthält AX einen Fehlercode:

```
AX = 8012H  Linear memory unavailable
     8013H  Physical memory unavailable
     8014H  Backing store unavailable
     8016H  Handle unavailable
     8021H  Invalid Value
```

Bei 16-Bit-Servern ist der obere Anteil des Offset für den ASCII-String auf 0 gesetzt. Der Name des Blockes ist maximal 128 Zeichen lang. Die lineare Adresse, die vom Server zurückgegeben wird, bleibt für alle Clienten gleich. Der Client braucht dann nur noch einen Descriptor für diesen Block einzurichten und kann dann zugreifen. Die ersten 16 Byte eines Blockes werden mit 00H initialisiert und können zur Identifikation benutzt werden. Terminiert der Client, der den Speicher eingerichtet hat, wird der Speicher freigegeben.

Free Shared Memory (INT 31, AX = 0D01H, V 1.0)

Diese Funktion ist zu benutzen, um von einem Clienten Shared Memory freizugeben.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0D01H Free Shared Memory  º
º SI:DI Handle Shared Memory     º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In SI:DI ist der bei der Allocierung des Shared Memory erzeugte Handle zu übergeben. Die Funktion gibt dann den Speicher frei und löscht das Carry-Flag.

Bei gesetztem Carry-Flag enthält AX einen Fehlercode:

```
AX = 8023H  Invalid Handle
```

Das Handle ist nach der Freigabe des Speichers ungültig. Der Server führt Zähler für virtuelle Maschinen und Clienten die den Block nutzen. Erreicht der Zähler den Wert 0, wird der Block wieder freigegeben. Die Clienten müssen jedoch selbst dafür sorgen, daß die Descriptoren wieder freigegeben werden.

Serialize on Shared Memory (INT 31, AX = 0D02H, V 1.0)

Diese Funktion fordert den Server zu Serialisierung des Shared Memory auf. Damit kann der Zugriff auf den Block eingeschränkt werden.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0D02H Serialize Block     º
º SI:DI Handle Shared Memory     º
º DX = Option Flags              º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In SI:DI ist der bei der Allocierung des Shared Memory erzeugte Handle zu übergeben. In DX ist die Option für den Zugriff auf den Speicher zu definieren:

```
DX Bit 0 = 0: suspend client until serilization
              available
           1: return immediately  with error if
              serialization not available
   Bit 1 = 0: exclusive serialization requested
           1: shared serialization requestet
   Bit 2-15   reserviert (0)
```

Bei gesetztem Carry-Flag enthält AX einen Fehlercode:

```
AX = 8004H  Deadlock
     8005H  Request canceled (AX 0D03H)
     8017H  Lock count exeeded
     8018H  Exclusive serialization already
            owned by another client
     8019H  shared serialization already
            owned by another client
     8023H  Invalid Handle
```

Ein erfolgreicher Serialisierungsaufruf für exclusive Zugriffe blockiert anschließend den Zugriff für alle anderen virtuellen Maschinen.

Free Serialization on Shared Memory (INT 31, AX = 0D03H, V 1.0)

Diese Funktion gibt die Serialisierung des Shared Memory auf. Damit kann der Zugriff auf den Block freigegeben werden.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0D03H Free Serial. Block  º
º SI:DI Handle Shared Memory     º
º DX = Option Flags              º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

In SI:DI ist der bei der Allocierung des Shared Memory erzeugte Handle zu übergeben. In DX ist die Option für den Zugriff auf den Speicher zu definieren:

```
DX Bit 0 = 0: realease exclusive serilization
           1: release shared serialization
   Bit 1 = 0: don't free pending serialization
           1: free pending serialization
   Bit 2-15   reserviert (0)
```

Bei gesetztem Carry-Flag enthält AX einen Fehlercode:

```
AX = 8002H  Invalid state
     8023H  Invalid Handle
```

Get Coprozessor Status (INT 31, AX = 0E00H, V 1.0)

Diese Funktion ermittelt den Status des Coprozessors.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0E00H Get Coproz. Status  º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º DX = Status Flags              º
º CY = 1   Fehler                º
Û--------------------------------ì
```

Bei erfolgreichem Aufruf ist das Carry-Flag gelöscht und AX enthält ein Status Flag:

```
AX Statusflag

  Bit 0 : MPv Bit im MSW/CR0
          0 Coprozessor disabled for client
          1 Coprozessor enabled for client
  Bit 1 : EMv Bit im MSW/CR0
          0 Client is not emulating coprozessor
          1 Client is emulating coprozessor
  Bit 2 : MPr Bit im MSW/CR0
          0 Host is not emulating coprozessor
          1 Host is emulating coprozessor
  Bit 4-7: Coprozessor Typ
           00H = no coprocessor
           02H = 80287
           03H = 80387
           04H = 80486
  Bit 8-15: unbenutzt
```

In DPMI Version 0.9 wird beim Aufruf das Carry-Flag gesetzt, d.h. die Funktion ist nicht implementiert.

Ist das *Real-Mode-EM-Bit* gesetzt, unterstützt der Server *Fließkomma Arithmetik Emulationen*.

(INT 31, AX = 0E01H, V 1.0)

Diese Funktion ersetzt den Status der Coprozessor-Emulation.

```
Ö--------------------------------Ì
º         CALL INT 31            º
º                                º
º AX = 0E01H Set Emulation       º
º BX = Coprozessor Bits          º
û--------------------------------À
º         Return:                º
º CY = 0   ok                    º
º CY = 1   Fehler                º
º AX = Fehlercode                º
Û--------------------------------ì
```

Beim Aufruf wird in BX der Status für die Emulation übergeben:

```
BX Statusflag
  Bit 0 : MPv Bit im MSW/CR0
          0 Disable Coprozessor for client
          1 Enable Coprozessor  for client
  Bit 1 : EMv Bit im MSW/CR0
          0 Client is not supporting coprozessor emulation
          1 Client is supporting coprozessor emulation
  Bit 2-15: unbenutzt
```

Bei erfolgreichem Aufruf ist das Carry-Flag gelöscht. Ist das Carry-Flag gelöscht, enthält AX enthält einen Fehlercode:

```
AX = 8026H Invalid request
```

Damit möchte ich die Beschreibung des DPMI-Interfaces beenden, auch wenn einige Fragen offen geblieben sind.

13 Ergänzungen des BIOS-INT 10 durch EGA- und VGA-Adapter

Der BIOS-Interrupt 10 dient zur Zeichenausgabe an den Bildschirm. Dabei werden verschiedene Funktionen angeboten. Seit Einführung des Enhanced-Grafik-Adapters (EGA) ist jedoch eine höhere Bildschirmauflösung möglich, die durch das Original-BIOS nicht unterstützt wird. Deshalb enthalten die EGA-Karten eigene BIOS-ROMs, die beim Systemstart den BIOS-INT 10 auf eigene Routinen umleiten. Mittlerweile werden VGA-Karten (Video-Grafik-Adapter) eingesetzt, die ebenfalls ein eigenes BIOS besitzen.

13.1 Die BIOS-Ergänzungen der EGA-Karten

Die IBM-EGA-Karte wird in drei Ausbaustufen geliefert. Die Grundkarte enthält 64 Kbyte und läßt sich um 64 Kbyte und 128 Kbyte erweitern. Kompatible Karten werden in der Regel mit 256- Kbyte-Speicher ausgeliefert. An den Adapter lassen sich sowohl Monochrom- als auch Farbmonitore verschiedener Auflösung anschließen. Die EGA-Karte bietet folgende Videomodi:

Mode	Bemerkungen
320x200	bei Farbgrafik-Monitoren
640x200	bei Farbgrafik-Monitoren
720x350	bei Monochrom-Monitoren
640x350	bei Monochrom-Monitoren
640x350	bei hochauflösenden Farbgrafik-Monitoren

Tabelle 13.1: Videomodi der EGA-Karte

Dabei lassen sich die Zeichen beim Monochrom-Monitor in einer 14x9-Matrix ausgeben. Bei normalen Farbmonitoren wird eine 8x8-Matrix verwendet. Hochauflösende Farbmonitore erlauben wieder die 14x8-Zeichenmatrix. Im normalen Color-Mode sind 16 Farben erlaubt, während dies bei den erweiterten Modes bis auf 64 Farben reicht.

Im Grafikmodus belegt die EGA-Karte den Bereich ab B800:0000, sofern sie im CGA-Mode betrieben wird (320 x 200). Das gleiche gilt für den 640-x-200-Pixel-Mode. Bei den erweiterten Modes mit 16 Farben (Mode 0DH und 0EH) beginnt die Adresse bereits bei A000:0000H.

Die EGA-Karten besitzen ein eigenes BIOS, welches beim Systemstart aktiviert wird. Dieses BIOS belegt im BIOS-Datenbereich ab Adresse 0040:0087 zwei Adressen, um die Konfigurierung zu speichern. Hierbei gilt folgende Belegung:

Adresse	Bedeutung
0040:0087	Monitor-Status
0040:0088	Video-Mode

Tabelle 13.2: BIOS-Adressen der EGA-Karte

Im ersten Byte wird hinterlegt, welcher Monitor an der EGA-Karte angeschlossen ist. Der Status läßt sich durch die INT 10-Funktion 12H abfragen. Durch direkte Auswertung des Bytes ab Offset 0088H läßt sich aber direkt der Typ des Monitors ermitteln. Ein hochauflösender Farbmonitor erhält in diesem Byte die Kodierung F9H, während ein normaler Farbmonitor mit den Zeichen F8H markiert wird. Ein Monochrom-Monitor erhält den Wert FBH zugewiesen. Die Einstellung wird im wesentlichen durch die Stellung der DIP-Schalter auf der Karte beeinflußt.

Die Adressen 0040:00A8 bis 0040:00AB im BIOS-Datenbereich enthalten einen 4-Byte-Vektor mit der Adresse des EGA-ROM-Zeichensatzes. Durch Umdefinition dieses Zeigers läßt sich ein eigener Zeichensatz installieren.

Informationen zum Cursor befinden sich ab 0040:0060 (Scan-Zeile) bis 0040:0061 (End-cursor). Ab Adresse **0**040:0062 findet sich die Einstellung der aktiven Bildschirmseite (0-7). Die folgenden beiden Bytes (0040:0063 und 0040:0064) enthalten die Port-Adresse des aktiven Bildschirmadapters. Der Status des aktiven Grafikadapters wird ab 0040:0065 geführt, während in 0040:0066 die Palette-Information steht. Die Zahl der Zeilen pro Bildschirmseite und die Anzahl der Bytes pro Zeichen werden im BIOS-Datenbereich zwischen 0040:0084 bis 0040:0086 geführt. Dabei enthält das letzte Wort die Zahl der Byte pro Zeichen. Die Information über die Einstellung der DIP-Schalter, bzw. die Kodierung des Monitortyps, ist nur bei EGA-Karten auf der Adresse **0040:0087** vermerkt. Für beide Adapter finden sich ab Adresse **0040:0089** die Informationen über den aktiven Zeichensatz, die Paletteauswahl und den angeschlossenen Monitor. Es gilt dabei folgende Belegung:

Bit	Monitorstatusbyte (VGA)
7	reserviert
.	.
5	reserviert
4	Zeichensatz 0: 8x8-Zeichensatz
	1: 8x16-Zeichensatz
3	0: Laden der Default-Palette aktiv
2	0: Colormonitor 1: Monochrom-Monitor
1	Color Summing aktiv
0	reserviert

Tabelle 13.3: Monitorstatusbyte der EGA-Karte

Bei der VGA-Karte zeigt das Doppelwort an Adresse 0040:00A8 wieder auf den Zeichensatz.

Bei VGA-Karten ermittelt das BIOS die Hardwarekonfiguration und legt die Informationen im Equipment-Status-Byte Adresse 0040:0010 ab. Dieses Byte besitzt dann folgende Kodierung:

Bit	Belegung des Equipment-Status-Byte (VGA)
15-14	Zahl der parallelen Druckeradapter
13	reserviert
12	reserviert
11- 9	Zahl der seriellen Schnittstellen
8	reserviert
7-6	Anzahl Disketten-Laufwerke
5-4	Video-Mode
	00: --
	01: 40x25 Zeichen Color
	02: 80x25 Zeichen Color
	03: 80x25 Zeichen Monochrom
3	reserviert
2	1: Maus vorhanden
1	Arithmetikprozessor vorhanden
0	1: Systemstart von Diskette

Tabelle 13.4: Equipment-Status-Byte

Die Funktionen des EGA-BIOS-INT 10

Zusätzlich zu den beim INT 10H beschriebenen CGA-Funktionsaufrufen bietet die EGA-Karte weitere Funktionen.

EGA-Set Palette Register (AH = 10H)

Dieser Aufruf dient dazu, die internen Register der EGA-Karte mit Farbwerten zu initialisieren. Es sind verschiedene Modi möglich, die gemäß folgender Übergabekonvention angesprochen werden:

```
CALL   :  INT 10

AH: 10H (Set EGA Palette)
AL: Subcode
```

```
RETURN

---
```

Die Routine erwartet im Register AL einen Unterfunktionscode, der die jeweilige Aktion steuert. Insgesamt sind die Codes AL = 0 bis 3 belegt. Es gelten folgende Aufrufkonventionen:

Set Palette Register (AL = 0)

Mit diesem Aufrufcode wird das Register BL auf die Farbe BH gesetzt.

Set Boarder Color Register (AL = 1)

Mit diesem Aufruf wird das Overscan-Register auf die Farbe im Register BH gesetzt.

Set all Palette Register (AL = 2)

Im Registerpaar ES:DX wird die Adresse einer 17 Byte langen Tabelle übergeben. Die Einträge in dieser Tabelle beeinflussen das Overscan-Register und die Farbattribute. Die Bytes 0-15 enthalten den Code für die jeweilige Farbpalette, während in Byte 16 der Overscan-Code steht.

Video-Toggle Intensity/Blinking Bit (AL = 3)

Dieser Aufrufcode steuert die Anzeige im Intensitäts- und Blinkmode. Mit BL = 0 wird der Intensitätsmode umgeschaltet, während BL = 1 den Blinkmode umschaltet.

EGA-Text Mode Character Generator (AH = 11H)

Dieser Aufruf dient zur Initialisierung des internen EGA-Zeichensatzes.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 11H (Set Character Code)    º
º AL: Subcode                     º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Die Routine erwartet im Register AL einen Unterfunktionscode, der die jeweilige Aktion steuert. Insgesamt sind die Codes AL = 0 bis 30H belegt. Es gelten folgende Aufrufkonventionen:

Load user spezified Characterset (AL = 0)

Mit diesem Aufrufcode wird ein neuer Zeichensatz in einen der vier Speicherräume der EGA-Karte geladen. Der Inhalt des Registers BL selektiert dabei die Seite (0-3). Der Zeichencode befindet sich in einer Tabelle, deren Adresse im Registerpaar ES:BX übergeben wird. Im Register BH steht, wie viele Bytes pro Zeichen in der Tabelle gespeichert sind, während die Zahl der Zeichen in CX übergeben wird. DX enthält den Offset auf das erste Zeichen des Blocks.

Load ROM-Monochrom Characterset (AL = 1)

Mit diesem Aufruf wird der Monochrom (8x14 Pixel)-Zeichensatz aus dem ROM der EGA-Karte aktiviert. Die Speicherbank wird dabei durch den Inhalt von BL selektiert.

Load ROM 8x8 double dot Characterset (AL = 2)

Mit diesem Aufruf läßt sich ein Monochrom (8x8 Pixel)-Zeichensatz aus dem ROM aktivieren, wobei die Speicherbank dabei durch den Inhalt von BL selektiert wird.

Set block spezifier (AL = 3)

Durch diese Unterfunktion läßt sich ein Zeichensatz wählen, wobei der Inhalt des Registers BL die aktive Seite bestimmt. Durch die Codes AL = 0 .. 2 lassen sich ja verschiedene Zeichensätze in die einzelnen Blöcke laden. Für den Inhalt von BL gilt:

```
Ö--------Ú--------------------Ì
º Bit 3  º Blocknummer        º
û--------é--------------------À
º   0    º Bit 0 und Bit 1    º
º   1    º Bit 2 und Bit 3    º
Û--------Ù--------------------ì
```

Das Bit 3 spezifiziert dabei, welche Bildschirmseite (0 .. 3) selektiert werden soll.

Set User 8x8 Graphic Characterset (AL = 20H)

Mit diesem Aufruf läßt sich ein Monochrom (8x8 Pixel)-Zeichensatz aktivieren. Es wird dabei der unter der INT 1F-Adresse abgespeicherte Zeichensatz geladen. Die Adresse wird per Registerpaar ES:BX übergeben.

Set User Graphic Characterset (AL = 21H)

Mit diesem Aufrufcode wird ein neuer Zeichensatz in einen der vier Speicherräume der EGA-Karte geladen. Der Zeichencode befindet sich in einer RAM-Tabelle, deren Adresse im Registerpaar ES:BX übergeben wird. Im Register CX steht, wie viele Byte pro Zeichen in der Tabelle gespeichert sind. Im Register BL wird die Zahl der Zeilen pro Seite spezifiziert:

```
Ö--------Ú--------------------Ì
º  BL    º Zeilen / Seite     º
û--------é--------------------À
º   1    º     14             º
º   2    º     25             º
º   3    º     43             º
Û--------Ù--------------------ì
```

Tabelle 13.5: Zeilen pro Seite

Set ROM 8x14 Characterset (AL = 22H)

Mit diesem Aufrufcode wird der 8x14-Pixel-Zeichensatz aus dem internen ROM geladen. Das Register BL spezifiziert dabei die Zeilenzahl pro Bildschirmseite. Es gilt die gleiche Nomenklatur wie bei der Funktion 22H.

Set ROM 8x8 double dot Characterset (AL = 23H)

Mit diesem Aufrufcode wird der 8x8-Pixel-Zeichensatz aus dem internen ROM geladen. Das Register BL spezifiziert dabei die Zeilenzahl pro Bildschirmseite. Es gilt die gleiche Nomenklatur wie bei der Funktion 22H.

Get Font Info (AL = 30H)

Diese Unterfunktion erlaubt es, den Status der EGA-Karte zu lesen. Das Register BH selektiert dabei den Typ des Zeichengenerators, dessen Adresse zurückzugeben ist.

BH	Bemerkung
0	INT 1F-Vektors
1	INT 44-Vektors
2	8x14-ROM-Zeichensatz
3	8x8-ROM-Zeichensatz
4	8x8 ROM Zeichensatz obere Hälfte
5	8x14 ROM Zeichensatz obere Hälfte

Tabelle 13.6: Zeichenfonts bei der EGA-Karte

Nach dem Aufruf finden sich die Statusinformationen in folgenden Registern:

Register	Information
ES:BP	Adresse des Zeichengenerators
CX	Zahl der Bytes pro Zeichen
DL	Zeilenzahl-1

EGA-Video alternate Function Select (AH = 12H)

Mit diesem Aufruf läßt sich die Konfiguration der EGA-Karte lesen sowie die Print-Screen-Einstellung verändern.

CALL : INT 10	
AH: 12 (EGA-Konfiguration)	
BL: Subcode	
RETURN	

Die Routine erwartet im Register BL einen Unterfunktionscode, der folgende Bedeutung besitzt:

Alternate Function select (BL = 10H)

Die Konfiguration der EGA-Karte wird gelesen. Die Informationen finden sich nach dem Aufruf in folgenden Registern:

Register	Information
BH	0 Farb-Modus aktiv 1 Monochrom-Modus
BL	Speichergröße 00 : 64 K 01 : 128 K 10 : 196 K 11 : 256 K

Select alternate Print Screen Routine (BL = 20H)

Mit diesem Aufruf wird die BIOS-Print-Screen-Funktion auf 43 Zeilen pro Seite umgestellt.

13.2 Die Funktionen des VGA-BIOS-INT 10

Die VGA-Karten erweitern die Funktionen des INT 10 erheblich. Nachfolgend werden diese Funktionen vorgestellt.

Set Video Mode (AH = 00H)

Dieser Aufruf dient dazu, den internen Videomode der VGA-Karte zu initialisieren. Es sind verschiedene Modi möglich, die gemäß folgender Übergabekonvention angesprochen werden:

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 00H (Set Video Mode)        º
º AL: Videomode                   º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Die Routine erwartet im Register AL den Code für den Videomode. Gegenüber der CGA-Karte (siehe Tabelle 2.1) wurden drei weitere Modes bei der IBM-VGA-Karte eingeführt:

```
Ö------Ú-------------------------------------Ì
º  AL  º Bedeutung                           º
û------é-------------------------------------À
º  11H º 640x480 Bildpunkte mit 2 Farben     º
º  12H º 640x480 Bildpunkte mit 16 Farben    º
º  13H º 320x200 Bildpunkte mit 256 Farben   º
Û------Ù-------------------------------------ì
```

Tabelle 13.7: Neue Videomodi der VGA-Karte

Mittlerweile existiert jedoch eine Vielzahl von VGA-Karten, die den INT 10 mit der Funktion AH = DOH benutzt. Nachfolgende Tabelle enthält eine Aufstellung der verschiedenen Modes.

AL	Text Mode	Pixel resul.	Graphic	Color	Pages	Adr.	Karte
00	40x25	8x4 8x14	---	B&W	8	B800	CGA ATI VIP
01	40x25	8x8 8x14	---	16	8	B800	CGA ATI VIP
02	80x25	8x8 8x8 8x14	---	B&W	4 8 8	B800	CGA EGA, VGA, MCGA ATI VIP
03	80x25	8x8 8x8	---	16	4 8	B800	CGA EGA, VGA, MCGA
04	40x25	8x8	320x200	4	1	B800	CGA
05	40x25	8x8	320x200	4 Grau	1	B800	CGA
06	80x25	8x8	640x200	B&W	1	B800	CGA
07	80x25	9x14	---	B&W	1 8	B000	MDA, Herkules EGA, VGA ATI VIP
08	20x25	8x8	160x200	16		B800	PCjr, Tandy 1000
09	40x25	8x8	320x200	16		B800	PCjr, Tandy 1000
0A	80x25	8x8	640x200	4		B800	PCjr, Tandy 1000
0B	reserviert (intern durch das EGA-BIOS belegt)						
0C	reserviert (intern durch das EGA-BIOS belegt)						
0D	40x25	8x8	320x200	16	8	A000	EGA, VGA
0E	80x25	8x8	640x200	16	4	A000	EGA, VGA
0F	80x25	8x14	640x350	B&W	2	A000	EGA, VGA
10	80x25	8x14	640x350	4/16	2	A000	EGA, VGA
11	80x30	8x16	640x480	B&W		A000	VGA,MCGA,ATI EGA ATI VIP
12	80x30	8x16	640x480	16 16/64		A000	VGA, ATI VIP ATI EGA Wonder
13	40x25	8x8	320x200	256		A000	VGA,MCGA,ATI VIP
14	80x25	8x8	640x200 640x400	 16		A000	LavaChromeII EGA Tecmar VGA/AD
15	80x25	8x14	640x350				LavaChromeII EGA
16	80x25	8x14	640x350 800x600	 16			LavaChromeII EGA Tecmar VGA/AD
17	80x34 132x25	8x14					LavaChromeII EGA Tecmar VGA/AD

AL	Text Mode	Pixel	Graphic resul.	Color	Pages	Adr.	Karte
18	132x44	8x8	640x480	B&W			Tseng Labs EVA
	80x34	8x14	640x480				LavaChromeII EGA
			1024x768	16			Tecmar VGA/AD
19	132x25	8x14		B&W			Tseng Labs EVA
1A	120x28	8x13	640x350	B&W			Tseng Labs EVA
			640x350	256			Tecmar VGA/AD
1B			640x400	256			Tecmar VGA/AD
1C			640x480	256			Tecmar VGA/AD
1D			800x600	256			Tecmar VGA/AD
22	132x44	8x8					Tseng Labs EVA
							Ahead EGA 2001
	132x43						Peacock (VGA)
23	132x25	6x14					Tseng Labs EVA
	132x25	8x14					Ahead EGA 2001
	132x25	8x8		16		B800	ATI EGA Wonder
	132x28						Peacock (VGA)
24	132x28	6x13					Tseng Labs EVA
	132x25						Peacock (VGA)
25	80x60	8x8	640x480				Tseng Labs EVA
			640x480	16			Vega VGA
26	80x60	8x8					Tseng Labs EVA
	80x60	8x8	640x480				Ahead EGA 2001
	80x60						Peacock (VGA)
27			720x512	16			Vega VGA
	132x25	8x8		B&W		B800	ATI EGA Wonder
29			800x600	16			Vega VGA
			800x600	16			Peacock (VGA)
2A	100x40						Peacock VGA
2D			640x350	256			Vega VGA
2E			640x480	256			Vega VGA
2F			720x512	256			Vega VGA
30			800x600	256			Vega VGA
33	132x44	8x8		16		B800	ATI EGA Wonder
36			960x720	16			Vega VGA
37			1024x768	16			Vega VGA
	132x44	8x8		B&W		B800	ATI EGA Wonder

AL	Text Mode	Pixel	Graphic resul.	Color	Pages	Adr.	Karte
40	80x25	8x16	640x400	2	1	B800	AT&T 6300,VDC600
	80x25	8x16	640x400	2	1	B800	Compac Portable
	80x43						Vega VGA, Tecmar
	80x43						Video7 VGA
	80x43						Tatung VGA
41			640x200	16	1		AT&T 6300
	132x25						Vega VGA
	132x25						Video7 VGA
	132x25						Tatung VGA
42	80x25	8x16	640x400	16			AT&T 6300,VDC600
	132x43						Vega VGA
	132x43						Video7 VGA
	132x43						Tatung VGA
43	80x60						Vega VGA
	80x60						Video7 VGA
	80x60						Tatung VGA
44	100x60						Vega VGA
	100x60						Video7 VGA
	100x60						Tatung VGA
45	132x28						Tatung VGA
	132x28						Video7 VGA
46	100x40	8x15	800x600	2			AT&T VDC600
47	100x37	8x16	800x600	16			AT&T VDC600
48	80x50	8x8	640x400	2		B800	AT&T 6300,VDC600
49	80x30	8x16	640x480				Lava Chrome II
4D	120x25						Vega VGA
4E	120x43						Vega VGA
4F	132x25						Vega VGA
50	132x25	9x14		B&W			Ahead EGA 2001
	80x30	8x16	640x480	16			Paradise EGA-480
	80x43	8x16	640x480	B&W			Vega VGA
			640x480				Taxan 565 EGA
	80x34						Lava Chrome II
51	80x30	8x16		B&W			Paradise EGA-480
	80x30	8x14	640x480	16			Lave Chrome II
	80x34	8x14	640x480	16			ATI EGA Wonder
	132x25			B&W			Vega VGA
	80x34						Lava Chrome II
52	132x44	9x8		B&W			Ahead EGA 2001
	132x43			B&W			Vega VGA
	94x29	8x14	752x410	16			ATI EGA Wonder
	80x60						Lava Chrome II
53	100x40	8x14	800x560	16			ATI EGA Wonder
	132x43						Lava Chrome II

AL	Text Mode	Pixel	Graphic resul.	Color	Pages	Adr.	Karte
54	132x43	8x8					Paradise EGA-480
	132x43	8x9		16		B800	Paradise VGA
	132x43						Taxan 565 EGA
	132x43						AST VGA Plus
	132x43						HP D1180A
	132x43	7x9		16			AT&T VDC600
	132x25						Lava Chrome II
	100x42	8x14	800x600	16		A000	ATI EGA/VGA
55	132x25	8x14					Paradise EGA-480
	132x25	7x16		16		B800	Paradise VGA
	132x25	8x16		16		B800	Paradise VGA
	132x25						Taxan 565 EGA
	132x25						AST VGA Plus
	132x25						HP D1180A
	132x25	7x16		16			AT&T VDC600
	94x29	8x14	752x410				Lava Chrome II
	80x66	8x8		16		A000	ATI VIP
56	132x43	7x9		4		B800	Paradise VGA
	132x43	8x9		4		B800	Paradise VGA
	132x43			B&W			Taxan 565 EGA
	132x43	7x9		2			AT&T VDC600
	132x25	7x16		4		B800	Paradise VGA
	132x25	8x16		4		B800	Paradise VGA
	132x25			B&W			Taxan 565 EGA
58	100x75	8x8	800x600	16		A000	Paradise VGA
	100x75	8x8	800x600	16			AT&T VDC600
	80x33	8x14		16		B800	ATI EGA/VIP
			800x600	16			AST VGA Plus
			800x600	16			HP D1180A
59	100x75	8x8	800x600	2		A000	Paradise VGA
	100x75	8x8	800x600	2			AT&T VDC600
	80x66	8x8		16		A000	ATI VIP
			800x600	2			AST VGA Plus
			800x600	2			HP D1180A
5E			640x400	256		A000	Paradise VGA
			640x400	256		A000	Vega VGA
			640x400	256			AST VGA Plus
	80x25	8x16	640x400	256			AT&T VDC600
5F			640x480	256		A000	Paradise VGA
			640x480	256			AST VGA Plus
			640x480	256			HP D1180A
	80x30	8x16	640x480	256			AT&T VDC600
60			752x410				Vega VGA
			752x410	16			Tatung VGA
			752x410	16			Video7 VGA
61			720x540				Vega VGA
			720x540	16			Tatung VGA
			720x540	16			Video7 VGA
			640x400	256		A000	ATI VGA Wonder

AL	Text Mode	Pixel	Graphic resul.	Color	Pages	Adr.	Karte
62			800x600				Vega VGA
			800x600	16			Tatung VGA
			800x600	16			Video7 VGA
			640x480	256		A000	ATI VGA Wonder
63			1024x768	2			Video7 VGA
			800x600	256		A000	ATI VGA Wonder
64			1024x768	4			Video7 VGA
65			1024x768	16			Video7 VGA
			1024x768	16		A000	ATI VGA Wonder
66			640x400	256			Tatung VGA
			640x400	256		A000	Video7 VGA
67			640x480	256			Video7 VGA
			1024x768	4		A000	ATI VGA Wonder
69			720x540	256			Video7 VGA
74			640x400	2		B800	Toshiba 3100

Tabelle 13.8: Videomodi verschiedener VGA-Karten

Set Cursor Size (AH = 01H)

Dieser Aufruf dient dazu, die Größe des sichtbaren Cursors im Textmode zu definieren. Die Form ist abhängig von der Zahl der Linien und wird durch das Register CX spezifiziert. Der Cursor kann dabei maximal die Größe eines Zeichenfeldes einnehmen.

```
CALL:  INT 10

AH: 01H (Cursor Size)
CH: Bit 0-4 Startzeile Cursor
CL: Bit 0-4 Endzeile Cursor

RETURN

--
```

Bei VGA-Karten werden die Bits 5-6 im Register CH für das Blink Attribut (00=normal, 01=unsichtbar, 10=langsam, 11=schnell) benutzt. Beim Coloradapter lagen die Werte zwischen 0 und 7, während sie beim Monochromadapter zwischen 0 und 13 variieren dürfen. Der VGA-Adapter erlaubte einen Bereich zwischen 0 und 32 Pixeln.

Set Cursor Position (AH = 02H)

Der Cursor läßt sich mit dieser Funktion auf dem Bildschirm positionieren. Die Koordinaten werden durch das DX-Register angegeben. Dabei wurde bei der VGA-Karte die Position an die Zahl der Zeilen pro Seite angepaßt.

```
Ö--------------------------------Ì
º      CALL:  INT 10             º
º                                º
º AH: 02H (Set Cursor)           º
º BH: Bildschirmseite            º
º DH: Zeilennummer               º
º DL: Spaltennummer              º
û--------------------------------À
º      RETURN                    º
º --                             º
Û--------------------------------ì
```

Die Position (0,0) befindet sich in der linken oberen Bildschirmecke. Durch den Inhalt des BH-Registers wird die Bildschirmseite definiert, für die die neue Cursorposition gilt. Bei Anwahl des Grafikmodus muß der Wert von BH = 0 gesetzt werden.

Get Cursor Position (AH = 03H)

```
Ö---------------------------------Ì
º      CALL : INT 10              º
º                                 º
º AH:  03H (Get Cursor)           º
º BH:  Bildschirmseite            º
û---------------------------------À
º      RETURN                     º
º DH:  Zeilennummer               º
º DL:  Spaltennummer              º
º CH:  Startzeile Cursor          º
º CL:  Endzeile Cursor            º
Û---------------------------------ì
```

Die Position des Cursor innerhalb einer Bildschirmseite läßt sich mit der Unterfunktion AH = 3 abfragen. Das Register BH spezifiziert dabei die Bildschirmseite (bei Grafik = 0). Die Funktion liefert als Ergebnis Lage und Größe des Cursors zurück. Die Cursorgröße findet sich analog zur Funktion AH = 1 im CX-Register.

Position Light Pen (AH = 04H)

Diese BIOS-Funktion entfällt bei der VGA-Karte.

Select Display Page (AH = 05H)

```
Ö---------------------------------Ì
º                                 º
º   CALL:  INT 10                 º
º                                 º
º AH:  05H (Set Page)             º
º AL:  Bildschirmseite            º
û---------------------------------À
º      RETURN                     º
º  --                             º
Û---------------------------------ì
```

Befindet sich der Bildschirmadapter im Textmodus (siehe Funktion AH = 00H), läßt sich mit dieser Funktion die aktuelle Bildschirmseite umschalten. Der Wert des Registers AL bestimmt die Bildschirmseite.

Page Scroll Up (AH = 06H)

Innerhalb der aktiven Bildschirmseite läßt sich im Textmode ein Fenster einblenden, in dem mit dieser Funktion der Text nach oben gescrollt wird. Dabei läßt sich der Ausschnitt

auf den kompletten Bildschirm vergrößern, um die Scrollfunktion auf die gesamte Seite auszudehnen. Im unteren Fensterbereich werden beim Scroll Leerzeilen eingefügt. Es sind folgende Register zu definieren:

```
Ö---------------------------------Ì
°      CALL : INT 10              °
°                                 °
° AH: 06H (Scroll Up)             °
° AL: Anzahl der Zeilen, um die   °
°     nach oben geschoben wird    °
° CH: Eckzeile oben links         °
° CL: Eckspalte oben links        °
° DH: Eckzeile unten rechts       °
° DL: Eckspalte unten rechts      °
° BH: Attribut der Leerzeile      °
û---------------------------------À
°      RETURN                     °
°  --                             °
Û---------------------------------ì
```

Mit den Werten in den Registern CX, DX werden die beiden Eckpunkte des ausgewählten Bildschirmfensters spezifiziert. Die Koordinaten (0,0),(24,79) setzen das Fenster über den gesamten Bildschirm. Das AL-Register gibt an, wie viele Zeilen nach oben gescrollt werden. Ist der Wert = 0, dann wird das Fenster gelöscht.

Der gleiche Effekt tritt auf, wenn der Wert in AL der Zeilenzahl innerhalb des Fensters entspricht, da ja von unten Leerzeilen eingefügt werden. Mit dem Wert des Registers BH wird das Attribut der Leerzeile (blinkend, unterstrichen, invers, etc.) festgelegt. Die Kodierung der Attribute wird bei der Unterfunktion AH = 8 vorgestellt.

Page Scroll Down (AH = 07H)

```
Ö---------------------------------Ì
°      CALL : INT 10              °
°                                 °
° AH: 07H (Scroll Down)           °
° AL: Anzahl der Zeilen, um die   °
°     nach unten geschoben wird   °
° CH: Eckzeile Ecke oben links    °
° CL: Eckspalte Ecke oben links   °
° DH: Eckzeile Ecke unten rechts  °
° DL: Eckspalte Ecke unten rechts °
° BH: Attribut der Leerzeile      °
û---------------------------------À
°      RETURN                     °
°  --                             °
Û---------------------------------ì
```

Analog der Unterfunktion (AH = 6) existiert die Möglichkeit, einen Bildschirmausschnitt um mehrere Zeilen nach unten zu scrollen. Am oberen Fensterrand werden Leerzeilen eingefügt.

Es gelten die gleichen Bedingungen wie bei der Unterfunktion (AH = 6).

Read Attributes and Character at Cursor Position (AH = 08H)

```
Ö---------------------------------Ì
º      CALL : INT 10              º
º                                 º
º AH: 08H (Get Char Attribut)     º
º BH: Bildschirmseite (Textmode)  º
û---------------------------------À
º      RETURN                     º
º AL: gelesenes Zeichen           º
º AH: Attributbyte (Textmode)     º
Û---------------------------------ì
```

Im Textmode läßt sich das an der aktuellen Cursorposition abgespeicherte Zeichen sowie das zugehörige Attributbyte lesen. Der Inhalt des BH-Registers spezifiziert dabei die Bildschirmseite. Das Ergebnis wird im AX-Register zurückgegeben.

Write Attributes and Characters at Cursor Position (AH = 09H)

Diese Funktion erlaubt die Ausgabe von Zeichen in den Bildspeicher, wobei jedem Zeichen ein Attributbyte angehängt wird. Es gelten folgende Registerbelegungen:

```
Ö---------------------------------Ì
º      CALL :  INT 10             º
º                                 º
º AH:  09H (Set Char & Attribut)  º
º AL:  Zeichencode                º
º BH:  Bildschirmseite der Aus-   º
º      gabe (nur im Textmode)     º
º BL:  Attributbyte des Zeichens  º
º CX:  Zahl der Kopien            º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

In BL steht das Attribut des Zeichens (Textmodus) oder die Farbe (Grafikmodus). Falls im Grafikmodus Bit 7 = 1 ist, werden die Zeichen mit XOR in das Bild eingeblendet. Die Funktion gibt alle Zeichen, inclusive **CR**, **LF** und **BS** auf dem Bildschirm als Zeichen aus. Es ergeben sich hier keine Änderungen gegenüber den CGA-Funktionen.

Write Character at Cursor Position (AH = 0AH)

Für die Fälle, wo lediglich Zeichen ohne Attributbyte ausgegeben werden müssen, läßt sich diese Funktion benutzen. Es gilt folgende Registerbelegung:

```
Ö---------------------------------Ì
º      CALL :  INT 10             º
º                                 º
º AH:  0AH (Write Char)           º
º AL:  Zeichencode                º
º BH:  Bildschirmseite der Aus-   º
º      gabe (nur im Textmode)     º
º CX:  Zahl der Kopien            º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Das im Bildschirmspeicher gesetzte Attributbyte bleibt beim Schreibvorgang erhalten. Schreib-Lese-Vorgänge im Grafikmodus lassen sich ebenfalls über die gerade vorgestellten Funktionen abwickeln. Die Zeichengenerierung erfolgt aber aus dem BIOS-ROM. Allerdings lassen sich so nur die ersten 128 ASCII-Zeichen ausgeben.

Sollen Sonderzeichen angezeigt werden, ist der Grafikvektor des INT 1F auf eine Tabelle mit den entsprechenden Bitmustern umzulenken. Die Funktion gibt auch die Steuerzeichen CR, LF und BS mit aus.

Set Color Palette (AH = 0BH)

Im Grafikmodus lassen sich Farben definieren. Für den Zugriff auf die neuen Farbpaletten existiert die Funktion 10H.

```
Ö---------------------------------Ì
º      CALL :  INT 10             º
º                                 º
º AH:  0BH (Select Palette)       º
º BH:  Code der zu setzenden      º
º      Farbpalette                º
º BL:  Farbwert (0 - 15) für      º
º      diese Palette              º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Die Funktion kann nur im 320x200-Videomode benutzt werden. Bei den Grafikmodi 04H und 05H (siehe Funktion AH = 00) sind folgende Auswahlcodes für den Farbwert in BL möglich:

```
BH : Auswahl der Farbpalette (0-127)
BL : Auswahl des Farbwertes aus der Palette
```

Mit BH = 01H läßt sich in BL die Palette 0-3 wählen. Im Textmode (AL = 00 und 01) legt der Farbwert in BL die Hintergrundfarbe (Boarder) im Bereich zwischen 0-31 fest, sofern BH=00 gesetzt wurde.

Set Graphik Pixel (AH = 0CH)

Im Grafikmodus lassen sich einzelne Punkte mit einer bestimmten Farbe setzen. Die Koordinaten dieses Punktes werden in der Notation (Zeile, Spalte) angegeben. Dabei wurde die Funktion für EGA- und VGA-Karten erheblich erweitert.

```
Ö---------------------------------Ì
º      CALL :  INT 10             º
º                                 º
º AH:  0CH  (Set Pixel)           º
º BH:  Seitennummer               º
º DX:  Zeile                      º
º CX:  Spalte                     º
º AL:  Bit 0 - 6 : Farbe          º
º      Bit 7 = 1 Verknüpfung des  º
º      Farbwertes über  OR mit    º
º      dem Punkt im RAM           º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Interessant ist Bit 7 im AL-Register. Die aktuell auszugebende Information an dem Punkt wird der bestehenden Farbe überlagert. Dies gilt jedoch nicht für den Videomode 13H.

Get Graphik Pixel (AH = 0DH)

Im Grafikmodus lassen sich einzelne Punkte mit dieser Funktion lesen.

```
Ö---------------------------------Ì
º      CALL : INT 10              º
º                                 º
º AH:  0DH  (Get Pixel)           º
º BH:  Seitennummer               º
º DX:  Zeile  (0-479)             º
º CX:  Spalte (0-639)             º
û---------------------------------À
º      RETURN                     º
º AL:  Farbwert des Punktes       º
Û---------------------------------ì
```

Der Farbwert wird im AL-Register zurückgegeben.

Write Character (AH = 0EH)

Mit dieser Funktion wird ein Teletype-Interface für die Bildschirmausgabe simuliert. Pro ausgegebenem Zeichen wird der Cursor um eine Stelle weiterbewegt. Die Funktion erwartet folgende Eingaben:

```
Ö---------------------------------Ì
º      CALL  : INT 10             º
º                                 º
º AH:  0EH  (Write Char)          º
º AL:  Zeichen                    º
º BH:  Seitennummer (Textmode)    º
º BL:  Vordergrundfarbe im        º
º      Grafikmode                 º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Der Bildschirmmode (25x40 oder 25x80 Zeichen) muß separat über die Funktion AH = 0 gesetzt werden. Die Zeichen **07H** (Bell), **08H** (BS), **0AH** (LF) und **0DH** (CR) werden interpretiert und ausgeführt.

Get current Video Mode (AH = 0FH)

Als Komplement zur Funktion AH = 0 (setze Bildschirm-Mode) läßt sich hier der Mode wieder auslesen. Die Funktion gibt folgende Werte zurück.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  0FH  (Read Status)         º
û---------------------------------À
º      RETURN                     º
º AL:  Bildschirmmode             º
º AH:  Bildschirmspalten (40 /80) º
º BH:  aktuelle Bildschirmseite   º
Û---------------------------------ì
```

Die Kodierung des Bildschirmmodes wurde bereits bei der Funktion AH = 0 vorgestellt. Bei der Abfrage kann auch der Mode 7 (interne 80x25-Monochromdarstellung) auftreten.

Set Palette Register (AH = 10H)

Dieser Aufruf wurde ab der EGA-Karte neu implementiert und dient zur Initialisierung der internen Register der EGA/VGA-Karte mit Farbwerten. Es sind verschiedene Modi möglich, die gemäß folgender Übergabekonvention angesprochen werden:

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set EGA Palette)       º
º AL: Subcode                     º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Die Routine erwartet im Register AL einen Unterfunktionscode, der die jeweilige Aktion steuert. Bei der EGA-Karte sind die Codes AL = 00 bis 03 belegt. Es gelten folgende Aufrufkonventionen:

Set Palette Register (AL = 00H)

Mit diesem Aufrufcode wird das Register BL auf die Farbe BH gesetzt.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 00H                         º
º BH: Farbwert          (0..63)   º
º BL: Palettenregister  (0..15)   º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Beim MCGA-Adapter ist für BX nur der Wert 0712H erlaubt. Die Funktion aktiviert defaultmäßig den 64-Register-Mode.

Set Overscan-Register (AL = 01H)

Mit diesem Aufruf wird das Overscan-Register auf die Farbe im Register BH gesetzt.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 01H                         º
º BH: Farbwert  (0..255)          º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Der Wert in BH bestimmt dann die Farbe des Bildrandes.

Set all Palette/Overscan-Register (AL = 02H)

Im Registerpaar ES:DX wird die Adresse einer 17 Byte langen Tabelle übergeben. Die Einträge in dieser Tabelle beeinflussen das Overscan-Register und die Farbattribute.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 02H                         º
º ES:DX Adresse Tabelle           º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Die Bytes 0-15 in der Tabelle enthalten den Code für die jeweilige Farbpalette, während in Byte 16 der Overscan-Code (Boarder Color) steht.

Toggle Intensity/Blink (AL = 03H)

Dieser Aufrufcode steuert die Anzeige im Intensitäts- und Blinkmode.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 03H                         º
º BL: Mode                        º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Mit BL = 0 wird auf den Intensitätsmode umgeschaltet, während BL = 1 den Blinkmode einschaltet.

Die Unterfunktionscodes 04H bis 06H sind reserviert.

Get individual Palette-Register (AL = 07H)

Es wird die Nummer des zu lesenden Palette-Registers in BL übergeben.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 07H                         º
º BL: Palette-Register (0-15)     º
û---------------------------------À
º        RETURN                   º
º BH: Inhalt Palette-Register     º
Û---------------------------------ì
```

Nach dem Aufruf enthält das Register BH den Inhalt des Palette-Registers. Die Funktion ist nur bei VGA-Karten implementiert.

Get Overscan-Register (AL = 08H)

Es wird das Overscan-Register mit der Farbe des Bildschirmrandes gelesen.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 08H                         º
û---------------------------------À
º        RETURN                   º
º BH: Wert Overscan-Register      º
Û---------------------------------ì
```

Nach dem Aufruf enthält das Register BH den Inhalt des Overscan-Registers. Die Funktion ist nur bei VGA-Karten implementiert.

Read all Palette/Overscan-Registers (AL = 09H)

Es werden alle Palette- und Overscan-Register gelesen.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 09H                         º
º ES:BX Zeiger auf Ergebnistabelleº
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Im Register ES:BX ist die Anfangsadresse einer 17 Byte langen Tabelle zu übergeben. In dieser Tabelle werden die Ergebnisse der Operation zurückgegeben (siehe AX=1002H). Die Bytes 0-15 enthalten die Palettewerte, während in Byte 16 der Overscanwert steht. Die Funktion ist nur bei VGA-Karten implementiert.

Set individual DAC-Register (AL = 10H)

Es wird das zu setzende Palette-Register in BX übergeben.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 10H                         º
º BX: Farbregister                º
º CH: Wert für Grün   (0..63)     º
º CL: Wert für Blau   (0..63)     º
º DH: Wert für Rot    (0..63)     º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Die Werte für die Farbanteile werden mit 6 Bit (Bit 0-5) dargestellt. Die Funktionen sind bei EGA-, VGA- und MCGA-Karten implementiert.

Die Unterfunktion 11H ist reserviert.

Set Block of DAC-Register (AL = 12H)

Es wird ein ganzer Block von Farbregistern gesetzt.

```
Ö--------------------------------Ì
º        CALL   :  INT 10        º
º                                º
º AH: 10H (Set Palette)          º
º AL: 12H                        º
º BX: Erstes Farbregister        º
º CX: Anzahl der Farbregister    º
º ES:BX Adresse Farbtabelle      º
û--------------------------------À
º        RETURN                  º
º ---                            º
Û--------------------------------ì
```

Im Register BX ist die Nummer des ersten Farbregisters zu übergeben. In CX steht die Zahl der zu modifizierenden Farbregister. In ES:BX ist ein Zeiger auf eine Tabelle mit den Definitionen für diese Farbregister zu übergeben. In der Tabelle ist für jedes Farbregister ein 3-Byte-Eintrag für Rot, Grün, Blau zu definieren. Die Funktion ist bei EGA-, VGA- und MCGA-Karten implementiert.

Select Video DAC Color Page (AL = 13H)

Die Funktion selektiert die entsprechende Farbseite.

```
Ö--------------------------------Ì
º        CALL   :  INT 10        º
º                                º
º AH: 10H (Set Palette)          º
º AL: 13H                        º
º BL: 00H Auswahl Paging Modus   º
º     01H Auswahl der Seite      º
º BH: Code                       º
û--------------------------------À
º        RETURN                  º
º ---                            º
Û--------------------------------ì
```

Die 256 Farbregister des DAC lassen sich über BL in mehrere Seiten aufteilen. Wird BL = 00H gesetzt, selektiert die Funktion den Paging Mode:

```
Ö-----Ú----------------------------------Ì
º BH  º Paging Mode                       º
û-----é----------------------------------À
º 00H º 4  Blöcke mit je 64 Registern     º
º 01H º 16 Blöcke mit je 16 Registern     º
Û-----------------------------------------ì
```

Mit BL = 01H erfolgt die Auswahl einer Farbseite. Deren Code ist in BH zu übergeben. Es gilt folgende Belegung:

```
Ö-----Ú----------------------------------Ì
º BH  º Seitennummer                      º
û-----é----------------------------------À
º 00H º  1. Block mit je 64 Registern     º
º 01H º  2. Block mit je 64 Registern     º
º 02H º  3. Block mit je 64 Registern     º
º 03H º  4. Block mit je 64 Registern     º
û-----é----------------------------------À
º 00H º  1. Block mit je 16 Registern     º
º 01H º  2. Block mit je 16 Registern     º
º  .  º ..  Bloc  mit je 16 Registern     º
º 0FH º 16. Block mit je 16 Registern     º
Û-----------------------------------------ì
```

Es lassen sich entweder 4 Blöcke mit 64 Registern oder 16 Blöcke mit 16 Registern selektieren. Die Funktion existiert nur bei VGA-Karten und ist im Videomode 13H ungültig.

Die Unterfunktion 14H ist reserviert.

Read individual DAC-Register (AL = 15H)

Es wird das spezifizierte Farbregister des DAC gelesen.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 15H                         º
º BX: Nummer Farbregister (0..255)º
û---------------------------------À
º        RETURN                   º
º CH: Farbwert Grün               º
º CL: Farbwert Blau               º
º DH: Farbwert Rot                º
Û---------------------------------ì
```

Die Funktion ist bei EGA-, VGA- und MCGA-Karten implementiert.

Die Unterfunktion 16H ist reserviert.

Read Block of DAC-Registers (AL = 17H)

Es wird ein kompletter Block von Farbregistern gelesen.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 17H                         º
º BX: erstes zu lesendes Farb-    º
º     register                    º
º CX: Zahl der zu lesenden Farb-  º
º     register                    º
º ES:DX Adresse Farbtabelle       º
û---------------------------------À
º        RETURN                   º
º CX: Zahl der gelesenen Farb-    º
º     register                    º
Û---------------------------------ì
```

In ES:DX wird die Adresse der Farbtabelle zurückgegeben, die die Farbdaten in der Folge: Rot, Grün, Blau für jedes Farbregister enthält. Die Zahl der gelesenen Farbregister steht in CX. Die Funktion ist bei EGA-, VGA- und MCGA-Karten implementiert.

Set PEL-Mask (AL = 18H, undokumentiert)

Diese undokumentierte Funktion ist bei EGA-, VGA- und MCGA-Karten implementiert und besitzt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 18H (PEL Mask)              º
º BL: neuer PEL-Wert              º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Get PEL-Mask (AL = 19H, undokumentiert)

Diese undokumentierte Funktion ist bei EGA-, VGA- und MCGA-Karten implementiert und besitzt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 19H (PEL Mask)              º
û---------------------------------À
º        RETURN                   º
º BL: PEL-Wert                    º
Û---------------------------------ì
```

Get Video DAC Color Page State (AL = 1AH)

Es wird der Status einer Farbseite abgefragt.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 1AH                         º
û---------------------------------À
º        RETURN                   º
º BL: Paging Mode                 º
º BH: aktuelle Farbseite          º
º ---                             º
Û---------------------------------ì
```

In BL steht nach dem Aufruf der Wert des Seitenmodus:

```
0:  4 Seiten aus 64
1: 16 Seiten aus 16
```

Das Register BH enthält die Nummer der aktuell eingestellten Farbseite. Die Funktion ist nur bei VGA-Karten implementiert.

Perform Gray-Scale Summing (AL = 1BH)

Es werden die Farben einzelner Farbregister addiert und in Grauwerte umgewandelt.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 10H (Set Palette)           º
º AL: 1BH                         º
º BX: Erstes Farbregister         º
º CX: Anzahl Farbregister         º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Bei der Addition werden die Anteile der Farben aus den Farbregistern ausgelesen und mit:

```
Rot = 30 %,  Grün = 59%,  Blau = 11%
```

gewichtet. Diese neu ermittelten Werte überschreiben dann die alten Registerwerte. Als Ergebnis wird dann ein Grauwert ausgegeben. Die Funktion ist nur bei VGA-Karten implementiert.

EGA/VGA-Video Textmode Character Generator (AH = 11H)

Dieser Aufruf dient zur Initialisierung des internen Zeichensatzes bei EGA- und VGA-Karten.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 11H (Set Character Code)    º
º AL: Subcode                     º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Die Routine erwartet im Register AL einen Unterfunktionscode, der die jeweilige Aktion steuert. Insgesamt sind die Codes AL = 0 bis 30H belegt. Im Textmode dienen die Unterfunktionen 00H, 01H und 02H zum Laden der Zeichenfonts. Die gleiche Aufgabe übernehmen die Unterfunktionen 20H, 21H, 22H und 23H im Grafikmodus. Die Unterfunktionen 10H, 11H und 12H laden ebenfalls neue Zeichenfonts im Textmode.

Da sie aber gleichzeitig den CRT-Controller neu initialisieren, sollten sie nur nach einem Mode-Kommando (AH = 00H) benutzt werden. Die VGA-Karte stellt insgesamt acht Zeichenfonts in der Bildspeicherebene (MAP 2) zur Verfügung. Pro Font werden in dieser MAP dann 256 Zeichen gespeichert. Der Zeichensatzgenerator reserviert für jedes Zeichen 32 Byte in der Tabelle, so daß sich Fonts mit bis zu 8x32 Pixeln definieren lassen. Ein Font mit 256 Zeichen belegt somit 8192 Byte im Zeichengenerator. Es gelten folgende Aufrufkonventionen für die Unterfunktionen:

Load User Characterfont (AL = 0)

Mit diesem Aufrufcode wird ein neuer Zeichensatz in einen der vier Speicherräume der Karte geladen.

```
Ö---------------------------------Ì
º       CALL   :  INT 10          º
º                                 º
ºAH: 11H (Set Character Code)     º
ºAL: 00                           º
ºBH: Anzahl Byte/Zeichen          º
ºBL: Block für Zeichensatz        º
ºCX: Anzahl Zeichen im Zeichensatzº
ºDX: Offset in Zeichensatztabelle º
ºES:BP Adresse Zeichensatztabelle º
û---------------------------------À
º       RETURN                    º
º---                              º
Û---------------------------------ì
```

Der Inhalt des Registers BL selektiert dabei die Seite des Zeichensatzes (0-3). Die Zeichencodes befinden sich in einer Tabelle, deren Adresse im Registerpaar ES:BP übergeben wird. Im Register BH steht, wie viele Byte pro Zeichen in der Tabelle gespeichert sind, während die Zahl der Zeichen in CX übergeben wird. DX enthält den Offset auf das erste Zeichen des Blocks.

Load ROM 8x14 Monochrom Characterfont (AL = 1)

Mit diesem Aufruf wird bei der EGA-Karte der Monochrom (8x14 Pixel)-Zeichensatz aus dem ROM aktiviert. Die Speicherbank wird dabei durch den Inhalt von BL selektiert. Bei der VGA-Karte wird der 8x14-Zeichensatz selektiert.

Load ROM 8x84 Monochrom Characterfont (AL = 2)

Mit diesem Aufruf läßt sich ein Monochrom (8x8 Pixel)-Zeichensatz aus dem ROM aktivieren, wobei die Speicherbank dabei durch den Inhalt von BL selektiert wird.

Set Block specifier (AL = 3)

Durch diese Unterfunktion läßt sich ein Zeichensatz aus acht Blöcken wählen, wobei der Inhalt des Registers BL die aktive Seite bestimmt. Durch die Codes AL = 0 .. 2 lassen sich ja verschiedene Zeichensätze in die einzelnen Blöcke laden. Der Inhalt von BL wird als Bitfeld interpretiert und selektiert in Abhängigkeit von Bit 3 den Font:

```
Ö--------Ú--------------------Ì
º Bit 3  º Blocknummer         º
û--------é--------------------À
º   0    º Bit 0, 1 und 4      º
º   1    º Bit 3, 2 und 5      º
Û--------Ù--------------------ì
```

Die Auswahl des Blockes durch Bit 4 oder 5 gilt jedoch nur bei der VGA-Karte, während bei der EGA-Karte nur zwei Bit benutzt werden. Das Bit 3 spezifiziert dabei, welche Bildschirmseite (0 .. 3) selektiert werden soll.

Load ROM 8x16 Characterfont (AL = 4)

Mit diesem Aufruf wird bei der VGA-Karte der 8x16-Zeichensatz geladen. In BL ist die Blocknummer (EGA: 0-3, VGA: 0-7) zu übergeben.

Die folgenden Routinen mit den Funktionscodes 1xH führen die gleichen Funktionen wie die Aufrufe AL = 00 bis 04 aus. Sie sollten aber nur direkt nach einem Wechsel des Videomodes genutzt werden, da sie die Register des CRT-Controllers neu setzen.

Load User Characterfont (AL = 10)

Mit dieser Funktion lassen sich eigene Textzeichensätze laden.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 11H (Set Character Code)    º
º AL: 10H                         º
º BH: Zahl der Zeichen / Byte     º
º BL: Blocknummer (0-3 EGA,       º
º     0-7 VGA)                    º
º CX: Zeichenzahl im Zeichensatz  º
º DX: Offset in die Tabelle       º
º ES:BP Zeiger auf die Tabelle    º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Das Format der Zeichensatztabelle sieht für jedes Zeichen n Byte vor. Ein 8x16-Pixel-Font belegt dann für jedes Zeichen 16 Byte. Das erste Byte eines Zeichens beschreibt die oberste Zeile.

```
              7 6 5 4 3 2 1 0    Wert
Byte 0        - - - - - - - -    00H
     1        - - - - - - - -    00H
     2        - - - * - - - -    10H
     3        - - * * * - - -    38H
     4        - * * - * * - -    6CH
     5        * * - - - * * -    C6H
     6        * * - - - * * -    C6H
     7        * * * * * * * -    FEH
     8        * * - - - * * -    C6H
     9        * * - - - * * -    C6H
    10        * * - - - * * -    C6H
    11        - - - - - - - -    00H
    12        - - - - - - - -    00H
    13        - - - - - - - -    00H
    14        - - - - - - - -    00H
```

Bild 13.1: Darstellung eines 8x14-Pixel-Zeichens

In den folgenden Bytes stehen dann die Daten für die restlichen Zeilen des Zeichens. In den Registern BX, CX und DX wird der Aufbau der Tabelle spezifiziert, während in **ES:BP** der Zeiger auf die Tabelle übergeben wird.

Load ROM 8x14 Monochrom Characterfont (AL = 11H)

Diese Funktion entspricht der Funktion AL = 01H.

Load ROM 8x8 double dot Characterfont (AL = 12H)

Diese Funktion entspricht der Funktion AL = 02H.

Load ROM 8x16 Characterfont (AL = 14H)

Diese Funktion entspricht der Funktion AL = 04H.

Set User 8x8 Graphik Characterfont (AL = 20H)

Mit diesem Aufruf läßt sich eine Tabelle mit dem Grafikzeichensatz laden. Es wird dabei der unter der INT 1F-Adresse abgespeicherte Zeichensatz geladen. Die Adresse wird per Registerpaar ES:BP übergeben.

Set User Graphik Characterfont (AL = 21H)

Mit diesem Aufrufcode wird ein neuer Zeichensatz in einen der vier Speicherräume der EGA/VGA-Karte geladen. Der Zeichencode befindet sich in einer RAM-Tabelle, deren Adresse im Registerpaar ES:BP übergeben wird. Im Register CX steht, wie viele Bytes pro Zeichen in der Tabelle gespeichert sind. Im Register BL wird die Zahl der Zeilen pro Seite spezifiziert:

BL	Zeilen / Seite
1	14
2	25
3	43

Ist der Wert von BL = 00H, wird der Wert Zeilen/Seite in DL übergeben.

Load ROM 8x14 Graphik Characterfont (AL = 22H)

Mit diesem Aufrufcode wird der 8x14-Pixel-Zeichensatz aus dem internen ROM geladen. Das Register BL spezifiziert dabei die Zeilenzahl pro Bildschirmseite. Ist BL = 00, dann enthält DL die Zeilenzahl pro Seite.

Load ROM 8x8 double dot Graphik Characterfont (AL = 23H)

Mit diesem Aufrufcode wird der 8x8-Pixel-Zeichensatz aus dem internen ROM geladen. Das Register BL spezifiziert dabei die Zeilenzahl pro Bildschirmseite. Es gilt die gleiche Nomenklatur wie bei der Funktion 22H. In DL steht die Zeilenzahl, falls BL = 00 ist.

Load ROM 8x16 Graphik Characterfont (AL = 24H)

Mit diesem Aufrufcode wird der 8x16 Pixel VGA/MCGA-Zeichensatz aus dem internen ROM geladen. Das Register BL spezifiziert dabei die Zeilenzahl pro Bildschirmseite. Ist BL = 00 dann, enthält DL die Zeilenzahl pro Seite.

Get Font Info (AL = 30H)

Diese Unterfunktion erlaubt es, den Status der EGA/VGA-Karte zu lesen. Das Register BH selektiert dabei den Typ des Zeichengenerators, dessen Adresse zurückzugeben ist.

```
 BH ° Bemerkung
----é------------
  0 ° INT 1F-Vektors
  1 ° INT 44-Vektors
  2 ° 8x14-ROM-Zeichensatz
  3 ° 8x8-ROM-Zeichensatz
  4 ° 8x8-ROM-Zeichensatz obere Hälfte
  5 ° 8x14-ROM-Zeichensatz obere Hälfte
  6 ° 8x16-ROM-VGA-Zeichensatz
  7 ° 9x16-ROM-VGA-Zeichensatz
```

Nach dem Aufruf finden sich die Statusinformationen in folgenden Registern:

```
 Register ° Information
----------é------------
  ES:BP   ° Adresse des Zeichengenerators
  CX      ° Zahl der Bytes pro Zeichen
  DL      ° Zeilen pro Seite - 1
```

Alternate Function select (AH = 12H)

Mit diesem Aufruf läßt sich die Konfiguration der Karte lesen, sowie die Print-Screen-Einstellung der EGA-, VGA-, MCGA- oder PS/2-Karten verändern.

```
Ö---------------------------------Ì
º        CALL   :  INT 10         º
º                                 º
º AH: 12 (Konfiguration)          º
º BL: Subcode                     º
û---------------------------------À
º        RETURN                   º
º ---                             º
Û---------------------------------ì
```

Die Routine erwartet im Register BL einen Unterfunktionscode, der folgende Bedeutung besitzt:

Get EGA/VGA Info (BL = 10H)

Die Konfiguration der EGA/VGA-Karte wird gelesen. Die Informationen finden sich nach dem Aufruf in folgenden Registern:

```
 Register ° Information
----------é-----------------------
  BH      ° 0 Farbmodus aktiv
          ° 1 Monochrom-Mode
  BL      ° Speichergröße
          ° 00 :  64 Kbyte  01 : 128 Kbyte
          ° 10 : 196 Kbyte  11 : 256 Kbyte
```

In CL stehen die *feature Bits*, während CH die Schalterstellung zurückgibt.

Alternate Print Screen (BL = 20H)

Mit diesem Aufruf wird die BIOS-Print-Screen-Funktion auf 43 Zeilen pro Seite umgestellt. Die Funktion ist bei EGA-, VGA-, MCGA- und PS/2-Karten implementiert.

Select vertical Resolution (BL = 30H)

Mit dieser Funktion wird die Zahl der Scan-Zeilen eingestellt:

```
AL:  00 200 Scan-Zeilen
     01 350 Scan-Zeilen
     02 400 Scan-Zeilen
```

Die Aktivierung erfolgt nach dem Setzen des Videomodes. Im Register AL steht nach dem Aufruf der Wert 12H, falls die Funktion implementiert ist (nur VGA-Karten).

Load Palette (BL = 31H)

Mit dieser Funktion erfolgt die Aktivierung der Default-Palette beim VGA/MCGA-Adapter:

```
AL:  00 Laden der Default-Palette beim Moduswechsel
     01 Deaktivierung der Default-Palette
```

Nach dem Aufruf gibt der Treiber in AL den Wert 12H zurück.

Video Addressing (BL = 32H)

Mit dieser Funktion erfolgt die Aktivierung des Bildschirmspeichers beim VGA/MCGA-Adapter:

```
AL:  00  Zugriff auf den Bildspeicher erlaubt
     01  Zugriff auf den Bildspeicher gesperrt
```

Nach dem Aufruf gibt der Treiber in AL den Wert 12H zurück.

Gray Scale Summing (BL = 33H)

Mit dieser Funktion wird die Farbaddition bei der VGA/MCGA-Karte gesteuert:

```
AL:  00  Aktivierung Color Summing
     01  Sperre Color Summing
```

Nach dem Aufruf gibt der Treiber in AL den Wert 12H zurück.

Cursor Emulation (BL = 34H)

Mit dieser Funktion wird die Cusor-Emulation beim VGA/MCGA-Adapter gesteuert:

```
AL:  00  Aktivierung Cursor-Emulation
     01  Blockierung Cursor-Emulation
```

Nach dem Aufruf gibt der Treiber in AL den Wert 12H zurück.

Display Switch Interface (BL = 35H)

Mit dieser Funktion erfolgt das Display-Switching mit dem Register AL beim VGA/MCGA-Adapter:

```
AL:  00  deaktiviere den Systemmonitor
     01  aktiviere den Systemmonitor
     02  deaktiviere den aktiven Monitor
     03  aktiviere den inaktiven Monitor
     80  setze »System Board Video Activ Flag«
```

In den Registern ES:DX wird die Adresse des 128 Byte langen Switch-Status-Puffers übergeben. Mit dieser Funktion lassen sich zwei Monitore gleichzeitig im System betreiben. Mit der Umschaltung werden die Parameter des deaktivierten Monitors in den Puffer gesichert. Nach dem Aufruf gibt der Treiber in AL den Wert 12H zurück.

Video Refresh Control (BL = 36H)

Mit dieser Funktion wird die Bildschirmanzeige beim PS/2-VGA-Adapter gesteuert:

```
AL:  00  Aktivierung Bildschirmanzeige
     01  Blockierung Bildschirmanzeige
```

Durch Aufruf dieser Unterfunktion läßt sich der Monitor leicht dunkel steuern. Nach dem Aufruf gibt der Treiber in AL den Wert 12H zurück.

Zeichenkette ausgeben (AH = 13H)

Beim EGA- und VGA-BIOS existiert allerdings noch der Aufruf 13H, um ganze Zeichenketten auszugeben. Es gilt folgende Aufrufkonvention:

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  13H  (Write String)        º
º AL:  Untercode                  º
º CX:  Stringlänge in Byte        º
º DX:  Cursorposition             º
º BH:  Bildschirmseite            º
º BL:  Attributcode               º
º ES:BP Anfangsadresse String     º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Der Aufruf kennt mehrere Unterfunktionen, die über den Wert im **AL**-Register selektiert werden.

Write String (AL = 0)

Die Zeichenkette enthält nur die auszugebenden Zeichen. Im Register BL wird das Attributbyte übergeben. Das Attribut gilt dann für den gesamten String. Die Ausgabe erfolgt ab der aktuellen Cursorposition. Zeichen wie CR, LF, BS und Bell werden innerhalb des Strings als Steuerzeichen für die Cursorbewegung interpretiert. Ansonsten bleibt die Cursorposition erhalten.

Write String with Attributbyte (AL = 1)

Die Zeichenkette enthält ebenfalls nur die auszugebenden Zeichen, wobei das Attributbyte per BL-Register gesetzt wird und für den ganzen String gilt. Bei diesem Aufruf wird der Cursor vor der Ausgabe auf die in DX angegebene Position gesetzt.

Write String and Attributebytes (AL = 2)

In der Zeichenkette findet sich immer ein Zeichen mit dem jeweiligen Attribut im folgenden Byte:

```
<Zeichen, Attribut, Zeichen, Attribut, . . . ,Zeichen, Attribut>
```

Damit läßt sich jedes Zeichen mit verschiedenen Attributen belegen. Das BL-Register wird bei diesem Aufruf nicht benutzt. Die Funktion gibt den String an der aktuellen Cursorposition aus, so daß das Register DX ebenfalls nicht belegt ist.

Write String and Attributbytes (AL = 3)

Wie beim Aufruf AL = 2 finden sich im String sowohl Zeichen als auch die zugehörigen Attribute. Das BL-Register bleibt unbenutzt. Der Wert in DX setzt den Cursor vor der Ausgabe auf die spezifizierte Position.

Die Funktionen interpretieren die Steuerzeichen (CR, LF, BS, etc.).

Read/Write Display Combination Code (AH = 1AH)

Beim VGA/MCGA-BIOS existiert zusätzlich ein Aufruf zum Lesen und Setzen der Bildschirmkonfiguration. Die Informationen werden im BIOS in der Video-Status-Tabelle verwaltet. Es gilt folgende Auf-rufschnittstelle:

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  1AH  (Read/Write Mode)     º
º AL:  Untercode                  º
û---------------------------------À
º      RETURN                     º
º ---                             º
Û---------------------------------ì
```

Der Aufruf kennt mehrere Unterfunktionen, die über den Wert im AL-Register selektiert werden.

Read Monitor Status (AL = 0)

Mit diesem Aufruf läßt sich der Status des Monitors abfragen. In AL wird der Wert 00 übergeben. Nach dem Aufruf enthält BL den *active display mode* und BH den *alternate display code*.

Set Display Combination (AL = 01H)

Mit dieser Unterfunktion läßt sich die Einstellung des Monitors modifizieren. Es gilt folgende Aufruf-schnittstelle:

```
Ö--------------------------------Ì
º      CALL  :  INT 10           º
º                                º
º AH:  1AH  (Read/Write Mode)    º
º AL:  01H  (Set Mode)           º
º BH:  Code d. alternativen Konst.º
º BL:  Code d. aktive Konst.     º
û--------------------------------À
º      RETURN                    º
º AL: 1AH Funktion unterstützt   º
Û--------------------------------ì
```

Im Register BH wird der Code für die neue Konstellation übergeben, während im Register BL der Code der aktiven Konstellation steht. Für die Karten werden folgende Werte benutzt:

```
Ö------Ú---------------------------------------Ì
º Code º Bedeutung                             º
û------é---------------------------------------À
º  00H º kein Monitor                          º
º  01H º MDA mit Monochrom-Monitor             º
º  02H º CGA mit Farbgrafik- oder EGA-Monitor  º
º  03H º reserviert                            º
º  04H º EGA mit Farbgrafik- oder EGA-Monitor  º
º  05H º EGA mit Monochrom-Monitor             º
º  06H º PGA mit professional Grafik-Monitor   º
º  07H º VGA mit analogem Monochrom-Monitor    º
º  08H º VGA mit analogem Farbgrafik-Monitor   º
º  09H º reserviert                            º
º  0AH º MCGA digital Farb-Monitor             º
º  0BH º MCGA mit analogem Monochrom-Monitor   º
º  0CH º MCGA mit analogem Farbgrafik-Monitor  º
º  0DH º reserviert                            º
º  0EH º reserviert                            º
º  0FH º reserviert                            º
Û------Ù---------------------------------------ì
```

Tabelle 13.9: Monitortyp

Wird der Code FFH zurückgegeben, erkennt das BIOS den Typ des angeschlossenen Monitors nicht. Ist die Funktion implementiert, gibt sie nach dem Aufruf in AL den Wert 1AH zurück.

Get Video-Status (AH = 1BH)

Beim VGA/MCGA-BIOS liefert diese neue Funktion Informationen über die internen BIOS-Tabellen, die eingangs bereits beschrieben wurden.

```
Ö--------------------------------Ì
º      CALL  :  INT 10           º
º                                º
º AH:  1BH  (Get Video Status)   º
º AL:  1BH                       º
º BX:  00H -> 64 Byte Tabelle    º
º ES:DI Zeiger auf Tabelle       º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Der Aufruf gibt im Register BX den Wert 00H zurück, falls die Tabelle 64 Byte umfaßt. Die Adresse dieser Tabelle findet sich nach dem Aufruf in ES:DI. Die Tabelle besitzt folgenden Aufbau:

Offset	Bytes	Feld
00H	2	Offset Video Funktionstabelle
02H	2	Segment Video Funktionstabelle
04H	1	Videomode (s. Funktion AH=00H)
05H	2	Spalten pro Seite (40 oder 80)
07H	2	Zeichen pro Seite
09H	2	Startadresse Bildschirmspeicher
0BH	16	Cursorposition für 8 Bildschirmseiten
1BH	2	Cursorgröße (Start, Ende)
1DH	1	aktive Bildschirmseite
1EH	2	Portadresse Controller
20H	1	Mode CRT-Register
21H	1	Palette des CRT-Registers
22H	1	Zeilen pro Seite
23H	2	Zeichenhöhe in Scan-Zeilen
25H	1	aktive Monitor (Display Combination Code)
26H	1	altern.Monitor (Display Combination Code)
27H	2	max. verfügbare Farben im akt. Videomode
29H	1	aktuell verfügbare Bildschirmseiten
2AH	1	aktuelle Anzahl Scan-Zeilen
		00=200, 01=350, 02=400, 03=480
2BH	1	primäre Zeichensatzblock Nummer
2CH	1	sekundäre Zeichensatzblock Nummer
2DH	1	Informationsbyte
		Bit 7,6 reserviert
		Bit 5: 0 Backgr. intensit. 1 Blinken
		Bit 4: 1 Kursor-Emulation aktiv
		Bit 3: 0 load default palette
		Bit 2: 1 Monochrom-Monitor
		Bit 1: 1 Farbsummierung aktiv
		Bit 0: 1 alle Monitore aktiv
2EH	30	reserviert
31H	1	Bildspeichergröße: 00 = 64 K, 1 = 128 K
		2 = 192 K, 3 = 256 K
32H	1	Status-Information-Save-Pointer
		Bit 7-5 reserviert
		Bit 4: 1 überschreiben Palette aktiv
		Bit 3: 1 überschreiben Grafikzeichens.
		Bit 2: 1 überschreiben Textzeichensatz
		Bit 1: 1 Dynamic Save Area aktiv
		Bit 0: 1 Zeichensatz 512 Char. aktiv
33H	13	reserviert

Tabelle 13.10: Video-Status-Tabelle

Der INT 42H zeigt auf den Beginn dieser Tabelle. Diese sollte aber immer über den BIOS-INT 10-Aufruf gelesen werden. Die ersten beiden Einträge verweisen ihrerseits wieder auf 16 Byte lange Tabellen mit den Video-Funktionstabellen. Diese besitzen folgenden Aufbau:

Offset	Bytes	Feld
00H	3	Videomode
03H	4	reserviert
07H	1	Scan-Zeilen: 0=200, 1=350, 2=400
08H	1	Zeichensatzblöcke im Textmode
09H	1	Zahl d. aktiven Zeichensatzblöcke
0AH	2	Flagword
		Bit 15 : paging Mode
		Bit 14 : Palettenregister
		Bit 13 : EGA-Palette
		Bit 12 : Cursor-Emulation
		Bit 11 : Default-Palette laden
		Bit 10 : Default-Zeichensatz laden
		Bit 9 : Farbaddition aktiv
		Bit 8 : alle Monitore aktiv
		Bit 7 : reserviert
		..
		Bit 4 : reserviert
		Bit 3 : Display Combination Code
		Bit 2 : Blinken/Hintergrund Intensität
		Bit 1 : Save/Restore Status
		Bit 0 : Light Pen (EGA)
0CH	2	reserviert
0EH	1	Save-Ptr-Funktionsbyte
		Bit 7 : reserviert
		..
		Bit 6 : reserviert
		Bit 5 : Display Combination Code
		Bit 4 : Überschreiben der Palette
		Bit 3 : Überschreiben Grafikzeichens.
		Bit 2 : Überschreiben Textzeichensatz
		Bit 1 : Dynamic Save Area
		Bit 0 : Zeichensatz mit 512 Zeichen

Tabelle 13.11: Video-Funktions-Tabelle

Save/Restore-Video-Status (AH = 1CH)

Das VGA-BIOS speichert und liest den mit der Funktion 1BH gelesenen Videostatus. Es gilt folgende Aufrufschnittstelle:

```
CALL  :  INT 10

AH:  1CH  (Get/Set Video Status)
AL:  Untercode
CX:  Status
-----------------------------------
RETURN
ES:BX Zeiger auf Tabelle
```

Im Register AL wird ein Untercode zur Selektion der gewünschten Funktion übergeben. Ist die Unterfunktion implementiert, gibt sie nach dem Aufruf den Wert AL = 1CH zurück.

State Buffer Size (AL=00H)

Hier ist im Register CX der Save/Restore-Status zu übergeben. Das Register besitzt folgende Kodierung:

```
Bit 15-3: reserviert
       2: 1  DAC-Status und Farbregister
       1: 1  BIOS-Datenbereich
       0: 1  Video-Hardwarestatus
```

Das Register BX enthält nach dem Aufruf die Anzahl der 64-Byte-Blöcke.

Save Video-Status (AL=01H)

Hier ist im Register CX der Save/Restore-Status zu übergeben.

Das Register besitzt den bei der Unterfunktion AL= 00 gezeigten Aufbau. Die Adresse des Puffers für die Datensicherung ist in ES:BX zu übergeben.

Restore Video-Status (AL=02H)

Hier ist im Register CX der Save/Restore-Status zu übergeben. Das Register besitzt die bei AL=00 gezeigte Kodierung. In ES:BX ist die Adresse des Puffers mit den Statusinformationen zu übergeben.

13.3 Die erweiterten VGA-Funktionen (AH=6FH)

Die VGA-Grafikkarten der Firmen VEGA und Video7 enthalten einige zusätzliche Funktionen, die sich über den Subcode AH = 6FH ansprechen lassen.

INT 10-Installation Check (AX = 6F00H)

Mit diesem Aufruf läßt sich prüfen, ob das BIOS die erweiterten Funktionen unterstützt. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  6FH (Extended Function)    º
º AL:  00                         º
û---------------------------------À
º      RETURN                     º
º BX: 5637H vorhanden             º
Û---------------------------------ì
```

Im Register BX gibt die Funktion dann den Wert 5637H (V7) zurück. Dann sind die erweiterten Funktionen vorhanden.

INT 10-GET INFO (AX = 6F01H)

Mit dieser Subfunktion lassen sich bei der VEGA/ VGA-Karte einige Informationen über den Hardwarestatus abfragen.

```
Ö--------------------------------Ì
º      CALL  :  INT 10           º
º                                º
º AH:  6FH (Extended Function)   º
º AL:  01                        º
û--------------------------------À
º      RETURN                    º
º AX: Statuscode                 º
Û--------------------------------ì
```

Im Register AL wird der Monitor-Typ zurückgegeben, während AH die Informationen des Status Registers enthält:

```
Bit  0 = display enable
       0 = display enabled
       1 = vertical or horizontal retrace in progress
Bit  1 = light pen flip flop set
Bit  2 = light pen switch activated
Bit  3 = vertical sync
Bit  4 = monitor resolution
       0 = high resolution (>200 lines)
       1 = low resolution (<=200 lines)
Bit  5 = display type
       0 = color
       1 = monochrome
Bits 6,7 = diagnostic bits
```

Die Bits 0-3 besitzen die gleiche Bedeutung wie die EGA-/VGA-Statusregister.

INT 10-Get Mode and Screen Resolution (AX = 6F04H)

Der Aufruf fragt die Auflösung des aktuellen Bildschirms ab. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL  :  INT 10           º
º                                º
º AH:  6FH (Extended Function)   º
º AL:  04H                       º
û--------------------------------À
º      RETURN                    º
º AL: Video Mode                 º
º BX: horizontale Auflösung      º
º CX: vertikale   Auflösung      º
Û--------------------------------ì
```

Die Auflösung wird entweder in Zeichen und Zeilen (Textmode) oder in Pixeln (Grafikmodus) zurückgegeben. In AL steht der mit der Funktion 05H gesetzte erweiterte Videomode.

INT 10-Set Video Mode (AX = 6F05H)

Mit dem Aufruf läßt sich der erweiterte Videomode der Karte setzen. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL  :  INT 10           º
º                                º
º AH:  6FH (Extended Function)   º
º AL:  05H                       º
º BL:  Mode                      º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Der zu setzende Mode wird in BL übergeben. Es gelten folgende Einstellungen:

Code	Text	Zeichen	Grafik	Farben
00 - 13H	= Standard IBM Modes			
40H	80x43	8x8		
41H	132x25	8x14		
42H	132x43	8x8		
43H	80x60	8x8		
44H	100x60	8x8		
45H	132x28	8x8		
60H			752x410	16
Code	Text	Zeichen	Grafik	Farben
00 - 13H	= Standard IBM Modes			
61H			720x540	16
62H			800x600	16
63H			1024x768	2
64H			1024x768	4
65H			1024x768	16
66H			640x400	256
67H			640x480	256
68H			720x540	256
69H			800x600	256
70H			752x410	16 gray
71H			720x540	16 gray
72H			800x600	16 gray
73H			1024x768	2 gray
74H			1024x768	4 gray
75H			1024x768	16 gray
76H			640x400	256 gray
77H			640x480	256 gray
78H			720x540	256 gray
79H			800x600	256 gray

Tabelle 13.12: Extended Modes

INT 10-Select Autoswitch Mode (AH = 6F06H)

Mit diesem Aufruf läßt sich einstellen, in welchem Mode die VGA-Karte zu betreiben ist.

```
CALL  :  INT 10

AH:  6FH (Extended Function)
AL:  06H
BL:  Mode
BH:  Flag
```

```
RETURN
---
```

In BL wird der Autoswitch-Mode gesetzt, wobei folgende Kodierung gilt:

```
BL = 00H Select EGA/VGA-only Mode
     01H Select Autoswitched
         VGA/EGA/CGA/MGA Mode
     02H Select 'bootup' CGA/MGA Mode
BH = enable/disable
     00H enable selection
     01H disable selection
```

Das Register BH dient zum Ein- und Ausschalten des Modes.

INT 10-Get Video Memory Configuration (AX = 6F07H)

Der Aufruf ermittelt die Speicherkonfiguration der Karte.

```
Ö---------------------------------Ì
º      CALL  :  INT 10            º
º                                 º
º AH:  6FH (Extended Function)    º
º AL:  07H                        º
û---------------------------------À
º      RETURN                     º
º AH:  Speicherstatus             º
º AL:  6FH                        º
º BH:  Chip Version               º
º BL:  Chip Version               º
º CX:  0000                       º
Û---------------------------------ì
```

Wird die Funktion unterstützt, gibt sie im Register AL den Wert 6FH zurück. Das Register CX ist dann 0000H. In AH steht die Speicherausrüstung der Karte:

```
Bits 0-6:  Zahl der 256K Blocks
           des Video RAM
       7:  DRAM/VRAM
           (0: DRAM, 1: VRAM)
```

Das Register BH und BL enthält einen Code, der die Revisionsnummer des Videochips angibt:

```
BH = Chip Revision (SR8F) (S/C Chip)
BL = Chip Revision (SR8E) (G/A Chip)
```

Nicht alle VGA-Karten unterstützen aber diese Aufrufe.

14 Netzwerkfunktionen

Bei der Beschreibung der Interrupts wurde bereits kurz auf verschiedene Netzwerkfunktionen eingegangen. So belegt die Novell Netzwerksoftware den INT 2F, Funktion 7AH zum Installations-Check. Auch die INT 2FH-Funktion B8H wird durch Netzwerkdienste belegt. Das gleiche gilt für die Funktion 11H des INT 2F. Einzelheiten finden sich in dem Kapitel über den INT 2F.

14.1 Die Funktionen des INT 2A

Das Microsoft- und das LANtastic-Netzwerk belegen einige Funktionen des INT 2A. Weiterhin gibt es Netzwerksoftware (z.B. DEC-Pathworks), die den INT 2A auf den INT 5C umlegen und damit die NetBIOS-Funktionen abwickeln. Nachfolgend werden die INT 2A-Funktionen des LANtastic-Netzwerkes beschrieben.

LANtastic Installation Check (INT 2A, AH = 00H)

Über diesen Funktionsaufruf wird die Installationsprüfung vorgenommen. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
°      CALL:  INT 2A              °
°                                 °
° AH = 00H Installation Check     °
û---------------------------------À
°      RETURN                     °
° AH: Status                      °
Û---------------------------------ì
```

Wird in AH ein Wert <> 0 zurückgegeben, ist der LANtastic-Netzwerktreiber installiert.

LANtastic Execute NetBIOS Request, no Error Request (INT 2A, AH = 01H)

Mit diesem Aufruf wird ein NetBIOS-Aufruf eingeleitet. Bei Fehlern gibt es keine Wiederholung.

```
Ö---------------------------------Ì
°      CALL:  INT 2A              °
°                                 °
° AH = 01H Execute NetBIOS        °
° ES:BX NCB                       °
û---------------------------------À
°      RETURN                     °
° AH: Status 0 -> o.k.            °
°            1 -> Fehler          °
° AL: Fehlercode                  °
Û---------------------------------ì
```

Im Registerpaar ES:BX ist die Adresse auf den Netzwork-Control-Block (NCB) zu übergeben. Dies ist eine Datenstruktur, die das gewünschte Kommando beschreibt. Der Aufbau eines NCB ist im folgenden Abschnitt (INT 5C) beschrieben.

In AH wird ein Status zurückgegeben:

```
AH = 0  Aufruf fehlerfrei
     1  Fehler beim Aufruf
```

Bei einem Fehler findet sich in AL der NetBIOS-Fehlercode.

LANtastic Check Direct I/O (INT 2A, AH = 03H)

Mit diesem Aufruf läßt sich prüfen, ob direkt auf ein Gerät (per INT 13 oder INT 25) zugegriffen werden darf. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2A              º
º                                 º
º AH = 03H Check Direct I/O       º
º DS:SI Adresse ASCIIZ-String     º
û---------------------------------À
º      RETURN                     º
º CF: 0 Direct Acces              º
º CF: 1 File Access               º
Û---------------------------------ì
```

Im Registerpaar DS:SI ist die Adresse auf einen ASCIIZ-String mit dem Gerätenamen zu übergeben. Das Carry-Flag signalisiert, ob ein direkter Zugriff auf das Gerät möglich ist. Bei gesetztem Carry-Flag kann nur über Dateien auf das Gerät zugegriffen werden. Ein direkter Zugriff muß dann entfallen. Die Funktion wird durch den DOS-Kern bei Aufrufen des INT 25 und INT 26 aktiviert.

LANtastic Execute NetBIOS Request (INT 2A, AH = 04H)

Mit diesem Aufruf wird ein NetBIOS-Aufruf eingeleitet. Es gilt dabei folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2A              º
º                                 º
º AH = 04H Execute NetBIOS        º
º AL = Error Retry                º
º ES:BX NCB                       º
û---------------------------------À
º      RETURN                     º
º AX: 0000H -> ok.                º
º AH: 01H   -> Fehler             º
º AL: Fehlercode                  º
Û---------------------------------ì
```

Im Registerpaar ES:BX ist die Adresse auf den Netzwork-Control-Block (NCB) zu übergeben. Dies ist eine Datenstruktur, die das gewünschte Kommando beschreibt. Der Aufbau eines NCB ist im folgenden Abschnitt (INT 5C) beschrieben. Der Wert in AL spezifiziert, ob der Aufruf im Fehlerfall zu wiederholen ist:

```
AL = 00H: automatische Wiederholung im Fehlerfall
          bei den Funktion 09H, 12H und 21H
     01H: keine Wiederholungen
```

In AH wird ein Status zurückgegeben:

```
AH = 0  Aufruf fehlerfrei
     1  Fehler beim Aufruf
```

Bei einem Fehler findet sich in AL der NetBIOS-Fehlercode. Der Aufruf des INT 2A wird in den meisten Fällen auf den INT 5C umgeleitet.

LANtastic Get Network Resource Availability (INT 2A, AH = 05H)

Mit diesem Aufruf läßt sich die Verfügbarkeit verschiedener Resourcen überprüfen. Es gilt dabei folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2A              º
º                                 º
º AH = 05H Get Resource Avail.    º
û---------------------------------À
º      RETURN                     º
º AX: reserviert                  º
º BX: Zahl der Netzwerknamen      º
º CX: Zahl der NCB's              º
º DX: Zahl der Sessions           º
Û---------------------------------ì
```

Nach dem Aufruf finden sich in verschiedenen Registern Informationen über die verfügbaren Resourcen. In BX wird die Zahl der verfügbaren Netzwerknamen angegeben. CX enthält die Zahl der verfügbaren Netzwerk-Control-Blocks (NCB). In DX wird die Zahl der verfügbaren Netzwerksitzungen angegeben.

LANtastic Network Print-Stream Control (INT 2A, AH = 06H)

Mit diesem Aufruf läßt sich die Druckerausgabe steuern. Es gilt dabei folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2A              º
º                                 º
º AH = 06H Print Stream Control   º
º AL: Status                      º
û---------------------------------À
º      RETURN                     º
º CY: 1 Fehler                    º
º AX: Fehlercode                  º
º CY: 0 ok                        º
Û---------------------------------ì
```

In AL wird ein Status übergeben, der die Printerausgabe steuert:

```
AL = 01H Set concatenation mode (all outout to 1 job)
     02H Set truncation mode
     03H Flush output and start new job
```

Nach dem Aufruf gibt ein gesetztes Carry-Flag an, daß ein Fehler aufgetreten ist. Der Fehlercode findet sich dann in AX.

Verschiedene andere LAN-Programme benutzen ebenfalls den INT 2A (siehe Kapitel über den INT 2A).

14.2 Die Funktionen des NetBIOS (INT 5CH)

Der Interrupt 5CH wird unter MS-DOS fest durch das NetBIOS belegt. Dies ist ein Treiber (BIOS-Erweiterung) der den meisten Netzwerkkarten beigefügt ist. Auf dessen Dienste setzen dann die Netzwerkprogramme verschiedener Hersteller auf. Der Treiber wird üblicherweise beim Systemstart in der Datei CONFIG.SYS installiert. Mit den NetBIOS-Diensten lassen sich bereits einige Kommunikationsfunktionen nutzen. Der Aufruf des NetBIOS erfolgt gemäß folgender Schnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 5C              º
º                                 º
º ES:BX Adresse NCB               º
û---------------------------------À
º      RETURN                     º
º AL: Status                      º
Û---------------------------------ì
```

Im Registerpaar ES:BX ist die Adresse eines Netzwerk-Control-Blockes zu übergeben. Dieser Block enthält die Steueranweisungen und Daten für die Übertragung. Der rufende Prozeß muß den NCB selbst aufbauen. Das NetBIOS gibt in diesem Block dann die Statusmeldungen der Operation zurück.

Der Block liegt jedoch immer im Speicherbereich des rufenden Prozesses. Der NetBIOS-Treiber merkt sich lediglich die Adressen auf den betreffenden Block. Durch diese Technik werden Mehrfachaufrufe von verschiedenen Anwenderprozessen möglich, da das NetBIOS die Daten nicht speichern muß.

Vor der Benutzung des NetBIOS-Interrupts muß das Programm prüfen, ob der Vektor des INT 5CH mit einer gültigen Adresse belegt ist.

14.3 Der Netzwerk-Control-Block (NCB)

Der Netzwerk-Control-Block (NCB) besitzt folgende Struktur:

```
Ö-----Ú-------Ú-------------------------------------------------Ì
º Ofs.º Bytes º Feld                                           º
û-----é-------é-------------------------------------------------À
º 00H º   1   º Kommandocode (ncb_command)                      º
º 01H º   1   º Returncode (ncb_retcode)                        º
º 02H º   1   º Session-Nummer (ncb_lsn)                        º
º 03H º   1   º Nummer des lokalen Adapters (ncb_num)           º
º 04H º   4   º Adresse des Datenpuffers (ncb_buffer)           º
º 08H º   2   º Länge des Datenpuffers (ncb_length)             º
º 0AH º  16   º Knotenname (ncb_callname)                       º
º 1AH º  16   º Knotenname (ncb_name)                           º
º 2AH º   1   º Receive Timeout (ncb_rto)                       º
º 2BH º   1   º Send Timeout (ncb_sto)                          º
º 2CH º   4   º Adresse POST-Routine (ncb_post)                 º
º 30H º   1   º Nummer des lokalen Adapters (ncb_lana_num)      º
º 31H º   1   º Netzwerk Completion Code (ncb_cmd_cplt)         º
º 32H º  14   º reservierte Felder (ncb_reserve)                º
Û-----Ù-------Ù-------------------------------------------------ì
```

Tabelle 14.1: Der Aufbau des Netzwerk-Control-Blocks (NCB)

Das erste Feld enthält ein Byte mit dem Kommandocode der aufzurufenden Funktion. Dieser Code bestimmt, welche Funktion des NetBIOS aktiviert wird. Eine Beschreibung aller implementierten Funktionen ist nachfolgend aufgeführt.

Das folgende Byte dient dem NetBIOS zur Rückgabe eines Statuswertes (Returncode). Dieses Feld muß vor Aufruf einer NetBIOS-Funktion immer auf den Wert FFH gesetzt werden.

Da das NetBIOS durchaus mehrere Aufträge (Requests) gleichzeitig bearbeiten kann, muß eine Unterscheidung dieser Aufträge möglich sein. Sobald ein Anwenderprozeß eine Kommunikationsanforderung an das NetBIOS richtet, beginnt dieses mit dem Aufbau einer Kommunikationsverbindung. Sobald diese Verbindung steht, gibt der Treiber im Feld *ncb_lsn* eine Session-Nummers zurück, mit der der Treiber die betreffende Verbindung identifiziert. Das Feld braucht nur bei der Funktion 10H (Start Session), nicht vom Anwenderprozeß belegt zu werden, da zu diesem Zeitpunkt die Nummer noch nicht bekannt ist. Alle weiteren Aufrufe erwarten eine korrekte Session-Nummer.

Das Feld *ncb_num* enthält die Nummer des lokalen Netzwerkknotens. Jeder Knoten im Netz erhält eine eindeutige Nummer, die auf der Adapterkarte eingestellt wird. Über diese Nummer erfolgt die Identifizierung der einzelnen Netzwerkteilnehmer. Der Treiber trägt die Nummer der Karte beim Verbindungsaufbau in den Block ein.

Das Feld *ncb_buffer* enthält einen 4-Byte-Adreßvektor auf den Puffer für die Sende- und Empfangsdaten. Der Anwenderprozeß muß diesen Puffer einrichten. Die Puffergröße ist genügend groß zu wählen. Auf Grund der Beschränkungen der 8086-Architektur kann die Größe allerdings auf 64 Kbyte beschränkt werden. Die Zahl der zu sendenden oder empfangenden Daten ist dann im folgenden Feld *ncb_length* als Wort zu übergeben.

Ab Offset 0AH findet sich ein 16 Byte langer Speicherbereich, in dem der ASCII-Name des gerufenen Knotens abgelegt wird. Das NetBIOS assoziiert dann die Session-Nummer beim Verbindungsaufbau mit diesem Namen. Das folgende Feld enthält den 16 Byte langen ASCII-Knotennamen des lokalen Knotens. Beide Einträge sind durch das rufende Programm zu initialisieren.

Die zwei Byte ab Offset 2AH spezifizieren die Timeout-Werte für den Sender (ncb_sto) und den Empfänger (ncb_rto). Die Werte sollten immer ungleich 0 gesetzt werden, da sonst keine Kommunikation zustande kommt, sobald nur die kleinste Verzögerung auftritt.

Ab Offset 2CH findet sich ein 4-Byte-Vektor auf die POST-Routine. Was hat es mit dieser Routine auf sich? In Tabelle 14.1 sind alle Kommandocodes für die einzelnen NetBIOS-Funktionen aufgeführt. Wird nur der Kommandocode (unterhalb 80H) angegeben, führt das BIOS erst die Funktion aus und gibt anschließend die Kontrolle an den wartenden Prozeß zurück (Wait). Dies bedeutet aber, daß das rufende Programm für die Dauer des Aufrufs blockiert ist. Oft soll diese Verhaltensweise aber vermieden werden. Das NetBIOS bietet deshalb einen Mechanismus an, bei dem der rufende Prozeß direkt die Kontrolle zurückerhält, sobald das NetBIOS den Auftrag akzeptiert hat (NoWait). Hierzu ist einfach das oberste Bit 8 im Kommandofeld *ncb_command* zu setzen (80H auf den Kommandocode addieren). Sobald der Auftrag im NetBIOS vorliegt, können beide Prozesse asynchron zueinander laufen. Nach Beendigung des Auftrages durch das NetBIOS muß im NoWait-Mode allerdings eine Synchronisierung mit dem Anwenderprozeß erfolgen. Hierzu dient die POST-Routine, die durch den Anwenderprozeß einzurichten ist. Die Adresse dieser Routine wird als Vektor im NCB mit übergeben. Sobald das NetBIOS den Auftrag abgewickelt hat, aktiviert es die unter der Adresse angegebene Routine per CALL-FAR-Aufruf. Diese Routine muß dann dafür sorgen, daß das Anwenderprogramm die entsprechenden Aktionen übernimmt.

Ab Offset 30H steht in einem Byte die Information, welcher lokale Adapter für die Kommunikation benutzt wird. Dieser Wert ist für den Standard-Adapter auf 0 zu setzen.

Im Byte ab Offset 31H gibt das Netzwerk einen *Completion Code* zurück, falls das NoWait-Bit (80H) gesetzt ist. Dieser signalisiert, ob der Aufruf korrekt übernommen wurde.

Ab Offset 32H sind im NCB weitere 14 Byte zu reservieren, die intern durch das BIOS benutzt werden.

14.4 Die NetBIOS-Kommandos

Nachfolgende Tabelle gibt die unterstützten NetBIOS-Funktionen wieder.

Code	Funktion
10H	Start Session with NCB_NAME Name (call)
11H	Listen for Call
12H	End Session with NCB_NAME Name (hangup)
14H	Send Data via NCB_LSN
15H	Receive Data from a Session
16H	Receive Data from any Session
17H	Send multiple Data Buffers
20H	Send unACKed Message (Datagram)
21H	Receive Datagram
22H	Send Broadcast Datagram
23H	Receive Broadcast Datagram
30H	Add Name to Name Table
31H	Delete Name from Name Table
32H	Reset Adapter Card and Tables
33H	Get Adapter Status
34H	Status of all Sessions for Name
35H	Cancel
36H	Add Group Name to Name Table
70H	Unlink from IBM Remote Program (no F0H Funct.)
71H	Send Data without ACK
72H	Send multiple Buffers without ACK
78H	Find Name
79H	Token-Ring Protocol Trace

Tabelle 14.2: Die NCB-Kommandocodes

Wird der Wert 80H zum Kommandocode addiert, geht die BIOS-Funktion in den NoWait-Status und gibt direkt die Kontrolle an den rufenden Prozeß zurück. Der NCB ist vor dem Aufruf mit den entsprechenden Werten zu belegen. Nicht jeder Aufruf belegt alle Felder des NCB.

Die Funktion 32H setzt den Netzwerkadapter zurück und löscht die benutzten Knotennamen.

Mit dem Aufruf 33H läßt sich der Status des Adapters abfragen. Der Aufruf gibt die Statusinformationen im NCB im Feld ncb_buffer zurück. Der Record besitzt folgenden Aufbau:

Ofs.	Bytes	Feld
00H	6	as_id
06H	1	as_jumpers
07H	1	as_post
08H	1	as_major
09H	1	as_minor
0AH	2	as_interval
0CH	2	as_crcerr
0EH	2	as_algerr

Tabelle 14.3: Statusinhalt der Funktion 33H

Ofs.	Bytes	Feld
10H	2	as_colerr
12H	2	as_abterr
14H	4	as_tcount
18H	4	as_rcount
1CH	2	as_retran
1EH	2	as_xresrc
20H	8	as_res0
28H	2	as_ncbfree
2AH	2	as_ncbmax
2CH	2	as_ncbx
2EH	4	as_res1
32H	2	as_sespend
34H	2	as_msp
36H	2	as_sesmax
38H	2	as_bufsize
3AH	2	as_names
3CH	16*x	as_name 16
		Name Structures:
		16 Byte nm_name
		1 Byte nm_num
		1 Byte nm-status

Tabelle 14.3: Statusinhalt der Funktion 33H (Ende)

Im ersten Byte steht die Knotenadresse (Wert **1** bis **FFH**). Diese Nummer wird bei der Installation des Adapters eingestellt und muß im Netzwerk eindeutig sein.

Mit der Funktion 30H (Add Name) läßt sich ein Name für den lokalen Knoten definieren, unter dem dieser Knoten angesprochen werden kann. Ein Adapter darf durchaus verschiedene Namen besitzen, alle Namen müssen aber eindeutig im Netzwerk sein.

Nach einem Aufruf (WAIT) enthält das Register **AL** einen Statuscode mit folgender Bedeutung:

AL	Status
00H	Successful
01H	Bad Buffer Size
03H	Invalid NETBIOS Command
05H	Timeout
06H	Receive Buffer too small
08H	Bad Session Number
09H	LAN Card out of Memory
0AH	Session Closed
0BH	Command has been Cancelled
0DH	Name already exists
0EH	Local Name Table full
0FH	Name still in use, can't delete
11H	Local Session Table full
12H	Remote PC not listening
13H	Bad NCB_NUM Field
14H	No Answer to CALL or no such Remote
15H	Name not in local Name Table
16H	Duplicate Name
17H	Bad Delete
18H	Abnormal End
19H	Name Error, multiple identical Names
1AH	Bad Packet
21H	Network Card busy
22H	Too many Commands queued
23H	Bad LAN Card Number
24H	Command finished while Cancelling
26H	Command can't be cancelled
FFH	NETBIOS busy

Tabelle 14.4: Fehlerstatus der NetBIOS-Aufrufe

Im NCB wird der Statuscode im Feld *ncb_retcode* ebenfalls zurückgegeben.

Weitere Informationen über die Funktionen des NetBIOS sind den Unterlagen der Adapterhersteller zu entnehmen.

Anhang

Die historische Entwicklung von MS-DOS

Mit der Einführung neuer Personalcomputer auf Basis der 8088/8086-Prozessoren, wurden im Jahre 1981 auch neue Betriebssysteme erforderlich. Neben der für die 16-Bit-Prozessoren portierten CP/M-86-Version von Digital Research erschien von Microsoft das Betriebssystem MS-DOS.

Dieses Betriebssystem war ursprünglich von der Fa. Seattle Computer unter dem bezeichnenden Namen QDOS (Quick and Dirthy Operating System) für die Intel-8086-Prozessoren entwickelt worden. Nachdem Microsoft die Rechte an diesem Produkt erworben hatte, wurde es ab 1981 unter dem Namen MS-DOS (Microsoft Disk Operating System) vertrieben. IBM erwarb ebenfalls Lizenzen und lieferte das Produkt unter dem Namen PC-DOS mit seinen Personalcomputern aus. Im Laufe der Zeit wurden verschiedene Versionen entwickelt, deren Geschichte nachfolgend kurz skizziert wird.

DOS 1.0

1981 wird mit dem IBM-PC die DOS-Version 1.0 vorgestellt. Sie unterstützt nur einseitige Disketten mit 160 Kbyte Kapazität. Die Hauptspeicher der PCs sind auf 16 bis 64 Kbyte ausgebaut. Der IBM-PC ist mit einem Basic-ROM und einem Kassettenrecorderausgang für die Datensicherung ausgestattet. Innerhalb der Version 1.0 wurden im Laufe der Zeit einige Ergänzungen vorgenommen, ohne die Versionsnummer zu ändern.

DOS 1.1

Die neue Version 1.1 unterstützte dann bereits doppelseitige Disketten mit 320 Kbyte Kapazität. Zusätzlich wird die serielle Schnittstelle mit unterstützt. Die Systemstruktur lehnt sich noch stark an CP/M 86 an.

DOS 2.0

Im März 1983 stellte IBM dann den IBM-PC/XT vor. Einhergehend mit dieser Entwicklung wurde auch eine neue DOS-Version eingeführt. Neben der Unterstützung der 10-Mbyte-Festplatte ist die Implementierung des hierarchischen Filesystems zu vermerken. Weitere Neuerungen sind die Möglichkeit zur Ein-/Ausgabeumleitung sowie das DOS-FILTER- und PIPE-Konzept. Damit werden einige der Ideen von UNIX übernommen. Mit PRINT wurde auch die Möglichkeit des Ausdrucks im Hintergrund geschaffen.

DOS 2.1

Die Version 2.1 wurde ab März 1983 von IBM mit dem PC/XT ausgeliefert. Damit werden erstmals auch länderspezifische Eigenschaften, wie Umlaute, Datums- und Zeitformat, sowie das Währungssymbol unterstützt. Der Hauptspeicher ist mittlerweile bis auf 128 Kbyte ausgebaut. An die Hersteller kompatibler PCs wird die Version 2.11 ausgeliefert, die in mehr als 60 Sprachen übersetzt wird. Sie trägt erheblich zur Verbreitung von MS-DOS bei.

Weitere Unterversionen werden im Laufe der Zeit angeboten. So werden OEMs im südostasiatischen Raum mit DOS 2.25 beliefert. Diese Version unterstützt insbesondere die japanischen (Kanji) und koreanischen Schriftzeichen. Da die Version 3.0 bereits vorhanden ist, werden die Programme ASSIGN, LABEL, DEBUG, SORT und EDLIN übernommen. Zusätzlich sind einige Fehler der 2.11-Version behoben.

DOS 3.0

Der nächste größere Versionssprung kommt mit der Einführung des IBM-PC/AT im August 1984. Diese unterstützt erstmalig die 1,2-Mbyte-Diskettenlaufwerke sowie größere Platten. Zusätzlich kommen einige neue Befehle hinzu.

DOS 3.1

Offensichtlich stand die Firma Microsoft bei der Entwicklung von DOS 3.0 stark unter Zeitdruck, denn bereits drei Monate später (November 1984) wird die Version 3.1 vorgestellt. Sie erlaubt den Zugriff der Benutzer auf Drucker, Verzeichnisse und Dateien innerhalb eines Netzwerkes, besitzt aber sonst den gleichen Leistungsumfang wie DOS 3.0.

DOS 3.2

Die nächste Version wird erst ab März 1986 ausgeliefert und erlaubt es, auch 3,5-Zoll-Diskettenlaufwerke zu benutzen. Weiterhin kommen einige neue Befehle (XCOPY, etc.) hinzu. So läßt sich ab DOS 3.2 zum Beispiel die Größe des Environments mit dem SHELL Befehl verändern.

DOS 3.3

Im Laufe des Jahres 1987 wird die DOS-Version 3.3 vorgestellt. Die IBM-Variante unterstützt neben dem PC auch die PS/2-Systeme. Neben einigen neuen Befehlen wird die Anbindung der Festplatten verbessert. So lassen sich erstmalig Platten größer als 32 Mbyte in mehrere logische Teile gliedern. Der FASTOPEN-Befehl hält die Informationen über den Zugriffspfad zu einer Datei im Speicher, so daß das Öffnen einer Datei sowie der Zugriff auf die Daten wesentlich schneller abläuft. Der Speicherbedarf für das DOS-Kernsystem wird von 44 Kbyte (DOS 3.2) auf 42 Kbyte reduziert.

DOS 4.0

Im Herbst 1988 kündigte IBM die neue Version des Betriebssystems PC-DOS 4.0 an. Diese Version wurde mit einer neuen Benutzeroberfläche (Shell) versehen und konnte erstmals Festplatten mit mehr als 32 Mbyte Speicher unterstützen. Die Version war aber so fehlerhaft und unkompatibel zu den früheren Versionen, daß bereits zwei Monate später die Version 4.01 nachgereicht wurde. Microsoft stellte diese Version zirka sechs Monate nach der Erstauslieferung an IBM auch als MS-DOS 4.01 zur Verfügung. Diese Version unterstützt unter anderem die Speichererweiterungen Extended Memory und Expanded Memory. Weiterhin ist die Behandlung multinationaler Zeichensätze stark verbessert worden.

DOS 5.0

Im Juli 1991 wurde dann MS-DOS 5.0 durch Microsoft vorgestellt. Das Betriebssystem besitzt erstmals die Möglichkeit zur Nutzung des Speichers zwischen 640 Kbyte und 1 Mbyte. Weiterhin wurden verschiedene Hilfsprogramme zur Restaurierung gelöschter

Dateien beigestellt. Mit QBASIC steht eine abgespeckte Version von Quick Basic zur Verfügung. Weiterhin bieten alle DOS-Befehle eine Online-Hilfe.

DOS 6.0

Ab April 93 ist die Version 6.0 von MS-DOS verfügbar. Aus Sicht des Programmierer hat sich mit dieser Version gegenüber DOS 5.0 nicht viel geändert. Die Endanwender werden aber mit einer Reihe neue Tools (UNDELETE, ANTIVIRUS, etc.) versorgt.

Mit WINDOWS 3.1 hat Microsoft eine erfolgreiche graphische Benutzeroberfläche als Erweiterung zu DOS. Wie die Entwicklung weiterhin verläuft, wird sich zukünftig zeigen.

Versionsspezifische Unterschiede in DOS

Die EXEC-Funktion findet sich bei DOS 2.1 noch im transienten Teil von COMMAND.COM, während sie ab DOS 3.0 im residenten Teil liegt. Die einzelnen DOS-Versionen besitzen einen unterschiedlichen Speicherbedarf. Während DOS 2.1 noch mit 17 Kbyte auskommt, belegt DOS 3.0-3.1 minimal zirka 23 Kbyte im Hauptspeicher. Von DOS 3.2 mit seinen 44 Kbyte wird der Kern bei DOS 3.3 auf 42 Kbyte reduziert. Ab DOS 5.0 können Teile von DOS in den HMA- und UMB-Bereich ausgelagert werden.

Device-Treiber

Das Attributfeld (Bit 6) ist nur in DOS 3.2 und DOS 3.3 definiert. Auswechselbare Speichermedien (Bit 11) werden ab DOS 3.x unterstützt. Die Kommandos 13, 14, 15 sind ab DOS 3.x implementiert, während der Code 19 nur ab DOS 3.2 gültig ist. Im Statusfeld werden die Fehlercodes 0DH bis 0FH erst ab DOS 3.0 zurückgegeben. Für die 1,2-Mbyte-Disketten wird ab DOS 3.0 das Media Descriptorbyte F9H eingeführt. Weiterhin wird ab DOS 3.0 ein 4-Byte-Adreßzeiger bei I/O-Aufrufen auf den Volume-Identifikations-Text zurückgegeben, falls das Medium unerlaubt gewechselt wurde.

Zahl der offenen Handles (Dateien)

Die DOS-Versionen bis 3.2 lassen maximal bis zu 20 offene Dateien zu. Ab DOS 3.3 läßt sich dieser Wert zwischen 20 und 64 Kbyte einstellen. Ab DOS 5.0 lassen sich bis zu 255 Handles einstellen.

Interrupts

Die Version 2.1 unterstützt die Interrupts 20H bis 27H und reserviert die Interrupts 28H bis 3FH. Die DOS-Versionen 3.X unterstützen die Interrupts 20H bis 2FH und reservieren die Interrupts 30H bis 3FH.

Netzwerkunterstützung

Netzwerkdienste werden erst ab der Version 3.1 unterstützt.

Extended-DOS-Partition

Die Verwaltung von Festplattenkapazitäten über 32 Mbyte ist ab DOS 3.3 möglich, indem die Platte in mehrere logische Teile aufgegliedert wird. Ab DOS 4.0 wurde die maximal unterstützte Plattengröße von 32 Mbyte auf 4 Gbyte erhöht.

Funktionsaufrufe (INT 21)

Je nach DOS-Version werden folgende Funktionsaufrufe des INT 21 unterstützt:

Code	DOS	Funktion
00	2.0 - 6.0	Terminate Program
01	2.0 - 6.0	Read Keyboard and Echo
02	2.0 - 6.0	Display Character
03	2.0 - 6.0	Auxiliary Input
04	2.0 - 6.0	Auxiliary Output
05	2.0 - 6.0	Print Character
06	2.0 - 6.0	Direct Console I/O
07	2.0 - 6.0	Direct Console Input
08	2.0 - 6.0	Read Keyboard
09	2.0 - 6.0	Display String
0A	2.0 - 6.0	Buffered Keyboard Input
0B	2.0 - 6.0	Check Keyboard Status
0C	2.0 - 6.0	Flush Buffer, Read Keyboard
0D	2.0 - 6.0	Reset Disk
0E	2.0 - 6.0	Select Disk
0F	2.0 - 6.0	Open File
10	2.0 - 6.0	Close File
11	2.0 - 6.0	Search for First Entry
12	2.0 - 6.0	Search for Next Entry
13	2.0 - 6.0	Delete File
14	2.0 - 6.0	Sequential Read
15	2.0 - 6.0	Sequential Write
16	2.0 - 6.0	Create File
17	2.0 - 6.0	Rename File
19	2.0 - 6.0	Get Current Disk
1A	2.0 - 6.0	Set Disk Transfer Address
1B	2.0 - 6.0	Get Default Drive Data
1C	2.0 - 6.0	Get Drive Data
1F	2.0 - 6.0	Get Default Disk Parameter Block
21	2.0 - 6.0	Random Read
22	2.0 - 6.0	Random Write
23	2.0 - 6.0	Get File Size
24	2.0 - 6.0	Set Relativ Record
25	2.0 - 6.0	Set Interrupt Vector
26	2.0 - 6.0	Create New PSP
27	2.0 - 6.0	Random Block Read
28	2.0 - 6.0	Random Block Write
29	2.0 - 6.0	Parse File Name
2A	2.0 - 6.0	Get Date
2B	2.0 - 6.0	Set Date
2C	2.0 - 6.0	Get Time
2D	2.0 - 6.0	Set Time
2E	2.0 - 6.0	Set/Get Verify Flag
2F	2.0 - 6.0	Get Disk Transfer Address
30	2.0 - 6.0	Get MS-DOS Version Number
31	2.0 - 6.0	Keep Process
32	2.0 - 6.0	Get Disk Parameter Block
33	2.0 - 6.0	Control-C Check
34	2.0 - 6.0	Get DOS Critical Interval Flag
35	2.0 - 6.0	Get Interrupt Vektor
36	2.0 - 6.0	Get Free Disk Space
37	2.0 - 6.0	Set Switch Character
38	2.0 - 6.0	Get Country Data
38	3.0 - 6.0	Set Country Data
39	2.0 - 6.0	Create Directory
3A	2.0 - 6.0	Remove Directory
3B	2.0 - 6.0	Change Current Directory
3C	2.0 - 6.0	Create Handle
3D	2.0 - 6.0	Open Handle
3E	2.0 - 6.0	Close Handle
3F	2.0 - 6.0	Read from a File or Device
40	2.0 - 6.0	Write to a File or Device
41	2.0 - 6.0	Delete (Unlink) Directory Entry
42	2.0 - 6.0	Move File Pointer
43	2.0 - 6.0	Get/Set File Attributes
4400	2.0 - 6.0	IOCTL Data
4402	2.0 - 6.0	IOCTL Character
4404	2.0 - 6.0	IOCTL Block
4406	2.0 - 6.0	IOCTL Status
4408	3.0 - 6.0	IOCTL is Changeable
4409	3.1 - 6.0	IOCTL is Redirected Block
440A	3.1 - 6.0	IOCTL is Redirected Handle
440B	3.0 - 6.0	IOCTL Retry
440C	3.3 - 6.0	Generic IOCTL Handle Request
440D	3.2 - 6.0	Generic IOCTL Request
440E	3.2 - 6.0	I/O Control for Device
440F	3.2 - 6.0	Set Logical Drive

45	2.0 - 6.0	Duplicate File Handle
46	2.0 - 6.0	Force Duplicate File Handle
47	2.0 - 6.0	Get Current Directory
48	2.0 - 6.0	Allocate Memory
49	2.0 - 6.0	Free Allocated Memory
4A	2.0 - 6.0	Set Block
4B00	2.0 - 6.0	Load and Execute
4B03	2.0 - 6.0	Load Overlay
4C	2.0 - 6.0	Terminate a Process
4D	2.0 - 6.0	Get Returncode
4E	2.0 - 6.0	Find First Matching File
4F	2.0 - 6.0	Find Next File
50	2.0 - 6.0	Set Active PSP
51	2.0 - 6.0	Get Active PSP
52	2.0 - 6.0	Get DOS Data Address
53	2.0 - 6.0	Set Disk Parameter Block
54	2.0 - 6.0	Get Verify State
56	2.0 - 6.0	Rename File
57	2.0 - 6.0	Get/Set File Date and Time
58	3.0 - 6.0	Get/Set Allocation Strategy
59	3.0 - 6.0	Get Extended Error
5A	3.0 - 6.0	Create Temporary File
5B	3.0 - 6.0	Create New File
5C	3.0 - 6.0	Lock/Unlock File Access
5D	3.1	Indirect Function Call
5E00	3.1 - 6.0	Get Machine Name
5E02	3.1 - 6.0	Set Printer Setup
5E03	3.1 - 6.0	Get Printer Setup
5F02	3.1 - 6.0	Get Redirection List Entry
5F03	3.1 - 6.0	Redirect Device
5F04	3.1 - 6.0	Cancel Redirection
62	3.0 - 6.0	Get Program Segment Prefix Address
63	2.25, 6.0	Get Lead Byte Table, DCBS Support
65	3.3 - 6.0	Get Extended Country Information
66	3.3 - 6.0	Get Global Code Page
66	3.3 - 6.0	Set Global Code Page
67	3.3 - 6.0	Set Handle Count
68	3.3 - 6.0	Commit File
69	4.0 - 6.0	Get/Set Disk Serial Number
6C	4.0 - 6.0	Extended Open/Create

Tabelle 4.3: INT 21-Funktionsaufrufe (Ende)

Fehlerbehandlung

Ab DOS 3.0 lassen sich die Extended Errorcodes abfragen. Die Fehlercodes werden von DOS-Version zu DOS-Version erweitert. In DOS 2.1 gelten alle Fehlermeldungen von 1 bis 18, während ab DOS 3.X zusätzliche Codes hinzukommen.

FAT-Tabelle

DOS 2.1 bis 3.3 benutzt 12-Bit-File-Allocation-Tabellen für Disketten, während die 16-Bit-FATs für die Festplatten erst ab Version 3.0 unterstützt werden.

COMMAND.COM Intern

Zum Abschluß möchte ich nachfolgend einige Tricks und Internas bezüglich des DOS-Kommandoprozessors COMMAND.COM dokumentieren.

Die internen DOS-Befehle

Der DOS-Kommandoprozessor unterscheidet zwischen den internen Befehlen wie DIR, DEL, etc. und externen Befehlen wie FORMAT, etc. Externe Befehle müssen in Form einer lauffähigen COM- oder EXE-Datei auf einem Speichermedium vorhanden sein.

Diese Dateien werden dann geladen und als Programm ausgeführt. Bei den internen Befehlen befindet sich der Code in COMMAND.COM, so daß der Kommandoprozessor nur einen Sprung auf die interne Routine ausführt.

In MS-DOS und PC-DOS lassen sich zwar die landesspezifischen Zeichensätze selektieren, und auch die Fehlermeldungen erfolgen in verschiedenen Sprachen. Beim Befehlssatz sind aber nach wie vor die englischen Kommandos einzugeben. Der Kommandoprozessor enthält intern nichtdokumentierte Tabellen, in denen auch die Befehle mit abgelegt sind. Die Befehlstabelle besitzt folgende Struktur:

```
Ö--Ú--------------Ú--Ú--Ú--Ì
ÛÚ-Ù---Ú----------ÙÚ-Ù-ÚÙ-Úì
 °     °           °   ÛÚ-ì
 °     °           °    Û-  Offset der Befehls-Routine
 °     °          Û-————   Status
 °     Û-————————————————   Befehl in ASCII-Zeichen
 Û--————————————————————   Länge des Befehlstextes
```

Das erste Byte eines Eintrages gibt an, wie viele ASCII-Zeichen der nachfolgende Befehl enthält. An dieses Byte schließt sich der Befehl im Klartext als ASCII-Zeichenkette (z.B. DIR) an. Die Bedeutung des Statusbytes ist nicht ganz klar. Anschließend folgt ein 2-Byte-Offset zur entsprechenden Routine in COMMAND.COM. Da der Kommandoprozessor als COM-File vorliegt, läßt er sich leicht mit DEBUG patchen.

Um nun den Namen eines internen Befehls zu ändern, ist lediglich der Eintrag in obiger Tabelle zu modifizieren. Es darf jedoch nur das Textfeld verändert werden. Der Patch sollte dabei immer auf einer Sicherungskopie erfolgen, um das Originalprogramm nicht zu zerstören. Nachfolgend wird ein Ausschnitt aus dieser Befehlstabelle gezeigt.

```
45 58 49 53 54 87 08 00-03 44 49 52 03 0D 0D 06
 E  X  I  S  T  .  .  .  .  D  I  R  .  .  .  .
52 45 4E 41 4D 45 01 3C-10 03 52 45 4E 01 3C 10
 R  E  N  A  M  E  .  <  .  .  R  E  N  .  <  .
05 45 52 41 53 45 01 E4-0F 03 44 45 4C 01 E4 0F
 .  E  R  A  S  E  .  .  .  .  D  E  L  .  .  .
04 54 59 50 45 01 B4 10-03 52 45 4D 02 04 01 04
 .  T  Y  P  E  .  .  .  .  R  E  M  .  .  .  .
43 4F 50 59 03 01 27 05-50 41 55 53 45 02 D7 0F
 C  O  P  Y  .  .  '  .  P  A  U  S  E  .  .  .
04 44 41 54 45 02 5B 1E-04 54 49 4D 45 00 5E 1F
 .  D  A  T  E  .  [  .  .  T  I  M  E  .  ^  .
       . . .
       . . .
47 4F 54 4F 02 69 09 05-53 48 49 46 54 02 22 09
 G  O  T  O  .  i  .  .  S  H  I  F  T  .  "  .
02 49 46 02 E0 07 03 46-4F 52 02 90 0B 03 43 4C
 .  I  F  .  .  .  .  F  O  R  .  .  .  .  C  L
44 4F 2A 2D 2B 4C 6C 61-7A 44 43 53 58 2E 3F 2E
 D  O  *  -  +  L  l  a  z  D  C  S  X  .  ?  .
2C 3A 2E 70 6A 6E 2E 43-4F 4D 2E 45 58 45 2E 42
 ,  :  .  p  j  n  .  C  O  M  .  E  X  E  .  B
41 54 56 42 41 50 57 20-00 00 00 00 00 00 00 00
 A  T  V  B  A  P  W  .  .  .  .  .  .  .  .  .
 |  |
 Û--Û- Extensionen für die Suche nach externen Befehlen
```

Änderung der Befehlspriorität

Der Kommandointerpreter bearbeitet eine Kommandoeingabe immer nach einem festen Schema. Wird ein Befehl eingegeben, prüft COMMAND.COM zuerst, ob es sich um einen internen Befehl handelt. Erst dann wird nach COM-, EXE- und BAT-Dateien gesucht. Dabei gilt immer folgende Reihenfolge, wobei die internen Befehle die höchste Priorität besitzen:

_ interne Befehle

_ COM-Dateien

_ EXE-Dateien

_ BAT-Dateien

An die Tabelle mit den Kommandos schließen sich auch die Namen der Dateiextensionen an (s. DUMP-Ausgabe). Durch Änderung des Textes COM.EXE.BAT läßt sich die Priorität der Befehlsabarbeitung verändern. Weiterhin kommt der Name der Datei AUTOEXEC innerhalb des Codes im Klartext vor. Durch einen Patch läßt sich dieser Name ebenfalls ändern.

Änderung der DOS-Farbeinstellung

DOS benutzt für die Textausgaben immer eine Schwarzweißdarstellung. Nur mit den ANSI-Treibern lassen sich Farben auf der DOS-Ebene nutzen. Mit einem geeigneten Patch läßt sich dies allerdings auch auf der DOS-Oberfläche beheben. Textausgaben und die Selektion der Farben erfolgen in COMMAND.COM durch Aufruf des INT 10H. Deshalb läßt sich der COM-File mittels DEBUG auf die Zeichenkombination

```
CD 10
```

untersuchen. Dies kann mit der Anweisung

```
s 100 7000 CD,10
```

erfolgen. DEBUG wird in der Regel mehrere Einsprünge in den INT 10 melden. An einer Stelle wird sich folgende Programmsequenz befinden:

```
MOV DL,AH
DEC DL
MOV DH,18
XOR AX,AX
MOV CX,AX
MOV BX,0700
MOV AH,06
INT 10
```

Hier benutzt COMMAND.COM den INT 10 zum Bildschirmscroll. In BX wird dabei das Farbattribut für die einzuscrollende Leerzeile (hier 07) übergeben. Durch Austausch dieses Attributes lassen sich in DOS auch farbige Ausgaben erzeugen. Die folgende Tabelle gibt mögliche Kombinationen wieder:

Hinter-	Vordergrundfarbe							
grundf.	Schwarz	Blau	Grün	Zyan	Rot	Magenta	Braun	Weiß
Schwarz	00	01	02	03	04	05	06	07
Blau	10	11	12	13	14	15	16	17
Grün	20	21	22	23	24	25	26	27
Zyan	30	31	32	33	34	35	36	37
Rot	40	41	42	43	44	45	46	47
Magenta	50	51	52	53	54	55	56	57
Braun	60	61	62	63	64	65	66	67
Weiß	70	71	72	73	74	75	76	77

Die erste Spalte enthält die Hintergrundfarben (Schwarz, Blau, etc.). Durch Kombination von Hintergrund- und Vordergrundfarbe erhält man den gewünschten Code in der Matrix. Die Einstellung der Farbe wird allerdings immer nur dann geändert, wenn ein Bildscroll

eine neue Zeile einblendet. Für diese Zeile gilt dann das neue Farbattribut. Bei der Anweisung CLS bezieht sich das Attribut allerdings auf den kompletten Bildschirm.

Änderung der ECHO-Einstellung

Beim Start eines Batchprogrammes ist die ECHO-Funktion in der Regel auf ON gestellt. Ab DOS 3.3 existiert allerdings die Möglichkeit, die Ausgabe eines Echos durch das @-Zeichen zu unterdrücken. Mit der Anweisung @ECHO OFF läßt sich auch die Ausgabe der ersten Anweisung einer Batchdatei unterdrücken. Für die früheren DOS-Versionen ist dies nicht möglich. Mit einem Trick kann das Problem jedoch umgangen werden. Der Kommandoprozessor speichert den Zustand des ECHO-Flags in einem internen Datenbereich, der vor jedem Befehl abgefragt wird. Diese Abfrage besitzt dabei in etwa folgendes Format:

```
 .
MOV AL,[xxx]
AND AL,01
PUSH AX
 .
```

Mit der ersten Anweisung wird der Inhalt des Flags in AL gelesen und anschließend mit der Konstanten 01H (ECHO ON) verglichen. Das Ergebnis bestimmt, ob der Befehl auf der Console gespiegelt wird. Durch Änderung der Abfrage in:

```
 .
MOV AL,[xxx]
AND AL,00
PUSH AX
 .
```

ist das Echo standardmäßig ausschaltbar. Die Befehle

```
AND AL,01H
PUSH AX
```

besitzen folgende Codefolge

```
24 01 50
```

und lassen sich mit der DEBUG-Anweisung

```
S 100 5000 24 01 50
```

suchen. Anschließend ist das Byte 01H auf 00H zu ändern, da dann der Zustand ECHO OFF standardmäßig eingestellt wird.

Die Ermittlung des Masterenvironments

Eine weitere interessante Option ist die Veränderung von Batchvariablen durch Anwenderprogramme. Eine Batchvariable ist immer in Prozentzeichen (%NAME%) einzuschließen, wobei der Name in Großbuchstaben geschrieben wird. Im Batchprogramm läßt sich der Inhalt der Variablen leicht auswerten:

```
IF %NAME% == Hallo GOTO Exit
```

In vielen Batchapplikationen ist es nun interessant, eine Batchvariable interaktiv von der Benutzeroberfläche zu verändern (6). Für diesen Zweck bietet es sich an, ein kleines Programm z.B. in Pascal zu schreiben, welches Benutzereingaben liest und in die Variable

zurückschreibt. Die Batchvariablen werden im Environmentbereich als ASCIIZ-Text mit dem Format:

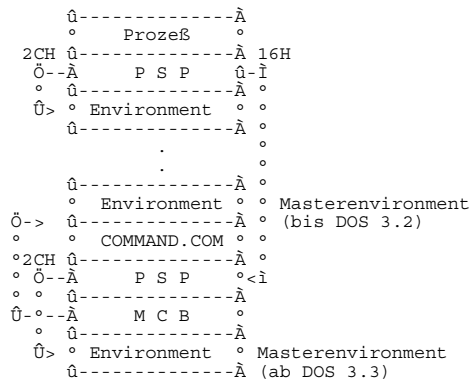
```
NAME=Text,00
```

geführt. Wie gelangt nun aber das Programm an die Adresse des Environments. Im PSP eines jeden Prozesses steht ab Offset 2CH die Segmentadresse des zugehörigen Environments. Eine Veränderung im eigenen Environment bringt aber nichts, da das Environment nur so lange besteht, wie der Prozeß installiert ist. Anschließend wird die Kopie des Environments gelöscht.

Um den Inhalt einer Batchvariablen dauerhaft zu modifizieren, läßt sich der SET-Befehl auf der Anwenderoberfläche benutzen. Dieser Befehl wirkt aber nicht in Batchprogrammen. Ein Aufruf des SET-Kommandos aus dem Anwenderprogramm per EXEC-CALL funktioniert auch nicht, da DOS dann eine zweite Kopie von COMMAND.COM im Speicher anlegt und damit das Environment dieser Kopie modifiziert. Diese Kopie wird aber nach der Rückkehr in den Anwenderprozeß wieder gelöscht.

Eine Alternative bietet die Verwendung des undokumentierten INT 2E zur Aktivierung des SET-Befehls. Damit läßt sich dann das Masterenvironment modifizieren. Wird diese Technik allerdings innerhalb einer Batchdatei angewendet, ist die Modifikation erst nach Beendigung der Batchdatei gültig. Deshalb muß eine weitere Alternative zum Zugriff auf das Masterenvironment gefunden werden. Hier gibt es wieder einige Tricks, die die Lage des Masterenvironments verraten.

Im PSP eines Prozesses steht im undokumentierten Bereich ab Offset 16H die Segmentadresse des Programmladers. Hier handelt es sich in der Regel um COMMAND.COM. Nur bei der Aktivierung des Prozesses per Debugger oder Entwicklungsumgebung stimmt dies nicht mehr. Der Lader (COMMAND.COM) verfügt seinerseits wieder über ein eigenes PSP, in dem ab Offset 2CH ein Zeiger auf die Segmentadresse des Masterenvironments stehen sollte. Die gilt aber erst ab DOS 3.3, während bis DOS 3.2 im entsprechenden Feld der Wert 00 00 steht.



Bei den früheren Versionen läßt sich die Lage des Masterenvironments daher nicht aus der PSP-Offsetadresse 2CH von COMMAND.COM ablesen. Vielmehr muß die Lage des MCB für den PSP bestimmt werden. Dieser MCB liegt ein Paragraph unterhalb des PSP-Anfangs. Der Wert ab Offset 16H im PSP des Anwenderprozesses ist also nur um den Wert

1 zu erniedrigen. Im MCB steht ab Offset 03H die Länge des Blockes (COMMAND.COM). Im Anschluß an COMMAND.COM steht dann ein MCB mit dem zugehörigen Environment. Hierbei handelt es sich dann um das Masterenvironment. Der Zugriff auf das Masterenvironment ist in (2) dargestellt.

Variation der Environmentgröße

Im Abschnitt 6.4 wurden Aufbau und Größe des DOS-Environments besprochen. Ab DOS 2.0 bis einschließlich 3.0 reserviert der Kommandoprozessor (COMMAND.COM) jeweils 160 Byte für das Environment.

Ab DOS 3.2 läßt sich die Größe mittels des SHELL-Befehls auf bis zu 32 Kbyte erweitern. Die gewünschte Größe läßt sich dabei direkt in Bytes (/E:xxxx) angeben.

DOS 3.1 bietet meist die undokumentierte Möglichkeit, die Größe beim Booten durch den SHELL-Befehl zu variieren. Dabei wird der Wert nicht in Bytes, sondern in Paragraphen (16-Byte-Blöcke) erwartet.

In den älteren DOS-Versionen bleibt allerdings die feste Grenze von 160 Byte, ein Relikt aus der Zeit als die PCs noch mit 64 Kbyte Speicherausbau geliefert wurden. Leider muß der Environmentbereich in einem zusammenhängenden Speicherblock vorliegen, so daß auch die Möglichkeit einer einfachen Erweiterung entfällt. Da weiterhin sich meist der MCB des PSP-Bereiches an das Environment anschließt, ist eine Vergrößerung durch den Anwenderprozeß selten möglich.

Es gibt allerdings Tricks, um diese Limitierung zumindest teilweise zu umgehen.

Einmal legt DOS die Segmentadresse des Environments im PSP-Bereich ab 2CH ab. Ein Anwenderprozeß kann sich nun einen eigenen Speicherbereich einrichten, den Inhalt des alten Environments in diesen Bereich kopieren, und die neue Adresse im PSP ablegen. Damit besitzt zumindest dieser Prozeß die Möglichkeit, den Environmentbereich auf beliebige Größen zu setzen. Leider bleiben die Größen der anderen Environments auf 160 Byte begrenzt und die SET-Anweisungen wirken sich auch nicht auf das neue Environment aus. Abhilfe kann ein residentes Programm schaffen, welches einmal alle Environmentbereiche verlagert und andererseits den SET-Befehl emuliert. Alles in allem aber kein optimales Vorgehen.

Aus diesem Grunde soll hier auf eine weitere Methode eingegangen werden. Im Grunde ist der Kommandoprozessor (COMMAND.COM) für die Reservierung des Environmentbereiches zuständig. Der SHELL-Befehl modifiziert ab DOS 3.1 deshalb auch die internen Einstellungen des Kommandoprozessors. In älteren Versionen ist er aber nicht implementiert. Nun tritt aber durchaus das Problem auf, daß neuere Softwareprodukte (Compiler, etc.) verschiedene Steuerinformationen über die Lage bestimmter Dateien über das Environment beziehen. Da ein Pfad bereits bis zu 65 Zeichen lang sein darf, reicht die Environmentgröße von 160 Byte meist nicht aus. Dies würde bedeuten, daß solche Produkte nicht auf den früheren DOS-Versionen laufen. Da Microsoft auch Compiler liefert, war eine Lösung absehbar. Bei neueren Produkten (z.B. MS-FORTRAN 4.0) findet sich deshalb ein kleines Hilfsmodul (SETENV.EXE), welches COMMAND.COM so modifiziert, daß auch größere Environmentbereiche möglich sind. Wer dieses Programm nicht besitzt, kann versuchen, diese Patches manuell durchzuführen. Hierzu reicht der DOS-Debugger (DEBUG.COM) sowie eine Diskettenkopie mit COMMAND.COM. Diese wird dann mit dem Befehl:

```
DEBUG A:COMMAND.COM
-r
```

geladen. Mit dem Befehl r läßt sich die Größe von COMMAND.COM ermitteln, da diese im Register BX:CX steht. Nun stellt sich noch die Frage, welche Adressen zu patchen sind. Hier hilft die Überlegung weiter, daß auch der Kommandoprozessor die INT 21-Funktion 48H zur Speicherreservierung benutzen muß. Es wird also die Sequenz:

```
Code      Anweisungen
            .
BB xx xx  MOV  BX,Size
B4 48     MOV  AH,48H
CD 21     INT  21
```

auftreten. Mittels des Debug-Suchbefehls

```
-s 100 L xxxxx  BB 0A 00 B4 48 CD 21
xxxx:xxxx
```

können dann die entsprechenden Stellen im Code gesucht werden. Die Länge des zu durchsuchenden Bereichs (L xxxxx) ist abhängig von der Codelänge, die im Register BX:CX beim r-Befehl ermittelt wurde. Falls keine solche Sequenz gefunden wird, nicht immer reserviert COMMAND einen Bereich von 00A0H-Paragraphen (160 Byte), müssen alle Aufrufe der Funktion 48H untersucht werden. In der Regel läßt sich dann die gesuchte Stelle finden und modifizieren. Die neue Größe sollte dabei experimentell ermittelt werden.

In diesem Zusammenhang ist noch eine Beobachtung interessant, wie COMMAND die Größe des freien Speichers ermittelt. Es wird ein Bereich von 0FFFFH-Paragraphen über die Funktion 48H angefordert. Da dies 1 Mbyte entspricht, gibt die Funktion immer einen Fehlercode im Register AX zurück, ohne Speicher zu reservieren. Gleichzeitig findet sich in BX die Zahl der Paragraphen des größten zusammenhängenden Speicherblocks. Damit läßt sich elegant die Größe des freien Speichers feststellen.

Der Keyboard-Interrupt (INT 9H)

Diese Routine wird bei jedem Tastendruck aktiviert. Sie übersetzt die Tastencodes und legt sie im BIOS-Datenbereich in den Tastaturpuffer ab. Weiterhin setzt die Routine verschiedene Tastatur- und Statuscodes im BIOS-Datenbereich.

```
Ö-------------------------------------Ì
º Status Byte 1 Adresse  0:417        º
û-------------------------------------À
º  0   right shift key depressed      º
º  1   left shift key depressed       º
º  2   control key depressed          º
º  3   alt key depressed              º
º  4   Scroll Lock state toggled      º
º  5   Num Lock state toggled         º
º  6   Caps Lock state toggled        º
º  7   insert state is active         º
û-------------------------------------À
º Status Byte 2 Adresse  0:418        º
û-------------------------------------À
º  0   left control key depressed     º
º  1   left alt key depressed         º
º  2   SysReq key depressed           º
º  3   Pause key has been toggled     º
º  4   ScrollLock key is depressed    º
º  5   NumLock key is depressed       º
º  6   CapsLock key is depressed      º
º  7   Insert key is depressed        º
û-------------------------------------À
º Status Byte 3 Adresse  0:496 (AT)   º
û-------------------------------------À
º  0   last code = E1h hidden code    º
º  1   last code = E0h hidden code    º
º  2   right control key down         º
º  3   right alt key down             º
º  4   101 key Enhanced keyboard inst.º
º  5   force NumLock if rd ID & kbx   º
º  6   last char = 1. ID character    º
º  7   doing a read ID (must be bit 0)º
û-------------------------------------À
º Status Byte 4 Adresse  0:497 (AT)   º
û-------------------------------------À
º  0   ScrollLock indicator           º
º  1   NumLock indicator              º
º  2   CapsLock indicator             º
º  3   circus system indicator        º
º  4   ACK received                   º
º  5   resend received flag           º
º  6   mode indicator update          º
º  7   keyboard transmit error flag   º
Û-------------------------------------ì
```

Diese Statusbytes lassen sich über verschiedene BIOS-Routinen abfragen.

Auf den AT-Rechnern wird bei jedem INT 9 zusätzlich ein INT 15H mit der Funktion AH=4FH ausgelöst, der normalerweise durch das BIOS abgefangen ist. Weiterhin wird bei AT-Rechnern ein INT 15H mit dem Code AH=85H ausgelöst, falls die SysReq-Taste gedrückt wurde.

Das TesSeRact Interface (INT 2F)

Mit der Funktion 54H des INT 2F wird ein einheitliches Kommunikationsinterface TeSeRact für Terminate Stay Resident (TSR) Programme geboten. Borlands THELP.COM unterstützt zum Beispiel dieses Interface.

Hierbei gilt folgende Aufrufschnittstelle.

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AX:  5453H (TeSeRact)          º
º BX:  Unterfunktion             º
û--------------------------------À
º      RETURN                    º
º ---                            º
Û--------------------------------ì
```

Im Register AX ist immer der Wert 5453H zu übergeben. Die Funktion besitzt verschiedene Unterfunktionen, die sich über einen Code im Register BX aktivieren lassen.

Installation Check (BX = 00H)

Die Funktion erlaubt an TeSeRact die Abfrage, ob ein bestimmtes Programm bereits installiert ist. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AX:  5453H (TeSeRact)          º
º BX:  0 (Get Install Status)    º
º DS:SI Name (8 Zeichen)         º
û--------------------------------À
º      RETURN                    º
º AX: FFFFH installiert          º
º CX: ID Nummer                  º
º AX: <> FFFFH                   º
º CX: ID Nummer                  º
Û--------------------------------ì
```

Im Register BX ist der Code 00H zur Installationsabfrage zu setzen. In DS:SI erwartet der Treiber einen Zeiger auf einen String mit dem Namen der zu installierenden Routine. Dieser Name ist acht Zeichen lang und an unbelegten Stellen mit Blanks aufzufüllen. Nach dem Aufruf gibt AX den Installationsstatus an.

Mit dem Wert FFFFH ist das residente Modul bereits installiert, und in CX steht die Identifikationsnummer des installierten Treibers. Nur falls AX <> FFFFH ist, kann das residente Programm installiert werden. In CX wird eine freie Nummer des INT 2F zurückgeliefert.

Get User Parameters (BX = 01H)

Mit diesem Aufruf läßt sich ein User- Parameterblock des residenten Programmes abfragen. Es gilt folgende Aufrufschnittstelle:

```
Ö--------------------------------Ì
º      CALL:  INT 2F             º
º                                º
º AX:  5453H (TeSeRact)          º
º BX:  01H (Get User Parameter)  º
º CX:  TSR ID Nummer             º
û--------------------------------À
º      RETURN                    º
º AX: 0000H ok                   º
º ES:BX Adresse Parameterblock   º
º AX: <> 0 Fehler                º
Û--------------------------------ì
```

In CX ist die ID-Nummer der betreffenden Routine zu übergeben. Diese wird bei der Installationsabfrage in CX zurückgegeben. Nach dem Aufruf enthält das Registerpaar ES:BX einen Zeiger auf den User-Parameterblock.

Check if Hotkey in use (BX = 02H)

Mit diesem Aufruf läßt sich prüfen, ob eine bestimmte Taste bereits als Hotkey für andere residente Programme belegt wurde. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  2 (Check Hotkey in use)    º
º CL:  Scan Code Hotkey           º
û---------------------------------À
º      RETURN                     º
º AX: FFFFH Hotkey belegt         º
º AX: <> FFFFH ok                 º
Û---------------------------------ì
```

Im Register CL ist der Scan-Code für die gewünschte Hotkey-Taste zu übergeben. Das Programm prüft dann, ob diese Taste bereits von anderen residenten Programmen belegt ist. In diesem Fall gibt die Funktion in AX den Wert FFFFH zurück. Damit sind Probleme im Zusammenspiel mit bereits installierten Programmen zu erwarten, falls das aktive Programm diese Taste auch benutzt. Alle anderen Werte in AX deuten darauf hin, daß die Taste nicht belegt ist.

Replace Default Critical Error Handler (BX = 03H)

Mit diesem Aufruf läßt sich der DOS-Critical-Error-Handler für die betreffende Routine ersetzen.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  3 (Replace Error Handler)  º
º CX:  TSR ID Nummer              º
º DS:SI Adresse neuer INT 24      º
û---------------------------------À
º      RETURN                     º
º AX: <> 00H Handler nicht        º
º           installierbar         º
Û---------------------------------ì
```

Im Register CX ist die Identifikationsnummer des residenten Prozesses zu übergeben. Die Funktion sucht dann die betreffende Routine und ersetzt den INT 24-Handler durch den im Registerpaar DS:SI angegebenen Handler. Enthält das Register AX nach dem Aufruf einen Wert ungleich 0, dann läßt sich der neue Handler nicht installieren. Dies ist zum Beispiel der Fall, wenn das residente Programm den *PSP-Bereich* freigegeben hat.

Get Internal Data Area (BX = 04H)

Die Funktion fragt den Inhalt eines internen Datenbereiches ab. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  4 (Get Internal Data Area) º
º CX:  TSR ID Nummer              º
û---------------------------------À
º      RETURN                     º
º AX: = 00H OK                    º
º ES:BX Adresse Datenbereich      º
Û---------------------------------ì
```

Im Register CX ist die Identifikationsnummer des residenten Prozesses zu übergeben. Die Funktion sucht dann die betreffende Routine und gibt im Registerpaar ES:BX die Adresse

der Tabelle zurück. Enthält das Register AX nach dem Aufruf einen Wert ungleich 0, dann wurde keine TSR-Routine unter dem ID-Code gefunden und die Inhalte von ES:BX sind ungültig. Der interne Datenbereich besitzt folgenden Aufbau:

Offs.	Bytes	Feld
00H	1	Revisionsnummer TesSeRact Library
01H	1	Typ des PopUp Menüs
02H	1	Zahl d. INT 08 seit dem letzten Aufruf
03H	1	Zahl d. INT 13 seit dem letzten Aufruf
04H	1	aktive Interrupts
05H	1	aktive Software-Interrupts
06H	1	DOS-Hauptversion
07H	1	Wartezeit vor einem PopUp
08H	4	Adresse INDOS-Flag
0CH	4	Adresse DOS-Critical-Error-Flag
10H	2	PSP-Segmentadresse des unterbrochenen Programmes
12H	2	PSP-Segmentadresse des durch d. INT 28 unterbrochenen Programmes
14H	4	DTA des unterbrochenen Programmes
18H	4	DTA des durch den INT 28 unterbrochenen Programmes
1CH	2	SS des unterbrochenen Programmes
1EH	2	SP des unterbrochenen Programmes
20H	2	SS des durch den INT 28 unterbrochenen Programmes
22H	2	SP des durch den INT 28 unterbrochenen Programmes
24H	4	INT 24 des unterbrochenen Programmes
28H	3 * 2	6 Byte DOS 3+ Extended Error Info
2EH	1	alte BREAK-Einstellung
2FH	1	alte VERIFY-Einstellung
30H	1	MS-WORD 4.0 wurde durch PopUp unterbr.
31H	1	MS-WORD 4.0 spezial PopUp-Flag
32H	1	Enhanced Keyboard Call aktiv
33H	1	Verzögerung für MS WORD 4.0
24H		11mal:
	4	alter Interruptvektor
	1	Interrupt-Nummer
	4	neuer Interruptvektor

Die Tabelle dient zur Speicherung verschiedener Daten, wie zum Beispiel die alten und neuen Interruptvektoren von 11 Interrupts.

Set multiple Hotkeys (BX = 05H)

Mit diesem Aufruf lassen sich mehrere Tasten als Hotkeys definieren.

```
CALL:  INT 2F

AX:  5453H (TeSeRact)
BX:  5 (Set Hotkeys)
CX:  TSR ID Nummer
DL:  Zahl der Hotkeys
DS:SI Adresse Hotkey Tabelle
```

```
RETURN

AX: = 00H OK
```

Im Register CX ist wieder der ID-Code der betreffenden Routine zu übergeben. DL enthält die Zahl der Hotkeys, die mit dem Aufruf gesetzt werden sollen. In DS:SI ist die Adresse einer Tabelle mit den Definitionen des Hotkeys zu übergeben. Die Tabelle besitzt folgenden Aufbau:

Bytes	Feld
1	Scan Code Hotkey
1	Shift state Hotkey
1	Flag-Wert, welcher an das TSR-Programm zu übergeben ist

Nur wenn nach dem Aufruf der Funktion der Inhalt des Registers AX = 0 ist, wurden die Hotkeys gesetzt. Andernfalls liegt ein Fehler vor.

Die Funktionscodes 06H - 0FH sind reserviert.

Enable TSR (BX = 10H)

Mit diesem Aufruf läßt sich ein installiertes TSR-Programm für eine Aktivierung per Hotkey freigeben (Enable).

```
CALL:  INT 2F

AX:  5453H (TeSeRact)
BX:  10 (Enable TSR)
CX:  TSR-ID-Nummer

RETURN
AX: = 00H OK
```

In CX ist die ID-Nummer zu übergeben. Werte ungleich 0 in AX zeigen nach dem Aufruf an, daß sich die betreffende TSR-Routine nicht freigeben läßt.

Disable TSR (BX = 11H)

Mit diesem Aufruf läßt sich ein installiertes TSR-Programm für eine Aktivierung per Hotkey sperren (Disable).

```
CALL:  INT 2F

AX:  5453H (TeSeRact)
BX:  11 (Disable TSR)
CX:  TSR-ID-Nummer

RETURN
AX: = 00H OK
```

In CX ist die ID-Nummer zu übergeben. Werte ungleich 0 in AX zeigen nach dem Aufruf an, daß sich die betreffende TSR-Routine nicht sperren läßt.

Unload TSR (BX = 12H)

Mit diesem Programm läßt sich ein installiertes TSR-Programm desinstallieren. Es gilt folgende Aufrufschnittstelle:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  12 (Unload TSR)            º
º CX:  TSR-ID-Nummer              º
û---------------------------------À
º      RETURN                     º
º AX: = 00H OK                    º
Û---------------------------------ì
```

In CX ist die ID-Nummer zu übergeben. Ein Wert AX <> 0 nach dem Aufruf zeigt an, daß unter dieser ID kein TSR-Programm installiert ist. Hat ein nachgeladenes Programm Interrupts der zu entfernenden Routine benutzt, dann wartet TeSeRact so lange, bis der Vektor ohne Probleme entfernt werden kann. Das Programm bleibt allerdings weiter im Speicher geladen.

Restart TSR (BX = 13H)

Mit diesem Aufruf läßt sich ein mit Unload deaktiviertes TSR-Programm wieder aktivieren.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  13 (Restart TSR)           º
º CX:  TSR-ID-Nummer              º
û---------------------------------À
º      RETURN                     º
º AX: = 00H OK                    º
Û---------------------------------ì
```

In CX ist die Nummer des TSR-Programmes zu übergeben, welches mit Unload deaktiviert wurde. Werte ungleich 0 in AX zeigen nach dem Aufruf an, daß sich die betreffende TSR-Routine nicht mehr aktivieren läßt.

Get Status Word (BX = 14H)

Mit diesem Aufruf läßt sich das interne Statuswort abfragen:

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  14 (Get Status Word)       º
º CX:  TSR-ID-Nummer              º
û---------------------------------À
º      RETURN                     º
º AX: = FFFFH ungültige ID        º
º AX: = 00H OK                    º
º   BX: Bit Flags                 º
Û---------------------------------ì
```

In CX ist die ID-Nummer zu übergeben. Werte gleich FFFFH in AX zeigen nach dem Aufruf an, daß sich die betreffende TSR-Routine nicht abfragen läßt. Alle anderen Werte belegen einen erfolgreichen Aufruf. Im Register BX findet sich dann der Wert des Status Word als Bitflag.

Set Status Word (BX = 15H)

Mit diesem Aufruf läßt sich das Status Word setzen.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  15 (Set Status Word)       º
º CX:  TSR-ID-Nummer              º
º DX:  neue Bit Flags             º
û---------------------------------À
º      RETURN                     º
º AX: = 00H OK                    º
Û---------------------------------ì
```

In CX ist die ID-Nummer zu übergeben, während in DX der neue Wert des Status Words steht. Der Wert AX = 0 nach dem Aufruf signalisiert einen fehlerfreien Ablauf.

Get INDOS State at PopUp (BX = 16)

Mit diesem Aufruf läßt sich der Zustand des INDOS-Flags zum Zeitpunkt des PopUp-Aufrufes abfragen.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  16 (Get INDOS-Flag)        º
º CX:  TSR-ID-Nummer              º
û---------------------------------À
º      RETURN                     º
º AX: = FFFFH ungültige ID        º
º AX: = 00H OK                    º
º   BX: INDOS Wert                 º
Û---------------------------------ì
```

In CX ist die ID-Nummer des aktiven Prozesses zu übergeben. Der Wert AX = 0 zeigt nach dem Aufruf, daß sich in BX ein gültiger Wert des .i.INDOS-Flag;s befindet. Dies ist der Wert, der in DOS beim Aufruf des TSR-Programmes vorlag.

Die Funktionen 17H-1FH sind reserviert.

Call User Procedure (BX = 20H)

Mit diesem Aufruf lassen sich Benutzerdaten an das TSR-Programm übergeben.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  20 (Call User Procedure)   º
º CX:  TSR ID Nummer              º
º ES:DI Adresse User Datenbereich º
û---------------------------------À
º      RETURN                     º
º AX: = 00H OK                    º
Û---------------------------------ì
```

In CX ist die ID-Nummer des betreffenden Prozesses zu übergeben. In ES:DI erwartet das Programm die Adresse auf eine Tabelle mit den Benutzerdaten. Ein Wert AX = 0 zeigt nach dem Aufruf, daß die Funktion korrekt abgewickelt wurde. Das Format des User-Parameterblocks ist folgender Tabelle zu entnehmen.

Offs.	Bytes	Feld
00H	8	TSR Name, aufgefüllt mit Blanks
08H	2	TSR-ID-Nummer
0AH	4	Bitmap der unterstützten Funktionen
0EH	1	Scan Code des ersten Hotkeys 00H = PopUp falls = Shift States FFH = kein PopUp bei Shift Status FFH
0FH	1	Shift Status erster Hotkey FFH = kein PopUp bei Scan Code = FFH
10H	1	Zahl der sekundären Hotkeys
11H	4	Zeiger auf extra Hotkey, gesetzt durch die Funktion BX = 05H
15H	2	aktuelles TSR-Status-Flag
17H	2	PSP-Segment des TSR-Programmes
19H	4	DTA des TSR-Programmes
1DH	2	Standard-DatenSegm. des TSR-Programmes
1FH	4	Stack beim PopUp
23H	4	Stack beim Aufruf im Hintergrund

Die Tabelle ist durch das Anwenderprogramm zu versorgen.

Stuff Keystrokes into Keyboard Buffer (BX = 21H)

Mit diesem Aufruf lassen sich Zeichen (Tasteneingaben) in den Tastaturpuffer speichern.

```
CALL:  INT 2F

AX:  5453H (TeSeRact)
BX:  21 (Stuff Keystrokes)
CX:  TSR-ID-Nummer
DL:  Geschwindigkeit
DH:  Scan Code Flag
SI:  Zahl der Zeichen
ES:DI Bufferadresse

RETURN

AX: = 00H OK
       F0F0H Ctrl-C Abbruch
```

In CX ist die ID-Nummer des betreffenden Prozesses zu übergeben. In ES:DI erwartet das Programm die Adresse auf einen Puffer, in den die Tastatureingaben zu speichern sind. In DL wird ein Code für die Eingabegeschwindigkeit übergeben. Es gilt folgende Belegung:

```
00H speichere Tastatureingaben nur, falls der Puffer leer ist
01H speichere bis zu vier Tasten pro Zeiteinheit
02H speichere bis zu 15 Tasten pro Zeiteinheit
```

Als Zeiteinheit werden die Clockticks der Systemuhr benutzt. In DH wird das Scan-Code-Flag mit folgender Bedeutung übergeben.

```
0: der Puffer enthält ASCII- und Scan Codes alternierend
<> 0: der Puffer enthält nur ASCII-Codes
```

In SI ist die Zahl der Tastenanschläge zu übergeben. Nach dem Aufruf zeigt AX den Fehlerstatus an. Mit dem Wert AX = 0H wurde die Funktion fehlerfrei bearbeitet. Mit AX = F0F0H wird signalisiert, daß die Zeichen Ctrl-C oder Ctrl-Break eingegeben wurden. Alle anderen Werte signalisieren, daß die Tastatureingaben nicht bearbeitet werden.

Die folgenden beiden Funktionsaufrufe sind nur in der Version 1.10 implementiert.

Trigger PopUp (BX = 22H)

Mit dem Aufruf läßt sich ein Programm aktivieren.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  22 (Trigger PupUp)         º
º CX:  TSR-ID-Nummer              º
û---------------------------------À
º      RETURN                     º
º AX: = 00H OK                    º
Û---------------------------------ì
```

Wird der Wert AX = 0000H zurückgegeben, war der Aufruf erfolgreich.

Invoke TSR Background Functions (BX = 23H)

Mit dem Aufruf läßt sich ein Programm aktivieren.

```
Ö---------------------------------Ì
º      CALL:  INT 2F              º
º                                 º
º AX:  5453H (TeSeRact)           º
º BX:  23 (TSR Background)        º
º CX:  TSR-ID-Nummer              º
û---------------------------------À
º      RETURN                     º
º AX: = Status                    º
Û---------------------------------ì
```

Nach dem Aufruf enthält das Register AX den Status:

```
AX = 0H, der Aufruf war erfolgreich
     FFFFH, es darf zur Zeit kein Hintergrundtask aufgerufen
     werden
```

Alle anderen Werte signalisieren, daß die übergebene ID-Nummer ungültig ist.

Die Funktionen 24H - 2FH sind reserviert.

Portadressen für Peripheriebausteine

Die PC-, XT-, AT- und 386-Modelle lehnen sich mit der Belegung der I/O-Adressen alle an die IBM-Vorgaben an. Nachfolgende Tabelle enthält eine Grobübersicht über die Portadressen.

```
Adresse     Funktion
-----------------------------------------------
0000-000F º 8237 DMA-Controller
0010-001F º 8237 DMA-Controller  (AT, PS/2)
0020-0027 º 8259A Interrupt Controller
0020-003F º 8259A Interrupt Controller  (AT)
0040-005F º 8253-5 programmierbarer Timer
0060-0067 º 8255 Peripheral Interface
0060-006F º 8042 Tastatur Controller  (AT)
0070-007F º CMOS-Uhr (AT,etc.)
0080-008F º DMA-Seitenregister (AT, etc.)
00A0-00BF º zweiter Interrupt-Controller (AT, etc.)
00C0-00DF º zweiter DMA-Controller (AT, etc.)
00F0-00FF º Arithmetikprozessor (AT, etc.)
0170-017F º zweiter Disk-Controller (AT, etc.)
01F0-01FF º erster Disk-Controller (AT, etc.)
0200-020F º Spieleadapter
0210-0217 º Expansion Box (PC, XT)
0F20-0277 º reserviert
0278-027F º LPT3
02E8-02EF º COM4
0F08-02F7 º reserviert
02F8-02FF º COM2
0300-031F º Prototype-Karte
0320-032F º Hard Disk Controller (PC)
0330-035F º reserviert
0360-0367 º Netzwerkkarten (Low-Adresse)
0368-036F º Netzwerkkarten (High-Adresse)
0370-037F º zweiter Disketten-Controller
0378-037F º LPT2
03BC-03BF º LPT1
03C0-03CF º 1. EGA-Video Controller
03D0-03DF º CGA, MCGA, VGA Adapter Controller
03F0-03F7 º Floppy Disk Controller
03E8-03EF º COM3
03F0-03F7 º Disketten-Controller
03F8-03FF º COM1
```

Nachfolgend wird die Belegung der wichtigsten I/O-Ports wiedergegeben.

DMA-Controller (Adresse 000 _ 01F)

Diese Adressen werden durch den 4-Kanal-DMA-Controller 8237 belegt. Ab den AT-Rechnern handelt es sich bei dem Controller um die Slave-Einheit. Dabei gilt folgende Belegung der I/O-Register.

Hauptsteuerregister (Adr. 008)

Wird das Register geschrieben, dient es als Steuerregister. Es gilt folgende Belegung:

Bit	Bedeutung
0	0 = disable Memory-to-Memory-Übertragung
	1 = enable Memory-to-Memory-Übertragung
1	0 = Adresse Kanal 0 nicht speichern
	1 = Adresse Kanal 0 speichern
2	0 = enable DMA-Controller
	1 = disable DMA-Controller
3	0 = normales Timing
	1 = komprimiertes Timing
4	0 = fixe Priorität
	1 = rotierende Priorität
5	0 = normale Schreibauswahl
	1 = extended Schreibauswahl
6	Zustand des DREQ-Signals
7	Zustand des DACK-Signals

Statusregister (Adr. 008)

Wird das Register gelesen, dient es als Statusregister. Es gilt folgende Belegung:

Bit	Bedeutung
0	1 = Kanal 0 Zählerüberlauf (Timer Count)
1	1 = Kanal 1 Zählerüberlauf (Timer Count)
2	2 = Kanal 2 Zählerüberlauf (Timer Count)
3	1 = Kanal 2 Zählerüberlauf (Timer Count)
4	1 = DMA-Request Kanal 0
5	1 = DMA-Request Kanal 1
6	1 = DMA-Request Kanal 2
7	1 = DMA-Request Kanal 3

Kanal-Request-Register (Adr. 009)

Auf das Register kann nur schreibend zugegriffen werden. Es dient zur Selektion des gewünschten DMA-Kanals. Es gilt folgende Belegung:

Bit 1 0	Kanalbelegung
00	Kanal 0
01	Kanal 1
10	Kanal 2
11	Kanal 3

Bit 2	Anforderungsbit
0	Anforderungsbit Kanal löschen
1	Anforderungsbit Kanal übernehmen

Die restlichen Bits sind unbelegt.

Kanal-Mask-Register (Adr. 00A)

Auf das Register kann nur schreibend zugegriffen werden. Es dient zur Selektion des gewünschten DMA-Kanals. Es gilt folgende Belegung:

```
 Bit 1 0 ° Kanalbelegung für Maskierung
---------é-------------------------------
 00      ° Kanal 0
 01      ° Kanal 1
 10      ° Kanal 2
 11      ° Kanal 3


 Bit 2   ° Maskierungsbit
---------é-------------------------------
 0       ° Maskierungsbit Kanal löschen
 1       ° Maskierungsbit Kanal übernehmen
```

Die restlichen Bits sind unbelegt.

Modeauswahlregister (00B)

Auf das Register kann nur schreibend zugegriffen werden.

```
 Bit 1 0 ° Kanalbelegung für Modeauswahl
---------é-------------------------------
 00      ° Kanal 0
 01      ° Kanal 1
 10      ° Kanal 2
 11      ° Kanal 3


 Bit 3 2 ° Mode
---------é-------------------------------
 00      ° Verify Mode
 01      ° Write Mode
 10      ° Read Mode
 11      ° unused


 Bit 4   ° Modebit
---------é-------------------------------
 0       ° Autoinitialize disable
 1       ° Autoinitialize enable

 Bit 5   ° Decrementbit
---------é-------------------------------
 0       ° Autoincrement Adresse
 1       ° Autodecrement Adresse


 Bit 7 6 ° Mode
---------é-------------------------------
 00      ° Request Mode
 01      ° Single Byte Mode
 10      ° Block Mode
 11      ° Cascade Mode
```

Maskenschreibregister (00F)

Das Register dient zur Selektion der Maskierungsbits der einzelnen Kanäle und läßt sich nur beschreiben.

```
Ö-----Ú-----------------------------------------Ì
° Bit ° Bedeutung                              °
û-----é-----------------------------------------À
° 0   ° 0 = clear Maskbits Kanal 0             °
°     ° 1 = set   Maskbits Kanal 0             °
û-----é-----------------------------------------À
° 1   ° 0 = clear Maskbits Kanal 1             °
°     ° 1 = set   Maskbits Kanal 1             °
û-----é-----------------------------------------À
° 2   ° 0 = clear Maskbits Kanal 2             °
°     ° 1 = set   Maskbits Kanal 2             °
û-----é-----------------------------------------À
° 3   ° 0 = clear Maskbits Kanal 3             °
°     ° 1 = set   Maskbits Kanal 3             °
Û-----Ù-----------------------------------------ì
```

Die restlichen Bits sind nicht belegt.

8-Kanal-Interrupt-Controller (8259)

Ab dem AT-Rechner arbeitet der Controller auf den IO-Adressen 20 bis 3F als Master. Dabei gilt folgende Belegung der Interrupteingänge:

Eingang	Signal
0	Takt Kanal 0 Timer
1	Keyboard Interrupt
2	Slave Interrupt Controller
3	COM 2
4	COM 1
5	Disk-Controller
6	Disketten-Controller
7	LPT Schnittstelle

Auf Adresse 020 liegt das Interrupt-Befehlsregister und auf Adresse 021 das Interrupt-Maskierungsregister. Nähere Hinweise über die Belegung der Dokumentation des 8259-Bausteins zu entnehmen.

Der zweite Interrupt-Controller liegt auf den IO-Adressen 0A0 bis 0BF und besitzt folgende Signalbelegung:

Eingang	Signal
8	Takt CMOS-Uhr
9	Grafikkarte
10	reserviert
11	reserviert
12	reserviert
13	Arithmetikprozessor
14	erster Disk-Controller
15	reserviert

3-Kanal-Timerbaustein (8253)

Ab Adresse 040 bis 05F liegt der Timerbaustein 8253, welcher folgende Register besitzt:

Adresse	Register
040	16 Bit Zählwert Kanal 0
041	16 Bit Zählwert Kanal 1
042	16 Bit Zählwert Kanal 2
043	Steuerregister

Die Belegung des Steuerregisters bei Schreibzugriffen und der Status des Moderegisters bei Lesezugriffen ist der Dokumentation des 8253-Bausteins zu entnehmen.

Tastatur-Controller (8042)

Alle PCs ab der AT-Serie benutzen einen Single Chip Prozessor zur Tastaturabfrage. Dieser Prozessor kommuniziert über die IO-Ports 60 bis 6F mit dem Hauptprozessor. Dabei gilt folgende Belegung:

Die Adresse 60 dient als Ein-/Ausgabepuffer, über den Daten ausgetauscht werden. Lesezugriffe aus DOS auf den Port liefern die Tastaturbytes. Schreibzugriffe dienen zur Übergabe von Zeichen an den Tastatur-Controller.

Der Port auf Adresse 64H dient als Steuerregister (Schreiben) und als Statusregister (Lesen).

Steuerregister (Adr. 064)

Wird das Register geschrieben, gilt folgende Belegung:

Bit	Bedeutung
0	kein INT nach Zeicheneingabe im Puffer
1	INT nach Zeicheneingabe im Puffer
2	System Status Bit
3	0 disable Tastaturfunktion 1 enable Tastaturfunktion
4	0 Tastatur freigeben 1 Tastatur sperren
5	0 Normal-Modus 1 Parity Check, Rohwerte als Code
6	0 Tastaturwerte nicht konvertieren 1 Tastaturwerte in IBM-Format
7	reserviert

Statusregister (Adr. 064)

Wird das Register gelesen, gibt der 8042 den Status zurück:

Bit	Bedeutung
0	0 Ausgabepuffer leer, 1 Zeichen vorh.
1	0 kein neuer Wert in Ausgabepuffer oder im Steuerregister 1 ein neuer Wert in Ausgabepuffer oder im Steuerregister
2	System Status Bit (Reset setzt Bit = 0)
3	0 last write auf Eingabepuffer 1 last write auf Steuerregister
4	0 Tastaturfunktion gesperrt 1 Tastaturfunktion freigegeben
5	0 Datenübertragung vom Controller ok 1 Fehler bei der Übertragung (Bit 6,7)
6	0 Übertragung ok 1 Time Out bei Übertragung
7	0 Übertragung ok 1 Parity-Fehler bei Übertragung

Die CMOS-Echtzeituhr

Ab den AT-Rechnern gelangt eine CMOS-Echtzeituhr zum Einsatz. Diese belegt die folgenden Adressen:

Adresse	Register
070	Adreßregister CMOS-Uhr
071	Datenregister CMOS-Uhr

Die Belegung der Daten in der CMOS-Uhr ist in Tabelle 11.2 beschrieben.

Die LPT-Schnittstelle

Die LPT-Schnittstellen benutzen verschiedene Adreßbereiche. Ab 278 bis 27F liegt in der Regel LPT1. Die Adressen besitzen folgende Codierung:

Das Datenregister wird über die I/O-Adresse 278 angesprochen, d.h., über diesen Port werden die Daten an den Drucker gesendet. Es lassen sich auch Daten aus diesem Port zurücklesen.

Das Statusregister liegt auf Adresse 279 und besitzt folgende Belegung:

```
 Bit ° Bedeutung
-----é----------------
 0   ° reserviert
 1   ° reserviert
 2   ° reserviert
 3   ° 0 = Druckerfehler
 4   ° 1 = Drucker Select
 5   ° 1 = Paper Out
 6   ° 0 = Acknowledge
 7   ° 0 = Busy
```

Das Steuerregister der Parallelschnittstelle liegt auf Adresse 27A und besitzt folgende Codierung:

```
 Bit ° Bedeutung
-----é----------------
 0   ° 1 = Data Strobe
 1   ° 1 = Auto Feed
 2   ° 0 = Initialize Printer
 3   ° 1 = Select Input
 4   ° 1 = Acknowledge (löst INT 07H aus)
 5   ° reserviert
 6   ° reserviert
 7   ° reserviert
```

Die restlichen Portadressen sind reserviert.

Für die anderen LPT-Schnittstellen gilt die gleiche Belegung, wobei allerdings andere Portadressen eingestellt werden.

Die COM-Schnittstelle

Die COM-Schnittstellen benutzen verschiedene Adreßbereiche. Ab 2F8 bis 2FF liegt in der Regel COM2. Die Adressen besitzen folgende Codierung:

Die 8-Bit-Sende- und -Empfangsregister werden über Adresse 2F8 angesprochen. Bei der Initialisierung wird über die gleiche Adresse die Baudrate eingestellt. In die Adresse 2F8 wird dabei das LSB des Divisors geschrieben, während das Highbyte auf Adresse 2F9 zu schreiben ist.

Das Interrupt-Freigabe-Register liegt auf Adresse 2F9 und besitzt folgende Codierung:

```
 Bit ° Bedeutung
-----é----------------
 0   ° 0 kein INT beim Datenempfang
     ° 1 INT beim Datenempfang
 1   ° 0 kein INT falls Sendepuffer leer
     ° 1 INT falls Sendepuffer leer
 2   ° 0 Empfängerstatus INT sperren
     ° 1 Empfängerstatus INT freigeben
 3   ° 0 Modemstatus INT sperren
     ° 1 Modemstatus INT freigeben
```

Die folgende Tabelle gibt die Belegung der I/O-Adressen wieder:

Adresse	Register
2F8	Sender / Empfänger
2F9	Interrupt-Freigabe
2FA	Interrupt-Identifikation
2FB	Leitungssteuerung Senden
2FC	Modemsteuerung Senden
2FD	Leitungssteuerung Empfangen
2FE	Modemstatus Empfangen

Für die Leitungssteuerung (Empfangen) gilt folgende Belegung:

Bit	Bedeutung
0	Receiver Data Ready
1	Overrun Error
2	Parity Error
3	Framing Error
4	Break Interrupt
5	Transmitter Holding Register empty
6	Transmitter empty
7	0

Für das Modemstatusregister gilt folgende Belegung:

Bit	Bedeutung
0	Delta Clear to Send
1	Delta Set Ready
2	Trailing Edge Ring Indicator
3	Delta Data Carrier Detect
4	Request to Send
5	Data Terminal Ready
6	Out1 Modemsteuerung
7	Out2 Modemsteuerung

Weitere Hinweise sind der Herstellerliteratur zu entnehmen.

Das DOS-Speichermodell

In DOS wird zwischen dem internen Hauptspeicher und externen Disketten- oder Plattenspeichern unterschieden. Der interne Speicher dient zur Aufnahme der aktiven Betriebssystemteile, verschiedener Daten und der auszuführenden Programme. Auf den externen Speichern werden Daten und Programme ausgelagert und für die Bearbeitung im internen Speicher bereitgehalten.

Bei der Konzeption des IBM-PC Anfang 1980 stand nur der 8088-Prozessor mit einem 1 Mbyte-Adressraum zur Verfügung. Der Entwurf von IBM sah deshalb folgende Aufteilung des Hauptspeicherbereiches vor:

- 640 Kbyte RAM-Speicher für Programme und Daten im unteren Adressbereich zwischen 0 und 9FFFFH.
- 384 Kbyte reservierter Bereich für Adapterkarten oberhalb des RAM-Bereiches bis zur 1 Mbyte-Grenze.

Die Entwickler von MS-DOS haben diese 640 Kbyte-Speichergrenze, bis zur Version 6.0, als Maximum manifestiert. Aber bereits Mitte der achtziger Jahre zeichnete sich ab, daß für viele Programme diese Beschränkung Probleme aufwirft. In den folgenden Jahren entstanden eine Reihe von Lösungen, die ausgehend von der gestiegenen Hardwareleistung, den unter DOS verfügbaren Speicher erweitern sollen. Begriffe wie

EMS, XMS und HMA stehen für diese Techniken. Ein System kann deshalb folgende Speicherbereiche aufweisen:

- Konventioneller Speicher (640 Kbyte)
- Reservierte Speicherbereiche (640 Kbyte bis 1 Mbyte)
- Expanded Memory
- Extended Memory (oberhalb 1 Mbyte)

Was sich hinter diesen einzelnen Begriffen verbirgt, wird im Kapitel über den Speichermanager erläutert.

Die Installation des XMS-Managers (HIMEM.SYS)

Ab MS-DOS 4.0 steht ein Treiber zur Verwaltung des Extended Memory zur Verfügung. Das Programm HIMEM.SYS ist in CONFIG.SYS mit der Anweisung:

```
DEVICE=[Lw:][Pfad]HIMEM.SYS [/hmamin=m] [/numhandles=n]
[/int15=xxxx] [/machine:xxxx] [/a20control:on/off]
[/shadowram:on/off]
```

zu aktivieren. Für die Paramameter gilt folgende Bedeutung:

Lw:	Laufwerk, auf dem der Treiber gespeichert ist. Pfad Pfad in dem der Treiber gespeichert ist.
/hmamin=m	Spezifiziert die minimale Speichergröße die eine Applikation anfordern muß, bevor dieser ein Zugriff auf den HMA-Bereich erlaubt wird. Dies ist notwendig, da nur ein Programm im HMA geladen werden kann. Die Werte für m dürfen zwischen 0 und 63 liegen.
/numhandles=n	Spezifiziert die maximale Zahl von Extended Memory Blocks (EMB) die gleichzeitig benutzt werden können. Der Wert in n darf zwischen 1 und 128 liegen und wird standardmäßig auf 32 gesetzt. Jeder Handle belegt 6 Byte resident im Speicher.

Die nachfolgenden Optionen sind erst ab MS-DOS 5.0 verfügbar.

/int15=xxxx	Reserviert einen Teil des Extended Memory für den Zugriff durch den INT 15. Dies ist hilfreich für ältere Programme, die XMS nur über diesen Interrupt anfordern können. Der Wert xxxx gibt die Größe des reservierten Bereiches in Kbyte an und sollte mindestens 64 (Kbyte) betragen.
/machine:xxxx	Der Treiber HIMEM.SYS muß zum Zugriff auf den Extended-Memory-Bereich die A20-Adressleitung freigeben können. Bei verschiedenen Rechnern gibt es unterschiedliche Methoden. Gibt es hierbei Probleme, läßt sich der Rechnertyp in xxxx angeben. Für xxxx gelten folgende Codes: 1 oder at für IBM PC/AT

2 oder ps2 für IBM PS/2
3 oder pt1cascade für Phoenix Cascade BIOS
4 oder hpvectra für HP Vectra
5 oder att6300plus für AT&T 6300 plus
6 oder acer1100 für Acer 1100
7 oder toshiba für Toshiba 1600 und 1200XE
8 oder wyse für 12,5 MHz Wyse 286 m/c
16 oder bull für 1BULL Micral 60

/a20control:on/off Wird der Schalter beim Aufruf auf »on« gesetzt, übernimmt HIMEM.SYS die Kontrolle über die A20-Leitung nur, falls A20 vorher auf »on« war. Beim Aufruf mit »off**Fehler! Verweisquelle konnte nicht gefunden werden.**die Kontrolle nur, falls A20 vorher auf »off**Fehler! Verweisquelle konnte nicht gefunden werden.**/shadowram:on/off Ist der Schalter beim Aufruf auf »offsetzt, versucht HIMEM.SYS das Shadow-RAM freizugeben und dem Speicherpool zuzuschlagen. Bei der Option »on640 Kbyte und 1 Mbyte eingeblendet.

Wird HIMEM.SYS ohne Parameter aufgerufen, gibt der Treiber die Standardeinstellung vor. Dann ist hmamin = 0, was bedeutet, daß das erste Programm, welches diesen Speicher anfordert, in den HMA geladen wird. Der HMA kann aber immer nur durch ein Programm genutzt werden. Deshalb sollte immer eine minimale Größe definiert werden, damit nicht ein Programm den 64 Kbyte großen HMA-Bereich mit wenigen Byte belegt und damit alle anderen Applikationen von der Benutzung ausschließt.

HIMEM.SYS muß geladen werden, bevor eine Anweisung in CONFIG.SYS versucht den XMS-Speicher anzusprechen. Der Treiber übernimmt folgende Aufgaben:

_ Verwaltung des XMS-Speicherbereiches.

_ Verwaltung des HMA-Bereiches.

_ Erlaubt Teile von DOS in den UMB und HMA zu laden.

Mit HIMEM.SYS wird sowohl das Extended Memory mit dem 64 Kbyte großen HMA-Bereich verwaltet, als auch die Zuteilung der UMB's vorgenommen. Zum Zugriff auf den UMB muß allerdings zusätzlich EMM386 aktiv sein. Der Treiber ist zusätzlich mit WINDOWS 3.0 kompatibel. Das folgende kleine Beispiel aktiviert den XMS-Manager.

```
DEVICE=C:\DOS\HIMEM.SYS /hmamin=40 /numhandles=128
```

Dieser kann anschließend 128 Handles verwalten und gibt den HMA-Bereich nur frei, falls ein Programm mindestens 40 Kbyte anfordert. HIMEM.SYS ist nur auf 80286, 80386 und 80486-Systemen einsetzbar und belegt einen kleinen Teil des 640-Kbyte-Speichers für den Code. Weiterhin ist HIMEM.SYS nicht mit allen Programmen (z.B. WINDOWS 386) kompatibel.

Die Installation des EMS-Managers

Um unter DOS den EMS-Bereich zu verwalten, gibt es verschiedene Speichermanager. Meist liefert der Hersteller des Boards diesen Treiber mit, der dann nur noch in CONFIG.SYS einzubinden ist.In DOS 4.0 existiert das Programm XMA2EMS.SYS, welches sich in CONFIG.SYS aktivieren läßt. Das Programm stellt eine Verwaltung für EMS nach dem LIM 4.0-Standard dar. Mit XMAEM.SYS bietet DOS 4.0 weiterhin einen Emulator, der XMS-Speicher in EMS umwandeln kann. Hinweise über den Aufruf und die Installation sind in den DOS-Handbüchern oder in /2,5/ zu finden.

Wesentlich interessanter wird die Situation bei 80386 und 80486-Systemen. Hier bieten MS-DOS 5.0 und DOS 6.0 das Programm EMM386.EXE. (Anmerkung: eine Version mit reduziertem Funktionsumfang wird auch mit MS-DOS 4.0 ausgeliefert). Hierbei handelt es sich einmal um einen Emulator, der Teile des Extended Memory in Expanded Memory nach dem LIM 4.0-Standard umwandeln kann. Weiterhin übernimmt dieses Programm die Verwaltung des UMA-Bereiches (erst ab DOS 5.0) und verlagert auf Anforderung Teile von DOS in diesen Bereich.

EMM386.EXE in MS-DOS 4.0-6.0

Ab MS-DOS 5.0 besitzt das Programm EMM386.EXE folgende Aufrufsyntax:

```
DEVICE=[Lw:][Pfad]EMM386.EXE [Memory] [MemoryOptions]
```

Für die Parameter gilt folgende Belegung:

Lw:	Ist das Laufwerk, auf dem sich das Programm befindet.
pfad	Ist der Pfad, in dem die Datei mit dem Gerätetreiber EMM386.SYS gespeichert ist.
Memory	Ist die Größe des XMS-Bereiches, der für die EMS-Emulation zur Verfügung gestellt wird (ab MS-DOS 4.0).

Die Parameter im Feld *MemoryOptions* müssen in der Regel nicht spezifiziert werden. EMM386 übernimmt Standardwerte für den Betrieb. Falls Probleme mit dieser Einstellung auftreten, läßt sich über diese Optionen die Arbeitsweise des Managers einstellen. Für die *MemoryOptions* gilt folgende Belegung:

w=[on/off]	Schaltet den Support für Weitek Coprozessoren ein oder aus. Standardmäßig ist die Unterstützung aus geschaltet.
Mx	Definiert die Segmentadresse, in dem das EMS-Fenster innerhalb des 1-Mbyte-Bereiches eingeblendet wird. X ist eine Zahl zwischen 1 bis 14 (8 bei MS-DOS 4.0) mit folgender Bedeutung: 1 C000H 2 C400H 3 C800H 4 CC00H 5 D000H 6 D400H 7 D800H 8 DC00H 9 E000H 10 8000H 11 8400H 12 8800H 13 8C00H 14 9000H

	Für die Werte 10 bis 14 darf das System nur 512 Kbyte Speicher aufweisen, da sonst das EMS-Fenster im Arbeitsbereich (640 Kbyte) liegt (ab MS-DOS 4.0).
Frame= adress	Spezifiziert die Lage des EMS-Segments direkt. Um EMS in C000 zu definieren, ist frame=C000 in die Aufrufzeile aufzunehmen.
/Paddress	Spezifiziert ebenfalls die Lage des EMS-Fensters direkt als Segment.
Pn=address	Mit dieser Option läßt sich die Segmentadresse einer EMS-Seite direkt angeben. Für »n**Fehler! Verweisquelle konnte nicht gefunden werden.**(N=0,1,2,3,254,255) anzugeben. Um die EMS-Seite 3 in den Speicher ab Adresse C800 zu legen, ist die Option P3=C800 in das Kommando aufzunehmen.
X=addressbereich	Diese Angabe verhindert die Belegung des Adressraumes durch EMM386. Mit X=8000-9000 wird der entsprechende Bereich von der Nutzung ausgespart (ab DOS 4.0).
B=adress	Definiert die unterste verfügbare Segmentadresse für eine EMS-Speicherbank. Der Standardwert ist 4000. Die Adressen dürfen im Bereich zwischen 1000 und 3000 liegen.
L=minXMS	Definiert einen Mindestkapazität des XMS-Bereiches, die nach Aufruf des Emulators als XMS erhalten bleibt. Um 640-Kbyte-XMS zu erhalten, ist L=640 anzugeben.
A=altregs	Diese Option definiert die Zahl der »alternate register sets« Es sind Werte zwischen 0 und 254 vorgesehen,»wobei der Standard auf 7 gesetzt ist.
H=handles	Spezifiziert wieviele Handles der EMM386-Manager benutzen darf. Standardmäßig ist 64 eingestellt, die Werte dürfen aber zwischen 2 und 255 schwanken.
Ixxx-yyy	Weist den Treiber an, den Bereich zwischen xxx und yyy als UMB zu nutzen.
ram	Weist den Treiber an, EMS-Speicher zur Verfügung zu stellen.
noems	Dieser Schalter signalisiert, daß EMM386 zur Verwaltung des UMB benutzt werden soll. Es wird dann kein EMS emuliert.

Die Parameter sind beim Aufruf nach den Systemanforderungen zu setzen. Mit dem Kommando:

```
DEVICE=C:\DOS\EMS386.EXE 512 frame=d000 X=e000-ec00
```

wird ein 512 Kbyte großer EMS-Bereich ab D000H zur Verfügung gestellt. Der Adressbereich zwischen E000 und EC00 ist im UMB nicht verfügbar.

EMM386 zum Zugriff auf den UMB

Die zweite Aufgabe des EMM386-Treibers (erst ab MS-DOS 5.0) besteht in der Verwaltung der Upper Memory Blocks (UMB). Ist der Treiber geladen, lassen sich Treiber, residente Programme und Teile von DOS in den UMB auslagern. Dadurch steht zusätzlicher freier Speicher im konventionellen RAM-Bereich zur Verfügung. Um die Programmteile im UMB zu laden, sind ab MS-DOS 5.0 die Befehle: DEVICEHIGH und LOADHIGH zu verwenden. Das Programm ist gegebenenfalls mit der Option *noems* zu aktivieren.

```
DEVICE=C:\DOS\HIMEM.SYS ....
DEVICE=C:\DOS\EMM386.EXE noems
```

Bei dieser Option steht dann aber kein EMS-Speicher mehr zur Verfügung. Lediglich bei Verwendung von WINDOWS 3.0 kann eine Applikation sich EMS reservieren, da hier die Verwaltung direkt durch WINDOWS erfolgt. Wird der Treiber mit:

```
DEVICE=C:\DOS\HIMEM.SYS ....
DEVICE=C:\DOS\EMM386.EXE ram
```

aktiviert, steht weniger Speicherplatz im UMB zur Verfügung, da ja in einen Block das 64-Kbyte-EMS-Fenster einzublenden ist. Benutzt das System eine EMS-Hardware, muß weiterhin der Treiber für die betreffende Karte in CONFIG.SYS eingebunden werden.

Verlagerung von DOS in den UMB-Bereich

Auf 386-Systemen lassen sich Teile des Betriebssystemes in den reservierten Bereich zwischen 640 Kbyte und 1 Mbyte auslagern. Voraussetzung ist, daß hier RAM eingeblendet wird und daß noch freie Speicherblöcke vorhanden sind. Weiterhin müssen die Speichermanager HIMEM.SYS und EMM386.EXE geladen werden. Um Teile von DOS auszulagern, muß folgende Sequenz in der beschriebenen Reihenfolge in CONFIG.SYS aufgenommen werden.

```
DEVICE=C:\DOS\HIMEM.SYS ....
DOS = high, umb
DEVICE=C:\DOS\EMM386.EXE ....
```

Das Kommandos DOS in der CONFIG.SYS signalisiert daß Teile von DOS oberhalb von 640 Kbyte auszulagern sind. Das Kommando besitzt folgende Syntax:

```
DOS = umb / noumb
DOS = high / low
```

Die Parameter signalisieren wie DOS zu installieren ist. Die Optionen besitzen dabei folgende Bedeutung:

umb	Erlaubt DOS eine Verbindung zwischen dem konventionellen Speicher und dem UMB-Bereich anzulegen. Dadurch lassen sich Programme im UMB installieren.
noumb	Dies ist die Standardeinstellung, die den UMB von der Nutzung durch Programme und Treiber ausschließt.
high	DOS versucht Teile des Kerns in den HMA-Bereich zu laden.
low	DOS wird in den konventionellen Bereich des Speichers geladen.

DOS besitzt die Möglichkeit, Teile der Einheitentreiber in den UMB-Bereich zu laden. Anschließend wird die Verbindung zum UMB abgebrochen. Sollen Anwenderprogramme mit LOADHIGH ebenfalls in den UMB verlagert werden, ist der Parameter *umb*

anzugeben. Dadurch bleibt die Speicherverwaltung für den UMB auch nach dem Systemstart aktiv. Mit der *high*-Option verlagert DOS Teile des Systems in den HMA-Bereich. Hierdurch werden Teile des konventionellen Speichers frei. Es lassen sich die folgenden Optionen kombinieren:

```
DOS=umb, low
DOS=high, umb
```

Diese Anweisungen lassen sich an jeder beliebigen Stelle in CONFIG.SYS angeben. Die weiter unten vorgeschlagene Reihenfolge sollte aber eingehalten werden.

DOS und Windows 3.x

Beim Betrieb von DOS-Anwendungen unter Windows im 286- und im 386-Mode treten einige Einschränkungen auf. Der DOS-Extender DOSX.EXE von Windows unterstützen nicht mehr alle definierten Funktionen. Bei der Weiterentwicklung von Software sollten deshalb folgende Änderungen berücksichtigt werden:

```
Nicht mehr unterstützte Interrupts

INT 20H  Terminate Programm
INT 25H  Absolute Disk Read
INT 26H  Absolute Disk Write
INT 27H  Terminate And Stay Resident
```

Weiterhin werden folgende INT 21-Funktionen nicht mehr im Protected Mode von Windwos unterstützt:

```
Nicht mehr unterstützte Funktionen

00H Terminate Process
0FH Open File with FCB
10H Close File with FCB
14H Sequential Read
15H Sequential Write
16H Create File with FCB
21H Random Read
22H Random Write
23H Get File Size
24H Set Relative Record
27H Random Block Read
28H Random Block Write
```

Weiterhin sollten alle undokumentierten Funktionen, die direkt auf Datenstrukturen (von DOS 5.0) zugreifen nur mit Vorsicht genutzt werden. Windows wird mit Sicherheit nicht alle spezifische DOS-Funktionen im Protected Mode umsetzen. Bei Verwendung der folgenden Funktionen des INT 21 ist ebenfalls mit Abweichungen im Protected Mode zu rechenen:

Funktionen 25H und 35H, Get/Set Interrupt

Diese Funktionen setzen und lesen die Interruptvektoren. Im Protected Mode werden die Interrupts (Ausnahme bilden die Interrupts 23H, 24H und 1CH) des Real Modes nicht unterstützt.

Funktion 38H, Get Country Data

Diese Funktion gibt einen 24-Byte-Puffer zurück, die ab Offset 12H einen 4-Byte-Zeiger auf die Adresse der Case-Map-Routine enthält. Der Zeiger ist eine Real-Mode-Adresse.

Um die Adresse des Case-Map-Routine im Protected Mode zu nutzen, ist die DPMI-Übersetzungsfunktion zu nutzen, um einen Real-Mode-Call zu simulieren.

Funktion 44H, Subcode 0CH

Es werden nur die Unterfunktionen 45H (Get Iteration Count) und 65H (Set Iteration Count) im Protected Mode unterstützt. Die Funktionen zur Umschaltung der Code-Pages (Code 4AH, 4CH, 4DH, 6AH und 6BH) werden nicht unterstützt. Um diese Funktionen zu nutzen, muß ein DPMI-Aufruf benutzt werden.

Funktion 65H, Get Extended Country Information

Die Funktion wird zwar unterstützt, aber alle zurückgegebenen Adressen sind Real-Mode-Adresse. Dies bedeutet, daß die DPMI-Übersetzungsfunktion beim Aufruf benutzt werden muß, da im Protected Mode andere Adressmodes gelten.

NetBIOS-Funktionen

Im Protected Mode werden auch die folgenden NetBIOS-Funktionen nicht unterstützt:

```
71H  Send No Ack
72H  Chain Send no Ack
73H  Lan Status Alert
78H  Find Name
79H  Trace
```

Dies ist auf jedem Fall bei der Entwicklung von DOS-Programmen zu beachten, die unter Windows im Protected Mode in einer DOS-Box laufen sollen.

Das SMARTDRIVE 4.0 Interface

Der Plattencache SMARTDRV wird ab MS-DOS 5.0 ausgeliefert. SMARTDRV nutzt ab der Version 4.0 die INT 2F-Schnittstelle (AX=4A10H), die nachfolgend beschrieben wird.

SMARTDRV - Installation check and hit ratios (INT 2F)

Zur Prüfung, ob SMARTDRV installiert ist, nutzen Sie folgende Schnittstelle.

```
Ö------------------------------Ì
°      CALL: INT 2F          °
°                            °
° AX: 4A10H SMARTDRV 4.X      °
° BX: 0000H Get Status        °
° CX: EBABH (ab V 4.1)        °
û------------------------------À
°       Return:              °
° AX: BABEH falls installiert °
° CX: Dirty Blocks           °
° DX:BX Cache hits           °
° DI:SI Cache misses         °
° BP: Version in BCD         °
```

Û----------ì

Beim Aufruf ist in CX die Signatur EBABH zu übergeben. Ist SMARTDRV installiert, enthält AX nach dem Aufruf die Signatur BABEH. In BP wird die SMARTDRV Versionsnummer als BCD-Zahl zurückgegeben. Bei installiertem Cache enthalten die Registerpaare DX:BX die Zahl der Treffer und DI:SI die Zahl der Fehlzugriffe. In CX findet sich nach dem Aufruf die Zahl der uncommitted (dirthy) blocks (meist 0). Die Aufruf verändert alle Register bis auf DS,ES.

SMARTDRV Version 3.x besitzt eine abweichende Schnittstelle und benutzt die INT 21 AH=44H AL=02,03 IOCTL-Aufrufe.

SMARTDRV - Flush (Commit) Buffers (INT 2F)

Dieser Aufruf leert die SMARTDRV-Puffer. Hierbei gilt folgende Aufrufschnittstelle.

Ö----------Ì
° CALL: INT 2F °
° °
° AX: 4A10H SMARTDRV 4.X °
° BX: 0001H Flush Buffer °
û----------À
° Return: °
° -- °
Û----------ì

Die Funktion gibt keine Daten nach dem Aufruf zurück und verändert keine Prozessor Register.

SMARTDRV - Reset Cache (INT 2F)

Dieser Funktionsaufruf setzt den kompletten Cache zurück. Der Inhalt der Puffer geht dabei verloren.

Ö----------Ì
° CALL: INT 2F °
° °
° AX: 4A10H SMARTDRV 4.X °
° BX: 0002H Reset Cache °
û----------À
° Return: °
° -- °
Û----------ì

Die Funktion benutzt alle Register bis auf DS und ES. Um den Cache zurückzusetzen, sollte aber die Funktion 0DH (Reset disk) des INT 21 verwendet werden, da diese auch die DOS-Datenstrukturen initialisiert.

SMARTDRV - Get Status & Set Cache Drive (INT 2F)

Diese Funktion erlaubt eine Statusabfrage bestimmter Cacheeinstellungen und ermöglicht einzelne Laufwerke auf Lese-/Schreibcache umzusetzen.

```
Ö------------------------------Ì
°       CALL: INT 2F         °
°                            °
° AX: 4A10H SMARTDRV 4.X      °
° BX: 0003H Status            °
° BP: Drive (0=A, 1=B, etc.)  °
° DL: Subfunktion             °
°    00H Get Info             °
°    01H Read Cache ein       °
°    02H Read Cache aus       °
°    03H Write Cache ein      °
°    04H Write Cache aus      °
û------------------------------À
°       Return:              °
° AX: BABEH falls OK          °
° DL: Status                  °
°  bit 7: 1 kein Cache        °
°  bit 6: 1 kein Schreibcache °
°  bit 5 --                   °
°  bits 0-4 Drive (0=A, 1=B..)°
°  FFH Laufw. existiert nicht °
Û------------------------------ì
```

In BP ist die Laufwerksnummer beim Aufruf zu übergeben. Das Register DL definiert beim Aufruf die auszuführende Subfunktion. Damit lassen sich verschiedenen Cachefunktionen für das Laufwerk ein- oder ausschalten. Nach dem Aufruf enthält DL einige Statusbits und den Laufwerksnamen. Schreib- und Lesecache arbeiten unabhängig voneinander und können getrennt ein- oder ausgeschaltet werden. Die Funktion benutzt alle Register bis auf DS, ES.

SMARTDRV - Get Cache Info (INT 2F)

Dies Funktion ermittelt die durch SMARTDRV reservierte Cachegröße.

```
Ö------------------------------Ì
°       CALL: INT 2F         °
°                            °
° AX: 4A10H SMARTDRV 4.X      °
° BX: 0004H Get Cache Size    °
û------------------------------À
°       Return:              °
° AX: Cache Blocks in DOS     °
° BX: Zahl der Cache Blocks   °
° CX: Größe Block in Byte     °
° DX: Cache BLocks in Windows °
```

```
Û------------------------------ì
```

In BX wird die aktuelle Zahl der Cacheblöcke zurückgegeben. Die Blockgröße in Byte findet sich in CX. DX definiert die Cachegröße (in Blocks), die unter Windows eingestellt wird. In AX finden Sie die Zahl der Cacheblocks unter DOS. Die Funktion benutzt alle Register bis auf DS, ES.

SMARTDRV - Get Double Buffer Status (INT 2F)

Diese Funktion liest den Double-Buffer-Status von SMARTDRV.

```
Ö------------------------------Ì
°       CALL: INT 2F        °
°                           °
° AX: 4A10H SMARTDRV 4.X      °
° BX: 0005H Double Buf Status  °
û------------------------------À
°       Return:             °
° AX: BABEH Double Buffer on   °
° ES:DI Pointer auf Puffer     °
Û------------------------------ì
```

Beo eingeschalteter Doppelpufferung enthält AX nach dem Aufruf die Signatur BABEH. In ES:DI findet sich ein Zeiger in den internen SMARTDRIVE-Datenbereich mit dem *double buffering byte array*.

```
ES:DI │ Physikalisches Laufwerk
──────┼──────────────────────────
+00   │ 80H
+01   │ 81H
....  │
```

Jedes Byte ab Offset 00H ist der Nummer eines physikalischen Laufwerkes (z.B: 80H, 81H etc.) zugeordnet. Diese Laufwerksnummern werden auch beim INT 13H benutzt. Ein Wert von FFH im betreffenden Byte signalisiert eine eingeschaltete Doppelpufferung für dieses Laufwerk. Bei Werte von 0 bis 4 ist unbekannt, ob für das Laufwerk eine Doppelpufferung erforderlich ist. Bei Werten größer gleich 5 erfordert das Laufwerk eine Doppelpufferung. Der Aufruf verwendet alle Register bis auf DS.

SMARTDRV - Check if drive cacheable (INT 2F)

Mit dieser Funktion läßt sich prüfen, ob ein Laufwerk durch den Cache unterstützt wird.

```
Ö------------------------------Ì
°       CALL: INT 2F        °
°                           °
° AX: 4A10H SMARTDRV 4.X      °
° BX: 0006H Drive cacheable   °
° CX: drive (1=A, 2=B, etc.)  °
û------------------------------À
```

```
°       Return:          °
° AX: 0006H drive not cacheable°
Û------------------------------ì
```

Wird in AX der Wert 0006H zurückgegeben, wird das betreffende Laufwerk nicht per Cache unterstützt. Der Aufruf wird von SMARTDRV beim Start benutzt um die unterstützten Laufwerke zu ermitteln. Trebier, die vor SMARTDRV geladen werden, können diesen Aufruf abfangen und SMARTDRV anschließend mitteilen, daß Caching nicht unterstützt werden soll. Der Aufruf verändert alle Register.

SMARTDRV - Get Device Driver for Drive (INT 2F)

Der Aufruf ermittelt die Adresse des Headers des Laufwerkstreibers, der mit dem SMARTDRV-Laufwerk korrespondiert.

```
Ö------------------------------Ì
°       CALL: INT 2F       °
°                          °
° AX: 4A10H SMARTDRV 4.X      °
° BX: 0007H Get driver adr.   °
° BP: drive (1=A, 2=B, etc.)  °
û------------------------------À
°       Return:          °
° DL:   Unit Code          °
° ES:DI Device Adresse       °
Û------------------------------ì
```

In ES:DI wird ein Zeiger auf den Kopf des Einheitentreibers zurückgegeben. DL enthält den Unit Code des Device Treibers. Der Aufruf ist für Utilities hilfreich, die einen Zugriff auf den Kopf des Treibers benötigen. Der Aufruf beeinflußt alle Register bis auf DS.

SMARTDRV - Signal serious Error (INT 2F)

Diese Funktion zeigt ein Meldungsfenster, wenn ein kritischer Fehler im Cache auftritt.

```
Ö------------------------------Ì
°       CALL: INT 2F       °
°                          °
° AX: 4A10H SMARTDRV 4.X      °
° BX: 1234H Error          °
û------------------------------À
°       Return:          °
° --                       °
Û------------------------------ì
```

Die Funktion wird nur intern von SMARTDRV benutzt.

Das DBLSPACE Interface

Das Programm DBLSPACE wird ab MS-DOS 6.0 ausgeliefert. DBLSPACE nutzt die INT 2F-Schnittstelle (AX=4A11H), die nachfolgend beschrieben wird.

DBLSPACE - Installation Check (INT 2F)

Dieser Aufruf dient zur Abfrage des Installationsstatus und zur Ermittlung der Versionsnummer.

```
Ö------------------------------Ì
°       CALL: INT 2F         °
°                            °
° AX: 4A11H DBLSPACE          °
° BX: 0000H Install check     °
û------------------------------À
°       Return:              °
° AX: 0000H OK                °
° BX: 444DH Signatur ("DM")   °
° CL: 1. DBLSPACE drive (0=A:) °
° CH: Zahl der DBLSPACE drives °
° DX: Versionsnummer (bits 14-0°
°  bit 15:1 Location flag     °
Û------------------------------ì
```

In CL wird das erste durch DBLSPACE verwaltete Laufwerk zurückgegeben. CH enthält die Zahl der durch DBLSPACE verwalteten Laufwerke. Die Bits 0-14 in DX definieren die interne Versionsnummer von DBLSPACE.BIN. Bit 15 = 1 signalisiert, daß DBLSPACE.BIN noch nicht an die endgültige Position im Speicher verschoben wurde.

DBLSPACE - Get Drive Mapping (INT 2F)

Dieser Aufruf ermittelt die Zuordnung der Hostlaufwerke.

```
Ö------------------------------Ì
°       CALL: INT 2F         °
°                            °
° AX: 4A11H DBLSPACE          °
° BX: 0001H Get drive mapping  °
° DL = Drive Nummer (0=A:)    °
û------------------------------À
°       Return:              °
° AX: Status                  °
° BL: Host Drive              °
°  (bit 7=1 Drive compressed)  °
° BH: DBLSpace Sequenznummer   °
Û------------------------------ì
```

Die komprimierten Volumendateien werden auf dem Hostlaufwerk als DBLSPACE.SEQ abgelegt, wobei SEQ eine fortlaufende Nummer ist. Diese wird in BH zurückgegeben. BL

definiert den Namen des Hostlaufwerkes. In AL werden folgende Statuswerte zurückgeliefert:

```
Ö-------Ú----------------------Ì
° Code  ° Status               °
û-------é----------------------À
° 0000H ° successful           °
° 0100H ° bad function         °
° 0101H ° invalid drive        °
° 0102H ° not a compressed drive°
° 0103H ° drive already swapped °
° 0104H ° drive not swapped    °
Û-------Ù----------------------ì
```

Nur falls der Status AX = 0000H zurückgegeben wird, sollten Sie die restlichen Register analysieren.

DBLSPACE - Swap Drive Letters of CVF & Host (INT 2F)

Mit diesem Aufruf läßt sich die Laufwerksbezeichnung zwischen Hostlaufwerk und DBLSPACE-Laufwerk austauschen.

```
Ö------------------------------Ì
°       CALL: INT 2F         °
°                            °
° AX: 4A11H DBLSPACE         °
° BX: 0002H Swap drive letter °
° DL = Drive Nummer (0=A:)   °
û------------------------------À
°       Return:              °
° AX: Status                 °
Û------------------------------ì
```

In DL wird die Laufwerksnummer des komprimierten Laufwerkes übergeben. Dieses wird mit der Laufwerksnummer des Hostlaufwerkes ausgetauscht. In AX gibt die Funktion den Statuscode (0000H, 0101H, 0102H, 0103H) zurück. Hierbei gilt die Kodierung wie bei der Funktion Get Drive Mapping.

DBLSPACE - Get Device Driver Entry Points (INT 2F)

Diese Funktion ermittelt den Einsprungpunkt für den Einheitentreiber.

```
Ö------------------------------Ì
°       CALL: INT 2F         °
°                            °
° AX: 4A11H DBLSPACE         °
° BX: 0003H Get Devixe Adr.  °
° CL = Drive Nummer (0=A:)   °
û------------------------------À
°       Return:              °
```

° CL: FFH Fehler °
° ES:SI Treiber strategie rout.°
° ES:DI Treiber interrupt rout.°
Û----------ì

In CL ist die Laufwerksnummer (0=A:) des komprimierten Laufwerkes zu übergeben. Enthält CL bei der Rückkehr den Wert FFH, wurde kein komprimiertes Laufwerk beim Aufruf definiert. Bei erfolgreichem Aufruf enthalten ES:SI und ES:DI die Zeiger auf die Strategie- und Interruptroutinen des Treibers.

Die offizielle Dokumentation reserviert diese Funktion für SMARTDRV, damit dieses Programm den Treiber verwenden kann.

DBLSPACE - Set Device Driver Entry Points (INT 2F)

Diese Funktion setzt den Einsprungpunkt für den Einheitentreiber.

Ö----------Ì
° CALL: INT 2F °
° °
° AX: 4A11H DBLSPACE °
° BX: 0004H Set Devixe Adr. °
° CL = Drive Nummer (0=A:) °
° ES:SI Treiber strategie rout.°
° ES:DI Treiber interrupt rout.°
û----------À
° Return: °
° CL: FFH Fehler °
Û----------ì

In CL ist die Laufwerksnummer (0=A:) des komprimierten Laufwerkes zu übergeben. Enthält CL bei der Rückkehr den Wert FFH, wurde kein komprimiertes Laufwerk beim Aufruf definiert. Vor dem Aufruf sind in ES:SI und ES:DI die Zeiger auf die Strategie- und Interruptroutinen des Treibers zu übergeben.

DBLSPACE.BIN - Mount Compressed Drive (INT 2F)

Mit dieser Funktion läßt sich ein komprimiertes Medium mounten.

Ö----------Ì
° CALL: INT 2F °
° °
° AX: 4A11H DBLSPACE °
° BX: 0005H Mount device °
° DL = Drive Nummer (0=A:) °
° ES:SI Activation record °
û----------À
° Return: °
° ES:SI Status °
Û----------ì

In DL wird die Laufwerksnummer des komprimierten Mediums angegeben. Das Registerpaar ES:SI zeigt auf einen Puffer, in dem die Funktion den Status des Activation records zurückgibt. Dieser besitzt folgenden Aufbau:

```
Ö------Ú-----Ú-----------------------------------Ì
°Offset° Byte° Beschreibung                      °
û------é-----é-----------------------------------À
° 00H  °  2  ° Signatur "MD" (4Dh 44h)           °
° 02H  °  1  ° 4Dh ('M') Mount Kommando          °
° 03H  °  1  ° Fehlercode (FFH vor dem Aufruf)   °
°      °     ° 01H Laufwerksbuchstabe für        °
°      °     °  DBLSPACEdrive nicht verfügbar    °
°      °     ° 02H Laufwerkbuchstabe belegt      °
°      °     ° 03H keine freien Diskeinheiten    °
°      °     ° 09H CVF zu fragmententiert        °
°04H   °  1  ° Host Laufwerk (0=A:)              °
°05H   °     ° --                                °
Û------Ù-----Ù-----------------------------------ì
```

DBLSPACE - Unmount Compressed Drive (INT 2F)

Dieser Aufruf gibt ein geladenes komprimiertes Laufwerk wieder frei (unmount).

```
Ö------------------------------Ì
°        CALL: INT 2F         °
°                             °
° AX: 4A11H DBLSPACE           °
° BX: 0006H Unmount device     °
° DL = Drive Nummer (0=A:)     °
û------------------------------À
°        Return:              °
° AX: Status                   °
Û------------------------------ì
```

In DL wird die Laufwerksnummer des komprimierten Mediums angegeben. Nach dem Aufruf enthält AX den Statuscode 0000H oder 0102H (siehe AX=4A11H/BX=0001H).

DBLSPACE - Get Space available on compressed Drive (INT 2F)

Dieser Aufruf ermittelt die freie Speicherkapazität auf dem komprimierten Laufwerk.

```
Ö------------------------------Ì
°        CALL: INT 2F         °
°                             °
° AX: 4A11H DBLSPACE           °
° BX: 0007H Get space          °
° DL = Drive Nummer (0=A:)     °
û------------------------------À
°        Return:              °
° AX: Status                   °
```

```
° DS:SI Freie Kapazität       °
Û------------------------------ì
```

Der Status in AX kann die Werte 0000H und 0102H annehmen (siehe BX=0001H). Bei erfolgreichem Aufruf enthalten DS:SI die Adresse auf den Record mit der Beschreibung der freien Kapazität. Dieser besitzt folgenden Aufbau:

```
Ö--------Ú-------Ú------------------------------Ì
° Offset ° Bytes ° Beschreibung                 °
û--------é-------é------------------------------À
°   00H  °    4  ° Sektorenzahl im Sektor       °
°        °       ° heap des Laufwerks           °
°   04H  °    4  ° Zahl der freien Sektoren     °
°        °       ° im Sektor heap des Laufwerks °
Û--------Ù-------Ù------------------------------ì
```

DBLSPACE - Get Size of Fragment Heap (INT 2F)

Dieser Aufruf ermittelt die Größe des Fragmentierungs Heaps.

```
Ö------------------------------Ì
°        CALL: INT 2F         °
°                             °
° AX: 4A11H DBLSPACE          °
° BX: 0008H Get Frag. heap    °
° DL = Drive Nummer (0=A:)    °
û------------------------------À
°        Return:              °
° AX: Status                  °
° BX: max. Einträge           °
° CX: verfügbare Einträge     °
Û------------------------------ì
```

In AX wird der Statuscode 0000H oder 0102H zurückgegeben (siehe BX=0001H). Bei erfolgreichem Aufruf (AX = 0) enthält BX die maximale Zahl der Einträge im File Fragment Heap. In CX steht die Zahl der verfügbaren Einträge im File Fragment Heap.

DBLSPACE - Determine Number of Disk_Unit Strutures (INT 2F)

Dieser Aufruf ermittelt die Zahl der Disk-Units.

```
Ö------------------------------Ì
°        CALL: INT 2F         °
°                             °
° AX: 4A11H DBLSPACE          °
° BX: 0009H Get disk units    °
° DL = Drive Nummer (0=A:)    °
û------------------------------À
°        Return:              °
° AX: Status                  °
```

° CL: DISK_Unit Strukturen °
Û------------------------------ì

In AX wird der Status 0000H bei erfolgreichem Aufruf zurückgegeben. In CL steht dann die Zahl der allokierten DISK_UNIT-Strukturen.

DBLSPACE - Relocate (INT 2F)

Diese Funktion erlaubt DBLSPACE.BIN im Speicher zu verschieben.

Ö------------------------------Ì
° CALL: INT 2F °
° °
° AX: 4A11H DBLSPACE °
° BX: FFEEH Relocate °
° ES: Segment °
û------------------------------À
° Return: °
° --- °
Û------------------------------ì

In ES wird die Segmentadresse übergeben, zu der DBLSPACE.BIN zu verschieben ist. Diese Funktion wird durch DBLSPACE.SYS aufgerufen um DBLSPACE.BIN an die endgültige Speicherposition zu verschieben.

DBLSPACE - Get Relocation Size (INT 2F)

Mit diesem Aufruf läßt sich die Größe des zu verschiebenden Treibers ermitteln.

Ö------------------------------Ì
° CALL: INT 2F °
° °
° AX: 4A11H DBLSPACE °
° BX: FFFFH Get Reloc. Size °
û------------------------------À
° Return: °
° AX: Paragraphenzahl °
Û------------------------------ì

In AX wird die Zahl der durch DBLSPACE.BIN belegten Paragraphen (16 Byte) zurückgegeben. DBLSPACE.SYS verwendet diesen Aufruf.

Microsoft Realtime Compression Interface (MRCI) - RAM-Based Server (INT 2F)

Unter den INT 2F läßt sich das Realttime Compression Interface von DBLSPACE ansprechen.

Ö------------------------------Ì
° CALL: INT 2F °

```
°                          °
° AX: 4A12H MRCI           °
° CX: 4D52H ("MR")         °
° DX: 4349H ("CI")         °
û----------------------------À
°       Return:            °
° CX: 4943H ("IC") installiert °
° DX: 524dH ("RM") installiert °
° ES:DI MRC-Info-Struktur  °
Û----------------------------ì
```

Beim Aufruf wird in CX:DX der String "MRCI" übergeben. Enthält CX nach dem Aufruf die gedrehte Signatur "IC", ist das Interface installiert. Im Register ES:DI wird ein Zeiger auf die MRCINFO-Struktur übergeben.

Power Management auf dem PC

Ab MS-DOS 5.0 wird bei bestimmten Geräten ein Power Management zur Schonung des Batteriebetriebs bei Laptops unterstützt.

Get Status Power Management (INT 15)

Über den INT 15 läßt sich feststellen, ob das BIOS dieses Power Management unterstützt.

```
Ö----------------------------Ì
°       CALL: INT 15       °
°                          °
° AX: 5300H Power Management  °
° BX: 0                    °
û----------------------------À
°       Return:            °
° CY: 0                    °
° BX: 504DH ("PM") installiert °
° AH: Hauptversion (BCD)     °
° AL: Unterversion (BCD)     °
° CX: Flags                  °
°  Bit 0: 16 Bit Protected Mode°
°  Bit 1: 32 Bit Protected Mode°
°  Bit 2: CPU Idle Call ok   °
°  Bit 3: BIOS Power Mgmt off  °
Û----------------------------ì
```

Ist beim Rücksprung CY=0 wird die Funktion unterstützt. Bei BX=504DH unterstützt der PC das Power Management. In AX findet sich dann die Versionsnummer der Routinen. Im Flag in CX definiert ein gesetztes Bit die betreffende Option. Das Powermanagement wird sowohl im 16- als im 32-Bit-Protected-Mode unterstützt. Ist Bit 2 = 1 gesetzt, unterstützt die Software eine Verringerung des Prozessortaktes beim CPU-Idle-Call. Wird die Funktion nicht unterstützt, ist nach dem Aufruf das Carry Flag gesetzt und AH = 86H.

Power Management - Interface Connect (INT 15)

Über diesen INT 15-Aufruf wird das Interface zwischen dem aufrufenden Programm und dem System-BIOS aktiviert.

```
Ö------------------------------Ì
°       CALL: INT 15         °
°                            °
° AX: 5301H Interf. connect    °
° BX: 0000H System BIOS        °
û------------------------------À
°       Return:              °
° CY: 0 ok                    °
° CY: 1 Fehler                °
° AX: Fehlercode              °
Û------------------------------ì
```

Bei fehlerhaftem Aufruf wird CY=1 und AX = 02H (Interface connection already in effect) oder AX = 09H (Unrecognized device ID) zurückgegeben.

Power Management - Protected Mode Interface (INT 15)

Der Aufruf (AX=5302H) erlaubt die Einrichtung des 16-Bit-Protected Mode-Interface im Real oder V86-Mode. Der Aufruf AX=5303H definiert die gleiche Funktion für das 32-Bit-Protected Mode-Interface.

Power Management - Disconnect Interface (INT 15)

Über diesen INT 15-Aufruf wird das Interface zwischen dem aufrufenden Programm und dem System-BIOS deaktiviert.

```
Ö------------------------------Ì
°       CALL: INT 15         °
°                            °
° AX: 5304H Interf. disconnect °
° BX: 0000H System BIOS        °
û------------------------------À
°       Return:              °
° CY: 0 ok                    °
° CY: 1 Fehler                °
° AX: Fehlercode              °
Û------------------------------ì
```

Bei fehlerhaftem Aufruf wird CY=1 und AX = 03H (Interface not connected) oder AX = 09H (Unrecognized device ID) zurückgegeben.

Power Management - CPU Idle (INT 15)

Über diesen INT 15-Aufruf läßt sich der IDLE-Status des Power Managements aktivieren.

```
Ö------------------------------Ì
°       CALL: INT 15        °
°                           °
° AX: 5305H CPU Idle         °
û------------------------------À
°       Return:             °
° CY: 0 ok                   °
Û------------------------------ì
```

Die Reaktionen des Systems hängen dann vom implementierten Funktionsumfang der Power-Management-Routinen ab.

Power Management - CPU Busy (INT 15)

Über diesen INT 15-Aufruf kann ein Prozess dem Power Management mitteilen, daß die Anwendung wieder aktiv ist.

```
Ö------------------------------Ì
°       CALL: INT 15        °
°                           °
° AX: 5306H CPU Busy         °
û------------------------------À
°       Return:             °
° CY: 0 ok                   °
Û------------------------------ì
```

Normalerweise sollte das System bei Interrupts etc. selbst die Aktivitäten erkennen und den Modus umschalten. Wenn externe Software jedoch die BIOS-Routinen umgeht, muß der Aufruf explizit erfolgen. Die Reaktionen des Systems hängen dann vom implementierten Funktionsumfang der Power-Management-Routinen ab (z.B. CPU-Takt erhöhen).

Power Management - Set Power State (INT 15)

Über diesen INT 15-Aufruf kann eine Systemkomponente in einen definierten Zustand geschaltet werden (sofern das BIOS dies unterstützt).

```
Ö------------------------------Ì
°       CALL: INT 15        °
°                           °
° AX: 5307H Set Power State  °
° BX: Power Devixe ID        °
° CX: System State ID        °
°    0000H Ready             °
°    0001H Stand-by          °
°    0002H Suspend           °
°    0003H Off               °
°    0004H-FFFFH reserviert  °
û------------------------------À
°       Return:             °
° CY: 0 ok                   °
```

```
° CY: 1 Fehler                °
° AH: Fehlercode              °
Û-------------------------------ì
```

Die System Status-Codes 0H und 03H (in CX) sind bei der Einheit (System ID 0001H) nicht zulässig. Als Fehlercodes werden folgende Meldungen zurückgegeben:

```
Ö------Ú----------------------------Ì
° Code ° Bemerkung                  °
û------é----------------------------À
° 01H  ° Power Management disabled  °
° 09H  ° Device ID unbekannt        °
° 0AH  ° ungültiger Wert in CX      °
° 60H  ° Status nicht aktivierbar   °
Û------Ù----------------------------ì
```

Das Interface ist per INT 15H und für die Protected Mode-Schnittstelle verfügbar.

Mit der Kombination BX=0001H und CX = 0001H läßt sich das System in den *Stand-by* Zustand versetzen.

Der gleiche Aufruf mit CX=0002H hebt diesen Zustand wieder auf.

Power Management - Status/Control (INT 15)

Dieser Aufruf setzt oder sperrt die automatische Power down-Funktion des Power Managements.

```
Ö-------------------------------Ì
°         CALL: INT 15        °
°                             °
° AX: 5308H Status/Control    °
° BX: FFFFH Enable/Disable    °
° CX: 0 disable               °
°     1 enable                °
û-------------------------------À
°         Return:             °
° CY: 0 ok                    °
° CY: 1 Fehler                °
° AH: Fehlercode              °
Û-------------------------------ì
```

Mit CX = 0 wird die Funktion gesperrt. Bei CY=1 ist ein Fehler aufgetreten. Dann enthält AH den Fehlercode.

```
Ö------Ú----------------------------Ì
° Code ° Bemerkung                  °
û------é----------------------------À
° 01H  ° Power Management disabled  °
° 09H  ° Device ID unbekannt        °
° 0AH  ° ungültiger Wert in CX      °
Û------Ù----------------------------ì
```

Das Interface ist per INT 15H und für die Protected Mode-Schnittstelle verfügbar.

Power Management - Restore Power On defaults (INT 15)

Dieser Aufruf initialisiert die Standardeinstellungen für die Power on-Funktion des Power Managements.

```
Ö------------------------------Ì
°       CALL: INT 15         °
°                            °
° AX: 5309H Status/Control    °
° BX: FFFFH                   °
û------------------------------À
°       Return:              °
° CY: 0 ok                    °
° CY: 1 Fehler                °
° AH: Fehlercode              °
Û------------------------------ì
```

Bei CY=1 wird nur der Fehlercode 09H (Device ID unbekannt) zurückgegeben. Das Interface ist per INT 15H und für die Protected Mode-Schnittstelle verfügbar.

Power Management - Get Power Status (INT 15)

Dieser Aufruf erlaubt die Abfrage des aktuellen Status.

```
Ö------------------------------Ì
°       CALL: INT 15         °
°                            °
° AX: 530AH Status/Control    °
° BX: 0001H                   °
û------------------------------À
°       Return:              °
° CY: 0 ok                    °
° BH: Status AC Leitung (Netz) °
°   0 = Off-Line              °
°   1 = On-Line               °
°  FF = unbekannt             °
° BL: Status Batterie         °
°   0 = gut                   °
°   1 = niedrig               °
°   2 = kritisch              °
°   3 = wird geladen          °
°   FF= unbekannt             °
° CL: Batterielebensdauer     °
°   0-100%                    °
°   FF= unbekannt             °
° CY: 1 Fehler                °
° AH: Fehlercode 09H          °
```

Û----------ì

In CL wird die Zeit angegeben, die das Akku noch Spannung liefern kann. Dies ist eine Prozentangabe, die sich auf eine neue 100% geladene Batterie bezieht.

Power Management - Event Notofication (INT 15)

Beim Power Management treten die Ereignisse meist asynchron auf (z.B. Benutzereingaben). Der Status läßt sich daher über die Funktion *Get PM Event* abrufen.

Ö----------Ì
° CALL: INT 15 °
° °
° AX: 530BH Get PM Event °
û----------À
° Return: °
° CY: 0 ok °
° BX: PM Event Code °
° 01H System Stand-by request °
° 02H System supend request °
° 03H Normal resume System °
° 04H Critical resume System °
° 05H Batterie Low °
° CY: 1 Fehler °
° AH: Fehlercode °
Û----------ì

Der Event-Code 01H in BX signalisiert, daß das Power Management das System in einen *Stand-by* Zustand schalten möchte. Bei fehlerhaftem Aufruf wird CY=1 zurückgegeben. AH enthält dann die Fehlercodes 03H (Interface connection not Established) oder 80H (No PM Events pending).

Glossar

CDS

Die *Current Directory Structure (CDS)* ist eine interne DOS-Datenstruktur, die die Zugriffe auf Dateien verwaltet. Die Handles aus dem PSP verweisen auf die CDS.

DCB

Beim *DOS Control Block* handelt es sich um eine interen DOS-Datenstruktur. Die Anfangsadresse wird durch den INT 21-Aufruf 52H zurückgeliefert.

DPB

Der *Disk Parameter Block (DPB)* enthält die Daten für die Verwaltung des Speichermediums. Der Block wird von DOS verwaltet und kann per INT 21 gelesen werden.

DTA

Die *Disk Transfer Area (DTA)* ist bei der Verwendung von FCB-Funktionen des INT 21 erforderlich. Es handelt sich um einen Puffer, in dem Daten von der Diskette/Platte zwischengepuffert werden.

EMS

Die *Expanded Memory Spezifikation (EMS)* beschreibt eine Möglichkeit im 1-Mbyte-Adressraum zusätzlichen Speicher einzublenden.

FCB

File Control Blocks (FCB) sind Datenstrukturen, die bei der Benutzung der älteren CP/M-orientierten INT 21-Aufrufe erforderlich sind. Im FCB werden Informationen über die Dateinamen abgelegt.

HMA

Die *High Memory Area (HMA)* ist der 64-Kbyte-RAM-Bereich, der direkt oberhalb von 1 Mbyte beginnt. Die HMA wird durch den XMS-Treiber verwaltet.

IFS

Das *Installable File System (IFS)* ist ein Ansatz unter DOS, die Routinen der Dateiverwaltung austauschbar zu machen. Das IFS wird zur Zeit nur bei Netzwerkbetrieb und bei CD-ROM's benutzt.

JFT

Die *Job File Table (JFT)* wird im PSP angelegt und enthält die Handles zum Zugriff auf die Dateien. Die Handles verweisen auf den jeweiligen Eintrag in der System File Table (SFT).

MCB

Die *Memory Control Blocks (MCB)* sind Datenstrukturen, die DOS zur Verwaltung von Speicherblöcken im Hauptspeicher dienen.

PSP

Das *Programm Segment Prefix (PSP)* ist eine 256 Byte lange Datenstruktur, die von DOS vor jedem Programm im Speicher angelegt wird. Im PSP finden sich Rücksprungadressen und andere Verwaltungsinformationen über den Prozeß.

SDA

Die *System Data Area (SDA)* ist ein anderer Ausdruck für den DOS Control Block.

SFT

Die *System File Table (SFT)* ist eine interne DOS Datenstruktur, in der die Verwaltung der offenen Dateien erfolgt. Die Hanldes aus der Job File Table verweisen auf die SFT.

XMS

Die *Extended Memory Specification (XMS)* erlaubt die Verwaltung des Speicherbereiches oberhalb von 1 Mbyte auf 20286/80386-Rechnern.

Literatur

/1/ Born, Günter: Systemtuning mit TSR-Programmen. Grundlagen, Tips und Utilities zur Programmierung residenter Programme. Addison Wesley Verlag, Bonn, 1989, ISBN 3-89319-262-X.

/2/ Michael, M.: VGA-Kompendium. Markt & Technik Verlag, München, 1988, ISBN 3-89090-663-X

/3/ Schulman, et. al: Undocumented DOS, Addison Wesley Verlag, Bonn, 1991, ISBN 0-201-57064-5

/4/ Schäpers, A.: DOS 5 für Programmierer, Addison Wesley Verlag, Bonn, 1991, ISBN 3-89319-350-2

/5/ Born, G.: Referenzhandbuch Fileformate, Addison Wesley Verlag, Bonn 1990, ISBN 3-89319-302-2.

/6/ Microsoft Windows SDK-Kit, Dokumentation

/7/ Born, G.: DOS 5 Tuning. Tips, Tricks und Utilities zur Installation, zur Konfigurierung und zur Batchprogrammierung. Markt & Technik Verlag, 1991, ISBN N3-87791-196-X

/8/ Microsoft MS-DOS 5, Technische Referenz. Microsoft Press, 1991, ISBN 3-86063-201-9.

Stichwortverzeichnis