

DOS EXTRA

18,- DM 145 Os 18,- sfr

Sonderausgabe DOS-EXTRA

Nr. 1 1987/88

DMV-Verlag

Eine Publikation der
DOS-INTERNATIONAL

So ist ein PC aufgebaut

- ◇ Portadressen
- ◇ Grafikkarten
- ◇ Controller

MS-DOS richtig ausgenutzt

- ◇ Alle Funktionen
- ◇ Umleitungen
- ◇ Filter

BIOS

Sämtliche
Interrupts
auf einen Blick

Assembler für Einsteiger

- ◇ Einführung
- ◇ Befehle und Bedeutung

Ein ganzes Heft voll Tips & Tricks
für MS-DOS-computer



Schreiben Sie erster Klasse mit ...

1st WORD PLUS

Textverarbeitungssystem der Spitzenklasse für IBM-PC, Schneider-PC und Kompatible:

- ★ außergewöhnlich einfach zu erlernen und zu bedienen
- ★ vollständig GEM™-gesteuert, mit Mausunterstützung, Drop-Down-Menüs, und bis zu vier Textfenstern gleichzeitig
- ★ ständiger Hilfe-Modus zuschaltbar (für Anfänger)
- ★ Ausschneiden und Einfügen von Blöcken innerhalb eines Dokumentes, oder aus einem Textfenster in ein anderes
- ★ gleichzeitige Darstellung, Bearbeitung und Ausdruck von Text und Graphik
- ★ Serienbriefgestaltung (1st Mail)
- ★ alle Druckarten genau am Bildschirm dargestellt (WYSIWYG), wie Fettschrift, Unterstreichen, Kursivschrift, Entwurfsdruck, Hoch- und Tiefstellen, Sonderzeichen, usw.
- ★ links- und rechtsbündiger und zentrierter Text
- ★ komfortable Such- und Ersetzfunktionen
- ★ mehrspaltige Druckmöglichkeit (1st Mail)
- ★ automatische Fußnotenverwaltung
- ★ eingebautes Wörterbuch zur Rechtschreibhilfe mit 40000 Wörtern, einschließlich Funktionen zum Durchblättern, Hinzufügen und Löschen von Wörtern



Text und Graphik gleichzeitig darstellen und ausdrucken



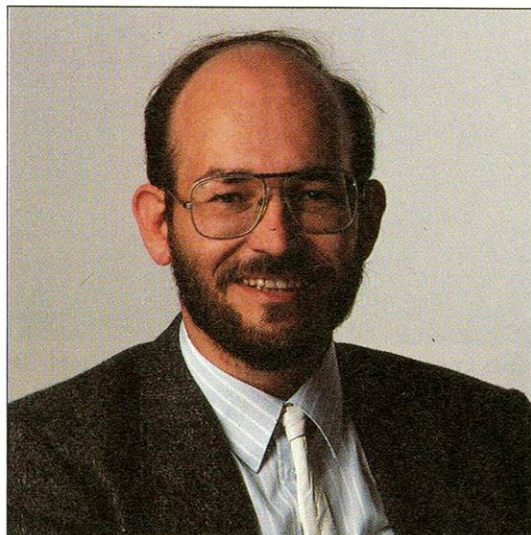
Einfachste Bedienung durch die komfortable GEM-Steuerung

1st WORD PLUS

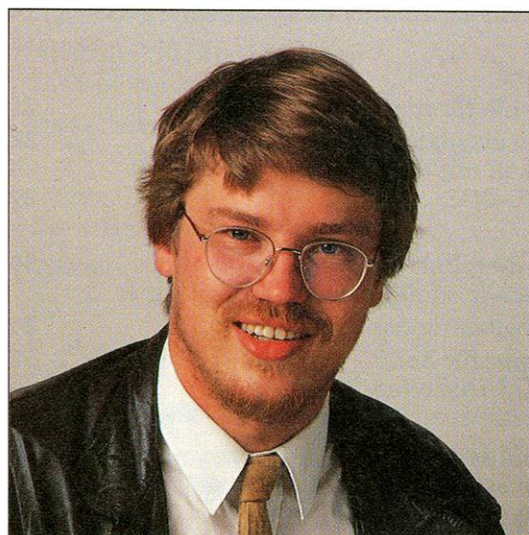
die Textverarbeitung, die keine Wünsche offen läßt

DM 299,-

Kennen Sie Ihren PC?



Volker Everts



Andreas Hagedorn

Sechs Jahre ist es her, daß sich der erste IBM-PC der breiten Öffentlichkeit vorstellte – mit 64 KByte Speicher, einem Kassettenrecorder als Massenspeicher und nur einem monochromen Monitor, der noch nicht einmal grafikfähig war. Das alles kostete damals weit mehr Geld, als Sie heute für die am besten ausgestatteten PCs anlegen müssen. Doch seitdem revolutionierte sich die Welt der Computer grundlegend.

In keinem Bereich der Technik gab es innerhalb so kurzer Zeit eine so rasante Entwicklung – bei gleichzeitig so drastischem Preisverfall – wie in der Computerbranche. Ganz klar, daß da der IBM-PC und seine kompatiblen Brüder sich nach und nach zu äußerst preiswerten und leistungsfähigen Geräten entwickelten.

Was nicht mithalten konnte, waren die Informationen rund um den Computer. Eine Unmenge an immer neuen Zeitschriften und Büchern überschwemmten in den letzten Jahren den Markt, doch unseres Wissens nach hat es bis heute keine Publikation geschafft, die technischen Informationen über PCs in komprimierter Form in einem einzigen Heft zu vereinigen.

Dieses Manko wollen wir mit diesem Sonderheft der DOS beheben, in dem Sie all das Wissen finden, das zur perfekten Beherrschung Ihres Computers notwendig ist. Die DOS International liefert Ihnen zwar bereits Monat für Monat jede Menge Tips und Tricks zu der Thematik MS-DOS, doch gibt es in der Welt der PCs noch viel zu erklären.

Damit Sie alle Informationen auch optimal verwerten können, vermittelt unser DOS-Sonderheft den Einstieg in noch nicht so geläufige Thematiken. Assembler bei-

spielsweise ist nach unserem Einführungskurs in die Maschinensprache auch für Nichtprofis kein Geheimnis mehr. Und Ihr Betriebssystem MS-DOS können Sie noch besser ausnutzen, wenn Sie MS-DOS intern kennen.

Die Zeit ist reif für mehr Informationen über Ihren Personalcomputer. Dafür halten Sie jetzt das erste DOS EXTRA in der Hand – DOS EXTRA, das ist die Sonderheftreihe von DOS International.

Denjenigen, die uns dabei geholfen haben, das Heft zusammenzustellen und die uns Ihr Wissen zur Verfügung gestellt haben, sei Dank gesagt – und wenn es sich dabei nur um einfache Fragen handelte, die uns in der Redaktion die notwendigen Denkanstöße gegeben haben. Denn gerade diese Fragen zeigen uns »Fachleuten« oft, daß wir so manches Mal schon in zu hohen Sphären schweben.

Und damit wünschen wir viel Spaß beim Lesen des ersten DOS EXTRA.

Ihr

Andreas Hagedorn
Andreas Hagedorn

Volker Everts
Volker Everts

MS-DOS:

für Anwender

MS-DOS-Befehle – und wie Sie sie zum Blühen bringen	6
– sämtliche Befehle auf einen Blick	
Edlin, der zeilenorientierte Editor unter MS-DOS	14
Der Debugger	18
– »debug.com«	
Die Batchbefehle für Stapeldateien	21
Die Konfigurationsdatei »config.sys«	22
Daten umlenken mit MS-DOS	24
Filtern mit MS-DOS	27

intern

MS-DOS im Überblick	30
– Referenztabelle sämtlicher MS-DOS-Befehle	
Die Speicherbelegung des PC	34
Durchblick Spur für Spur	35
– Das MS-DOS-Disketten-Format	

Funktionen

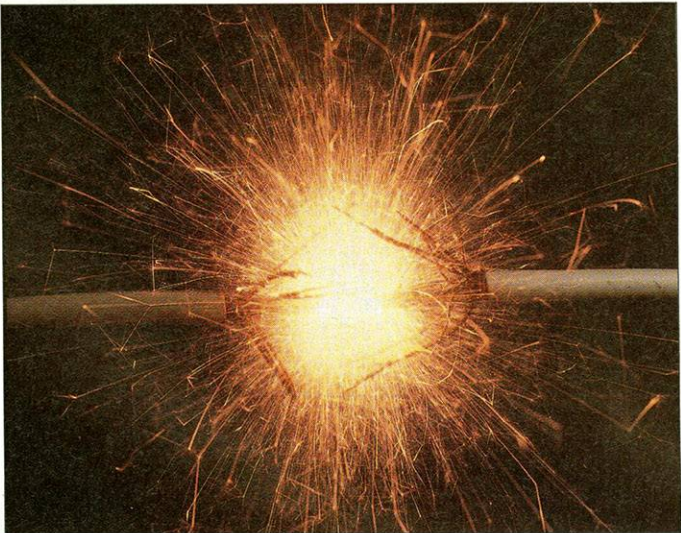
Der MS-DOS-Interrupt 21	38
– Alle Betriebssystemroutinen direkt aufgerufen	
MS-DOS-Funktionen in der Praxis	45
Turbo-Tempo für GW-Basic	46
– Funktionsaufrufe unter GW-Basic	
Turbo Pascal und MS-DOS	50
– Funktionsaufrufe unter Turbo Pascal	

Assembler:

Im Dickicht des Assemblercode: Programmieren in Maschiensprache	62
– Assembler-Kurs (Teil 1)	
Com und Exe – gewichtige Unterschiede	66
– Assembler-Kurs (Teil 2)	
MASM – ein leistungsfähiger Assembler	72
– Assembler-Kurs (Teil 3)	
SAM – der DOS-Assembler	76
– Der Assembler von der Databox	
Der Prozessor im PC	78
– Der 8088/8086 von Intel	
Die Assemblerbefehle des PC	80
– Der Befehlssatz des 8088/8086	
Der Arithmetikprozessor – das Rechengenie	90
– Der 8087 von Intel	
Umrechnung ganz einfach	94
– hexadezimale und dezimale Zahlen	

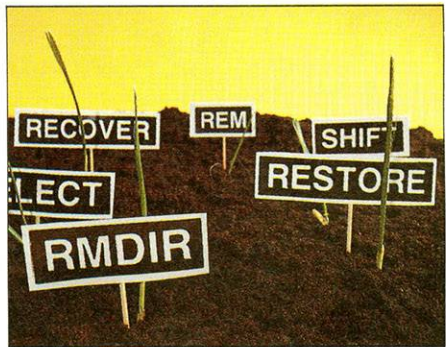
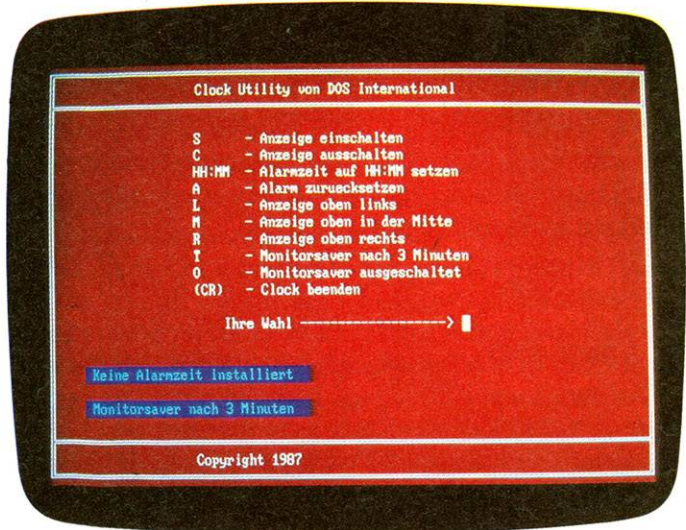
BIOS:

Nichts geht ohne Sie: die Software-Interrupts im PC	96
– Grundlagen	
Der Video-Interrupt 10	97
Der Interrupt 13 für Disketten und Festplatte	98
Der Interrupt 14 für serielle Schnittstellen	100
Der Interrupt 15 für Kassette	101
Der Tastatur-Interrupt 16	102
Der Drucker-Interrupt 17	103
Der Systemuhr-Interrupt 1A	103
Real-Time-Clock im AT manipulieren	161



Über Interrupts wird der PC gesteuert. Wie man Sie benutzt und auch unter Hochsprachen einsetzt, lesen Sie ab **S. 38**

MS-DOS ist nicht schwer, wenn man alle Befehle des Betriebssystems kennt und richtig einzusetzen weiß. Mit den richtigen Parametern werden auch die einfachsten Anweisungen ungeheuer leistungsfähig. **S. 36**



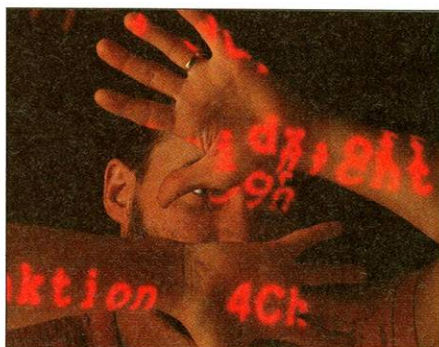
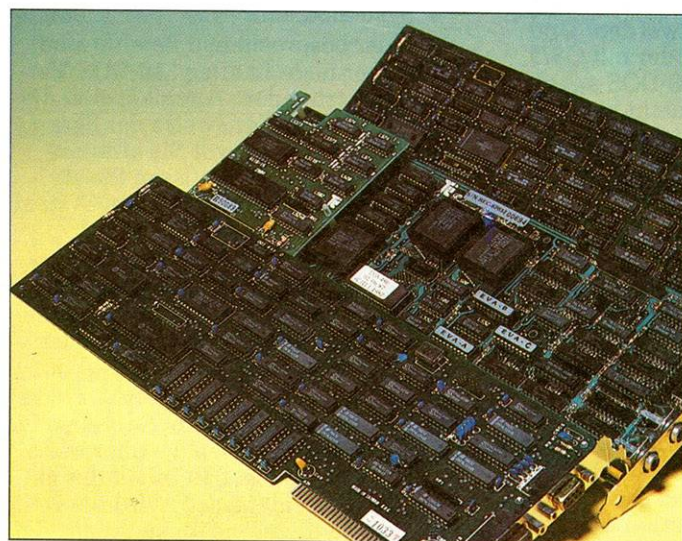
Nach drei Minuten wird der Bildschirm dunkel geschaltet, die Uhr ist ständig eingeblendet und zur vollen Stunde gibt es einen Signalton – das alles kann »Clock«. **S. 153**

Allgemeine Regeln für den Umgang mit den Tabellen in diesem Heft

- 1.) Alle numerischen Angaben (Adressen, Bytes und Maschinencode) sind hexadezimal notiert, sofern nicht »dezimal« angegeben ist.
- 2.) Ausdrücke in eckigen Klammern (»[« und »]«) sind wahlweise und dürfen weggelassen werden.
- 3.) Sind zwei oder mehrere Ausdrücke durch Schrägstriche (»/«) voneinander getrennt, so darf jeweils nur einer der aufgeführten Ausdrücke benutzt werden.
- 4.) Setzt sich ein Byte aus konstanten und variablen Bits zusammen, so sind die konstanten Bits binär als »0« und »1« angegeben, und die variablen Bits sind als Parameter an der entsprechenden Position im Binärwert aufgeführt.

Programmieren in Assembler kitzelt erst sämtliche Leistungsreserven aus Ihrem PC heraus. Mit »Im Dickicht des Assemblercode« startet die Einführung in Assembler.

S. 62



Die Grafikkarten des PCs auf einen Blick braucht jeder Programmierer, wenn er Programme für jedes Einsatzgebiet schreiben will.

S. 137

Hardware:

So ist ein PC aufgebaut	104
– Grundlagen	
Ausbau leicht gemacht – die Steckplätze des PC	106
Die Portadressen des PC	109
Anschluß für den Drucker:	
Die Centronics-Schnittstelle	113
Disketten unter Kontrolle	115
– Der Disketten-Controller UPD 765	
Bit für Bit:	
Die serielle Schnittstelle RS 232C	118
Wie aus Daten Bilder werden	119
– Der Video-Controller 6845	
Schnelle Speicherverwaltung	
mit dem DMA-Controller	123
Die Zeit im Griff mit dem Timerbaustein 8253	124
Unterbrechungen unter Kontrolle	
mit dem Interrupt-Controller 8259	126

Grafik:

Grundlagen von Grafik und Text beim PC	127
Grafik-Programmierung in GW-Basic	132
Farbgrafikkarte und Monochrom-Adapter	137
– CGA und monochrom	
Der Hercules-Karte ins Register geschaut	148
Die EGA-Karte klipp und klar	150

Utilities:

Hallo PC, wie spät ist es?	153
Systemnahe Programmierung in Turbo Pascal	158
Die DOS-Trickkiste	166
– Programme drucken mit dem Turbo-Editor	
– Copy-Befehl als Texteditor	
– Bit-Manipulation mit Turbo Pcsal	
– Tab-Funktion für Turbo Pascal	
– Farbiger Bildschirmrand	
– Basic-Aufruf mit Parametern	
– Caps-Lock per Programm	
– Basic-Datensegment feststellen	
– Grafik-Hardcopy	
– Basic-Tricks mit dem Shell-Befehl	
– Schnelle Integer-Division	

Rubriken:

Kennen Sie Ihren PC?	3
– Editorial	
Impressum	170
Inserentenverzeichnis	170



MS-DOS-Befehle

– und wie Sie sie zum Blühen bringen

Grundlage jedes IBM PC/XT und kompatiblen Computern ist das Betriebssystem MS-DOS (oder kompatible Software-Produkte wie PC-DOS). Viele PC-Einsteiger verwirrt die Befehls-Vielfalt und Syntax, doch bei genauerer Betrachtung ist der Umgang mit MS-DOS gar nicht schwer.

Am Anfang ging alles etwas durcheinander. IBM kündigte im März 1983 den Mikrocomputer PC an, zur selben Zeit gab Microsoft die Version 2.0 des Betriebssystems MS-DOS heraus. Doch erst ein Jahr später wurde mit der Version 2.05 von MS-DOS (für den Markt der Kompatiblen) auf die Bedürfnisse des internationalen Marktes eingegangen. Der Befehl »country« erlaubte dem Anwender, länderspezifische Datum- und Zeit-Formate zu benutzen.

Die Version für IBM trug übrigens einen anderen Namen (2.1) – und konnte ein bißchen mehr. Das ließ die Konkurrenz allerdings nicht auf sich sitzen. Um Forderungen der Hersteller kompatibler Personalcomputer nachzukommen, kombinierte Microsoft die beiden Versionen und so entstand MS-DOS 2.11, das eine weite Verbreitung fand.

Mit der Leistungsfähigkeit der Hardware mußte aber auch die Software – in diesem Fall das Betriebssystem »aufgebohrt« werden. Um den Anforderungen eines Netzwerkbetriebes nachzukommen, wurde im August 1984 die Version 3.0 (für IBM) vorgestellt. Die Version für alle anderen Computerhersteller hieß 3.05. Über die Zwischenstufe 3.1 wurde im Juni

1985 die Version MS-DOS 3.2 erreicht. Dieses Betriebssystem wird heute mit fast jedem kompatiblen PC ausgeliefert. Höhere Versionen existieren bereits, haben allerdings noch keine weitere Verbreitung gefunden.

Sie sehen, auch ein Betriebssystem muß sich immer seiner Zeit anpassen und entsprechend der Hardware immer leistungsfähiger werden. Und auch in einer Zeit, in der mit OS/2 ein Nachfolger für MS-DOS angekündigt ist, zählt dieses nicht zum alten Eisen. Microsoft sieht in weiter Zukunft einen Markt für OS/2 und für MS-DOS, so daß der PC-Besitzer auf immer leistungsfähigere Varianten von MS-DOS nicht verzichten muß.

Wie jede Software läßt sich auch MS-DOS in verschiedene Bestandteile gliedern. MS-DOS besteht im wesentlichen aus sechs Teilen:

- Das ROM-BIOS (ROM steht für Read Only Memory und BIOS für Basic Input Output System) stellt die grundlegenden Befehle, mit denen ein Computer arbeitet, zur Verfügung. Sie werden als residente Befehle bezeichnet.
- Im Boot-Sektor der Diskette steht der IPL (Initial Program Loader). Beim Start des PC lädt er das ei-

gentliche Betriebssystem. Dieses steht in den Dateien »msbio.com« und »msdos.com«. Beim IBM-PC heißen die Dateien mit dem PC-DOS »ibmbio.com« und »ibmdos.com«.

- »msbio.com« stellt die Verbindung mit dem ROM auf niedriger Ebene her. Zusammen mit ROM-BIOS verwaltet dieses Programm die Ein-/Ausgabefunktionen des Computers.
- »msdos.com« stellt die Benutzerprogramme auf hoher Ebene zur Verfügung. Es ist damit die »Steuerzentrale« des Betriebssystems und dem BIOS übergeordnet.
- »command.com« ist der »Befehls-Prozessor«. Er besteht aus mehreren Teilen und leitet die Befehle des Anwenders an das Betriebssystem weiter.
- Die externen Befehle sind der für den Anwender wichtigste Teil von MS-DOS. Im Folgenden wird ausschließlich dieser Teil behandelt, da die Funktionen der ersten fünf Bestandteile für den Anwender im Verborgenen abläuft.

MS-DOS kennt zwei grundlegend unterschiedliche Befehlsgruppen

Grundsätzlich lassen sich die Befehle von MS-DOS in zwei verschiedene Gruppen einteilen. Zum einen unterscheidet man interne und externe Befehle (siehe Tabelle 1), zum anderen unterteilt man sie nach ihren logischen Funktionen. Dies sind:

- Dateibefehle
- Verzeichnisbefehle
- Diskettenbefehle
- Gerätebefehle
- Konfigurationsbefehle
- Systembefehle

Die hier besprochenen Befehle sind nun aber nicht in allen MS-DOS-Versionen vorhanden. Auch hat sich die Zahl und Funktionsweise der Parameter geändert. Bevor Sie eine Anweisung benutzen, müssen Sie sie deshalb erst einmal auf Ihrem System ausprobieren.

Anweisungen in eckigen Klammern (beispielsweise »[Path]«) können wahlweise angegeben werden. Ohne die Klammern wird der Befehl aber auch bearbeitet.

append ist ein externer Befehl. Er definiert Pfade für Dateien. Soll eine Datei geöffnet werden, so überprüft MS-DOS zunächst den Inhalt des aktuellen Verzeichnisses. Wird die Datei hier nicht gefunden, sucht das Betriebssystem auf den mit »append« angegebenen Pfaden weiter. Ohne diese Anweisung würde sofort mit der Meldung »Datei nicht gefunden«

abgebrochen werden. »append« ohne Pfad-Angabe zeigt den definierten Pfad.

Syntax:
append (Pfad):(Pfad)...

Beispiele:
append \texte

Beim Öffnen einer Datei wird zuerst im aktuellen Verzeichnis nach ihr gesucht, dann im Verzeichnis »texte«.

append
Nach Eingabe von »append« zeigt MS-DOS in diesem Beispiel:

append= \texte

Mit
append ;
werden die Vereinbarungen wieder aufgelöst.

assign ist ein externer Befehl. Er weist einem Laufwerk eine neue Kennung zu. Die Angabe von Assign ohne Parameter hebt alle Zuweisungen auf und stellt den ursprünglichen Zustand (wie nach einem neuen

Start) wieder her. Die Laufwerks-Bezeichnungen werden entgegen der sonst üblichen Syntax ohne Doppelpunkte angegeben.

Syntax:
assign (x=z) (y=z) ...

Beispiele:

assign a=c

Alle Aufrufe des Laufwerks A werden auf C umgeleitet.

assign a=c b=c

Alle Aufrufe an die Laufwerke A und B werden an C weitergeleitet.

assign
Alle Laufwerke haben wieder ihre ursprüngliche Zuweisung.

Die Befehle »format«, »diskcopy« und »diskcomp« ignorieren Assign-Anweisungen. »backup«, »join«, »label«, »print«, »restore« und »subst« führen in Zusammenhang mit umdefinierten Laufwerken zu Fehlern beziehungsweise zu Programmabstürzen.

attrib ist ein externer Befehl. Er weist Dateien eine neue Statusangabe zu. Um einen gemeinsamen Dateizugriff mehrerer Anwender beispielsweise in einem Netzwerk zu ermöglichen, erzwingt »attrib +r« bei Dateien einen Status, daß sie vom Programm zwar zum Lesen und Schreiben geöffnet werden können, trotzdem aber der Schreibzugriff gesperrt ist. Das bedeutet in der praktischen Anwendung:

Viele Anwenderprogramme prüfen vor einem Dateizugriff, ob Lese- und Schreibzugriff erlaubt sind. Um die Zerstörung einer Datei bei gleichzeitigem Schreibzugriff zweier Benutzer zu vermeiden, simuliert MS-DOS mit »attrib +r« die Erlaubnis für einen Schreibzugriff, obwohl er nicht zulässig ist.

Der Status Archiv findet in Zusammenhang mit »backup«, »restore« und »search« Anwendung.

Syntax:
attrib (+r)(-r)(+a)(-a) (Laufwerk:)(Pfad)Dateiname

»+r« kennzeichnet eine Datei als schreibgeschützt. Sie kann nur gelesen werden. »-r« hebt den Schreibschutz wieder auf. »+a« kennzeichnet eine Datei als Archiv-Datei. »-a« hebt die Kennzeichnung als Archiv-Datei wieder auf.

Beispiele:

attrib +r Daten1

Mit diesem Befehl wird für die Datei »Daten1« der Schreibschutz gesetzt.

attrib Daten1

Ohne Angabe von Attributen wird der aktuelle Status einer Datei angezeigt.

attrib *.*

oder

attrib ?.*

Mit Joker-Angabe (Als Joker beziehungsweise Ersatzzeichen gelten bei MS-DOS »?« und »*«) wird der Status aller Dateien des aktuellen Verzeichnisses angezeigt.

backup ist ein externer Befehl.

»backup« kopiert den Inhalt einer Festplatte auf Diskette oder eine zweite Festplatte zur Datensicherung. Auch für den Übertrag zwischen zwei Disketten kann das Programm benutzt werden. Sieben verschiedene Parameter-Angaben spezifizieren den Backup-Befehl für die »tägliche« Anwendung.

Die Parameter und ihre Funktionen:
»/a« fügt Dateien an das Ende der letzten Sicherungsdatei eines Backup-Vorgangs an. Der Anwender erhält so die Möglichkeit, weitere Dateien an eine vorhandene Sicherungssequenz anzuhängen.

»/d« sichert nur die Dateien, deren Datumsangabe zeitlich nach dem letzten Sicherungsdatum liegt. Das bedeutet (aktuelle Datumsangabe bei

Interne Befehle	Version				
	2.0	2.1	3.0	3.1	3.2
break	x	x	x	x	x
chdir (cd)	x	x	x	x	x
cls	x	x	x	x	x
copy	x	x	x	x	x
ctty	x	x	x	x	x
date	x	x	x	x	x
del	x	x	x	x	x
dir	x	x	x	x	x
echo	x	x	x	x	x
erase	x	x	x	x	x
exit	x	x	x	x	x
mkdir (md)	x	x	x	x	x
path	x	x	x	x	x
prompt	x	x	x	x	x
ren	x	x	x	x	x
rmdir (rd)	x	x	x	x	x
set	x	x	x	x	x
time	x	x	x	x	x
type	x	x	x	x	x
ver	x	x	x	x	x
verify	x	x	x	x	x
vol	x	x	x	x	x
Interne Konfigurations-Befehle	Version				
	2.0	2.1	3.0	3.1	3.2
break	x	x	x	x	x
buffers	x	x	x	x	x
country			x	x	x
device	x	x	x	x	x
fcbs			x	x	x
files	x	x	x	x	x
lastdrive			x	x	x
shell	x	x	x	x	x
stacks					x
Befehle für Batch-Dateien	Version				
	2.0	2.1	3.0	3.1	3.2
echo	x	x	x	x	x
for	x	x	x	x	x
goto	x	x	x	x	x
if	x	x	x	x	x
pause	x	x	x	x	x
rem	x	x	x	x	x
shift	x	x	x	x	x

Externe Befehle	Version				
	2.0	2.1	3.0	3.1	3.2
append					x
assign	x	x	x	x	x
attrib			x	x	x
backup	x	x	x	x	x
chkdsk	x	x	x	x	x
command	x	x	x	x	x
comp	x	x	x	x	x
debug	x	x	x	x	x
diskcomp	x	x	x	x	x
diskcopy	x	x	x	x	x
edlin	x	x	x	x	x
exe2bin	x	x	x	x	x
fc	x	x	x	x	x
fdisk	x	x	x	x	x
find	x	x	x	x	x
format	x	x	x	x	x
grftabl				x	x
graphics	x	x	x	x	x
join				x	x
keybda				x	x
keybfr				x	x
keybgr				x	x
keybsp				x	x
keybsv				x	x
keybuk				x	x
keybit				x	x
keybno				x	x
keybsf				x	x
keybsg				x	x
keybsu				x	x
label				x	x
mode	x	x	x	x	x
more	x	x	x	x	x
print	x	x	x	x	x
recover	x	x	x	x	x
replace					x
restore	x	x	x	x	x
select				x	x
share					x
sort	x	x	x	x	x
subst				x	x
sys	x	x	x	x	x
tree	x	x	x	x	x
xcopy					x

Tabelle 1. Aufteilung nach internen und externen Befehlen

Systemstart vorausgesetzt), nur diejenigen Dateien werden gesichert, die sowohl datums- als auch zeitmäßig nach den bisher gesicherten Dateien liegen.

»d:mm-tt-jj« macht das Gleiche wie »/d«. Als Datum wird jetzt aber nicht das aktuelle, sondern das angegebene genommen.

»/l« bewirkt den Eintrag von Datum, Uhrzeit, Dateinamen und laufender Nummer der Sicherung in die Datei »backup.log«.

»/l:Dateiname« macht das Gleiche wie »/l«, allerdings jetzt unter dem angegebenen Namen.

»/m« sichert die Dateien, die nach dem letzten Sicherungsvorgang verändert oder neu angelegt wurden – bei denen also das Archivbyte gesetzt ist.

»/p« packt die Dateien so eng wie möglich.

»/s« sichert alle Dateien des aktuellen Verzeichnisses und zusätzlich alle Dateien weiter hinten liegender Unterverzeichnisse.

»/t« ist wichtig für Backups während eines Arbeitstages. Die Anweisung sichert nur die Dateien, die nach der letzten Backup-Zeit gesichert wurden.

»/t:hh:mm:ss« macht das Gleiche wie »/t« allerdings mit der angegebenen Zeit.

Syntax:
backup Laufwerk:(Pfad)(Dateiname)
Laufwerk:(/Parameter)

Beispiele:

backup c:*. * a:

Alle Dateien des aktuellen Verzeichnisses von Laufwerk C werden auf Laufwerk A gesichert.

backup c: a:/m/s

Alle Dateien aus allen Verzeichnissen, die nach dem letzten Sicherungsvorgang verändert oder neu angelegt wurden, werden mit der Baumstruktur auf Laufwerk A gesichert.

break ist ein interner Befehl. Er legt fest, ob durch die Tastenkombination [Ctrl-C] eine Programmunterbrechung zugelassen ist.

Syntax:
break (on/off)

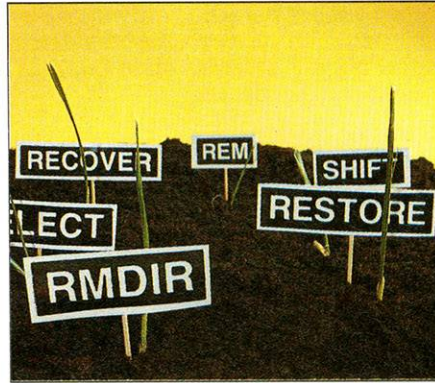
Beispiel:

break on

Eine Unterbrechung des laufenden Programmes ist zugelassen. »break« ohne Parameter zeigt den aktuellen Status der Funktion.

chdir (cd) ist ein interner Befehl. Damit wird die aktuelle Position in einer Baumstruktur geändert.

Syntax:
chdir (..)(Pfad)
oder
cd (..)(Pfad)
Beispiele:
cd ..



ruft das nächsthöhere Verzeichnis auf.

cd \

ruft das Grundverzeichnis auf.

cd texte

ruft das Unterverzeichnis »texte« auf.

chkdsk ist ein externer Befehl. Er überprüft die Dateistrukturen eines Speichermediums. Die Angabe »/f« repariert gefundene Fehler (allerdings nicht defekte Sektoren). »/v« zeigt welche Dateien und Verzeichnisse gerade überprüft werden und welche Fehler sie enthalten.

Syntax:

chkdsk (Laufwerk:)(Dateiname) (/f)(/v)

Beispiel:

chkdsk c: /f/v

Das Laufwerk C wird auf Fehler untersucht, Fehler werden beseitigt, die überprüften Dateien und Verzeichnisse werden angezeigt.

Mit

chkdsk a: > fehler

werden alle Angaben in die Datei »fehler« geschrieben.

cls ist ein interner Befehl. Er löscht den Bildschirm und positioniert den Cursor in die linke obere Ecke des Bildschirms. Die Bildschirmattribute werden nicht verändert.

Syntax:

cls

command ist ein externer Befehl. Damit wird die Kommandodatei erneut geladen und gestartet. Sie können damit in einer höheren Ebene mit »command.com« arbeiten. Änderungen des Pfads und so weiter können hier verändert werden, ohne daß sie sich in der aufrufenden Ebene verändern. Mit »exit« wird die Ebene verlassen.

Syntax:

command (Laufwerk:)(Pfad) (ctty Ein-/Ausgabegeräte)(/Parameter)

Als Parameter können Sie folgende Werte angeben:

»/c« definiert eine Zeichenkette mit maximal 127 Zeichen, die von »command.com« als MS-DOS-Befehl ausgeführt wird.

»/e:Zahl« legt die Größe des Environment-Bereichs fest. Ohne Angabe liegt dieser bei 128.

»/p« überschreibt die Kommandoda-

tei im Speicher. Damit bleibt die Ebene mit neuen voreingestellten Parametern erhalten und kann nicht mit »exit« verlassen werden.

Beispiel:

command /c copy a:*. * b:

Damit wird eine neue Kommandoebene aufgerufen und der Copy-Befehl ausgeführt.

comp ist ein externer Befehl. Er vergleicht die Inhalte zweier Dateien miteinander.

Syntax:

comp (Laufwerk:)(Pfad)Dateiname (Laufwerk:)(Pfad)Dateiname (/Parameter)

Als Parameter sind erlaubt:

»/a« zeigt den Unterschied, die Zeile davor und danach.

»/b« sorgt für zeichenweises Vergleichen.

»/c« alle Buchstaben werden als Großbuchstaben verglichen.

»/l« sorgt für zeilenweisen Vergleich.

»/lZahl« macht das Gleiche wie »/l«. Die Größe des Zwischenspeichers wird nun aber durch Zahl in der Zahl der Zeilen definiert.

»/n« zeigt zusätzlich die Zeilennummer an.

»/t« sorgt dafür, daß Tabulatorzeichen nicht als Leerräume gewertet werden.

»/w« wertet Tabulatorzeichen oder Leerräume mit mehr als einem Zeichen als ein Zeichen.

»/#« legt die Zahl der Zeilen fest, nach denen die Ausgabe von Unterschieden in den beiden Dateien neu erfolgt.

Beispiel:

comp text1 text2

Der Inhalt von »text1« wird mit »text2« verglichen, Unterschiede werden am Bildschirm angezeigt.

copy ist ein interner Befehl. Er dient zum Kopieren einer oder mehrerer Dateien.

Die Parameter und ihre Funktionen:

»/a« bei der Quelldatei: Die Datei wird als ASCII-Datei behandelt und bis zum Auftreten des ersten Dateiende-Zeichens kopiert.

»/a« bei der Zieldatei: Dateiende-Zeichen wird angefügt.

»/b« bei der Quelldatei: Die gesamte Datei wird nach der Größenangabe des Inhaltsverzeichnisses kopiert.

»/b« bei der Zieldatei: Verhindert das Anfügen eines Dateiende-Zeichens.

»/v« hat dieselbe Wirkung wie der Befehl »verify = on«.

Syntax:

copy (Laufwerk:)(Pfad)Dateiname(/a)(/b) (Laufwerk:)(Pfad)Zieldatei(/a)(/b)(/v)

Beispiele:

copy date1 date2

»date1« wird in »date2« kopiert.

copy a:*. * c:/v

Der gesamte Inhalt der Diskette in Laufwerk A wird auf Laufwerk C kopiert und die Richtigkeit mit Verify überprüft.

ctty ist ein interner Befehl. Er definiert den Gerätetreiber. »ctty« steuert normalerweise den Bildschirm und die Tastatur an.

Syntax:
ctty Gerätename

Beispiel:
ctty = prn

Der Ausgabekanal wird zum Drucker geleitet. Die erlaubten Schnittstellen finden Sie in Tabelle 2.

date ist ein interner Befehl. Mit ihm wird das Datum eingestellt. »date« ohne Parameter zeigt das aktuelle Systemdatum.

Syntax:
date (mm-tt-jj)

Beispiel:
date 09-21-87

Das Datum wird auf den 21. September 1987 eingestellt. Bei manchen MS-DOS-Versionen kann der »-« durch einen ».« ersetzt werden.

debug ist ein externer Befehl. Er ruft das Programm »debug.com« auf. Da der Debugger sehr umfangreich ist, befaßt sich mit ihm ein eigener Artikel.

del ist ein interner Befehl. Er löscht die angegebenen Dateien.

Syntax:
del (Laufwerk:)(Pfad)Dateiname.erg

Beispiele:
del dateil
»dateil« wird gelöscht.

del *.*
Alle Dateien im aktuellen Verzeichnis werden gelöscht.

dir ist ein interner Befehl. Er zeigt den Inhalt von Verzeichnissen.

Die Parameter und ihre Funktionen:
»p« steuert das (Bildschirm-)seitenweise Ausgeben der Verzeichnisse.
»w« zeigt die Dateinamen im weiten Format, das heißt, es werden jeweils fünf Dateinamen in eine Zeile geschrieben.

Syntax:
dir (Laufwerk:)(Pfad)(Dateiname)(/p)(/w)

Beispiele:
dir dateil.*

Alle Dateien im aktuellen Verzeichnis, die als Namen »Datei« haben, werden angezeigt.

dir
Der Inhalt des aktuellen Verzeichnisses wird angezeigt.

dir /p
Der Inhalt des aktuellen Verzeichnisses wird seitenweise angezeigt.

diskcomp ist ein externer Befehl. Er vergleicht den Inhalt zweier Disketten.

Die Parameter und ihre Funktion:
»/1« bewirkt, daß nur eine Seite (Seite 0) der Disketten verglichen wird.
»/8« bewirkt, daß nur acht Sektoren einer Spur verglichen werden.

Syntax:
diskcomp (Laufwerk:) (Laufwerk:)
(/1)(/8)

Beispiel:
diskcomp a:b:/8

Der Computer vergleicht den Inhalt der Diskette in Laufwerk A mit der Diskette in Laufwerk B, wobei jeweils nur acht Sektoren pro Spur verglichen werden.

diskcopy ist ein externer Befehl. Er dient zum Kopieren von Disketten, wobei die Quelldiskette das Aufzeichnungsformat der Zieldiskette bestimmt.

Die Parameter und ihre Funktion:
»/f« steuert eine Neuformatierung der Zieldiskette.

»/k« bewirkt den Beginn des Kopiervorganges ohne weitere Abfragen.
»/r« steuert die Rückkehr in das Betriebssystem, wenn keine Kopierfehler aufgetreten sind.

»/o« erlaubt das Kopieren auf einem Computer mit Festplatte, wobei zwischenzeitlich eine Datei auf der Festplatte eingerichtet wird, sofern der Arbeitsspeicher nicht ausreicht.
»/s« zeigt den Programmnamen, Versionsnummer und Entstehungsdatum.

»/v« steuert die Ausgabe von Bildschirmmeldungen über den Kopiervorgang.

»/1« kopiert nur die erste Seite einer Diskette.

»/?« zeigt einen Hilfstext an, der Befehl wird nicht ausgeführt.

Syntax:
diskcopy (Laufwerk:) (Laufwerk:) (Parameter)

Beispiel:

diskcopy /?
Hilfstexte über »diskcopy« werden angezeigt.

diskcopy /t/v

Die Zieldiskette wird vor dem Kopieren formatiert. Während des Kopierens werden die Arbeitsvorgänge auf dem Bildschirm angezeigt.

echo ist ein interner Befehl. Er steuert die Anzeige von Stapelbefehlen. Mit »on« wird die Anzeige an und mit »off« ausgeschaltet.

Syntax:
echo (on/off)

edlin ist ein externer Befehl. »edlin.com« ist ein Editor zum Schreiben und Ändern von Textdateien. Da das Programm sehr umfangreich ist, finden Sie es in einem eigenen Artikel in diesem Heft erklärt.

erase ist ein interner Befehl. Er hat die gleiche Funktion wie »del«

exe2bin ist ein externes Dienstprogramm. Es wandelt Exe-Dateien in Binär-Dateien um.

Syntax:
exe2bin (Pfad)Dateiname.EXE

exit ist ein interner Befehl. Er steuert das Verlassen des aktuellen Befehlsprozessors und die Rückkehr in den vorhergehenden Befehlsprozessor.

Syntax:
exit

fc ist ein externer Befehl. Er vergleicht den Inhalt zweier Dateien miteinander.

Die Parameter und ihre Bedeutung:
»/a« gibt die Unterschiede in ASCII-Zeichen aus.

»/c« hebt Unterschiede der Groß- und Kleinschreibung auf.

»/l« steuert den Vergleich im ASCII-Modus.

»/lb n« setzt den internen Zeilenpuffer auf »n«.

»/n« gibt die Anzahl der Zeilen nach einem ASCII-Vergleich aus.

»/t« verhindert das Umwandeln von Tabulatoren in Leerzeichen.

»/w« komprimiert Tabulator und Leerzeichen auf eine Leerstelle.

»/nnnn« beschreibt die Anzahl der Zeilen, die unterschiedlich sein dürfen, bis »fc« den Vergleich abbricht.

Syntax:
fc (Parameter) Dateiname1
Dateiname2

name	Schnittstelle
aux	asynchrone Schnittstelle
clock\$	Treiber für die Uhr
com1	erste serielle Schnittstelle
com2	zweite serielle Schnittstelle
con	Console (Bildschirm und Tastatur)
lpt1	erste parallele Schnittstelle
lpt2	zweite parallele Schnittstelle
lpt3	dritte parallele Schnittstelle
nul	nicht vorhandener Ausgang
prn	Drucker

Tabelle 2. Die Schnittstellen von MS-DOS

n	Wirkung
40	Bildschirmbreite = 40 Zeichen
80	Bildschirmbreite = 80 Zeichen
BW40	Schwarz/Weiß, 40 Zeichen Breite
BW80	Schwarz/Weiß, 80 Zeichen Breite
CO40	Farbe, 40 Zeichen Breite
CO80	Farbe, 80 Zeichen Breite
MONO	monochrom, 80 Zeichen Breite

Tabelle 3. Sieben verschiedene Bildschirmmodi gibt es

fdisk ist ein externer Befehl. »fdisk« bereitet eine Festplatte als Speichermedium vor. Mit der Routine können Sie Ihre Festplatte in bis zu vier verschiedene Bereiche aufteilen, die dann als getrennte Laufwerke angesprochen werden. Jedem Bereich kann ein eigenes Betriebssystem zugeordnet werden. »fdisk« arbeitet menügesteuert. Die Syntax ist damit sehr einfach.

Syntax:
fdisk

find ist ein externer Befehl. Er durchsucht Dateien nach einer vom Benutzer definierten Zeichenfolge.

Die Parameter und ihre Funktion:

»/c« zeigt die Anzahl der Übereinstimmungen an.

»/n« gibt die relative Zeilennummer einer Zeile mit Übereinstimmung an.

»/v« zeigt die Zeilen, die keine Übereinstimmung enthalten.

Syntax:

find (Parameter) "Zeichenfolge"
(pfad)Dateiname.erg

Beispiel:

find /n "München" datei1

»datei1« wird nach dem Begriff »München« durchsucht und die Zeilen mit Übereinstimmung werden mit relativer Zeilennummer angezeigt.

for ist ein interner Befehl. Damit wird ein MS-DOS-Befehl mehrmals ausgeführt. Damit ist die Anweisung besonders für Stapeldateien interessant.

Syntax:

for %(%Variable in (Liste) do Befehl
Der Variablen werden die Werte aus der Liste zugeordnet. Dann wird der Befehl ausgeführt.

Beispiel:

for %z in (datei1.txt datei2.txt
datei3.txt) do ren %z *.sik

Mit dieser Zeile bekommen die Dateien »datei1.txt«, »datei2.txt« und »datei3.txt« Namen mit der Erweiterung »sik«.

format ist ein externes Dienstprogramm. Es dient der Vorbereitung eines Datenträgers zur Speicherung von MS-DOS-Dateien. »format« richtet ein Grundverzeichnis ein und legt die einzelnen Sektoren des Datenträgers fest.

Die Parameter und ihre Funktion:

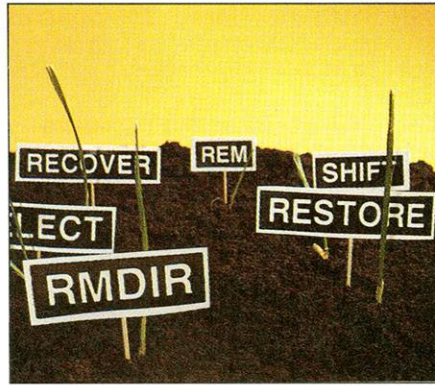
»/b« reserviert auf der Diskette ausreichenden Speicherplatz für die System-Dateien.

»/s« überträgt nach dem Formatieren das Betriebssystem MS-DOS auf den Datenträger.

»/v« schreibt nach dem Formatieren eine Kennung auf die Diskette.

»/1« erzeugt auf einem doppelseitigen Laufwerk eine einseitig formatierte Diskette.

»/4« formatiert eine doppelseitige Diskette mit 360 KByte auf einem Laufwerk mit 1,2 MByte Speicherkapazität.



»/8« formatiert eine Diskette mit 8 Sektoren pro Spur (Standard: 9)

Syntax:

format Laufwerk: (Parameter)

Beispiel:

format a: /v/s

Die Diskette im Laufwerk A wird formatiert. Danach wird eine Kennung und das System übertragen. Die Diskette bekommt eine Kennung.

goto ist ein interner Befehl. »goto« führt einen Sprungbefehl in einer Stapeldatei aus. Die Sprungadresse wird durch ein Label gekennzeichnet.

Syntax:

goto marke

Syntax für das Label:

:marke

Beispiel:

:xy

goto xy

Vorsicht beim Ausprobieren dieses Beispiels. Der Computer gelangt in eine Endlosschleife, die allerdings mit [Ctrl-C] abgebrochen werden kann.

grftabl ist ein externer Befehl. Er lädt eine zusätzliche Zeichentabelle für den Farbgrafik-Adapter in den Arbeitsspeicher. »grftabl« darf nur einmal, und zwar nach dem Laden des Betriebssystems, aufgerufen werden.

Syntax:

grftabl

graphics ist ein externer Befehl. Er gibt den Inhalt des Bildschirms auf einem Matrixdrucker aus.

Die Parameter und ihre Funktion:

»/b« druckt auch die Hintergrundfarbe.

»/r« gibt den Inhalt des Bildschirms invers aus.

Syntax:

graphics (/r)(/b)

if ist ein interner Befehl. Der bei »if« stehende Befehl wird nur ausgeführt, wenn die Bedingung erfüllt ist.

Syntax:

if (not) Bedingung Befehl

Als Bedingung können Sie unter drei Varianten wählen. »exist Dateiname« fragt nach dem Vorhandensein einer Datei. »Zeichenkette1 =

Zeichenkette2« vergleicht zwei Zeichenketten miteinander. Die wichtigste Bedingung lautet »errorlevel

Nummer«. Die Fehler auf die Ihr Computer stoßen kann, tragen nämlich alle eine Kennnummer. Diese Nummer wird mit »errorlevel« zurückgegeben.

Beispiel:

if errorlevel 0 echo Der Fehlercode ist 0.

Stößt der Computer in einer Stapeldatei auf diese Anweisung, so erfolgt die Klartextmeldung.

join ist ein externer Befehl. Er verbindet ein Laufwerk mit einem Verzeichnis auf einem anderen Laufwerk. Es wird aus den voneinander unabhängigen Verzeichnissen eine einzige Verzeichnisstruktur erzeugt. »join« ohne Parameter zeigt die bestehenden Verknüpfungen, der Parameter »/d« löst die bestehende Verknüpfung für das angegebene Laufwerk auf.

Syntax:

join (Laufwerk1:) (Laufwerk2:Pfad)

oder

join (Laufwerk1:) (/d)

keybxx ist ein externer Befehl. Er lädt eine länderspezifische Tastaturanpassung. Außer der deutschen Tastaturanpassung (»keybgr«) stehen folgende Anpassungen zur Verfügung:

keybda.com	= Dänemark
keybfr.com	= Frankreich
keybit.com	= Italien
keybno.com	= Norwegen
keybsf.com	= französische Schweiz
keybsg.com	= deutsche Schweiz
keybsp.com	= Spanien/Mexico
keybsu.com	= Finnland
keybsv.com	= Schweden
keybuk.com	= England

Syntax:

keybxx

»xx« steht für die beiden länderspezifischen Buchstaben.

label ist ein externer Befehl. Er stellt eine Kennung auf einer Diskette oder Festplatte her, er verändert oder löscht sie. Wird »label« ohne Kennung eingegeben, zeigt der Befehl die aktuelle Kennung und fordert zur Neueingabe auf. Soll die Kennung gelöscht werden, gibt man »label« ohne Kennung ein und drückt direkt nach der Aufforderung zur Neueingabe [Return].

Syntax:

label (Laufwerk:) (Kennung)

Beispiel:

label c: platte1

Dem Laufwerk C wird die Kennung »platte1« zugewiesen.

mkdir (md) ist ein interner Befehl, mit dem Unterverzeichnisse angelegt werden.

Syntax:

mkdir (Pfad)

Beispiel:

mkdir dos

Im Verzeichnis, aus dem »mkdir« aufgerufen wurde, wird das Unterverzeichnis »dos« angelegt.

mode ist ein externer Befehl. Er legt bestimmte Arbeitsparameter für den Bildschirm, den Drucker und die seriellen Schnittstellen fest. Für diesen Befehl gibt es vier verschiedene Eingabeformen.

mode (n), (m),(t))

bestimmt die Parameter für den Bildschirm. Die Werte für »n« entnehmen Sie Tabelle 3.

Der Parameter »m« verschiebt die Bildschirmanzeige um ein/zwei Zeichen (40/80 Zeichen Breite) nach links oder rechts. Je nach Richtung in der verschoben werden soll, muß »r« oder »l« eingegeben werden.

Wird »t« in Verbindung mit »m« benutzt, erwartet MS-DOS vom Anwender eine Bestätigung des Befehls.

mode lptnn:(n)(m)(p)

bestimmt die Parameter für den Drucker.

Die Parameter und ihre Funktion:

nn – Nummer des Druckers, der eingestellt werden soll (1, 2 oder 3).

n – Zeilenbreite des Druckers (80 oder 136)

m – Zeilenvorschub (6 oder 8 Zeichen pro Zoll)

p – Bei Fehlern wird mehrfach erneut versucht, den Vorgang zu wiederholen.

mode com n:Baudrate (,Paritätsbit(,Datenbits(,Stopbits(p)))

Bestimmt die Parameter für eine serielle Schnittstelle.

Die Parameter und ihre Funktion:

n: Nummer der Schnittstelle, die eingestellt werden soll (1 oder 2).

Baudrate: Legt die Baudrate fest (es müssen nur die ersten beiden Ziffern angegeben werden, zum Beispiel 30 für 300 Baud).

Parität: Legt die Parität fest (»e« für gerade, »o« für ungerade und »n« für keine).

Datenbits: Anzahl der Datenbits (7 oder 8).

Stopbits: Anzahl der Stopbits (1 oder 2).

p: Bei Fehlern wird mehrfach erneut versucht, den Vorgang zu wiederholen.

mode lpt nn:=com n

Die Ausgabe von der parallelen Schnittstelle wird auf eine serielle Schnittstelle umgeleitet.

»nn« gibt die Nummer der parallelen Schnittstelle an, »n« die Nummer der seriellen Schnittstelle.

more ist ein externer Befehl. Mit ihm werden Inhalte von Text-Dateien zeilenweise am Bildschirm angezeigt. Um jeweils auf die nächste Seite zu kommen, muß die Leer- oder Return-Taste gedrückt werden.

Syntax:

more

Beispiel:

type datei.txt | more

»datei.txt« wird jetzt zeilenweise auf den Bildschirm ausgegeben.

park ist ein externer Befehl. Er positioniert die Schreib-/Leseköpfe der Festplatte in einen speziellen Plattenbereich, was eine Beschädigung der Festplatte durch Vibration (beispielsweise beim Transport) verhindert.

Syntax:

park

path ist ein interner Befehl. Er gibt dem Computer einen festen Zugriffspfad vor.

Die Eingabe von »path« alleine zeigt den festgelegten Pfad.

Syntax:

path (Laufwerk:)(Pfad)(:Pfad)...

Beispiel:

path c:\prog\test

Damit sind die Programme im Unterverzeichnis »prog« und »test« von überall aus aufrufbar.

pause ist ein interner Befehl. Er veranlaßt eine Unterbrechung bei der Bearbeitung einer Stapeldatei. Am Bildschirm wird die Meldung »Strike any key when ready ...« (Drücken Sie irgendeine Taste, wenn bereit) ausgegeben. Der Benutzer kann einen bis zu 121 Zeichen langen Kommentar, der dann vor der System-Meldung ausgegeben wird, definieren.

Syntax:

pause (Kommentar)

Beispiel:

pause Wechseln Sie die Diskette in Laufwerk B:

Stößt MS-DOS beim Bearbeiten einer Stapel-Datei auf diesen Befehl, wird der Ablauf unterbrochen und am Bildschirm folgende Meldung ausgegeben:

Wechseln Sie die Diskette in Laufwerk B:

Strike any key when ready ...

print ist ein externer Befehl. Er druckt Dateien im Hintergrund, während das Betriebssystem für weitere Aufgaben frei ist. Sollen mehrere Dateien gleichzeitig gedruckt werden, stellt »print« die Dateien in eine Warteschlange.

Die Parameter und ihre Funktionen: »/b:n1« setzt die interne Größe auf »n1« Bytes. Die Voreinstellung beträgt 512 Byte.

»/c« löscht die benannten Dateien aus der Warteschlange.

»/d:Gerätename« spezifiziert das Ausgabegerät. Wird beim ersten Aufruf des Befehles kein Ausgabegerät durch »/d« angegeben, fragt das Betriebssystem gesondert nach dieser Angabe.

»/m:n3« legt die Prozessorzeit fest, die »print« in Anspruch nehmen darf.

»/q:n5« bestimmt, wieviele Dateien in der Warteschlange stehen dürfen. Der Maximalwert beträgt 32.

»/p« druckt die benannten Dateien.

»/s:n4« legt die Prozessorzeiten fest, die das Betriebssystem für »print« reservieren soll.

»/t« beendet den Druckvorgang und löscht die Dateien aus der Warteschlange.

»/u:n2« bestimmt den Zeitraum, den die Ausgaberroutine abwartet, wenn der Drucker nicht bereit ist. Der Zeitraum beträgt »n2« Zeiteinheiten. Ist die Wartezeit verstrichen, gibt

»print« die zugeeilte Prozessorzeit an MS-DOS zurück.

Syntax:

print (/d:Gerätename) (/b:n1) (/u:n2) (/m:n3) (/s:n4) (/q:n5) (/c) (/p) (/t) (Laufwerk:)(Pfad)Dateiname(, ...)

prompt ist ein interner Befehl, der die Betriebssystemkennung den eigenen Bedürfnissen anpaßt. »prompt« ohne Parameter stellt den normalen Prompt wieder ein.

Syntax:

prompt (Text)

Beispiel:

prompt Mach hin!

Anstelle des »A>« kommt jetzt

»Mach hin!« als Systemkennung.

Da das System bestimmte ASCII-Zeichen schon selbst benutzt, muß in solch einem Fall eine Fluchtsequenz eingefügt werden. Sämtliche Fluchtsequenzen auf einen Blick finden Sie in Tabelle 4.

Beispiel:

prompt \$n\$p

Der Prompt besteht jetzt aus dem aktuellen Laufwerk und dem aktuellen Verzeichnis.

recover ist ein externer Befehl. Er wird eingesetzt, wenn das Betriebssystem aufgrund eines Fehlers eine Datei nicht lesen kann. Er kennzeichnet fehlerhafte Sektoren und stellt die Verkettung der anderen Sektoren wieder her. Die Daten aus fehlerhaften Sektoren gehen verloren.

Syntax:

recover Laufwerk:

rem ist ein interner Befehl. Er dient der Kennzeichnung von Kommentarseiten in Stapel-Dateien.

Syntax:

rem Kommentar

ren ist ein interner Befehl. Er dient der Umbenennung von Dateien.

Syntax:

ren (Pfad)Dateiname.erg Dateiname.erg

Beispiel:

ren test1 test1.1

Die Datei »test1« wird in »test1.1« umbenannt.

replace ist ein externer Befehl. Er ersetzt mehrfach vorhandene Dateien durch die jeweils neueste Version. Besonders bei der Arbeit mit einer Festplatte kann so die Software leicht auf dem aktuellsten Zustand gehalten werden. Nicht vorhandene Dateien werden hinzugefügt.

Die Parameter und ihre Funktion:

»/a« addiert die entsprechenden Dateien zum Zielverzeichnis, wenn Unterschiede festgestellt werden. Es werden keine Dateien überschrieben.

»/d« ersetzt die Dateien im Zielverzeichnis, die älter sind als die Dateien im Quellverzeichnis.

»/p« steuert eine »Ersetzen

Ja/Nein«-Abfrage vor dem Kopiervorgang.

»/r« ersetzt auch schreibgeschützte Dateien (Read only).

»/s« veranlaßt den Replace-Befehl, daß auch alle Unterverzeichnisse nach Zieldateien durchsucht werden.

»/w« veranlaßt den Computer vor jedem Kopiervorgang auf einen Tastendruck zu warten.

Syntax:

```
replace (Laufwerk:)(Pfad)Dateiname
(Laufwerk:)(Pfad) (/a)(/d)(/p)(/r)(/s)(/w)
```

Beispiel:

```
replace telnr /s
```

Die Dateien »telnr« werden in allen Verzeichnissen durch die aktuelle Version von »telnr« ersetzt.

restore ist ein externer Befehl. Er kopiert Dateien, die mit »backup« erstellt wurden.

Die Parameter und ihre Funktion:

»/a:mm-tt-jj« überschreibt nur die Dateien, die mit dem angegebenen Datum übereinstimmen oder später angelegt wurden.

»/b:mm-tt-jj« überschreibt nur Dateien, die vor dem angegebenen Datum angelegt wurden.

»/e:hh:mm:ss« überschreibt nur Dateien, die vor der angegebenen Uhrzeit angelegt wurden.

»/l:mm-tt-jj« überschreibt nur Dateien, die mit der Uhrzeit übereinstimmen oder hinterher angelegt wurden.

»/m« kopiert nur die Dateien auf die Festplatte, die seit der letzten Sicherung verändert wurden.

»/n« kopiert nur die Dateien, die auf der Festplatte nicht existieren.

»/p« steuert eine Abfrage, ob auch schreibgeschützte Dateien und Dateien, die nach der letzten Sicherung verändert wurden, überschrieben werden sollen.

»/s« kopiert alle Dateien in allen Unterverzeichnissen zurück.

Syntax:

```
restore Laufwerk: (Laufwerk:)(Pfad)Da-
teiname (/s)(/p)(/a:mm-tt-jj)(/b:mm-tt-jj-
)(/e:hh:mm:ss)(/l:hh:mm:ss)(/m)(/n)
```

rmdir (rm) ist ein interner Befehl mit dem ein Unterverzeichnis gelöscht wird.

Syntax:

```
rmdir (Pfad)
```

select ist ein externer Befehl. Er paßt eine neue Systemdiskette an die Tastaturbelegung und das Format von Datum und Uhrzeit an länderspezifische Gegebenheiten an.

Syntax:

```
select Landcode Tastaturcode
```

set ist ein interner Befehl. Er weist einer Variablen eine beliebige Zeichenkette zu.

Syntax:

```
set (Variable = (Zeichenkette))
```

share ist ein externer Befehl. Dieser

Code	Bedeutung
\$b	Pipesymbol
\$d	Systemdatum (Tag, Monat, Jahr)
\$e	Escapezeichen (27dez)
\$g	>
\$h	Löscht vorangegangenes Zeichen
\$l	<
\$n	Aktuelles Laufwerk
\$p	Aktuelles Unterverzeichnis (Pfad) des aktuellen Laufwerks
\$q	=
\$t	Systemzeit (Stunde, Minute, Sekunde, hundertstel Sekunde)
\$v	Versionsnummer von MS-DOS (PC-DOS)
\$_	Wagenrücklauf und Zeilenvorschub
\$	\$

Tabelle 4: Die Fluchtsequenzen des Prompt-Befehls

Befehl ist eine Grundlage für den Netzwerkbetrieb.

Die Parameter und ihre Funktion:

»/f:bytes« reserviert den mit »bytes« angegebenen Speicherplatz für die Informationen, die für einen gemeinsamen Zugriff benötigt werden.

»/l:nns« bestimmt die Anzahl (»nns«) von Sperren, die für den Multiuser-Betrieb nötig sind.

Syntax:

```
share (/f:bytes)(/l:nns)
```

shift ist ein interner Befehl für Stapeldateien. Damit werden in einer Datei Eingaben der Reihe nach gelesen.

»shift« holt immer den nächsten Befehl aus der eingegebenen Zeile.

Syntax:

```
shift
```

sort ist ein externer Befehl. Mit diesem Befehl werden die Daten einer Datei alphanumerisch (ASCII-Tabelle) sortiert.

Die Parameter und ihre Funktion:

»Befehl| « bestimmt einen externen Befehl, dessen Ausgabe sortiert werden soll.

»/r« bestimmt, daß absteigend (rückwärts) sortiert wird.

»/ + n« bestimmt, daß ab der Spalte » + n« sortiert wird.

»<« alle diesem Symbol folgenden Parameter dienen als Eingabe-Daten.

»>« alle diesem Symbol folgenden Parameter dienen zur Ausgabe.

Syntax:

```
(Befehl| ) sort [/r]/ + n [[Laufwerk:]Eingabe] [[Laufwerk:]Ausgabe]
```

subst ist ein externer Befehl. Er ordnet einem Laufwerk oder Pfad eine neue Laufwerks-Bezeichnung zu. Mit dem Parameter »/d« wird die Zuweisung widerrufen.

Syntax:

```
subst (Laufwerk 1:) (Laufwerk 2:)(Pfad))
```

sys ist ein externes Dienstprogramm. Mit ihm wird das Betriebssystem auf eine Diskette oder Festplatte gespielt.

Syntax:

```
sys Laufwerk:
```

time ist ein interner Befehl. Mit ihm wird die Uhrzeit eingestellt.

Syntax:

```
time (hh)(.mm)(.ss)
```

»hh« bezeichnet die Stunde, »mm« die Minute und »ss« die Sekunden.

tree ist ein externer Befehl. Er zeigt alle bestehenden Pfade eines Laufwerks an. Mit dem Parameter »/f« werden zusätzlich alle Dateinamen der Unterverzeichnisse gezeigt.

Syntax:

```
tree (Laufwerk:)(/f)
```

type ist ein interner Befehl. Er zeigt den Inhalt von Textdateien an.

Syntax:

```
type (Pfad)Dateiname
```

ver ist ein interner Befehl. Er zeigt die Versionsnummer des Betriebssystems, mit dem Sie zur Zeit arbeiten.

Syntax:

```
ver
```

verify ist ein interner Befehl. Er stellt sicher, daß beim Schreiben auf Diskette oder Festplatte keine Fehler aufgetreten sind. »verify« ohne Parameter zeigt den aktuellen Zustand an.

Syntax:

```
verify (on/off)
```

vol ist ein interner Befehl. Er zeigt die Datenträgerkennung. Mit dem Parameter »/c« kann anschließend die Kennung verändert werden.

Syntax:

```
vol (Laufwerk:)(/c)
```

xcopy ist ein externes Dienstprogramm. Es kopiert selektiv Verzeichnisse und Dateien.

Die Parameter und ihre Funktion:

»/a« kopiert ausschließlich Dateien, die den Status »Archiv« haben.

»/d:mm-tt-jj« kopiert nur die Dateien, die mit dem Datum übereinstimmen oder älter sind.

»/e« kopiert jedes Unterverzeichnis.

»/m« funktioniert wie »/a«, jedoch wird der Status der Quelldatei zurückgesetzt.

»/p« steuert jeweils vor einem Kopiervorgang die Abfrage, ob eine Datei tatsächlich kopiert werden soll.

»/s« kopiert die Unterverzeichnisse eines aktuellen Verzeichnisses.

»/v« schaltet Verify für den Kopiervorgang ein.

»/w« veranlaßt das Dienstprogramm, vor dem Starten von »xcopy« auf Tastendruck zu warten.

Syntax:

```
xcopy (Laufwerk:)(Pfad)Dateiname
(Laufwerk:)(Dateiname
(/a)(d:mm-tt-jj)(/e)(/m)(/p)(/s)(/v)(/w)
```

(Christian Rogge/hg)

WIE HÄTTEN SIE'S GERN?



Informativ – lehrreich – interessant

Zeitschriften aus dem Hause DMV

Jeden Monat bei Ihrem Zeitschriftenhändler
oder direkt beim Verlag.



DMV Daten & Medien Verlagsgesellschaft mbH
Postfach 250 · Fuldaer Str. 6
3440 Eschwege · Tel. (05651) 87 02

Edlin, der zeilenorientierte Editor unter MS-DOS

Edlin ist, wie der Name schon sagt, ein zeilenorientierter Editor. Der externe MS-DOS-Befehl kann deshalb nicht als Textverarbeitungs-Programm bezeichnet werden. Edlin ist vielmehr ein Dienstprogramm, das beim Erzeugen und Editieren von ASCII-Dateien von großem Nutzen ist.

Das Einsatzgebiet von Edlin ist weit gestreut. Das Programm kann in Verbindung mit jeder ASCII-Textdatei eingesetzt werden, deren Zeilenlänge nicht länger als 253 Zeichen ist. Meist wird Edlin zur Bearbeitung von Stapelverarbeitungs- oder System-Dateien wie »config.sys« oder »autoexec.bat« eingesetzt.

Die Länge des zu bearbeitenden Textes ist auf 65529 Zeilen begrenzt. Eine weitere Einschränkung liegt bei der Speichergrenze von 64 KByte. Edlin ist nämlich nicht in der Lage, mehr als 65535 Byte zu bearbeiten. Mit dieser externen DOS-Funktion lassen sich Texte schreiben, speichern und überarbeiten. Sie eignet sich zum Löschen und Einfügen ganzer Zeilen. Es ist sogar eine Such- und Ersetzfunktion integriert.

In der Bedienung ist Edlin gewöhnungsbedürftig

Der zeilenorientierte Editor ist ein externer MS-DOS-Befehl und muß deshalb beim Aufruf auf der Diskette im aktuellen Laufwerk stehen. Der Syntax dieser Anweisung lautet:

edlin Laufwerk:(Pfad)Dateiname(/x)
Soll zum Beispiel im Laufwerk A die Datei »autoexec.bat« editiert werden, so genügt die Eingabe von

edlin a:autoexec.bat

Der Anhang »/x« (oder »/b«) muß folgen, falls die Textdatei den Ctrl-Z-Code enthält. [Ctrl-Z] ist normalerweise die Kennung für das Dateende (»eof«).

Der Befehlssatz von »edlin« ist einfach und kurz. Die Befehle selbst bestehen lediglich aus einem Buchstaben und einigen Zeilenangaben (Tabelle 1). Zur Verdeutlichung dieser Kurzbeschreibung laden Sie eine ASCII-Datei. Am einfachsten erzeugen

Sie diese mit dem Copy-Befehl. Tippen Sie bitte folgende Zeilen ein:

```
A> copy com:demo.asc
Dies ist Zeile 1
Zeile 2 folgt sogleich
Mit Zeile 3 bist Du dabei
↑Z
```

»↑Z« steht für [Ctrl-Z] und wird mit der F6-Taste eingegeben. [Return] beendet die Datei. Es wird die Meldung »1 Datei(en) kopiert« ausgege-

ben und die Demo-Datei steht auf Ihrer Diskette. Mit diesem ASCII-Text können Sie nun experimentieren. Da Edlin, wie gesagt, ein externer MS-DOS-Befehl ist, muß die Datei auf die aktuelle Diskette kopiert werden. Um den Demo-Text zu bearbeiten, geben Sie folgenden Befehl ein.

```
A> edlin demo.asc
```

Danach erscheint die Meldung »Ende der Eingabedatei« und ein Stern in der nächsten Zeile auf dem Bildschirm. Dieser Stern markiert den Anfang einer Zeile und erlaubt dem Anwender Befehle einzutippen. Geben Sie nun »l [Return]« ein, um den gesamten Text zu listen:

```
*1
1: *Dies ist Zeile 1
2: Zeile 2 folgt sogleich
3: Mit Zeile 3 bist Du dabei
*
```

Die Zeilennummern vergibt Edlin automatisch. Beim Kopieren, Löschen oder Verschieben passen sich die Zeilennummern selbstständig an.

Wie aus Tabelle 1 ersichtlich ist, kennt Edlin sowohl einen Kopier- als auch Verschiebe-Befehl. Die Kopieranweisung hat folgenden Syntax:

Befehl	Syntax	Bedeutung
A	(Zeile) a	Zeilen anfügen
C	(Anf),(End),Ziel,(Anzahl) c	Zeile(n) kopieren
D	(Anf),(End) d	Zeile(n) löschen
E	e	Text speichern und Ende.
I	(Zeile) i	Zeile(n) einfügen
L	(Anf),(End) l	Zeilen auflisten
M	(Anf),(End),Ziel m	Zeilen verschieben
P	(Anf),(End) p	Seitenweise auflisten
Q	q	Ende.
R	(Anf),(End)(?) rStr1 F6 Str2	String suchen und ersetzen
S	(Anf),(End)(?) sString	String suchen
T	(Zeile) tDateiname	ASCII-Datei importieren
W	(Anzahl) w	Zeilen speichern

Tabelle 1. Alle Edlin-Befehle auf einen Blick

Taste	Beschreibung
F1/->	Kopiert aus dem Eingabepuffer ein Zeichen in neue Zeile
F2 z	Kopiert alle Zeichen bis zum Zeichen »z«
F3	Kopiert aus dem Eingabepuffer alle Zeichen in neue Zeile
F4 z	Alle Zeichen im Eingabepuffer werden bis »z« ignoriert
F5	Beschreibt den Eingabepuffer neu
F6	Ctrl-Z (↑Z)
F7	Nullzeichen
Backspace	Löscht letztes Zeichen in aktueller Zeile
Ins	Zeichen einfügen
Del	Zeichen im Eingabepuffer löschen
Esc	Löscht die gesamte Eingabezeile

Tabelle 2. Editier-Befehle bei »edlin« und MS-DOS

(Anfangszeile),(Endzeile),Zielzeile,(Anzahl) c

Alle Parameter in Klammern sind syntaktisch nicht unbedingt nötig.

Die ersten beiden Parameter bestimmen den zu kopierenden Block (einschließlich Anfangs- und Endzeilen). Als Zielzeile muß diejenige Zeilennummer, von der an der Text eingefügt werden soll, angegeben werden. Der »Anzahl«-Parameter legt fest, wie oft der Text-Block vor die Zielzeile kopiert werden soll.

Als konkretes Beispiel kopieren Sie in Ihrer Beispiel-Datei die ersten beiden Zeilen vor Zeile drei. Dies geschieht mit folgender Anweisung:

```
1,2,3 c
*1
1: Dies ist Zeile 1
2: Zeile 2 folgt sogleich
3:*Dies ist Zeile 1
4: Zeile 2 folgt sogleich
5: Mit Zeile 3 bist Du dabei
*
```

Wie Sie sicher schon bemerkt haben, erscheint beim Auflisten des gesamten ASCII-Textes immer ein Stern vor einer Zeile (hier in Zeile 3). Dieser Stern markiert die gerade aktuelle Zeile. Einige Befehle wirken, wenn keine Zeilennummern angegeben sind, immer auf die gerade aktuelle Zeile.

Zum Verschieben benutzt Edlin eine ähnliche Syntax:

(Anfangszeile),(Endzeile),Zielzeile m
Auch hier wird der ausgewählte Textblock vor die Zielzeile geschoben. Fehlt in diesem Fall die Angabe der Anfangs- und End-Zeile, so verschiebt Edlin nur die aktuelle Zeile.

Im obigen Fall soll die dritte Zeile an das Textende kopiert werden. Hierzu genügt schon folgende Angabe:

```
6 m
```

Um den Ausgangszustand unserer ASCII-Datei wieder herzustellen, müssen zwei Zeilen (Zeile 3 und 5) wieder gelöscht werden. Hierfür bietet Edlin den Delete-Befehl:

(Anfangszeile),(Endzeile) d

Zum Löschen der beiden Zeilen genügen folgende Kommandos »3 d« und »5 d«. Zum Einfügen von Textzeilen steht dem Benutzer der Insert-Befehl zur Verfügung:

(Zeile) i

Wurde dieser Befehl einmal aktiviert, so können beliebig viele Zeilen vor der angegebenen eingefügt werden.

Abgebrochen wird der Einfüge-Modus durch [Ctrl-C] oder [Ctrl-Z] mit anschließendem [Return]. Fällt der Parameter »Zeile« weg, erfolgt die Einfügung nach der aktuellen Zeile.

ASCII-Text listen mit Komfort, geht auch mit Edlin

Auch der List-Befehl akzeptiert Parameter, so daß die vollständige Syntax wie folgt lautet:

(Anfangszeile),(Endzeile) l

Die Angabe von Parameter ist erst bei längeren Programmen nötig. Gibt der Benutzer bei längeren Programmen nur »l« ein, so erfolgt das Auflisten nur bis zum Bildschirmende. Dabei markiert Edlin die aktuelle Zeile wieder mit einem Stern. Versucht der Anwender nun, die nächsten Zeilen über »l« zu erreichen, so stellt er fest, daß Edlin immer die selben Textzeilen darstellt (beginnend mit der aktuellen Zeile). Abhilfe schafft in diesem Fall »page«. Dieser Befehl besitzt die selbe Syntax wie »list«.

(Anfangszeile),(Endzeile) p

Der ASCII-Text wird mit »page« seitenweise dargestellt. Nach Darstellung der Textzeilen am Bildschirm definiert Edlin die zuletzt angezeigte Zeile als aktuelle Textzeile. Drückt der Anwender nun nur [p] und [Return], so werden die nächsten 20 Zeilen am Bildschirm ausgegeben.

Ab MS-DOS Version 2.x erfuhr Edlin eine Erweiterung um die nützlichen Optionen »Suchen und Ersetzen«. Die Syntax dieser beiden Befehle ist entsprechend komplexer:

(Anfangszeile),(Endzeile)(?)sString

Dieser Befehl sucht im angegebenen Zeilenbereich nach dem Suchstring (»String«). Bei Angabe des Fragezeichens erfolgt nach jedem Fund die Rückfrage an den Benutzer, ob dies die richtige Zeile ist.

Der Befehl »Ersetzen« ist um ein zusätzliches Trennzeichen ([Ctrl-Z] – mit der F6-Taste eingeben) und um einen zweiten String erweitert worden:

(Anfangszeile),(Endzeile)
(?)rString1↑zString2

Auch bei diesem Befehl wird durch Angabe des Fragezeichens eine Sicherheitsabfrage vorgenommen, bevor ein String durch einen anderen ersetzt wird.

In den aktuellen Text weitere ASCII-Dateien einzulesen, erreicht man durch den Befehl »transfer«:

(Zeile) tDateiname

Der Text wird dabei ab der angegebenen oder aktuellen Zeile abgelegt. Im Anschluß daran numeriert Edlin das gesamte Programm neu.

Mit »write« werden beliebig viel Textzeilen auf Diskette geschrieben. Die geschriebenen Zeilen werden automatisch aus dem aktuellen Text entfernt. Die Syntax für diesen Befehl lautet:

(Anzahl) w

Wird der Parameter »Anzahl« weggelassen, so schreibt Edlin nur die aktuelle Zeile auf Diskette.

Um Edlin zu verlassen, stehen dem Benutzer zwei Befehle zur Auswahl. »e« beendet das Programm ordnungsgemäß und führt alle Änderungen auf Diskette aus. Verlassen Sie das Programm jedoch mit »quit« (»q«), so bleiben alle Änderungen ohne Wirkung. Der geänderte Text geht dabei verloren.

Die wichtigste Funktion: So editiert man eine Textzeile

Um eine Textzeile zu editieren, muß diese zuerst angewählt werden. Dies geschieht einfach durch Angabe der Zeilennummer gefolgt von [Return]. Darauf erscheint die gewünschte Zeile am Bildschirm. Genau darunter blinkt der Cursor, und erlaubt so, dem Benutzer diese Zeile neu einzugeben.

Dabei müssen Sie natürlich nicht den gesamten Text erneuern. MS-DOS bietet hierfür sehr elegante Lösungen (Tabelle 2), um den Text aus dem Eingabespeicher erneut verwenden zu können.

Abgeschlossen werden die Änderungen mit der Return-Taste. Danach erscheint erneut ein Stern und Edlin wartet auf eine Eingabe. Wie Sie sehen, ist der zeilenorientierte Text-Editor von MS-DOS gar nicht so unbrauchbar, wie es ihm nachgesagt wird. Für einen externen DOS-Befehl bietet Edlin erstaunlich viele wichtige Funktionen eines Textverarbeitungs-Programmes.

(sp)

Der Überblick

Star-Writer PC 3.0 ist die neue Version dieser Textverarbeitungsreihe. Seit seiner Einführung am 17. 11. 1986 wurde Star-Writer PC – von 1.0 bis 2.0 – bereits von über 15.000 zufriedenen Anwendern gekauft.

Star-Writer PC 3.0 ist die konsequente Weiterentwicklung eines hervorragenden deutschen Textprogramms mit vielfältigen Zusatznutzen.

Überarbeitet wurden mit besonderer Sorgfalt das Handbuch, nun mit übersichtlichem Inhaltsverzeichnis, separatem Übungsteil und Registerblättern, die Hilfstexte, Menues und Bildschirmmasken, zum Teil auf Anregung von engagierten Anwendern.

Eine ganze Reihe praktischer Programmerweiterungen – Grafik in Farbe (EGA und CGA), Druckertreiber für Laser, mehrspaltige Textverarbeitung, Fuß- und Endnotenverwaltung, Snapshot, deutsche Rechtschreibkorrektur und Silbentrennung, Konvertierung u.s.w. – geben Star-Writer PC 3.0 das professionelle Finish, das Sie von einer Textverarbeitung erwarten sollten.

Wenn Sie bereits Anwender von Star-Writer PC sind, können Sie die Update-Version Star-Writer PC 3.0 für DM 98,-* bei STAR-DIVISION GmbH bestellen.

Erhältlich im
Fachhandel



Text- verarbeitung

M I T

Z U S A T Z N U T Z E N

F Ü R

Für alle Anwender

Star-Writer PC 3.0 ist ein Textsystem, das ganz bewußt für alle Anwender geschaffen wurde. Ob Sie nun Anfänger oder Profi sind, einfache Briefe schreiben oder aber ein wissenschaftliches Buch mit mehrspaltigem Layout, Fuß- und Endnoten herausgeben wollen, Star-Writer PC 3.0 bietet die Lösung für alle, die schreiben und gestalten.

Die Bedienungsfreundlichkeit von Star-Writer PC ist schon oft in der Fachpresse gelobt worden, nicht zuletzt in der DM 8/87. Entscheidend sind Ihre Fähigkeiten, ob Sie lieber mit der Maus, dem Pull-Down-Menue, den Funktionstasten oder den Controll-Codes arbeiten möchten.

Neue Qualitäten

Star-Writer PC 3.0 ist die neue überarbeitete Version der Star-Writer PC Reihe. Sie zeichnet sich in allen Funktionsbereichen, genau wie Ihre Vorgänger, durch weit über dem Durchschnitt liegende Verarbeitungsschnelligkeit aus. Sie verfügt – wie die Versionen 1.0 und 2.0 – über die Vorteile der schnellen Grafikeinbindung in den Text, sowie über die Adressverwaltung und das DFÜ-Programm zu schnellen Datenübertragung. Aber bei allem was bewährt war, ist Star-Writer PC nun erwachsen geworden.

Die Qualität eines Programms hängt auch im Wesentlichen mit der Überschaubarkeit des Handbuchs zusammen. Und gerade hier setzt Star-Writer PC 3.0 neue Maßstäbe. Es wurde ein Manual mit bei-

spielhaftem Layout geschaffen, mit übersichtlichem Inhaltsverzeichnis, hervorragendem Index, separatem Übungsteil und Registerblättern, Kurzreferenzkarte sowie einem Menue-Poster mit der gesamten Programmstruktur.

Die Hilfstexte, Menues und Bildschirmmasken wurden aktualisiert, textlich völlig neu gestaltet und professionell auf den anspruchsvollen Anwender zugeschnitten.

Layout & Grafik

Die professionelle Darstellung eines beliebigen Textes auf einer zunächst leeren Seite stellte zu allen Zeiten hohe Anforderungen an den Gestalter.

Gutenberg mit der Erfindung der beweglichen Lettern und Mergenthaler mit der Erfindung der Zeilensetzmaschine haben Marksteine auf dem Weg zur schnellen gestalterischen Perfektion gesetzt. Die Anwendung des Computers bei der Texterfassung und dessen Umformung in ein kreatives Layout sind ein weiterer Markstein auf dem Weg zu einer kostengünstigen Information für alle. Textprogramme der Zukunft werden sich an ausgesuchten Funktionen messen lassen müssen. Kann man mit ihnen z. B. Basislayouts erstellen, die auf den nachfolgenden Seiten verändert werden können. Verfügen sie über Proportionschrift im Blocksatz, unterschiedliche Schriftschnitte (fett, kursiv e.t.c.), können sie eine Seite in verschiedenen Spalten aufteilen, mit unterschiedlichen Kopf-, Fuß-, Außen- und

STAR-DIVISION

Star-Writer^{PC}



DM-Urteil 8/87:

Ein sehr gutes
Textprogramm mit
Zusatz-Nutzen zum
sensationellen Preis

A L L E I B M - K O M P A T I B L E N C O M P U T E R

Bundabständen. Ist es möglich den Text rechtsbündig, linksbündig oder auf Mittelachse in einer Spalte oder auf der ganzen Seite zu plazieren. Haben sie umfangreiche Tabulatorfunktionen, die den Text in der ersten Zeile eines Absatzes einziehen oder ganze Absätze einer Layoutseite einrücken lassen (und das in cm oder Zoll). Und nun das Wichtigste an einem guten Layout, die Abbildung. Jeder noch so gut durchdachte Text gewinnt durch die Aussagekraft einer integrierten Grafik. Kann das Textverarbeitungsprogramm Grafiken erstellen oder Fremdgrafiken einlesen, diese verändern, vielleicht sogar farbig gestalten. Alles natürlich schnell und absolut problemlos. Zeigt auch der Ausdruck auf jedem Laser- oder Matrixdrucker das Bild an der richtigen Stelle. Alle diese Fragen kann Star-Writer PC 3.0 ohne Einschränkung mit „JA“ beantworten. Darüberhinaus verfügt er noch über Druckformatsvorlagen (Style-Sheets) und mit Snapshot können Sie z. B. den aktuellen Bildschirminhalt als grafische Abbildung in Ihren Text einbinden und die Aussagefähigkeit dadurch erheblich erhöhen.

Fordern Sie doch bitte ausführliches Informationsmaterial mit dem INFO-COUPON auf dieser Seite an. Denn alle Funktionen und Vorzüge von Star-Writer PC 3.0 können wir Ihnen beim besten Willen auf diesen zwei Seiten nicht nennen.

Bearbeiten

Star-Writer PC 3.0 verfügt über eine perfekte Fußnoten-, ja sogar über eine Endnotenverwaltung. Star-Writer PC 3.0

erstellt Ihnen automatisch Stichwort- und Inhaltsverzeichnisse, Programmvorteile, die gerade beim Schreiben von umfangreicheren Werken, die Arbeit erheblich erleichtern. Markieren, verschieben, kopieren, suchen/ersetzen und löschen Sie Textblöcke, fügen Sie Zeichen, Worte, Absätze oder ganze Texte ein. Bearbeiten Sie bis zu 7 Texte in unterschiedlichen Fenstern gleichzeitig. Springen Sie blitzschnell an den Textanfang, das Textende oder an jede beliebige andere Stelle Ihres Manuskriptes. Star-Writer PC 3.0 wird Ihnen immer präzise und überdurchschnittlich schnell folgen. Star-Writer PC 3.0 verarbeitet selbstverständlich alle im ASCII-Format geschriebenen Texte. Mit Star-Writer PC 3.0 können Sie verhandene Texte aus anderen Programmen importieren, bearbeiten und gegebenenfalls exportieren. Ein entscheidender Vorteil um Ihr gesamtes Equipment optimal zu nutzen.

Korrektur & Silbentrennung

Als neue Funktion im Star-Writer PC 3.0 verfügen Sie mit der Rechtschreibkorrektur über einen Grundwortschatz von 120.000 Wörtern, den Sie beliebig ergänzen können. Damit überprüfen Sie nahezu jedes Wort auf seine richtige Schreibweise, eine Programmerweiterung im Star-Writer PC 3.0, die allein schon den Programmpreis wert ist. Eine weitere Arbeitserleichterung bietet die deutsche Silbentrennung.

Fazit

Star-Writer PC 3.0 ist mit seiner überlegenen Bedienerfreundlichkeit ein Textverarbeitungsprogramm für alle Anwender, die Schreiben und ihren Text gestalten. Sie können, auch als Newcomer sofort mit Star-Writer PC 3.0 arbeiten, denn umfangreiche Hilfstexte leiten Sie sicher und schnell durch Ihre Arbeit. Star-Writer PC 3.0 macht Ihr Computersystem durch seine Kompatibilität, Grafik- und Adressprogrammteile zu einer professionellen Workstation auf dem Schreibtisch. Arbeiten, an die Sie früher nicht einmal zu denken wagten, können nun zu einem Preis realisiert werden, der nahezu konkurrenzlos ist. Weit über 15.000 zufriedene Anwender von Star-Writer PC Programmen bestätigen mit ihrer Wahl, daß Star-Writer PC das Textverarbeitungssystem der Zukunft ist.

DM **398,-***

* Unverbindliche Preisempfehlung



I N F O - C O U P O N



Ja, ich möchte mehr Informationen über Star-Writer PC 3.0 und das weitere Software-Angebot von



STAR-DIVISION

Name, Vorname

Straße, Nr.

PLZ, Ort

Bitte ausschneiden, auf eine Postkarte kleben oder in einen Briefumschlag stecken und an STAR-DIVISION GmbH, Postfach 2830 in 2120 Lüneburg abschicken

Der Debugger

Obwohl er auf fast jeder MS-DOS-Diskette vorhanden ist, führt der Debugger »debug.exe« ein Schattendasein. Das mag darauf zurückzuführen sein, daß die Bedienung nicht gerade bequem oder gar anwenderfreundlich ist. Hat man sich aber erst einmal an die etwas umständliche Syntax gewöhnt, dann verfügt man über ein vielseitiges Hilfsmittel bei der Software-Entwicklung.

Der Debugger kann für eine Vielzahl durchaus unterschiedlicher Aufgaben eingesetzt werden. Dazu gehören unter anderem das Ausführen von Obj-Dateien, das Starten von ROM-Programmen, die Eingabe kleinerer Assembler-Programme und – als Hauptanwendung – das »Entwanzeln« (Debugging) von neu geschriebener Software, also das Beseitigen von Fehlern, die sich erst während des Programmablaufs bemerkbar machen. In erster Linie wird man »debug« auf Assembler-Programme ansetzen, da das Debugging von Hochsprachen auf dieser Ebene zwar möglich, gleichwohl aber sehr aufwendig ist.

Der Debugger wird durch den Befehl »debug« aufgerufen, wobei wahlweise noch der Name eines zu untersuchenden Programms mit angegeben werden kann. Die Bereitschaftsmeldung erfolgt in der Form »-«. Alle Debug-Befehle werden durch einen Buchstaben abgekürzt, wobei der Debugger sowohl Groß- als auch Kleinschreibung akzeptiert (Tabelle 1). Jeder Befehl wird erst nach dem abschließenden Drücken der Return-Taste ausgeführt.

Während der Benutzung des Debuggers bleiben die Standard-Editier- und Funktionstasten von MS-DOS gültig. Die Bildschirmausgabe läßt sich beispielsweise mit [Ctrl-Num-Lock] oder [Ctrl-S] anhalten und mit einer beliebigen Taste fortsetzen. Ebenso bricht [Ctrl-Break] oder [Ctrl-C] die gerade ausgeführte Befehlssequenz ab. Eventuelle Fehlermeldungen werden unter Anzeige der fehlerhaften Zeile angegeben.

Wie bereits gesagt, dient der Debugger unter anderem zum Ausführen von Obj-Dateien und zum Starten von ROM-Software. Um ein Obj-Programm (ein übersetztes, aber nicht gelinktes Programm) zu laden und zu starten, sind drei Befehle notwendig: `n <name> .obj`

`l`
`g = <Startadresse>`
 Angaben in spitzen Klammern sind hier wie auch in allen folgenden Beispielen als Parameter zu verstehen. Ist beispielsweise die Startadresse 0, dann würden Sie statt `g = <Startadresse>` in Wirklichkeit `g = 0`

eingeben. Die tatsächliche Startadresse hängt dabei vom jeweiligen Obj-Programm ab.

Im ROM gespeicherte Programme werden durch einen einfachen Go-Befehl aufgerufen, wobei die Angabe der Segment- und Offset-Adresse notwendig ist:

`g = <Segment> : <Offset>`

Diese Art des Aufrufs wird beispielsweise beim Aufruf der Vorformatierungsroutinen auf den meisten Festplatten-Controllern verwendet; der entsprechende Befehl lautet bei Standard-Festplatten häufig

`g = c800:0005` oder
`g = c800:0006`

Besonders interessant macht der eingebaute einfache Zeilenassembler das Debug-Programm. So ist es sehr einfach möglich, zumindest kürzere Assembler-Programme direkt mit dem Debugger einzugeben und als ausführbare Com-Datei zu speichern. Der Zeilenassembler wird durch den Debug-Befehl »A« aufgerufen. Bei einem Aufruf ohne weitere Parameter ist die Startadresse für das einzugebende Programm 0100hex, die normale Startadresse für ein Com-Programm. Soll jedoch Code an eine bestimmte Stelle im Speicher assembliert werden, dann kann die gewünschte Adresse gleich mit angegeben werden: Mit dem Aufruf »a 0200« wird beispielsweise der as-

semblierte Code ab Offset-Adresse 200hex im Speicher abgelegt. Diese Art des Aufrufs wird insbesondere dann verwendet, wenn eine bereits bestehende Com-Datei nachträglich gepatcht werden soll.

Nach dem Aufruf der Assembler-Funktion gibt Debug die derzeitige aktuelle Adresse aus, in die der nächste Befehl abgelegt werden wird und wartet auf die Eingabe der Assembler-Instruktionen in mnemonischer Form:

```

α 200
3900:0200 mov al,01
3900:0203 mov bx,αx
3900:0205

```

Beachten Sie, daß die Operanden vom Befehl durch ein Leerzeichen getrennt sein müssen, und daß zwischen den Operanden ein Komma steht. Wird ein ungültiger Befehl oder eine syntaktisch falsche Instruktion eingegeben, dann reagiert der Debugger mit einer Fehlermeldung. Außerdem wird die betreffende Adresse angezeigt und eine erneute Eingabe ermöglicht. Ohne Eingabe verlassen Sie den Zeilenassembler durch Drücken der Return-Taste wieder. Im Gegensatz zu einem »echten« Assembler kennt Debug keine symbolischen Adressen und Labels. Assembler-Programme, die beispielsweise mit dem Microsoft Makroassembler MASM erstellt wurden, lassen sich also in der Regel nicht direkt mit Debug eingeben. Für das Programmieren eigener, kleiner Routinen eignet sich der Zeilenassembler von Debug jedoch durchaus.

Der umgekehrte Vorgang der Assemblierung wird bei Aufruf der Disassembler-Funktion durchgeführt, und heißt beim Debugger »Unassembling« (un-assemblieren). Geben Sie den folgenden Befehl ein, um die im vorhergehenden Beispiel erzeugten Assembler-Befehle wieder sichtbar zu machen:

Befehl	Name	Funktion
A	Assemble	Assemblieren einzelner Befehle
C	Compare	Vergleich zweier Speicherblöcke
D	Display	Hexdump eines Speicherbereiches
E	Enter	Eingabe von Hex- oder ASCII-Werten
F	Fill	Füllen eines Speicherbereiches
G	Go	Starten eines Programms
H	Hex	Hexadezimal-Arithmetik
I	Input	Lesen eines I/O-Ports
L	Load	Laden eines Programms
M	Move	Verschieben eines Speicherblocks
N	Name	Dateinamen für Load definieren
O	Output	Ausgabe eines Wertes auf einen I/O-Port
P	Proceed	Unterprogramm aufrufen
Q	Quit	Debug beenden
R	Register	Registerinhalte anzeigen und ändern
S	Search	Zeichenfolge im Speicher suchen
T	Trace	Einzelschrittmodus
U	Unassemble	Disassembler-Funktion
W	Write	Auf Diskette schreiben

Tabelle 1. Alle Debug-Kommandos auf einen Blick


```

-
-n mode.com
-l
-d 0300 0400
2F31:0300 54 68 65 20 49 42 4D 20-50 65 72 73 6F 6E 61 6C The IBM Personal
2F31:0310 20 43 6F 6D 70 75 74 65-72 20 4D 6F 64 65 20 63 Computer Mode c
2F31:0320 6F 6D 6D 61 6E 64 20 56-65 72 73 69 6F 6E 20 33 ommand Version 3
2F31:0330 2E 32 30 20 28 43 29 43-6F 70 79 72 69 67 68 74 .20 (C)Copyright
2F31:0340 20 49 42 4D 20 43 6F 72-70 20 31 39 38 36 20 4C IBM Corp 1986 L
2F31:0350 69 63 65 6E 73 65 64 20-4D 61 74 65 72 69 61 6C icensed Material
2F31:0360 20 2D 20 50 72 6F 67 72-61 6D 20 50 72 6F 70 65 - Program Prope
2F31:0370 72 74 79 20 6F 66 20 49-42 4D 20 41 75 74 68 6F rty of IBM Autho
2F31:0380 72 73 20 2D 20 4D 65 6C-20 48 61 6C 6C 65 72 6D rs - Mel Hallerm
2F31:0390 61 6E 2C 20 52 6F 6E 20-48 65 69 6E 65 79 2C 20 an, Ron Heiney,
2F31:03A0 4B 65 72 72 79 20 53 61-79 65 72 73 20 61 6E 64 Kerry Sayers and
2F31:03B0 20 45 64 20 4B 69 73 65-72 62 77 38 30 62 77 34 Ed Kiserbw80bw4
2F31:03C0 30 63 6F 38 30 63 6F 34-30 63 6F 6D 31 63 6F 6D 0co80co40comicom
2F31:03D0 32 38 30 00 00 3D 63 6F-6D 31 3D 63 6F 6D 32 3D 280..=com1=com2=
2F31:03E0 63 6F 6D 31 3A 3D 63 6F-6D 32 3A 00 00 34 30 00 com1:=com2:..40.
2F31:03F0 00 6C 70 74 31 6C 70 74-32 6C 70 74 33 6D 6F 6E .1pt1lpt2lpt3mon
2F31:0400 6F
-
-
-
-

```

Bild 1. Hexdump von »mode.com« (Ausschnitt)

u 3900:0200

Die Unassembler-Funktion ist besonders nützlich bei der Analyse fertiger Com-Programme, ermöglicht Sie doch ein weitergehendes Verständnis des Programmaufbaus, als dies bei einem einfachen Hexdump der Fall wäre. Und damit wird gleich beim nächsten Thema.

Eine Vielzahl von Debug-Befehlen dient der Speicher-Handhabung, also dem Lesen, Anzeigen und Verändern von Speicherinhalten in vielerlei Form. Die häufigste Anwendung dieser Funktionen ist wohl die Analyse und Modifikation von Programmdateien. Um beispielsweise das MS-DOS-Kommando »mode« zu analysieren, rufen Sie den Debugger einfach auf mit:

```
debug mode.com
```

Es wird automatisch das Mode-Programm hinzugeladen und zur weiteren Analyse bereitgestellt. Beispielsweise können Sie mit dem Dump-Befehl einen Hexdump beliebiger Bereiche des Programms auf den Bildschirm oder Drucker ausgeben:

```
d 0300 0400
```

Der Debugger führt nun einen Hexdump des Programmbereichs zwischen den Adressen 0300hex und 0400hex aus. Wünschen Sie eine Ausgabe auf den Drucker, so drücken Sie vor der Eingabe des Befehls die Tastenkombination [Ctrl-PrtSc] oder [Ctrl-P]. Bild 1 zeigt den dazugehörigen Hexdump. Sie brauchen die Adressen nicht jedesmal einzugeben. Der Aufruf von »D« alleine bewirkt, daß jeweils ein weiteres Teilstück der Mode-Datei ausgegeben wird.

Oft ist es nützlich, in einem Programm nach bestimmten Zeichenfolgen suchen zu können. Dazu dient die Debug-Funktion »S« (Search). Um bei unserem Beispiel zu bleiben, ver-

suchen Sie einmal, den Namen »Mel« im Hexdump zu suchen und in »Jim« abzuändern. Dazu nehmen Sie als erstes eine ASCII-Zeichentabelle zur Hand und schlagen die (hexadezimalen) Codes für die entsprechenden Buchstaben nach. Die ASCII-Codes für »Mel« sind 4Dhex, 65hex und 6Chex, diejenigen für »Jim« sind 4Ahex, 69hex und 6Dhex. Das allgemeine Format des Search-Befehls ist s <Startadresse> <Endadresse> <Liste der zu suchenden Bytes> Daher geben Sie für dieses Beispiel ein:

```
s 0100 0800 4d 65 6c
```

Achten Sie darauf, alle Zahlen hexadezimal anzugeben und durch jeweils ein Leerzeichen voneinander zu trennen. Debug durchsucht nun den angegebenen Bereich und meldet das erste Auftreten der entsprechenden Zeichenfolge. Nun hätte man es sich in diesem Falle noch etwas einfacher machen können, denn gesucht war ja ein Name, also eine ASCII-Zeichenfolge. ASCII-Zeichen lassen sich auch direkt eingeben, wobei jedes Zeichen in Anführungszeichen eingeschlossen sein muß:

```
s 0100 0800 "M" "e" "l"
```

Auch diese Eingabe liefert das gleiche Ergebnis wie die erste. Sie können sogar hexadezimale Angaben mit ASCII-Zeichen beliebig vermischen,

Flagname	gelöscht	gesetzt
Overflow	NV	OV
Direction	UP	DN
Interrupt	DI	EI
Sign	PL	NG
Zero	NZ	ZR
Aux Carry	NA	AC
Parity	PO	PE
Carry	NC	CY

Tabelle 2. Die Debug-Anzeige des Flag-Registers

Debug erkennt automatisch, was jeweils gemeint ist und liefert die Adresse, an der die Zeichenfolge gefunden wurde. Sie können nun den Dump-Befehl benutzen, um den Speicherinhalt ab der gefundenen Adresse aufzulisten und werden feststellen, daß die ersten drei Bytes tatsächlich dem gesuchten Namen entsprechen.

Um den Namen nun in »Jim« zu verändern, verwenden Sie den Enter-Befehl. Er hat das allgemeine Format

```
e <Adresse>
```

Wenn Sie ihn direkt hinter dem Search-Befehl verwenden, ersparen Sie sich die Angabe der Adresse; es wird dann automatisch die letzte aktuelle Adresse, eben das Ergebnis des Search-Befehls, übernommen. Nun zeigt Debug Ihnen die Adresse und darauf folgend den Inhalt dieser Adresse an. Sie können jedes Byte nun unverändert lassen und zur nächsten Adresse springen, indem Sie die Leertaste drücken. Der angezeigte Wert läßt sich aber auch ändern, indem Sie einfach einen neuen Wert angeben (hexadezimal oder als ASCII-Zeichen in Anführungszeichen) und dann erst die Leertaste betätigen. Durch Eingabe eines einzelnen Minuszeichens gelangen Sie um eine Adresse zurück. Dies ist wichtig für den Fall, daß Ihnen einmal ein Fehler bei der Eingabe unterlaufen ist. Um den Enter-Befehl abzuschließen, drücken Sie einfach die Return-Taste.

Bitte lassen Sie beim Modifizieren von Speicherinhalten stets eine gewisse Vorsicht walten, denn schon die Änderung eines einzelnen Byte in einem Maschinenspracheprogramm kann dazu führen, daß das Programm nicht mehr lauffähig ist und – im schlimmsten Fall – beim Start Ihr Computer »abstürzt«. Dennoch brauchen Sie jetzt beim Experimentieren nicht übervorsichtig zu sein. Bislang geschehen alle Änderungen lediglich im Speicher Ihres PC, das Programm auf Diskette ist davon nicht betroffen. Erst wenn Sie ein Programm speichern, werden alle Änderungen, die Sie vorgenommen haben, wirklich wirksam. Zum Speichern dient der Write-Befehl des Debuggers. Dieser Befehl hat die allgemeine Form

```
w <adresse> <Laufwerk>
```

wobei Adresse die Startadresse des Programms ist. Wenn Sie alle Parameter weglassen, dann wird einfach der Bereich gespeichert, der zu Anfang geladen wurde. Wollen Sie davon abweichen, dann muß die Länge

des abzuspeichernden Bereichs – kompliziert genug – in das CX-Register des Prozessors geschrieben werden, was weiter unten beim Register-Befehl ausführlicher zur Sprache kommt. Zunächst jedoch wenden wir uns noch einigen anderen interessanten Funktionen des Debuggers zu.

Da ist zum Beispiel der Compare-Befehl. Er dient dazu, die Inhalte zweier Speicherbereiche zu vergleichen. Das allgemeine Format dieses Befehls ist

c <Startadresse 1> <Endadresse 1>
<Startadresse 2>

Verglichen wird der Speicherbereich ab Startadresse 1 bis Endadresse 1 mit dem Speicherbereich, der bei Startadresse 2 beginnt. Eine zweite Endadresse ist unnötig, da die Länge des zweiten Speicherbereichs durch die Länge des ersten bereits festgelegt ist. Wenn beide Speicherbereiche übereinstimmen, sie also identischen Code enthalten, dann passiert gar nichts weiter und Sie sehen den Debug-Prompt »-«. Andernfalls listet Debug alle gefundenen Unterschiede fein säuberlich auf, wobei jeweils Adresse und Inhalt der beiden Speicherblöcke nebeneinander stehen, soweit Unterschiede vorhanden sind.

Wie Sie bereits gesehen haben, arbeitet der Debugger im hexadezimalen Zahlensystem. Da das Rechnen mit Hexzahlen nicht jedermanns Sache ist, beinhaltet der Debugger auch eine Option, die dem Benutzer die Addition und die Subtraktion zweier Hexzahlen abnimmt:

h <Zahl 1> <Zahl 2>

Dieser Befehl gibt sowohl die Summe als auch die Differenz beider Zahlen aus. Die Ausgabe erfolgt in einer einzigen Zeile und in der Reihenfolge Summe, Differenz.

Möchten Sie Teile des Speichers mit bestimmten Bytes oder Bytefolgen füllen, dann nimmt der Fill-Befehl Ihnen diese Arbeit ab. Um zum Beispiel den Bereich von 0800hex bis 08FFhex mit FFhex-Bytes zu füllen, geben Sie folgendes ein:

f 0800 08ff ff

Wenn Sie anstelle des einzelnen Wertes FFhex zwei Byte angeben, dann wird der Speicherbereich abwechselnd mit diesen beiden Bytes aufgefüllt.

Ähnlich arbeitet der Move-Befehl. Er kopiert einen angegebenen Speicherbereich an eine beliebige Adresse. Das verwendete Format lautet:

m <Startadresse> <Endadresse>
<Zieladresse>

```
-r
AX=0000 BX=0000 CX=1AD0 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=2F31 ES=2F31 SS=2F31 CS=2F31 IP=0100 NV UP EI PL NZ NA PO NC
2F31:0100 E9AB03 JMP 04AB
-r ax
AX 0000
:ffff
-r
AX=FFFF BX=0000 CX=1AD0 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=2F31 ES=2F31 SS=2F31 CS=2F31 IP=0100 NV UP EI PL NZ NA PO NC
2F31:0100 E9AB03 JMP 04AB
-
-
```

Bild 2. Die Registeranzeige durch das R-Kommando

Beispielsweise kopiert der Befehl

m 0100 01ff 0800

den Speicherbereich von 0100hex bis 01FFhex nach 0800hex.

Wie eingangs bereits erwähnt, besteht ein großes Anwendungsfeld für den Debugger in der Fehlersuche und in der Analyse fertiger Programme. Eine ganze Reihe von Befehlen ist denn auch für diesen Zweck ausgelegt.

Wie wird ein Programm nun analysiert? Da wäre zunächst einmal die bereits beschriebene Möglichkeit der Disassemblierung über den Unassemble-Befehl. Damit erhält man (gute Kenntnisse der Assemblersprache natürlich vorausgesetzt) bereits einen ersten, allerdings noch statischen Eindruck vom jeweiligen Programm. Aber Debug hat noch mehr zu bieten: Mit dem Trace-Befehl ist es möglich, ein Programm schrittweise abzuarbeiten, wobei nach jedem ausgeführten Assembler-Befehl eine Anzeige der Inhalte aller Prozessorregister erfolgt. So kann man praktisch Schritt für Schritt mitverfolgen, was in einem Programm nach dem Start alles intern vorgeht. Programmenschleifen, die in Echtzeit nur Sekundenbruchteile dauern, erfordern so allerdings unter Umständen stundenlange Trace-Zeiten.

Ein zeitraubender Nachteil des Trace-Befehls ist die Tatsache, daß Unterprogramme ebenfalls bei Aufruf Befehl für Befehl im Trace-Modus abgearbeitet werden. Für solche Fälle gibt es den Proceed-Befehl »P«. Dieser Befehl arbeitet prinzipiell ganz genauso wie der Trace-Befehl, allerdings mit einem wichtigen Unterschied: Unterprogrammaufrufe werden nicht im Trace-Modus, sondern in Echtzeit durchgeführt. Das Gleiche gilt für Maschinenbefehle, die eine automatische Wiederholfunktion beinhalten;

insbesondere also für die 8086-Stringbefehle.

Des weiteren können auch sogenannte »Breakpoints« gesetzt werden. Ein Breakpoint ist eine definierte Programmunterbrechung, die der Debugger kontrolliert.

Breakpoints werden im Zusammenhang mit dem bereits besprochenen Go-Kommando gesetzt. Beispielsweise bewirkt die Eingabe von »g 0234«, daß das geladene Programm an der aktuellen Adresse gestartet wird, wobei ein Breakpoint an die Adresse 0234hex kommt.

Wenn das Programm die Adresse 0234hex erreicht, bricht es durch den gesetzten Breakpoint sofort ab und die Kontrolle geht wieder an den Debugger zurück.

Eine sehr vielseitige Instruktion des Debuggers ist das Register-Kommando. Durch die Eingabe »R« erfolgt die Anzeige der Inhalte aller Prozessorregister, wie sie durch das im Trace- oder Echtzeitmodus laufende zu untersuchende Programm gesetzt wurden.

Bild 2 zeigt hierzu ein Beispiel. Natürlich können Sie die Registerinhalte auch verändern. Dazu geben Sie hinter dem R-Befehl einfach den Namen eines 8086-Registers an. Ähnlich wie beim Enter-Befehl erscheint nochmals der Registername und der momentane Inhalt, den Sie dann beliebig ändern können. Die Anzeige des Flag-Registers wird Flag für Flag durchgeführt. Tabelle 2 zeigt die Bedeutung der einzelnen Angaben zum Flag-Register.

Zu guter Letzt wollen wir nicht länger verschweigen, wie Sie den Debugger wieder verlassen können: Der Quit-Befehl beendet Ihre Arbeit und Sie verlassen damit den Debugger.

(Martin Hepp/ev)

Stapeldateien (auch Batch-Dateien genannt) erlauben die Programmierung von sich regelmäßig wiederholenden Vorgänge, so daß eine Stapeldatei dem Anwender viel Arbeit abnehmen kann.

Eine Stapeldatei ist eigentlich nur eine Textdatei, die die auszuführenden MS-DOS- und Batchbefehle enthält.

Eine typische Batch-Datei sieht folgendermaßen aus:

```
path c:\sys;c:\bat
rem *** Pfad für Betriebssystem-Befehle ***
rem *** und Batch-Dateien definieren ***
prompt $p$g
rem *** MS-DOS-Bereitschaftszeichen definieren ***
verify on
rem *** Sicherheitsüberprüfung bei ***
rem *** Schreibzugriffen aktivieren ***
keybgr
rem *** Deutsche Tastaturbelegung wählen ***
date
rem *** Datum einstellen ***
time
rem *** Zeit einstellen ***
```

Wenn Sie dieser Stapeldatei die Bezeichnung »autoexec.bat« zuweisen, wird die Datei bei jedem Betriebssystemstart automatisch aufgerufen und abgearbeitet, so daß Sie nach dem Einschalten Ihres Computers keine weiteren Einstellungen vornehmen müssen.

Kurze Stapeldateien lassen sich unter MS-DOS (ohne Textverarbeitungsprogramm) mit der Befehlsfolge »copy con name.bat« auf einfache Art erzeugen. Sämtliche Zeichen, die Sie nach diesem Befehl eingeben, werden in der Datei »name.bat« abgelegt. Wenn Sie die Eingabe beendet haben, müssen Sie die Taste [F6] und [Enter] drücken, um MS-DOS das Ende der Stapeldatei mitzuteilen.

Tabelle 1 listet die Batchbefehle auf, die speziell für den Einsatz in Stapeldateien vorgesehen sind.

Die erklärungsbedürftigen Parameter werden detailliert in Tabelle 2 vorgestellt.

(ma)

Die Batchbefehle für Stapeldateien

Wenn Stapeldateien nur aus den MS-DOS-Befehlen bestehen dürften, wäre ihr Einsatzgebiet stark eingeschränkt.

Deshalb haben die Programmierer von MS-DOS einige Sonderbefehle (Batchbefehle) für Stapeldateien konzipiert, die beispielsweise bedingte und unbedingte Verzweigungen sowie Schleifenstrukturen erlauben.

Die Batchbefehle für Stapeldateien

Befehl	Bedeutung
ECHO	Zeigt Zustand des Echo-Schalters an
ECHO off	Zeigt Kommando, das gegenwärtig ausgeführt wird, nicht an
ECHO on	Zeigt Kommando, das gegenwärtig ausgeführt wird, an
ECHO <text>	Gibt Text <text> auf den Bildschirm aus
ECHO <text> > PRN	Gibt <text> auf den Drucker aus
FOR %%<v> IN <liste> DO <befehl> %%<v>	Weist der Variablen <v> nacheinander die Werte aus der Parameterliste <liste> zu und führt den Befehl <befehl> aus (Verschachtelung nicht erlaubt)
GOTO <marke>	Verzweigt zu der Marke <marke> (zum Beispiel »:label«) innerhalb der Batch-Datei
IF <string1> == <string2> <befehl>	Führt Befehl <befehl> aus, wenn <string1> und <string2> identisch sind
IF errorlevel <nummer> <befehl>	Führt Befehl <befehl> aus, wenn Fehler <nummer> aufgetreten ist
IF EXIST <name> <befehl>	Führt Befehl <befehl> aus, wenn Datei <name> existiert
IF NOT EXIST <name> <befehl>	Führt Befehl <befehl> aus, wenn Datei <name> nicht existiert
PAUSE [<text>]	[Gibt Text <text> aus und] fordert das Drücken einer beliebigen Taste an
REM <kommentar>	Beachtet Befehl und Kommentar <kommentar> nicht
SHIFT	Verschiebt Liste der Parameter um eins nach unten, so daß der nächste Wert im Aufruf von »autoexec.bat« dem Parameter %9 zugeordnet wird (%9 wird zu %8, %8 zu %7, und so weiter)

Tabelle 1. Diese Batchbefehle sind speziell für den Einsatz in Stapeldateien vorgesehen

Die Parameter der Batchbefehle

Parameter	Bedeutung
<befehl>	MS-DOS-Befehl
<liste>	Liste von durch Leerzeichen getrennten Dateien (auch Wildcards erlaubt) oder Daten
<name>	Dateiname (Angabe des Verzeichnisses und Wildcards (*?« und *») optional)
<string>	Von Anführungszeichen eingeschlossene Zeichenfolge
<text>	Zeichenfolge ohne Anführungszeichen
	Im Prompt-Befehl haben folgende Zeichen eine besondere Bedeutung:
\$b:	Pipesymbol ()
\$d:	Datum im nationalen Format
\$e:	Escape-Zeichen (Einleitung für Escape-Sequenz)
\$g:	Größer-Zeichen (>)
\$h:	Backspace (löscht das vorangehende Zeichen)
\$l:	Kleiner-Zeichen (<)
\$n:	Aktuelles Laufwerk (z. B. A, B oder C)
\$p:	Aktuelles Verzeichnis
\$q:	Gleichheits-Zeichen (=)
\$s:	Leerzeichen
\$t:	Uhrzeit im nationalen Format
\$v:	Nummer der MS-DOS-Version
\$:	Dollar-Zeichen (\$)
\$_:	Neue Zeile (Wagenrücklauf und Zeilenvorschub)

Tabelle 2. Die Bedeutung der Parameter in den Batchbefehlen

Die Konfigurationsdatei »config.sys«

Über die Datei »config.sys« kann der Anwender mit wenigen Befehlen die gewünschte Konfiguration seines PCs festlegen.

Die Konfigurationsdatei »config.sys« wird beim Neustart von MS-DOS – sofern vorhanden – automatisch eingelesen. Die Angaben in dieser Datei definieren die Konfiguration des PC, beispielsweise die Zahl der angeschlossenen Laufwerke, das landesspezifische Format für Datum und Uhrzeit sowie die Größe der RAM-Disk.

Tabelle 1 listet alle Befehle, die innerhalb der Datei »config.sys« verwendet werden dürfen, alphabetisch sortiert auf. Tabelle 2 gibt die Bedeutung der verschiedenen Parameter an. Weil die Datei »config.sys« eine Textdatei ist, läßt sie sich, ähnlich wie eine Stapeldatei, über die Tastatur mit der Befehlsfolge »copy con config.sys« erzeugen.

(ma)

Die Parameter der Konfigurations-Anweisungen

Parameter	Bedeutung
<land>	Landescode <land> = 001: USA <land> = 033: Frankreich <land> = 034: Spanien <land> = 039: Italien <land> = 044: Großbritannien <land> = 049: Deutschland
<name>	Dateiname (Angabe des Verzeichnisses und Wildcards »?« und »*« optional)
<n>	Laufwerksbezeichnung mit Doppelpunkt (»A:«, »B:«, »C:« und so weiter)
<opt>	Optionen eines Befehls (einzeln und kombiniert erlaubt)

Tabelle 2. Die Bedeutung der Parameter in den Konfigurations-Anweisungen

Die Anweisungen für die Konfigurationsdatei »config.sys«

Befehl	Bedeutung
BREAK	Zeigt den Zustand des Break-Schalters an
BREAK = off	Tastenkombination [Ctrl-C] und [Ctrl-Break] unterbricht nicht das aktuelle Programm
BREAK = on	Tastenkombination [Ctrl-C] und [Ctrl-Break] unterbricht das aktuelle Programm
BUFFERS = <anzahl>	Legt <anzahl> Diskettenpuffer an (Voreinstellung: 2, Wertebereich: 1 bis 99)
COUNTRY = <land>	Legt als Landeskennzahl <land> fest (bestimmt zum Beispiel Format der Datum- und Zeitanzeige)
DEVICE = <name>	Installiert Datei <name> als Gerätetreiber
DEVICE = ANSI.SYS	Installiert »ansi.sys« als Gerätetreiber (definiert Escape-Sequenzen des Ansi-Standards)
DEVICE = DRIVER.SYS /d: <nummer> [/f: <art>] [/h: <köpfe>] [/s: <sektoren>] [/t: <spuren>] [<opt>]	Installiert »driver.sys« als Treiber für ein zusätzliches Laufwerk <nummer> : Nummer des zusätzlichen Laufwerks (A = 0, B = 1, C = 2 und so weiter) <art> = 0: 320/360 KByte <art> = 1: 1,2 MByte <art> = 2: 720 KByte (Voreinstellung) <art> = 3: 8-Zoll-Laufwerk (einfache Dichte) <art> = 4: 8-Zoll-Laufwerk (doppelte Dichte) <art> = 5: Festplatte <art> = 6: Bandlaufwerk <art> = 7: anderes Laufwerk <köpfe> : Anzahl der Schreib-/Lese-Köpfe (Wertebereich: 1 bis 99) <sektoren> : Anzahl der Sektoren pro Spur (Wertebereich: 1 bis 99) <spuren> : Anzahl der Spuren pro Seite (Wertebereich: 1 bis 999) <opt> = /c: Sicherheitshebel muß zum Schließen des Laufwerks betätigt werden <opt> = /n: Laufwerk ist nicht entfernbar (zum Beispiel Festplatte)
DEVICE = RAMDRIVE.SYS [<größe>] [<byte>] [<anzahl>] [<opt>]	Installiere »ramdrive.sys« als Treiber für eine RAM-Disk im Arbeitsspeicher <größe> : Größe der RAM-Disk in KByte (Voreinstellung: 64, Minimum: 16) <byte> : Sektorgröße in Byte (Voreinstellung: 128, Wertebereich: 128, 256, 512, 1024) <anzahl> : Maximale Anzahl der Dateien im Verzeichnis (Voreinstellung: 64, Wertebereich: 2 bis 1024) <opt> = /e: RAM-Disk liegt im Speicherbereich oberhalb der 1-MByte-Grenze (nur für AT-Benutzer)
DRIVPARM = /d: <nummer> [/f: <art>] [/h: <köpfe>] [/s: <sektoren>] [/t: <spuren>] [<opt>]	Definiert Parameter für Laufwerke um <nummer> : Nummer des angesprochenen Laufwerks (A = 0, B = 1, C = 2 und so weiter) <art> = 0: 320/360 KByte <art> = 1: 1,2 MByte <art> = 2: 720 KByte (Voreinstellung) <art> = 3: 8-Zoll-Laufwerk (einfache Dichte) <art> = 4: 8-Zoll-Laufwerk (doppelte Dichte) <art> = 5: Festplatte <art> = 6: Bandlaufwerk <art> = 7: Anderes Laufwerk <köpfe> : Anzahl der Schreib-/Lese-Köpfe (Wertebereich: 1 bis 99) <sektoren> : Anzahl der Sektoren pro Spur (Wertebereich: 1 bis 99) <spuren> : Anzahl der Spuren pro Seite (Wertebereich: 1 bis 999) <opt> = /c: Sicherheitshebel muß zum Schließen des Laufwerks betätigt werden <opt> = /n: Laufwerk ist nicht entfernbar (zum Beispiel Festplatte)
FCBS = <anzahl1>, <anzahl2>	Erlaubt maximal <anzahl1> geöffnete FCB-Dateien (Voreinstellung: 4), von denen MS-DOS <anzahl2> Dateien (Voreinstellung: 0) nicht automatisch schließen darf (nur für Netzwerke)
FILES = <anzahl>	Es dürfen maximal <anzahl> Dateien gleichzeitig geöffnet sein, von denen MS-DOS bereits fünf für interne Zwecke benötigt (Voreinstellung: 8, Wertebereich: 5 bis 99)
LASTDRIVE = <n>	Letztes Laufwerk, das angesprochen werden kann, ist Laufwerk <n> (Voreinstellung: »e«)
SHELL = <name>	Definiert Datei <name> als neuen Kommandoprozessor

Tabelle 1. Diese Befehle dürfen in der Datei »config.sys« verwendet werden



...UND PLÖTZLICH BRAUCHEN SIE ZUR DATEN- SICHERUNG VON 20 MB ANSTATT 50 DISKETTEN NUR NOCH 40 MINUTEN

Sie sichern auf einem kompakten Band 25 MB Daten mit dem vortex FT 1525.

Wenn Sie regelmäßig größere Datenmengen zu sichern haben, wissen Sie, daß man zur Komplettsicherung einer 20 MB-Festplatte 50 herkömmliche 360 KB-Disketten benötigt. Und daß man dafür über 3 Stunden als Disc-jockey in Action ist.

Das muß aber nicht sein. Denn das vortex Band-Backup-Subsystem FT 1525 faßt in 40 Minuten automatisch auf einem kompakten Band volle 25 MB.

Und so wird's gemacht:

- vortex FT 1525 Adapterkarte in den PC einstecken;
- FT 1525 mit der Adapterkarte verbinden (Kabel wird mitgeliefert);
- die Software installieren;
- das mitgelieferte Band formatieren, fertig!
- Die Datensicherung kann beginnen.

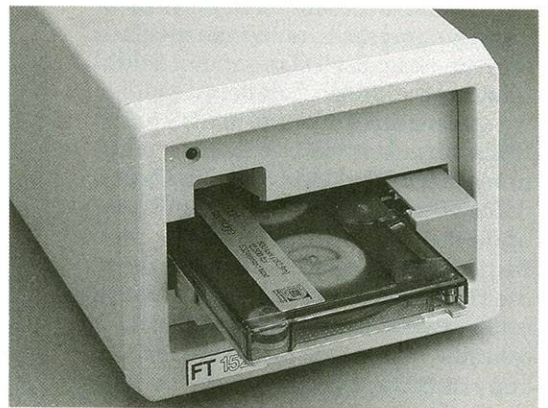
Übrigens verfügt der FT 1525 über ein eigenes Netzteil mit einem angenehm leisen Lüfter. Das Netzteil des PC wird somit nicht zusätzlich belastet.

Lieferumfang:

FT 1525, 1 Controller-Steckkarte, 1 deutsches Handbuch, 1 Systemdiskette, 1 leere Datencassette.

Erforderliche Hardware:

PC 1512 oder andere IBM-Kompatible.



vortex
COMPUTERSYSTEME

...UND PLÖTZLICH LEISTET IHR COMPUTER MEHR

I·N·F·O·S·C·H·E·C·K

Bitte senden Sie mir weitere Informationen über den FT 1525.

vortex Computersysteme GmbH
Falterstraße 51-53 · 7101 Flein · Telefon (071 31) 5 20 61

Daten umleiten mit MS-DOS

Ein kleines Problem führte zur Entdeckung einer nützlichen Einrichtung von MS-DOS: der Umleitung der Standardeingabe- und Standardausgabeeinheit. Beschrieben ist das Ganze zwar auch in einer kurzen, unscheinbaren Passage im dicken MS-DOS-Handbuch, aber erst auf den zweiten Blick zeigt sich, was man damit alles anfangen kann.

So fing es an: Das Problem bestand darin, daß auf einem Computer, der lediglich über Diskettenlaufwerke verfügt, eine RAM-Disk eingerichtet werden sollte, also ein scheinbares oder virtuelles Diskettenlaufwerk, das im Arbeitsspeicher des Computers simuliert wird. Zu Beginn der Arbeit kopiert man einen Teil der wichtigsten Programme auf diese RAM-Disk, um dann während der Arbeit direkten und schnellen Zugriff darauf zu haben. Dazu werden Copy-Befehle in der Autostartdatei »autoexec.bat« eingebaut. Um nicht den gesamten Bildschirm mit Meldungen vollzuschreiben, wird der Befehl »echo off« ganz zu Anfang in die Autoexec-Datei eingefügt. Die entsprechenden Zeilen Ihrer Autoexec-Datei lauten dann:

```
echo off
copy ...
```

So weit, so gut. Ein Problem macht nun aber die Meldung

1 Datei(en) kopiert
die nach jedem Kopiervorgang auf dem Bildschirm steht. Um diese Meldung zu umgehen, muß man alle Daten, die normalerweise auf den Bildschirm geschrieben werden, irgendwo anders hin umleiten, so daß der Bildschirm leer bleibt. Und das geht unter MS-DOS relativ einfach. Die dafür notwendigen Zeichen sind »>« und »<«.

Bisher lautete ein Kopierbefehl, der die Datei »diskcopy« von der Diskette im Laufwerk A auf die RAM-Disk C schreibt

```
copy diskcopy.com c:
Ergänzt man diesen Befehl nun einfach durch eine Umleitungsanweisung, so sieht er folgendermaßen aus:
copy diskcopy.com c: >xxx
Dadurch werden alle Bildschirmausgaben, die der Copy-Befehl verursacht, in die Datei »xxx« umgeleitet. Wenn eine Datei dieses Namens noch nicht existiert, wird sie neu angelegt. Gibt es die Datei bereits, dann wird ihr Inhalt zunächst gelöscht, ehe die
```

umgeleiteten Daten hineingeschrieben werden. Wenn alle 16 Copy-Befehle so verändert sind, erfolgt beim Systemstart tatsächlich keine störende Bildschirrmeldung mehr.

Stattdessen steht erwartungsgemäß der Text »1 Datei(en) kopiert« in der Datei »xxx«. Doch da die Datei »xxx« eigentlich niemals gebraucht wird, ist diese Lösung nur bedingt schön. Wesentlich eleganter löst man die Sache, wenn die umgeleiteten Daten irgendwohin geschickt werden, wo sie gar nichts mehr bewirken – am besten also in eine Art »Datensenke«, wo sie einfach verschwinden. Und auch das ist möglich. Der Weg dorthin geht über die »reservierten Gerätenamen« des MS-DOS.

Im Gegensatz zu den frei wählbaren Dateinamen gibt es im Betriebssystem auch bestimmte Einheitenamen, die spezielle Ein-/Ausgabegeräte bezeichnen. »prn« (für »Printer«) steht zum Beispiel für den Drucker. Tabelle 1 zeigt eine Liste der von MS-DOS unterstützten speziellen Ein-/Ausgabegeräte. Und hier finden Sie auch ein Gerät, das eigentlich gar keins ist, das Null-Gerät oder das »Nichts«. Der Gerätenamen dieser Einheit ist »nul«. Alles, was an das Null-Gerät geschickt wird, verschwindet einfach in der Versenkung, ohne weitere Spuren zu hinterlassen. Also lautet die endgültige Fassung der Copy-Befehle:

```
copy diskcopy.com >nul
Damit ist das »kommentarlose« Kopieren von Dateien schließlich erreicht. Etwaige Fehlermeldungen des Copy-Befehls werden dadurch aber sinnvollerweise nicht unterdrückt, zum Beispiel wenn ein Laufwerk nicht bereit ist.
```

In der Praxis empfiehlt sich aber zur Beruhigung des Benutzers ein kurzer Hinweis:

```
echo Dateien werden kopiert, bitte warten ...
```

Nach dieser ersten Begegnung entfaltet die Ein-/Ausgabe-Umleitung von

MS-DOS so nach und nach ihre vollen Fähigkeiten.

Als nächstes Beispiel wird das Problem der Disketteninhalte gelöst: So mancher ordnungsliebende Benutzer eines PC hat es sich zur lieben Angewohnheit gemacht, den Inhalt von wichtigen Disketten, den er mit dem Befehl »dir« auf dem Bildschirm angezeigt bekommt, mit kleiner Schrift auszudrucken und auf die Diskettenhülle zu kleben. Und so mancher war damit noch nicht zufrieden, sondern hätte gerne die etwas überflüssige Angabe der Uhrzeit der Speicherung weggelassen und stattdessen lieber einen etwas ausführlicheren Kommentar zum Inhalt der jeweiligen Datei dort stehen gehabt – schließlich sind die MS-DOS-Dateinamen recht kurz und daher oft nichtssagend.

Die Lösung, eine mit dem Texteditor erzeugte Beschreibung mit Schere und Kleister neben dem ausgedruckten Verzeichnis anzubringen, ist eines PC-Besitzers grundsätzlich unwürdig. Abhilfe schafft hier die Umleitung der Ausgabe in eine Datei »dir.txt«:

```
dir > dir.txt
```

Gibt der Benutzer dies ein, so arbeitet das Laufwerk emsig, aber der Bildschirm bleibt dunkel – die Ausgabe des Dir-Befehls wird durch das Umleitungszeichen »>« in die Datei »dir.txt« umgeleitet. Dieser Dateiname ist natürlich frei wählbar und kann durch eine vorangestellte Laufwerks- und Pfadangabe ergänzt werden. Ebenso kann der Dir-Befehl mit dem üblichen Laufwerkszusatz und einer Dateiqualfikation versehen werden. Der Befehl

```
dir b:*.bat >c:\temp\dirliste
```

ist also absolut legal. Die so erzeugte Datei läßt sich dann mit einem Editor verändern, der ASCII-Dateien bearbeiten kann. Bild 1 zeigt Ihnen ein kurzes Beispiel eines solchen ergänzten Disketten-Inhaltsverzeichnisses.

Name	Gerät
aux	Serielle Schnittstelle
com1	Erste serielle Schnittstelle
com2	Zweite serielle Schnittstelle
con	Tastatur und Bildschirm
lpt1	Erste Druckerschnittstelle
lpt2	Zweite Druckerschnittstelle
lpt3	Dritte Druckerschnittstelle
nul	Null-Gerät (Datensenke)
prn	Standard-Drucker (Parallele Schnittstelle)

Tabelle 1.
Reservierte Gerätenamen unter MS-DOS


```

***** PROFESSIONAL EDITOR ***** Arbeitsdiskette
Dskt/Platte in Laufwerk A ist DISK 034 A
Verzeichnis von A:\

EDIT      EXE      82176  24.08.84  IBM Prof. Editor V 1.2
ED        BAT       165   25.10.84  Aufruf Editor
TITELB    TXT      1256   17.09.86  Titelverzeichnis
SICHERN   BAT       593   21.01.85  Diskettensicherung
BRIEF     STD       526    1.10.84  Standardbrief
BRIEF     PRP       526    1.10.84  dito, Proportionalschrift
HELVP     LQ       7125   26.10.84  Helvetica LQ Schrift Prop.
DISK034A  ...        18    24.08.84  Diskettenkennung
          ...        ...    ...      Rest Arbeitsdateien

          76 Datei(en)   126624 Byte frei

```

Bild 1. Mit Kommentaren ergänztes Disketten-Inhaltsverzeichnis

Das Ganze braucht man dann nur noch (eventuell nach Einschalten einer kleinen Schrift) zu drucken, entweder gleich von der Textverarbeitung aus oder mittels der Ausgabe-Umleitung auf die Einheit »prn«:

```
type dirli.txt >prn
```

Alle notwendigen Arbeitsschritte lassen sich zur Vereinfachung in einer kleinen Stapelprozedur mit dem Namen »dirli.bat« zusammenfassen (Listing 1). Diese Prozedur wird analog zum normalen Dir-Befehl aufgerufen, zum Beispiel mit:

```
dirli b:*.*bat
```

Eine weitere Anwendung der Ausgabe-Umleitung sehen Sie in der Stapelprozedur »typedru.bat«, die ASCII-Dateien zusammen mit ihrem Namen auf den Drucker ausgibt (Listing 2). Testen Sie dies einmal mit

```
typedru typedru.bat
```

Etwas ist hier neu: die Verwendung des Echo-Befehls. Er gibt einfach die auf ihn folgenden Zeichen weiter, seien es konstante Texte wie die mit Sternchen umrahmten Wörter und die Minuszeichen oder ersetzte %n-Parameter der Prozedur. In unserem Beispiel dienen die Konstanten der optischen »Garnierung« der Ausgabe, während

```
echo %1 >prn
```

den Namen der Datei zur Dokumentation mit angibt. Aus dem Aufruf »typedru dirli.bat« wird also der Echo-Befehl »echo dirli.bat >prn«, der einfach den Dateinamen an den Drucker sendet.

Als weiteres Beispiel zeigt Listing 3 einen kleinen elektronischen Notizzettel. Der Befehl »notiz« öffnet die Tastatur zur Eingabe und hängt die gesamte Benutzereingabe (die mit [F6] oder [Ctrl-Z] abgeschlossen wird) an die vorhandene Datei »notiz.txt« an. Das wird dadurch erreicht, daß im Copy-Befehl die alte Datei »notiz.tx« durch das Zeichen »+« mit der Ta-

statureingabe (Einheitenname »con«) verknüpft, und dann zusammen wieder in die Datei »notizen.txt« kopiert wird. Auch hier wird die Meldung des Copy-Befehls unterdrückt.

Auch die Eingabe-Umleitung kann sinnvoll eingesetzt werden

Umgekehrt geht es aber auch. Die Umleitung der Eingabe bewirkt das Zeichen »<«. Dabei wird eine Eingabe, die normalerweise von der Tastatur kommen soll, von einer anderen Einheit geholt, zum Beispiel wieder aus einer Datei. Hier muß aber die komplette Eingabe für das betreffende Programm, beziehungsweise den betreffenden Befehl enthalten sein. Werden also zwei Eingaben erwartet (wie zum Beispiel vom Format-Befehl), so müssen auch beide in der »Umleitungsdatei« vorhanden sein. Durch geschickten Aufbau von Umleitungsdateien lassen sich so auch DOS-Befehle automatisch ausführen, die eigentlich auf Benutzereingaben warten, wie zum Beispiel »diskcopy« oder »format«.

In diesem Zusammenhang gibt es noch einen wenig bekannten Trick: Auch für die Eingabe-Umleitung kann ebenfalls das Null-Gerät verwendet werden; sie liefert einfach das Zeichen [Return]. Wenn Sie also innerhalb einer Stapeldatei beispielsweise nur das aktuelle Datum anzeigen, aber auf die Eingabe eines neuen Datums verzichten möchten, dann geben Sie einfach den Befehl

```
date <nul
```

ein. Dadurch bleibt dem Anwender das Betätigen der Return-Taste erspart. In analoger Weise kann man auch die Diskcopy-Abfrage »Beliebige Tasten drücken wenn fertig...« umgehen. Allerdings hat die Sache hier einen Haken: »diskcopy« stellt

nämlich nach dem Kopiervorgang noch eine zweite Frage nach weiteren zu kopierenden Disketten, die damit unbeantwortet bleibt. Und das Programm fragt so hartnäckig weiter, daß nur noch die Taste [Ctrl-Break] hilft, um dieser »Falle« wieder zu entkommen. Hier ist also etwas mehr Aufwand gefordert.

Die Lösung besteht darin, beide Fragen automatisch beantworten zu lassen, zum Beispiel aus einer Datei »ein2«:

```
diskcopy a: b: <ein2
```

Die Datei »ein2« enthält dabei als erste Zeile ein beliebiges Zeichen (die zu betätigende Taste) und als zweite Zeile ein »n« (Als Antwort auf die Frage »Weitere Diskette kopieren?«). Wie stellen Sie diese Datei her? Ganz einfach – mit der Ausgabeumleitung:

```
echo x >ein2
```

öffnen Sie eine Datei »ein2« und schreiben Sie das Zeichen »x« als erste Textzeile hinein.

```
echo n >>ein2
```

ergänzt die Datei dann um eine zweite Zeile mit der Antwort »n«. Der Doppelpfeil »>>« bedeutet, daß die Datei nicht neu erzeugt wird, sondern die umgeleitete Ausgabe an das Ende der bestehenden Datei »ein2« angehängt wird. Wenn allerdings keine solche Datei existiert, dann wird sie neu erzeugt und die Doppelpfeil-Umleitung arbeitet wie das einfache »>«.

Eine weitere praktische Stapelprozedur dient als Abhilfe für ein bekanntes und leidiges Problem, mit dem »festplattenlose« PC-Besitzer zu kämpfen haben: die Datensicherung. Mit »diskcopy a: b:« geht das routine-mäßig schnell und einfach. Aber da zuckt der nervöse Finger zur Taste, bevor die richtige Sicherungsdiskette eingelegt ist – und eine andere, wertvolle wird überschrieben. Da wird versehentlich die alte Sicherungsdiskette auf die aktuelle Arbeitsdiskette »gesichert« und damit ein Tagewerk vernichtet und so weiter. Sie wissen ja, was einem alles so passiert, wenn man in Eile ist. Hier ein einfaches, aber sicheres System. Jede Diskette bekommt einen eindeutigen Namen (oder eine Nummer) und den Zusatz »A« für Arbeitsdiskette oder »S« für Sicherungsdiskette. Nach dem Prinzip »doppelt hält besser!« verankern Sie den Namen der Diskette sogar am besten gleich drei Mal: im Kennsatz (mit dem Label-Befehl) und in Form einer Datei gleichen Namens, die, um das

Maß voll zu machen, als Inhalt zusätzlich ihren eigenen Namen trägt. Wenn die Diskette zum Beispiel den Namen »disk034a« trägt, dann sollte sie nach diesem Schema also eine Datei mit dem Namen »disk034a« enthalten und der Inhalt dieser Datei sollte aus einem einzigen Wort bestehen, nämlich wiederum »disk034a«.

Den tieferen Sinn dieser seltsam anmutenden Konstellation werden Sie gleich entdecken. Betrachten Sie aber zunächst die darauf aufbauende Sicherungsprozedur (Listing 4).

Nicht Thema dieses Beitrags, aber trotzdem interessant und zweckmäßig ist ein »Standardkopf« für alle Stapeldateien, den man sinnvollerweise immer davorschaltet. Da man nämlich im Laufe der Zeit zu immer mehr Prozeduren mit gegebenenfalls

unterschiedlichen %n-Parametern kommt, findet man sich bald nicht mehr durch. Daher empfiehlt es sich, jede Prozedur mit ihrem Namen gefolgt von einem »?« aufrufen zu können, um dann Informationen über ihren korrekten Aufruf zu erhalten. Weiter sehen Sie hier auch einen bekannten Trick, mit dem man prüfen kann, ob die Prozedur ohne jeden Parameter aufgerufen wurde (das Zeichen »*« in der If-Abfrage kann dabei durch jedes beliebige Zeichen ersetzt werden). Der erste Teil der Prozedur »sichern.bat« bis zur Marke »:run« dient also der Dokumentation und Fehlerbehandlung. Interessierte Leser mögen ergründen, was dort im einzelnen passiert – zum Thema gehört es nicht unmittelbar. Interessant ist der zweite Teil hinter »:run«. Dort stehen zwei fast identi-

sche Blöcke von Befehlen, die abhängig von der Bestückung der Laufwerke (Arbeitsdiskette in Laufwerk A oder B, Sicherungsdiskette jeweils im anderen Laufwerk) richtig kopieren.

Weil die Diskcopy-Anfrage aus der beschriebenen Datei »ein2« befriedigt wird, sorgt ein selbstprogrammierter Stop (»echo« und »pause«) für die notwendige Unterbrechung – falls erwünscht.

Soviel zu einigen sinnvollen und nützlichen Anwendungen der Ein-/Ausgabeumleitung von MS-DOS. In der praktischen Arbeit – vor allem in Verbindung mit den dazu notwendigen Stapelprozeduren – ist dies, wie Sie gesehen haben, ein nützliches und vielseitiges Instrument.

(Jürgen Beetz/ev)

```
0: echo off
1: dir %1 >xxx
2: edit xxx
3: type xxx >prn
4: del xxx
```

Listing 1. Stapelprozedur »dirli.bat« zum Erstellen und Editieren eines Inhaltsverzeichnisses

```
0: echo off
1: rem Drucke Datei mit ihrem Namen als Kopf.
2: rem Aufruf: TYPEDRU Datei.Ext
3: echo * Start * >prn
4: echo %1 >prn
5: echo ----- >prn
6: type %1 >prn
7: echo * Ende * >prn
```

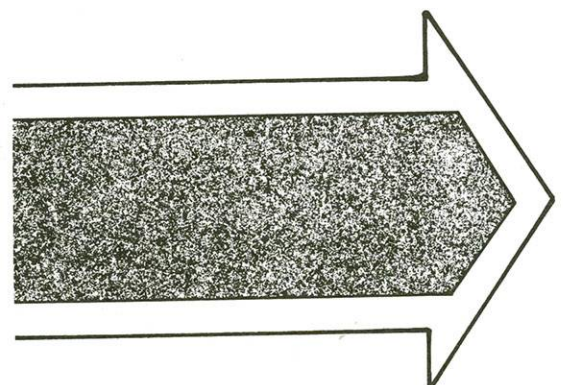
Listing 2. Stapelprozedur »typedru.bat« zum Drucken von Text-Dateien zusammen mit ihrem Namen

```
0: echo off
1: echo Notiz für NOTIZ.TXT eingeben, mit F6 beenden:
2: COPY Notiz.TXT+CON Notiz.TXT >nul
```

Listing 3. Stapelprozedur »notiz.bat« zum Erfassen von kurzen Notizen

```
0: echo off
1: if not %1==* goto exec
2: echo *** FEHLER ! *** Aufruf SICHERN nnn (nnn = Nummer der Diskette)
3: goto end
4: :exec
5: if not %1==? goto run
6: echo Diskette nnn komplett sichern auf DISKnnnS
7: echo Aufruf: SICHERN nnn (nnn=Nummer, muß Datei DISKnnnA enthalten)
8: goto end
9: :run
10: echo SICHERN Diskette %1
11: if exist b:DISK%1a goto lwb
12: echo Quellendiskette LINKS (A:), Zieldiskette RECHTS (B:) einlegen.
13: pause
14: if not exist DISK%1a goto err
15: if not exist b:DISK%1s goto err
16: DISKCOPY a: b: <ein2 >nul
17: RENAME b:DISK%1A DISK%1S
18: goto end
19: :lwb
20: echo Quellendiskette schon in B:, Zieldiskette LINKS (A:) einlegen.
21: pause
22: if not exist DISK%1s goto err
23: DISKCOPY b: a: <ein2 >nul
24: RENAME DISK%1A DISK%1S
25: goto end
26: :err
27: echo *** FEHLER ! *** DISK%1A und DISK%1S nicht korrekt geladen!
28: :end
```

Listing 4. Stapelprozedur zum Sichern von Disketten



Filtern mit MS-DOS

Wer den Begriff »Filter« hört, denkt vielleicht unwillkürlich zunächst an einen Kaffeefilter: Oben läuft etwas hinein, etwas ist darin und unten kommt etwas Neues heraus. Und so abwegig ist der Vergleich gar nicht einmal: In einen MS-DOS-Filter laufen Daten hinein; im Filter ist Programmcode enthalten und aus dem Filter kommen veränderte Daten wieder heraus. Welche praktischen Auswirkungen das hat, erklärt dieser Beitrag.

Ein Filter, so ist es im Handbuch etwas präziser formuliert, »ist ein Programm oder ein Befehl, das beziehungsweise die Daten von einer Standardeingabeeinheit liest, die Daten ändert und das Ergebnis dann auf die Standardausgabereinheit schreibt«. Diese Daten sind also gewissermaßen gefiltert, beim Durchlaufen des Filters geändert worden. »sort« ist so ein Filter, der mit dem DOS mitgeliefert wird. Der Filtervorgang ist das Sortieren der durch den Filter hindurchlaufenden Daten.

Probieren Sie es einfach aus: Die Standardeingabeeinheit ist die Tastatur, die Standardausgabereinheit der Bildschirm. Also tippen Sie folgendes ein:

sort

Zur Erläuterung: »sort« folgt auf die normale, bekannte DOS-Eingabeaufforderung »A>«. Dann bewegt sich der Cursor nach links und nimmt Eingabedaten an, bis Sie mit [F6] oder [Ctrl-Z] die Eingabe beenden. Tippen Sie also weiter:

München
Zürich
Hamburg
Berlin
(Ctrl-Z) oder (F6)

jeweils gefolgt von der Taste [Return]. Das Resultat ist die Anzeige

Berlin
Hamburg
München
Zürich
am Bildschirm.

Nach den Gesetzen der Ein-/Ausgabe-Umleitung können Sie die Ein-

und Ausgabe auch über Dateien abwickeln. Folgender Befehl schreibt also die sortierten Textzeilen nach dem Eintippen in die Datei »stadt«:

sort >stadt

Hier nimmt die Tastatur die Eingaben an wie beim ersten Beispiel, zeigt aber kein Ergebnis am Bildschirm, weil die Ausgabe durch den Zusatz »>stadt« in diese Datei umgeleitet wurde. Davon können Sie sich durch die Eingabe von

type stadt
überzeugen.

Ebenso hätten Sie auch Daten einer Datei »unsort« entnehmen und auf dem Bildschirm sortiert anzeigen können:

sort <unsort

Aus Sicht des Betriebssystems sind Filter externe DOS-Befehle, also nicht in der Programmdatei »command.com« enthalten, die ja bekanntlich beim Starten des DOS in den Hauptspeicher geladen wird und dort zur Ausführung der internen Befehle (wie zum Beispiel »dir« oder »copy«) bereitsteht. Filter sind gleichnamige Programme mit dem Zusatz »exe« oder »com«, die natürlich im Standardverzeichnis auffindbar sein müssen. Wollen Sie also unsere Beispiele nachvollziehen und erhalten die Fehlermeldung »Falscher Befehl oder Dateiname«, dann müssen Sie von Ihrer DOS-Diskette die Datei »sort.exe« und die anderen Filter in das angemeldete Verzeichnis kopieren.

Durch die Einrichtung »Datenübergabe« kann die Bildschirmausgabe ei-

nes Programms als Tastatureingabe eines anderen Programms benutzt werden. Die für die Datenübergabe vorgesehenen Daten werden vom DOS in temporären Dateien gespeichert, die im Stammverzeichnis des Standardlaufwerkes erstellt und wegen ihres Namens »%pipex.\$\$\$« auch einfach als »Pipes« (Röhren) bezeichnet werden – ein Begriff, der manchmal auch für den Vorgang der Datenübergabe selbst verwendet wird (»Pipelining«).

Als Datenübergabe bezeichnet man also das Verketteten von Programmen mit automatischer Umleitung von Standard- und Ausgabe. Die Namen der zu verketteten Programme verbindet das Datenübergabe-Zeichen von DOS, der Senkrecht-Strich. Er wird durch den Alt-Code 124 eingegeben (Taste [Alt] gedrückt halten und auf dem numerischen Ziffernblock »124« tippen). Dadurch reichen Sie die Ausgabe eines Befehls direkt an einen Filter weiter.

Folgender Befehl sortiert also das Dateiinhaltsverzeichnis nach Dateinamen:

dir | sort

Die Ausgabe des Dir-Befehls erscheint nicht am Bildschirm, sondern wird an den Filter »sort« übergeben, der das sortierte Inhaltsverzeichnis anzeigt. Durch die bekannte Ausgabeumleitung wird das Ergebnis in eine Datei »aus« geschrieben, um von dort weiter verarbeitet zu werden:

dir | sort > aus

Der Sort-Filter erlaubt einen Zusatz in der Form »/ + n«, wobei »n« die Spalte angibt, ab der sortiert werden soll. Wenn Sie also in der Dir-Ausgabe richtig abzählen, sehen Sie, daß Sie mit

dir | sort / + 14

nach der Dateigröße sortieren. Dabei werden aber die beiden ersten Zeilen sowie die letzte ebenso wie die anderen behandelt, also mitsortiert, wodurch sie an das Ende beziehungsweise den Anfang der Ausgabe rutschen.

Auf Wunsch können Sie natürlich auch absteigend sortieren, nämlich mit dem Zusatz »/r« (Bild 1):

dir | sort /r / + 14

```
A> DIR *.exe | SORT /+14 /R
```

```
Verzeichnis von A:\
Dskt/Platte in Laufwerk A ist DISK DOS A
      8 Datei(en)      14336 Byte frei
SUBST   EXE      16627  22.04.85  12.00
JOIN    EXE      15987  22.04.85  12.00
ALARM   EXE      12654  16.10.85  15.04
ATTRIB  EXE      15107  22.04.85  12.00
SHARE   EXE       8304  22.04.85  12.00
FIND    EXE       6412  22.04.85  12.00
SP      EXE       2048   5.12.82   0.07
SORT    EXE       1664  22.04.85  12.00
```

Bild 1. Anwendung des Sort-Filters zur Anzeige von Dateien absteigend nach ihrer Größe

```
Dskt/Platte in Laufwerk A ist DISK DOS A
+++++
TEST
Datum ist: Sa. 26.09.1987
Zeit ist: 19.31.12.86
+++++
Testdatei Zeile 1
Testdatei Zeile 2
Testdatei Ende
```

Bild 2. Beispiel einer Dokumentation mit »doku«

Bei großen Inhaltsverzeichnissen rollt allerdings die Ausgabe über den Bildschirm. Was Sie beim einfachen »dir« durch den Zusatz »/p« verhindern konnten, klappt hier allerdings nicht mehr. Aber auch dafür gibt es einen Filter:

```
dir | sort /+ 14 | more
```

»more« ist ein recht einfacher, aber nützlicher Filter: Er stoppt die durch ihn hindurchlaufenden Daten, sobald der Bildschirm voll ist, zeigt am Fuß das Wort »Fortsetzung« an und wartet dann auf das Drücken einer Taste, um dann das Spiel zu wiederholen. So wird das Davonrollen langer Ausgaben am Bildschirm verhindert.

Doch zurück zum Sortieren: Unterverzeichnisse (erkennbar am Zusatz »<DIR>« im Inhaltsverzeichnis) rutschen beim Sortieren nach Dateigröße an den Anfang beziehungsweise an das Ende der Liste, je nach Sortierordnung. Immerhin stehen sie zumindest beisammen, denn oft interessieren nur sie in der Anzeige des aktuellen Verzeichnisses. Gibt es auch eine Möglichkeit, die Unterverzeichnisse direkt herauszufiltern?

Der Filter »find« hilft beim Suchen nach Zeichenketten in Textdateien

Der DOS-Filter »find« siebt alle Datensätze heraus, in denen er eine bestimmte Zeichenkette findet, beziehungsweise (mit einem entsprechenden Befehlszusatz) nicht findet:

```
dir | find "<DIR>"
oder auch nur
dir | find "<"
zeigt nur die Unterverzeichnisse an.
dir | find ".85"
```

findet die alten Dateien heraus, die Sie 1985 abgespeichert haben. Nützlich ist das zum Beispiel für Textdateien, wie zum Beispiel ein Adressen- oder Telefonverzeichnis. Dazu kann man zusätzlich im Find-Befehl einen oder mehrere Dateinamen angeben:

```
find "089-" telefon.txt > telefon.muc
```

stellt alle Ihre Münchner Bekannten in eine neue Datei, aus der Sie dann Serienbriefe schreiben können. Und

```
dir b: | find /v ".EXE"
```

zeigt alle die Dateien im Laufwerk B an, die den Zusatz »exe« nicht haben (wegen der Angabe »/v« im Find-Befehl). Bei der zu suchenden Zeichenkette unterscheidet DOS übrigens Groß-/Kleinschreibung:

```
dir b: | find /v ".EXE"
funktioniert, aber
dir b: | find /v ".exe"
```

klappt nicht. Auch hier können Sie das Resultat in eine Datei übernehmen. Wollen Sie zum Beispiel eine Datei dokumentieren, dazu ihren Namen und die Diskette, auf der sie sich befindet, angeben, so tippen Sie

```
dir | find "Laufwerk" > xxx
echo xyz.bat > > xxx
type xyz.bat > > xxx
```

und erhalten in der Datei »xxx« das gewünschte Ergebnis. Als erstes steht die Laufwerksangabe mit Kennsatz-Information, danach der Name der »xyz.bat« und ihr Inhalt, die beide durch »>>« an die Datei »xxx« angehängt werden.

Die kleine nützliche Prozedur »finde.bat« (Listing 1) durchsucht Dateien nach bestimmten Zeichenketten. Sie enthält den eben beschriebenen sinnvollen Dokumentationskopf und die Prüfung auf fehlende Parameter. Ihr Kernstück ist jedoch die Zeile

```
for %x in (%1) do find %2 %x
```

Das ist eine »Schleife« zum wiederholten Ausführen der Befehle, die auf das »do« folgen. Dafür wird die Variable »x« jeweils auf den nächsten Wert der Liste »(%1)« gesetzt. »%1« ist aber gerade der erste übergebene Parameter. Wurde die Prozedur also mit

```
finde c*.txt "..."
```

aufgerufen und sind auf der Diskette drei Dateien »adresse.txt«, »artikel.txt« und »antwort.txt«, so werden in dieser Schleife die Befehle

```
find "... " adresse.txt
find "... " artikel.txt
find "... " antwort.txt
```

nacheinander ausgeführt. Wer möchte, kann die beiden Parameter des Finde-Kommandos in der Reihenfolge vertauschen und damit die Syntax derjenigen von »find« angleichen. Hier wurde dies jedoch unterlassen, um nicht fälschlicherweise zu suggerieren, daß »find« auch die "Wildcard"-Angabe »*« im Dateinamen erlaubt. Dafür hatten Sie ja die »for..do«-Konstruktion gewählt.

Filter und Datenumleitung arbeiten zusammen

Benutzen Sie nun Filter und Datenumleitung in einer nützlichen Stapelprozedur mit dem Namen »doku.bat« (Listing 2). Sie zeigt als erstes den üblichen Kopf mit Fehlerprüfung und Dokumentation und danach das Löschen der Zwischendatei »xxx«, falls sie existiert:

```
if exist c:xxx del c:xxx
```

Jetzt bauen Sie die Zwischendatei »xxx« auf, die die zu dokumentierende Datei aufnehmen soll. Zuerst »beruhigen« Sie den ungeduldrigen Benutzer mit dem Echo-Befehl und zeigen ihm an, welche Datei jetzt doku-

mentiert werden soll. Danach filtern Sie aus der Directory-Anzeige die Zeile heraus, in der der Disketten-Kennsatz genannt ist. Jetzt werden ein paar Pluszeichen zur Abgrenzung und der Name der Datei aus dem »%1«-Parameter in die Zwischendatei hineingeschrieben. Bild 2 zeigt ein Beispiel einer solchen Dokumentation mit »doku.bat«.

Die Befehle »date« und »time« liefern Datum und Uhrzeit der Dokumentation, damit Sie sie von älteren Versionen unterscheiden können. Beide Befehle haben aber ein zweizeiliges Ergebnis, deswegen filtern Sie jeweils die Zeile »Datum/Zeit ist:« mit »find« heraus.

Beide Befehle fragen aber auch nach einem neu zu setzenden Wert, den Sie aus einer leeren Datei »cr« automatisch als Eingabe holen. Ihr Name soll an »carriage return« (Wagenrücklauf) erinnern, den aus der Schreibmaschinenzeit übernommenen Begriff für die Taste, die eine Eingabe abschließt. Genau genommen ist die Datei natürlich nicht leer, denn sie enthält die Zeichenfolge für »Carriage Return« und »Line Feed«. Die Datei »cr« muß aber sehr sorgfältig konstruiert werden, damit alles reibungslos läuft. Sie bauen die CR-Datei wie immer mit dem Befehl »copy con« auf:

```
copy con cr
```

Nun tun Sie nichts weiter, als einmal die Return-Taste zu drücken, danach [F6] (oder alternativ dazu [Ctrl-Z]) und schließlich noch einmal die Return-Taste. Die resultierende Datei »cr« muß dann genau zwei Byte enthalten. Das und nur das kann als umgeleitete Eingabe für den Date- oder Time-Befehl verwendet werden. Testen Sie die Datei »cr« daher zuerst mit

```
date <cr
```

Das aktuelle Datum muß angezeigt werden und die zweite Zeile mit der Eingabeaufforderung auch, aber ohne daß die Eingabe wirklich abgewartet wird (Rückkehr sofort zur Anzeige »A>«).

In Bild 2 sehen Sie noch einmal den Lohn der Mühe: Die automatisch erstellte Dokumentation einer dreizeiligen Datei »test«. Wenn Sie nun die Doku-Prozedur sich selbst dokumentieren lassen wollen, dann tippen Sie ein:

```
doku doku.bat
```

Damit haben Sie die wesentlichsten Filter kennengelernt. Nun ist es an Ihnen, Sie besitzen jetzt jede Menge Möglichkeiten zum Experimentieren mit der DOS-Filtertechnik. Ganz sicher werden Sie bei der näheren Beschäftigung mit diesen Techniken noch auf völlig neue, ungeahnte Einsatzmöglichkeiten stoßen.

(Jürgen Beetz/ev)


```

echo off
if not %1==? goto exec
echo "Text" in Textdatei(en) finden
echo Aufruf: FINDE Datei.Ext "Text" (Datei-Angabe mit
"wildcards")
echo Beispiel: FINDE *.BAT "Aufruf:" oder FINDE Telefon "089-"
echo Achtung: Blanks in "Text" nicht erlaubt (FIND:
Syntaxfehler)
goto end
:exec
if not %2==* goto run
echo *** FEHLER ! *** Aufruf FINDE Datei.Ext "Text"
goto end
:run
for %%X in (%1) do find %2 %%X
:end

```

Listing 1. Stapelprozedur »finde« zum Auffinden von Zeichenketten in mehreren Dateien

```

echo off
if not %1==? goto exec
echo Textdatei DOKUMENTIEREN auf Drucker
echo mit Datum/Zeit + Disketten-Kennung
echo Aufruf: DOKU Datei.Ext (1 Datei, z.B. AUTOEXEC.BAT)
goto end
:exec
if not %1==* goto run
echo *** FEHLER ! *** Aufruf DOKU Datei.Ext (1 Datei!)
goto end
:run
if exist c:xxx del c:xxx
echo Drucken von "%1":
dir | find "Laufwerk" >c:xxx
echo ++++++++ >>c:xxx
echo %1 >>c:xxx
date <cr | find "ist:" >>c:xxx
time <cr | find "ist:" >>c:xxx
echo ++++++++ >>c:xxx
copy c:xxx + %1 c:xxx >nul
type c:xxx >prn
:end

```

Listing 2. Stapelprozedur »doku« zum Dokumentieren von Textdateien

Desktop Publishing Pur

Vom Scanner bis zum Laserdruck

Halo DPE
GEM Draw plus
Ventura Publisher
Wyse WY 700
Sigma Laserview
JLaser plus AT 3
Canon Scanner IX 12
Canon Laserdrucker



Wir versprechen Ihnen in unseren Anzeigen nicht das Blaue vom Himmel herunter. Denn wir wissen, daß Desktop Publishing - ernstgenommen - doch nicht ganz so einfach ist.

DTP als neues Arbeitsgebiet erfordert gründliche Beratung vor dem Kauf. Wer weiß denn schon, welche Hardware und welche Software harmonisieren und effizient zu dem gewünschten Ergebnis führen? Postscript oder J Laser ist für viele die große Frage.

Und Schulung und Support nach dem Kauf. Denn nicht nur alle Geräte und die Software sind neu und ungewohnt, sondern das gesamte Arbeitsfeld des Publizierens am Schreibtisch.

Deshalb bieten wir neben dem Ventura Publisher auch DTP Hardware wie Bildschirme von Sigma und Wyse, Scanner von Canon, Laserdrucker von Canon und QMS, J Laser-Boards von Tall Tree bis hin zur schlüsselfertigen Installation und der dazugehörigen Schulung an. Fordern Sie unsere ausführlichen Unterlagen an.

CCP

Software Entwicklungs GmbH
Am Grün 54
D-3550 Marburg / Lahn
Tel.: 06421 24081
TTX: 6421920=CCPSOFT

Wir sind autorisierter Distributor für Digital Research, Rank Xerox und Zorland und führen DTP-Software wie: GEM Draw plus, GEM Graph, Ventura Publisher, Halo DPE, Publishers Paintbrush, OCR Read Write

Die erwähnten Produkt- und Firmennamen sind Warenzeichen der jeweiligen Hersteller.

MS-DOS im Überblick

Nahezu jeder PC wird mit dem Betriebssystem MS-DOS ausgeliefert, doch nur wenige PC-Besitzer kennen alle Befehle dieses Betriebssystems und beherrschen es, sie anzuwenden.

Die Hardware eines Computers trägt nicht allein zur Leistungsfähigkeit des Gerätes bei. Mindestens ebenso wichtig ist eine Sammlung von Software-Routinen, die die gegebenen Möglichkeiten der Hardware optimal ausnutzt. Diese Software-Sammlung, die die Hardware eines Computers verwaltet, Ein- und Ausgaben bearbeitet und den Dialog mit Peripheriegeräten steuert, wird als Betriebssystem bezeichnet.

Das Betriebssystem MS-DOS von Microsoft ist das mit Abstand am weitesten verbreitete Betriebssystem für PCs und wird in der Regel beim Kauf eines PC gleich mitgeliefert.

MS-DOS besteht aus drei Komponenten. Die Basiskomponente sind die beiden versteckten Dateien »io.sys« und »msdos.sys«, die für alle grund-

legenden Ein- und Ausgabevorgänge sowie für die Speicher- und Dateiverwaltung zuständig sind.

Der Kommandoprozessor »command.com« enthält alle residenten (ständig im Arbeitsspeicher verfügbaren) MS-DOS-Befehle (»dir«, »copy«, »tree« und so weiter) und überprüft bei der Eingabe von MS-DOS-Befehlen deren Zulässigkeit. Eine Hälfte des Kommandoprozessors wird am Anfang des Arbeitsspeichers abgelegt und die zweite Hälfte am Ende des Arbeitsspeichers. Diese Hälfte wird vom Betriebssystem bei zusätzlichem Speicherbedarf gelöscht und später wieder nachgeladen. Erkennbar ist dies an einer Meldung in der Art »Please insert disk with command.com in drive a:«. Diese Meldung erscheint bei Beendigung des

Programmes, das den Speicherplatz der oberen Hälfte des Kommandoprozessors belegt hat.

Die dritte Komponente von MS-DOS sind die transienten (extern auf Datenträger vorliegenden) MS-DOS-Befehle, die als eigenständige, lauffähige Dateien existieren (»format«, »keybgr«, »diskcopy« und so weiter).

Die Befehle von MS-DOS sind immer nach dem gleichen Schema aufgebaut. Den Beginn eines Befehls bildet das Befehlswort, darauf folgen die Parameter, und als Schlußlicht stehen die Optionen, die aber nur bei Bedarf angegeben werden müssen. In dem Befehl

format a:/v/s
repräsentiert beispielsweise »format« das Befehlswort, »a:« den Parameter und »/v/s« die Optionen.

Tabelle 1 listet alle MS-DOS-Befehle und deren Bedeutung auf. Es werden Befehle bis einschließlich der MS-DOS-Version 3.2 berücksichtigt. Sofern die Bedeutung eines Parameters nicht aus dem Zusammenhang hervorgeht, wird dieser in Tabelle 2 erklärt. (ma)

Die Befehle von MS-DOS (bis Version 3.2)

Befehle	Bedeutung	Befehle	Bedeutung
APPEND	Zeige Verzeichnis(se) an, in dem (denen) Dateien zusätzlich gesucht werden	CD	Zeige aktuelles Verzeichnis
APPEND ;	Lösche aktuelle APPEND-Definition	CD [<n>] [<pfad>]	Ändere aktuelles Verzeichnis zu Verzeichnis <pfad> [auf Diskette in <n>]
APPEND [<n>] [<pfad>]	Suche Dateien zusätzlich im Verzeichnis <pfad> [auf Diskette in <n>]		<pfad> = ..: Kehre in übergeordnetes Verzeichnis zurück
APPEND [<n>] [<pfad1>]; ...	Suche Dateien zusätzlich in den Verzeichnissen <pfad> bis <pfadn> [auf Diskette in <n>]	CHDIR	Zeige aktuelles Verzeichnis
; <pfadn>		CHDIR [<n>] [<pfad>]	Ändere aktuelles Verzeichnis zu Verzeichnis <pfad> [auf Diskette in <n>]
ASSIGN	Hebe Umleitungen der Laufwerke auf		<pfad> = ..: Kehre in übergeordnetes Verzeichnis zurück
ASSIGN <n1> = <n2>	Leite Laufwerk <n1> auf Laufwerk <n2> um (Laufwerksbezeichnungen ohne Doppelpunkt)	CHKDSK [<n>] [<name>]	Überprüfe [Datei <name>] [auf Diskette in <n>]
ATTRIB [<n>] [<name>]	Lese Attribut von Datei <name> [auf Diskette in <n>]	[<opt>]	<opt> = /f: Korrigiere behebbare Fehler
ATTRIB [+R/-R] [+A/-A] [<n>]	+ R: Setze Schreibschutz-Statusbit		<opt> = /v: Zeige überprüfte Dateien an
<name>	- R: Lösche Schreibschutz-Statusbit	CLS	Lösche Bildschirm
	+ A: Setze Archiv-Statusbit (wird nach BACKUP-Sicherung zurückgesetzt)	COMMAND [/c <befehl>] [/e: <größe>]	Starte den Kommandoprozessor »command.com«
	- A: Lösche Archiv-Statusbit		
BACKUP [<n>] [<name>] [<n>]	Kopiere Datei(en) <name> als Sicherheitskopie(n) auf Diskette in <n>		<befehl>: Befehl, der vom Kommandoprozessor »command.com« bearbeitet werden soll
[<opt>]	<opt> = /a: Lösche Zieldiskette vor dem Beschreiben nicht		<größe>: Größe der Betriebssystem-Umgebung (Bereich für SET-Definitionen) in Byte (Voreinstellung: 128)
	<opt> = /d: <datum>: Kopiere nur Dateien, die zu oder nach dem Datum <datum> geändert oder erzeugt wurden		<opt> = /p: neuen Kommandoprozessor »command.com« laden und alten löschen
	<opt> = /l: Lege Datei »backup.log« mit Informationen zum BACKUP-Vorgang auf der ersten Sicherheitskopie-Diskette an	COMP [<opt>] <name1> <name2>	Vergleiche Datei <name1> mit Datei <name2>
	<opt> = /m: Kopiere nur Dateien, die seit dem letzten BACKUP-Vorgang geändert oder erzeugt wurden		<opt> = /a: Zeige Unterschied(e) mit Zeile davor und danach an
	<opt> = /s: Kopiere auch alle Dateien aus den Unterverzeichnissen		<opt> = /b: Vergleiche Zeichen für Zeichen
	<opt> = /t: <zeit>: Kopiere nur Dateien, die zu oder nach der Uhrzeit <zeit> geändert oder erzeugt wurden		<opt> = /c: Ignoriere beim Vergleichen Groß- und Kleinschreibung
BREAK	Zeige den Zustand des Break-Schalters an		<opt> = /i: Vergleiche Zeile für Zeile
BREAK off	Tastenkombination [Ctrl-C] und [Ctrl-Break] unterbricht das aktuelle Programm nicht		<opt> = /n: Zeige beim Vergleichen Zeilennummern an
BREAK on	Tastenkombination [Ctrl-C] und [Ctrl-Break] unterbricht das aktuelle Programm		<opt> = /t: Interpretiere Tabulatoren nicht als Leerzeichen
			<opt> = /w: Interpretiere Tabulatoren als Leerzeichen
			<opt> = <anzahl>: Zeige nach <anzahl> Zeilen Unterschied(e) erneut

Fortsetzung Tabelle 1 auf Seite 31

Befehle	Bedeutung	Befehle	Bedeutung
COPY [<i><n1></i>] <i><name1></i> [<i><opt></i>] [<i><n2></i>] [<i><name2></i>] [<i><opt></i>]	Kopiere Datei <i><name1></i> [von Diskette in <i><n1></i>] [auf Diskette in <i><n2></i>] [als Datei <i><name2></i>] <i><opt></i> = /a: Behandle Datei <i><name1></i> als ASCII-Datei, kopiere Datei nur bis zur ersten Ctrl-Z-Marke <i><opt></i> = /b: Behandle Datei <i><name1></i> als Binärdatei, ignoriere alle Ctrl-Z-Marken <i><opt></i> = /v: Überprüfe nach Kopiervorgang, ob Datei <i><name2></i> mit Datei <i><name1></i> identisch ist		<i><drucker></i> = graphics: Grafikdrucker (Standarddrucker) <i><opt></i> = /b: Drucke auch Hintergrundfarbe (nur bei color4 und color8) <i><opt></i> = /c: Zentriere Bildschirminhalt auf der Seite <i><opt></i> = /f: Drehe Bildschirminhalt um 90 Grad <i><opt></i> = /r: Drucke Bildschirminhalt invers
CTTY <i><gerät></i>	Setze Ein-/Ausgabeeinheit <i><gerät></i> als Standardeingabe-/Ausgabeeinheit	JOIN JOIN <i><n1></i> [<i><n2></i>] <i><pfad></i>	Zeige alle Laufwerks-Umleitungen an Leite Laufwerk <i><n1></i> auf [Laufwerk <i><n2></i> ,] Verzeichnis <i><pfad></i> um (Bedingung: <i><n1></i> <> <i><n2></i>)
DATE	Zeige aktuelles Datum im nationalen Format an	JOIN <i><n></i> /d	Hebe Umlenkung von Laufwerk <i><n></i> auf
DATE <i><datum></i>	Setze Datum auf den Wert <i><datum></i>	KEYBFR	Aktiviere französischen Zeichensatz
DEBUG [<i><n></i>] [<i><name></i>] [<i><parameter></i>]	Starte Debugger Debug [, lade Datei <i><name></i> [von Diskette in <i><n></i>] nach] [und lade Parameter <i><parameter></i> (näheres siehe MS-DOS-Handbuch)]	KEYBGR	Aktiviere deutschen Zeichensatz
DEL [<i><n></i>] <i><name></i>	Lösche Datei <i><name></i> [von Diskette in <i><n></i>]	KEYBIT	Aktiviere italienischen Zeichensatz
DELETE [<i><n></i>] <i><name></i>	Lösche Datei <i><name></i> [von Diskette in <i><n></i>]	KEYBSP	Aktiviere spanischen Zeichensatz
DIR [<i><n></i>] [<i><name></i>] [<i><opt></i>]	Zeige [Datei(en) <i><name></i> von] Verzeichnis [auf Diskette in <i><n></i>] <i><opt></i> = /p: Halte Ausgabe nach jeder Bildschirmseite an und warte auf Tastendruck <i><opt></i> = /w: Zeige lediglich Dateinamen ohne weitere Informationen	KEYBUK	Aktiviere britischen Zeichensatz
DIR MORE	Zeige Verzeichnis seitenweise	KEYBUS	Aktiviere amerikanischen Zeichensatz
DIR SORT	Sortiere Verzeichnis alphabetisch	LABEL [<i><n></i>]	Zeige Namen von Diskette [in <i><n></i>]
DISKCOMP <i><n1></i> <i><n2></i> [<i><opt></i>]	Vergleiche Diskette in <i><n1></i> mit Diskette in <i><n2></i> <i><opt></i> = /1: Vergleiche nur erste Seite der Disketten <i><opt></i> = /8: Vergleiche nur die ersten 8 (statt 9) Sektoren jeder Spur	LABEL [<i><n></i>] <i><diskname></i>	Weise Diskette [in <i><n></i>] Namen <i><diskname></i> zu
DISKCOPY <i><n1></i> <i><n2></i> [<i><opt></i>]	Kopiere Diskette in <i><n1></i> nach Diskette in <i><n2></i> <i><opt></i> = /1: Kopiere nur erste Seite der Diskette auf Diskette in <i><n1></i>	MD [<i><n></i>] <i><pfad></i>	Lege Verzeichnis <i><pfad></i> [auf Diskette in <i><n></i>] neu an
EDLIN [<i><n></i>] [<i><name></i>] [<i><opt></i>]	Starte Zeileneditor »edlin« [und lade Datei <i><name></i> [von Diskette in <i><n></i>] nach] <i><opt></i> = /b: Behandle Datei <i><name></i> als Binärdatei, ignoriere [Ctrl-Z]-Marken	MKDIR [<i><n></i>] <i><pfad></i>	Lege Verzeichnis <i><pfad></i> [auf Diskette in <i><n></i>] neu an
ERASE [<i><n></i>] <i><name></i>	Lösche Datei <i><name></i> [von Diskette in <i><n></i>]	MODE <i><modus></i> [L/R] [,t]	Schalte Bildschirm in Modus <i><modus></i> L: Verschiebe Bild um zwei Zeichen nach links R: Verschiebe Bild um zwei Zeichen nach rechts t: Erzeuge Testbild (nur für Farbmonitor)
EXE2BIN <i><name1></i> [<i><n></i>] <i><name2></i>	Wandle »exe«-Datei <i><name1></i> in »com«-Datei <i><name2></i> [auf Diskette in <i><n></i>] um	MODE LPT1/LPT2/LPT3: [<i><zeichen></i>] [, <i><zeilen></i>] [, <i><opt></i>]	Setze Druckformat für Drucker LPT1, LPT2 oder LPT3 <i><zeichen></i> : Zeichen pro Druckzeile (Voreinstellung: 80) <i><zeilen></i> : Druckzeilen pro Zoll (Voreinstellung: 6) <i><opt></i> = p: Starte nach Überschreiten der Wartezeit erneuten Druckversuch (Abbruch durch [Ctrl-Break])
EXIT	Kehre in übergeordneten Kommandoprozessor zurück	MODE COM1/COM2, <i><baud></i> [, <i><parität></i>] [, <i><daten></i>] [, <i><stop></i>] [, <i><opt></i>]	Bestimme Parameter für Betrieb der seriellen Schnittstelle COM1 oder COM2 <i><baud></i> = Datenübertragungsrate in Bit pro Sekunde <i><parität></i> = E: gerade Parität <i><parität></i> = N: keine Parität <i><parität></i> = O: ungerade Parität <i><daten></i> = Anzahl der Datenbits (7 oder 8) <i><stop></i> = Anzahl der Stopbits (1 oder 2) <i><opt></i> = p: Starte nach Überschreiten der Wartezeit erneuten Übertragungsversuch (Abbruch durch [Ctrl-Break])
FDISK	Richte Festplatte ein (menügeführtes Kommando)	MODE LPT1/LPT2/LPT3 = COM1/COM2	Leite Druckerkanal für LPT1, LPT2 oder LPT3 auf die serielle Schnittstelle COM1 oder COM2 um
FIND [<i><opt></i>] <i><string></i> <i><name></i>	Suche in Datei <i><name></i> nach Zeichenfolge <i><string></i> <i><opt></i> = /c: Zähle nur Anzahl der Zeilen, in denen die Zeichenfolge <i><string></i> enthalten ist <i><opt></i> = /n: Zeige Zeilen mit Zeilennummern, in denen die Zeichenfolge <i><string></i> gefunden wurde <i><opt></i> = /v: Zeige Zeilen/Anzahl der Zeilen, in denen die Zeichenfolge <i><string></i> nicht enthalten ist	MODE LPT1/LPT2/LPT3 = 1/2 MORE	Hebe Umleitung von LPT1, LPT2 oder LPT3 auf COM1 oder COM2 auf
FORMAT [<i><n></i>] [<i><opt></i>]	Formatiere Diskette [in <i><n></i>] <i><opt></i> = /1: Formatiere nur einseitig <i><opt></i> = /4: Formatiere doppelseitig mit 360 KByte Diskettenkapazität (für AT-Benutzer) <i><opt></i> = /8: Formatiere mit acht Sektoren pro Spur <i><opt></i> = /b: Reserviere Bereich für »io.sys« und »msdos.sys« <i><opt></i> = /o: Formatiere Diskette/Festplatte im für MS-DOS 1.x lesbaren Format <i><opt></i> = /s: Formatiere als Systemdiskette/-Festplatte, kopiere automatisch Dateien »io.sys«, »msdos.sys« und »command.com« (diesen Parameter zuletzt eingeben) <i><opt></i> = /v: Erwarte die Eingabe eines Namens für die Diskette/Festplatte	MORE < <i><name></i>	Zeige Informationen seitenweise auf dem Bildschirm (Filterkommando)
GRAFTABL	Definiere Zeichen mit ASCII-Wert 128 bis 255 für Farbgrafikkarte	PATH	Zeige Datei <i><name></i> seitenweise auf dem Bildschirm
GRAPHICS [<i><drucker></i>] [<i><opt></i>]	Schalte Hardcopy von Text- auf Grafikmodus um [und wähle Modus für Drucker <i><drucker></i>] <i><drucker></i> = color1: Farbdrucker mit schwarzem Farbband <i><drucker></i> = color4: Farbdrucker mit vierfarbigem Farbband <i><drucker></i> = color8: Farbdrucker mit vierfarbigem Farbband (cyan-magenta-gelb) <i><drucker></i> = compact: Kompaktdrucker	PATH ; PATH [<i><n></i>] <i><pfad></i>	Zeige Verzeichnis(se) an, in dem (denen) transienten MS-DOS-Kommandos zusätzlich gesucht werden Lösche aktuelle Path-Definition
		PATH [<i><n></i>] <i><pfad1></i> ; ... ; <i><pfadn></i>	Lösche aktuelle Path-Definition Suche transiente MS-DOS-Kommandos zusätzlich im Verzeichnis <i><pfad></i> [auf Diskette in <i><n></i>] Suche transiente MS-DOS-Kommandos zusätzlich in den Verzeichnissen <i><pfad1></i> bis <i><pfadn></i> [auf Diskette in <i><n></i>]
		PRINT	Zeige die Dateien, die sich in der Spooler-Warteschlange befinden Installiere residenten Drucker-Spooler
		PRINT [/b: <i><puffer></i>] [/d: <i><gerät></i>] [/m: <i><takte1></i>] [/q: <i><anzahl></i>] [/s: <i><takte2></i>] [/u: <i><takte3></i>]	<i><puffer></i> : Größe des Druckpuffers in Byte (Voreinstellung: 512, Wertebereich: 1 bis 16384) <i><gerät></i> : Gerät, über das die Ausgabe der Druckerdaten erfolgt (Voreinstellung: PRN) <i><takte1></i> : Anzahl der Taktzyklen, die der Drucker-Spooler für Ausdruck verwenden darf (Voreinstellung: 2, Wertebereich: 1 bis 255) <i><anzahl></i> : Anzahl der Dateien, die in die Spooler-Warteschlange aufgenommen werden können (Voreinstellung: 10, Wertebereich: 1 bis 32) <i><takte2></i> : Anzahl der Taktzyklen, nach denen Drucker-Spooler wieder aufgerufen wird (Voreinstellung: 8, Wertebereich: 1 bis 255)

Fortsetzung Tabelle 1 auf Seite 32

Befehle	Bedeutung	Befehle	Bedeutung
	<take3> : Anzahl der Taktzyklen, bis Drucker nach Ausgabe eines Zeichens wieder betriebsbereit ist (Voreinstellung: 1, Wertebereich: 1 bis 255)	SET <var> = <wert>	Setze Variable <var> auf den Wert <wert>
PRINT <name> [<opt>]	Drucke Datei <name> aus <opt> = /c: Streiche Datei <name> aus der Spooler-Warteschlange <opt> = /p: Reihe Datei <name> in die Spooler-Warteschlange ein <opt> = /t: Lösche alle Dateien aus der Spooler-Warteschlange, schließe gegenwärtig im Druck befindliche Datei und breche Ausdruck ab	SHARE [/f: <anzahl1>] [/l: <anzahl2>]	[Reserviere <anzahl1> Bytes für den File-Sharing-Puffer in Netzwerken (Voreinstellung: 2048)] [Erlaube das Verschießen von maximal <anzahl2> Dateien vor dem Zugriff anderer Benutzer (Voreinstellung: 20)]
PRINT <name1> [<opt>] <name2> [<opt>]	Drucke Dateien <name1> bis <namen> aus	SORT [<opt>] [<name>]	Sortiere eingegebene Daten [Datei <name>] alphabetisch und zeige das Ergebnis (Filterkommando) <opt> = /r: Sortiere nicht ansteigend, sondern absteigend alphabetisch («Z» zuerst) <opt> = /+ <spalte> : Sortierkriterium beginnt in Spalte <spalte>
... <namen> [<opt>]	<opt> = /c: Streiche angegebene Dateien aus der Spooler-Warteschlange <opt> = /p: Reihe angegebenen Dateien in die Spooler-Warteschlange ein <opt> = /t: Lösche alle Dateien aus der Spooler-Warteschlange, schließe gegenwärtig im Druck befindliche Datei und breche Ausdruck ab	SUBST	Zeige alle Laufwerks-Umleitungen an (nur für Netzwerk-Laufwerke)
PROMPT	Definiere Standardprompt (Bezeichnung des aktuellen Standardlaufwerks und Größer-Zeichen) als Prompt	SUBST <n1> [<n2>] <pfad> [<opt>]	Leite Laufwerk <n1> [auf Laufwerk <n2>] in Verzeichnis <pfad> um (nur für Netzwerk-Laufwerke)
PROMPT <text>	Definiere Zeichenfolge <text> als Prompt	SUBST <n>/d	Hebe die Umleitung von Laufwerk <n> auf (nur für Netzwerk-Laufwerke)
RECOVER <n>	Restauriere intakte Sektoren der Diskette in <n>	SYS <n>	Kopiere Dateien »io.sys« und »msdos.sys« auf Diskette in <n>
RECOVER [<n>] <name>	Restauriere intakte Sektoren der defekten Datei <name> [von Diskette in <n>]	TIME	Zeige aktuelle Uhrzeit im nationalen Format an
REN [<n>] <name1> <name2>	Benenne Datei <name1> in <name2> um	TIME <zeit>	Setze Uhrzeit auf Wert <zeit>
RENAME [<n>] <name1> <name2>	Benenne Datei <name1> in <name2> um	TREE [<n>] [<opt>]	Zeige aktuelle Verzeichnisstruktur [von Diskette/Festplatte in <n>] <opt> = /f: Zeige auch alle Dateien in den Unterverzeichnissen
REPLACE [<n1>] <name> <n2> [<pfad>]	Ersetze Datei(en) <name> auf Diskette in <n2> [im Verzeichnis <pfad>] durch Datei(en) <name> im aktuellen Verzeichnis [von Diskette in <n1>]	TYPE <name>	Zeige Datei <name> an
[<opt>]	<opt> = /a: Kopiere nur Dateien, die auf Diskette in <n2> [im Verzeichnis <pfad>] nicht vorhanden sind <opt> = /d: Ersetze Dateien, wenn die bisherigen älter sind <opt> = /p: Warte bei jeder Datei auf eine Bestätigung vor dem Ersetzen <opt> = /r: Ersetze auch schreibgeschützte Dateien <opt> = /s: Kopiere nur die Dateien, die bereits auf Diskette in <n2> [im Verzeichnis <pfad>] vorhanden sind <opt> = /w: Warte mit der Ausführung des Befehls, bis eine Taste gedrückt wurde	TYPE <name> MORE	Zeige Datei <name> seitenweise an
RESTORE <n1> [<n2>] <name> [<opt>]	Ersetze Dateien [auf Laufwerk <n2>] gegen mit BACKUP erzeugte Sicherheitskopien <name> von Laufwerk <n1> <opt> = /p: Warte auf Bestätigung beim Ersetzen von schreibgeschützten Dateien und von Dateien, die nach dem letzten BACKUP-Vorgang geändert wurden <opt> = /s: Kopiere auch alle Dateien aus den Unterverzeichnissen	VER	Zeige die Nummer der MS-DOS-Version
RD [<n>] <pfad>	Lösche Verzeichnis <pfad> [auf Diskette in <n>], wenn es leer ist	VERIFY	Zeige den Zustand des Verify-Schalters an
RMDIR [<n>] <pfad>	Lösche Verzeichnis <pfad> [auf Diskette in <n>], wenn es leer ist	VERIFY off	Überprüfe nach Kopiervorgang nicht, ob Kopie mit Original identisch ist
SELECT <land> <tastatur>	Wähle Landeskennzahl <land> (entspricht COUNTRY-Befehl in CONFIG.SYS) und Tastaturbelegung <tastatur> (entspricht KEYBxx-Befehl)	VERIFY on	Überprüfe nach jedem Kopiervorgang, ob Kopie mit Original identisch ist
SET	Zeige aktuelle Werte der Betriebssystem-Umgebung an (zum Beispiel PATH, PROMPT und COMSPEC)	VOL [<n>]	Zeige Namen der Diskette/Festplatte [in Laufwerk <n>] an
SET <var> =	Lösche Variable <var>	XCOPY [<n1>] <name1> [<opt>]	Kopiere Datei(en) <name1> [von Diskette in <n1>] [auf Diskette in <n2>] [als Datei(en) <name2>], kopiere ebenfalls die Verzeichnisstrukturen <opt> = /a: Kopiere nur Dateien, die seit dem letzten Kopiervorgang geändert oder erzeugt wurden, lasse das Archiv-Statusbit unverändert <opt> = /d: <datum> : Kopiere nur Dateien, die zu oder nach dem Datum <datum> geändert oder erzeugt wurden <opt> = /e: Erzeuge beim Kopieren auch dann Unterverzeichnisse, wenn sie leer sind <opt> = /m: Kopiere nur Dateien, die seit dem letzten Kopiervorgang geändert oder erzeugt wurden, lösche das Archiv-Statusbit <opt> = /p: Warte bei jeder Datei auf eine Bestätigung vor dem Kopieren <opt> = /s: Kopiere auch die Unterverzeichnisse <opt> = /v: Überprüfe nach Kopiervorgang, ob Kopie mit Original identisch ist <opt> = /w: Warte mit der Ausführung des Befehls, bis eine Taste gedrückt wurde
		[<n2>] <name2> [<opt>]	
		<befehl> > <gerät>	Leite Ergebnis des Befehls <befehl> auf Gerät <gerät> um (zum Beispiel PFAD > PRN: Liste Verzeichnis auf Drucker)
		<befehl> > <name>	Leite Ergebnis des Befehls <befehl> in Datei <name> um (zum Beispiel Tree > aufbau.txt: Schreibe Verzeichnisstruktur in Datei »aufbau.txt«)

Tabelle 1. Alle MS-DOS-Befehle mit Parametern, Optionen und deren Bedeutung

Die Parameter der MS-DOS-Befehle

Parameter	Bedeutung	Parameter	Bedeutung
<befehl>	MS-DOS-Befehl		
<datum>	Datum im nationalen Format		
<gerät>	Peripheriegerät		
	<gerät> = CON: Tastatur und Bildschirm		
	<gerät> = PRN: Standarddrucker		
	<gerät> = LPT1/LPT2/LPT3: Zeilendrucker		
	<gerät> = COM1/COM2: serielle Schnittstelle		
	<gerät> = AUX: Hilfsschnittstelle (serielle Schnittstelle)		
<land>	Landescode	<name>	Dateiname (Angabe des Verzeichnisses und Wildcards (*?« und »*«) optional)
	<land> = 001: USA	<n>	Laufwerksbezeichnung mit Doppelpunkt (»a:«, »b:«, »c:« und so weiter)
	<land> = 033: Frankreich	<opt>	Optionen eines Befehls (einzeln und kombiniert erlaubt)
	<land> = 034: Spanien	<pfad>	Verzeichnis auf einer Diskette (zum Beispiel »b:\text\word«)
	<land> = 039: Italien	<string>	Von Anführungszeichen eingeschlossene Zeichenfolge
	<land> = 044: Großbritannien	<tastatur>	Tastaturcode
	<land> = 049: Deutschland		<tastatur> = FR: Frankreich
<modus>	Bildschirmmodus		<tastatur> = GR: Deutschland
	<modus> = mono: 80-Zeichen-Textmodus für Monochromadapter		<tastatur> = IT: Italien
	<modus> = 40: 40-Zeichen-Bildschirmbreite für CGA-Farbgrafikkarte		<tastatur> = SP: Spanien
	<modus> = 80: 80-Zeichen-Bildschirmbreite für CGA-Farbgrafikkarte		<tastatur> = UK: Großbritannien
	<modus> = bw40: 40-Zeichen-Modus, Umschaltung auf CGA-Farbgrafikkarte, Farbmodus ausgeschaltet		<tastatur> = US: USA
	<modus> = bw80: 80-Zeichen-Modus, Umschaltung auf CGA-Farbgrafikkarte, Farbmodus ausgeschaltet	<text>	Zeichenfolge ohne Anführungszeichen
	<modus> = co40: 40-Zeichen-	<zeit>	Uhrzeit im nationalen Format

Tabelle 2. Die Bedeutung der Parameter in den MS-DOS-Befehlen

Hardware PC

1512-Typen	auf Anfrage
1640 MD/SD	1.614,-
1640 CD/SD	2.088,-
1640 MD/HD20	2.614,-
1640 ECD/SD	2.848,-
1640 ECD/HD20	3.848,-
NEC P6 deutsch	1.198,-
Star NL 10 / NG 10	698,-
DMP 2000	520,-
DMP 3160	660,-
DMP 4000	898,-

Neu! Handy Scanner	848,-
Grafpad III	698,-
Software: Starwriter	398,-
TAS+ (relat. Datenb.)	398,-
ABLE ONE (So-Paket)	498,-
Beckerbase PC	99,-
Textomat PC	99,-
Datamat PC	99,-
Sybox Adressverwalt.	95,-
Sybox Dateiverwaltung	145,-
GFA Desk	99,-
GFA Fakt	149,-

Public Domain MS-DOS Software

Mehr als 600 Disketten mit über 10 000 Programmen sofort lieferbar! Bitte fordern Sie unsere Liste an!

Es handelt sich hierbei um Software, die der kommerziell vertriebenen in Leistung und Bedienerfreundlichkeit kaum nachsteht. Wesentlich ist, daß diese Software frei von Copyrights und kostenlos ist. Angegebene Preise dienen der Deckung von Unkosten (Leerdisketten, Kopieren u.s.w.). Das Angebot der Public Domain Software ist sehr vielseitig.

Ein Garant, daß die freien Programme funktionieren und welche Leistung sie bieten, kann nicht gegeben werden!

Nachfolgend einige Beispiele:

• Spieldisketten (z.B. Schach, Flugsimulator, Kartenspiele u.s.w.) • Textverarbeitung (PC-Write) • PC-Musician (Musik zum Selbstkomponieren) • Best Games (Sammlung der besten Spiele) • Key Draw (Farbgrafiksystem) • Present (Diavorführung) • Turbosprite (Demos in Turbo Pascal) • Draftsman (Erstellen von Grafiken) • Deskmates (Hintergrund-Programme) • Cavequest (Adventure) • und viele mehr, Liste anfordern

Jede Diskette

nur 10,- DM

PD-10er Blöcke:

Block 1	Block 2	Block 3	Block 4
- Textverarbeitung	- Datenbankorganisation	- PC-Write	- Drucker-Utilities
- Flugsimulator	- Brasef + Make	- Poster	- Musik
- PC File III	- Bowling Secretary	- Tabellenkalkulation	- Touchtype
- Printer Utilities	- Finance	- Spiele deutsch	- Applikation
- Basic Progr. Utilit.	- Math und Stat	- Prolog	- PC-Calc
- Wertpapier/Finanzen	- Print Grafik I-II	- Datenbank	- Infobase Datenbank
- Mini Assembler	- Basic Translator	- Finance	- PC-ZAP
- Programm Control	- ESIE künstl. Intelligenz	- DOS Tutor, deutsch	- Hilfsprogramme
- "Crossref"	- Wizard's Castle Game	- Monopoly	- Drucker Utilities
- "Startrek"	- Packman and Newtrek	- Horoskop	- Basic Utilities

je Block

nur 68,- DM

Zubehör PC

B-Laufwerk 360 KB	398,-
Aufrüstsatz auf 640 KB	78,-
Lüfterbaustein	98,-
Vortex Drive Card	1.098,-
TANDON Business Card 21 (MB)	898,-
Festplatten (Seagate / Tandon mit Controller und Lüfter):	
20 MB	898,-
30 MB	998,-
40 MB	1.798,-
Farbband DMP 2000/3000	19,80
Farbband DMP 4000	24,80

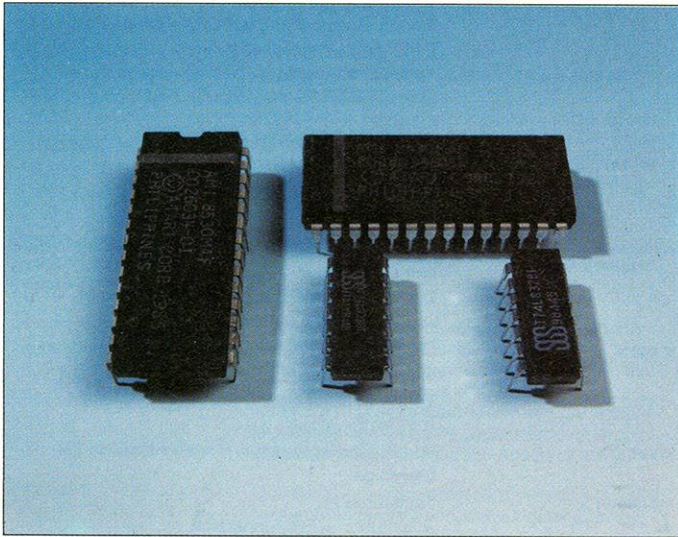
Karl-Heinz Weeske • Potsdamer Ring 10 • 7150 Backnang • Telex 724410 weesba d • Kreissparkasse BK (BLZ 60250020) 74397

weeske
COMPUTER-ELEKTRONIK

Zahlung per Nachnahme erwünscht. Versandkostenpauschale 6,80 DM (Ausland 13,60 DM).

07191/1528-29 od. 60076

Die Speicherbelegung des PC



Der 1-MByte-Speicher des PC ist nach einem festen Schema unterteilt. Welche Daten an welcher Speicheradresse abgelegt sind, erfahren Sie in der folgenden Übersicht.

Im PC arbeitet entweder der Prozessor 8086 oder 8088 von Intel. Beide Prozessoren können über ihre 20 Adreßleitungen maximal 1 MByte (entspricht 2^{20} Bytes) an Speicher adressieren. Die unteren 640 KByte im PC sind als Arbeitsspeicher für das Betriebssystem und Anwender-

programme reserviert. Zusätzlich enthalten die ersten 2 KByte die Interruptvektoren der BIOS- und MS-DOS-Interrupts sowie Systemvariablen von BIOS, Betriebssystem und GW-Basic. Die oberen 384 KByte Speicher des PC stehen für Bildschirmspeicher, Betriebssystem-

Erweiterungen, das BIOS und ähnliches zur Verfügung. Die Tabelle zeigt die Speicheraufteilung eines PC's. Weil sich die Speicherbereiche für Grafikkarten teilweise überschneiden, darf in diesen Fällen nur jeweils eine Grafikkarte in den PC eingebaut werden. (ma)

Die Speicherbelegung des PC

Adresse	Speichergr. (dezimal)	Funktion
00000-0007F	128 Byte	BIOS-Unterbrechungsaufrufe
00080-003FF	896 Byte	MS-DOS-Unterbrechungsaufrufe
00400-0048F	144 Byte	ROM-BIOS Arbeitsspeicher
00413-00414	2 Byte	Größe des Arbeitsspeichers
0041A-0041B	2 Byte	Zeiger auf Anfang des Tastaturpuffers
0041C-0041D	2 Byte	Zeiger auf Ende des Tastaturpuffers
0041E-0043D	32 Byte	Tastaturpuffer (16 Einträge zu je zwei Byte)
00449		Aktueller Bildschirmmodus
0044A-0044B	2 Byte	Textzeichen pro Textzeile
0044C-0044D	2 Byte	Textzeilen pro Bildschirmseite
0044E-0044F	2 Byte	Offsetadresse der aktuellen Bildschirmseite
00450-0045F	16 Byte	Cursorpositionen für acht Bildschirmseiten
00460-00461	2 Byte	Cursorgöße in Rasterzeilen
00462		Aktuelle Bildschirmseite
00490-004EF	96 Byte	Reserviert
004F0-004FF	16 Byte	Von MS-DOS für Anwender reserviert
00500-005FF	256 Byte	Für MS-DOS und Basic reserviert
00500		Statusbyte für Hardcopy (0: Hardcopy beendet, 1: Hardcopy in Bearbeitung, FF: Fehler aufgetreten)
00504		Statusbyte für Einzellaufwerk-Modus
00510-00511		Basic-Segmentadresse
00512-00513		Segmentadresse für Systemuhr-Interrupt Routine (Basic)
00514-00515		Adresse für Systemuhr-Interrupt-Routine (Basic)
00516-00517		Segmentadresse für Break-Tasten-Interrupt-Routine (Basic)
00518-00519		Adresse für Break-Tasten-Interrupt-Routine (Basic)
0051A-0051B		Segmentadresse für Diskettenfehler-Routine (Basic)
0051C-0051D		Adresse für Diskettenfehler-Routine (Basic)

Adresse	Speichergr. (dezimal)	Funktion
00600-nnnnn		»io.sys«, »msdos.sys« und residenter Teil von »command.com«
nnnnn-9FFFF		Freier Arbeitsspeicher
Monochromadap- ter:		
B0000-B0FFF	4 KByte	Text-Bildschirmspeicher (Monochromadapter)
Hercules-Karte:		
B0000-B7FFF	32 KByte	Erste Bildschirmseite (Hercules-Karte)
B8000-BFFFF	32 KByte	Zweite Bildschirmseite (Hercules-Karte)
CGA-Karte:		
B8000-BBFFF	16 KByte	Farb-Bildschirmspeicher (CGA-Karte)
B8000-B8FFF	4 KByte	Erste Bildschirmseite für Textmodus (CGA-Karte)
B9000-B9FFF	4 KByte	Zweite Bildschirmseite für Textmodus (CGA-Karte)
BA000-BAFFF	4 KByte	Dritte Bildschirmseite für Textmodus (CGA-Karte)
BB000-BBFFF	4 KByte	Vierte Bildschirmseite für Textmodus (CGA-Karte)
EGA-Karte:		
A0000-DFFFF	256 KByte	EGA-Karte
C0000-DFFFF	128 KByte	4x16-KByte-Pages für LIM-EMS
C8000-CBFFF	16 KByte	Festplatten-Betriebssystem
CC000-EFFFF	80 KByte	Reserviert für ROM-Erweiterungen
F0000-F5FFF	24 KByte	Ungenutzter ROM-Bereich
F6000-FDFFF	32 KByte	ROM Basic
FE000-FFFFF	8 KByte	ROM BIOS

Tabelle. Bei PCs mit weniger als 640 KByte Arbeitsspeicher liegt die höchste Speicheradresse des Arbeitsspeichers (640 KByte: 9FFFF) entsprechend niedriger

Durchblick Spur für Spur

Die Daten auf Diskette zu speichern ist eine der wichtigsten Aufgaben eines Betriebssystems. Dies geschieht nach einem bestimmten Schema, Format genannt. Das Format von MS-DOS lernen Sie hier kennen.

Die Daten für einen Computer sind meist auf einer Diskette gespeichert, das weiß jeder Computer-Benutzer. Nach welchem System die Daten dort aber abgelegt sind, dafür interessiert man sich meist erst, wenn die tägliche Arbeit durch einen Lesefehler gestört wird. Um den Schaden zu begrenzen, müssen Sie in solch einem Fall Spur für Spur überprüfen, was noch zu lesen ist. Eine Diskette ist unter MS-DOS in 40 Spuren eingeteilt. Man kann sich das als 40 konzentrische Kreise vorstellen, die per Software auf den Datenträger geschrieben werden. Im Gegensatz zu einer herkömmlichen Schallplatte, bei der eine einzige Spur spiralförmig nach innen läuft, sind die Spuren auf der Diskette nicht miteinander verbunden. Die Spuren sind von 0 bis 39 durchnummeriert. Jede Spur wiederum ist in neun Sektoren (Stücke) aufgeteilt. Diese »Tortenstücke« tragen die Nummern 1 bis 9. Insgesamt stehen damit 360 (40 mal 9) Sektoren auf der Diskette. Pro Sektor speichert ein PC 512 Byte, so daß insgesamt 180 KByte auf eine Disketten-seite passen. Werden beide Seiten (Seite 0 und Seite 1) benutzt, so steigt der Speicherplatz solch einer beidseitigen Diskette auf 720 Sektoren, also 360 KByte.

Um einen Sektor genau zu bezeichnen, benötigen Sie drei Werte: Seite, Spur und Sektor. Zur einfacheren Verwaltung werden diese drei Parameter aber zu einer logischen Sektornummer zusammengefaßt. Die Sektornummer liegt immer zwischen 1 und 720. Der logische Sektor 1 entspricht dabei Sektor 1 in Spur 0 auf Seite 0. Die Formel zum Berechnen der logischen Sektoren lautet:

logischer Sektor = $9 \times (\text{Spur} \times \text{Seitenzahl} + \text{Seite}) + \text{Sektor}$

Die Seitenzahl ist entweder 1 oder 2, je nachdem ob die Diskette einseitig oder zweiseitig bespielt ist.

Der erste logische Sektor spielt für den Computer eine sehr wichtige Rolle. Dieser sogenannte »Bootsektor« enthält nämlich alle wichtigen Informationen über die Diskette selbst. Der Name Bootsektor kommt daher, daß hier gegebenenfalls auch die Informationen zum Booten des Betriebssystems stehen. Dazu sind die ersten 30 Byte (Tabelle 1) besonders wichtig. Aus dem Sektor Null (dem ersten Sektor der Diskette) erfahren Sie alles über die Diskette. Das Programm aus Listing 1 (»sector«) zeigt diese Information auf dem Bildschirm. Falls Sie Probleme beim Umrechnen der hexadezimalen Zahlen bekommen, so denken Sie daran, daß die Werte im Intel-Format vorliegen – das heißt, das niederwertige Byte steht vor dem höherwertigen Byte. Für die Bildschirmausgabe betreibt das Programm deshalb auch einen verhältnismäßig großen Aufwand.

In Tabelle 1 taucht zweimal das Wort »FAT« auf. Diese Abkürzung steht für »File Allocation Table«. In dieser Tabelle stehen alle Informationen, die zum Auffinden einer Datei notwendig sind. Unter MS-DOS gibt es zwei dieser Tabellen. Meist sind sie jeweils drei Sektoren lang und belegen die logischen Sektoren 2 bis 4 und 5 bis 7. Hinter den Tabellen steht das Inhaltsverzeichnis. Da jeder Dateieintrag 32 Byte beansprucht, können maximal 144 Dateien eingetragen werden. Das Inhaltsverzeichnis belegt damit neun Sektoren ($144 \times 32 / 512$) – die logischen Sektoren 8 bis 16. Tabelle 2 zeigt, was alles im Inhaltsverzeichnis steht. In Tabelle 3 finden Sie die Dateiattribute, mit denen Sie Dateien verstecken und besonders kennzeichnen können. Diese Werte lassen sich durch Addition beliebig kombinieren.

Das Datum und die Uhrzeit sind nach dem Muster aus Tabelle 4 codiert. Sie müssen dabei wiederum das Intel-Format beachten. HHHHHH ist die Stunde (0 bis 23), MMMMMM die Minute (0 bis 59) und SSSSSS die Sekunde (0 bis 59 in Zweiserschritten). JJJJJJ bezeichnet das Jahr (1980 bis 2099), MMMM den Monat (1 bis 12) und TTTTT den Tag (1 bis 31). Die Informationen stehen als »nor-

Byte	Inhalt	Beschreibung
1 bis 2		Sprung zum Boot-Programm
2 bis 11	verschiedene	Diskettenbezeichnung
12 bis 13	0002	Byte pro Sektor (hier 512)
14	02	Sektoren pro Cluster (hier 2)
15 bis 16	0100	Reservierte Sektoren am Disketten Anfang (hier 1)
17	02	Zahl der FATs (hier 2)
18 bis 19	9000	Maximale Zahl der Dateien im Inhaltsverzeichnis (hier 144)
20 bis 21	D002	Zahl der Sektoren auf Diskette (hier 720)
22	F8	Kennung für einseitig
	F9	Kennung für zweiseitig
23 bis 24	0300	Zahl Sektoren pro FAT (hier 3)
25 bis 26	0900	Sektoren pro Spur (hier 9)
27 bis 28	0100	Anzahl der Seiten (hier 1)
	0200	(hier 2)
29 bis 30	0000	Zahl der versteckten Sektoren (hier 0)

Tabelle 1. Die ersten 30 Byte des Bootsektors. Die Inhaltswerte liegen im Intelformat (niederwertiges Byte zuerst) vor

Byte	Bedeutung
1 bis 8	Name der Datei, immer Großbuchstaben
9 bis 11	Erweiterung der Datei (Buchstaben hinter Punkt)
12	Dateiattribute (siehe Tabelle 3)
13 bis 22	nicht verwendet
23 bis 24	Uhrzeit der letzten Änderung
25 bis 26	Datum der letzten Änderung
27 bis 28	Zeiger auf Dateianfang (erster Cluster)
29 bis 32	Länge der Datei in Byte

Tabelle 2. Der Eintrag im Inhaltsverzeichnis

Wert	Bedeutung
1	Schreibgeschützte Datei
2	Versteckte Datei
4	Betriebssystemdatei
8	Diskettenname
16	Eintrag ist ein Subdirectory
32	Datei ist Archiv
64	nicht verwendet
128	nicht verwendet

Tabelle 3. Die Attribute sind für eine Datei lebenswichtig

Byte 24	– HHHHHMMM
Byte 23	– MMMSSSSS
Byte 26	– JJJJJJJM
Byte 25	– MMMTTTTT

Tabelle 4. Das Datum und die Uhrzeit ist etwas kompliziert codiert

```

0. program sector:
1.
2. ($I getsec.pas)
3.
4. var buffer : feld;
5.     i,temp : integer;
6.
7. begin
8.     get_sector(0,0,0,1,buffer);
9.     write ('Name: ');
10.    for i:=3 to 10 do
11.        write (chr(buffer[i]):1);
12.    writeln;
13.    temp:=buffer[11]+256*buffer[12];
14.    writeln ('Bytes pro Sektor: ',temp);
15.    writeln ('Sektoren pro Cluster: ',buffer[13]);

```

Listing 1. Mit »sector« lesen Sie den Bootsektor

```

16. temp:=buffer[14]+256*buffer[15];
17. writeln ('reservierte Sektoren: ',temp);
18. writeln ('Anzahl FATs: ',buffer[16]);
19. temp:=buffer[17]+256*buffer[18];
20. writeln ('Anzahl Directory Einträge: ',temp);
21. temp:=buffer[19]+256*buffer[20];
22. writeln ('Zahl der Sektoren: ',temp);
23. writeln ('Media Byte: ',buffer[21]);
24. temp:=buffer[22]+256*buffer[23];
25. writeln ('Größe der FAT: ',temp);
26. temp:=buffer[24]+256*buffer[25];
27. writeln ('Sektoren pro Spur: ',temp);
28. temp:=buffer[26]+256*buffer[27];
29. writeln ('Zahl der Köpfe: ',temp);
30. temp:=buffer[28]+256*buffer[29];
31. writeln ('Versteckte Sektoren: ',temp);
32. end.

```



```

0: program direc;
1:
2: const   max_sec = 9;
3:         byte_sec = 512;
4: (9 Sektoren pro Spur; evtl. ändern )
5:
6: type   dir_entry = record
7:     name       : array[1..8] of char;
8:     ext         : array[1..3] of char;
9:     attr        : byte;
10:    time        : byte;
11:    date        : byte;
12:    cluster      : integer;
13:    size         : real;
14: end;
15:
16: ($I getsec.pas )
17:
18: var buffer : feld;
19:     i, temp, res_sec, fat, dir_sec, seite : integer;
20:     dir_len, j, l : integer;
21:     direc : array[1..144] of dir_
22: entry;
23: begin
24:   get_sector(0,0,0,1,buffer);
25:   res_sec:=buffer[14]+256*buffer[15];
26:   fat:=buffer[16];
27:   dir_len:=(buffer[17]+256*buffer[18]) div 16;
28:   fat:=fat*(buffer[22]+256*buffer[23]);
29:   dir_sec:=1+res_sec*fat;
30:   seite:=0;
31:   for l:=1 to dir_len do
32:   begin
33:     if dir_sec>max_sec then
34:       seite:=1;
35:     dir_sec:=dir_sec - max_sec

```

```

36:   end;
37:   get_sector(0,seite,0,dir_sec,buffer);
38:   for j:=0 to 15 do
39:   with direc[(l-1)*16+j+1] do
40:   begin
41:     for i:=1 to 8 do
42:       name[i]:=chr(buffer[j*32+i-1]);
43:     for i:=1 to 3 do
44:       ext[i]:=chr(buffer[j*32+i+7]);
45:     attr:=buffer[j*32+11];
46:     cluster:=buffer[j*32+26]+256*buffer[j*32+27];
47:     size:=0;
48:     for i:=4 downto 1 do
49:       size:=size*256+buffer[j*32+i+27];
50:   end;
51:   dir_sec:=dir_sec+1;
52: end;
53: for i:=1 to dir_len*16 do
54: with direc[i] do
55: begin
56:   if name[1]=chr(0) then halt;
57:   write(i:3,' ');
58:   if name[1]=chr(229) then
59:     write('geloscht');
60:   else
61:   begin
62:     write(name:8);
63:     write(' ');
64:     write(ext:3,' ');
65:     write(attr:3,' ');
66:     write(cluster:3,' ');
67:     write(size:6:0);
68:     write(' ');
69:   end;
70: end;
71: end.

```

Listing 2. »direc« zeigt das Directory ganz intern

male« 4-Byte-Zahl im Intel-Format. Das Programm aus Listing 2 (»direc«) zeigt die Information des Directory. Das Listing ist bewusst unvollständig belassen und soll zum Experimentieren anregen.

Um die Dateibelegungstabelle klein zu halten, wird einer Datei immer mindestens ein ganzer Cluster auf der Diskette zugeteilt. Wieviele Sektoren einen solchen Cluster bilden, steht im 14. Byte des Bootsektors. Auf einer normalen MS-DOS-Diskette sind es zwei Sektoren, auf einer Festplatte oft mehr.

Auf einer normalen Diskette bezeichnet der erste Daten-Cluster die beiden Sektoren unmittelbar hinter dem Inhaltsverzeichnis. Er trägt die Nummer 2 und bezeichnet die logischen Sektoren 17 und 18. Die logischen Sektoren 19 und 20 bilden entsprechend Cluster 3 und so weiter.

Die Dateibelegungstabelle (File Allocation Table) beinhaltet nun die Einträge von 0 bis 352. Mit Hilfe dieser Tabelle finden Sie die Cluster, die zu einer Datei gehören.

Der generelle Aufbau der Dateibelegungstabelle ist einfach. Im Inhaltsverzeichnis steht die Nummer des ersten Clusters (Byte 27 und 28 eines Eintrags).

Damit sind die ersten 1024 Byte (2 x 512 Byte) gefunden.

Deuten die Informationen auf Cluster 7, so findet sich im siebten Eintrag der Dateibelegungstabelle der zweite Cluster der Datei. Nehmen Sie an, dort steht eine 12, so stehen im Cluster 12 die Informationen von Byte 1025 bis 2048 der Datei. Cluster 7 beinhaltet ja die ersten 1024 Byte.

Gleichzeitig befindet sich im Eintrag 12 der Dateibelegungstabelle die Nummer des dritten Cluster und so weiter. Eine Datei ist also nichts weiter als eine verkettete Liste. Doch woran erkennen Sie den letzten Cluster einer Datei?

In dem Fall braucht der Eintrag in der Dateibelegungstabelle nicht auf den nächsten Cluster zu zeigen. Deshalb steht nun ein Wert zwischen FF8hex bis FFFhex darin. Das sind Clusternummern, die es auf keinen Fall geben kann. Wieviele Bytes aus dem letzten Cluster tatsächlich noch zur Datei gehören, das müssen Sie aus der Dateilänge im Inhaltsverzeichnis berechnen.

Das Programm aus Listing 3 (»fat«) gibt eine Dateibelegungstabelle in lesbarer Form aus. Die Verkettung läßt sich sehr einfach erkennen. In der Dateibelegungstabelle stehen aber noch weitere

Byte:	1	2	3
Nibble:	1 2	3 4	5 6
Eintrag:	A A	B A	B B

Tabelle 5. Drei Byte entsprechen sechs Nibble

Informationen. Lautet ein Eintrag 000hex, so ist der Cluster frei. Falls Sie nun eine Datei speichern wollen, müssen Sie die Tabelle von vorne durchsuchen, bis Sie den ersten freien Cluster finden. Dieser wird beschrieben und der nächste freie Cluster gesucht. Dieser Vorgang wiederholt sich, bis die Datei vollständig gespeichert ist.

Beim Löschen einer Datei werden glücklicherweise die Daten nicht physikalisch überschrieben, sondern nur das erste Zeichen des Namens im Inhaltsverzeichnis wird auf E5hex gesetzt. Die restliche Informationen sind noch vorhanden. Schauen Sie sich den entsprechenden Eintrag auf der Diskette an, so finden Sie den ersten Cluster – und die Länge der Datei.

Nun durchsuchen Sie einfach nach obigem Kriterium die Dateibelegungstabelle, um die entsprechenden freien Cluster zu finden. Das sind dann die Clu-

```

0: program fat;
1:
2: const   max_sec = 9;
3:         byte_sec = 512;
4: (9 Sektoren pro Spur; evtl. ändern )
5:
6: ($I getsec.pas )
7:
8: var buffer : feld;
9:     i, nfat, res_sec, fat, secs, cluster : integer;
10:    dir_len, j, l, h1, h2, h3, e1, e2 : integer;
11:    fat_tab : array[1..1536] of byte;
12:    e;
13:    fat_list : array[0..720] of integer;
14: begin
15:   get_sector(0,0,0,1,buffer);
16:   res_sec:=buffer[14]+256*buffer[15];
17:   fat:=buffer[22]+256*buffer[23];
18:   nfat:=buffer[16];
19:   dir_len:=(buffer[17]+256*buffer[18]) div 16;
20:   secs:=buffer[19]+256*buffer[20]-res_sec-dir_len;
21:   cluster:=secs-fat*nfat;

```

```

22:   for l:=res_sec to res_sec+fat do
23:   begin
24:     get_sector(0,0,0,1+l,buffer);
25:     for i:=1 to byte_sec do
26:       fat_tab[(l-1)*byte_sec+i]:=buffer[i-1];
27:   end;
28:   for l:=0 to (cluster div 2)-1 do
29:   begin
30:     h1:=fat_tab[l*3+1];
31:     h2:=fat_tab[l*3+2];
32:     h3:=fat_tab[l*3+3];
33:     fat_list[2*l]:=h1+256*(h2 and 15);
34:     fat_list[2*l+1]:=(h2 and 240) div 16 + h3*16;
35:   end;
36:   for l:=0 to cluster do
37:   begin
38:     write(l:3,' ');
39:     case fat_list[l] of
40:       0 : write('leer ');
41:       1..4086 : write(fat_list[l]:4,' ');
42:       4087 : write('bad ');
43:       else : write('ende ');
44:     end;
45:   end;
46: end.

```

Listing 3. »fat« zeigt die Verkettung deutlich an


```

0: ( siehe Turbo Pascal Handbuch für diesen Record )
1: type registers = record case integer of
2:   1: (ax,bx,cx,dx,bp,si,di,ds,es,flags: in
3:     teger);
4:   2: (al,ah,bl,bh,cl,ch,dl,dh      : by
5:     te);
6:   end;
7:   field      = array [0..511] of byte;
8: procedure get_sector(drive,seite,spur,sector: byte; var buf
9:   fer: field);
10: ( drive: 0 für Laufwerk A:, 1 für B: usw.
11:   seite: 0 oder 1
12:   bei einseitigen disketten nur 0 verwenden
13:   spur: 0 bis 39 für 40 Spur Laufwerk
14:   0 bis 79 für 80 Spur Laufwerk
15:   sector: 1 bis 9 für 9 Sektoren pro Spur
16:   1 bis 8 für 8 Sektoren pro Spur

```

Listing 4. Auch Sektoren lassen sich lesen

ster der Datei. Da E5hex ein Zeichen ist, das auch in Dateinamen vorkommen darf, wird das Löszeichen mit dem ASCII-Wert 229 (E5hex) als 05hex im Directory eingetragen.

Der Eintrag 001hex ist nicht erlaubt. Die Einträge 002hex bis FF6hex stellen die Zeiger auf den jeweils nächsten Cluster dar. FF7hex zeigt einen defekten Cluster an, die Einträge FF8hex bis FFFhex das Dateiende.

Das Problem der Dateibelegungstabelle ist die Länge der Einträge mit 1,5 Byte. Dazu kommt noch der Eintrag im Intel-Format. Die Aussage dieser Informationen zu verstehen ist also nicht ganz einfach. Ein halbes Byte (4 Bit) wird auch als Nibble bezeichnet. 3 Nibble brauchen Sie also für einen Eintrag in der Tabelle. Tabelle 5 zeigt 3 Byte. Die Ziffern 1 bis 6 markieren die entsprechenden Nibble. Die Buchstaben A und B

bezeichnen den Eintrag in der Dateibelegungstabelle.

Der erste Eintrag (A) steht in den Nibbles 4, 1 und 2 (wegen dem Intel-Format in dieser Reihenfolge, den Eintrag B beinhalten die Nibbles 5, 6 und 3).

Die Cluster 0 und 1, die ja im Datenbereich nicht existieren, haben noch einen Extra-Eintrag. Der Eintrag in 0 lautet Fxxhex, wobei xx für F8 einseitig, 80 Spuren, F9 zweiseitig, 80 Spuren FC einseitig, 40 Spuren oder FD zweiseitig, 40 Spuren, steht. Im Eintrag 1 steht bei allen Disketten FFFhex.

Damit lauten die ersten drei Byte der Dateibelegungstabelle xxFFFFhex. Daran erkennt der Computer, ob die Tabelle Einträge mit 12 oder 16 Bit enthält. Besonders auf Festplatten wird oft das größere Format benutzt. Der erste Eintrag lautet dann xxFFFFFhex.

Unter MS-DOS können Sie Unterver-

zeichnisse anlegen. Das ist relativ einfach, da ein Unterverzeichnis im Grunde nichts anderes als eine normale Datei ist.

Im Inhaltsverzeichnis ist lediglich das Attribut »Subdirectory« (siehe Tabelle 2 und 3) gesetzt. Die Datei ist wie ein normales Inhaltsverzeichnis aufgebaut. Lediglich die beiden ersten Einträge ».« und »..« haben eine besondere Funktion. ».« zeigt mit der Clusternummer auf den Beginn des Unterverzeichnisses. Daher kennt das Betriebssystem den Eintrag in der Dateibelegungstabelle und den zweiten Cluster des Unterverzeichnisses

»..« zeigt auf das Stammverzeichnis. Lautet die Clusternummer des Stammverzeichnisses Null, so ist das Hauptinhaltsverzeichnis gemeint. Listing 4 zeigt eine Prozedur zum Lesen eines Sektors.

(Rainer W. Gerling/hg)

Wer wo wann?

Bei Zuschriften bitten wir Folgendes zu beachten: Haben Sie Fragen, Anregungen oder Programme, die Sie der DOS-Redaktion zukommen lassen möchten, adressieren Sie wie folgt:

**Redaktion
DOS International
Elfenstr. 40
8000 München 43
Tel. (089) 60 20 66**

Haben Sie Fragen, die zum Beispiel Abonnements, Kleinanzeigen oder Databox-Bestellungen betreffen, wenden Sie sich bitte an den Verlag:

**DMV Verlag GmbH
DOS International
Fuldaer Straße 6
3440 Eschwege
Tel. (05651) 87 02
Telex 993 210 dmv d**

Die Software zum Anschluß von 40-, 80-Track, single-, double-, high-density, 3"-, 3,5"-, 5,25"- und 8"-Laufwerken an Ihren PC/XT/AT-kompatiblen DOS-Rechner unter MS-DOS 2.1, 3.1, 3.2 (bis 2000 KB netto):

RWMSDOSX: Lesen, Schreiben und Formatieren von bel. DOS-Disketten von 160 KB bis 1800 KB. Mischbetrieb für bis zu vier Laufwerke. Automatische Formaterkennung. Z. B. netto über 420 KB auf 40-Track-, über 840 KB auf 80-Track-, 1500 KB auf AT-Laufwerk, 2000 KB auf TEAC 35 HFN-22. DM 300,—

RWCPM: Lesen, Schreiben und Formatieren von CP/M-Disketten unter MS-DOS. Fast jedes(!) CP/M-Format — natürlich auch 8"-IBM-Standard — einstellbar von 3" bis 8". Preis incl. CP/M-80-Emulator: DM 300,—

RWMAIN: Lesen, Schreiben und Formatieren von 8"-MAINFRAME-Disketten (IBM, Siemens usw.) DM 2000,—

Huckepackplatte (leer) zum Umrüsten des FDC auf Multifunktionskarte auf High-Density-Betrieb (max. 1800 KB netto) ebenfalls erhältlich!

Bei A.S.S.-WARE kaufen Sie nicht die Katze im Sack: Sie erhalten vorab die gesamte(!) Dokumentation + Demosoftware auf Diskette. (5,— in Bfm. erwünscht.)

Günstige Preise für Schüler, Studenten usw. (z. B. Super-Bios, RWMSDOS DM 100,—)

**A.S.S.-Ware, Alfred Herrmann
Schimmelshahn
5461 Roßbach, Tel.: 0 26 38/45 13**

C COMPILER MI-C für CP/M, CP/M 86, MS-DOS

vereint hohen Bedienungskomfort mit hervorragender Leistung

- Vollständige Version mit 13stelliger BCD-Arithmetik für Gleitkommazahlen
- Erzeugt kurze und schnelle Programme, die auch in ein ROM gebracht werden können.
- Ausgabe in Z80-, 8080-, 8086-Assemblercode
- Kompatibel zu M80/L80 (MASM) von Microsoft
- Fehlerverfolgung mittels Trace möglich
- Umfangreiche Bibliothek incl. math. Funktionen
- für MS-DOS/CP/M 86: 4 Speichermodelle
- 8087 Math. Prozessor Unterstützung enthalten
- AMD 9511 Unterstützung erhältlich
- Unix-kompatibel
- Deutsche oder englische Version lieferbar
- 8"-/5,25"-/3,5"-/3"-Disk + deutsches Handbuch

MI-C für CP/M 445,— DM

MI-C für CP/M 86, MS-DOS 575,— DM

MI-C Crosscompiler (Ziel Z80/8080) 745,— DM

MI-C Crossassembler + Linker 645,— DM

MI-C Crosscompiler/Assembler (Ziel 8051) 1 495,— DM

MI-C Crossassembler (Ziel 68000) 795,— DM

MI-C AMD 9511 Unterstützung 798,— DM

Herbert Rose EDV, Bogenstraße 32, 4390 Gladbeck, Telefon (0 20 43) 2 49 12 oder 4 35 97
Vertrieb in Österreich:
Dr. Willibald Kraml, Microcomputer-Software, Degengasse 27/16, A-1160 Wien

Der MS-DOS-Interrupt 21

Dem MS-DOS-Interrupt 21 gebührt eine besondere Beachtung, denn über diesen Interrupt lassen sich fast alle Funktionen aufrufen, die MS-DOS bietet.

Die Funktionen des MS-DOS-Interrupt 21 unterteilen sich in CP/M- und Xenix-kompatible Funktionen. Die CP/M-kompatiblen Funktionen lassen sich über die AH-Werte 00 bis 37 aufrufen, und die Xenix-kompatiblen Funktionen folgen ab 38.

Das Konzept der Xenix-kompatiblen Funktionen ist moderner und wird von Microsoft, dem Hersteller von MS-DOS, fortgeführt, während die CP/M-kompatiblen Funktionen lediglich aus Kompatibilitätsgründen zu älteren MS-DOS-Versionen beibehalten werden.

Der wesentliche Unterschied zwischen den beiden Funktionsgruppen besteht darin, daß bei den CP/M-kompatiblen Funktionen der Dateizugriff über einen Dateisteuerblock (FCB) erfolgt, während bei den Xenix-kompatiblen Funktionen der Handle (ein Zugriffskanal für Dateien und Peripheriegeräte) den Zugriff auf eine Datei steuert.

Auch die Fehlermeldungen werden bei den CP/M- und Xenix-kompatiblen Funktionen unterschiedlich gehandhabt. Während bei den CP/M-kompatiblen Funktionen das

Register AL durch den Wert FF das Auftreten eines Fehlers anzeigt, signalisiert bei den Xenix-kompatiblen Funktionen das Übertrag-Statusbit C einen Fehler und der Inhalt des AL-Registers gibt eine Nummer an, die die Art des Fehlers repräsentiert.

Tabelle 1 zeigt alle CP/M-kompatiblen Funktionen und die wichtigsten Xenix-kompatiblen Funktionen des MS-DOS-Interrupt 21. Die Fehlermeldungen der Xenix-kompatiblen Funktionen sind in Tabelle 2 aufgeführt und Tabelle 3 zeigt, welche CP/M-kompatiblen Funktionen durch Xenix-kompatible Funktionen ersetzt werden können. Tabelle 4 gibt schließlich Einblick in den Aufbau der MS-DOS-Parameter und Parameterblöcke (wie DTA, FCB und PSP), die in Tabelle 1 verwendet werden.

(ma)

Die Funktionen des MS-DOS-Interrupt 21

AH-Register	Eingabe	Ausgabe	Bedeutung
CP/M-kompatible Funktionsaufrufe			
00	CS=Segmentadresse des PSP	--	Beende Programm und setze Interrupt-Vektoren im PSP zurück
01	--	AL=ASCII-Zeichen	Lese ASCII-Zeichen von CON (Tastatur) mit Ausgabe auf den Bildschirm
02	DL=ASCII-Zeichen	--	Gebe ASCII-Zeichen ab Cursorposition auf CON (Bildschirm) aus
03	--	AL=ASCII-Zeichen	Lese ASCII-Zeichen von AUX (serielle Schnittstelle) ein
04	DL=ASCII-Zeichen	--	Gebe ASCII-Zeichen auf AUX (serielle Schnittstelle) aus
05	DL=ASCII-Zeichen	--	Gebe ASCII-Zeichen auf PRN (Drucker) aus
06	DL=FF: Lese ASCII-Zeichen	AL=ASCII-Zeichen (wenn gelesen) Z=1: kein Zeichen gelesen	Lese ASCII-Zeichen von CON (Tastatur)
06	DL=ASCII-Zeichen	--	Schreibe ASCII-Zeichen auf CON (Bildschirm)
07	--	AL=ASCII-Zeichen	Lese ASCII-Zeichen von CON (Tastatur), und erkenne [Ctrl-C] nicht
08	--	AL=ASCII-Zeichen	Lese ASCII-Zeichen von CON (Tastatur), und erkenne [Ctrl-C]
09	DS:DX=Zeiger auf Zeichenkette (\$=Ende)	--	Gebe Zeichenkette ab Cursorposition auf CON (Bildschirm) aus
0A	DS:DX=Zeiger auf Eingabepuffer (DS:DX)=Maximal erlaubte Zeichenzahl	(DS:DX+1)=Tatsächliche Zeichenzahl (DS:DX+2)=Erstes Zeichen	Lese Zeile von CON (Tastatur) als Zeichenkette ein
0B	--	AL=0: Keine Taste AL=FF: Taste gedrückt	Überprüfe, ob Taste gedrückt wurde
0C	AL=1/6/7/8/A: Lösche Puffer und führe in AL angegebene Funktion aus AL=Anderer Wert: Nur löschen	AL=0: Puffer nur gelöscht	Lösche Tastaturpuffer [und führe weitere Funktion aus]
0D	--	--	Setze Standardlaufwerk zurück
0E	DL=0: Laufwerk A DL=1: Laufwerk B, und so weiter	AL=Anzahl der Laufwerke	Wähle Standardlaufwerk aus
0F	DS:DX=Ungeöffneter FCB	AL=0: Datei gefunden AL=FF: Datei nicht gefunden	Öffne Datei über FCB
10	DS:DX=Geöffneter FCB	AL=0: Datei gefunden AL=FF: Fehler aufgetreten	Schließe Datei über FCB
11	DS:DX=Ungeöffneter FCB	AL=0: Datei gefunden AL=FF: Datei nicht gefunden	Suche erste passende Datei über FCB
12	DS:DX=Ungeöffneter FCB	AL=0: Datei gefunden AL=FF: Datei nicht gefunden	Suche nächste passende Datei über FCB
13	DS:DX=Ungeöffneter FCB	AL=0: Datei gefunden AL=FF: Datei nicht gefunden	Lösche Datei über FCB
14	DS:DX=Geöffneter FCB	AL=0: Kein Fehler AL=1: EOF erreicht, keine Daten gelesen AL=2: DTA zu klein, keine Daten gelesen AL=3: EOF neu gesetzt, weil altes EOF nicht korrekt	Lese Datensatz aus sequentieller Datei

Die Funktionen des MS-DOS-Interrupt 21

AH-Register	Eingabe	Ausgabe	Bedeutung
15	DS:DX=Geöffneter FCB	AL=0: Kein Fehler AL=1: Diskette voll, keine Daten geschrieben AL=2: DTA zu klein, keine Daten geschrieben	Schreibe Datensatz in sequentielle Datei
16	DS:DX=Ungeöffneter FCB	AL=0: Datei angelegt AL=FF: Datei nicht angelegt, weil Verzeichnis voll	Lege Datei über FCB an
17	DS:DX=Modifizierter FCB	AL=0: Datei gefunden AL=FF: Datei nicht gefunden/Datei existiert bereits	Benenne Datei über FCB um
18	--	--	Für MS-DOS reserviert
19	--	AL=0: Laufwerk A AL=1: Laufwerk B, und so weiter	Bestimme Standardlaufwerk
1A	DS:DX=Zeiger auf neuen DTA	--	Definiere neuen DTA
1B	--	AL=Sektoren pro Datengruppe CX=Sektorgroße in Bytes DX=Datengruppen pro Laufwerk DS:BX=Zeiger auf ID der FAT	Lese Daten des Standardlaufwerks
1C	DL=0: Standardlaufwerk, DL=1: Laufwerk A, DL=2: Laufwerk B und so weiter	AL=Sektoren pro Datengruppe CX=Sektorgroße in Bytes DX=Datengruppen pro Laufwerk DS:BX=Zeiger auf ID der FAT	Lese Daten eines Laufwerks
1E : :	--	--	Für MS-DOS reserviert : :
20	--	--	Für MS-DOS reserviert
21	DS:DX=Geöffneter FCB	AL=0: Kein Fehler AL=1: EOF erreicht, keine Daten gelesen AL=2: DTA zu klein, keine Daten gelesen AL=3: EOF neu gesetzt, weil altes EOF nicht korrekt	Lese Datensatz wahlfrei
22	DS:DX=Geöffneter FCB	AL=0: Kein Fehler AL=1: Diskette voll, keine Daten geschrieben AL=2: DTA zu klein, keine Daten geschrieben	Schreibe Datensatz wahlfrei
23	DS:DX=Ungeöffneter FCB	AL=0: Datei gefunden AL=FF: Datei nicht gefunden	Ermittle Dateigröße
24	DS:DX=Geöffneter FCB	--	Setze RRF neu
25	AL=Interruptnummer DS:DX=Interruptvektor	--	Setze Interruptvektor neu
26	DX=Segmentadresse des PSP	--	Lege neuen PSP an
27	CX=Anzahl der Datensätze DS:DX=Geöffneter FCB	AL=0: Kein Fehler AL=1: EOF erreicht, keine Daten gelesen AL=2: DTA zu klein, keine Daten gelesen AL=3: EOF neu gesetzt, weil altes EOF nicht korrekt CX=Anzahl der gelesenen Datensätze	Lese mehrere Datensätze einer Datei
28	CX=Anzahl der Datensätze DS:DX=Geöffneter FCB	AL=0: Kein Fehler AL=1: Diskette voll, keine Daten geschrieben AL=2: DTA zu klein, keine Daten geschrieben	Schreibe mehrere Datensätze in eine Datei
29	AL=Suchcode ES:DI=Ungeöffneter FCB DS:SI=Zeiger auf Befehlszeile	AL=0: Keine Wildcards AL=1: Wildcards enthalten AL=FF: Laufwerk ungültig DS:SI=Zeiger auf unbearbeiteten Teil der Befehlszeile ES:DI=Ungeöffneter FCB	Isoliere Dateinamen aus Befehlszeile und reserviere FCB
2A	--	AL=Wochentag, CX=Jahr DH=Monat, DL=Tag	Lese Datum
2B	CX=Jahr, DH=Monat, DL=Tag	AL=0: Gültiges Datum AL=FF: Ungültiges Datum	Setze Datum

Die Funktionen des MS-DOS-Interrupt 21

AH-Register	Eingabe	Ausgabe	Bedeutung
2C	--	CH=Stunden, CL=Minuten DH=Sekunden DL=1/100 Sekunden	Lese Uhrzeit
2D	CH=Stunden, CL=Minuten DH=Sekunden DL=1/100 Sekunden	AL=0: Gültige Uhrzeit AL=FF: Ungültige Uhrzeit	Setze Uhrzeit
2E	AL=0: Lösche Statusbit AL=1: Setze Statusbit	--	Setze/lösche Verify-Statusbit
2F	--	ES:BX=Zeiger auf DTA	Lese DTA-Adresse
30	--	AX=Version (z.B. 0302=3.2) BH=OEM-Seriennummer BL: CX=24-Bit-Seriennummer	Lese MS-DOS-Version und Seriennummer
31	AL=Fehlercode DX=Speicherbedarf in 16-Byte-Einheiten	--	Beende Programm und behalte es für Neustart im Arbeitsspeicher
32	--	--	Für MS-DOS reserviert
33	AL=0: Lese Break-Statusbit	AL=FF: Fehler aufgetreten DL=0: Statusbit gelöscht DL=1: Statusbit gesetzt	Lese Break-Statusbit ([Ctrl-C])
33	AL=1: Ändere Break-Statusbit DL=0: Lösche Break-Statusbit DL=1: Setze Break-Statusbit	AL=FF: Fehler aufgetreten	Ändere Break-Statusbit ([Ctrl-C])
34	--	--	Für MS-DOS reserviert
35	AL=Interruptnummer	ES:BX=Interruptvektor	Lese Interruptvektor
36	DL=0: Standardlaufwerk DL=1: Laufwerk A, DL=2: Laufwerk B, u.s.w.	AX=FFFF (Laufwerk ungültig) AX=Sektoren pro Datengruppe BX=verfügbare Datengruppen CX=Sektorgröße in Bytes DX=Datengruppen pro Laufwerk	Ermittle freien Diskettenspeicher
37	--	--	Für MS-DOS reserviert
Xenix-kompatible Funktionsaufrufe (Carry=1: Fehler aufgetreten)			
38	AL=0: Voreingestellte Landes- kennzahl verwenden AL=FF: BX=Landeskennzahl-255 sonst: AL=Landeskennzahl DS:DX=Pufferbeginn für gewünschte Informationen	BX=Landeskennzahl Wenn C=1: AL=2	Lese Landeskennzahl
38	DX=FFFF AL=FF: BX=Landeskennzahl-255 sonst: AL=Landeskennzahl	C=1: AL=2	Setze Landeskennzahl
39	DS:DX=Zeiger auf Pfadnamen	C=1: AL=3/5	Lege neues Verzeichnis an
3A	DS:DX=Zeiger auf Pfadnamen	C=1: AL=3/5/10	Lösche leeres Verzeichnis
3B	DS:DX=Zeiger auf Pfadnamen	C=1: AL=3	Wechsele aktuelles Verzeichnis
3C	CX=Dateiattribut DS:DX=Zeiger auf Dateieintrag	AX=Handle C=1: AL=3/4/5	Lege Handle für neue Datei an
3D	AL=0: Lesen AL=1: Schreiben AL=2: Lesen und schreiben DS:DX=Zeiger auf Pfadnamen	AX=Handle C=1: AL=2/4/5/C	Öffne Handle auf Datei
3E	BX=Handle	C=1: AL=6	Schließe Handle auf Datei
3F	BX=Handle CX=Anzahl der zu lesenden Zeichen DS:DX=Zeiger auf Eingabepuffer	AX=Anzahl der gelesenen Zeichen AX=0: EOF erreicht C=1: AL=5/6	Lese über Handle aus Datei oder von Gerät
40	BX=Handle CX=Zahl der zu schreibenden Zeichen DS:DX=Zeiger auf Ausgabepuffer	AX=Anzahl der geschriebenen Zeichen AX=0: Diskette voll C=1: AL=5/6	Schreibe über Handle in Datei oder auf Gerät
41	DS:DX=Zeiger auf Dateieintrag	C=1: AL=2/5	Lösche Datei
42	AL=0: Dateianfang + Offset AL=1: Alte Zeigerposition + Offset AL=2: Dateiarbeit + Offset BX=Handle CX:DX=Offset	DX:AX=Neue Zeigerposition C=1: AL=1/6	Positioniere Schreib-/Lesezeiger innerhalb einer Datei auf die durch den Wert von AL definierte Art
43	AL=0: Lese Dateiattribut DS:DX=Zeiger auf Dateieintrag	CX=Dateiattribut C=1: AL=1/3/5	Lese Dateiattribut
43	AL=1: Ändere Dateiattribut CX=Dateiattribut DS:DX=Zeiger auf Dateieintrag	C=1: AL=1/3/5	Ändere Dateiattribut

Die Funktionen des MS-DOS-Interrupt 21

AH-Register	Eingabe	Ausgabe	Bedeutung
44	AL=Unterfunktion, weitere Register: Je nach Funktion	Register: Je nach Funktion C=1: AL=1/5/6/D/F	Ändere/lese Geräte-Informationen, gerätespezifische Ein-/Ausgabe (IOCTL-Funktion)
44.00	BX=Handle	DX=Kanalcode C=1: AL=1/6	Lese Kanalinformationen
44.01	BX=Handle DX=Kanalcode	C=1: AL=1/6	Schreibe Kanalinformationen
44.02	BX=Handle CX=Anzahl der zu sendenden Zeichen DS:DX=Zeiger auf Sendepuffer	AX=Anzahl der gesendeten Zeichen C=1: AL=1/6	Sende Steuerzeichen an zeichenorientiertes Gerät
44.03	BX=Handle CX=Anzahl der zu empfangenden Zeichen DS:DX=Zeiger auf Empfangspuffer	AX=Anzahl der empfangenen Zeichen C=1: AL=1/6	Empfange Steuerzeichen von zeichenorientiertem Gerät
44.04	BL=0: Angewähltes Laufwerk BL=1: Laufwerk A BL=2: Laufwerk B, usw. CX=Anzahl der zu sendenden Zeichen DS:DX=Zeiger auf Sendepuffer	AX=Anzahl der gesendeten Zeichen C=1: AL=1/5	Sende Steuerzeichen an blockorientiertes Gerät
44.05	BL=0: Angewähltes Laufwerk BL=1: Laufwerk A, BL=2: Laufwerk B, usw. CX=Anzahl der zu empfangenden Zeichen DS:DX=Zeiger auf Empfangspuffer	AX=Anzahl der empfangenen Zeichen C=1: AL=1/5	Empfange Steuerzeichen von blockorientiertem Gerät
44.06	BX=Handle	AL=0: Gerät nicht empfangsbereit/ EOF der Datei erreicht AL=FF: Gerät empfangsbereit/ Datei kann beschrieben werden C=1: AL=1/5/6/D	Prüfe Empfangsbereitschaft der Datei/des Gerätes
44.07	BX=Handle	AL=0: Gerät nicht sendebereit/ Datei kann gelesen werden AL=FF: Gerät sendebereit/ Datei kann gelesen werden C=1: AL=1/5/6/D	Prüfe Sendebereitschaft der Datei/des Gerätes
44.08	BL=0: Angewähltes Laufwerk BL=1: Laufwerk A, BL=2: Laufwerk B, usw.	AL=0: Medium ist austauschbar AL=1: Medium ist nicht austauschbar (z.B. Festplatte) C=1: AL=1/F	Prüfe Austauschbarkeit des Mediums in einem bestimmten Laufwerk
45	BX=Alter Handle	AX=Neuer Handle C=1: AL=4/6	Öffne zweiten Handle für den Zugriff auf eine geöffnete Datei
46	BX=Erster Handle CX=Zweiter Handle	C=1: AL=4/6	Schließe eventuell vorhandenen zweiten Handle und öffne neuen, gleichberechtigten zweiten Handle für Zugriff auf eine geöffnete Datei
47	DL=0: Angewähltes Laufwerk DL=1: Laufwerk A DL=2: Laufwerk B, usw. DS:SI=Zeiger auf Lesebuffer (64 Byte)	DS:SI=Zeiger auf Bezeichnung C=1: AL=F	Lese Bezeichnung des aktuellen Verzeichnisses
48	BX=Speicherbedarf in 16-Byte-Einheiten	AX=Segmentadresse für Speicherbereich C=1: AL=7/8, BX=Größter freier Speicherbereich in 16-Byte-Einheiten	Reserviere Arbeitsspeicher
49	ES=Segmentadresse für Speicherbereich	C=1: AL=7/9	Gebe reservierten Arbeitsspeicher frei
4A	BX=Speicherbedarf in 16-Byte-Einheiten ES=Segmentadresse für Speicherbereich	C=1: AL=7/8/9, BX=Größe des Speicherbereichs	Ändere Größe des reservierten Arbeitsspeichers
4B	AL=0 DS:DX=Zeiger auf Dateieintrag ES:BX=Zeiger auf Parameterblock	C=1: AL=1/2/8/A/B	Lade und starte Programm
4B	AL=3 DS:DX=Zeiger auf Dateieintrag ES:BX=Zeiger auf Parameterblocks	C=1: AL=1/2/8/A	Lade Overlay-Datei

Die Funktionen des MS-DOS-Interrupt 21			
AH-Register	Eingabe	Ausgabe	Bedeutung
4C	AL=Statuscode des aufrufen- den Programms	--	Beende Programm, schlieÙe geöffn- ete Dateien und übergebe Status- meldung an aufrufendes Programm
4D	--	AL=Statuscode des beendeten Programms	Ermittle Statusmeldung des mit Funktion 4C beendeten Programms
4E	CX=Dateiattribut DS:DX=Zeiger auf Dateieintrag	C=1: AL=2/12	Suche zum Dateieintrag passende Datei und schreibe Daten in DTA
4F	--	C=1: AL=12	Suche nächste zum Dateieintrag passende Datei und schreibe Daten in DTA
50 : :	--	--	Für MS-DOS reserviert : :
53	--	--	Für MS-DOS reserviert
54	--	AL=0: Statusbit gelöscht AL=1: Statusbit gesetzt	Teste Verify-Statusbit
55	--	--	Für MS-DOS reserviert
56	DS:DX=Zeiger auf alten Dateieintrag ES:DI=Zeiger auf neuen Dateieintrag	C=1: AL=2/5/11	Benenne Datei um
57	AL=0 BX=Handle	CH=Uhrzeit (LSB) CL=Uhrzeit (MSB) DH=Datum (LSB) DL=Datum (MSB) C=1: AL=1/6	Lese Uhrzeit und Datum einer Datei
57	AL=1 BX=Handle CH=Uhrzeit (LSB) CL=Uhrzeit (MSB) DH=Datum (LSB) DL=Datum (MSB)	C=1: AL=1/6	Ändere Uhrzeit und Datum einer Datei
Alle unbeteiligten Register bleiben erhalten.			

Tabelle 1. Der MS-DOS-Interrupt 21 ist einer der wichtigsten Interrupts, weil über ihn fast alle MS-DOS-Funktionen aufgerufen werden können

Die Fehlermeldungen der Xenix-kompatiblen Funktionsaufrufe (Carry=1)	
AL	Fehlermeldung
01	Ungültiger Funktionsaufruf
02	Datei nicht gefunden
03	Pfad nicht gefunden
04	Zu viele Dateien geöffnet (kein Zugriffskanal mehr frei)
05	Unerlaubter Zugriff
06	Ungültiger Zugriffskanal
07	Speicher-Steuerblock zerstört
08	Freier Arbeitsspeicher zu klein
09	Ungültiger Speicherbereich
0A	Ungültige Betriebssystem-Umgebung
0B	Ungültiges Dateiformat
0C	Ungültige Zugriffsart
0D	Ungültige Daten
0E	Für MS-DOS reserviert
0F	Ungültige Laufwerksangabe
10	Verzeichnis kann nicht gelöscht werden
11	Ungültiger Laufwerkswechsel
12	Keine weiteren Dateien vorhanden

Tabelle 2. Die Fehlermeldungen der Xenix-kompatiblen Funktionen sind im AL-Register kodiert

Folgende Xenix-kompatiblen Funktionsaufrufe sollten anstelle der veralteten CP/M-kompatiblen Funktionen benutzt werden:	
Xenix-kompatible	statt CP/M-kompatible(r)
Funktion 4C	Funktion 00
Funktion 3D	Funktion 0F
Funktion 3E	Funktion 10
Funktion 4E	Funktion 11
Funktion 4F	Funktion 12
Funktion 41	Funktion 13
Funktion 3F	Funktion 14
Funktion 3D	Funktion 15
Funktion 3C, 5A und 5B	Funktion 16
Funktion 56	Funktion 17
Funktion 3F	Funktion 21
Funktion 40	Funktion 22
Funktion 42	Funktion 23
Funktion 42	Funktion 24
Funktion 4B.00	Funktion 26
Funktion 3F	Funktion 27
Funktion 40	Funktion 28
Funktion 4C	Interrupt 20
Funktion 31	Interrupt 27

Tabelle 3. Veraltete CP/M-kompatible Funktionen sollten aus Gründen der Aufwärtskompatibilität durch Xenix-kompatible Funktionen ersetzt werden

Die Parameter und Parameterblöcke des MS-DOS-Interrupts 21

Begriff	Bedeutung
Betriebssystem-Umgebung	Speicherbereich, in dem die Systemvariablen (zum Beispiel »comspec«, »path« und »prompt«) abgelegt sind
Dateiattribut	<p>Zeichen, das den Status einer Datei angibt</p> <p>Attribut= 0: Unbeschränkter Zugriff</p> <p>Attribut= 1: Datei kann nur gelesen werden (Schreibschutz-Statusbit)</p> <p>Attribut= 2: Verborgene Datei (Öffnen durch Funktion 0F nicht möglich)</p> <p>Attribut= 4: Systemdatei (Öffnen durch Funktion 0F nicht möglich)</p> <p>Attribut= 8: Hauptverzeichnis (Ändern durch Funktion 43 nicht möglich)</p> <p>Attribut=10: Unterverzeichnis (Ändern durch Funktion 43 nicht möglich)</p> <p>Attribut=20: Datei wurde beschrieben (Archiv-Statusbit)</p> <p>Attribut=40: Für MS-DOS reserviert</p> <p>Attribut=80: Für MS-DOS reserviert</p>
Dateieintrag	<p>Byte 00 bis 07: Dateiname (bei Bedarf mit Leerzeichen aufgefüllt)</p> <p>Byte 08 bis 0A: Dateityp (bei Bedarf mit Leerzeichen aufgefüllt)</p> <p>Byte 0B: Dateiattribut</p> <p>Byte 0C bis 15: Für MS-DOS reserviert</p> <p>Byte 16 und 17: Uhrzeit des letzten Schreibzugriffs</p> <p>Byte 18 und 19: Datum des letzten Schreibzugriffs</p> <p>Byte 1A und 1B: Erste belegte Datengruppe auf der Diskette/Festplatte</p> <p>Byte 1C und 1D: Dateigröße=(Wort in 1C und 1D)+(Wort in 1E und 1F)x10</p> <p>Byte 1E und 1F: Dateigröße=(Wort in 1E und 1F)x10+(Wort in 1C und 1D)</p>
Datengruppe	Gruppe von Sektoren
DTA	<p>Disketten-Übertragungsbereich (disk transfer area) für die Ein-/Ausgabe von Daten auf und von Diskette/Festplatte</p> <p>Byte 00 bis 14: Für MS-DOS reserviert</p> <p>Byte 15: Dateiattribut</p> <p>Byte 16 und 17: Uhrzeit des letzten Schreibzugriffs</p> <p>Byte 18 und 19: Datum des letzten Schreibzugriffs</p> <p>Byte 1A und 1B: Dateigröße=(Wort in 1A und 1B)+(Wort in 1C und 1D)x10</p> <p>Byte 1C und 1D: Dateigröße=(Wort in 1C und 1D)x10+(Wort in 1A und 1B)</p> <p>Byte 1E bis 2A: Dateiname</p> <p>Byte 2B bis 7F: Pufferbereich</p>
EOF	Ende der Datei (end of file)
FAT	Dateibelegungstabelle (file allocation table) für die Zuordnung der Dateien auf Diskette/Festplatte zu den Angaben im Inhaltsverzeichnis des Datenträgers
FCB	<p>Dateisteuerblock (file control block), der bei den CP/M-kompatiblen Funktionen den Zugriff auf eine Datei regelt</p> <p>Folgende Angaben sind nur in einem erweiterten FCB vorhanden:</p> <p>Byte -07=FF: Kennzeichen für einen erweiterten FCB (Datei mit speziellem Dateiattribut)</p> <p>Byte -06 bis -02: Für MS-DOS reserviert</p> <p>Byte -01: Dateiattribut</p> <p>Folgende Angaben sind in jedem FCB (auch ungeöffnet) enthalten:</p> <p>Byte 00: Nummer des Laufwerks (0=angewähltes Laufwerk, 1=Laufwerk A und so weiter)</p> <p>Byte 01 bis 08: Dateiname (bei Bedarf mit Leerzeichen aufgefüllt)</p> <p>Byte 09 bis 0B: Dateityp (bei Bedarf mit Leerzeichen aufgefüllt)</p> <p>Folgende Angaben sind (durch Funktion 0F) nur in einem geöffneten FCB vorhanden:</p> <p>Byte 0C und 0D: Aktuelle Datenblocknummer (Voreinstellung: 0)</p> <p>Byte 0E und 0F: Datenblockgröße (Voreinstellung: 80)</p> <p>Byte 10 und 11: Dateigröße (Register)</p> <p>Byte 12 und 13: Dateigröße (Segment)</p> <p>Byte 14 und 15: Datum des letzten Schreibzugriffs</p> <p>Bit 0 bis 4: Tag</p> <p>Bit 5 bis 8: Monat</p> <p>Bit 9 bis F: Jahr (0=1980, 1=1981, und so weiter)</p> <p>Byte 16 und 17: Uhrzeit des letzten Schreibzugriffs</p> <p>Bit 0 bis 4: Sekunden (1=2 Sekunden, 2=4 Sekunden, und so weiter)</p> <p>Bit 5 bis A: Minuten</p> <p>Bit B bis F: Stunden</p> <p>Byte 18 bis 1F: Für MS-DOS reserviert</p> <p>Byte 20: Aktueller Datensatz, sequentiell</p> <p>Byte 21 und 22: Aktueller Datensatz, wahlfrei (Register)</p> <p>Byte 23 und 24: Aktueller Datensatz, wahlfrei (Segment)</p> <p>Byte 25 bis 27: Nicht verwendet</p> <p>(Die Registerkombination DS:DX enthält nicht die Startadresse, sondern jeweils den Zeiger auf den Anfang eines FCB!)</p>
Handle	<p>16-Bit-Dateikennsatz zur Identifikation einer Datei bzw. Zugriffskanal auf eine Datei (file handle), ersetzt das veraltete FCB-Verfahren</p> <p>Folgende Handles sind bereits für Geräte vergeben:</p> <p>Handle=0: Standardeingabe CON (Tastatur)</p> <p>Handle=1: Standardausgabe CON (Bildschirm)</p> <p>Handle=2: Gerät, das für die Fehlerausgabe zuständig ist</p> <p>Handle=3: Ein-/Ausgabe AUX (serielle Schnittstelle)</p> <p>Handle=4: Druckerausgabe PRN (parallele Schnittstelle)</p>
ID	Identifikationsbyte (identify byte) eines Diskettensektors

Die Parameter und Parameterblöcke des MS-DOS-Interrupts 21

Begriff	Bedeutung
IOCTL	Kontrollfunktion (device I/O-control) für Peripheriegeräte
Kanalcode	Code für den Zugriff auf ein Gerät oder eine Datei Gerät: Bit 0=1: Gerät ist Standardeingabe (CON) Bit 1=1: Gerät ist Standardausgabe (CON) Bit 2=1: Gerät ist Pseudoeinheit (NUL) Bit 3=1: Gerät ist Systemuhr Bit 4: Für MS-DOS reserviert Bit 5=0: Control-Zeichen werden verarbeitet Bit 5=1: Control-Zeichen [Ctrl-C], [Ctrl-P], [Ctrl-S] und [Ctrl-Z] werden nicht verarbeitet ([Ctrl-C] muß zusätzlich über Break oder Funktion 33 deaktiviert werden) Bit 6=0: EOF erreicht Bit 7=1: Kanalcode spezifiziert Gerät Bit 8 bis D: Für MS-DOS reserviert Bit E=1: Gerät kann Steuerdaten empfangen und senden Bit F: Für MS-DOS reserviert Datei: Bit 0 bis 5: Angesprochenes Laufwerk (0=A, 1=B, und so weiter) Bit 6=0: Kanal wurde beschrieben Bit 7=0: Kanalcode spezifiziert Datei Bit 8 bis F: Für MS-DOS reserviert
LSB	Niederwertiger Teil einer 16-Bit-Adresse oder eines 16-Bit-Wertes (least significant byte)
MSB	Höherwertiger Teil einer 16-Bit-Adresse oder eines 16-Bit-Wertes (most significant byte)
Parameter	Laden und starten: Byte 00 und 01: Segmentadresse der Programmumgebung Byte 02 und 05: Segmentadresse und Adresse der Befehlszeile, die ab Byte 80 (DTA) des PSP übergeben werden soll Byte 06 und 09: Segmentadresse und Adresse eines ungeöffneten FCB, der ab Byte 5C im neuen PSP abgelegt werden soll (FCB 1) Byte 0A und 0D: Segmentadresse und Adresse eines ungeöffneten FCB, der ab Byte 6C im neuen PSP abgelegt werden soll Nur laden: Byte 00 und 01: Segmentadresse, ab der Programm geladen werden soll Byte 02 und 03: Relokationsfaktor (in der Regel wie Byte 0 und 1)
PSP	Programmsegmentvorspann (program segment prefix), der vom Betriebssystem ab Segmentadresse 0000hex vor jeder FCB-Datei im Arbeitsspeicher installiert wird und Informationen über die Behandlung der Datei enthält Dieser Bereich darf von keinem Anwenderprogramm verändert werden: Byte 00 und 01: Adresse des MS-DOS-Interrupt 20 (Programm beenden) Byte 02 und 03: Erstes freies Byte hinter der Datei Byte 04: Reserviert Byte 05: Intersegment-Sprungbefehl Byte 06 bis 09: Intersegmentadresse (IP:CS) zum Aufruf der MS-DOS-Funktionen 0 bis 24 (CL=Funktion) Byte 0A bis 0D: Intersegmentadresse (IP:CS), von der aus diese Datei aufgerufen wurde Byte 0E bis 11: Intersegmentadresse (IP:CS), die bei Auftreten von [Ctrl-C] aufgerufen wird Byte 12 bis 15: Intersegmentadresse (IP:CS), die bei Auftreten eines Lesefehlers aufgerufen wird Byte 16 bis 2B: Für MS-DOS reserviert Byte 2C und 2D: Segmentadresse der Betriebssystem-Umgebung Byte 2E bis 4F: Für MS-DOS reserviert Byte 50 und 51: Adresse des MS-DOS-Interrupt 21 (Aufruf einer MS-DOS-Funktion) Byte 52: RET (Intersegment) Byte 53 bis 5B: Für MS-DOS reserviert Dieser Bereich kann von einem Anwenderprogramm verändert werden: Byte 5C bis 67: Ungeöffneter FCB für den ersten im Dateiaufruf übergebenen Dateinamens Byte 68 bis 6B: Nicht verwendet Byte 6C bis 77: Ungeöffneter FCB des zweiten im Dateiaufruf übergebenen Dateinamens Byte 78 bis 7F: Nicht verwendet Byte 80 bis FF: DTA (Ein-/Ausgabepuffer für Disketten- und Festplatten-Operationen)
RRF	Nummer eines relativen Datensatzes (relative record field)
Statuscode	Statuscode=0: Programm korrekt beendet Statuscode=1: Programm durch [Ctrl-C] beendet Statuscode=2: Programm durch schweren Gerätefehler beendet Statuscode=3: Programm durch Funktion 31 beendet
Suchcode	Code zur Behandlung einer Dateibezeichnung, nach der gesucht werden soll Bit 0=0: Abbrechen bei Auftreten eines Separators (»« oder »\«) Bit 0=1: Separatoren vor Dateinamen ignorieren Bit 1=0: Wenn keine Laufwerksangabe vorhanden ist, »A« in FCB eintragen Bit 1=1: Wenn keine Laufwerksangabe vorhanden ist, FCB unverändert lassen Bit 2=0: Wenn kein Dateiname vorhanden ist, acht Leerzeichen in FCB eintragen Bit 2=1: Wenn kein Dateiname vorhanden ist, FCB unverändert lassen Bit 3=0: Wenn keine Dateitypangabe vorhanden ist, drei Leerzeichen in FCB eintragen Bit 3=1: Wenn keine Dateitypangabe vorhanden ist, FCB unverändert lassen

Tabelle 4. Die Parameter der MS-DOS-Funktionen mit Bedeutung und internem Aufbau

MS-DOS-Funktionen in der Praxis

Die Funktionen des MS-DOS-Interrupt 21 können dem Programmierer viel Arbeit abnehmen. Doch wie werden sie in Assemblerprogramme eingebunden?

Der MS-DOS-Interrupt 21 stellt dem Programmierer (fast) alle Funktionen zur Verfügung, die er für die Datenein- und -ausgabe, für den Zugriff auf Diskette und Festplatte sowie für die Dateiverwaltung benötigt. Dank der klaren Software-Interrupt-Technik des PC lassen sich diese Funktionen aus Assemblerprogrammen heraus auch sehr einfach aufrufen.

Der Assemblerbefehl »INT n« löst den Software-Interrupt mit der Nummer »n« aus. Deshalb bewirkt der Befehl »INT 21« den Aufruf des MS-DOS-Interrupt 21. Um nun eine ganz bestimmte Funktion des Interrupt 21 auszuwählen, muß sich die Nummer der gewünschten Funktion im AH-Register befinden. Erfordert die aufzurufende Funktion die Übergabe von Parametern, so müssen diese Parameter zuvor in die entsprechenden Register geladen werden. Möchten Sie beispielsweise mit der

MS-DOS-Funktion 02 ein Zeichen auf den Bildschirm ausgeben, so müssen Sie zuerst das Register DL mit dem ASCII-Code des auszugebenden Zeichens laden, darauf in das AH-Register den Wert 2 schreiben und zuletzt über »INT 21« den Interrupt und damit den Funktionsaufruf auslösen.

Umgekehrt kann eine MS-DOS-Funktion auch Parameter zurückgeben. Möchten Sie beispielsweise den Zustand des Verify-Statusbit testen, das aus der MS-DOS-Ebene mit »verify on« gesetzt und mit »verify off« zurückgesetzt wird, dann rufen Sie die MS-DOS-Funktion 54 auf. Diese Funktion gibt den Zustand des Verify-Statusbit im AL-Register zurück. Beträgt der Wert von AL 1, so ist auch das Verify-Statusbit gesetzt, und ist der Wert von AL gleich 0, dann liegt das Verify-Statusbit ebenfalls auf 0. Natürlich gibt es auch MS-DOS-Funktionen die sowohl

die Übergabe von Parametern erfordern, als auch nach Beendigung ihrer Aufgabe Parameter zurückgeben.

Getreu dem Motto »Ein Listing sagt mehr als tausend Worte« haben wir ein kleines Listing für Sie abgedruckt, das über die MS-DOS-Funktion 3D eine Datei öffnet, mit der Funktion 3F jeweils 128 Byte in einen String ausliest und diesen String über die Funktion 09 auf den Bildschirm ausgibt. Beendet wird das Programm durch die MS-DOS-Funktion 4C.

Selbstverständlich läßt sich das Programm noch erweitern. Der Dateiname kann zum Beispiel über die Tastatur abgefragt werden, die Fehlermeldung läßt sich durch Überprüfen des Fehlercodes im AL-Register präzisieren und die Ausgabe des Dateinhaltes könnte wahlweise über Bildschirm oder Drucker erfolgen.

Scheuen Sie sich nicht! Probieren Sie Ihre Ideen aus und Sie werden überrascht sein, wie schnell sich selbst in Assembler-Sprache mit Hilfe der MS-DOS-Funktionen komplette Programme entwickeln lassen.

(ma)

```

name      int_call

open_handle macro      path,access      ;Makro »open_handle« definieren
mov        DX,offset path ;Zeiger auf Pfad in DX schreiben
mov        AL,access      ;Zugriffscode in AL schreiben
mov        AH,3DH         ;MS-DOS-Funktion 3DH auswählen
int        21H            ;MS-DOS-Interrupt aufrufen
endm

read_handle macro      handle,buffer,bytes ;Makro »read_handle« definieren
mov        BX,handle      ;Handle in BX schreiben
mov        DX,offset buffer ;Zeiger auf Puffer in DX schreiben
mov        CX,bytes       ;Anzahl der Bytes in CX schreiben
mov        AH,3FH         ;MS-DOS-Funktion 3FH auswählen
int        21H            ;MS-DOS-Interrupt aufrufen
endm

display macro      string
mov        DX,offset string ;Zeiger auf String in DX schreiben
mov        AH,09H         ;MS-DOS-Funktion 09H auswählen
int        21H            ;MS-DOS-Interrupt aufrufen
endm

assume     cs:code,ds:data,es:nothing,ss:nothing

data       segment
filename   db      "b:text.doc",0 ;Dateinamen mit Pfad definieren
message    db      "Read error!$" ;Fehlermeldung definieren
buffer     db      129 dup(?)      ;129 Bytes in einem nichtinitialisierten
                                   ;Speicherbereich reservieren (129 x ?)

ds_old     dw      ?              ;Wort für altes Datensegment ...
handle     dw      ?              ;... und Wort für Handle reservieren
data       ends                  ;Datensegment-Ende

```

```

code       segment                      ;Programmsegment-Anfang
start:     push        DS              ;Programmstart
           mov         AX,data         ;DS auf Stapel legen
           mov         DS,AX          ;Datensegment
           pop         AX              ;in DS laden
           mov         ds_old,AX      ;altes DS in AX zurückholen ...
           open_handle filename,0     ;... und sichern
           jc          error          ;Datei zum Lesen öffnen
           mov         handle,AX      ;Datei-Routine aufrufen
           read_line:  read_handle    ;Handle sichern
           mov         handle,buffer,128 ;128 Bytes einlesen
           jc          error          ;bei Carry=1 Fehler-Routine aufrufen
           cmp         AX,0           ;Ende der Datei
           je          return         ;wenn ja, dann Ende
           mov         BX,AX          ;wenn nicht, dann Anzahl der
           mov         buffer[bx],"$";gelesenen Bytes in BX schreiben
           display     buffer         ;String im Puffer durch »$« beenden
           jmp         read_line      ;String auf den Bildschirm ausgeben
error:      display     message       ;die nächsten 128 Bytes einlesen
           ;Fehlermeldung anzeigen

return:     mov         DS,ds_old      ;Datensegment wiederherstellen
           mov         AH,4CH         ;MS-DOS-Funktion 4CH auswählen ...
           int         21H            ;... und Programm beenden

code       ends                        ;Programmsegment-Ende
start     ;Programmende

```

Listing. Dieses kleine Listing demonstriert, wie sich mit Hilfe der MS-DOS-Funktionen bei den Standardaufgaben eines Programmierers viel Arbeit sparen läßt

DIE VIER* vermissen Sie sicher schon lange..

RAMfrei

entfernt **selektiv** residente Programme (per Tastendruck) und liefert freien Speicher (auch EMS) für hungrige Software.

Nottaste

Rechner abgestürzt? Programm aufgehängt? Ein Tastendruck, und Sie können sicher weiterarbeiten! Ein Muß für Programmierer und RAM-Disk-User!

WS_PATH

erweitert Wordstars Horizont. Bearbeiten Sie Ihre Texte ab jetzt in **allen** angeschlossenen Laufwerken und Verzeichnissen!

Treiber

Universeller Druckertreiber. Einfache und flexible Zeichenkonvertierung für parallele und serielle Druckerausgabe.

DIE VIER ZUSAMMEN inkl. Anleitung nur DM

99,-

*VIER neue Utilities für DOS 2.xx und 3.xx für PC/XT/AT-Kompatible

Andreas Vehrenberg
Blücherstr. 42
4000 Düsseldorf 30

02 11/
44 60 18

Turbo-Tempo für GW-Basic

Leider sind im Befehlsumfang von GW-Basic keine Interrupt-Befehle enthalten, um die Betriebssystemroutinen des PC aufzurufen. Benutzen können Sie sie aber trotzdem und damit Sie Ihre Programme erheblich beschleunigen.

Das BIOS (Basic Input Output System) im PC und das Betriebssystem MS-DOS stellen dem Programmierer leistungsfähige Routinen für Tastatur-, Bildschirm-, Datei-, Datenträger- und Schnittstellenverwaltung zur Verfügung. Doch diese Routinen lassen sich nur in Assemblerprogrammen und einigen Hochsprachen direkt aufrufen, dem GW-Basic-Programmierer bleiben sie verschlossen.

Fast alle BIOS-Routinen sind Interruptroutinen oder Funktionen von Interruptroutinen, das heißt, sie lassen sich über Unterbrechungsanforderungen der Peripherie oder über die sogenannten Software-Interrupts 00 bis FFhex aufrufen. Das Arbeiten mit Hardware-Interrupts erhöht die Leistungsfähigkeit des PC, denn die Peripherie nimmt den Mikroprozessor im PC nur dann in Anspruch, wenn es wirklich notwendig ist. Ebenso hat das Konzept der Software-Interrupts wesentliche Vorteile. Wenn die Interruptroutinen nämlich mit den Routinen des Betriebssystems belegt sind, lassen sich die Betriebssystemfunktionen damit über einfache Interruptbefehle aufrufen.

Nur der GW-Basic-Programmierer hat hier das Nachsehen, weil unter GW-Basic keine Interruptbefehle existieren. Mit dem kleinen Maschinenspracheprogramm »interbas« sind Sie jedoch in der Lage, die Interruptroutinen und damit die Betriebssystemroutinen des PC zu nutzen. Diese neugewonnene Fähigkeit von GW-Basic eröffnet Ihnen Anwendungsgebiete, die weit über das übliche Einsatzgebiet des Basic-Interpreters hinausgehen.

Listing 1 zeigt »interbas« in seiner Rohfassung als Assemblerprogramm. Sie müssen jedoch keinen Assembler besitzen, um »interbas« benutzen zu können, denn in Listing 2 bekommen Sie »interbas« in ein Basic-Programm eingebunden. In diesem Fall liegt »interbas« in den Data-Zeilen als Maschinencode vor. GW-Basic schreibt

diesen Maschinencode beim Programmstart lediglich in den Arbeitsspeicher des PC und hält die Routine dort für den Aufruf durch den Call-Befehl bereit.

Sie müssen also die Basic-Zeilen von »interbas« lediglich in das Programm einbinden, aus dem Sie die Interruptroutinen des PC aufrufen möchten. Selbstverständlich können Sie die Zeilennummern Ihren eigenen Vorstellungen und Wünschen anpassen. Listing 3 demonstriert an einigen Beispielen die Arbeitsweise von »interbas«. Tippen Sie dieses Programm unbedingt ab, denn Sie lernen auf diese Weise sehr gut den praktischen Umgang mit »interbas« kennen.

Um Ihnen die Arbeitsweise von »interbas« noch zu verdeutlichen, im folgenden ein weiteres Beispiel zum Einsatz dieses Programms. Fügen Sie dazu die genannten Variablenzuweisungen als Hauptteil in das Listing 2 ein. Die Aufgabenstellung besteht darin, einen quadratischen Bildschirmbereich aufwärts zu scrollen. Für solche Bildschirmoperationen ist der Video-Interrupt 10, Funktion 6 zuständig.

Grundsätzlich muß »interbas« durch einen eigenen Speicherbereich (Anwenderstapel) unterstützt werden. Dieser Speicherbereich wird durch die Variable »stapel\$« repräsentiert. Die ersten einzufügenden Zeilen lauten:

```
100 nummer.des.interrupts=&H10
110 stapel$=space$(128)
```

Sie geben in diesem Fall als Interruptnummer den Wert 10hex an (es hat sich eingebürgert, die PC-Interrupts mit hexadezimalen Ziffern zu numerieren) und reservieren großzügig 128 Byte für den Anwenderstapel (entspricht 64 16-Bit-Adressen).

Als nächstes müssen Sie die relevanten Register mit den Übergabeparametern belegen. Die Werte der übrigen Register sind gleichgültig. Der Video-Interrupt 10, Funktion 6, benötigt Parameter in den Registern AH, AL, BH, CH, CL, DH und DL. AH enthält die Funktionsnummer, also den Wert 6, AL gibt die Anzahl der Zeilen

an, die gescrollt werden sollen, CH und CL enthalten die Koordinate der linken oberen, sowie DH und DL die Koordinate der rechten unteren Ecke.

Schließlich müssen Sie in BH noch das Attribut für die gelöschten Zeilen angeben.

Folgende Werte scrollen das Fenster innerhalb der Koordinaten 2/2 und 20/20 um fünf Zeilen aufwärts und weisen den dadurch freiwerdenden Zeilen das Attribut Null zu:

```
120 AH=6
130 AL=5
140 CH=2
150 CL=2
160 DH=20
170 DL=20
180 BH=0
```

Als letzten Parameter müssen Sie einen Rückgabeparameter anführen, der jedoch nur wichtig ist, wenn die aufgerufene Routine einen Fehlercode zurückgibt.

Wenn Sie in diesem Fall

```
190 rueckgabeparameter=255
```

in das Programm einfügen, dann ist nach dem Aufruf der Routine immer dann kein Fehler aufgetreten, wenn die Variable »rueckgabeparameter« weiterhin den Wert 255 enthält. Hat sich der Wert dagegen verändert, so ist während des Ablaufs der Routine ein Fehler aufgetreten.

Haben Sie das Hauptprogramm mit den aufgeführten Variablenzuweisungen versorgt, so erfolgt der Aufruf der Betriebssystemroutine durch

```
200 GOSUB 10000
```

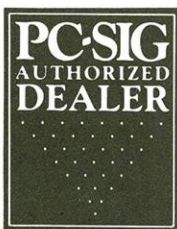
Vergessen Sie danach nicht, Ihr Programm ordnungsgemäß mit

```
210 END
```

abzuschließen.

Und nun Start frei für Ihre eigenen Ideen. Die einzelnen Beiträge in dieser Ausgabe, die sich den verschiedenen Interrupts des PC widmen, geben Ihnen genügend Hinweise zu eigenen Experimenten. Sie wissen ja: Probieren geht über Studieren!

(Martin Althaus/
Markus Zietlow/ma)



Kirschbaum Software GmbH
Kronau 15, D-8091 Emmering
Tel.: (08067) 1220, FAX (08067) 1053

QMODEM deluxe V2.3, Stock Charting V3.5, PC-Stock V1.4, Kermit V2.29B CHASM V4.09, Blackboard V7.25, Procomm V2.4.2, Screen-Writer V4, Still River Shell V1.82, RBBS 14.1A, ACE 1.02, HDML V4.4, newBase V3.49, MailMonster V2.4 SoftTouch V2.03, PC-TYPE+, PC-Prompt 3.0, PC-VT V8.4, PC-KEY Draw V3.24, Homeware V3.2, New YorkWord V2.2, Packdisk V1.3, FAMILY TREE V3.02, Sidewriter V 3.01, PC-File+

Preis je PC-SIG Diskette zwischen DM 10,— und DM 16,—

PC-SIG bietet Ihnen 800 unterschiedliche Programmdisketten mit über 10000 Programmen für IBM und kompatible PC's. Jeden Monat ca. 20 neue Disketten und mind. 20 Updates zu früheren Disketten. Wir liefern immer neueste Originaldisketten auch im 3 1/2" Format.

Werden Sie jetzt PC-SIG Mitglied! Keine Kaufverpflichtung, aber diese Leistungen:

- aktuelle Beschreibung der gesamten PC-SIG Library 1-705 gedruckt auf 430 Seiten, Rest auf Diskette
- zweimonatliche Zusendung des PC-SIG NEWS MAGAZINE u. des PC-SIG HOT SHEET
- ab Bestellung von 5 Disketten ein Mitgliederrabatt von DM 2,— je Diskette
- Sonderaktionen nur für PC-SIG Mitglieder

Schicken Sie noch heute einen Scheck über DM 68,— für Ihre einjährige Mitgliedschaft!

Haben Sie eine HERCULES-Karte?

... dann haben Sie auch viele Programme, welche bisher auf Ihrem Rechner nicht liefen. Sie brauchen multigraph II! Bestellen Sie dieses Programm noch heute! multigraph II emuliert folgende Grafikauflösungen auf einer HERCULES Monochrom-Grafikkarte oder Kompatiblen:

- 300 x 200 Punkte 3 Graustufen CGA kompatibel
- 640 x 200 Punkte monochrom CGA kompatibel
- 640 x 400 Punkte monochrom Olivetti kompatibel

(für die Olivetti-Emulation brauchen Sie einen guten Monitor)
Mehr als 90% der Software, welche für Grafik eigentlich die CGA-Karte benötigt, ist jetzt auf Rechnern mit HERCULES-Karte lauffähig. Die Programme laufen genauso schnell. Auch Programme mit eigenem Betriebssystem (Nightmission Flipper,...) laufen. Endlich auch Grafik mit GW-Basic auf Rechnern mit HERCULES-Karte möglich.

multigraph II (Programmdiskette und 20-seitige Beschreibung)

DM 112,—

Information und Bestellungen:

Kirschbaum Software GmbH
Kronau 15, D-8091 Emmering
Tel. 08067/1220, Fax 08067/1053

Händleranfragen erwünscht

Schweiz:

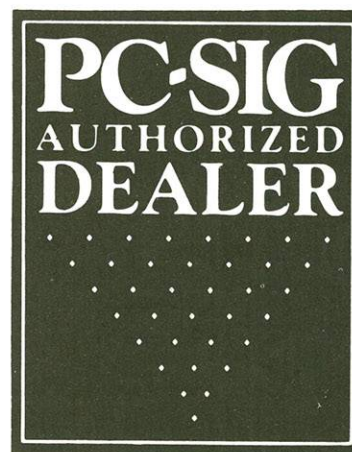
DSE Datasystems-Engineering AG
Badener Str. 262, CH-8004 Zürich
Österreich:
Überreuter Media
Handels- und Verlagsges. mbH
Alser Straße 24, A-1091 Wien

PC-SIG

PUBLIC DOMAIN und SHAREWARE-PROGRAMME aktuell aus USA

jeden Monat ca. 20 neue Disketten und zusätzlich ca. 20 Updates zu älteren PC-SIG Nummern. Bitte unterstützen Sie die Arbeit von PC-SIG in USA. Kaufen Sie auch zu Ihrer Sicherheit nur die grauen Original-PC-SIG Disketten vom autorisierten Händler. Bedenken Sie die folgenden Leistungsmerkmale:

- Erstklassiger Service durch ausgewählte Händler
- Originaldisketteninhalt, da Originaldisketten aus 1. Hand
- immer neueste Programmversionen
- Neuheiten genauso schnell verfügbar wie in den USA
- Rücknahmegarantie bei Disketten- und Softwarefehlern
- immer aktuelle Information über Sharewareneuheiten in den USA
- Sonderleistungen nur für PC-SIG Mitglieder (z.B. Sonderpreise für multigraph II DM 85,—, PC-Write V2.71 mit Hardcover-Manual DM 168,—, PC-File+ in der europäischen Version DM 168,—, registriertes Powermenu mit Support DM 178,—, Homepage mit Support DM 178,—, Automenu DM 98,—, INTUIT IS-2000 DM 128,—).



Künstliche Intelligenz

- 148 XLISP Version 1.7
- 417 A.D.A. PROLOG v1.91p
- 761 IMP Shell

Kommunikationssoftware

- 41 42 Kermit Version 2.29B
- 310 QMODEM Communications Version 2.3
- 499 ProComm Version 2.4.2 **Bestseller**

Datenbankprogramme

- 5 730 PC-FILE+ superschnelle, relationale Datenbank
- 287 288 File Express V3.78
- 599 600 601 Datenbanksystem für gehobene Ansprüche

Utilities der Extraklasse

- 124 Extended Batch Language V3.05A by Seaware
- 356 650 Fansi-Console V2.00
- 405 PC-DESKTEAM V1.04 speicherresidentes Hintergrundprogramm mit Wecker, Kalender, Taschenrechner, Notizblock, DOS-Kommandos und Schreibmaschinenfunktion
- 481 Still River Shell V1.82 — Benutzeroberfläche zu DOS
- 498 DOSAMATIC - Laden Sie mehrere Programme gleichzeitig
- 535 COMPUTER USER'S HANDBAG, u.a. neueste Vers. v. CED
- 598 DISK TOOL (Masterkey V1.6c mehr als ein Diskettendat.)
- 608 AUTOMENU V4.0 Spitzenmenüprogramm in Assembler
- 610 PACKDISK V1.3 Harddiskoptimierer
- 611 BlackBeard V7.25 - leistungsstarker Dateieditor **Bestseller** geeignet für Texte mit mehr als 1 MB, bis zu 254 Zeichen/Zeile, einfache Bedienung durch Hilfschirme
- 661 RESICALC speicherresidenter wissenschaftlicher Rechner
- 727 PowerMenu - mehr als nur ein Menüsystem
- 728 729 Homepage V2.0 Desktop Organizer **Bestseller**

Lernsoftware

- 612 Languages - Vokabeln und Redewendungen
- 816 817 Turbo C Tutorial

Spiele

- 120 PC-CHESS - unser bestes Schachprogramm
- 274 BEST GAMES - Packman, Mondlandung usw.
- 445 Willy the worm - programmieren Sie Ihr eigenes Spiel!
- 457 Greatest Arcade Games - Striker, Spacewar und mehr
- 604 Landing Party - Adventure Game
- 723 Super pinball - fünf Flipper auf einer Diskette
- 740 EasyMenu (GAMES) - Chess, Brick, usw. **anschauen!!!**

Grafikprogramme

- 344 345 PC-KEY Draw - einfaches CAD-Programm
- 762 Image 3D - dreidimensionale Konstruktionen
- 763 Fingerpaint - erstklassiges Zeichenprogramm!

Programmiersprachen

- 10 CHASM V4.09, einfacher Assembler mit Beispielen
- 510 VISIBLE-PASCAL - brauchbarer Pascal-Compiler
- 527 B-WINDOW TOOLBOX schnelle Windows für Basic u. C

Tabellenkalkulation

- 199 PC-CALC - vom Altmeister Jim Button **Bestseller**
- 695 EZ-SPREADSHEET - besonders für den Einsteiger
- 696 QubeCalc - echt dreidimensional 64*64*64

Textverarbeitung

- 78 PC-WRITE Version 2.4 **Bestseller**
- 455 681 682 PC-TYPE+ von Jim Button
- 528 NEW YORK WORD V2.2 - fantastische Trennautomatik
- 765 GALAXY, einfach und superschnell

Religion

- 766 - 772 DIE BIBEL, englisch auf insgesamt 7 Disketten

PC-SIG Distributor in der Schweiz:

DSE Datasystems-Engineering AG
Badener Straße 262, CH-8004 Zürich
Tel. 01-2418882, Telex 813950

PC-SIG Distributor in Österreich:

Überreuter Media Handels- und Verlagsges. mbH
Alser Straße 24, A-1091 Wien
Tel. 0222-481538-0, FAX 0222-481538-337

PUBLIC DOMAIN SOFTWARE und SHAREWARE aus der PC-SIG Library

☐ PC-SIG Mitgliedschaft (ohne Kaufverpflichtung!!) **DM 68,—**

- aktuelle Beschreibung der gesamten PC-SIG Library 1-705 gedruckt auf 430 Seiten, Rest auf Diskette
- zweimonatliche Zusendung des PC-SIG NEWS MAGAZINES
- ab 5 Disketten Mitgliedsrabatt von DM 2,— pro Disk
- Sonderverkaufsaktionen nur für PC-SIG Mitglieder

Schicken Sie noch heute den Scheck über DM 68,— für Ihre einjährige PC-SIG Mitgliedschaft oder wählen Sie:

☐ PC-SIG Mitgliedschaft + 4 PC-SIG Disketten Ihrer Wahl **DM 98,—**

- Mitgliedschaft wie oben beschrieben und vier beliebige Programmdisketten aus der PC-SIG Library
- Zögern Sie nicht, überzeugen Sie sich durch diesen Test!

☐ 5 Stck. Katalog-Disketten mit kompl. Beschreibung **je DM 10,—**

☐ Komplette Beschreibung der PC-SIG Disketten 1-705

- gedruckt als Buch mit ca. 430 Seiten, gegliedert nach Sachgebieten, **sehr empfehlenswert**

DM 28,—

Wenn Sie sich für PUBLIC DOMAIN und SHAREWARE

näher interessieren, so gehört dieses Buch zur Grund-

ausstattung!

Information und Bestellungen

innerhalb Deutschland:

Kirschbaum Software GmbH

Kronau 15, D-8091 Emmering

Tel. 08067/1220 FAX 08067/1053

Bitte beachten Sie, daß auf allen Disketten aus der PC-SIG Library sowohl Programme wie auch Textdateien nur englischsprachig sind. 5 1/4" Diskettenpreise für Nichtmitglieder: 1-9 Disketten je DM 16,—/ 10-24 je DM 15,—/ 25-49 je DM 14,—, für 3 1/2" Aufpreis DM 4,— je Disk. Wir verrechnen DM 14,— Versandkostenanteil bei Bestellungen bis DM 150,—.

Bitte haben Sie Verständnis, daß wir an Privatpersonen nur noch gegen Nachnahme oder Vorauskasse liefern.


```

0:      ; Assemblerprogramm für den Interruptaufruf aus GW-Basic
      heraus
1:      ;
2:      ; Aufrufform in Basic:
3:      ; CALL INTERRUPT(AH,AL,BH,BL,CH,CL,DH,DL,Interruptnummer,
      Rückgabeparameter,Stapel$)
4:      ;
5:      cseg      segment      para public 'code'
6:      public    intrrpt
7:      intrrpt   proc          far
8:      assume    cs:cseg,ds:nothing,ss:nothing,es:nothing
9:      push      bp            ;Basiszeiger sichern und
      auf die
10:     mov       bp,sp         ; Übergabeparameter richt
      en
11:     mov       di,sp         ;Basic-Stapel übernehmen
12:     mov       si,[bp+6]     ;Neuen
13:     mov       al,[si]       ; Stapel
14:     mov       ah,0          ; einlesen
15:     inc       si            ; und für
16:     add       ax,[si]       ; den
17:     dec       ax            ; Interrupt-Aufruf
18:     mov       sp,ax         ; vorbereiten
19:     push      di            ;Basic-Stapel sichern
20:     mov       ax,0          ;Den Befehlszeiger, der a
      uf die
21:     mov       ds,ax         ; Original-Fehlerbehandlu
      ngsroutine
22:     mov       ax,[ds:144]   ; zeigt,
23:     push      ax            ; retten und
24:     mov       ax,[ds:146]   ; den
25:     push      ax            ; Zeiger
26:     call      routine       ; auf die
27: routine: pop      bx        ; Fehlerabfangroutine
28:     db        129,195,86,0  ; setzen,
29:     mov       ds:90h,bx     ; die sich
30:     push      cs            ; in diesem
31:     pop       ax            ; Programm
32:     mov       ds:92h,ax     ; befindet
33:     mov       ax,es         ;Datensegment
34:     mov       ds,ax         ; wiederherstellen
35:     mov       si,[bp+26]    ;Die einzelnen Register
36:     mov       ah,[si]      ; von AH bis DL übernehme
      n
37:     mov       si,[bp+24]
38:     mov       al,[si]
39:     mov       si,[bp+22]
40:     mov       bh,[si]
41:     mov       si,[bp+20]
42:     mov       bl,[si]
43:     mov       si,[bp+18]
44:     mov       ch,[si]
45:     mov       si,[bp+16]
46:     mov       cl,[si]
47:     mov       si,[bp+14]
48:     mov       dh,[si]
49:     mov       si,[bp+12]
50:     mov       dl,[si]
51:     push      ax            ;Register AX und BX
52:     push      bx            ; vor Änderungen schützen
53:     mov       si,[bp+10]    ;Interruptnummer in das
54:     mov       al,[si]      ; AL-Register laden
55:     call      routin2       ;Aktuellen Befehlszeiger
      feststellen
56: routin2: pop      bx        ; und in BX speichern
57:     db        129,195,14,0  ;Setzt Zeiger auf Interru
      ptaufruf
58:     mov       cs:[bx].al    ; und speichert die in Ba
      sic angegebene Interruptnummer ab
59:     jmp       short inter
60: inter: pop      bx        ;Angegebene Register
61:     pop       ax            ; wiederherstellen
62:     push      bp            ;Adresse des Rückgabepara
      meters sichern
63:     db        205,0         ;Interrupt aufrufen und
64:     jmp       weiter       ; weitermachen, wenn kein
      Fehler aufgetreten ist
65:     sti       ;Fehlerbehandlung:
66:     mov       ax,di        ;Nummer des aufgetretenen
      Fehlers
67:     mov       ah,0         ; in AL abspeichern
68:     add       sp,18        ;Unbenutzten Stapel übers
      pringen
69:     pop       bp            ; und die
70:     pop       ds            ; benötigten Register
71:     pop       es            ; wiederherstellen
72:     add       sp,6         ;Stapel weiter verkleinern
      n und
73:     pop       bp            ; den Zeiger auf die über
      gegebenen Parameter wiederherstellen
74:     mov       si,[bp+8]    ; Fehlercode
75:     mov       [si],ax      ; im Rückgabeparameter sp
      eichern
76:     jmp       abbruch      ;Aktualisierung der Regis
      ter überspringen
77:     weiter: pop      bp    ;Parameterzeiger wiederhe
      rstellen
78:     mov       si,[bp+26]   ;Die einzelnen Register
79:     mov       [si],ah      ; von AH bis DL
80:     mov       si,[bp+24]   ; in den zugehörigen
81:     mov       [si],al      ; Basic-Variablen
82:     mov       si,[bp+22]   ; abspeichern
83:     mov       [si],bh
84:     mov       si,[bp+20]
85:     mov       [si],bl
86:     mov       si,[bp+18]
87:     mov       [si],ch
88:     mov       si,[bp+16]
89:     mov       [si],cl
90:     mov       si,[bp+14]
91:     mov       [si],dh
92:     mov       si,[bp+12]
93:     mov       [si],dl
94:     abbruch: mov      ax,0  ;Den Befehlszeiger,
95:     mov       ds,ax        ; der auf die
96:     pop       ax           ; Original-Fehlerbehandlu
      ngsroutine
97:     mov       ds:92h,ax   ; zeigt, sowie das
98:     pop       ax           ; ursprüngliche
99:     mov       ds:90h,ax   ; Code-Segment erneuern
100:    mov       ax,es       ;Das vorherige Datensegme
      nt
101:    mov       ds,ax        ; wiederherstellen
102:    pop       di           ;Den Original-Stapel
103:    mov       sp,di        ; und den
104:    pop       bp           ; Basiszeiger zurückholen
105:    ret       16H          ;Zurück ins Basic-Programm
      m
106:
107: intrrpt   endp
108: cseg      ends
109:
110:

```

Listing 1. Die Assemblerfassung von »interbas« gibt Aufschluß über die Funktionsweise dieser Routine

```

1 'Programm für den Interruptaufruf aus GW-Basic heraus
2 '-----
3 'Bevor ein Interrupt aufgerufen werden kann, müssen zuerst alle
  benötigten
4 'Register initialisiert werden (AH,AL,BH,BL,CH,CL,DH,DL). Deswe
5 'müssen Sie einen neuen Stapel für das Maschinenspracheunterpro
  gramm und
6 'die Interruptnummer sowie den Rückgabeparameter angeben.
7
8 'INITIALISIERUNG
9
10 DEFINT A-Z
20 RESTORE
30 READ CODE$:IF CODE$<>"ENDE" THEN INTERRUPT$=INTERRUPT$+CHR$(VA
  L("&H"+CODE$)):GOTO 30
40 DEF SEG
100 '
200 'Hauptprogramm
300
400 END
500
10000 'Unterprogramm für den Interruptaufruf
10010
10020 INTERRUPT!=PEEK(VARPTR(INTERRUPT$)+2)*256+PEEK(VARPTR(INTER
  RUPT$)+1)
10030 CALL INTERRUPT!(AH,AL,BH,BL,CH,CL,DH,DL,NUMMER.DES.INTERRU
  TS,RUECKGABEPARAMETER,STAPEL$)
10040 RETURN
60000
60010 'Daten für das Maschinenspracheprogramm
60020
60030 DATA 55,8B,EC,8B,FC,8B,76,06,8A,04
60040 DATA B4,00,46,03,04,48,8B,E0,57,B8
60050 DATA 00,00,8E,D8,A1,90,00,50,A1,92
60060 DATA 00,50,E8,00,00,5B,81,C3,56,00
60070 DATA 89,1E,90,00,0E,5B,A3,92,00,8C
60080 DATA C0,8E,D8,8B,76,1A,8A,24,8B,76
60090 DATA 18,8A,04,8B,76,16,8A,3C,8B,76
60100 DATA 14,8A,1C,8B,76,12,8A,2C,8B,76
60110 DATA 10,8A,0C,8B,76,0E,8A,34,8B,76
60120 DATA 0C,8A,14,50,53,8B,76,0A,8A,04
60130 DATA E8,00,00,5B,81,C3,0E,00,2E,88
60140 DATA 07,EB,00,5B,58,55,CD,00,EB,18
60150 DATA 90,FB,8B,C7,B4,00,83,C4,12,5D
60160 DATA 1F,07,83,C4,06,5D,8B,76,08,89
60170 DATA 04,EB,2A,90,5D,8B,76,1A,8B,24
60180 DATA 8B,76,18,8B,04,8B,76,16,8B,3C
60190 DATA 8B,76,14,8B,1C,8B,76,12,8B,2C
60200 DATA 8B,76,10,8B,0C,8B,76,0E,8B,34
60210 DATA 8B,76,0C,8B,14,8B,00,00,8E,D8
60220 DATA 58,A3,92,00,5B,A3,90,00,8C,C0
60230 DATA 8E,D8,5F,8B,E7,5D,CA,16,00,ENDE

```

Listing 2. In dieser Form müssen Sie »interbas« in Ihre Basic-Programme einbinden


```

1 'Beispielprogramm für den Interruptaufruf aus GW-Basic heraus
2
3
4 'INITIALISIERUNG
5
10 DEFINT A-Z
20 RESTORE
30 READ CODE$: IF CODE$ <> "ENDE" THEN INTERRUPT$=INTERRUPT$+CHR$(VAL
L("&H"+CODE$)):GOTO 30
40 DEF SEG
50 KEY OFF
60 CLS
70 STAPEL$=SPACE$(80)
80 AH=0:AL=0:BH=0:BL=0:CH=0:CL=0:DH=0:DL=0
90 RUECKGABEPARAMETER=255
100
110 'Hauptprogramm
120
130 CLS
140 PRINT "      Es folgt nun ein Beispiel für einen Interrupt-A
150 PRINT "=====
160 LOCATE 4,17:PRINT"Drücken Sie bitte eine Taste, um anzufangen
170 A$=INKEY$:IF A$="" THEN 170
180 LOCATE 4,1:PRINT" Der erste Aufruf ist die DOS-Funktion 2AH,
mit der man das Datum und den Tag lesen kann. Da es in Basic a
190 PRINT:PRINT"Der aufrufende Interrupt hat die Nummer 21H, di
e Funktionsnummer wird als 2AH in AH übergeben. Nachdem wir
200 PRINT:PRINT" Drücken Sie bitte eine Taste, um die Fun
ktion 2AH aufzurufen.
210 A$=INKEY$:IF A$="" THEN 210
220 NUMMER.DES.INTERRUPTS=&H21
230 AH=&H2A
240 GOSUB 10000
250 LOCATE 12,1:PRINT"      Der Funktionsaufruf lieferte die
folgenden Registerwerte: "
310 LOCATE 14,1:PRINT"Uninteressant: AH=";AH
320 LOCATE 14,27:PRINT"Wochentag : AL=";AL
330 LOCATE 14,53:PRINT"Uninteressant: AX=";AH*256+AL
340 LOCATE 15,1:PRINT"Uninteressant: BH=";BH
350 LOCATE 15,27:PRINT"Uninteressant: BL=";BL
360 LOCATE 15,53:PRINT"Uninteressant: BX=";BH*256+BL
370 LOCATE 16,1:PRINT"Uninteressant: CH=";CH
380 LOCATE 16,27:PRINT"Uninteressant: CL=";CL
390 LOCATE 16,53:PRINT"Jahr : CX=";CH*256+CL
400 LOCATE 17,1:PRINT"Monat : DH=";DH
410 LOCATE 17,27:PRINT"Tag : DL=";DL
420 LOCATE 17,53:PRINT"Uninteressant: DX=";DH*256+DL
430 PRINT:PRINT" Die Tage sind fortlaufend von 0 (Sonntag) bi
s 6 (Samstag) nummeriert.
440 PRINT:PRINT"      Es ist der ";MID$(DATE$,4,2)"
.LEFT$(DATE$,2)".RIGHT$(DATE$,4)";, also ein ";
450 IF AL=0 THEN PRINT"Sonntag." ELSE IF AL=1 THEN PRINT"Montag."
ELSE IF AL=2 THEN PRINT"Dienstag." ELSE IF AL=3 THEN PRINT"Mittw
460 PRINT:PRINT" Drücken Sie bitte eine Taste, damit ich Ihnen w
eitere Beispiele zeigen kann.
470 A$=INKEY$:IF A$="" THEN 470
480
490 'Weitere Beispiele
500
510 CLS
520 PRINT"      Programmierhilfen und Tips & Tricks unter
r GW-Basic"
530 PRINT"=====
540 PRINT
550 PRINT" Es ist aber ebenso möglich, hilfreiche Funktionen zu
simulieren, die unter normalen Umständen nicht in GW-Basic
560 PRINT" sowie wieviel Speicherplatz auf diesem Laufwerk noch
ausgenutzt werden kann.":PRINT:PRINT"Sehen Sie sich den genauen P
570 PRINT:PRINT" Drücken Sie bitte eine Taste, um die aktuel
len Daten zu ermitteln !"
580 A$=INKEY$:IF A$="" THEN 580
590 LOCATE 12,1:PRINT STRING$(79,32)
600
610 'Get MS-DOS Version Number (Function 30H)
620

```

```

630 AH=&H30:NUMMER.DES.INTERRUPTS=&H21
640 GOSUB 10000
650 MAJOR$=RIGHT$(STR$(AL),LEN(STR$(AL))-1):MINOR$=RIGHT$(STR$(AH
),LEN(STR$(AH))-1)
660 PRINT
670 PRINT"Die MS-DOS-Versionsnummer lautet : ";MAJOR$"."M
680
690 'Get Current Disk (Function 19H)
700
710 AH=&H19:NUMMER.DES.INTERRUPTS=&H21
720 GOSUB 10000
730 PRINT
740 PRINT"Das aktuelle logische Laufwerksnummer ist: ";CHR$(65+AL
750
760 'Get Disk Free Space (Function 36H)
770
780 AH=&H36:DL=&H0:NUMMER.DES.INTERRUPTS=&H21
790 GOSUB 10000
800 PRINT
810 PRINT"Freier Platz auf dem Laufwerk (in Bytes) :";(256*AH+AL)
*(256*BH+BL)*(256*CH+CL)
820 KEY ON:END
10000 'Unterprogramm für den Interruptaufruf
10010
10020 INTERRUPT!=PEEK(VarPTR(INTERRUPT$)+2)*256+PEEK(VarPTR(INTER
RUPT$)+1)
10030 CALL INTERRUPT!(AH,AL,BH,BL,CH,CL,DH,DL,NUMMER.DES.INTERRU
PT$,RUECKGABEPARAMETER,STAPEL$)
10040 RETURN
60000
60010 'Daten für das Maschinenspracheprogramm
60020
60030 DATA 55,8B,EC,8B,FC,8B,76,06,8A,04
60040 DATA B4,00,46,03,04,48,8B,E0,57,B8
60050 DATA 00,00,8E,D8,A1,90,00,50,A1,92
60060 DATA 00,50,E8,00,00,5B,81,C3,56,00
60070 DATA 89,1E,90,00,0E,58,A3,92,00,8C
60080 DATA C0,8E,D8,8B,76,1A,8A,24,8B,76
60090 DATA 18,8A,04,8B,76,16,8A,3C,8B,76
60100 DATA 14,8A,1C,8B,76,12,8A,2C,8B,76
60110 DATA 10,8A,0C,8B,76,0E,8A,34,8B,76
60120 DATA 0C,8A,14,50,53,8B,76,0A,8A,04
60130 DATA E8,00,00,5B,81,C3,0E,00,2E,88
60140 DATA 07,EB,00,5B,58,55,CD,00,EB,18
60150 DATA 90,FB,8B,C7,B4,00,83,C4,12,5D
60160 DATA 1F,07,83,C4,06,5D,8B,76,08,89
60170 DATA 04,EB,2A,90,5D,8B,76,1A,8B,24
60180 DATA 8B,76,18,8B,04,8B,76,16,8B,3C
60190 DATA 8B,76,14,8B,1C,8B,76,12,8B,2C
60200 DATA 8B,76,10,8B,0C,8B,76,0E,8B,34
60210 DATA 8B,76,0C,8B,14,8B,00,00,8E,D8
60220 DATA 58,A3,92,00,58,A3,90,00,8C,C0
60230 DATA 8E,D8,5F,8B,E7,5D,CA,16,00,ENDE

```

Listing 3. Das Demo-Programm zeigt an einigen Beispielen, wie sich »interbas« zum Aufruf von Interruptroutinen einsetzen läßt

Ärgern Sie sich auch ab und zu über die umständliche Bedienung Ihres PCs, über merkwürdige Fehlermeldungen und unverständliche Handbücher? Das muß nicht so sein. Die »DOS-Praxis« schafft hier Abhilfe. In jeder Ausgabe werden wir uns an dieser Stelle mit für den Benutzer wichtigen Aspekten des MS-DOS/PC-DOS- Betriebssystems und des PCs allgemein auseinandersetzen. Ihre Fragen, Anregungen und Ergänzungen sind uns dabei jederzeit willkommen.

Schreiben Sie an die

Redaktion
DOS international
Stichwort DOS-Praxis
Elfenstr. 40
8000 München 83.

Disketten-Kopierstationen

Wollen Sie Ihren Ersatzprozeß beschleunigen?
Sind Ihnen durch geschützte Software die Hände gebunden?
Unsere Kopierstationen lösen Ihr Problem!
Kopie auf Knopfdruck in 50 Sekunden! Keine weitere Hard- oder Software erforderlich. Ein bestehender Kopierschutz wird übertragen.

FDD 1001 für PC/XT-Bereich (40 Spuren)	DM 2448,—
FDD 1002 für PC/XT und AT-Bereich (40/80 Spuren)	DM 2998,—
FDD 1005 NEUHEIT kopiert 5 1/4"- auf 3 1/2"-Disketten	DM 2748,—
FDD 1006 NEUHEIT 3 1/2"-Kopierstation für 2D/2DD-Disketten	DM 2948,—

Mehrfach-Kopierstation auf Anfrage.

alex bauer **a.b.c. - computer-vertrieb**
HARDWARE · SOFTWARE · PERIPHERIE
salzstraße 16 · 8170 bad tölz · telefon 08041/6294
telex 8041806 ABCOMP btx *2086*

Turbo Pascal und MS-DOS

Erst der direkte Zugriff auf das Betriebssystem verschafft dem Programmierer endgültig die vollständige Kontrolle über den PC. Doch die maschinennahe Programmierung ist ohne Grundkenntnisse der Assembler-Sprache nicht einfach. Daher wird hier eine umfassende Bibliothek von Pascal-Prozeduren vorgestellt, die jedem Programmierer alle Leistungen des Betriebssystems zugänglich machen.

Mit Turbo Pascal lassen sich auf einem MS-DOS-Computer schon eine Vielzahl von Betriebssystemfunktionen ansprechen. Insbesondere die Ein-/Ausgabeoperationen für Bildschirm, Tastatur und Dateien sind bemerkenswert gut entwickelt und gehen weit über den Pascal-Standard hinaus. Dennoch gibt es noch eine ganze Reihe von Funktionen, die auch Turbo Pascal nicht unterstützt. Dazu gehören beispielsweise die Speicherverwaltung oder die Datums- und Zeitfunktionen. In vielen Fällen eröffnen sich durch den direkten Zugriff auf MS-DOS-Routinen auch neue Möglichkeiten mit eigentlich ganz einfachen Funktionen. Bei der noch sehr weit verbreiteten älteren Version 2.0 des Turbo-Compilers ist der direkte Zugriff auf das Betriebssystem noch wichtiger als bei den neueren Versionen: Turbo 2.0 unterstützt keine Subdirectories bei der Programmierung und führt auch die Bildschirmausgaben nicht über MS-DOS, sondern direkt über BIOS-Interrupts aus. Dies bringt zwar einen Geschwindigkeitsvorteil, führt aber gleichzeitig dazu, daß zum Beispiel der Ansi-Treiber mit seinen vielfältigen Möglichkeiten zur Bildschirmsteuerung und zur Funktionstastenbelegung unter Turbo Pascal 2.0 wirkungslos bleibt.

Es lohnt sich daher also auf jeden Fall, einmal näher auf die verschiedenen MS-DOS-Funktionen einzugehen. Allerdings lassen sich diese Funktionen normalerweise nur mit einem soliden Fundament an Assembler-Programmierkenntnissen nutzen. Da müssen Prozessor-Register mit bestimmten Werten geladen werden, da muß das Flag-Register beachtet werden und was der Dinge mehr sind. Wer sich nicht mit Assembler auskennt und vor allem kein MS-DOS Reference Manual zur Hand hat, der konnte die große Vielfalt der DOS-Funktionen in seinen Programmen bislang nicht einsetzen.

Doch damit ist nun Schluß. Die in diesem Beitrag vorgestellte DOS-Bibliothek für Turbo Pascal stellt alle wesentlichen MS-DOS-Funktionen in Form von einfach aufzurufenden Turbo-Routinen zur Verfügung. Das bedeutet: kein langes Herumblättern in Programmier-Handbüchern, kein um-

ständliches und fehleranfälliges Programmieren in Assembler mehr. Einfach die entsprechende Prozedur oder Funktion aufrufen und Sie haben auf einfachste Art und Weise Ihren Computer vollständig in der Hand.

Doch zunächst ein Wort zur Begriffsklärung. In vielen Publikationen werden abwechselnd die Begriffe MS-DOS und PC-DOS als Bezeichnung für die Betriebssystem-Software von IBM-kompatiblen PCs verwendet. Um es einmal für alle PC-Anwender klarzustellen: PC-DOS und MS-DOS sind nicht identisch. PC-DOS ist das Betriebssystem für den IBM-PC, das aus urheberrechtlichen Gründen in kompatiblen PCs nicht verwendet werden darf. Microsoft, der Schöpfer des PC-DOS, bietet aber ein zu PC-DOS kompatibles Betriebssystem an, eben MS-DOS. MS-DOS gibt es für alle IBM-kompatiblen PCs und für eine Reihe weiterer, nicht kompatibler PCs wie den Siemens PCD, den Sirius I oder den DEC Rainbow. Alle Funktionsaufrufe haben bei PC-DOS und MS-DOS identische Wirkung (gleiche Versionen vorausgesetzt). Aus diesem Grunde wird in den folgenden Abschnitten aus Gründen der Vereinfachung einfach nur von MS-DOS gesprochen, wenn MS-DOS oder PC-DOS gemeint sind.

Die MS-DOS-Funktionen werden über den Interrupt 21hex aufgerufen, wobei die Funktionsnummer zuvor in das AH-Register verfrachtet werden muß. Beispielsweise sieht ein Aufruf der MS-DOS-Funktion Nummer 0 (»Programm beenden«) in Assembler folgendermaßen aus:

```
mov ah,0 ; MS-DOS-Funktion 0
int 21h ; DOS-Interrupt ausführen
```

Viele DOS-Funktionen benötigen noch zusätzliche Parameter. Diese werden im Bedarfsfalle in weiteren Prozessor-Registern übergeben. Außerdem sind die meisten DOS-Routinen in der Lage, Fehler zu erkennen und eine Fehlernummer zu übergeben. In der Regel zeigt das Carry-Flag des Prozessors einen Fehler an. Kehrt eine DOS-Routine mit gesetztem Carry-Flag zurück, so bedeutet dies in den meisten Fällen, daß ein Fehler aufgetreten ist. In einem solchen Fall enthält das Register AX in der Regel eine Fehlernummer. Tabelle 1 zeigt

alle DOS-Fehlermeldungen, Tabelle 2 die zusätzlichen Informationen, wie sie von der DOS-Funktion 59hex (»Get Extended Error«) bei Bedarf geliefert werden können. Ist das Carry-Flag gelöscht, dann ist alles in Ordnung und der Inhalt von AX hängt von der konkreten Routine ab; jedenfalls ist es kein Fehlercode.

Die Anzahl der MS-DOS-Funktionsaufrufe wurde von DOS-Version zu DOS-Version erhöht; insgesamt sind die Funktionsnummern 0 bis 127 reserviert. Wie bereits erwähnt, laufen alle diese Funktionsaufrufe über den Interrupt 21hex. Daneben verwendet MS-DOS noch weitere Interrupts (Tabelle 3). Bis auf den Funktions-Interrupt 21hex sind die DOS-Interrupts jedoch in erster Linie für die interne Verwendung durch MS-DOS vorgesehen. Microsoft und IBM raten daher dringend von der Verwendung anderer DOS-Interrupts als 21hex ab. Einige dieser Interrupts sind mittlerweile

Sie finden hier aufgelistet die Fehlercodes und ihre Bedeutung, wie sie von den meisten neuen DOS-Funktionen ab Version 2 im AX-Register zurückgeliefert werden. Die Fehlernummern von 1 bis 12 sind die sogenannten »einfachen Fehlercodes«, die direkt von den DOS-Routinen stammen. Bei den Codes 13 bis 35 handelt es sich um die »erweiterten Fehlercodes«, wie sie von der DOS-Funktion 59H (Get Extended Error) geliefert werden. Die Fehlernummern 36 bis 49 sind für zukünftige Verwendung reserviert; Fehlermeldungen zwischen 50 und 79 sowie ab 84 sind bei Netzwerk-Betrieb vorgesehen.

Code	Bedeutung
1	Ungültiger Funktionscode
2	Datei nicht gefunden
3	Pfad nicht gefunden
4	Zu viele offene Dateien
5	Zugriff nicht möglich
6	Ungültige Dateinummer
7	Speicherkontrollblock zerstört
8	Unzureichende Speicherkapazität
9	Ungültige Speicherkontrollblockadresse
10	Ungültige Umgebung
11	Ungültiges Format
12	Ungültiger Zugriffscode
13	Ungültige Daten
14	(reserviert)
15	Ungültiges Laufwerk
16	Versuch, aktuelles Verzeichnis zu löschen
17	Nicht das gleiche Gerät
18	Keine weiteren Dateien
19	Diskette hat Schreibschutz
20	Fehlerhafte Disketten-Einheit
21	Laufwerk nicht bereit
22	Unzulässiges Disketten-Kommando
23	CRC-Fehler
24	Falsche Längenangabe bei Disketten-Operation
25	Suchfehler
26	Keine MS-DOS-Diskette
27	Sektor nicht gefunden
28	Drucker ohne Papier
29	Schreibfehler
30	Lesefehler
31	Allgemeiner Fehler
32	Sharing-Verletzung
33	Lock-Verletzung
34	Falsche Diskette
35	Zugriff auf FCB nicht möglich
36	Datei existiert bereits
37	(reserviert)
38	Aufgabe nicht durchführbar
39	Fehler innerhalb der Fehlerbehandlungsroutine

Tabelle 1. Die Fehlermeldungen der MS-DOS-Funktionen

durch gleichartige oder verbesserte Funktionsaufrufe ergänzt worden und nur noch aus Kompatibilitätsgründungen in den Versionen ab 3.0 enthalten. Es gibt keine Garantie, daß derartige DOS-Interrupts auch in zukünftigen Versionen von MS-DOS unterstützt werden.

Eine ganze Reihe von Funktionen stammen noch aus der Anfangszeit des MS-DOS, wo auf Kompatibilität zum CP/M-Betriebssystem Wert gelegt wurde. Diese Funktionsaufrufe sind in neuen DOS-Versionen ebenfalls nur noch aus Kompatibilitätsgründen enthalten. Von ihrer Verwendung raten bis auf wenige Ausnahmen sowohl IBM als auch Microsoft ab. Statt dessen sind neue, Unix-ähnliche Funktionsaufrufe vorgesehen, welche die alten CP/M-kompatiblen Aufrufe ersetzen.

Die Turbo-Bibliothek »msdos« (Listing 1) ist ein modulares System aus Pascal-Prozeduren und Funktionen zum direkten Zugriff auf die DOS-Ebene. Diese Bibliothek muß mittels der Include-Option des Turbo-Compilers eingebunden werden, wenn man die Routinen dieser Bibliothek in eigenen Programmen nutzen möchte. Bis auf die globalen Vereinbarungen, die Fehlerbehandlung und natürlich dem Modul »DOS-Funktionen aufrufen« können Sie jedes einzelne Modul nach Belieben fortlassen, ändern oder ergänzen. Somit bauen Sie sich auf der Grundlage der abgedruckten MS-DOS-Bibliothek leicht Ihre ganz persönliche Schnittstelle zum Betriebssystem, die alles enthält was Sie an Funktionen für Ihre Programmierung benötigen.

Der Zugriff auf die Prozessor-ebene ist für Turbo Pascal kein Problem

Turbo Pascal bietet unter MS-DOS zwei Standardprozeduren zum direkten Zugriff auf das Betriebssystem an. Es handelt sich dabei um die sehr allgemeinen Prozeduren »intr« und »msdos«. Die Intr-Prozedur wird aufgerufen in der Form

```
intr(nr, reg)
```

Die beiden Parameter der Prozedur haben dabei die folgende Bedeutung. »nr« ist die Nummer des gewünschten Interrupts, also ein Integer-Wert zwischen 0 und 255. »reg« (für Register) ist eine Variable mit der Typdefinition

```
RegisterTyp = record AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags: integer end;
```

Die Register-Variable repräsentiert praktisch die einzelnen Prozessor-Register. Vor der Ausführung des Interrupts werden die tatsächlichen Prozessor-Register mit den Inhalten der

1. Wert im Register BH: Fehlerklasse	
Code	Bedeutung
1	Unzureichende Ressourcen wie Speicher oder Kanäle
2	Kein Fehler, sondern ein nur zeitweiliges Problem
3	Nicht autorisierter Dateizugriff
4	Interner Fehler im Betriebssystem
5	Hardware-Fehler
6	Nicht vom aktuellen Prozeß verursachter DOS-Fehler
7	Fehler im Anwender-Programm
8	Datei oder anderes Betriebsmittel wurde nicht gefunden
9	Datei oder Betriebsmittel in ungültigem Format oder sonstwie unbrauchbar
10	Datei oder Betriebsmittel für Zugriff gesperrt
11	Falsche oder schadhafte Diskette oder anderes Problem mit einem Speichermedium
12	Sonstiger Fehler
2. Wert im Register BL: vorgeschlagene Aktion	
Code	Bedeutung
1	Gewünschte Aktion nochmals versuchen, bei Fehlschlag Benutzer benachrichtigen
2	Gewünschte Aktion nach einer Pause nochmals versuchen
3	Wenn der Benutzer eine Eingabe gemacht hat (Wie Laufwerks- oder Dateiname), dann nochmals um (korrekte) Eingabe bitten
4	Programm geordnet beenden
5	Programm sofort ohne weitere Verzögerungen abbrechen
6	Informeller Fehler, kann ignoriert werden
7	Den Benutzer zu bestimmten Tätigkeiten auffordern (Diskettenwechsel etc.)
3. Wert im Register CH: Ort der Fehlerursache	
Code	Bedeutung
1	Unbekannt
2	Fehler liegt bei einem blockorientierten Gerät (beispielsweise Diskettenstation)
3	Fehler liegt in einem angeschlossenen Netzwerk
4	Fehler liegt bei einem zeichenweise anzusprechendem Gerät (beispielsweise Drucker)
5	Fehler liegt im Schreib-/Lesespeicher (RAM)

Tabelle 2. Fehlerklasse, Fehlerort und vorgeschlagene Reaktion, wie sie von der DOS-Funktion 59hex (»Get Extended Error«) geliefert werden

Register-Variablen geladen und nach Ausführung des Interrupts umkehrt die Prozessor-Register in den Register-Record gerettet. Auf diese Art und Weise ist es möglich, den Interrupt-Routinen Registerinhalte als Parameter zu übergeben. Andererseits kann man nach dem Interrupt-Aufruf die von der Interrupt-Routine zurückgegebenen Werte aus der Register-Variablen auslesen.

In der MS-DOS-Bibliothek wird allerdings eine etwas verfeinerte Definition für diesen Registertyp verwendet. Der 8086-Prozessor kann ja die Register AX, BX, CX und DX wahlweise auch als ein Paar aus zwei 8-Bit-Registern ansprechen. Um sich bei Verwendung von 8-Bit-Werten umständliche Umrechnungen zu ersparen, ist der Register-Record in der MS-DOS-Bibliothek als vollständig varianter Record (»Union«) realisiert. So kann man beispielsweise das AX-Register ganz normal mit »reg.ax« als 16-Bit-Register ansprechen, gleichzeitig aber auch als Paar aus den zwei 8-Bit-Registern »reg.al« und »reg.ah«. Auch die Standardprozedur »msdos« von Turbo-Pascal arbeitet mit der eben beschriebenen Registervariablen. Der Aufruf geschieht in der Form

```
msdos(reg)
```

Der einzige Unterschied zur Intr-Prozedur ist also, daß keine Interrupt-Nummer angegeben wird. Die Prozedur »msdos« ruft immer den Funk-

tions-Interrupt 21hex des MS-DOS-Systems auf. Über diesen Interrupt werden, wie bereits zu Anfang beschrieben, alle Funktionsaufrufe abgewickelt. Dazu wird im Prozessorregister AH die Nummer der gewünschten Funktion übergeben. In Turbo Pascal sieht daher der Aufruf beispielsweise der DOS-Funktion 0 (Programm beenden) so aus (die eben besprochene Typdefinition des Register-Records vorausgesetzt):

```
reg.ah := 0;
msdos(reg);
```

Das Ergebnis dieses Funktionsaufrufes ist die Beendigung des gerade laufenden Turbo-Programms. Der Aufruf entspricht also der Turbo-Prozedur »halt«.

Die Fehlerbehandlung unter MS-DOS ist nicht einheitlich

Die Entwicklung von MS-DOS erfolgte in bisher drei großen Schritten: Die ersten Versionen des MS-DOS-Betriebssystems waren aus Gründen der damals notwendig erscheinenden weitgehenden Kompatibilität zu CP/M sehr stark an dieses Betriebssystem angelehnt. Erst die MS-DOS Version 2 brachte die heute allgemein verwendeten Unix-ähnlichen Funktionen zur Steuerung des Dateisystems und eine konsequente, einheitliche Fehlerbehandlung. Fast alle ab Version 2 neu hinzugekommenen Funktionsaufrufe arbeiten mit der

Interrupt	Beschreibung
20hex	Programm beenden (wie DOS-Funktion 4Chex)
21hex	Hauptinterrupt für alle DOS-Funktionsaufrufe
22hex	Programmende-Routine, wird vor der Rückkehr zu DOS durchlaufen
23hex	Routine zur Behandlung von Break (CTRL-C)
24hex	Routine zur Behandlung kritischer I/O-Fehler
25hex	Disk-Sektoren absolut lesen
26hex	Disk-Sektoren absolut schreiben
27hex	Programm beenden und resident im Speicher verbleiben
2Fhex	Druckerspoober (nur DOS-Version 3)

Tabelle 3. Die MS-DOS-Interrupts

Vereinbarung, einen aufgetretenen Fehler durch ein gesetztes Carry-Flag bei Rückkehr der Routine zu signalisieren. Das aufrufende Programm kann also sehr leicht anhand des Carry-Flags prüfen, ob ein Fehler vorliegt oder nicht. Falls ein Fehler aufgetreten ist, dann (und nur dann) enthält das AX-Register einen Fehlercode, dessen Bedeutung anhand der Tabelle 1 entschlüsselt werden kann.

Ab der MS-DOS Version 3 ist neben einigen weiteren Funktionen, die unter anderem Netzwerke unterstützen, eine wichtige Funktion zur Fehlerbehandlung neu hinzugekommen. Es handelt sich dabei um die Funktion 59hex »Get Extended Error«. Diese Funktion liefert bei Aufruf genauere Angaben über Art und Ursprung eines Fehlers bei der jeweils zuletzt aufgerufenen DOS-Funktion. Außerdem wird ein Hinweis geliefert, welche Aktionen seitens des laufenden Programms zur Behandlung des Fehlers sinnvoll sind.

Insgesamt werden vier Angaben über den Fehler gemacht: Erweiterter Fehlercode (»Extended Error Code«), Fehlerklasse (»Error Class«), Aktionsvorschlag (»Suggested Action«) und Fehlerort (»Location«). Die Bedeutung des erweiterten Fehlercodes kann anhand der Tabelle 1 bestimmt werden. Nur die ersten zwölf Fehlercodes in dieser Tabelle werden von den MS-DOS-Funktionen direkt zurückgegeben, alle weiteren Codes werden nur durch die Funktion 59hex als erweiterte Fehlercodes geliefert. Die möglichen Werte für Fehlerklasse, Aktionsvorschlag und Fehlerort sind in Tabelle 2 zusammengefaßt.

Die Turbo-Bibliothek »msdos« (Listing 1) macht von dieser Möglichkeit der genaueren Fehlerfeststellung Gebrauch. Die Prozedur »GetXErrCode« ruft die DOS-Funktion 59hex auf und liefert die Ergebnisse an die vier globalen Variablen »ErrorCode«, »ErrorClass«, »ErrorAction« und »ErrorLocation«. Der Aufruf von »GetXErrCode« erfolgt automatisch nach jedem erkannten DOS-Fehler über die allgemeine Prozedur »Error«. Die vier

zusätzlichen Fehlervariablen kann das Turbo-Anwenderprogramm ebenso wie die Variable »ErrorNum« (enthält den einfachen Fehlercode) auswerten. Nach einer Abfrage sollten diese Variablen mittels der Prozedur »ClearError« wieder auf Null zurückgesetzt werden. Erwähnt werden sollte noch die Stringvariable »ErrorMsg«, die den Text der letzten Fehlermeldung enthält, ebenso die Schaltervariable »Trap«, eine Variable vom Typ Boolean, welche die Ausgabe von Fehlermeldungen regelt. Hat »Trap« den Wert »true«, so werden von der Fehleroutine nur die entsprechenden Fehlervariablen gesetzt, der Programmablauf wird jedoch nicht unterbrochen. Ist »Trap« dagegen auf »false« gesetzt, dann erscheint bei jedem Fehler eine Meldung am Bildschirm und das Programm bricht ab.

Bei DOS-Versionen früher als 3.0 ist der Funktionsaufruf 59hex und damit die Durchführung der Prozedur »GetXErrCode« nicht möglich. In diesem Falle sollte diese Prozedur aus der MS-DOS-Bibliothek entfernt werden. Die MS-DOS-Bibliothek enthält die folgenden Module (vergleiche Tabelle 4):

(1) Globale Vereinbarungen: Hier werden die für die Fehlerbehandlung wichtigen Variablen als typisierte Konstanten definiert. Außerdem erfolgt die Typdefinition und Variablenvereinbarung für einfache Zeiger auf Speicherbytes, für Pfadnamen, Dateinummern und schließlich für die Registerstruktur des 8086-Prozessors. Diese letzte Vereinbarung ist die wichtigste Grundlage für alle Bibliotheks-Module rund um das MS-DOS-Betriebssystem. Die Prozessor-Register sind als variabler Record definiert, so daß sowohl 16- als auch 8-Bit-Register direkt und einfach mit ihrem Namen angesprochen werden können.

(2) DOS-Funktionen aufrufen: Dieses Modul besteht aus den beiden Prozeduren »DOS« und »Carry«. »DOS« wird mit der gewünschten DOS-Funktionsnummer als Parameter aufgerufen und setzt einen global definierten Register-Record »reg« voraus. Die Funktion »Carry« testet den Status des Carry-Flags innerhalb des Flag-Wortes des Register-Records. Wenn »Carry« den Wert »true« liefert, war das Carry-Flag gesetzt, was bei den meisten DOS-Funktionen gleichbedeutend mit dem Auftreten eines Fehlers ist.

(3) Fehlerbehandlung: Dieses Modul besteht aus den beiden schon besprochenen Prozeduren »GetXErrCode« und »Error«. Falls eine DOS-Version kleiner als 3.0 verwendet wird, muß die Prozedur »GetXErrCode« gelöscht werden, ebenso der Aufruf dieser

Prozedur innerhalb der Error-Prozedur.

(4) Allgemeine I/O-Funktionen: Dieses Modul ist besonders interessant für Besitzer älterer Versionen von Turbo Pascal (2.x). Wenn Sie statt mit »read« und »write« die Ein-/Ausgabe über die Funktionen dieses Moduls abwickeln, dann stehen Ihnen bisher versperrte Optionen wie die Ein-/Ausgabe-Umlenkung auf DOS-Ebene zur Verfügung.

(5) Allgemeine Disk-Funktionen: Hier sind die grundlegenden Funktionen zum Umgang mit dem Diskettensystem zusammengefaßt. Damit können Sie beispielsweise eine nähere Information über die Laufwerke erhalten oder zwischen den verschiedenen Laufwerken umschalten.

(6) Allgemeine DOS-Funktionen: Hier sind Routinen für das Setzen und Lesen von Interrupt-Vektoren, landesspezifischen Daten und der Verify- und Break-Flags zusammengefaßt. Auch die DOS-Version läßt sich hiermit feststellen.

(7) Datums- und Zeitfunktionen: Neben vier Prozeduren zum Lesen und Schreiben von Uhrzeit und Datum sind noch zwei weitere interessante Prozeduren realisiert, mit denen sich Datum- und Zeitangaben von Dateien setzen und lesen lassen.

(8) Directory-Funktionen: Dieses Modul faßt alle Manipulationsmöglichkeiten zusammen, die MS-DOS für Unterverzeichnisse bietet. Viele dieser Funktionen sind in Turbo Pascal 3.0 bereits integriert; für Besitzer älterer Turbo-Versionen ist dieses Modul aber sicherlich sehr interessant. Auch das wichtige Problem des Auslesens der Directory-Informationen (Dateinamen, Längenangaben etc.) wird gelöst.

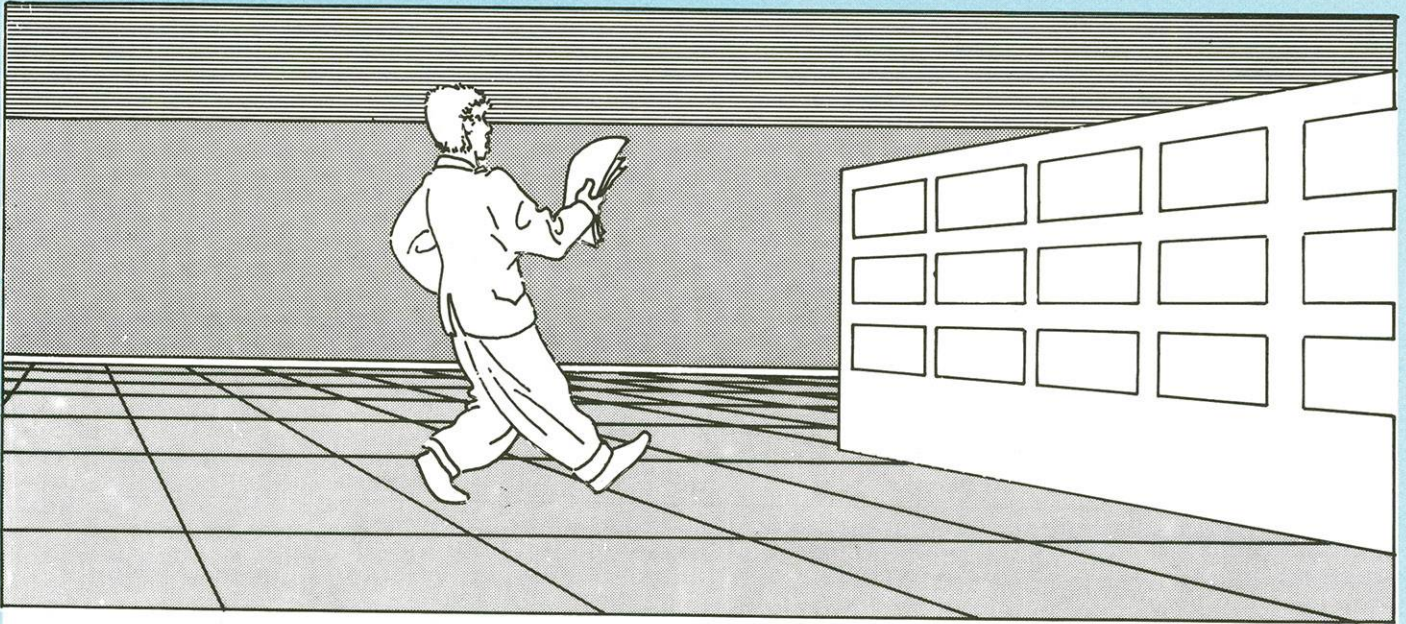
(9) Datei-Funktionen: Hier bietet sich Ihnen der direkte Zugriff auf das MS-DOS-Dateisystem, was für Spezialanwendungen durchaus von Nutzen sein kann.

(10) Speicherverwaltung: Mit den Prozeduren dieses Moduls greifen Sie direkt in die Speicherverwaltung von MS-DOS ein. Daher ist dies eines der »gefährlichsten« Module überhaupt; von einem erfahrenen Programmierer eingesetzt, entfalten sich mit den Routinen dieses Moduls aber ungeahnte Möglichkeiten: Doch äußerster Vorsicht ist geboten, damit es nicht zu Kollisionen mit der Speicherverwaltung durch Turbo Pascal selbst kommt.

(11) Ein-/Ausgabesteuerung für Geräte: Hier sind die wichtigsten Unterfunktionen des Geräte-Interrupts 68 für Pascal-Programmierer zugänglich gemacht. Eine gewisse Vertrautheit mit der MS-DOS-Gerätesteuerung ist für den sinnvollen Einsatz dieser Routinen unbedingt notwendig. (ev)

Modul 1: Globale Vereinbarungen	
const NUL	Nullbyte
const trap	Fehlerabfang-Flag
const ErrorLevel	Anzahl Fehler
const ErrorNum	Fehlernummer
const ErrorCode	Fehlercode
const ErrorClass	Fehlerklasse
const ErrorAction	vorgeschlagene Aktion
const ErrorLocation	Bereich des Fehlers
const ErrorMessage	Text der Fehlermeldung
type pointer = ^byte;	Universalzeiger
type PathName = string[128];	Pfadname
type FileHandle = integer;	Dateinummer
type register = record;	Register-Record
var reg: register	Register-Variable
Modul 2: DOS-Funktionen aufrufen	
procedure dos(nr: byte);	MS-DOS-Funktion nr aufrufen
function carry: boolean;	Zustand des Carry-Flags
Modul 3: Fehlerbehandlung	
procedure GetXErrCode(var code, class, action, loc integer);	erweiterten Fehlercode melden
procedure error;	Fehlerbehandlungsroutine
procedure ClearError;	Alle Fehlervariablen zurücksetzen
Modul 4: Allgemeine I/O-Funktionen	
function key: char;	Tastatureingabe mit Echo
procedure DisplayChr(ch: char);	BildschirmAusgabe
function SerialIn: char;	Serielle Eingabe
procedure SerialOut(ch: char);	Serielle Ausgabe
procedure Print(ch: char);	Druckerausgabe
function GetKey: char;	Direkte Tastatureingabe ohne Echo und Ctrl-C
function InKey: char;	Direkte Tastatureingabe ohne Echo, mit Ctrl-C
procedure FlushKeys;	Tastaturpuffer löschen
Modul 5: Allgemeine Disk-Funktionen	
function DriveName(drive: Laufwerk): char;	Laufwerksnamen bilden
procedure ResetDisk;	Laufwerke zurücksetzen
procedure SelectDisk(drive: Laufwerk);	Standardlaufwerk setzen
function CurrentDisk: Laufwerk;	Standardlaufwerk feststellen
procedure SetDTA(Adresse: pointer);	Disk Transfer Adresse setzen
procedure GetDTA(var Adresse: pointer);	Disk Transfer Adresse lesen
function DriveType(drive: Laufwerk): Laufwerkstyp;	Laufwerkstyp ermitteln
function ClustersPerDrive(drive: Laufwerk): integer;	Blocks pro Diskette feststellen
function SectorsPerCluster(drive: Laufwerk): integer;	Sektoren pro Block feststellen
function BytesPerSector(drive: Laufwerk): integer;	Bytes pro Sektor feststellen
function AvailableClusters(drive: Laufwerk): integer;	Freie Blöcke feststellen
Modul 6: Allgemeine DOS-Funktionen	
procedure SetInterrupt(Nummer: byte; Routine: pointer);	Interruptvektor setzen
procedure GetInterrupt(Nummer: byte; var Routine: pointer);	Interruptvektor lesen
procedure GetCountryData(Code: integer; Data: Pointer);	Zeiger auf landesabhängige Informationen holen
procedure SetCountryCode(Code: integer);	Landescode setzen
procedure SetVerify(Status: boolean);	Verify-Status setzen
function VerifyStatus: boolean;	Verify-Status abfragen
procedure SetBreak(Status: boolean);	Break-Status (Control-C) setzen
function BreakStatus: boolean;	Break-Status (Control-C) abfragen
procedure GetVersion(Version: Versionstyp);	MS-DOS-Version und Seriennummer ermitteln
Modul 7: Datums- und Zeitfunktionen	
procedure GetDate(var Datum: Datumstyp);	Datum abfragen
procedure SetDate(Datum: Datumstyp);	Datum setzen
procedure GetTime(var Zeit: Zeittyp);	Uhrzeit lesen
procedure SetTime(Zeit: Zeittyp);	Uhrzeit setzen
procedure GetFileDate(fh: FileHandle; var Date: Datumstyp; var Time: Zeittyp);	Datum und Zeit einer Datei holen
procedure PutFileDate(fh: FileHandle; Date: Datumstyp; Time: Zeittyp);	Datum und Zeit einer Datei setzen
Modul 8: Directory-Funktionen	
procedure MakeDir(Name: PathName);	Subdirectory anlegen
procedure RemoveDir(Name: PathName);	Subdirectory löschen
procedure ChangeDir(Name: PathName);	Subdirectory wechseln
function FirstFile(Name: PathName; Attribut: integer): PathName;	Ersten Directory-Eintrag suchen
function NextFile: PathName;	Nächsten Directory-Eintrag suchen
function CurrentDir(Drive: Laufwerk): PathName;	Pfadnamen des aktuellen Directory ermitteln
procedure MoveDir(AlterName, NeuerName: PathName);	Datei und Pfad umbenennen
Modul 9: Datei-Funktionen	
procedure CreateFile(Name: PathName; Attribut: integer; var fh: FileHandle);	Datei erzeugen
procedure OpenFile(Name: PathName; AccessCode: byte; var fh: FileHandle);	Datei öffnen
procedure CloseFile(fh: FileHandle);	Datei schließen
function ReadFile(fh: FileHandle; Anzahl: integer; Puffer: pointer);	Anzahl Bytes in Puffer einlesen
function WriteFile(fh: FileHandle; Anzahl: integer; Puffer: pointer);	Anzahl Bytes aus Puffer in Datei schreiben
procedure DeleteFile(Name: PathName);	Datei löschen
procedure SetFileAttribute(Name: PathName; Attr: integer);	Datei-Attribute setzen
function FileAttribute(Name: PathName): integer;	Datei-Attribute abfragen
function DuplicateHandle(fh: FileHandle): FileHandle;	Zweite Dateinummer erzeugen
procedure ForceDuplicateHandle(fh1, fh2: FileHandle);	Dateinummern gleichschalten
procedure CreateTempFile(Name: PathName; Attr: integer; var fh: FileHandle);	Temporäre Datei anlegen
procedure CreateNewFile(Name: PathName; Attr: integer; var fh: FileHandle);	Neue Datei anlegen
Modul 10: Speicherverwaltung	
function AllocMem(Paragraphs: integer): integer;	Speicher anfordern
procedure FreeMem(Adress: integer);	Speicher freigeben
procedure ChangeMemSize(Adress, Size: integer);	Speicherblockgröße ändern
procedure KeepProgram(Size: integer; Code: byte);	Programm speicherresident machen und beenden
procedure EndProcess(Code: byte);	Prozeß mit Return-Code abbrechen
function GetReturnCode: integer;	Return-Code holen
procedure GetAllocationStrategy(var n: integer);	Speicherbelegungs-Strategie melden
procedure SetAllocationStrategy(n: integer);	Speicherbelegungs-Strategie setzen
function GetPSP: integer;	Segment-Adresse des PSP holen
Modul 11: Ein-/Ausgabesteuerung für Geräte	
procedure GetDeviceData(Device: FileHandle; var Data: integer);	Gerätedatenwort holen
procedure SetDeviceData(Device: FileHandle; Data: integer);	Gerätedatenwort setzen
function SendControlData(Device: FileHandle; Data: pointer; Anzahl: integer);	Kontrolldaten senden
function ReceiveControlData(Device: FileHandle; Data: pointer; Anzahl: integer);	Kontrolldaten empfangen
function SendControlDataBlock(Device: Laufwerk; Data: pointer; Anzahl: integer);	Kontrolldaten an Laufwerk senden
function ReceiveControlDataBlock(Device: Laufwerk; Data: pointer; Anzahl: integer);	Kontrolldaten von Laufwerk empfangen
function DeviceInputStatus(Device: FileHandle): boolean;	Geräte-Eingabestatus feststellen
function DeviceOutputStatus(Device: FileHandle): boolean;	Geräte-Ausgabestatus feststellen
function is_Diskette(Drive: Laufwerk): boolean;	Feststellen, ob Laufwerk mit Disketten arbeitet

Tabelle 4. Übersicht über Vereinbarungen, Prozeduren und Funktionen in der Bibliothek »msdos.inc«



```

0: ( ----- )
1: ( MSDOS.INC - Include-Datei für direkten MS-DOS-Zugriff )
2: ( ----- )
3: ( **** (c) 1987 Volker Everts & DOS International )
4: ( **** Beschreibung siehe DOS-Sonderheft 1, 1987 )
5: ( ----- )
6:
7:
8:
9: ( ----- Globale Vereinbarungen ----- )
10:
11:
12: const
13:   NUL = #0;
14:   trap: boolean = false;      ( Fehlermeldungsflag )
15:   ErrorLevel:   byte = 0;     ( Anzahl Fehler )
16:   ErrorNum:     byte = 0;     ( Fehlernummer )
17:   ErrorCode:    integer = 0;  ( Fehlercode )
18:   ErrorClass:   integer = 0;  ( Fehlerklasse )
19:   ErrorAction:  integer = 0;  ( vorgeschlagene Aktion )
20:   ErrorLocation: integer = 0; ( Bereich des Fehlers )
21:   ErrorMessage: string[40] = 'OK'; ( Fehlertext )
22:
23:
24:
25: type
26:   pointer = ^byte;      ( Universalzeiger )
27:   PathName = string[128]; ( Pfadname )
28:   FileHandle = integer; ( Dateinummer )
29:   register = record
30:     case boolean of
31:       true:
32:         (ax,bx,cx,dx,bp,si,di,ds,es,Flags: integer
33:       false:
34:         (al,ah,bl,bh,cl,ch,dl,dh: byte);
35:     end;
36:
37:
38:
39:
40: var
41:   reg: register;
42:
43:
44:
45: ( ----- DOS-Funktionen aufrufen ----- )
46:
47:
48:
49: procedure dos(nr: byte);
50:
51: ( MS-DOS-Funktion nr aufrufen )
52:
53: begin
54:   reg.ah := nr;
55:   MsDos(reg);
56: end;
57:
58:
59:
60: function carry: boolean;
61:
62: ( gibt den Zustand des Carry-Flags aus )
63:
64: begin
65:   carry := odd(reg.flags);
66: end;
67:
68:
69:
70: ( ----- Fehlerbehandlung ----- )

```

```

71:
72:
73: procedure GetXErrCode(var code, class, action, loc: integer
74: );
75: ( erweiterten Fehlercode melden (nur DOS 3.x !) )
76: ( Aufruf nur direkt nach letztem DOS-Aufruf möglich )
77:
78: begin
79:   reg.bx := 0;
80:   dos(89);
81:   code := reg.ax;      ( Fehlercode )
82:   class := reg.bh;     ( Fehlerklasse )
83:   action := reg.bl;    ( vorgeschlagene Aktion )
84:   loc := reg.ch;       ( Fehlerort )
85: end;
86:
87:
88:
89: procedure error;
90:
91: ( Diese Prozedur kann vom Anwender nach )
92: ( seinen Erfordernissen abgeändert werden )
93:
94: const
95:   BEL = #7;
96:
97: var
98:   msg: string[40];
99:   nr: integer;
100:
101: begin
102:   nr := reg.ax;
103:   GetXErrCode(ErrorCode, ErrorClass, ErrorAction, ErrorLoca
104:   tion);
105:   case nr of
106:     1: msg := 'ungültige Funktionsnummer';
107:     2: msg := 'Datei nicht gefunden';
108:     3: msg := 'Pfad nicht gefunden';
109:     4: msg := 'zu viele Dateien offen';
110:     5: msg := 'Zugriff nicht möglich';
111:     6: msg := 'Datei nicht offen';
112:     7: msg := 'Speicherkontrollblock beschädigt';
113:     8: msg := 'zuwenig Speicher';
114:     9: msg := 'unkorrekte Segmentadresse';
115:     10: msg := 'ungültige Umgebung';
116:     11: msg := 'ungültiges Format';
117:     12: msg := 'unzulässiger Zugriffscode';
118:     13: msg := 'ungültige Daten';
119:     14: msg := 'ungültiges Laufwerk';
120:     15: msg := 'aktuelles Directory';
121:     16: msg := 'nicht das gleiche Gerät';
122:     17: msg := 'keine weiteren Dateien';
123:     18: msg := 'Diskette hat Schreibschutz';
124:     19: msg := 'fehlerhafte Disketteneinheit';
125:     20: msg := 'Laufwerk nicht bereit';
126:     21: msg := 'Unzulässiges Disketten-Kommando';
127:     22: msg := 'CRC-Fehler';
128:     23: msg := 'Falsche Längenangabe';
129:     24: msg := 'Suchfehler';
130:     25: msg := 'keine MS-DOS-Diskette';
131:     26: msg := 'Sektor nicht gefunden';
132:     27: msg := 'Drucker ohne Papier';
133:     28: msg := 'Schreibfehler';
134:     29: msg := 'Lesefehler';
135:     30: msg := 'Allgemeiner Fehler';
136:     31: msg := 'Sharing-Verletzung';
137:     32: msg := 'Lock-Verletzung';
138:     33: msg := 'Falsche Diskette';
139:     34: msg := 'Zugriff auf FCB nicht möglich';
140:     35: msg := 'Datei existiert bereits';
141:     36: msg := 'Aufgabe nicht durchführbar';
142:     37: msg := 'Fehler in der Fehlerbehandlung';
143:     else msg := 'Unbekannter Fehler';
144:   end; ( case )

```


SAVE OUR SOULS!

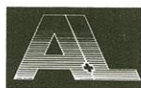
SOFTWARE-ENGINEERING MIT PASCAL UND MODULA-2

Sie verwenden eine gute Sprache. Wir haben die passenden Werkzeuge. ● **Datenbanken:** Ob am Einzelplatz oder im Netzwerk: B-Tree ISAM mit verbessertem B-Baum-Algorithmus. Inklusive Quellcode in Modula-2 oder Turbo-Pascal. Die Netzwerk-Version mit vollem File- und Record-Locking für 999,- DM, die Einzelplatz-Version für knapp die Hälfte. Weitere Tools: dBASE III-Interfaces, Btrieve mit der Abfragesprache XQL. ● **Grafik:** Ein Grafikpaket mit 200+ Routinen für alle Karten: MetaWindow unterstützt CGA, EGA, VGA, Hercules sowie weitere 40 Karten. Darunter auch viele »Super-EGAs« in der höchsten Auflösung sowie Spezial-Karten bis zu 1280x800 Punkten. MetaWindow für Turbo-Pascal kostet 295,- DM, für Modula-2 495,- DM. Versionen für C und Fortran erhältlich. Außerdem: GKS, GEM-Bindings, Turbo-HALO mit Business-Grafik-Interpreter. ● **DeskEngine:** Der erste Desktop-Manager in Turbo-Source-Code! Rechner, Notizblock, Kartei, Terminverwaltung und mehr mit genauer Beschreibung der Routinen. Auch für eigene Applikationen. ● **Kommunikation:** Ohne eine Zeile Assembler: XMODEM-Protokoll mit Turbo Asynch Plus, 57 600 Baud mit Turbo Talk. ● **Mathematik:** Zum Beispiel Mathpak 87: Ihr 8087/-287/-387 wird durch Assembler-routinen unter Hochsprachen so richtig gefordert. Spektralanalyse, FFT, Integration - Präzision und Geschwindigkeit in Vollendung. Des weiteren Bibliotheken für Statistik, Numerik und Optimierung. ● **Systemsoftware:** Die besten Compiler sind uns gerade gut genug: Turbo-Pascal 4.0, MetaWare Professional Pascal für 80386 oder UNIX, Logitech Modula-2/86 3.0, Modula-2 Entwicklungssystem M2SDS. Und natürlich Debugger, Profiler, Multi-Tasking-Kernels, Konverter. ● **Vorschlag:** Wir sollten uns mal kennenlernen. 0821/571081.

 LOGITECH



L&W



Software Service GmbH

Alter Postweg 101, 8900 Augsburg

Telefon 0821/571081

WILKEN & SABELBERG



Software GbR

ENZ

EDV-BERATUNG GMBH


```

144:   if not trap then
145:     begin
146:       writeln;
147:       writeln(BEL, '++++ Fehler', nr:3, ': ', msg);
148:       repeat until KeyPressed;
149:     end;
150:     ErrorLevel := ErrorLevel + 1;
151:     ErrorNum := nr;
152:     ErrorMessage := msg;
153:   end;
154:
155:
156: procedure ClearError;
157:
158: ( Fehlerstatus zurücksetzen )
159:
160: begin
161:   trap := false;      ( Fehlermeldung anzeigen )
162:   ErrorLevel := 0;      ( Anzahl Fehler )
163:   ErrorNum := 0;        ( Fehlernummer )
164:   ErrorCode := 0;       ( Fehlercode )
165:   ErrorClass := 0;      ( Fehlerklasse )
166:   ErrorAction := 0;     ( vorgeschlagene Aktion )
167:   ErrorLocation := 0;   ( Bereich des Fehlers )
168:   ErrorMessage := 'OK'; ( Fehlertext )
169: end;
170:
171:
172:
173: ( ----- allgemeine I/O-Funktionen ----- )
174:
175:
176: function key: char;
177:
178: ( Tastatureingabe mit Echo )
179:
180: begin
181:   dos(1);
182:   key := chr(reg.al);
183: end;
184:
185:
186:
187: procedure DisplayChr(ch: char);
188:
189: ( Bildschirmausgabe )
190:
191: begin
192:   reg.dl := ord(ch);
193:   dos(2);
194: end;
195:
196:
197:
198: function SerialIn: char;
199:
200: ( Serielle Eingabe )
201:
202: begin
203:   dos(3);
204:   SerialIn := chr(reg.al);
205: end;
206:
207:
208:
209: procedure SerialOut(ch: char);
210:
211: ( Serielle Ausgabe )
212:
213: begin
214:   reg.dl := ord(ch);
215:   dos(4);
216: end;
217:
218:
219:
220: procedure Print(ch: char);
221:
222: ( Druckerausgabe )
223:
224: begin
225:   reg.dl := ord(ch);
226:   dos(5);
227: end;
228:
229:
230:
231: function GetKey: char;
232:
233: ( Direkte Tastatureingabe ohne Echo und CTRL-C )
234:
235: begin
236:   dos(7);
237:   GetKey := chr(reg.al);
238: end;
239:
240:
241:
242: function InKey: char;
243:
244: ( Direkte Tastatureingabe ohne Echo, mit CTRL-C )
245:
246: begin
247:   dos(8);
248:   InKey := chr(reg.al);
249: end;
250:
251:

```

```

252:
253: procedure FlushKeys;
254:
255: ( Tastaturpuffer löschen )
256:
257: begin
258:   reg.al := 255;
259:   dos(12);
260: end;
261:
262:
263:
264: ( ----- allgemeine Disk-Funktionen ----- )
265:
266:
267: type
268:   Laufwerk = (A, B, C, D, E, F);
269:   LaufwerksTyp = (
270:     unbekannt,
271:     HD, ( Festplatte )
272:     DS15, ( doppelseitig, 15 Sektoren/Spur )
273:     SS9, ( einseitig, 9 Sektoren/Spur )
274:     DS9, ( doppelseitig, 9 Sektoren/Spur )
275:     SS8, ( einseitig, 8 Sektoren/Spur )
276:     DS8, ( doppelseitig, 8 Sektoren/Spur )
277:   );
278:
279:
280:
281: function DriveName(drive: Laufwerk): char;
282:
283: ( Laufwerksnamen bilden )
284:
285: begin
286:   DriveName := chr(ord(drive) + 65);
287: end;
288:
289:
290:
291: procedure ResetDisk;
292:
293: ( Laufwerke zurücksetzen )
294:
295: begin
296:   dos(13);
297: end;
298:
299:
300:
301: procedure SelectDisk(drive: Laufwerk);
302:
303: ( Standardlaufwerk setzen )
304:
305: begin
306:   reg.dl := ord(drive);
307:   dos(14);
308: end;
309:
310:
311:
312: function CurrentDisk: Laufwerk;
313:
314: ( Standardlaufwerk feststellen )
315:
316: begin
317:   dos(25);
318:   CurrentDisk := Laufwerk(reg.al);
319: end;
320:
321:
322:
323: procedure SetDTA(Adresse: pointer);
324:
325: ( Disk Transfer Adresse setzen )
326: ( Standardeinstellung ist CS:80H )
327: ( Achtung ! Nur mit Vorsicht verwenden ! )
328:
329: begin
330:   reg.dx := ofs(Adresse);
331:   reg.ds := seg(Adresse);
332:   dos(26);
333: end;
334:
335:
336:
337: procedure GetDTA(var Adresse: pointer);
338:
339: ( Disk Transfer Adresse lesen )
340:
341: begin
342:   dos(47);
343:   Adresse := ptr(reg.es, reg.bx);
344: end;
345:
346:
347:
348: function DriveType(drive: Laufwerk): LaufwerksTyp;
349:
350: ( Laufwerkstyp ermitteln )
351:
352: var p: pointer;
353:
354: begin
355:   reg.al := ord(drive);
356:   dos(28);
357:   if reg.al = 255 then
358:     begin
359:       reg.ax := 15;

```



```

360:     error;
361: end;
362: p := ptr(reg.ds, reg.bx);
363: case p of
364:   $F8: DriveType := HD;
365:   $F9: DriveType := DS15;
366:   $FC: DriveType := SS9;
367:   $FD: DriveType := DS9;
368:   $FE: DriveType := SS8;
369:   $FF: DriveType := DS8;
370: else DriveType := unbekannt;
371: end;
372: end;
373:
374:
375:
376: function ClustersPerDrive(drive: Laufwerk): integer;
377:
378: ( Blocks pro Diskette feststellen )
379:
380: begin
381:   reg.dl := ord(drive) + 1;
382:   dos(54);
383:   if reg.ax = -1 then
384:     begin
385:       reg.ax := 15;
386:       error;
387:     end;
388:   ClustersPerDrive := reg.dx;
389: end;
390:
391:
392:
393: function SectorsPerCluster(drive: Laufwerk): integer;
394:
395: ( Sektoren pro Block feststellen )
396:
397: begin
398:   reg.dl := ord(drive) + 1;
399:   dos(54);
400:   if reg.ax = -1 then
401:     begin
402:       reg.ax := 15;
403:       error;
404:     end;
405:   SectorsPerCluster := reg.ax;
406: end;
407:
408:
409:
410: function BytesPerSector(drive: Laufwerk): integer;
411:
412: ( Bytes pro Sektor feststellen )
413:
414: begin
415:   reg.dl := ord(drive) + 1;
416:   dos(54);
417:   if reg.ax = -1 then
418:     begin
419:       reg.ax := 15;
420:       error;
421:     end;
422:   BytesPerSector := reg.cx;
423: end;
424:
425:
426:
427: function AvailableClusters(drive: Laufwerk): integer;
428:
429: ( freie Blöcke feststellen )
430:
431: begin
432:   reg.dl := ord(drive) + 1;
433:   dos(54);
434:   if reg.ax = -1 then
435:     begin
436:       reg.ax := 15;
437:       error;
438:     end;
439:   AvailableClusters := reg.bx;
440: end;
441:
442:
443:
444: ( ----- allgemeine DOS-Funktionen ----- )
445:
446:
447: type
448:   Versionstyp = record
449:     major: byte;
450:     minor: byte;
451:     serial: byte;
452:   end;
453:
454: var
455:   Version: Versionstyp;
456:
457:
458:
459: procedure SetInterrupt(Nummer: byte; Routine: pointer);
460:
461: ( Interruptvektor setzen )
462: ( Achtung ! Nur mit Vorsicht verwenden )
463:
464: begin
465:   reg.al := Nummer;
466:   reg.ds := seg(Routine);
467:   reg.dx := ofs(Routine);

```

```

468:   dos(37);
469: end;
470:
471:
472:
473: procedure GetInterrupt(Nummer: byte; var Routine: pointer);
474:
475: ( Interruptvektor lesen )
476:
477: begin
478:   reg.al := Nummer;
479:   dos(53);
480:   Routine := ptr(reg.es, reg.bx);
481: end;
482:
483:
484:
485: procedure GetCountryData(Code: integer; Data: Pointer);
486:
487: ( Zeiger auf landesabhängige Informationen holen )
488: ( Data muß auf 34 Bytes freien Speicher zeigen ! )
489: ( Achtung ! Nur mit Vorsicht verwenden ! )
490:
491: begin
492:   if Code < 255 then reg.al := Code
493:   else
494:     begin
495:       reg.al := 255;
496:       reg.bx := Code;
497:     end;
498:   reg.dx := ofs(Data);
499:   reg.ds := seg(Data);
500:   dos(56);
501:   if carry then error;
502: end;
503:
504:
505:
506: procedure SetCountryCode(Code: integer);
507:
508: ( Landescode setzen )
509:
510: begin
511:   if Code < 255 then reg.al := Code
512:   else
513:     begin
514:       reg.al := 255;
515:       reg.bx := Code;
516:     end;
517:   reg.dx := -1;
518:   dos(56);
519: end;
520:
521:
522:
523: procedure SetVerify(Status: boolean);
524:
525: ( Verify-Status setzen )
526:
527: begin
528:   reg.al := ord(Status);
529:   dos(46);
530: end;
531:
532:
533:
534: function VerifyStatus: boolean;
535:
536: ( Verify-Status abfragen )
537:
538: begin
539:   dos(84);
540:   VerifyStatus := reg.al = 1;
541: end;
542:
543:
544:
545: procedure SetBreak(Status: boolean);
546:
547: ( Break-Status (Control-C) setzen )
548:
549: begin
550:   reg.al := 1;
551:   reg.dl := ord(Status);
552:   dos(51);
553: end;
554:
555:
556:
557: function BreakStatus: boolean;
558:
559: ( Break-Status (Control-C) abfragen )
560:
561: begin
562:   reg.al := 0;
563:   dos(51);
564:   BreakStatus := boolean(reg.dl);
565: end;
566:
567:
568:
569: procedure GetVersion(Version: Versionstyp);
570:
571: ( MS-DOS-Version und Seriennummer ermitteln )
572:
573: begin
574:   dos(48);
575:   version.major := reg.al;
576:   version.minor := reg.ah;

```



```

577:   version.serial := reg.bh;
578: end;
579:
580:
581:
582:
583: { ----- Datums- und Zeitfunktionen ----- }
584:
585:
586:
587: type
588:   Wochentagstyp = (Sonntag, Montag, Dienstag, Mittwoch,
589:                   Donnerstag, Freitag, Samstag);
590:
591:   Monatstyp = (Januar, Februar, Maerz, April,
592:               Mai, Juni, Juli, August, September,
593:               Oktober, November, Dezember);
594:
595:   Datumstyp = record
596:     Jahr: 1980..2099;
597:     Monat: Monatstyp;
598:     Tag: 1..31;
599:     Wochentag: Wochentagstyp;
600:   end;
601:
602:   Zeittyp = record
603:     Stunde: 0..23;
604:     Minute: 0..59;
605:     Sekunde: 0..59;
606:     MilliSekunde: 0..99;
607:   end;
608:
609:
610: var
611:   Datum: Datumstyp;
612:   Zeit: Zeittyp;
613:
614:
615:
616: procedure GetDate(var Datum: Datumstyp);
617: { Datum abfragen }
618:
619:
620: begin
621:   dos(42);
622:   with Datum do
623:   begin
624:     Jahr := reg.cx;
625:     Monat := Monatstyp(reg.dh);
626:     Tag := reg.dl;
627:     Wochentag := Wochentagstyp(reg.al);
628:   end;
629: end;
630:
631:
632:
633:
634: procedure SetDate(Datum: Datumstyp);
635: { Datum setzen }
636:
637:
638: begin
639:   with Datum do
640:   begin
641:     reg.cx := Jahr;
642:     reg.dh := ord(Monat);
643:     reg.dl := Tag;
644:   end;
645:   dos(43);
646:   if reg.al = 255 then error;
647: end;
648:
649:
650:
651: procedure GetTime(var Zeit: Zeittyp);
652: { Uhrzeit lesen }
653:
654:
655: begin
656:   dos(44);
657:   with Zeit do
658:   begin
659:     Stunde := reg.ch;
660:     Minute := reg.cl;
661:     Sekunde := reg.dh;
662:     MilliSekunde := reg.dl;
663:   end;
664: end;
665:
666:
667:
668: procedure SetTime(Zeit: Zeittyp);
669: { Uhrzeit setzen }
670:
671:
672: begin
673:   with Zeit do
674:   begin
675:     reg.ch := Stunde;
676:     reg.cl := Minute;
677:     reg.dh := Sekunde;
678:     reg.dl := MilliSekunde;
679:   end;
680:   dos(45);
681:   if reg.al = 255 then error;
682: end;
683:
684:

```

```

685:
686: procedure GetFileDate(fh: FileHandle;
687:                      var Date: Datumstyp;
688:                      var Time: Zeittyp);
689: { Datum und Zeit einer Datei holen }
690:
691:
692: begin
693:   reg.al := 0;
694:   reg.bx := fh;
695:   dos(87);
696:   if carry then error;
697:   Time.MilliSekunde := 0;
698:   Time.Sekunde := reg.cx mod 32;
699:   Time.Minute := (reg.cx shr 5) mod 64;
700:   Time.Stunde := reg.ch shr 3;
701:   Date.Tag := reg.dh mod 32;
702:   Date.Monat := Monatstyp((reg.dh shr 5) mod 16);
703:   Date.Jahr := (reg.dh shr 1) + 1980;
704: end;
705:
706:
707:
708: procedure PutFileDate(fh: FileHandle;
709:                      Date: Datumstyp;
710:                      Time: Zeittyp);
711: { Datum und Zeit einer Datei setzen }
712:
713:
714: begin
715:   reg.al := 1;
716:   reg.bx := fh;
717:   with Time do
718:   begin
719:     reg.cx := (((Stunde shl 6) + Minute) shl 5) + Sekunde;
720:     reg.dh := (((Jahr - 1980) shl 4) + ord(Monat)) shl 5;
721:   end;
722:   dos(87);
723:   if carry then error;
724: end;
725:
726:
727: { ----- Directory-Funktionen ----- }
728:
729:
730:
731: type
732:   DirEntryType = record
733:     reserviert: array[0..20] of byte;
734:     Attribut: byte;
735:     Zeit: integer;
736:     Datum: integer;
737:     LaengeLo: integer;
738:     LaengeHi: integer;
739:     Name: array[1..13] of char;
740:   end;
741:
742: var
743:   DirEntry: DirEntryType;
744:
745:
746:
747: procedure MakeDir(Name: PathName);
748: { Subdirectory anlegen }
749:
750:
751: begin
752:   Name := Name + NUL;
753:   reg.ds := seg(Name[1]);
754:   reg.dx := ofs(Name[1]);
755:   dos(57);
756:   if carry then error;
757: end;
758:
759:
760:
761: procedure RemoveDir(Name: PathName);
762: { Subdirectory löschen }
763:
764:
765: begin
766:   Name := Name + NUL;
767:   reg.ds := seg(Name[1]);
768:   reg.dx := ofs(Name[1]);
769:   dos(58);
770:   if carry then error;
771: end;
772:
773:
774:
775: procedure ChangeDir(Name: PathName);
776: { Subdirectory wechseln }
777:
778:
779: begin
780:   Name := Name + NUL;
781:   reg.ds := seg(Name[1]);
782:   reg.dx := ofs(Name[1]);
783:   dos(59);
784:   if carry then error;
785: end;
786:
787:
788:
789: function FirstFile(Name: PathName; Attribut: integer): Path
790: Name;

```



```

791: ( ersten Directory-Eintrag suchen )
792:
793: var
794:   DTA: pointer;
795:   fn: PathName;
796:
797: begin
798:   GetDTA(DTA);
799:   Name := Name + NUL;
800:   reg.ds := seg(Name[1]);
801:   reg.dx := ofs(Name[1]);
802:   reg.cx := Attribut;
803:   dos(78);
804:   if carry then error;
805:   DirEntry := addr(DTA^);
806:   fn := DirEntry^.Name;
807:   fn[0] := chr(pos(NUL, fn) - 1);
808:   FirstFile := fn;
809: end;
810:
811:
812:
813: function NextFile: PathName;
814:
815: ( nächsten Directory-Eintrag suchen )
816:
817: var
818:   DTA: pointer;
819:   fn: PathName;
820:
821: begin
822:   GetDTA(DTA);
823:   dos(79);
824:   if carry then error;
825:   DirEntry := addr(DTA^);
826:   fn := DirEntry^.Name;
827:   fn[0] := chr(pos(NUL, fn) - 1);
828:   NextFile := fn;
829: end;
830:
831:
832:
833: function CurrentDir(Drive: Laufwerk): PathName;
834:
835: ( Pfadnamen des aktuellen Directory ermitteln )
836:
837: var
838:   Name: PathName;
839:
840: begin
841:   reg.dl := ord(Drive) + 1;
842:   reg.ds := seg(Name[1]);
843:   reg.si := ofs(Name[1]);
844:   dos(71);
845:   if carry then error;
846:   Name[0] := chr(pos(NUL, Name) - 1);
847:   CurrentDir := Name;
848: end;
849:
850:
851:
852: procedure MoveDir(AlterName, NeuerName: PathName);
853:
854: ( Datei und Pfad umbenennen )
855:
856: begin
857:   AlterName := AlterName + NUL;
858:   reg.ds := seg(AlterName[1]);
859:   reg.dx := ofs(AlterName[1]);
860:   NeuerName := NeuerName + NUL;
861:   reg.es := seg(NeuerName);
862:   reg.di := ofs(NeuerName);
863:   dos(86);
864:   if carry then error;
865: end;
866:
867:
868:
869: ( ----- Datei-Funktionen ----- )
870:
871:
872:
873: const
874:   _ReadWrite = 0;
875:   _ReadOnly = 1;
876:   _Hidden = 2;
877:   _System = 4;
878:   _Volume = 8;
879:   _SubDirectory = 16;
880:   _Archive = 32;
881:
882:
883:
884: procedure CreateFile(Name: PathName; Attribut: integer; var
      fh: FileHandle);
885:
886: ( Datei erzeugen )
887:
888: begin
889:   Name := Name + NUL;
890:   reg.ds := seg(Name[1]);
891:   reg.dx := ofs(Name[1]);
892:   reg.cx := Attribut;
893:   dos(60);
894:   if carry then error;
895:   fh := reg.ax;
896: end;
897:

```

```

898:
899:
900: procedure OpenFile(Name: PathName; AccessCode: byte; var fh
      : FileHandle);
901:
902: ( Datei öffnen )
903:
904: begin
905:   Name := Name + NUL;
906:   reg.ds := seg(Name[1]);
907:   reg.dx := ofs(Name[1]);
908:   reg.al := AccessCode;
909:   dos(61);
910:   if carry then error;
911:   fh := reg.ax;
912: end;
913:
914:
915:
916: procedure CloseFile(fh: FileHandle);
917:
918: ( Datei schließen )
919:
920: begin
921:   reg.bx := fh;
922:   dos(62);
923:   if carry then error;
924: end;
925:
926:
927:
928: function ReadFile(fh: FileHandle; Anzahl: integer; Puffer:
      pointer): integer;
929:
930: ( Anzahl Bytes in Puffer einlesen )
931:
932: begin
933:   reg.bx := fh;
934:   reg.cx := Anzahl;
935:   reg.dx := ofs(Puffer^);
936:   reg.ds := seg(Puffer^);
937:   dos(63);
938:   if carry then error;
939:   ReadFile := reg.ax;
940: end;
941:
942:
943:
944: function WriteFile(fh: FileHandle; Anzahl: integer; Puffer:
      pointer): integer;
945:
946: ( Anzahl Bytes aus Puffer in Datei schreiben )
947:
948: begin
949:   reg.bx := fh;
950:   reg.cx := Anzahl;
951:   reg.dx := ofs(Puffer^);
952:   reg.ds := seg(Puffer^);
953:   dos(64);
954:   if carry then error;
955:   WriteFile := reg.ax;
956: end;
957:
958:
959:
960: procedure DeleteFile(Name: PathName);
961:
962: ( Datei löschen )
963:
964: begin
965:   Name := Name + NUL;
966:   reg.ds := seg(Name[1]);
967:   reg.dx := ofs(Name[1]);
968:   dos(65);
969:   if carry then error;
970: end;
971:
972:
973:
974: procedure SetFileAttribute(Name: PathName; Attr: integer);
975:
976: ( Datei-Attribute setzen )
977:
978: begin
979:   Name := Name + NUL;
980:   reg.ds := seg(Name[1]);
981:   reg.dx := ofs(Name[1]);
982:   reg.cx := Attr;
983:   reg.al := 1;
984:   dos(67);
985:   if carry then error;
986: end;
987:
988:
989:
990: function FileAttribute(Name: PathName): integer;
991:
992: ( Datei-Attribute abfragen )
993:
994: begin
995:   Name := Name + NUL;
996:   reg.ds := seg(Name[1]);
997:   reg.dx := ofs(Name[1]);
998:   reg.al := 0;
999:   dos(67);
1000:   if carry then error;
1001:   FileAttribute := reg.cx;
1002: end;

```



```

1003:
1004:
1005:
1006: function DuplicateHandle(fh: FileHandle): FileHandle;
1007:
1008: ( zweite Dateinummer erzeugen )
1009:
1010: begin
1011:   reg.bx := fh;
1012:   dos(69);
1013:   if carry then error;
1014:   DuplicateHandle := reg.ax;
1015: end;
1016:
1017:
1018:
1019: procedure ForceDuplicateHandle(fh1, fh2: FileHandle);
1020:
1021: ( Dateinummern gleichschalten )
1022:
1023: begin
1024:   reg.bx := fh1;
1025:   reg.cx := fh2;
1026:   dos(70);
1027:   if carry then error;
1028: end;
1029:
1030:
1031:
1032: procedure CreateTempFile(Name: PathName; Attr: integer; var
      fh: FileHandle);
1033:
1034: ( temporäre Datei anlegen )
1035:
1036: begin
1037:   reg.cx := Attr;
1038:   Name := Name + NUL;
1039:   reg.ds := seg(Name[1]);
1040:   reg.dx := ofs(Name[1]);
1041:   dos(90);
1042:   if carry then error;
1043:   fh := reg.ax;
1044: end;
1045:
1046:
1047:
1048: procedure CreateNewFile(Name: PathName; Attr: integer; var
      fh: FileHandle);
1049:
1050: ( Neue Datei anlegen )
1051:
1052: begin
1053:   reg.cx := Attr;
1054:   Name := Name + NUL;
1055:   reg.ds := seg(Name[1]);
1056:   reg.dx := ofs(Name[1]);
1057:   dos(91);
1058:   if carry then error;
1059:   fh := reg.ax;
1060: end;
1061:
1062:
1063:
1064: ( ----- Speicherverwaltung ----- )
1065:
1066:
1067:
1068: function AllocMem(Paragraphs: integer): integer;
1069:
1070: ( Speicher anfordern )
1071:
1072: begin
1073:   reg.bx := Paragraphs;
1074:   dos(72);
1075:   if carry then error;
1076:   AllocMem := reg.ax;
1077: end;
1078:
1079:
1080:
1081: procedure FreeMem(Adress: integer);
1082:
1083: ( Speicher freigeben )
1084:
1085: begin
1086:   reg.es := Adress;
1087:   dos(73);
1088:   if carry then error;
1089: end;
1090:
1091:
1092:
1093: procedure ChangeMemSize(Adress, Size: integer);
1094:
1095: ( Speicherblockgröße ändern )
1096:
1097: begin
1098:   reg.es := Adress;
1099:   reg.bx := Size;
1100:   dos(74);
1101:   if carry then error;
1102: end;
1103:
1104:
1105:
1106: procedure KeepProgram(Size: integer; Code: byte);
1107:
1108: ( Programm speicherresident machen und beenden )

```

```

1109: ( Achtung ! Nur mit Vorsicht verwenden ! )
1110:
1111: begin
1112:   reg.al := Code;
1113:   reg.dx := Size;
1114:   dos(49);
1115: end;
1116:
1117:
1118:
1119: procedure EndProcess(Code: byte);
1120:
1121: ( Prozeß mit Return-Code abbrechen )
1122:
1123: begin
1124:   reg.al := Code;
1125:   dos(76);
1126: end;
1127:
1128:
1129: function GetReturnCode: integer;
1130:
1131: ( Return-Code holen )
1132:
1133: begin
1134:   dos(77);
1135:   GetReturnCode := reg.ax;
1136: end;
1137:
1138:
1139:
1140: procedure GetAllocationStrategy(var n: integer);
1141:
1142: ( Speicherbelegungs-Strategie melden )
1143:
1144: begin
1145:   reg.al := 0;
1146:   dos(88);
1147:   if carry then error;
1148:   n := reg.ax;
1149: end;
1150:
1151:
1152:
1153: procedure SetAllocationStrategy(n: integer);
1154:
1155: ( Speicherbelegungs-Strategie setzen )
1156:
1157: begin
1158:   reg.al := 1;
1159:   reg.bx := n;
1160:   dos(88);
1161:   if carry then error;
1162: end;
1163:
1164:
1165:
1166: function GetPSP: integer;
1167:
1168: ( Segment-Adresse des PSP holen )
1169:
1170: begin
1171:   dos(98);
1172:   GetPSP := reg.bx;
1173: end;
1174:
1175:
1176:
1177: ( ----- Ein-/Ausgabesteuerung für Geräte ----- )
1178:
1179:
1180:
1181: procedure GetDeviceData(Device: FileHandle; var Data: integer);
1182:
1183: ( Gerätedatenwort holen )
1184:
1185: begin
1186:   reg.bx := Device;
1187:   reg.al := 0;
1188:   dos(68);
1189:   if carry then error;
1190:   Data := reg.dx;
1191: end;
1192:
1193:
1194:
1195: procedure SetDeviceData(Device: FileHandle; Data: integer);
1196:
1197: ( Gerätedatenwort setzen )
1198:
1199: begin
1200:   reg.bx := Device;
1201:   reg.dx := Data;
1202:   reg.al := 1;
1203:   dos(68);
1204:   if carry then error;
1205: end;
1206:
1207:
1208:
1209: function SendControlData(Device: FileHandle; Data: pointer;
      Anzahl: integer): integer;
1210:
1211: ( Kontrolldaten senden )
1212:
1213: begin
1214:   reg.bx := Device;

```

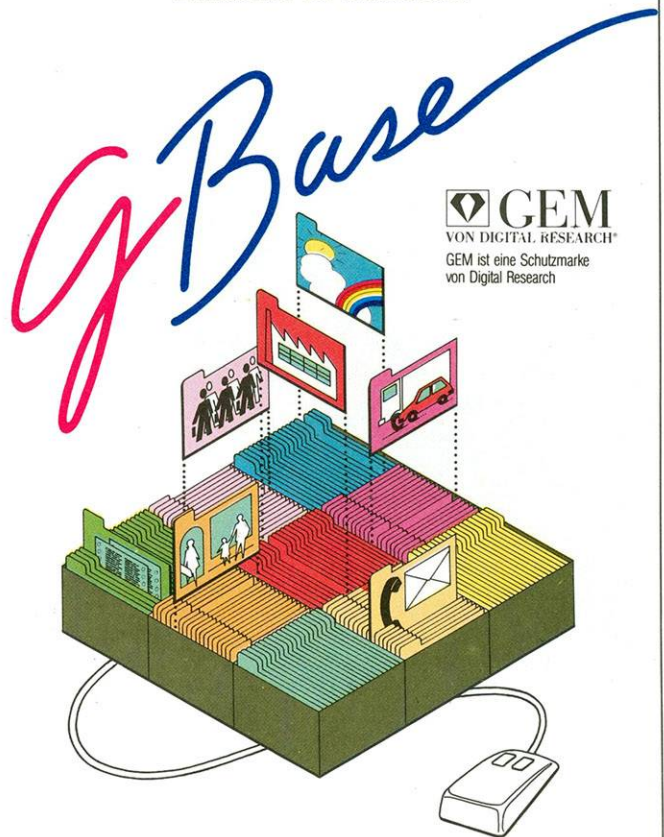


```

1215:  reg.dx := ofs(Data^);
1216:  reg.ds := seg(Data^);
1217:  reg.cx := Anzahl;
1218:  reg.al := 2;
1219:  dos(68);
1220:  if carry then error;
1221:  SendControlData := reg.ax;
1222: end;
1223:
1224:
1225:
1226: function ReceiveControlData(Device: FileHandle; Data: point
er; Anzahl: integer): integer;
1227:
1228: ( Kontrolldaten empfangen )
1229:
1230: begin
1231:  reg.bx := Device;
1232:  reg.dx := ofs(Data^);
1233:  reg.ds := seg(Data^);
1234:  reg.cx := Anzahl;
1235:  reg.al := 3;
1236:  dos(68);
1237:  if carry then error;
1238:  ReceiveControlData := reg.ax;
1239: end;
1240:
1241:
1242:
1243: function SendControlDataBlock(Device: Laufwerk; Data: point
er; Anzahl: integer): integer;
1244:
1245: ( Kontrolldaten an Laufwerk senden )
1246:
1247: begin
1248:  reg.bl := ord(Device) + 1;
1249:  reg.dx := ofs(Data^);
1250:  reg.ds := seg(Data^);
1251:  reg.cx := Anzahl;
1252:  reg.al := 4;
1253:  dos(68);
1254:  if carry then error;
1255:  SendControlDataBlock := reg.ax;
1256: end;
1257:
1258:
1259:
1260: function ReceiveControlDataBlock(Device: Laufwerk; Data: po
inter; Anzahl: integer): integer;
1261:
1262: ( Kontrolldaten von Laufwerk empfangen )
1263:
1264: begin
1265:  reg.bl := ord(Device) + 1;
1266:  reg.dx := ofs(Data^);
1267:  reg.ds := seg(Data^);
1268:  reg.cx := Anzahl;
1269:  reg.al := 5;
1270:  dos(68);
1271:  if carry then error;
1272:  ReceiveControlDataBlock := reg.ax;
1273: end;
1274:
1275:
1276:
1277: function DeviceInputStatus(Device: FileHandle): boolean;
1278:
1279: ( Geräte-Eingabestatus feststellen )
1280:
1281: begin
1282:  reg.bx := Device;
1283:  reg.al := 6;
1284:  dos(68);
1285:  if carry then error;
1286:  DeviceInputStatus := reg.al <> 0;
1287: end;
1288:
1289:
1290:
1291: function DeviceOutputStatus(Device: FileHandle): boolean;
1292:
1293: ( Geräte-Ausgabestatus feststellen )
1294:
1295: begin
1296:  reg.bx := Device;
1297:  reg.al := 7;
1298:  dos(68);
1299:  if carry then error;
1300:  DeviceOutputStatus := reg.al <> 0;
1301: end;
1302:
1303:
1304:
1305: function is_Diskette(Drive: Laufwerk): boolean;
1306:
1307: ( feststellen, ob Laufwerk mit Disketten arbeitet )
1308:
1309: begin
1310:  reg.bx := ord(Drive) + 1;
1311:  reg.al := 8;
1312:  dos(68);
1313:  if carry then error;
1314:  is_Diskette := reg.ax = 0;
1315: end;
1316:
1317:
1318:
1319: ( ----- Ende Bibliothek MSDOS.INC ----- )

```

Einfacher zu bedienen



Die relationale GEM Datenbank.

- **GEM Benutzeroberfläche.** Einfache Dateneingabe und -abfrage
- **Voll relational.** Fünf Dateien sind miteinander verknüpfbar
- **Virtuelle Speichertechnik.** Keine Begrenzung durch die Speicherkapazität Ihres Rechners
- **Selbstgestaltete Eingabeformulare.** Nach Ihren ganz persönlichen Anforderungen
- **Report-System.** Berichte in jeder gewünschten Form
- **Ergebnis-Transfer.** Woher und wohin Sie wollen
- **ab DM 395,-** unverbindl. Preisempfehlung

Sie wollen sich persönlich überzeugen? Wir schicken Ihnen detaillierte Unterlagen und laden Sie zu einer Vorführung ein.

Ihr Name: _____

Beruf/Funktion: _____

bei Firma: _____

Anschrift: _____

Telefon: _____

DOS-SH 11/87



SPI

SOFTWARE PRODUCTS INTERNATIONAL
Rosenkavalierplatz 14, D-8000 München 81, Telefon 089/92 10 06-0, Telex (17) 897174

Markt & Technik Verlag AG

Geschäftsbereich Software-Verlag
Hans-Pinsel-Straße 2
8013 Haar
089/4 61 30

SOFTSAL

CSSE Exclusive Distributor
Hansastraße 15
8000 München 21
089/57 60 31

BSP T. K. Krug

EDV-Beratung-Systeme
Weißenburgstraße 49
8400 Regensburg
0941/79 20 14

EDTZ

Hard & Softwarebüro Dotzauer
Haidgraben 3
8012 Ottobrunn
089/60 980 95



Im Dickicht des Assembler-code: Programmieren in Maschinensprache

Maschinensprache ist die einzige Sprache, die der Computer auf Anhieb versteht. Deshalb arbeiten Maschinensprache-Programme auch besonders schnell. Wie solche Programme funktionieren und was Maschinensprache überhaupt ist, das lesen Sie hier.

Die Mehrheit der PC-Besitzer arbeitet mit einer höheren Programmiersprache wie Basic oder Pascal. Der Prozessor – die Schaltzentrale des PC – versteht jedoch nur Maschinensprache. Damit Basic- und Pascal-Programme trotzdem auf dem Computer laufen, müssen diese Programme in Maschinensprache übersetzt werden.

Um Programme von Hochsprache in Maschinensprache zu übersetzen, gibt es das Interpreter- und Compiler-Verfahren. Ein Interpreter übersetzt ein Programm während der Ausführ-

ung Befehl für Befehl. Weil beim Ausführen eines Befehls neben der Übersetzung und den gesamten Fehlerabfang-Routinen auch noch die vom jeweiligen Befehl benötigten Betriebssystem-Routinen aufgerufen werden müssen, ist die erzielte Verarbeitungsgeschwindigkeit in der Regel gering.

Ein Compiler übersetzt dagegen das Programm nach Beseitigung aller Fehler an einem Stück in einen Maschinensprache-ähnlichen (maschinennahen) Code, der bereits ziemlich schnell ist. Die Geschwindigkeit von reiner Maschinensprache erreicht er jedoch nicht.

Die Situation zwischen Hochsprache und Maschinensprache ist mit der eines Deutschen, der nur deutsch spricht, und eines Engländers, der ausschließlich Englisch spricht, vergleichbar. Wenn der Deutsche mit dem Engländer sprechen möchte, be-

nötigt er einen Dolmetscher, der seine Worte ins Englische übersetzt.

Es leuchtet ein, daß die Kommunikation zwischen dem Deutschen und dem Engländer über den Dolmetscher viel Aufwand, eventuell einige Mißverständnisse und vor allem Zeit kostet. Einfacher wäre es, wenn sich zwei Engländer direkt unterhalten würden, weil sich dann der Dolmetscher und damit die Zeit, die das Übersetzte beansprucht, einsparen ließe.

Entsprechendes gilt für den Computer. Auch beim Ablauf von Programmen läßt sich viel Zeit sparen, wenn diese nicht erst von einem Dolmetscher (Interpreter oder Compiler) in Maschinensprache übersetzt werden müssen. Ist ein Programm von vornherein in Maschinensprache geschrieben, so kann der Prozessor dieses Programm direkt verarbeiten. Geschwindigkeitssteigerungen gegen-

über Hochsprachen-Programmen um den Faktor 10 bis 100 sind dann keine Seltenheit.

Ein Maschinensprache-Programm besteht wie jedes andere Programm auch aus einzelnen Befehlen, den Maschinensprache-Befehlen. Doch der Computer, genauer der Prozessor, kann die Maschinensprache-Befehle direkt verarbeiten, während ihm die Befehle von Hochsprachen ohne Übersetzung unverständlich bleiben. Im PC arbeitet entweder der Prozessor 8086 oder 8088. Da beide Prozessoren für den Maschinensprache-Programmierer identisch sind, werden sie künftig als 8086/8088 zusammengefaßt. Der Beitrag »Der Prozessor des PC« in dieser Ausgabe gibt erschöpfend Auskunft über Aufbau und Arbeitsweise des 8086/8088.

Der 8086/8088 versteht 120 verschiedene Maschinensprache-Befehle. Jeder Befehl wird in einem Maschinensprache-Programm durch eine Zahl – den Maschinencode – repräsentiert. Der Maschinencode kann je nach Komplexität des Befehls ein oder zwei Byte lang sein.

Operanden, die für arithmetische, logische oder sonstige Operationen benötigt werden, sind in Maschinensprache-Programmen ebenfalls als Zahlen abgelegt. Auch die Operanden können ein oder zwei Byte lang sein. Weil ein Befehl maximal zwei Operanden haben darf, reicht damit die Länge der Maschinensprache-Befehle des 8086/8088 von ein bis sechs Byte.

Befehle und Operanden werden zwar gemeinsam im Programm als Zahlen abgelegt, aber der Prozessor kann dennoch Befehle und Operanden voneinander unterscheiden, weil er die erste Zahl immer als Maschinencode für einen Befehl behandelt. Der Maschinencode gibt an, ob kein, ein oder zwei Operanden folgen. Dadurch erkennt der Prozessor auch, welche Zahl den nächsten Maschinencode bildet.

So stellen beispielsweise der ein Byte lange Maschinencode 184 und die zwei Byte lange Zahl 0 einen gültigen Maschinensprache-Befehl mit einem Operanden dar, sofern die drei Byte im Speicher in aufeinanderfolgenden Speicherzellen abgelegt sind.

Der Maschinencode 184 weist den Prozessor an, den Wert, der in den beiden folgenden Speicherzellen abgelegt ist, in das Register AX zu laden. Weil nun die folgenden beiden Speicherzellen den Wert 0 enthalten, wird das AX-Register folglich auf 0 gesetzt. Ein Register ist ein schneller Zwischenspeicher, der sich direkt im

Prozessor befindet und damit einen zeitaufwendigen Speicherzugriff überflüssig macht.

Da sich in den Zahlenkolonnen von längeren Maschinensprache-Programmen auf die Dauer kein Mensch zurechtfinden kann, hat sich Intel, der Hersteller des 8086/8088, für seinen Prozessor sinnvolle Kurzbezeichnungen einfallen lassen, die anstelle der Zahlenfolgen verwendet werden. Um den 8086/8088 zu veranlassen, den Wert 0 in das AX-Register zu schreiben, muß der Programmierer somit nicht mehr 184 und 0 angeben, sondern darf stattdessen `mov ax,0` schreiben. Die Kurzbezeichnung »mov« steht für das englische »to move« und bedeutet »bewegen«.

Assemblerbefehle statt Maschinencode erleichtern die Programmierung in Maschinensprache

Diese Kurzbezeichnungen für die Maschinensprachebefehle heißen Assemblerbefehle. Der Beitrag »Die Assemblerbefehle des PC« in dieser Ausgabe stellt sämtliche Assemblerbefehle des 8086/8088 mit allen erlaubten Kombinationen und dem dazu passenden Maschinencode vor. Sie werden jetzt vielleicht einwenden, daß durch die Assemblerbefehle doch wieder ein Übersetzungsvorgang nötig ist, nämlich von Assemblersprache in Maschinensprache. Sie haben recht, doch im Gegensatz zu Hochsprachen-Befehlen lassen sich die Assemblerbefehle 1:1 in Maschinensprache übersetzen, so daß keine Übersetzungsverluste auftreten. Ein Programm, das Assemblerbefehle in Maschinencode übersetzt, heißt – wie soll es auch anders sein – Assembler.

Die Assemblersprache – und damit natürlich auch die Maschinensprache – greift direkt auf den Speicher des Computers zu. Nun ist dieser Speicher allerdings so aufgebaut, daß er nur zwischen zwei elektrischen Zuständen unterscheiden kann. Ein Speicherbaustein kennt nur die beiden Zustände »Spannung ein« und »Spannung aus«, die auch als logische Zustände 1 und 0 bezeichnet werden.

Aus diesem Grund ist es zum Erlernen der Assemblersprache unumgänglich, sich mit einer vom Dezimalsystem abweichenden Methode der Zahlendarstellung zu beschäftigen: dem Binär- oder Dualsystem. Das Dezimalsystem funktioniert nach einem recht einfachen Prinzip: Jede

Stelle einer Dezimalzahl kann zehn verschiedene Werte (0 bis 9) annehmen und stellt damit das Vielfache einer bestimmten Zehnerpotenz dar. Die letzte Ziffer zeigt immer das Vielfache von 10^0 an, die vorletzte Ziffer das Vielfache von 10^1 , und so weiter. Die Dezimalzahl 1234 setzt sich beispielsweise wie folgt zusammen:

$$\begin{aligned} 1234 &= 1000 + 200 + 30 + 4 \\ &= 1 \times 10^3 \\ &+ 2 \times 10^2 \\ &+ 3 \times 10^1 \\ &+ 4 \times 10^0 \\ &= 1234 \end{aligned}$$

Das Binärsystem arbeitet nach dem gleichen Prinzip, doch können hier die Stellen einer Zahl nur zwei verschiedene Werte (0 und 1) annehmen, so daß jede Stelle das Vielfache einer bestimmten Zweierpotenz darstellt. Die letzte Ziffer zeigt immer das Vielfache von 2^0 an, die vorletzte Ziffer das Vielfache von 2^1 , und so weiter.

Die Dezimalzahl 1234 wird im Binärsystem wie folgt dargestellt:

$$\begin{aligned} 1234 &= 1024 \\ &+ 128 \\ &+ 64 \\ &+ 16 \\ &+ 2 \\ &= 1 \times 2^{10} \\ &+ 0 \times 2^9 \\ &+ 0 \times 2^8 \\ &+ 1 \times 2^7 \\ &+ 1 \times 2^6 \\ &+ 0 \times 2^5 \\ &+ 1 \times 2^4 \\ &+ 0 \times 2^3 \\ &+ 0 \times 2^2 \\ &+ 1 \times 2^1 \\ &+ 0 \times 2^0 \\ &= 10011010010_{\text{bin}} \end{aligned}$$

Zum besseren Verständnis eine Gegenüberstellung der 10 ersten Dezimal- und Binärzahlen:

dezimal 0	= binär 0000
dezimal 1	= binär 0001
dezimal 2	= binär 0010
dezimal 3	= binär 0011
dezimal 4	= binär 0100
dezimal 5	= binär 0101
dezimal 6	= binär 0110
dezimal 7	= binär 0111
dezimal 8	= binär 1000
dezimal 9	= binär 1001

Versuchen Sie einmal zur Probe den dezimalen Wert 127 in eine Binärzahl und den Wert 1100100bin in eine Dezimalzahl umzurechnen.

Binärzahlen haben den Vorteil, daß sich mit Ihnen die kleinste Informationseinheit, das Bit, darstellen läßt. Doch die Binärdarstellung hat auch einen Nachteil. Wenn Sie den dezimalen Wert 1234 mit seinem binären Gegenwert 10011010010 vergleichen, stellen Sie fest, daß Binärwerte mit zunehmender Größe auch in die Länge wachsen, so daß sie schnell »unhandlich« werden.

**Drei Zahlensysteme
in der Gegenüberstellung**

Dezimal	Binär	Hexadezimal
0	00000	00
1	00001	01
2	00010	02
3	00011	03
4	00100	04
5	00101	05
6	00110	06
7	00111	07
8	01000	08
9	01001	09
10	01010	0A
11	01011	0B
12	01100	0C
13	01101	0D
14	01110	0E
15	01111	0F
16	10000	10
17	10001	11
18	10010	12
19	10011	13

Die ersten 20 Zahlen des Dezimal-, Binär- und Hexadezimalsystems

Abhilfe schafft das Hexadezimalsystem, das vier Binärziffern zu einer hexadezimalen Ziffer zusammenfaßt, indem jede Stelle einer Hexadezimalzahl $2^4 = 16$ verschiedene Zustände (0 bis 9 und A bis F) annehmen kann. Dadurch lassen sich 8-Bit-Werte mit zwei und 16-Bit-Werte mit nur vier hexadezimalen Stellen darstellen. Die dezimale Zahl 1234 wird beispielsweise zu dem kompakten Wert 4D2hex.

Tabelle 1 zeigt eine Gegenüberstellung der 20 ersten Dezimal-, Binär- und Hexadezimalzahlen. Jetzt dürfte es Ihnen auch nicht mehr schwer fallen, die Dezimalzahl 127 und die Binärzahl 1100100bin in das Hexadezimalsystem umzurechnen. Zur Umrechnung von binär in hexadezimal fassen Sie einfach, von hinten beginnend, je vier Binärziffern zu einer hexadezimalen Ziffer zusammen. Der Binärwert 1011100 ergibt beispielsweise die Ziffern 0101 und 1100, so daß das hexadezimale Ergebnis 5Chex lautet.

Der 8086/8088-Prozessor besitzt einen reichhaltigen Registersatz

Der Aufbau des 8086/8088 bestimmt nämlich die Art der Befehle, sowie die Anzahl der Register und Adressierungsarten für Speicherzellen, die dem Programmierer beim Schreiben

seiner Assemblerprogramme zur Verfügung stehen.

Der 8086/8088 besitzt acht 16-Bit-Datenregister AX, BX, CX, DX, SI, DI, SP und BP. Die Register AX bis DX lassen sich zusätzlich in je zwei 8-Bit-Datenregister AL und AH, BL und BH, CL und CH sowie DL und DH aufteilen. Für die Segmentadressierung (eine besondere Form der Adressierung beim 8086/8088) stehen die vier 16-Bit-Segmentregister CS, DS, ES und SS bereit. Das 16-Bit-Statusregister PSW zeigt den Zustand der Statusbits an und der 16-Bit-Befehlszeiger IP weist jeweils auf den nächsten zu bearbeitenden Befehl im Speicher des Computers.

Auf den ersten Blick fällt auf, daß alle oben genannten Register 16 Bit breit sind. Mit einem 16-Bit-Wert lassen sich jedoch nur $2^{16} = 65536$ Speicherstellen adressieren (64 KByte). Dagegen beträgt der Speicherbereich des 8086/8088 1 MByte. Wie kann der 8086/8088 nun den gesamten Speicherbereich adressieren?

Der 8086/8088 benutzt dazu die Segmentregister, die die Startadresse (Basis) für ein Speichersegment liefern. Die 16-Bit-Adresse eines Registers gibt damit nur die Speicherposition innerhalb dieses Segments an und wird deshalb auch als Offset bezeichnet.

Zum Glück übernimmt der Assembler die korrekte Adressierung durch die Segmentregister. Der Programmierer muß in einem Assemblerprogramm lediglich definieren, welcher Teil eines Programms zu ein und demselben Segment gehört. Die Berechnung der Adressen für die Segmentregister erledigt dann der Assembler.

Nach so viel Theorie sollen Sie Ihr erstes Assemblerprogramm schreiben. Für die Eingabe eines Assemblerprogramms benötigen Sie lediglich einen Editor, der ASCII-Texte erzeugen kann. Dies ist beispielsweise ein Textverarbeitungsprogramm wie »Word«, das MS-DOS-Programm »Edlin« oder das Schneider PC-Programm »Rped«. Das Listing zeigt Ihr erstes Assemblerprogramm »Print«, das den gleichnamigen Basic-Befehl simuliert, indem es einen kurzen Text auf dem Bildschirm ausgibt. Tippen Sie dieses Listing einfach Buchstabe für Buchstabe ab und speichern Sie es unter dem Namen »print.asm« auf Diskette.

Wenn Sie noch nie mit Assemblerprogrammen zu tun hatten, wird Ihnen das Listing kaum etwas sagen. Deshalb finden Sie das Programm im folgenden detailliert erklärt. Und

weil Print ein typisches Assemblerprogramm ist, lernen Sie dabei auch gleich den allgemeinen Aufbau eines solchen Assemblerprogramms kennen.

Allgemein besteht der Text von Assemblerprogrammen aus vier Spalten. Die erste Spalte enthält die Namen von Sprungmarken (Labels) und Variablen. Marken müssen immer von einem Doppelpunkt gefolgt sein. In der zweiten Spalte stehen die Assemblerbefehle und Pseudobefehle. Pseudobefehle sind Anweisungen, die nur für den Assembler gelten und nicht in Maschinencode übersetzt werden. Das Listing Print enthält einige Pseudobefehle, die noch erläutert werden.

Die dritte Spalte des Programmtextes enthält die Bezeichnungen von Registern und Operanden für die Assembler- und Pseudobefehle. Die vierte und letzte Spalte enthält Kommentare, die durch ein Semikolon gekennzeichnet sind und die der Assembler ignoriert. Diese Kommentare dürfen auch am Anfang einer Zeile stehen, sofern sie von einem Semikolon angeführt werden.

In der ersten Zeile von Print wird mit dem Pseudobefehl »title« der Programmname definiert. Darauf folgen drei Kommentarzeilen, die die Funktion des Programmes erläutern. Mit dem Pseudobefehl »assume« wird den Segmentregistern CS und DS das (einzige) Segment »code« zugeordnet. Die Segmentregister ES und SS werden im Programm Print nicht benötigt und sind deshalb auch nicht aufgeführt.

Der Pseudobefehl »segment« definiert den Segmentstart von »code«. Mit dem Pseudobefehl »org« läßt sich die Startadresse des Programms innerhalb dieses Segments festlegen. In diesem Fall wurde die Adresse 100hex gewählt, weil das Programm dadurch als Datei mit der Endung »com« lauffähig ist. Print ist als Com-Datei angelegt, weil dies den Programmaufbau in Assemblersprache vereinfacht und die Programm länge dem Pseudobefehl »org« läßt sich die Startadresse des Programms innerhalb dieses Segments festlegen. In diesem Fall wurde die Adresse 100hex gewählt, weil das Programm dadurch als Datei mit der Endung »com« lauffähig ist. Print ist als Com-Datei angelegt, weil dies den Programmaufbau in Assemblersprache vereinfacht und die Programm länge minimiert. Den genauen Unterschied zwischen Dateien mit der Endung »com« und Dateien mit der Endung

»exe« finden Sie im zweiten Teil dieses Kurses.

»start:« ist die erste und einzige Marke im Programm. Sie bezieht sich auf den Pseudobefehl »end« und markiert den Startpunkt des Programmes, denn der einem End-Befehl folgende Name definiert jeweils die Sprungmarke, die beim Start eines Programms aufgerufen wird.

Der erste echte Assemblerbefehl von Print ist der Sprungbefehl »jmp«. Dieser Befehl überspringt den Datenbereich, der bei Com-Dateien am Programmstart stehen muß und ruft den eigentlichen Anfang des Programms auf.

Im Datenbereich von Print wird der Text der Variablen »text« über den Pseudobefehl »db« (define byte) definiert. Selbstverständlich können Sie hier auch einen anderen Text einsetzen. Nur das Dollarzeichen dürfen Sie nicht verändern, denn sonst findet die MS-DOS-Funktion 09hex von Print das Ende der Zeichenkette nicht. Die dezimalen Werte 10 und 13 sind die ASCII-Zeichen für Zeilenvorschub und Wagenrücklauf. Dadurch wird vor und nach dem Text je eine Leerzeile ausgegeben.

Mit dem ersten Mov-Befehl beginnt das eigentliche Programm. Hier wird durch den Pseudobefehl »offset« die Adresse der Variablen »text«, die die auszugebende Zeichenkette enthält, in das DX-Register geschrieben. Würde der Pseudobefehl »offset« fehlen, so würde nicht die Adresse, sondern das erste Zeichen der Variablen »text« selbst in das DX-Register geladen.

Anschließend wird durch den Wert 09hex im AH-Register und über den Software-Interrupt 21hex eine MS-DOS-Funktion aufgerufen, die die in »text« enthaltene Zeichenkette auf den Bildschirm ausgibt. Der Wert in DX teilt der MS-DOS-Funktion 09hex die Startadresse dieser Zeichenkette mit und das Dollarzeichen kennzeichnet das Ende der Zeichenkette.

Die Betriebssystem-Routinen des PC lassen sich über Software-Interrupts aufrufen

Auf die Befehlskombination

```
mov ah,xxh
```

```
int 21h
```

werden Sie noch öfters stoßen. Dies

ist nämlich ein eleganter Weg, die Betriebssystemroutinen von MS-DOS aufzurufen. Diese Routinen lassen sich über den Software-Interrupt 21hex ansprechen. Durch den Wert im AH-Register wird die Funktion »xx« ausgewählt. Mehr über diese Art des Betriebssystem-Aufrufs erfahren Sie in dem Beitrag »MS-DOS-Funktionen in Assembler« in dieser Ausgabe.

Auch das Programmende von Print führt der Software-Interrupt 21hex herbei. Diesmal wird die Funktion 4Chex aufgerufen. Über diese Funktion kann das Programm bequem in die MS-DOS-Kommandoebene zurückkehren. Der Pseudobefehl »ends« gibt das Ende des Segments »code« an und der Befehl »end« definiert das Programmende.

Mit dem Programm Print haben Sie gleich die wichtigsten Pseudobefehle eines Assemblers und ihre Anwendung kennengelernt. Doch wo ist nun der Assembler, der zum Übersetzen des Listings in Maschinencode notwendig ist?

Zum einen finden Sie auf der Data-box zu dieser Ausgabe einen kompletten Assembler mit Anleitung zum Abtippen und zum anderen können Sie den Microsoft-Makroassembler MASM benutzen, der unter den Assemblern die weiteste Verbreitung erlangt hat.

Damit Sie sich nicht erst in die Bedienung des MASM einarbeiten müssen, schreiben Sie sich am einfachsten eine Stapeldatei, die ein Assemblerprogramm automatisch übersetzt. Geben Sie dazu in der Kommandoebene folgendes ein:

```
copy con asm.bat
masm %1 ;
link %1 ;
exe2bin %1 %1.com
```

Anschließend betätigen Sie die Enter-Taste, die Taste [F6] und nochmals die Enter-Taste. Damit besitzen Sie eine kleine Stapeldatei, die Ihre Assemblerlistings auf Befehl übersetzt.

Sorgen Sie dafür, daß sich die oben beschriebene Stapeldatei, der MASM, der Linker »link« des MASM, der MS-DOS-Befehl »exe2bin« und das Assemblerprogramm Print auf der gleichen Diskette befinden. Möchten Sie nun das

Programm Print übersetzen, so geben Sie einfach

```
asm print
```

ein, und den Rest besorgt die Stapeldatei.

Zuerst erzeugt der MASM aus dem Assemblerprogramm ein Objektmodul, dann formt der Linker das Modul in eine Datei mit der Endung »exe« um, und zuletzt wandelt »exe2bin« die Exe-Datei in eine Datei mit der Endung »com« um. Ist die Stapeldatei beendet, so finden Sie im Verzeichnis Ihrer Diskette die beiden neuen Dateien »print.exe« und »print.com«. Die Datei »print.exe« können Sie löschen, weil sie nur eine Zwischenstufe darstellt und nicht korrekt funktioniert. »print.com« ist dagegen das Endergebnis unserer Bemühungen und läßt sich durch Eingabe von

```
print
```

starten. Sollte die Textausgabe von Print nicht funktionieren, dann müssen Sie die oben beschriebenen Schritte nochmals sorgfältig nachvollziehen. Haben Sie das Listing richtig abgetippt und unter dem Namen »print.asm« auf Diskette gespeichert? Stimmt der Inhalt von »asm.bat« mit dem oben beschriebenen Befehlen überein? Sind alle notwendigen Dateien auf der Diskette vorhanden? Sicherlich brummt Ihnen jetzt der Kopf von so vielen neuen Informationen. Lehnen Sie sich deshalb etwas zurück und lassen Sie all dies auf sich einwirken. Wenn Sie Lust haben, lösen Sie die Aufgaben und probieren Sie das beschriebene Assemblerprogramm aus. Lesen Sie sich auch die anderen Beiträge, auf die in diesem Artikel verwiesen wird, durch.

Danach sind Sie für den zweiten Teil dieses Kurses bereit, der Sie tiefer in die Geheimnisse der Maschinen- und Assemblersprache einführen wird.

(Martin Althaus/Markus Zietlow/ma)

Auflösung:
Der dezimale Wert 127 entspricht der Binärzahl 1111111 und dem Hexadezimalwert 7Fhex. Der Wert 1100100bin ist gleichbedeutend mit der Dezimalzahl 100 und dem Hexadezimalwert 64hex.

```
0: title print ;Programmname: »print«
1:
2: ; Dieses Programm gibt auf dem Bildschirm
3: ; eine Zeichenkette aus und kehrt darauf
4: ; in die MS-DOS-Kommandoebene zurück
5:
6: assume cs:code,ds:code ;Segment »code« ist für die
7: ; Segmentregister CS und DS
8: ; vorgesehen
9: code segment ;Start von Segment »code«
10: org 100h ;Programm startet ab 100hex
11: ;((für COM-Dateien erforderlich))
12: start: jmp anfang ;Sprung zum Programmstart
13:
14: ; Datenbereich:
15:
```

```
16: text db 13,10,"Mein erstes Assemblerprogramm",13,10,"$"
17:
18: ; Programmbereich:
19:
20: anfang: mov dx,offset text ;Adresse der Zeichenkette
21: mov ah,09h ;Zeichenkette über Interrupt 2
1hex,
22: int 21h ; Funktion 09hex auf den
23: ; Bildschirm ausgeben
24: mov ah,4Ch ;Programm über Interrupt 21hex,
25: int 21h ; Funktion 4Chex beenden
26:
27: code ends ;Ende von Segment »code«
28: end start ;Programmende und Definition
29: ; des Startpunktes
```


Com und Exe

– gewichtige Unterschiede

Wie die Programmierung von Maschinen- und Assemblerprogrammen im Prinzip funktioniert, wissen Sie schon. Doch jetzt lernen Sie alle Raffinessen der Programmierung kennen, die auch die Profis anwenden.

Wenn Sie die erste Folge dieses Kurses aufmerksam gelesen haben, wissen Sie bereits, was der Unterschied zwischen Maschinen- und Assemblersprache ist, wie das Binär- und Hexadezimalsystem funktioniert und wie ein typisches Assemblerprogramm aufgebaut ist.

Sicher erinnern Sie sich auch noch daran, daß der 8086/8088-Prozessor seinen Speicher über Segmentregister verwaltet. Dies ist nötig, weil sich mit den 16-Bit-Registern des 8086/8088 nur $2^{16} = 65536$ Speicherzellen (64 KByte) adressieren lassen, während der maximale Speicherbereich des Prozessors 1 MByte beträgt. Die Segmentregister geben nun jeweils die Startadresse (Basis) für ein Speichersegment an und die Adresse in einem anderen 16-Bit-Register (Offset) zeigt auf eine einzelne Speicherzelle innerhalb dieses Segments. Weil das 16-Bit-Register 64 KByte Speicherbereich adressieren kann, darf auch ein Speichersegment maximal 64 KByte groß sein.

Ganz konkret wird die Segmentadresse eines Segmentregisters mit 16 multipliziert und zum Wert der 16-Bit-Adresse addiert, das heißt, die effektive Adresse ist gleich 16 mal Basis plus Offset. Enthält beispielsweise das Segmentregister DS die Adresse B800hex und das Register DX die Adresse 0100hex, so beträgt die effektive Adresse dieser beiden Register $10\text{hex} * \text{B800hex} + 0100\text{hex} = \text{B8100hex}$. Mit dem DX-Register läßt sich damit der Speicherbereich von B8000hex (DX = 0000hex) bis C7FFFhex (DX = FFFFhex) adressieren.

Jedes Segmentregister des 8086/8088 hat seinen eigenen Zuständigkeitsbereich. CS (code segment) ist für den Befehlszeiger IP, SS (stack segment) für den Stapelzeiger SP und Basiszeiger BP, sowie ES (extra segment) für das Zielregister DI bei Zeichenketten-

Operationen zuständig. In allen anderen Fällen ist DS (data segment) angesprochen, so daß dieses Segmentregister für die Daten in den Registern AX, BX, CX, DX, SI und DI gilt.

In Assemblerprogrammen müssen Sie sich um diese Feinheiten nicht kümmern, wenn Ihr Programm nicht länger als 64 KByte ist (und welches Assemblerprogramm ist schon länger?). Dann können Sie nämlich das Speichermodell für Dateien mit der Endung »com« verwenden, das mit nur einem einzigen Speichersegment arbeitet.

In diesem Speichersegment sind Programmcode und Daten gemeinsam enthalten. Der Programmcode beginnt ab der Adresse 0100hex und der Stapel (Stack) befindet sich immer am oberen Ende des Segments. Wenn nun die Daten dem Programmcode unmittelbar folgen und zusammen mit dem Programm weniger als 64 KByte belegen, kommen sich die Bytes von Programm, Daten und Stapel nicht ins Gehege.

Der Speicherbereich von Adresse 0000hex bis 00FFhex innerhalb des Speichersegments wird für das Kennzeichen einer Datei mit der Endung »com« benötigt: das PSP (program segment prefix). Das PSP enthält Daten für die Verwaltung von Com-Dateien durch das Betriebssystem.

Das beschriebene Speichermodell wurde bereits im Assemblerlisting »print« im ersten Teil dieses Kurses verwendet.

Für den Aufbau von Assemblerprogrammen existieren zwei grundlegende Speichermodelle

Der Kopf eines Programms, das mit dem Speichermodell der Com-Dateien arbeitet, sieht immer gleich aus. Zuerst wird mit dem Pseudobefehl »assume« den Segmentregistern CS und DS ein gemeinsames Speichersegment zugewiesen.

Wenn das Programm keine Zeichenketten-Operationen enthält, muß das ES-Segmentregister nicht initialisiert werden. Eine Zuweisung für das Segmentregister SS ist sogar überflüssig, weil es automatisch initialisiert wird.

Dazu richtet das Betriebssystem den Stapel am Ende des Speichersegments ein und legt als Kennzeichen für die Stapelspitze (top of stack) in den Speicherzellen FFFEhex und FFFFhex die 16-Bit-Adresse 0 ab. Diese Adresse weist auf die erste Adresse im PSP, die wiederum die Adresse zum Beenden des Programms enthält.

Weil das Speichermodell für Com-Dateien mit nur einem Speichersegment arbeitet, muß allen zwei oder drei Segmentregistern das gleiche Segment zugeordnet werden. Das Segment trägt in der Regel die Bezeichnung »code«, so daß der komplette Assume-Befehl

```
assume cs:code,ds:code
oder
```

```
assume cs:code,ds:code,es:code
lautet.
```

Der Linker des MASM warnt in diesem Fall zwar, daß kein Segment für den Stapel definiert wurde, doch das braucht uns nicht zu stören.

Der nächste Befehl im Programm muß jetzt

```
code segment
```

lauten, um den Start des Speichersegments zu definieren und gleichzeitig dem Segment den Namen »code« zuzuweisen.

Der darauf folgende Pseudobefehl

```
org 0100hex
```

gibt dem Assembler die Anordnung, das Programm ab der Adresse 0100hex innerhalb des Speichersegments zu beginnen, so daß die ersten 100hex Byte für das PSP reserviert bleiben.

Nun muß der unbedingte Sprungbefehl »jmp« folgen, der mit der Marke, die im Befehl »end« steht, versehen ist und den eigentlichen Programm-anfang aufruft. Ein Programm, das mit dem Speichermodell der Com-Dateien arbeitet, darf nämlich nicht gleich »loslegen«, weil zu Programmbeginn der Datenbereich stehen muß, das heißt der Speicherbereich, in dem Variablen, Konstanten und Datenpuffer definiert sind. Dieser Datenbereich muß sich an den Sprungbefehl unmittelbar anschließen. Danach erst beginnt das Programm, das mit dem Sprungbefehl aufgerufen wird.

Um das Programm elegant zu beenden, empfiehlt sich die Funktion 4Ch des MS-DOS-Interrupt 21hex, die eine Rückkehr in die MS-DOS-Kommandoebene bewirkt. Zwar schließt auch der MS-DOS-Interrupt 20hex ein Programm ab, doch dies ist ein veralteter Interrupt, der bei Dateien mit der Endung »exe« nicht korrekt arbeitet und aus Gründen der Aufwärtskompatibilität nicht benutzt werden sollte.

```
mov ah,4Ch
int 21h
```

lautet die richtige Befehlsfolge zum Beenden des Programms.

Zu guter Letzt wird mit den Pseudobefehlen

```
code ends
```

```
und
```

```
end start
```

das Segmentende von »code« sowie das Programmende festgelegt. Die Angabe von »start« im Befehl »end« markiert die Zeile, in der die Sprungmarke »start« steht als Einstiegspunkt beim Programmstart. Listing 1 zeigt zusammenfassend den beschriebenen Aufbau eines Assemblerprogramms nach dem Speichermodell der Com-Dateien.

Natürlich existiert auch für Dateien mit der Endung »exe« ein besonderes Speichermodell. Dabei können Sie im Unterschied zum Speichermodell für Com-Dateien beliebig viele Speichersegmente definieren. Listing 2 zeigt ein Beispiel, das mit einem Programmsegment, zwei Datensegmenten, sowie einem Extrasegment und einem Stapelsegment arbeitet. Der Wechsel zwischen den Datensegmenten und die Initialisierung von zwei Stapelzeigern (System- und Programm-Stapelzeiger) geht aus diesem Beispiel deutlich hervor.

Interessant ist, daß bei diesem Speichermodell die Segmentregister DS, ES und SS einzeln initialisiert werden müssen. Während dies bei CS beim Programmstart automatisch geschieht (CS gibt schließlich mit dem Befehlszeiger IP die Startadresse des Programms an), muß die Initialisierung der übrigen Segmentregister durch den Programmierer erfolgen. Zuerst wird jeweils das AX-Register mit der Segmentadresse des entsprechenden Speichersegments geladen, anschließend wird dieser Wert in DS geschrieben. Ein direkter Befehl in der Art »mov ds,segment« ist beim 8086/8088 leider nicht möglich, so daß der Umweg über das AX-Register notwendig ist.

Die bedingten Sprungbefehle des 8086/8088-Prozessors

Befehl	englisch	entspricht Befehl	Bedeutung
JA	Jump if above	JNBE	Verzweige, wenn darüber
JAE	Jump if above or equal	JNB	Verzweige, wenn darüber oder gleich
JB	Jump if below	JNAE	Verzweige, wenn darunter
JBE	Jump if below or equal	JNA	Verzweige, wenn darunter oder gleich
JCXZ	Jump if CX equal zero		Verzweige, wenn CX gleich 0
JE	Jump if equal	JZ	Verzweige, wenn gleich
JG	Jump if greater	JNLE	Verzweige, wenn größer
JGE	Jump if greater or equal	JNL	Verzweige, wenn größer oder gleich
JL	Jump if less	JNGE	Verzweige, wenn kleiner
JLE	Jump if less or equal	JNG	Verzweige, wenn kleiner oder gleich
JNA	Jump if not above	JBE	Verzweige, wenn nicht darüber
JNAE	Jump if not above and not equal	JB	Verzweige, wenn nicht darüber und nicht gleich
JNB	Jump if not below	JA	Verzweige, wenn nicht darunter
JNBE	Jump if not below and not equal	JA	Verzweige, wenn nicht darunter und nicht gleich
JNE	Jump if not equal	JNZ	Verzweige, wenn nicht gleich
JNG	Jump if not greater	JLE	Verzweige, wenn nicht größer
JNGE	Jump if not greater and not equal	JL	Verzweige, wenn nicht größer und nicht gleich
JNL	Jump if not less	JGE	Verzweige, wenn nicht kleiner
JNLE	Jump if not less and not equal	JG	Verzweige, wenn nicht kleiner und nicht gleich
JNO	Jump if not overflow		Verzweige, wenn Überlauf-Statusbit gleich 0
JNP	Jump if not parity	JPO	Verzweige, wenn Parität-Statusbit gleich 0
JNS	Jump if not sign		Verzweige, wenn Vorzeichen-Statusbit gleich 0
JNZ	Jump if not zero	JNE	Verzweige, wenn nicht 0
JO	Jump if overflow		Verzweige, wenn Überlauf-Statusbit gleich 1
JP	Jump if parity	JPE	Verzweige, wenn Parität-Statusbit gleich 1
JPE	Jump if parity even	JP	Verzweige, wenn Parität-Statusbit gleich 1
JPO	Jump if parity odd	JNP	Verzweige, wenn Parität-Statusbit gleich 0
JS	Jump if sign		Verzweige, wenn Vorzeichen-Statusbit gleich 1
JZ	Jump if zero	JE	Verzweige, wenn 0

Tabelle. 29 bedingte Sprungbefehle kennt der 8086/8088-Prozessor, doch die meisten kommen doppelt vor

Wie Sie sehen, ist beim Speichermodell für Exe-Dateien der Umgang mit den Segmenten und Segmentregistern schon etwas komplizierter, so daß sich zumindest für den Anfang das Speichermodell für Com-Dateien empfiehlt.

Wenn Sie mit dem Assembler MASM arbeiten, vergessen Sie nicht, die vom Linker erzeugten Exe-Dateien mit dem MS-DOS-Befehl »exe2bin« in Com-Dateien umzuformen. Exe-Dateien, die nach dem Speichermodell der Com-Dateien aufgebaut sind, funktionieren nämlich nicht korrekt. Umgekehrt lassen sich Exe-Dateien, die mit mehreren Speichersegmenten arbeiten, von vornherein nicht in Com-Dateien konvertieren.

Listing 2 zeigt neben dem Aufbau des Speichermodells für Exe-Dateien auch den Pseudobefehl »dw« (define word). Dieser Befehl ähnelt dem Befehl »db«, nur definiert er nicht ein oder mehrere Byte, sondern ein Word beziehungsweise mehrere Words. Während ein Byte als 8-Bit-Wert nur eine Speicherzelle belegt, ist das Word ein 16-Bit-Wert und belegt zwei unmittelbar aufeinanderfolgende Speicherzellen. Dabei wird zuerst das niederwertige und dann das höherwertige Byte im Speicher abgelegt. Dieses Verfahren ist als Intel-Format bekannt.

Listing 2 enthält auch eine neue Variante der Befehle »db« und »dw«.

Hier erhalten die vor den Befehlen stehenden Variablen keinen absoluten Wert zugewiesen, sondern es wird nur ein Speicherbereich initialisiert. Diese Initialisierung geschieht über den Befehl »dup«. So initialisiert beispielsweise der Befehl

```
puffer db 100h dup(?)
```

einen Speicherbereich von 100hex Speicherzellen für die Variable »puffer«. Das Fragezeichen gibt in diesem Fall an, daß der Inhalt von »puffer« bei der Initialisierung gleichgültig ist. Soll dagegen nach der Initialisierung ein bestimmter Wert in »puffer« stehen, so muß dieser Wert anstelle des Fragezeichens in die Klammern gesetzt werden. So setzt beispielsweise der Befehl

```
puffer db 100h dup(0)
```

die 100hex Speicherzellen für »puffer« auf den Wert 0.

Mit einer Handvoll Befehle lassen sich bereits komplette Programme schreiben

Nun ist es an der Zeit, Ihnen einige wichtige Assemblerbefehle genauer zu erklären. Der Befehl »mov« ist der wichtigste Befehl des 8086/8088. Mit »mov« lassen sich Register und Speicherzellen auslesen und laden. Kein Programm kommt ohne diesen Befehl aus. Der Befehl

```
mov ax,0
```


setzt beispielsweise das AX-Register auf 0. Der Befehl

```
mov flag,cl
```

schreibt in die Speicherstelle »flag« den Wert des CL-Registers. Der Befehl

```
mov dx,offset text
```

schreibt schließlich die Adresse der Variablen »text« in das DX-Register. Ohne die Offset-Angabe würde der Prozessor den Inhalt der Variablen »text« in das DX-Register schreiben. Selten kommt es vor, daß ein Programm ausschließlich linear verläuft. Immer wieder sind im Programmfluß Verzweigungen notwendig. Hierfür existieren beim 8086/8088 die Sprungbefehle.

Der Befehl »jmp« (jump), den Sie bereits aus den Programmbeispielen kennen, ist ein unbedingter Sprungbefehl und entspricht dem Goto-Befehl in Basic-Programmen. Dem Befehl »jmp« muß eine Sprungmarke (ein Label) folgen, damit der Assembler weiß, auf welche Adresse sich dieser Befehl bezieht. Die Zieladresse des Sprungbefehls im Assemblerprogramm muß in der ersten Spalte mit dieser Sprungmarke und einem angehängten Doppelpunkt versehen sein.

Mit

```

        jmp      weiter
weiter:  text db   "Dummy-Text"
        mov     ax,offset text

```

überspringt das Programm die Definition der Variablen »text« und fährt mit dem Befehl »mov« fort.

Die zweite Sorte der Sprungbefehle sind die bedingten Sprungbefehle. Diese Befehle verzweigen nur unter bestimmten Bedingungen und sind mit dem Basic-Befehl »if« vergleichbar. Die Tabelle zeigt alle bedingten Sprungbefehle des 8086/8088. Die Angabe des Sprungziels erfolgt wie beim Befehl »jmp« über Sprungmarken.

Damit die Sprungbefehle auch einen Grund zum Verzweigen haben, muß zuvor ein Vergleich stattfinden, der die Statusbits (Null, Übertrag, Überlauf und so weiter) entsprechend setzt. Für diese Art von Vergleichen ist der Befehl »cmp« vorgesehen, der zwei Register oder ein Register und einen Operanden miteinander vergleicht.

```
cmp bl,0
jz fehler
div bl
```

vergleicht den Inhalt des BL-Registers mit 0. Ist der Inhalt 0, so wird eine Fehleroutine aufgerufen, im an-

deren Fall erfolgt die Division von AX durch BL.

Für Unterprogrammaufrufe stehen die Befehle »call« und »ret« zur Verfügung. »call« (call subroutine) ruft ein Unterprogramm über eine Sprungmarke auf und mit »ret« (return from subroutine) kehrt man aus dem Unterprogramm in das Hauptprogramm zurück. Der Befehl »call« legt beim Aufruf eines Unterprogramms eine Rücksprungadresse auf dem Stapel ab, damit der Befehl »ret« im Unterprogramm diese Adresse als Zieladresse für die Rückkehr ins Hauptprogramm verwenden kann.

Mit

```

:      :      :
:      call   sub
:      :      :
:      end
sub:   xor     ax,ax
       xor     bx,bx
       xor     cx,cx
       xor     dx,dx
       ret

```

setzt das Unterprogramm »sub« durch eine Exklusiv-Oder-Verknüpfung die Register AX, BX, CX und DX auf 0 (Probieren Sie es aus, die Verknüpfung funktioniert immer und ist schneller als Befehle in der Art von »mov ax,0«.).

Der Befehl »int« spielt bei dem PC eine äußerst wichtige Rolle. Der Programmierer erspart sich nämlich viel Arbeit, indem er beim PC direkt auf die Routinen des MS-DOS-Betriebssystems und des BIOS (Basic Input Output System) zugreift.

Diese Routinen müssen nicht über absolute Adressen angesprochen werden (die sich dann von Version zu Version ändern würden), sondern lassen sich über Software-Interrupts aufrufen. Der Befehl »int« erzeugt wiederum die Software-Interrupts, die die Arbeit des Prozessors ähnlich wie Unterprogramme unterbrechen.

Mit dem Befehl »int« muß ein Wert im Bereich von 0 bis FFhex angegeben werden. Dieser Wert bezeichnet die Nummer des Software-Interrupts. Zu jedem dieser Interrupts befindet sich im ersten KByte des PC-Speichers eine vier Byte lange Adresse. Der 8086/8088 sucht sich selbständig die zu dem Interrupt passende Adresse (auch Interruptvektor genannt) und ruft diese Adresse, und damit die zu dem Interrupt gehörende Betriebssystem-Routine, auf. Wenn der Software-Interrupt beendet ist, kehrt

er automatisch zu dem Befehl im aufrufenden Programm zurück, der dem Int-Befehl folgt.

Viele Software-Interrupts stellen mehrere Routinen (Funktionen) zur Auswahl. Die Nummer der gewünschten Funktion wird in diesem Fall im AH-Register übergeben. Benötigt eine Funktion die Eingabe von Parametern oder gibt sie Parameter zurück, so sind hierfür bestimmte Register speziell reserviert.

So ruft beispielsweise die Befehlsfolge

```
mov al,07h
mov ah,00h
int 10h
```

die Funktion 00hex des Video-Interrupt 10hex auf. Der Wert des AL-Registers ist ein Parameter für die Funktion 0 und gibt an, daß die Funktion den Videomodus 7 (Monochromadapter) eingestellt werden soll. Ganz flinke Leser werden bemerkt haben, daß sich das Beispiel auch kürzer schreiben läßt:

```
mov ax,0007h
int 10h
```

Die Funktionen des Video-Interrupts und die Funktionen weiterer Software-Interrupts mit Bedeutung und Ein-/Ausgabe-Parametern sind in gesonderten Beiträgen in diesem Heft detailliert aufgeführt.

Die Funktionen der Software-Interrupts des PC reichen eigentlich zum Programmieren aller gängigen Routinen völlig aus. So stellen die Software-Interrupts Funktionen zur Datenein- und -ausgabe über Tastatur, Bildschirm und Drucker, Funktionen zur Datenträger- und Dateiverwaltung sowie Funktionen zur Programmverwaltung und -steuerung zur Verfügung.

Nur wenn Sie es besonders eilig haben, sollten Sie sich dafür eigene Funktionen schreiben. Das kostet dann natürlich Zeit und setzt fundierte Kenntnisse der Computer-Hardware und der Arbeitsweise von Betriebssystem und BIOS voraus.

Das Assemblerprogramm »input« simuliert den gleichnamigen Basic-Befehl

Nachdem Sie im ersten Teil dieses Kurses gesehen haben, wie der Basic-Befehl »print« durch ein Assemblerprogramm simuliert wird, lernen Sie nun das Gegenteil kennen: die Input-

Anweisung. Ganz konkret soll das Programm dem Basic-Befehl

```
input "Geben Sie bitte Ihren Namen  
ein: ";name$
```

entsprechen.

Das Programm muß also zuerst den Text »Geben Sie bitte Ihren Namen ein: « ausgeben und darauf einen Text von der Tastatur lesen, der unter einer Variablen gespeichert wird.

Wie Sie die Textausgabe auf dem Bildschirm in Assembler lösen, wissen Sie bereits von dem Programm »Print«. Sie benutzen wieder den MS-DOS-Interrupt 21hex mit der Funktion 09hex. Im DX-Register wird die Startadresse der auszugebenden Zeichenkette übergeben.

Das Ende der Zeichenkette muß ein Dollarzeichen markieren. Im Programmbereich muß also

```
mov dx,offset text  
mov ah,09h  
int 21h
```

und im Datenbereich

```
text db "Geben Sie bitte Ihren Namen  
ein: $"
```

stehen.

Für die Eingabe von Text eignet sich die Funktion 0Ahhex des MS-DOS-Interrupt 21hex. Diese Funktion läßt sich wie die Funktion 09hex aufrufen, nur enthält das DX-Register diesmal die Adresse eines Puffers, in den der eingegebene Text abgelegt wird.

Die Funktion 0Ahhex speichert alle über die Tastatur eingegebenen Zeichen der im DX-Register angegebenen Adresse ab. Die Eingabe wird durch die Taste [Enter] beendet und läßt sich mit [Ctrl-C] unterbrechen.

Ein großer Vorteil der Funktion 0Ahhex ist, daß alle eingegebenen Zeichen auch auf dem Bildschirm angezeigt werden und die Zeile bis zum Drücken von [Enter] oder [Ctrl-C] frei editiert werden kann.

Der Eingabepuffer für die Funktion 0Ahhex muß folgendermaßen aufgebaut sein:

1. Byte: Anzahl der maximal folgenden Zeichen
2. Byte: Anzahl der tatsächlich eingegebenen Zeichen
3. Byte: Erstes eingegebenes Zeichen

4. Byte: Zweites eingegebenes Zeichen und so weiter

Beginnt der Puffer beispielsweise mit dem Wert 21, so lassen sich maximal 20 Zeichen eingeben, weil der zweite Wert bereits mit der Anzahl der tatsächlich eingegebenen Zeichen belegt ist.

Listing 3 zeigt das komplette Assemblerprogramm »Input«. Das Programm arbeitet wieder nach dem Speichermodell für Com-Dateien.

Ganz aufmerksamen Lesern wird nicht entgangen sein, daß die Befehlsfolge

```
mov dx,offset adresse  
mov ah,nummer  
int 21h
```

zum Aufruf einer Funktion des Interrupt 21hex zweimal im Programm erscheint.

In größeren Programmen kommt diese Befehlsfolge noch weitaus öfter zum Einsatz, doch muß sie deshalb nicht jedesmal neu eingegeben werden.

Der MASM ist nämlich in der Lage, sogenannte Makros zu verwalten. Makros sind immer wiederkehrende Programmteile, die der Programmierer nur ein einziges Mal zu schreiben braucht. Wenn er den im Makro definierten Programmteil dann an einer Stelle in seinem Programm benötigt, muß er hier lediglich den Namen des Makros angeben.

Makro oder Unterprogramm, das Einsatzgebiet ist entscheidend

Der Assembler fügt beim Übersetzen des Programms in Maschinencode das Makro an den entsprechenden Stellen automatisch ein.

Anstelle eines Makros ließe sich auch ein Unterprogramm einsetzen. Doch im Gegensatz zum Makro muß das Unterprogramm jedesmal über den Befehl »call« aufgerufen und mit dem Befehl »ret« beendet werden. Das kostet Zeit. Dafür hat ein Unterprogramm aber auch einen Vorteil:

Es ist nämlich nur ein einziges Mal im Programm vorhanden, während der Assembler das Makro an jeder Stelle, an der es benötigt wird, neu einsetzt.

Wenn Sie also Geschwindigkeit wünschen, sind Makros geeignet, wenn

Sie Speicherplatz sparen wollen (oder müssen), benutzen Sie besser Unterprogramme.

Das Makro für den oben angegebenen Funktionsaufruf lautet nun:

```
funktion macro nummer,adresse  
mov dx,offset adresse  
mov ah,nummer  
int 21h  
endm
```

Das Makro trägt damit den Namen »funktion« und erwartet die Übergabe der Parameter »nummer« und »adresse«. Der erste Parameter dient als Funktionsnummer für das AH-Register und der zweite Parameter als Speicheradresse für das DX-Register. Die beiden Pseudobefehle »macro« und »endm« kennzeichnen Anfang und Ende des Makros.

Wenn Sie dieses Makro zum Programmbeginn definieren, können Sie für den Aufruf der Funktion 09hex in Listing 3

```
funktion 09h,text
```

und für den Aufruf der Funktion 0Ahhex

```
funktion 0Ah,max
```

schreiben.

Der Assembler fügt dann bei der Übersetzung in Maschinencode anstelle dieser beiden Zeilen jeweils das Makro ein und ordnet im ersten Fall dem Parameter »funktion« den Wert 09hex und dem Parameter »adresse« die Variable »text« zu. Im zweiten Fall wird »funktion« auf 0Ahhex und »adresse« auf »max« gesetzt.

Das bedeutet, daß die Angaben, die in einem Makroaufruf hinter dem Makronamen stehen, an die symbolischen Parameter in der Makrodefinition (in der selben Reihenfolge) übergeben werden. Listing 4 zeigt, wie Sie das Programm »input« für den Einsatz des beschriebenen Makros ändern müssen.

Ein Programm, das mit Makros arbeitet, ist übersichtlicher und benötigt bei häufigen Makroaufrufen erheblich weniger Tippaufwand, als ein Programm ohne Makros.

Zudem lassen sich mit Makros komplette Routinen-Bibliotheken aufbauen, die später bei der Entwicklung von neuen Programmen viel Zeit sparen helfen.

(Martin Althaus/
Markus Zietlow/ma)


```

0: assume    cs:code,ds:code    ;Segmentzuweisungen für CS und
   DS
1:
2: code      segment            ;Start des Speichersegments
3: org       0100hex            ;Programmstart ab Adresse 0100
   hex
4:
5: start:    jmp    anfang       ;Programmanfang aufrufen
6:
7: ;         Datenbereich:
8:
9: ;         Programmbereich:
10:
11: anfang:   ...                ;Erster Befehl im Programm
12:           ...
13:           ...                ;Letzter Befehl im Programm
14:
15:         mov    ah,4Ch          ;Programm beenden
16:         int     21h
17:
18: code      ends                ;Ende des Speichersegments
19:         end     start          ;Programmende und Definition
20:           ; des Startpunktes

```

Listing 1. Der grundlegende Aufbau von Assemblerprogrammen, die mit dem Speichermodell der Com-Dateien arbeiten

```

0:          title    exedatei
1: assume    cs:code,ds:data0,es:extra,ss:stack
2:           ;Segmentzuweisungen
3:
4: data0     segment            ;1. Datensegment
5: a         db      10         ;a=10
6: b         db      11         ;b=11
7: c         db      12         ;c=12
8: data0     ends
9:
10: data1     segment            ;2. Datensegment
11: d         db      13         ;d=13
12: e         db      14         ;e=14
13: f         db      15         ;f=15
14: data1     ends
15:
16: extra     segment            ;Extrasegment für
17: string1   db      100h dup(?) ;drei maximal
18: string2   db      100h dup(?) ;256 Bytes lange
19: string3   db      100h dup(?) ;Zeichenketten
20: extra     ends
21:
22: stack     segment            ;Stapelsegment
23: dw        400h dup(?)        ;1 KByte für System-
24: tos       label word         ;stapel reservieren
25: dw        400h dup(?)        ;1 KByte für Programm-
26: tob       label word         ;stapel reservieren
27: stack     ends
28:
29: code      segment            ;Programmsegmentanfang
30: start:    mov     ax,data0     ;DS für Datensegment
31:           mov     ds,ax        ;data0 initialisieren
32:           mov     ax,extra     ;ES initialisieren
33:           mov     es,ax
34:           mov     ax,stack     ;SS initialisieren
35:           mov     ss,ax
36:           mov     sp,offset tos ;SP initialisieren
37:           mov     bp,offset tob ;BP initialisieren
38:
39:           mov     bh,a         ;BH=10
40:           mov     bl,b         ;BL=11
41:           mov     ch,c         ;CH=12
42:
43:           mov     ax,data1     ;DS für Datensegment
44:           mov     ds,ax        ;data1 initialisieren
45: assume     ds:data1           ;Neue Segmentzuweisung
46:           ; für DS
47:           mov     cl,d         ;CL=13
48:           mov     dh,e         ;DH=14
49:           mov     dl,f         ;DL=15
50:           ...
51:           ...
52:           ...                ;Weitere Befehle
53:           mov     ah,4Ch       ;Programm über MS-DOS-
54:           int      21h         ;Funktion beenden
55: code      ends                ;Programmsegmentende
56:           end     start        ;Programmende

```

Listing 2. Assemblerprogramme, die mit dem Speichermodell für Exe-Dateien arbeiten, sind im Aufbau komplexer als Com-Dateien

```

0:          title    input
1:
2: ; Dieses Programm gibt auf den Bildschirm
3: ; eine Zeichenkette aus und liest eine
4: ; Zeichenkette ein
5:
6: assume    cs:code,ds:code    ;Segmentzuweisungen
7:
8: code      segment            ;Segmentanfang
9: org       100h               ;Programmstart bei 1
   00hex
10:
11: start:    jmp     anfang       ;Sprung zum Programm
12: anfang
13: ;         Datenbereich:
14:
15: text      db      13,10,"Bitte geben Sie Ihren Namen ein: $"
16: max       db      21          ;Erstes Byte des Eingabe-
17:           ; puffers (Pufferlänge)
18:           db      21 dup (?)   ;Eingabepuffer
19:
20: ;         Programmbereich:
21:
22: anfang:   mov     dx,offset text ;Adresse der Zeichen
   kette
23:           mov     ah,09h        ;Zeichenkette ausgeb
   en
24:           int      21h
25:           mov     dx,offset max  ;Adresse des Puffers
26:           mov     ah,0Ah        ;Zeichenkette einles
   en
27:           int      21h
28:
29:           mov     ah,4Ch        ;Programm beenden
30:           int      21h
31:
32: code      ends                ;Segmentende
33:           end     start        ;Programmende

```

Listing 3. Das Assemblerprogramm »input« simuliert den gleichnamigen Basic-Befehl

```

0:          title    makro
1:
2: ; Dieses Programm gibt über ein Makro eine
3: ; Zeichenkette auf den Bildschirm aus und
4: ; liest eine Zeichenkette ein
5:
6: funktion macro    funktion,adresse ;Makrodefinition
7:                 mov     dx,offset adresse
8:                 mov     ah,funktion
9:                 int      21h
10:                endm
11:
12: assume    cs:code,ds:code    ;Segmentzuweisungen
13:
14: code      segment            ;Segmentanfang
15: org       100h               ;Programmstart bei 1
   00hex
16:
17: start:    jmp     anfang       ;Sprung zum Programm
18: anfang
19: ;         Datenbereich:
20:
21: text      db      13,10,"Bitte geben Sie Ihren Namen ein: $"
22: max       db      21          ;Erstes Byte des Eingabe-
23:           ; puffers (Pufferlänge)
24:           db      21 dup (?)   ;Eingabepuffer
25:
26: ;         Programmbereich:
27:
28: anfang:   funktion    09h,text ;Zeichenkette ausgeb
   en
29:           funktion    0Ah,max  ;Zeichenkette einles
   en
30:
31:           mov     ah,4Ch        ;Programm beenden
32:           int      21h
33:
34: code      ends                ;Segmentende
35:           end     start        ;Programmende

```

Listing 4. Das Assemblerprogramm »makro« ist eine Makro-Version von »input«

ProSoft-Preise liegen richtig!

☎ 0261/40 47-1 • TX 862476 PSOFT • Telefax 0261/40 47-252

Wir suchen ständig günstige Einkaufsquellen für die angebotenen und neue innovative Produkte. Günstige Möglichkeit der Finanzierung durch Ratenkredit. Fordern Sie die Unterlagen an.

Commodore-PC

PC-10 S 512 KB Hauptspeicher, serielle und parallele Schnittstelle, 1 Diskettenlaufwerk 360 KB, AGA Monochrom- und Colorgrafikkarte, Tastatur, MS-DOS, GW-Basic
PC-10 S/20 wie PC-10 S, jedoch mit 20 MB Festplatte (65 ms)
PC-10 S/30 wie PC-10 S, jedoch mit 30 MB Festplatte (65 ms)
PC-10 S 2512 KB Hauptspeicher, serielle und parallele Schnittstelle, 2 Diskettenlaufwerke 360 KB, AGA Monochrom- und Colorgrafikkarte, Tastatur, Monitor 12", MS-DOS, GW-Basic
PC-10 S/20 wie PC-10 S/2, jedoch mit 20 MB Festplatte (65 ms)
PC-10 S/30 wie PC-10 S/2, jedoch mit 30 MB Festplatte (65 ms)
PC-AT 40/40 80286 CPU (6/8 MHz), 1 MB Hauptspeicher, 1 Diskettenlaufwerk 1.2 MB, 1 Festplatte 40 MB, serielle u. parallele Schnittstelle, EGA Grafikkarte, 14" Monitor, Multifunktions-tastatur, MS-DOS, GW-Basic
Amiga 500 1098,- Amiga 2000 2298,- Monitor 1081 729,-

Tandon
XPC 2/20 2598,- XPC 2/30 2698,- XPC 20 2698,-
PCA 4198,- PCA 20 4698,- PCA 30 5078,-
PCA 40 5298,- PCA 70 6498,-
Target 20 5598,- Target 40 6098,- PAC Tower 3798,-
Diskettenaufsatz für PAC Tower 898,- 30 MB PAC 798,-
PCA 110 80286 CPU (6/8 MHz), 1 MB Hauptspeicher, serielle und parallele Schnittstelle, 1 Diskettenlaufwerk 1.2 MB, 1 Festplatte 110 MB formatiert (18 ms), Hercules kompatible Grafikkarte, Monitor 14", Tastatur, MS-DOS, GW-Basic, MS-Windows
Aufpreis Farbsystem 898,- Aufpreis EGA-System 1078,-

Zusatzkarten-Erweiterungen-Software

20 MB Festplatte Seagate ST 225"
(65 ms) Controller, Kabelsatz u. Einbauanleitung 643,-
30 MB Festplatte (Seagate ST 238)
incl. RLL-Controller u. Kabelsatz für XT 699,-
20 MB Festplatte (Seagate ST 225), 65 ms für XT/AT 548,-
30 MB Festplatte (Seagate ST 238), 65 ms für XT/AT 578,-
30 MB Festplatte (Seagate ST 4038), 40 ms für XT/AT 1078,-
40 MB Festplatte (Seagate ST 251), 40 ms für AT 898,-
40 MB Festplatte (Seagate ST 251/1) 28 ms 978,-
80 MB Festplatte (Seagate ST 4096), 28 ms 1798,-
Festplattencontroller für XT incl. Bedienungsanleitung (formatiert 10-30 MB Festplatten) 198,-
Festplattencontroller WD 1002 348,-
AT-RLL Controller OMTI 8627 498,-

Tandon
Es muß nicht immer Seagate sein, 20 MB von Tandon - das Qualitätsprodukt!
TM 965-2 360 K-Floppy 198,-
TM 975-8 1.2 MB-Floppy 298,-
TM 9252 slave 10 MB-Festplatte 348,-
TM 9252 AI 10 MB-Festplatte + Controller 498,-
TM 9262 slave 20 MB-Festplatte 498,-
TM 9262 AI 20 MB-Festplatte + Controller 648,-
TM 9755 AI 40 MB-Festplatte 1078,-
TM 9755 AI 40 MB-Festplatte + Controller 1378,-
Formatierungssoftware für Festplatten größer als 30 MB
Disk Manager by Ontrack 48,-
SpeedStar 128,- V-Feature de Luxe 248,-

Microscience-Festplatten
HH 725 20 MB 5 1/4" incl. Controller + Kabelsatz 658,-
HH 738 30 MB 5 1/4" incl. RLL-Controller + Kabelsatz 718,-
HH 1050 40 MB, 28 ms 1198,-
HH 325 20 MB, 3 1/2" 528,-
HH 330 30 MB, RLL, 3 1/2" 538,-

Hardcards
Microscience 30 MB Hardcard 848,- Tandon Hardcard 20 MB (40 ms) 648,-
Fujitsu Hardcard 40 MB (40 ms) 1698,-

NEC-Festplatten
D5126 (20 MB - 85 ms) 798,- D3126 (20 MB - 3.25" - 85 ms) 998,-
D5126 H (20 MB - 40 ms) 1098,- D5452 (85MB - 23MS-full height) 3298,-
D5146 (40 MB - 40 ms) 1498,- D5652 (170MB-23MS-full height) 3698,-
Concept-Card 898,-

Priam-Festplatten
V-185 18 ms, 73 MB 1898,-
Interdyne 20 MB Tape-Streamern intern 598,-
40 MB Tape Streamer „APT 40“ (ALLOY) 848,-
Wangtek Tape-Streamern 52 MB FAD 5000 999,-
TakeTen 10 MB Disk Cartridge Subsystem-External 1698,-

Grafik-Adapter
NEUI EGA-Wonder Enhanced nur 448,-
VEGA de Luxe Autoswitch 598,-
ATI-„Graphic Solution“ 298,-
Hercules Graphic Card plus incl. RAM-FONT 578,-
Hercules kompatible Grafik-Karte 148,-
Farbgrafik-Adapter 128,- Paradise EGA 388,-
Paradise EGA Autoswitch 80-Zeichen 418,-
Paradise EGA Autoswitch 132-Zeichen 478,-
Genoa Super HI-RES 548,-
Neut Ujr plus serielle Schnittstelle 98,-
NEC JC 1401 P3E Multisync, 14" EGA Monitor 1298,-
NEC Multisync plus Paradise EGA Autoswitch 80-Zeichen 1648,-

Co-Prozessoren
8088 8-bit-Prozessor 49,- 8086 16-bit-Mikropr. 49,-
8087 (8 MHz) 349,- 8087 (5 MHz) 249,-
80287 (8 MHz) 399,- 80287 (6 MHz) 399,-
80287 (8 MHz) 599,- 80287 (10 MHz) 649,-
Handy Scanner 675,-
Alles Zubehör und Software rund um den PC zu sehr günstigen Preisen. Fordern Sie die Preisliste an!

Plantron - Plantron - Plantron

Plantron PT 16 LC umschaltbar 4,77/8 MHz, 256 KB Hauptspeicher, Monochrom-Grafikkarte (Hercules kompatibel), paralleler Druckeranschluß, 1 Diskettenlaufwerk 360 KB, Tastatur (deutsch) mit separatem Cursorblock, MS-DOS 3.2/GW-Basic 1245,-
Plantron PT LC/20 wie PT LC, zusätzl. 20 MB Festpl. 1998,-
Plantron PT LC/30 wie PT LC, zusätzl. 30 MB Festpl. 2098,-
Plantron PT XT Turbo 4,77 oder 8 MHz Takt, 256 KB Hauptspeicher, Monochrom-Grafikkarte (Hercules kompatibel), Multifunktionskarte mit paralleler Schnittstelle, serieller Schnittstelle, Game Port, Echtzeituhr, MS-DOS 3.2 incl. Basic, RAM Disk, Druckerspools, Bedienungsanleitung und Zubehör, 2 Diskettenlaufwerke à 360 KB, Tastatur mit separatem Cursorblock 1698,-
Plantron PT XT 2/20 Turbo wie PT XT Turbo, zusätzl. 20 MB Festplatte 2398,-
Plantron PT XT 2/30 wie PT XT Turbo, zusätzl. 30 MB Festplatte 2498,-
Plantron PT ST 6 6 MHz oder 8 MHz Takt, 640 KB Hauptspeicher (bis 1 MB on Board), Monochrom-Grafikkarte (Hercules kompatibel), parallele Drucker-schnittstelle, Diskettenlaufwerk 1,2 MB, Tastatur (deutsch) mit sep. Cursorblock, Echtzeituhr, MS-DOS 3.2/GW-Basic, Bedienungsanleitung und Zubehör 2498,-
Plantron PT-ST/20 wie PT-ST, zusätzl. 20 MB-Platte und Floppy-Hard-Disk-Controller 3198,-
Plantron PT-ST/30 3298,-
Plantron PT-AT wie PT-ST, zusätzl. Multi I/O-Karte mit paralleler und serieller Schnittstelle, Game Port, Floppy-Hard-Disk-Controller 2998,-
Plantron PT AT/20 wie PT AT zusätzl. m. 20 MB Festpl. 3498,-
Plantron PT AT/30 wie PT AT zusätzl. m. 30 MB Festpl. 3798,-
Plantron PT-AT/40 wie PT-AT zusätzl. 40 MB Festpl. 3998,-
PT-AT/73 wie PT-AT zusätzl. 73 MB-Platte (18 ms) 4798,-
PT-AT 95 wie PT-AT zusätzl. 95 MB-Festplatte 4998,-
PT-286 AT/64 Baby AT mit 64-MB-Festplatte 3898,-
PT-386 HT/2 EGA-Karte und 32-MB-Festplatte 5598,-
PT-386 HT EGA-Karte und 64-MB-Festplatte 6998,-
PT-386 HT/95 EGA-Karte und 95-MB-Festplatte 7998,-
Auf Wunsch der Fa. Plantron bestätigen wir, daß wir die günstigsten Preise durch Selbstaufrüsten und Einbau der Festplatten und Karten durch unsere geschulten Techniker erreichen können. - Selbstverständlich gilt dies nicht nur für Plantron-Produkte.

Software

Microsoft
MS-Word 3.01 878,- MS-Word mit MS-Mouse 1228,-
MS-Chart 598,- MS-Multiplan 558,-
MS-Multiplan mit MS-Mouse 798,- MS-Windows 238,-
MS-Windows mit MS-Mouse 548,- MS-Project 768,-
R-Base 578,-

Multitech

Acer 500 S 1448,- Acer 500 D 1798,- Acer 500 E 2148,-
Acer 710 B 2448,- Acer 710 E 3098,-
Acer 910 B 3248,- Acer 910 F 3798,- Acer 910 E 4198,-
Acer 900 B 3898,- Acer 900 F 4598,- Acer 900 E 4998,-
Acer 1100 B 8998,- Acer 1100 E 9898,- Acer 1100 H 10698,-

MPC 1100 H/95 80386 CPU (16 MHz), 1 MB Hauptspeicher, 1 Diskettenlaufwerk 1.2 MB, 1 Festplatte 95 MB formatiert (18 ms), 2 serielle, 1 parallele Schnittstelle, Monitor 14", Multifunktions-tastatur, MS-DOS 3.2 11448,-

Schneider
Schneider 6128 grün 749,- Joyce PCW-8256 999,-
DMP-2000 528,- DMP-3000 558,-
DMP-4000 848,- Joyce + PCW-8512 2098,-

Schneider PC - Schneider PC

PC 1512 MM/SD 1349,- PC 1512 MM/HD 20 (Seagate 65 ms) 2199,-
PC 1512 MM/DD 1699,- PC 1512 MM/HD 30 (Seagate) 2299,-
PC 1512 MM/HD 20 2699,- Vortex PC 1512 MB-Drive Card 1198,-
Speichererweiterung auf 640 KB RAM 148,-
PC 1640 MD/SD 1498,- PC 1640 MD/HD 20 (Seagate) 2298,-
PC 1640 MD/DD 1898,- PC 1640 CD/HD 20 (Seagate) 2798,-
PC 1640 CD/SD 1978,- PC 1640 MD/HD 30 (Seagate) 2898,-
PC 1640 CD/DD 2348,- PC 1640 MD/HD 30 (Seagate) 2998,-
PC 1640 MD/HD 20 2848,- PC 1640 ECD/HD 20 (Seagate) 3498,-
PC 1640 CD/HD 20 3248,- PC 1640 ECD/HD 30 (Seagate) 3598,-
PC 1640 ECD/SD 2678,- PC 1640 ECD/DD 2948,-
PC 1640 ECD/HD 20 3798,-

Brother
M-1109 469,- M-1409 799,- M-1509 999,-
M-1709 1199,- M-2024 L+ 1999,-

Star - Star - Star

NL-10 539,- zusätzliches Interface für NL-10 78,-
ND-10 898,- ND-15 1198,- NR-10 1148,-
NR-15 1398,- NB 24-10 1398,- NB 24-15 1798,-
NB-15 2348,- SR-10 798,-

Atari

Atari 520 STM Tastatur, 512 KB RAM, 192 KB ROM, HF-Modulator, Floppy SF 354, ohne Monitor, Maus, Basic 598,-
Atari 1040 STF Tastatur, 1024 KB RAM, 192 KB ROM, integrierte Floppy 720 KB, ohne Monitor, Maus, Basic 1165,-
Monitor SM-124 469,-

Der Preishammer von ProSoft:

Atari 1040 STF, Tastatur, 1024 KB RAM, 192 KB ROM, integrierte Floppy 720 KB, Monochrom-Monitor SM 124, Maus, Basic 1498,-

Trans Net - Trans Net - Trans Net

Die interessanteste Netzwerklösung auf der High-Tech Szene Trans-Net ist eines der bedienerfreundlichsten Low-Cost Netzwerke für PC's auf dem Markt!
Möglichkeiten: beliebige Festplattenzugriffe im Netzwerk, Diskettenzugriffe im Netzwerk, DOS-Transparenz, File Record Locking, TRANS-NET Druckerspools, erweiterte Netzwerk DOS-Befehle, Multi-Tasking, Schüler/Lehrer Monitor, NET-BIOS Emulator, NOVELL Emulator.
Starterkit 2 Netzwerkkarten, Kabel und Software 1698,-
Netzwerkkarte 698,-

KAYPRO Kaypro-Produkte können wir preiswert liefern!

OKI - OKI - OKI

Okimate 20 Farbdrucker mit Interface 498,-

Die günstigen Preise für die gesamte OKI-Produktpalette incl. Laserline erfahren Sie bei uns am Telefon!

Olivetti - Olivetti - Olivetti

M 240/55 G und M 240/33 G 8086 CPU (10 MHz), 640 KB Hauptspeicher, 2 Diskettenlaufwerke (5,25" 360 KB oder 3,5" 720 KB), serielle und parallele Schnittstelle, batteriegepufferte Echtzeituhr, OGC Bildschirmcontroller, Monitor 12", Tastatur 2, MS-DOS/GW-Basic 3348,-
M 240/0520 G und M 240/0320 G wie M 240/55 G und M 240/33 G, jedoch 20 MB Festplatte (65 ms) zusätzl. 3998,-
M 28 AT-Einstiegskonfiguration 1 16 Bit 80286, 8 MHz, 1 MB Hauptspeicher, serielle und parallele Schnittstelle, 1 Diskettenlaufwerk 1.2 MB, 1 Festplatte 20 MB, Monitor, Tastatur, MS-DOS, GW-Basic 6498,-

Sharp Sharp Sharp

Pocket-Computer
PC-1246 S 94,- PC-1280 268,- PC-1460 275,-
PC-1248 123,- PC-1360 344,- PC-1475 275,-
PC-1260 206,- PC-1421 240,- PC-1500 A 344,-
PC-1262 275,- PC-1425 275,- PC-1600 690,-
PC-1270 115,- PC-1450 219,- PC-2500 549,-

EPSON - EPSON - EPSON

Der neue Renner von EPSON: LX-800 548,-

LX-800 VC/P 648,-

FX-800 937,- FX-1000 1208,-

LQ-800 1098,- LQ-850 1298,- LQ-1000 1898,-
LQ1050 1698,- LQ-2500 2448,- IX-800 1579,-
EX-800 1319,- EX-1000 1649,- SX-2500 3198,-
Einzelblatteinzug: für LQ-850 298,- für LQ-1050 398,-
Görllitz-Interface EPSON/C64 cd. C128 198,-

NEC - NEC - NEC - NEC - NEC

P 5 1998,- **P 2200** 988,-
P 7 1398,- **P 6** 1148,-
P 7 color 1648,- **P 6 color** 1448,-
P 7 seriell 1848,- **P 6 seriell** 1398,-
P 6 seriell color 1848,- **P 7 seriell color** 2298,-
Pin-Feed-Tractor für P 6 109,- Bidirektionaler Tractor für P 6 298,-
Pin-Feed-Tractor für P 7 228,- Bidirektionaler Tractor für P 7 348,-
Cut-Sheet-Feeder für P 7 598,-

Kyocera-Laserdrucker Kyocera-Laserdrucker

F-1000 5298,- F-2200 11198,-
F-1200 7798,- F-3000 15598,-

Citizen - Citizen - Citizen

Nun hat sich Citizen dem Preisniveau des NL-10 angepaßt:
LSP-120 D Commodore- oder Parallel-Schnittstelle 398,-

Panasonic - Panasonic - Panasonic

KX-P 1081 478,- KX-P 1082 698,- KX-P 1083 948,-
KX-P 1592 1195,- KX-P 1595 1598,-

Seikosha - Seikosha - Seikosha

SP-180 A 449,- SP-1200 AS 528,- SP-180 VC 449,-
MP-1300 AI 1098,- SP-1200 AI 528,- MP-5300 AI 1378,-

SL-80 AI nur 768,- **SL-80 VC** 768,-

Disketten No-Name

3" Maxell 10 Stück 79,- DM 100 Stück 650,- DM
3 1/2" 1D 70,- DM 330,- DM 650,- DM
3 1/2" 1D 35,- DM 149,- DM 249,- DM
3 1/2" 2D 39,- DM 159,- DM 299,- DM
5 1/4" 2D 19,- DM 59,- DM 89,- DM

5 1/4" 1 D 100 Stück 79,- DM
5 1/4" 1 D 1000 Stück 698,- DM
5 1/4" 2 D 1000 Stück 699,- DM

5 1/4" No-Name HD, 1,6 MB 10 Stück 48,- DM

ProSoft GmbH

Bogenstraße 51-53, Postfach 207, D-5400 Koblenz-Goldgrube, Telefon (0261) 40 47-1, Telex 8 62 476, Telefax (0261) 40 47-252

Filiale München Theresienstraße 56, 8000 München 2, Tel. 0 89 2 80 93 89 direkt bei der technischen Hochschule. Bitte beachten Sie, daß nicht ständig sämtliche Ware in unserer Filiale München vorrätig ist. Rufen Sie an!

Alle Preise zuzügl. 10,- DM Versandkosten pro Paket. Lieferung per Nachnahme oder Vorkassascheck - Versandkosten Ausland DM 40,- pro Paket. Lassen Sie sich keinen Bären aufbinden! ProSoft liefert Original-Produkte der führenden Hersteller. Überzeugen Sie sich selbst durch Abholung der Ware in unseren Verkaufsräumen in Koblenz. Wir gewähren Ihnen bei Barzahlung (kein Scheck) 2% Skonto auf alle Preise, was vielleicht schon zur Deckung ihrer Reisekosten ausreicht. Einige unserer Vorlieferanten liefern Produkte ohne die Seriennummer des Herstellers. In diesem Fall übernehmen wir anstelle der Herstellergarantie die unbeschränkte gesetzliche Gewährleistung.

MASM

– ein leistungsfähiger Assembler

Nachdem Sie erste Erfahrungen im Umgang mit Assembler- und Maschinensprache gesammelt haben, finden Sie nun weitergehende Informationen über die Programmierung mit dem MASM und über die Anwendung der MS-DOS-Funktionen des Interrupt 21hex.

Bis jetzt haben Sie den MASM nur beiläufig kennengelernt und ihn über die Stapeldatei »asm.bat« aufgerufen, die Ihnen alles weitere abgenommen hat. Das soll sich nun ändern, denn der MASM ist ein äußerst leistungsfähiger Assembler, der eine Vielzahl von Pseudobefehlen verarbeitet, die dem Programmierer das Arbeiten erleichtern.

Tabelle 1 zeigt die Pseudobefehle des MASM im Überblick. Die grundlegenden Pseudobefehle, die auch andere Assembler verarbeiten, kennen Sie bereits. Nur die Makro-Pseudobefehle »macro« und »endm« gehören schon zu dem Kreis der etwas exklusiveren Pseudobefehle.

Diese beiden Befehle erlauben dem MASM-Programmierer den Aufbau eigener Makro-Bibliotheken, die das ständige Neuprogrammieren von Standardroutinen ersparen. So ist es zum Beispiel möglich, die Input-Routine aus dem letzten Kursteil als Makro zu definieren. Wird dann in einem anderen Programm dieser Eingabebefehl benötigt, so muß nur der Name des Makros in den Programmtext eingefügt werden. Dieses Verfahren macht Assemblerprogramme übersichtlich und durchschaubar.

Wenn Sie Makros, die als eigenständige Dateien vorliegen, in einem anderen Programm verwenden möchten, müssen Sie natürlich dafür sorgen, daß der MASM beim Übersetzen Ihres Programms diese externen Dateien auch in den Maschinencode mit einbindet.

Das geschieht mit dem Pseudobefehl »include«, der den MASM veranlaßt, die benötigten externen Dateien von einem Datenträger zu lesen. Dazu müssen Sie nach dem Include-Befehl nur den Namen der Datei angeben, die das oder die Makros beinhaltet. Möchten Sie mehrere Makro-Dateien einbinden, so verwenden Sie den Include-Befehl entsprechend mehrmals im Programm.

Sie wollen beispielsweise eine Input-Routine mit dem Namen »input.asm«, die auf der Diskette im

Die Pseudo-Befehle des MASM

Symboldefinitionen:

equ	Weist einem symbolischen Namen einen konstanten Wert zu, Beispiel: anzahl equ 100*20
=	Wie »equ«, doch läßt sich der Wert umdefinieren, Beispiel: wert = 200 * anzahl

Datendefinitionen:

db	Definiert Byte-Speicherbereich für Variable oder Konstante (Daten). Es lassen sich mehrere Werte zuweisen, Beispiel: text db 1,20h,"beliebiger Text",20h
dw	Wie »db«, jedoch werden 2 Byte-Werte (Words) definiert, Beispiel: text dw 1,7FFFh,"Beliebiger Text",0FFFFh
dd	Wie »db«, jedoch werden 4-Byte-Werten (Double) definiert, Beispiel: text dd 7FFFFFFFh,"Text",0FFFFFFFh

Makro-Definitionen:

macro	Die vorausgehende Marke (Label) definiert ein Makro mit den nachfolgenden Parametern, Beispiel: input macro adresse
endm	Ende einer Makrodefinition, Beispiel: input endm
local	Mit »local« werden Marken innerhalb eines Makros definiert, um den Fehler »Mehrfachdefinition« zu vermeiden, Beispiel: local marke
rept	Alle Anweisungen zwischen »rept« und »endm« werden mehrmals wiederholt, Beispiel: rept 100
irp	Die Anweisungen zwischen »irp« und »endm« werden für jedes Argument hinter »irp« wiederholt (vergleichbar mit dem Batchbefehl »for in() do«), Beispiel: irp parameter,(2,4,8,16)
irpc	Wie »irp«, jedoch werden hier die Anweisungen für jedes Zeichen einer Zeichenkette wiederholt, Beispiel: irpc paramter,string
purge	Löscht ein bestimmtes Makro aus dem »Gedächtnis« des Assemblers und gibt den Speicher frei, Beispiel: purge input
exitm	Beendet ein Makro vorzeitig (für bedingte Assemblierung), Beispiel: exitm

Segmentbefehle:

segment	Kennzeichnet den Beginn eines Speichersegments, Beispiel: code segment
ends	Kennzeichnet das Ende eines Speichersegments, Beispiel: code ends
assume	Teilt dem Assembler mit, welche Segmentregister welchem Segment zugeordnet sind, Beispiel: assume cs:code,ds:data

Weitere Befehle:

proc	Kennzeichnet den Beginn einer Prozedur (Unterprogramm), Beispiel: output proc far
endp	Kennzeichnet das Ende einer Prozedur, Beispiel: output endp
end	Setzt das Ende des Assemblerprogramms, Beispiel: end start

Ausgabe- und Formatsteuerung:

page	Definiert Länge und Breite einer Druckseite, Beispiel: page 72,80
title	Definiert eine Kopfzeile für die Druckseiten, Beispiel: title PC-Druckertreiber
subttl	Definiert eine Unterzeile für die Kopfzeile, Beispiel: subttl auch für Laserdrucker

Externes Arbeiten:

public	Kennzeichnet bestimmte Marken als »öffentlich«, so daß sie auch für andere Module verwendbar sind, Beispiel: public common
extrn	Definiert Marke als extern, das heißt in einem anderen Modul vorhanden (Gegenbefehl zu public), Beispiel: extrn common:near
include	Liest komplette Programmdateien in den aktuellen Programmtext ein (sehr wirkungsvoll für Makro-Bibliotheken), Beispiel: include c:\assem\lib\input.lib

Tabelle 1. Die große Zahl der Pseudobefehle macht den MASM so leistungsfähig

Laufwerk A steht, in Ihr neues Programm »inpout.asm« einbinden. In diesem Fall müssen Sie die Befehlszeile

```
include a:input.asm
```

zu Beginn von »inpout.asm« einfügen.

Dadurch liest der Assembler beim Übersetzen von »inpout.asm« alle Befehle aus der Datei »input.asm« ein und behandelt sie so, als würden sie direkt im Programm »inpout.asm« stehen.

Assemblerprogramme lassen sich in Form von Objektmodulen miteinander verbinden

Eine weitere Methode, Programme zeitsparend und komfortabel zu entwickeln, besteht im Zusammenfügen (Binden) von mehreren Objektmodulen. Diese Objektmodule erkennen Sie an der Endung »obj«. Sie werden so vom MASM erzeugt.

Jetzt erfahren Sie auch, wofür der Linker (Binder) »link« des MASM zuständig ist. Er nämlich erzeugt aus den Objektmodulen des MASM die ausführbaren Exe-Dateien. Bis jetzt haben Sie immer nur ein einziges Modul »gelinkt«, doch der Linker kann auch mehrere unabhängig voneinander entwickelte Objektmodule zu einem einzigen Programm zusammenbinden.

Die Routine »reset.asm« (Listing 1) setzt alle Diskettenpuffer von MS-DOS in ihren ursprünglichen Zustand zurück und initialisiert das aktuelle Laufwerk, das sich über die MS-DOS-Funktion 0Ehex auswählen läßt. Das Löschen der Diskettenpuffer und die Neuinitialisierung eines Laufwerks erweisen sich beispielsweise dann als nützlich, wenn die Fehlermeldung »falsche Dateizuordnungstabelle« auftritt.

Damit auch andere Programme auf das Modul »reset.asm« zugreifen können, enthält es den Pseudobefehl »public«, gefolgt von einer Sprungmarke. Dieser Public-Befehl macht diese Marke »öffentlich«, das heißt, sie läßt sich auch von anderen Modulen aus aufrufen. Wenn ein Modul mehrere Marken besitzt, die öffentlich sein sollen, so lassen sich diese Marken hintereinander, durch Komata getrennt, im Public-Befehl angeben.

Ein anderes Modul kann diese Marke nun über einen Sprung- oder Unterprogramm-Befehl aufrufen. Damit der Assembler jedoch weiß, daß diese Marke in einem externen Modul vorliegt, muß die Marke in dem aufrufenden Programm als extern deklariert werden. Dies geschieht mit dem Befehl »extrn« (extern), der von der entsprechenden Marke gefolgt sein muß.

Handelt es sich bei der Marke um eine Sprungmarke, so muß sie von ei-

nem Doppelpunkt und der Bezeichnung »near« oder »far« gefolgt sein. »near« steht für nah und bedeutet, daß sich die angegebene Sprungmarke über eine 16-Bit-Adresse erreichen läßt, sich also im gleichen Segment wie das aufrufende Programm befindet. »far« (weit) bedeutet entsprechend, daß die Sprungmarke in einem anderen Segment liegt, und daß für deren Adressierung eine 20-Bit-Adresse, bestehend aus Segmentadresse und Offsetadresse, erforderlich ist.

Die Marke kann jedoch auch die Bezeichnung für eine Variable sein. Dann muß sie von einem Doppelpunkt und der Bezeichnung »byte«, »word« oder »dword« gefolgt sein. »byte« gibt an, daß es sich bei dem Wert der Variablen um einen 1-Byte-Wert handelt, »word«, daß es sich um einen 2-Byte-Wert und »dword«, daß es sich um einen 4-Byte-Wert handelt.

Listing 2 zeigt das Programm »initial.asm«, das die Deklaration `extrn reset:near` enthält und damit die Sprungmarke »reset« des Moduls »reset.asm« als nahes Sprungziel definiert. Die Reset-Marke wird nach der Initialisierung des Segmentregisters DS über den Unterprogramm-Befehl »call« aufgerufen, so daß durch den Befehl »ret« in »reset.asm« die Rückkehr ins aufrufende Programm erfolgt. Die beiden Module »reset.obj« und »initial.obj« lassen sich mit dem Linker »link« des MASM über den Aufruf

```
link initial.obj reset.obj
```

und durch das dreimalige Betätigen der Enter-Taste zu einem lauffähigen Programm mit dem Namen »initial.exe« zusammenfügen.

Sicher werden Sie schon gemerkt haben, daß beide Module mit dem Speichermodul für Dateien mit der Endung »exe« arbeiten. Ein Kennzeichen ist, daß der Befehl »org 0100h« fehlt, dafür jedoch das Segmentregister DS zu Programmbeginn initialisiert wird. Das verwendete Speichermodell ist in diesem Fall sinnvoll, weil Assemblerprogramme bei der Modulbauweise leicht eine Länge von 64 KByte überschreiten können. Außerdem entstünden bei dem Speichermodell für Com-Dateien unlösbare Probleme bei der Adressierung, weil jedes Modul auf die Startadresse 0100hex fixiert wäre. Dadurch würden wiederum Modul-Überschneidungen auftreten, die das gesamte Programm abstürzen ließen.

Weil jedoch die beiden beschriebenen Module nur wenige Bytes belegen, sind sie im Pseudobefehl »segment« als »public« definiert. Dieser Befehl weist den Linker an, den Inhalt von gleichnamigen Speichersegmenten in ein gemeinsames Speichersegment zu legen.

Dadurch wird der Maschinencode von »reset.asm« und »initial.asm« in

ein gemeinsames Segment direkt hintereinander abgelegt. Nur so ist es auch möglich, daß sich die Sprungmarke »reset« über eine 16-Bit-Adresse ansprechen läßt.

Wären die Speichersegmente von »reset.asm« und »initial.asm« nicht als »public« deklariert, so würde der Linker den Maschinencode der beiden Module in verschiedenen Segmenten ablegen. Dadurch wäre wiederum eine 20-Bit-Adressierung der Sprungmarke »reset« erforderlich und sie müßte als »far« deklariert werden.

Mit der Fähigkeit Objektmodule zusammenzufügen besitzt der MASM eine sehr effektive Möglichkeit zur modularen Programmierung. Zwar ließen sich anstelle von Modulen auch Makros verwenden, doch würde sich dieses Verfahren bei längeren und komplexen Routinen nicht mehr auszahlen, weil bei jedem Auftritt eines Fehlers das gesamte Programm erneut übersetzt werden müßte. Dies wäre auf die Dauer sehr zeitraubend.

Weil sich Objektmodule dagegen getrennt assemblieren und testen lassen und erst gelinkt werden müssen, wenn alle Module fehlerfrei arbeiten, ist die Programmierung von Modulen die schnellere Variante der Programmentwicklung.

Eine RAM-Disk verkürzt die Assemblierungs- und Linkzeiten deutlich

Wenn Sie das beschriebene Beispiel mit den Modulen »reset.asm« und »initial.asm« einmal durchhexerzieren, werden Ihnen die langen Assemblierungs- und Linkzeiten unangenehm auffallen (sofern Sie nicht eine Festplatte besitzen). Es empfiehlt sich deshalb, bei der Arbeit mit dem MASM eine RAM-Disk zu benutzen.

Die beste Methode besteht darin, die Programme »masm.exe«, »link.exe« und einen geeigneten Texteditor in die RAM-Disk zu kopieren. Danach erklären Sie die RAM-Disk mit den Befehlen »append« und »path« zum Standardsuchverzeichnis. Dann können Sie auf eines der Diskettenlaufwerke oder auf die Festplatte umschalten.

MS-DOS schreibt ab diesem Zeitpunkt alle Dateien auf das aktive Disketten- oder Festplatten-Laufwerk, durchsucht aber beim Lesen von Dateien auch die RAM-Disk. Dies gewährleistet schnelles Arbeiten, und daß die wertvollen Daten können nicht verlorengehen. Beim Assemblieren und Linken von »reset.asm« und »initial.asm« benötigt der MASM auf dem Schneider PC ohne RAM-Disk beispielsweise 23 Sekunden, während sich die Wartezeit mit RAM-Disk auf 10 Sekunden verkürzt. Der Textkasten zeigt die Konfigurationsdatei und die Stapeldateien, die

notwendig sind, um die RAM-Disk wie beschrieben zu installieren. Nun ist es wieder Zeit für ein selbst-geschriebenes Assemblerprogramm. Nach »Print« und »Input« soll diesmal der MS-DOS-Befehl »dir« simuliert werden. Das hört sich schwer an, ist es aber nicht, denn auch zur Disketten- und Dateiverwaltung stellt der Software-Interrupt 21hex dem Programmierer leistungsfähige MS-DOS-Funktionen zur Verfügung. Listing 3 zeigt das Assemblerprogramm »showdir.asm«, das alle Dateien im aktuellen Verzeichnis einer Diskette oder Festplatte auf den Bildschirm ausgibt. Kernstück dieses Programms sind die MS-DOS-Funktionen 11hex (ersten Eintrag suchen) und 12hex (nächsten Eintrag suchen). Vor dem Aufruf dieser beiden Funktionen muß im DX-Register die Adresse eines Datenbereichs angegeben werden, der die gesuchte Datei spezifiziert. Diesen Datenbereich nennt man Dateisteuerblock, oder kurz: FCB (file control block).

Ohne FCB und DTA läuft nichts beim Zugriff auf Diskette oder Festplatte

Die Tabelle 2 zeigt den Aufbau eines FCB. Der FCB wird für die Parameterübergabe von und zu MS-DOS benutzt. Neben dem FCB gibt es noch einen erweiterten FCB (extended FCB), der zusätzliche Parameter zu einer Datei enthält, doch diese FCB-Art braucht uns nicht zu interessieren.

Schauen Sie sich die Definition des FCB in »showdir« einmal genauer an. Um alle Dateien eines beliebigen Verzeichnisses zu finden, muß das erste Byte des FCB mit

```
db 0
auf 0 gesetzt werden, so daß jeweils das aktuelle Laufwerk angesprochen ist. Der Dateiname im FCB muß mit »*.« angegeben werden, damit das Programm auch wirklich alle Dateien findet. Hier ist jedoch darauf zu achten, daß die restlichen Bytes des Dateinamens im FCB mit Leerzeichen aufgefüllt sind. Also
```

```
db " " " " " "
Der übrige Teil des FCB braucht mit db 28 dup(?) lediglich reserviert werden. Eine Definition ist unnötig, weil diese bei Be-
```

Der Aufbau des Dateisteuerblocks (FCB)

Adresse	Länge in Bytes	Bedeutung
00	1	Laufwerksnummer (0 = aktuelles Laufwerk, A = 1, B = 2, und so weiter)
01	8	Dateiname (bei Bedarf mit Leerzeichen aufgefüllt)
09	3	Dateityp (bei Bedarf mit Leerzeichen aufgefüllt)
0C	2	Aktuelle Datenblocknummer (Voreinstellung: 0)
0E	2	Datenblocklänge (Voreinstellung: 128)
10	2	Dateilänge (Offset)
12	2	Dateilänge (Segment)
14	2	Datum des letzten Schreibzugriffs
		Bit 0 bis 4: Tag
		Bit 5 bis 8: Monat
		Bit 9 bis F: Jahr (0 = 1980, 1 = 1981, und so weiter)
16	2	Uhrzeit des letzten Schreibzugriffs
		Bit 0 bis 4: Sekunden (1 = 2 Sekunden, 2 = 4 Sekunden, usw.)
		Bit 5 bis A: Minuten
		Bit B bis F: Stunden
18	8	Für MS-DOS reserviert
20	1	Aktueller Datensatz (sequentiell)
21	2	Aktueller, relativer Datensatz (Offset)
23	2	Aktueller, relativer Datensatz (Segment)
25	3	Nicht verwendet

Tabelle 2: Der Dateisteuerblock FCB regelt den Zugriff auf Dateien, die sich auf Diskette befinden

darf von den MS-DOS-Funktionen des Interrupt 21hex vorgenommen wird.

Wenn »showdir.asm« eine Datei gefunden hat, erfolgt die Ausgabe ihres Namens durch einen Trick. Der Zugriff auf Diskette oder Festplatte erfordert nämlich die Definition eines Speicherbereichs für die Datenübertragung von und zu der Diskette. Dieser Speicherbereich wird mit DTA (disk transfer area) bezeichnet und über die MS-DOS-Funktion 1Ahex definiert.

Nun ist nach einem Dateizugriff der Name der Datei immer ab dem zweiten Byte (Marke »filename«) des DTA abgelegt. Weil der Name acht Zeichen lang ist, reicht er folglich bis zum neunten Byte (Marke »continue«) des DTA. Die Startadresse dieses Dateinamens, das heißt die Marke »filename«, wird nun (im Makro) an das DX-Register übergeben, so daß die inzwischen altbekannte MS-DOS-Funktion 09hex diesen Namen ausgeben kann. Damit die Funktion auch das Ende des Dateinamens erkennt, hat »showdir.asm« zuvor das dem Dateinamen folgende Byte (Marke »continue«) kurzerhand mit dem Dollarzeichen belegt.

Wenn Sie »showdir.asm« korrekt abgetippt und ohne Fehlermeldungen assembliert und gelinkt haben, können Sie es gefahrlos an Ihren Disketten oder auf der Festplatte ausprobie-

ren. Sie werden überrascht sein, wie viele Dateinamen das Programm auf einen einzigen Bildschirm ausgeben kann.

(Martin Althaus/Markus Zietlow/ma)

So läßt sich ein PC ohne Festplatte für die Programmierung mit dem MASM einrichten:

```
Die Datei »config.sys«:
files=20
buffers=5
device=ramdrive.sys 189
rem ODER: device=vdisk.sys 189
country=049
```

```
Die Datei »autoexec.bat«:
echo off
keybgr
mouse rem MOUSE-Befehl nur,
wenn Maus-Treiber vorhanden
echo on
assem
```

```
Die Datei »assem.bat«:
echo off
echo.
copy ascopy.bat c: > nul
c:
ascopy
```

```
Die Datei ascopy.bat:
echo off
copy a:masm.exe c: > nul
copy a:link.exe c: > nul
copy a:edlin.com c: > nul
rem ODER anderen Editor statt edlin einsetzen
copy a:append.com c: > nul
append c: \
path = c: \
echo.
echo Bitte entfernen Sie die Assembler-Diskette
echo und legen Sie eine Datendiskette ein!
pause
cls
a:
edlin
```

Inhaltsverzeichnis der Assembler-Diskette:

Volume in Laufwerk A ist ASSEMBLER
Verzeichnis von A: \

COMMAND.COM	(MS-DOS)	RAMDRIVE.SYS	(RAM-Disk-Treiber)
CONFIG.SYS	(siehe oben)	MOUSE.COM	(Maus-Treiber)
AUTOEXEC.BAT	(siehe oben)	APPEND.COM	(Append-Befehl)
ASSEM.BAT	(siehe oben)	EDLIN.COM	(Editor nach eigener Wahl)
ASCOPY.BAT	(siehe oben)	MASM.EXE	(MASM)
KEYBGR.EXE	(deutsche Tastatur)	LINK.EXE	(Linker des MASM)

...sowie die versteckten Dateien von IO.SYS und MSDOS.SYS


```

0:          title      reset
1:          ;
2:          ; Dieses Programm löscht alle Diskettenpuffer
3:          ; und initialisiert das Standardlaufwerk
4:          ;
5: funktion  macro      nummer      ;Makro für den Aufruf
6:          mov        ah,nummer    ;der MS-DOS-Funktionen
7:          int        21h
8:          endm
9:
10: public   reset      ;Öffentliches Modul
11: assume   cs:code,ds:code ;Segmentzuweisungen
12:
13: code     segment     public      ;Öffentliches Segment
14: reset:    mov        ax,code      ;DS initialisieren
15:          mov        ds,ax        ;((für EXE-Dateien erforder
16:          rlich)
17:
18: funktion ODh      ;alle Diskettenpuffer lös
19:          chen
20:          ; und Laufwerk initialisi
21:          eren
22:          ret
23:          ;In aufrufendes Modul
24:          ; zurückkehren
25:
26: code     ends
27:          end          reset      ;Segmentende
28:          ;Programmende

```

Listing 1. Das Assemblerprogramm »reset.asm« löscht die Diskettenpuffer und initialisiert das aktuelle Standardlaufwerk

```

0:          title      initial
1:          ;
2:          ; Dieses Programm ruft das Modul »reset.asm« auf
3:          ; und gibt anschließend eine Statusmeldung aus
4:          ;
5: funktion  macro      nummer      ;Makro für Aufruf
6:          mov        ah,nummer    ;von MS-DOS-Funktionen
7:          int        21h
8:          endm
9:
10: extrn    reset:near      ;Externe Sprungmarke über
11:          ;16-Bit-Adresse zu erreich
12:
13: hen
14: assume   cs:code,ds:code ;Segmentzuweisungen
15: code     segment     public      ;Öffentliches Segment
16:
17: start:    mov        ax,code      ;DS initialisieren
18:          mov        ds,ax        ;((für EXE-Dateien erforder
19:          rlich)
20:          call       reset        ;Modul »reset.asm« aufruf
21:          en
22:          mov        dx,offset text
23:          funktion  09h          ;Zeichenkette ausgeben
24:          funktion  4Ch          ;Programm beenden
25:
26: text      db 13,10,"Laufwerk ist zurückgesetzt.",13,10
27:          ;"$"
28:
29: code     ends
30:          end start

```

Listing 2. Das Programm »initial.asm« benutzt »reset.asm« als Modul

```

11: assume   cs:code,ds:data,ss:stack
12:
13: ngen
14: data     segment
15:          art
16:          db 0
17:          fcb
18:          werk
19:          db " "
20:          db 28 dup(?)
21:          e für
22:          en
23:          dta
24:          ngsbereich
25:          filename
26:          continue
27:          Byte
28:          text
29:          data
30:          de
31:          stack
32:          nfang
33:          lbereich
34:          tos
35:          elspitze
36:          )
37:          stack
38:          nde
39:          code
40:          tanfang
41:          start:
42:          ren
43:          ren
44:          ren
45:          ren
46:          find:
47:          eren
48:          chen
49:          ?
50:          Fehler
51:          loop:
52:          für
53:          x hinter
54:          DTA setzen
55:          geben
56:          er
57:          on 09hex
58:          reinander
59:          suchen
60:          ?
61:          Ende
62:          Ausgeben
63:          53:
64:          nofile:
65:          en
66:          ende:
67:          en
68:          code
69:          58:

```

Listing 3. Mit dem Assemblerprogramm »showdir.asm« läßt sich das Verzeichnis einer Diskette auf den Bildschirm ausgeben

```

0:          title      showdir
1:          ;
2:          ; dieses Programm zeigt die Namen der Dateien an, die
3:          ; sich auf der Diskette im aktuellen Laufwerk befinden
4:          ;
5: funktion  macro      variable,nummer ;Makro für
6:          mov        dx,offset variable ; Funktions-
7:          mov        ah,nummer        ; aufrufe
8:          int        21h
9:          endm
10:

```


SAM

Viele Beiträge dieser Ausgabe zeigen die fantastischen Fähigkeiten von Assemblerprogrammen auf. Wenn Sie auch auf den Geschmack gekommen sind, aber noch keinen Assembler besitzen, finden Sie auf der Databox genau das Richtige: SAM, der Assembler von DOS International.

der DOS-Assembler

Für den PC-Besitzer ist die Auswahl an Assemblern recht spärlich. Ein sehr leistungsfähiges Produkt ist der Makroassembler MASM von Microsoft, doch dieses Programm kostet knapp 400 Mark und ist damit nahezu unerschwinglich. Eine günstige Alternative ist »sam« auf der Databox zu dieser Sonderausgabe.

SAM ist ein 2-Pass-Assembler und übersetzt Ihre Assemblerprogramme im Handumdrehen in Maschinencode. Sie werden von SAMs Geschwindigkeit überrascht sein. Der Assembler legt den assemblierten Maschinencode als Datei mit der Endung »com« auf Diskette ab, so daß sich die übersetzten Programme direkt aus

der MS-DOS-Kommandoebene durch Eingabe des Programmnamens starten lassen.

Die Assemblerprogramme für SAM lassen sich mit jeder Textverarbeitung schreiben, die ASCII-Dateien erzeugt. Im Zweifelsfall tut es auch der MS-DOS-Zeileneditor »edlin.com«. Natürlich kann SAM nicht all das, was der MASM kann. Doch gerade für den Einsteiger und Gelegenheitsprogrammierer ist SAM ideal, weil seine Bedienung wesentlich einfacher, als die des MASM ist. So benötigt SAM beispielsweise nur einen einzigen Arbeitsgang, um aus einem Assemblerprogramm den lauffähigen Maschinencode zu generieren. Das

oft zeitraubende »Linken« ist bei SAM nicht notwendig.

Bei SAM müssen Sie keine Segmente zuweisen, keine Segmentregister initialisieren und damit auch keine Speichersegmente definieren. Die Assembler-Pseudobefehle »assume«, »segment« und »ends« sind damit überflüssig.

SAM erzeugt vielmehr Programme, die nach dem Speichermodell für Com-Dateien aufgebaut sind. Dieses Speichermodell arbeitet mit nur einem Segment, das beim Programmstart automatisch initialisiert wird. Dadurch lassen sich zwar nur Maschinencode-Programme bis zu einer Größe von 64 KByte schreiben, doch

SAMs Pseudobefehle

Befehl	Bedeutung
db	Definiert ein oder mehrere Byte
ds	Reserviert einen Speicherbereich
end	Logisches Programmende
endp	Ende einer Prozedur
equ	Wertzuweisung für eine Variable
org	Definiert die Startadresse des Programms innerhalb des Segments
proc	Anfang einer Prozedur

Tabelle. Der Pseudobefehlssatz von SAM ist zwar klein, enthält jedoch alle notwendigen Befehle

```

0: ; Dieses Programm gibt auf dem Bildschirm
1: ; eine Zeichenkette aus und kehrt darauf
2: ; in die MS-DOS-Kommandoebene zurück
3:
4:      org      100h      ;Programm startet ab 100hex
5:      ;          ;(für COM-Dateien erforderlich)
6: start  jmps    anfang   ;Sprung zum Programmanfang
7:
8:      ;          Datenbereich:
9:
10: text   db 13,10,'Mein erstes Assemblerprogramm',13,10,'$'
11:
12:      ;          Programmbereich:
13:
14: anfang  mov dx,offset(text);Adresse der Zeichenkette
15:          mov ah,09h      ;Zeichenkette über Interrupt 21hex,
16:          int  21h        ; Funktion 09hex auf den
17:          ;          ; Bildschirm ausgeben
18:          mov ah,4Ch      ;Programm über Interrupt 21hex,
19:          int  21h        ; Funktion 4Chex beenden
20:
21:          end      start  ;Programmende und Definition
22:          ;          ; des Startpunktes
23:

```

Listing. An diesem Beispiel sehen Sie deutlich die Unterschiede zu einem MASM-Listing

selbst leistungsfähige Textverarbeitungen (wie zum Beispiel PC-Write 2.4) lassen sich in diesem Rahmen unterbringen.

SAM verarbeitet keine Makros, so daß die Befehle »macro« und »endm« nicht akzeptiert werden. Dafür lassen sich jedoch – wie beim MASM – Prozeduren mit den beiden Befehlen »proc« und »enp« definieren. Sie dürfen dabei die Prozeduren auch verschachteln.

Weil der Parser (der Programmteil, der das Assemblerprogramm in Marken, Befehle, Operanden und Kommentare zerlegt), einfach (aber schnell) arbeitet, müssen Sie bei der Adressierung über einen Offset-Befehl die Marke, deren Adresse ermittelt werden soll, in runde Klammern einschließen. Zwischen »offset« und der Marke darf kein Leerraum stehen. Sonst erkennt der Parser die Marke aufgrund des Leerzeichens nämlich nicht. Beim Zerlegen der Programmzeile würde der Rest einfach abgeschnitten. Ein Beispiel:

```
mov ax, offset(text)
```

Aus dem gleichen Grund muß das Displacement (Verschiebung) bei der indirekten Adressierung ohne Pluszeichen direkt vor die eckige Klammer, die die Quell- oder Zieladresse enthält, geschrieben werden.

Ein paar Beispiele:

```
mov AX, 80h(BX+SI)
mov CX, index(DX)
mov -16(DI), SI
```

SAM unterscheidet übrigens nicht zwischen Groß- und Kleinschreibung, so daß die Schreibweise Ihnen überlassen bleibt. Die Tabelle zeigt eine Aufstellung der Pseudobefehle von SAM.

Mit dem Pseudobefehl »ds« reservieren Sie einen Speicherbereich – beispielsweise für einen Puffer. Der Befehl

```
puffer ds 100h,0
```

reserviert ab der Marke »puffer« einen Speicherbereich von 100hex Byte, der mit dem Wert 0 gefüllt wird.

Wie Sie sehen, sind weniger wichtige Pseudobefehle wie »name«, »page«, »title« oder »subttl« bei SAM nicht

implementiert, doch die zum Arbeiten grundlegenden Befehle sind alle vorhanden.

SAM ist ein schneller, vollwertiger Assembler für jeden PC-Besitzer

Der Aufbau einer Assembler-Programmzeile für SAM entspricht dem üblichen Standard. Jede Zeile besteht aus maximal vier Feldern, die von links beginnend Marken und Variablen, Befehle, Operanden und Kommentare enthalten.

Die linke Spalte ist nur für Marken und Variablen vorgesehen.

Die Marken werden bei SAM wie Variablen grundsätzlich ohne Doppelpunkt angegeben. Kommentartexte müssen Sie – wie üblich – mit einem Semikolon anführen, und in jeder Programmzeile muß der Kommentar von neuem durch ein Semikolon gekennzeichnet werden.

Texte dürfen nur in einfache Hochkommas (»'«) eingeschlossen werden – das doppelte Anführungszeichen »“« ist nicht zulässig.

Die Sprungbefehle »jmp« und »call« müssen bei SAM direkt von einem »n« oder »f« gefolgt sein, wenn die Zieladresse zu den Befehlen in einem Register angegeben ist.

Das »n« (near) steht für einen kurzen Sprung innerhalb des Segments (Intrasegment), und das »f« (far) gibt entsprechend einen weiten Sprung über die Segmentgrenzen hinweg (Intersegment) an. Ein Beispiel:

```
call n (BX)
```

Der Befehl »jmp« muß bei SAM als »jmps« (jump short) geschrieben werden, wenn er einen kurzen Sprung (-128 bis +127 Byte) bezeichnet.

Ist der Befehl dagegen für einen längeren Sprung gedacht (über 128 Byte), so muß er als »jmp« geschrieben werden. Achten Sie jedoch darauf, daß diese Variante nur den Sprung vorwärts erlaubt.

»jump« arbeitet nämlich nur mit einer vorzeichenlosen 16-Bit-Adresse. Möchten Sie Daten über die Programmsegmentgrenzen hinaus schreiben oder lesen (beispielsweise in den Bildschirmspeicher), so müssen Sie das

Segmentregister DS mit der gewünschten Adresse laden.

Wollen Sie beispielsweise den Wert des ersten Byte des Bildspeichers einer CGA-Farbgrafikkarte in das AL-Register einlesen, so müssen Sie

```
mov ax, B800h ;Segmentadresse
mov ds, ax ;in DS schreiben
mov si, 0 ;und erste Speicherzelle auslesen
```

schreiben. Auch ein Sprung über die Segmentgrenzen hinweg, beispielsweise in das BIOS, ist erlaubt:

```
basis equ F000h
adresse equ 1000h
jmp adresse, basis
```

und ein entsprechender Unterprogrammaufruf sieht folgendermaßen aus:

```
bios equ F000h
adresse equ 2000h
mov (di), adresse
mov 2(di), bios
callf (di)
```

Die jeweils angegebenen BIOS-Adressen sind natürlich rein fiktiv und von der BIOS-Version Ihres Computers abhängig.

Und noch eine letzte Besonderheit von SAM: Die Präfixbefehle »lock«, »rep« und »seg«, die normalerweise direkt vor den betreffenden Befehl gehören, müssen bei SAM in eine eigene Zeile geschrieben werden, damit sie korrekt verarbeitet werden. Das Listing zeigt als konkretes Beispiel ein kleines Assemblerprogramm, das an SAM angepaßt ist. Die Unterschiede zu einem MASM-Listing können Sie feststellen, wenn Sie das Programm mit dem Programm aus dem ersten Teil des Assemblerkurses in dieser Ausgabe vergleichen. Beide Programme haben nämlich dieselbe Funktion.

Aufmerksamen Leser, die sich in der Public-Domain-Szene auskennen, wird nicht entgangen sein, daß SAM einige Ähnlichkeiten zu dem Assembler »chasm« aufweist.

So ist SAM zu »chasm« voll befehlskompatibel und arbeitet mit dem gleichen Fehlercodes. Wenn Sie sich mit »chasm« bereits auskennen, müssen Sie daher nicht umlernen, um in den Genuß des schnelleren (und deutlichen) SAM zu kommen.

(Andreas Mauerbrecher/ma)

Um nicht für jeden benötigten Wert umständlich und zeitraubend auf den Arbeitsspeicher des Computers zugreifen zu müssen, verfügen Prozessoren über eigene, kleine interne Speichereinheiten, die sogenannten Register. Auf diese Register kann der Prozessor nun besonders schnell zugreifen und dadurch Zeit sparen. Je

Die erste Gruppe der 8086/8088-Register ist für alle Arten von Daten zuständig. Das AX-Register (accumulator) setzt sich aus den beiden 8-Bit-Registern AL und AH zusammen. AX ist der Akkumulator des 8086/8088, der primär für arithmetische und logische Operationen zuständig ist. Das BX-Register (base register) besteht aus den beiden 8-Bit-Registern BL

Die dritte Gruppe der 8086/8088-Register sind die Segmentregister. Diese Register geben das jeweilige Speichersegment für die Daten- und Adreßregister sowie den Befehlszeiger an. Das Programmsegment CS (code segment) ist für den Befehlszeiger IP, das Datensegment DS (data segment) für das Quellregister SI und Zielregister DI, das Stapelsegment SS (stack segment) für den Stapelzeiger SP und Basiszeiger BP, sowie das Extrasegment ES (extra segment) für das

The diagram illustrates the internal structure of the 8085 microprocessor. It shows a 16-bit Program Counter (PC) and an 8-bit Status Register. The PC is divided into two 8-bit halves, labeled 'F' (high byte) and 'O' (low byte). The Status Register is divided into two 4-bit halves, labeled 'PSW' (high byte) and 'C' (low byte). The bits in the Status Register are labeled 'O', 'D', 'I', 'T', 'S', 'Z', 'A', 'P', and 'C'.

DOS EXTRA 1'87/88

Die Adressierungsarten des 8086/8088	
Adressierungsart	Berechnung der effektiven Adresse (EA)
Unmittelbar	Operand steht im Befehl
Direkt	EA = Offset
Indirekt	EA = (BX) oder (BP) oder (SI) oder (DI)
Basis	EA = [(BX) oder (BP)] + Offset
Indiziert	EA = [(SI) oder (DI)] + Offset
Indiziert Basis	EA = [(BX) oder (BP)] + [(SI) oder (DI)]
Indiziert Basis mit Offset	EA = [(BX) oder (BP)] + [(SI) oder (DI)] + Offset

Tabelle: Die Adressierungsarten des 8086/8088 im Überblick

Zielregister DI bei Zeichenketten-Operationen zuständig.

Die vierte und letzte Gruppe der 8086/8088 umfaßt nur zwei Register. Da ist zum einen der Befehlszeiger IP (instruction pointer), der auf den nächsten in die Befehlswarteschlange zu übernehmenden Befehl zeigt, und zum anderen das Statusregister PSW (processor status word), das wiederum aus den einzelnen Statusbits besteht. Das Statusregister PSW bilden folgende Statusbits:

- Bit 0: Übertrag-Statusbit C (carry flag)
- Bit 2: Parität-Statusbit P (parity flag)
- Bit 4: Hilfsübertrag-Statusbit A (auxiliary carry flag)
- Bit 6: Null-Statusbit Z (zero flag)
- Bit 7: Vorzeichen-Statusbit S (sign flag)
- Bit 8: Einzelschritt-Statusbit T (trace flag)
- Bit 9: Interrupt-Statusbit I (interrupt flag)
- Bit A: Richtung-Statusbit D (direction flag)
- Bit B: Überlauf-Statusbit O (overflow flag)

Um mit dem 8086/8088 eine bestimmte Speicherzelle im Arbeitsspeicher oder Festwertspeicher des PC zu adressieren, existieren sieben verschiedene Methoden.

1.) Unmittelbare Adressierung

Bei der unmittelbaren Adressierung wird der Operand direkt im Befehl angegeben. Beispiel: ADD AX,7FFFH.

2.) Direkte Adressierung

Bei der direkten Adressierung ist die Adresse des Operanden im Befehl mit angegeben und wird vom Prozessor unter Berücksichtigung der Segmentadresse (in der Regel DS) verwertet. Beispiel: MOV CX,0FFFFH.

3.) Indirekte Adressierung

Bei der indirekten Adressierung enthält das Register BX, BP, SI oder DI die Adresse des Operanden. Beispiel: MOV DX,[SI].

4.) Basis-Adressierung

Bei der Basis-Adressierung enthält das Basisregister BX oder BP die Basisadresse des Operanden, zu der ein Offset, der im Befehl angegeben sein muß, addiert wird. Beispiel: MOV DX,[BX + 8000H].

5.) Indizierte Adressierung

Bei der indizierten Adressierung enthält das Indexregister SI oder DI die Basisadresse des Operanden, zu der ein Offset, der im Befehl angegeben sein muß, addiert wird. Beispiel: MOV DX,[SI + 8000H].

6.) Indizierte Basis-Adressierung

Die indizierte Basis-Adressierung ist eine Kombination aus der Basis-Adressierung und der indizierten Adressierung. Hier bildet sich die effektive Adresse des Operanden aus dem Inhalt des Basisregisters BX oder BP, zu dem der Inhalt des Indexregisters SI oder DI addiert wird. Beispiel: MOV DX,[BX + SI].

7.) Indizierte Basis-Adressierung mit Offset

Diese Adressierung entspricht der indizierten Basis-Adressierung, nur wird zu der Summe des Basis- und Indexregister-Inhaltes zusätzlich ein Offset addiert, der im Befehl angegeben sein muß. Beispiel: MOV DX,[BX + SI + 8000H].

Die Tabelle zeigt die sieben Adressierungsarten nochmals im Überblick.

Die Speicherverwaltung erfolgt beim 8086/8088 über die Segmentregister

Der 8086/8088-Prozessor kann mit seinen 20 Adreßleitungen 1 MByte an Speicherbereich adressieren. Weil sämtliche Register des 8086/8088 jedoch nur 16 Bit breit sind, läßt sich mit dieser Bitbreite nur ein Speicherbereich von 64 KByte ansprechen.

Um nun den vollen Speicherbereich des 8086/8088 adressieren zu können, wurden die Segmentregister eingerichtet, die jeweils ein 64-KByte-Segment für eine Adresse in einem Nicht-Segmentregister bestimmen und über eine bestimmte Verknüpfung, die noch erläutert wird, die 16-Bit-Adressen dieser Register auf 20 Bit vervollständigen.

Das Programmsegment CS (code segment) ist für die Adresse im Befehlszeiger, das Datensegment DS (data segment) für eine Adresse im Quell- oder Zielregister, das Stapelsegment SS (stack segment) für die Adressen im Stapel- oder Basiszeiger und das Extrasegment ES (extra segment) für eine Adresse im Zielregister bei Zeichenketten-Operationen zuständig.

Die effektive Adresse aus einem Nicht-Segmentregister und dem zugehörigen Segmentregister wird gebildet, indem zu dem Wert des Registers, das die 16-Bit-Adresse enthält, der mit 10 (hex) multiplizierte Wert des Segmentregisters addiert wird.

Zwei Beispiele:

Enthält SI den Wert 8000 und DS den Wert B000, so beträgt die effektive Adresse $8000 + B000 \times 10 = B8000$. Enthält IP die Adresse 0000 und CS den Wert FFFF, so beträgt die effektive Adresse $0000 + FFFF \times 10 = FFFF0$.

Für Sprünge (»jp«), Unterprogrammaufrufe (»call«) und die Rückkehr aus Unterprogrammen (»ret«) stehen grundsätzlich zwei verschiedene Arten der Adressierung zur Verfügung. Die erste Art der Adressierung ist die Intersegment-Adressierung mit 20 Bit, die aus einer 16-Bit-Adresse und einer 16-Bit-Segmentadresse besteht und die Adressierung über die Segmentgrenzen hinweg (far call) erlaubt. Die zweite Art der Adressierung ist die Intra-segment-Adressierung, die nur aus einer 16-Bit-Adresse besteht und sich jeweils auf das aktuelle Speichersegment (near call) bezieht.

(ma)

Wer — wo — wann?

Bei Zuschriften bitten wir Folgendes zu beachten: Haben Sie Fragen, Anregungen oder Programme, die Sie der DOS-Redaktion zukommen lassen möchten, adressieren Sie wie folgt:

Redaktion · DOS International · Elfenstr. 40 · 8000 München 43 · Tel. (089) 60 20 66

Haben Sie Fragen, die zum Beispiel Abonnements, Kleinanzeigen oder Databox-Bestellungen betreffen, wenden Sie sich bitte an den Verlag:

DMV Verlag GmbH · DOS International · Fuldaer Straße 6 · 3440 Eschwege
Tel. (056 51) 87 02 · Telex 993 210 dmv d

Die Assemblerbefehle des PC

Assembler ist für viele PC-Besitzer ein Buch mit sieben Siegeln. Dabei läßt sich beispielsweise mit dem MASM von Microsoft oder dem SAL von Lauer&Wallwitz ähnlich komfortabel programmieren, wie mit einer höheren Sprache. Nur die Befehle und Register lassen sich – zu-gegebenermaßen – schwer im Kopf behalten. Deshalb finden Sie hier Referenztabellen, die das ewige Nachschlagen in Handbüchern überflüssig machen.

Die meisten PC-Besitzer programmieren zwar in GW-Basic und Turbo Pascal, doch eigentlich läßt sich der Prozessor des PC (und das gilt auch für jeden anderen Computer) nur in Maschinencode programmieren. Bei GW-Basic übersetzt deshalb der Interpreter ein Programm während des Ablaufs Befehl für Befehl in Maschinencode und Turbo Pascal-Programme werden von einem Compiler zuerst vollständig in Maschinencode übertragen, bevor sie überhaupt ablauffähig sind.

Wenn man nun direkt den Maschinencode programmieren würde, könnte das Programm viel Zeit spa-

ren, weil der Übersetzungsvorgang vor oder während des Programmablaufs entfallen würde, das leuchtet ein. Doch Maschinencode besteht nur aus endlosen Zahlenkolonnen, die für den Menschen, den Programmierer also, völlig unübersichtlich sind. Aus diesem Grund haben sich die Entwickler von Mikroprozessoren zu allen Maschinencode-Befehlen kurze, aber verständliche Bezeichnungen (zum Beispiel ADD für »addieren«, INT für »unterbrechen« oder OUT für »ausgeben«) einfallen lassen. Diese Kurzbezeichnungen – auch Assemblerbefehle genannt – vereinfachen das Programmieren der Prozessoren

erheblich. Um die Assemblerbefehle jedoch wieder in Maschinencode umzuformen, ist der Einsatz eines Übersetzungsprogramms, des Assemblers, erforderlich.

Tabelle 1 zeigt alle Assemblerbefehle des 8086/8088-Prozessors, der im PC eingesetzt wird, jeweils mit der ausführlichen englischen Bezeichnung und der direkten, deutschen Übersetzung. Tabelle 2 erläutert die grundlegende Funktion jedes Assemblerbefehls. Tabelle 3 listet die Assemblerbefehle alphabetisch geordnet mit dem passenden Maschinencode und genauer Bedeutung auf. Diese Tabelle ist besonders hilfreich bei der Suche nach einem bestimmten Assembler-Befehl und beim Übersetzen des Befehls in Maschinencode. Tabelle 4 führt dagegen die Assemblerbefehle nach dem Maschinencode numerisch geordnet auf. Sie können mit dieser Tabelle kurze Maschinencode-Listings sehr schnell von Hand übersetzen, wenn Ihnen gerade kein Disassembler zur Verfügung steht. Die Bedeutungen und Codierungen der im Maschinencode verwendeten Parameter und Abkürzungen gehen aus den Tabellen 5 bis 11 hervor. (ma)

Die 8086/8088-Befehle in Englisch und Deutsch

Abkürzung	Englisch	Deutsch	Abkürzung	Englisch	Deutsch
AAA	ASCII adjust for addition	ASCII-Anpassung für Addition	JAE	Jump if above or equal	Springe, wenn darüber oder gleich
AAD	ASCII adjust for divide	ASCII-Anpassung für Division	JB	Jump if below	Springe, wenn darunter
AAM	ASCII adjust for multiply	ASCII-Anpassung für Multiplikation	JBE	Jump if below or equal	Springe, wenn darunter oder gleich
AAS	ASCII adjust for subtract	ASCII-Anpassung für Subtraktion	JCXZ	Jump if CX equal zero	Springe, wenn CX gleich 0
ADC	Add with carry flag	Addiere mit Übertrag-Statusbit	JE	Jump if equal	Springe, wenn gleich
ADD	Add	Addiere	JG	Jump if greater	Springe, wenn größer
AND	And	Und	JGE	Jump if greater or equal	Springe, wenn größer oder gleich
CALL	Call	Rufe auf	JL	Jump if less	Springe, wenn kleiner
CBW	Convert byte to word	Wandle Byte in Wort um	JLE	Jump if less or equal	Springe, wenn kleiner oder gleich
CLC	Clear carry flag	Lösche Übertrag-Statusbit	JMP	Jump	Springe
CLD	Clear direction flag	Lösche Richtung-Statusbit	JNA	Jump if not above	Springe, wenn nicht darüber
CLI	Clear interrupt flag	Lösche Unterbrechung-Statusbit	JNAE	Jump if not above and not equal	Springe, wenn nicht darüber und nicht gleich
CMC	Complement carry flag	Invertiere Übertrag-Statusbit	JNB	Jump if not below	Springe, wenn nicht darunter
CMP	Compare	Vergleiche	JNBE	Jump if not below and not equal	Springe, wenn nicht darunter und nicht gleich
CMPS	Compare string	Vergleiche Zeichenkette	JNE	Jump if not equal	Springe, wenn nicht gleich
CMPSB	Compare string byte-wise	Vergleiche Zeichenkette byte-wise	JNG	Jump if not greater	Springe, wenn nicht größer
CMPSW	Compare string word-wise	Vergleiche Zeichenkette word-wise	JNGE	Jump if not greater and not equal	Springe, wenn nicht größer und nicht gleich
CWD	Convert word to double word	Wandle Wort in Doppelwort um	JNL	Jump if not less	Springe, wenn nicht kleiner
DAA	Decimal adjust for addition	Dezimale Anpassung für Addition	JNLE	Jump if not less and not equal	Springe, wenn nicht kleiner und nicht gleich
DAS	Decimal adjust for subtract	Dezimale Anpassung für Subtraktion	JNO	Jump if not overflow	Springe, wenn nicht Überlauf
DEC	Decrement	Vermindere	JNP	Jump if not parity	Springe, wenn nicht Parität
DIV	Divide (unsigned)	Dividiere (vorzeichenlos)	JNS	Jump if not sign	Springe, wenn nicht Vorzeichen (positiv)
ESC	Escape (to external device)	Entweiche (zu externer Einheit)	JNZ	Jump if not zero	Springe, wenn nicht 0
HLT	Halt	Halt	JO	Jump if overflow	Springe, wenn Überlauf
IDIV	Integer divide (signed)	Ganzzahlige Division (vorzeichenbehaftet)	JP	Jump if parity	Springe, wenn Parität
IMUL	Integer multiply (signed)	Ganzzahlige Multiplikation (vorzeichenbehaftet)	JPE	Jump if parity even	Springe, wenn Parität gerade
IN	Input from	Lese ein von	JPO	Jump if parity odd	Springe, wenn Parität ungerade
INC	Increment	Erhöhe	JS	Jump if sign	Springe, wenn Vorzeichen (negativ)
INT	Interrupt	Unterbreche	JZ	Jump if zero	Springe, wenn 0
INTO	Interrupt if overflow	Unterbreche, wenn Überlauf			
IRET	Interrupt return	Kehre von Unterbrechungsprogramm zurück			
JA	Jump if above	Springe, wenn darüber			

Fortsetzung Tabelle 1 auf Seite 81

Abkürzung	Englisch	Deutsch	Abkürzung	Englisch	Deutsch
LAHF	Load AH with flags	Lade AH mit den Statusbits	REP	Repeat CX times	Wiederhole CX-mal
LDS	Load pointer to data segment	Lade Zeiger in Datensegment	REPE	Repeat CX times while equal	Wiederhole CX-mal solange gleich
LEA	Load effective address to register	Lade effektive Adresse in Register	REPNE	Repeat CX times while not equal	Wiederhole CX-mal solange nicht gleich
LES	Load pointer to extra segment	Lade Zeiger in Extrasegment	REPZ	Repeat CX times while not zero	Wiederhole CX-mal solange nicht gleich 0
LOCK	Bus lock prefix	Busverriegel-Vorzeichen	REPZ	Repeat CX times while zero	Wiederhole CX-mal solange 0
LODS	Load string to AL/AX	Lade Zeichenkette in AL/AX	RET	Return from call	Kehre von Unterprogramm zurück
LODSB	Load string byte-wise to AL	Lade Zeichenkette byte-wise in AL	ROL	Rotate left	Rotiere nach links
LODSW	Load string word-wise to AX	Lade Zeichenkette wort-wise in AX	ROR	Rotate right	Rotiere nach rechts
LOOP	Loop CX times	Führe Schleife CX-mal aus	SAHF	Store AH into flags	Speichere AH in die Statusbits
LOOPE	Loop CX times while equal	Führe Schleife CX-mal aus solange gleich	SAL	Shift arithmetic left	Schiebe arithmetisch nach links
LOOPNE	Loop CX times while not equal	Führe Schleife CX-mal aus solange nicht gleich	SAR	Shift arithmetic right	Schiebe arithmetisch nach rechts
LOOPNZ	Loop CX times while not zero	Führe Schleife CX-mal aus solange nicht 0	SBB	Subtract with borrow	Subtrahiere mit Borgen
LOOPZ	Loop CX times while zero	Führe Schleife CX-mal aus solange 0	SCAS	Scan string	Taste Zeichenkette ab
MOV	Move	Bewege	SCASB	Scan string byte-wise	Taste Zeichenkette byte-wise ab
MOVS	Move string	Bewege Zeichenkette	SCASW	Scan string word-wise	Taste Zeichenkette wort-wise ab
MOVESB	Move string byte-wise	Bewege Zeichenkette byte-wise	SEG	Segment override	Überschreibe Segment
MOVESW	Move string word-wise	Bewege Zeichenkette wort-wise	SHL	Shift logical left	Schiebe logisch nach links
MUL	Multiply (unsigned)	Multipliziere (vorzeichenlos)	SHR	Shift logical right	Schiebe logisch nach rechts
NEG	Negate	Negiere	STC	Set carry flag	Setze Übertrag-Statusbit
NOP	No operation	Keine Operation	STD	Set direction flag	Setze Richtung-Statusbit
NOT	Invert	Invertiere	STI	Set interrupt flag	Setze Unterbrechung-Statusbit
OR	Or	Oder	STOS	Store string from AL/AX	Speichere Zeichenkette aus AL/AX
OUT	Output to	Gebe aus nach	STOSB	Store string byte-wise from AL	Speichere Zeichenkette byte-wise aus AL
POP	Pop	Hole	STOSW	Store string word-wise from AX	Speichere Zeichenkette wort-wise aus AX
POPF	Pop flags	Hole Statusbits	SUB	Subtract	Subtrahiere
PUSH	Push	Schiebe	TEST	Test	Teste
PUSHF	Push flags	Schiebe Statusbits	WAIT	Wait for test	Warte auf Test-Signal
RCL	Rotate through carry flag left	Rotiere durch Übertrag-Statusbit nach links	XCHG	Exchange	Tausche aus
RCR	Rotate through carry flag right	Rotiere durch Übertrag-Statusbit nach rechts	XLAT	Translate byte to AL	Setze Byte nach AL um
			XOR	Exclusive or	Exklusiv-Oder

Tabelle 1. Die Befehle des 8086/8088 mit englischer Bezeichnung und deutscher Übersetzung

Die Befehle des 8086/8088 und ihre Funktion

Bezeichnung	Funktion	Bezeichnung	Funktion
AAA	Passe Ergebnis nach ASCII-Addition an	INT	Unterbreche
AAD	Passe Ergebnis nach ASCII-Division an	INTO	Unterbreche bei Überlauf
AAM	Passe Ergebnis nach ASCII-Multiplikation an	IRET	Kehre von Unterbrechungsroutine zurück
AAS	Passe Ergebnis nach ASCII-Subtraktion an	JA	Verzweige, wenn darüber (entspricht JNBE)
ADC	Addiere mit Übertrag	JAE	Verzweige, wenn darüber oder gleich (entspricht JNB)
ADD	Addiere ohne Übertrag	JB	Verzweige, wenn darunter (entspricht JNAE)
AND	Führe logische AND-Verknüpfung aus	JBE	Verzweige, wenn darunter oder gleich (entspricht JNA)
CALL	Rufe Unterprogramm auf	JCXZ	Verzweige, wenn Inhalt von CX gleich 0
CBW	Wandle Byte in Wort um	JE	Verzweige, wenn gleich (entspricht JZ)
CLC	Lösche Übertrag-Statusbit	JG	Verzweige, wenn größer (entspricht JNLE)
CLD	Lösche Richtung-Statusbit	JGE	Verzweige, wenn größer oder gleich (entspricht JNL)
CLI	Lösche Unterbrechung-Statusbit	JL	Verzweige, wenn kleiner (entspricht JNGE)
CMC	Invertiere Übertrag-Statusbit	JLE	Verzweige, wenn kleiner oder gleich (entspricht JNG)
CMP	Führe Vergleich aus	JMP	Verzweige
CMPS	Vergleiche Zeichenketten	JNA	Verzweige, wenn nicht darüber (entspricht JBE)
CMPSB	Vergleiche Zeichenketten byte-wise	JNAE	Verzweige, wenn nicht darüber und nicht gleich (entspricht JB)
CMPSW	Vergleiche Zeichenketten wort-wise	JNB	Verzweige, wenn nicht darunter (entspricht JAE)
CWD	Wandle Wort in Doppelwort um	JNBE	Verzweige, wenn nicht darunter und nicht gleich (entspricht JA)
DAA	Passe Ergebnis nach dezimaler Addition an	JNE	Verzweige, wenn nicht gleich (entspricht JNZ)
DAS	Passe Ergebnis nach dezimaler Subtraktion an	JNG	Verzweige, wenn nicht größer (entspricht JLE)
DEC	Um 1 vermindern	JNGE	Verzweige, wenn nicht größer und nicht gleich (entspricht JL)
DIV	Führe vorzeichenlose Division aus	JNL	Verzweige, wenn nicht kleiner (entspricht JGE)
ESC	Übergib an externe Einheit	JNLE	Verzweige, wenn nicht kleiner und nicht gleich (entspricht JG)
HLT	Halte Prozessor an	JNO	Verzweige, wenn kein Überlauf
IDIV	Führe vorzeichenbehaftete Division aus	JNP	Verzweige, wenn keine Parität (entspricht JPO)
IMUL	Führe vorzeichenbehaftete Multiplikation aus	JNS	Verzweige, wenn positiv
IN	Lies über Port ein	JNZ	Verzweige, wenn nicht 0 (entspricht JNE)
INC	Erhöhe um 1	JO	Verzweige, wenn Überlauf

Fortsetzung Tabelle 2 auf Seite 82

Bezeichnung	Funktion	Bezeichnung	Funktion
JP	Verzweige, wenn Parität (entspricht JPE)	REP	Wiederhole Zeichenketten-Operation bis Zähler CX gleich 0
JPE	Verzweige, wenn Parität gerade (entspricht JP)	REPE	Wiederhole Zeichenketten-Operation bis ungleich oder Zähler CX gleich 0
JPO	Verzweige, wenn Parität ungerade (entspricht JNP)	REPNE	Wiederhole Zeichenketten-Operation bis gleich oder Zähler CX gleich 0
JS	Verzweige, wenn negativ	REPZ	Wiederhole Zeichenketten-Operation bis ungleich 0 oder Zähler CX gleich 0
JZ	Verzweige, wenn 0 (entspricht JE)	RET	Von Unterprogramm zurückkehren
LAHF	Lade untere 8 Statusbit in AH-Register	ROL	Rotiere nach links
LDS	Lade DS-Register und weiteres Register	ROR	Rotiere nach rechts
LEA	Berechne und lädt effektive Adresse	SAHF	Lade AH-Register in untere 8 Statusbits
LES	Lade ES-Register und weiteres Register	SAL	Schiebe arithmetisch nach links (entspricht SHL)
LOCK	Aktiviere Lock-Signal für nachfolgenden Befehl	SAR	Schiebe arithmetisch nach rechts
LODS	Lade Zeichenkette	SBB	Subtrahiere mit Übertrag
LODSB	Lade Zeichenkette byteweise	SCAS	Durchsuche Zeichenkette
LODSW	Lade Zeichenkette wortweise	SCASB	Durchsuche Zeichenkette byteweise
LOOP	Wiederhole Schleife bis Zähler CX gleich 0	SCASW	Durchsuche Zeichenkette wortweise
LOOPE	Wiederhole Schleife bis ungleich oder Zähler CX gleich 0	SEG	Definiere gültiges Segmentregister für nachfolgenden Befehl
LOOPNE	Wiederhole Schleife bis gleich oder Zähler CX gleich 0	SHL	Schiebe logisch nach links (entspricht SAL)
LOOPNZ	Wiederhole Schleife bis gleich 0 oder Zähler CX gleich 0	SHR	Schiebe logisch nach rechts
LOOPZ	Wiederhole Schleife bis ungleich 0 oder Zähler CX gleich 0	STC	Setze Übertrag-Statusbit
MOV	Übertrage Zeichen	STD	Setze Richtung-Statusbit
MOVB	Übertrage Zeichenkette	STI	Setze Unterbrechung-Statusbit
MOVESB	Übertrage Zeichenkette byteweise	STOS	Speichere Zeichenkette
MOVESW	Übertrage Zeichenkette wortweise	STOSB	Speichere Zeichenkette byteweise
MUL	Führe vorzeichenlose Multiplikation aus	STOSW	Speichere Zeichenkette wortweise
NEG	Negiere (2er-Komplement bilden)	SUB	Subtrahiere ohne Übertrag
NOP	Keine Operation ausführen	TEST	Vergleiche über AND-Verknüpfung
NOT	Invertiere (1er-Komplement bilden)	WAIT	Warte auf Test-Signal
OR	Führe logische OR-Verknüpfung aus	XCHG	Tausche Registerwerte (und Speicherinhalt) aus
OUT	Gib über Port aus	XLAT	Adressiere Tabelle und lädt Zeichen
POP	Lies Wort vom Stack	XOR	Führe logische XOR-Verknüpfung aus
POPF	Lies Statusbits vom Stack		
PUSH	Lege Zeichen auf Stack		
PUSHF	Lege Statusbits auf Stack		
RCL	Durch Übertrag-Statusbit nach links rotieren		
RCR	Durch Übertrag-Statusbit nach rechts rotieren		

Tabelle 2. Die Funktion der Befehle des 8086/8088 gibt Aufschluß über die Leistungsfähigkeit des Prozessors

Die Befehle des 8086/88-Prozessors in alphabetischer Reihenfolge

Assemblercode	Maschinencode	Anweisung	Assemblercode	Maschinencode	Anweisung
AAA	37	Passe AX nach ASCII-Addition an	CBW	98	Wandle Byte in AL in ein Wort für AX um
AAD	D5, 0A	Passe AX nach ASCII-Division an	CLC	F8	Lösche Übertrag-Statusbit
AAM	D4, 0A	Passe AX nach ASCII-Multiplikation an	CLD	FC	Lösche Richtung-Statusbit
AAS	3F	Passe AX nach ASCII-Subtraktion an	CLI	FA	Lösche Unterbrechung-Statusbit
ADC AL/AX, data	0001010w, data	Addiere data und Übertrag-Statusbit zu AL/AX	CMC	F5	Invertiere Übertrag-Statusbit
ADC r/m, data	100000sw, mod 010 r/m	Addiere data und Übertrag-Statusbit zu r/m	CMP AL/AX, data	0011110w, data	Vergleiche data mit AL/AX
ADC r/m1, r/m2	000100dw, mod reg r/m	Addiere r/m2 und Übertrag-Statusbit zu r/m1	CMP r/m, data	100000sw, mod 111 r/m	Vergleiche data mit r/m
	[addr/displ]			[addr/displ], data	
ADD AL/AX, data	0000010w, data	Addiere data zu AL/AX	CMP r/m1, r/m2	001110dw, mod reg r/m	Vergleiche r/m2 mit r/m1
ADD r/m, data	100000sw, mod 000 r/m	Addiere data zu r/m		[addr/displ]	
ADD r/m1, r/m2	000000dw, mod reg r/m	Addiere r/m2 zu r/m1	CMPS	1010011w	Vergleiche Zeichenkette ab SI mit Zeichenkette ab DI
	[addr/displ]		CMPSB	A6	Vergleiche Zeichenkette ab SI byteweise mit Zeichenkette ab DI
AND AL/AX, data	0010010w, data	Verknüpfe data mit AL/AX über die logische AND-Funktionen	CMPSW	A7	Vergleiche Zeichenkette ab SI wortweise mit Zeichenkette ab DI
AND r/m, data	1000000w, mod 100 r/m	Verknüpfe data mit r/m über die logische AND-Funktion	CWD	99	Wandle Wort in AX in ein Doppelwort für DX (MSB) und AX (LSB) um
AND r/m1, r/m2	001000dw, mod reg r/m	Verknüpfe r/m2 mit r/m1 über die logische AND-Funktion	DAA	27	Passe AL nach dezimaler Addition an
	[addr/displ]		DAS	2F	Passe AL nach dezimaler Subtraktion an
CALL address	9A, addr1, addr2	Rufe Unterprogramm ab addr1 (CS=addr2) auf	DEC reg16	01001reg16	Vermindere Register reg16 um 1
CALL disp16	E8, disp16	Rufe Unterprogramm ab disp16+naddr auf	DEC r/m	1111111w, mod 001 r/m	Vermindere r/m um 1
CALL mem	FF, mod 011 r/m	Rufe Unterprogramm ab mem (CS=mem+2) auf		[addr/displ]	
CALL r/m16	FF, mod 010 r/m	Rufe Unterprogramm ab r/m auf	DIV r/m	1111011w, mod 110 r/m	Dividiere DX/AX durch r/m (vorzeichenbehaftet)
	[addr/displ]			[addr/displ]	

Fortsetzung Tabelle 3 auf Seite 83

Assemblercode	Maschinencode	Anweisung
ESC	11011xxx, mod xxx r/m [addr/disp]	Übergebe r/m an externe Einheit
HLT	F4	Halte Prozessor an
IDIV r/m	1111011w, mod 111 r/m [addr/disp]	Dividiere DX/AX durch r/m (vorzeichenlos)
IMUL r/m	1111011w, mod 101 r/m [addr/disp]	Multipliziere AL/AX mit r/m (vorzeichenlos)
IN AL/AX, (DX)	1110110w	Lese Wert über Port (DX) in AL/AX
IN AL/AX, port	1110010w, port	Lies Wert über Port port in AL/AX
INC reg16	01000reg16	Erhöhe Register reg16 um 1
INC r/m	1111111w, mod 000 r/m [addr/disp]	Erhöhe r/m um 1
INT data8	CD, data8	Führe Software-Unterbrechung data8 aus
INT 3	CC	Führe Software-Unterbrechung 3 aus
INTO	CE	Führe Software-Unterbrechung 4 aus, wenn Überlauf (IP=Inhalt von 00010, CS=Inhalt von 00012)
IRET	CF	Kehre von Interruptroutine zurück, lies CS vom Stapel
JA disp8	77, disp8	Verzweige zu disp8+naddr, wenn darüber (C=0 und Z=0)
JAE disp8	73, disp8	Verzweige zu disp8+naddr, wenn darüber oder gleich (C=0)
JB disp8	72, disp8	Verzweige zu disp8+naddr, wenn darunter (C=1)
JBE disp8	76, disp8	Verzweige zu disp8+naddr, wenn darunter oder gleich (C=1 oder Z=1)
JCXZ disp8	E3, disp8	Verzweige zu disp8+naddr, wenn Inhalt von CX gleich 0
JE disp8	74, disp8	Verzweige zu disp8+naddr, wenn gleich (Z=1)
JG disp8	7F, disp8	Verzweige zu disp8+naddr, wenn größer (S=0 und Z=0)
JGE disp8	7D, disp8	Verzweige zu disp8+naddr, wenn größer oder gleich (O=S)
JL disp8	7C, disp8	Verzweige zu disp8+naddr, wenn kleiner (O < S)
JLE disp8	7E, disp8	Verzweige zu disp8+naddr, wenn kleiner oder gleich (Z=1 oder O < S)
JMP address	EA, addr1, addr2	Verzweige zu addr1 (CS=addr2)
JMP disp8	EB, disp8	Verzweige zu disp8+naddr
JMP disp16	E9, disp16	Verzweige zu disp16+naddr
JMP mem	FF, mod 101 r/m [addr/disp]	Verzweige zu mem (CS=mem+2)
JMP r/m16	FF, mod 100 r/m [addr/disp]	Verzweige zu r/m16+naddr
JNA disp8	76, disp8	Verzweige zu disp8+naddr, wenn nicht darüber (C=1 oder Z=1)
JNAE disp8	72, disp8	Verzweige zu disp8+naddr, wenn nicht darüber und nicht gleich (C=1)
JNB disp8	73, disp8	Verzweige zu disp8+naddr, wenn nicht darunter (C=0)
JNBE disp8	77, disp8	Verzweige zu disp8+naddr, wenn nicht darunter und nicht gleich (C=0 und Z=0)
JNE disp8	75, disp8	Verzweige zu disp8+naddr, wenn nicht gleich (Z=0)
JNG disp8	7E, disp8	Verzweige zu disp8+naddr, wenn nicht größer (Z=1 oder O < S)
JNGE disp8	7C, disp8	Verzweige zu disp8+naddr, wenn nicht größer und nicht gleich (O < S)
JNL disp8	7D, disp8	Verzweige zu disp8+naddr, wenn nicht kleiner (O=S)
JNLE disp8	7F, disp8	Verzweige zu disp8+naddr, wenn nicht kleiner und nicht gleich (S=0 und Z=0)
JNO disp8	71, disp8	Verzweige zu disp8+naddr, wenn kein Überlauf (O=0)
JNP disp8	7B, disp8	Verzweige zu disp8+naddr, wenn keine Parität (P=0)
JNS disp8	79, disp8	Verzweige zu disp8+naddr, wenn positiv (S=0)
JNZ disp8	75, disp8	Verzweige zu disp8+naddr, wenn nicht Null (Z=0)
JO disp8	70, disp8	Verzweige zu disp8+naddr, wenn Überlauf (O=1)

Assemblercode	Maschinencode	Anweisung
JP disp8	7A, disp8	Verzweige zu disp8+naddr, wenn Parität (P=1)
JPE disp8	7A, disp8	Verzweige zu disp8+naddr, wenn Parität gerade (P=1)
JPO disp8	7B, disp8	Verzweige zu disp8+naddr, wenn Parität ungerade (P=0)
JS disp8	78, disp8	Verzweige zu disp8+naddr, wenn negativ (S=1)
JZ disp8	74, disp8	Verzweige zu disp8+naddr, wenn 0 (Z=1)
LAHF	9F	Lade die unteren 8 Statusbit in AH
LDS reg16, mem	C5, mod reg16 r/m [addr/disp]	Lade reg16 mit mem und DS mit mem+2
LEA reg16, mem	8D, mod reg16 r/m [addr/disp]	Lade effektive Adresse von mem in reg16
LES reg16, mem	C4, mod reg16 r/m [addr/disp]	Lade reg16 mit mem und ES mit mem+2
LOCK	F0	Aktiviere den LOCK-Ausgang des Prozessors für den nachfolgenden Befehl
LODS	1010110w	Lade Zeichenkette ab SI in AL/AX
LODSB	AC	Lade Zeichenkette ab SI byteweise in AL
LODSW	AD	Lade Zeichenkette ab SI wortweise in AX
LOOP disp8	E2, disp8	Wiederhole Schleife bis CX=0 (CX wird jeweils um 1 dekrementiert)
LOOPE disp8	E1, disp8	Wiederhole Schleife bis CX=0 oder Z=0 (CX wird jeweils um 1 dekrementiert)
LOOPNE disp8	E0, disp8	Wiederhole Schleife bis CX=0 oder Z=1 (CX wird jeweils um 1 dekrementiert)
LOOPNZ disp8	E0, disp8	Wiederhole Schleife bis CX=0 oder Z=1 (CX wird jeweils um 1 dekrementiert)
LOOPZ disp8	E1, disp8	Wiederhole Schleife bis CX=0 oder Z=0 (CX wird jeweils um 1 dekrementiert)
MOV reg, data	1011wreg, data	Übertrage data in reg
MOV r/m, data	1100011w, mod 000 r/m [addr/disp], data	Übertrage data in r/m
MOV AL/AX, addr	1010000w, mem	Übertrage mem in AL/AX
MOV addr, AL/AX	1010001w, mem	Übertrage AL/AX in mem
MOV r/m1, r/m2	100010dw, mod reg r/m [addr/disp]	Übertrage r/m2 in r/m1
MOV sr, r/m16	8E, mod 0sr r/m [addr/disp]	Übertrage r/m16 in sr
MOV r/m16, sr	8C, mod 0sr r/m [addr/disp]	Übertrage sr in r/m16
MOVS	1010010w	Übertrage Zeichenkette ab SI nach DI
MOVESB	A4	Übertrage Zeichenkette byteweise ab SI nach DI
MOVESW	A5	Übertrage Zeichenkette wortweise ab SI nach DI
MUL r/m	1111011w, mod 100 r/m [addr/disp]	Multipliziere AL/AX mit r/m (vorzeichenbehaftet)
NEG r/m	1111011w, mod 011 r/m [addr/disp]	Negiere Wert von r/m (bilde 2er-Komplement)
NOP	90	Führe keine Operation aus
NOT r/m	1111011w, mod 010 r/m [addr/disp]	Invertiere Wert von r/m (bilde 1er-Komplement)
OR AL/AX, data	0000110w, data	Verknüpfe data mit AL/AX über die logische OR-Funktion
OR r/m, data	1000000w, mod 001 r/m [addr/disp], data	Verknüpfe data mit r/m über die logische OR-Funktion
OR r/m1, r/m2	000010dw, mod reg r/m [addr/disp]	Verknüpfe r/m2 mit r/m1 über die logische OR-Funktion
OUT (DX), AL/AX	1110111w	Gebe AL/AX über Port (DX) aus
OUT port, AL/AX	1110011w, port	Gebe AL/AX über Port port aus
POP reg16	01011reg16	Lese in Register reg16 ein Wort vom Stapel
POP sr	000sr111	Lese in Segmentregister sr ein Wort vom Stapel
POP r/m16	8F, mod 000 r/m [addr/disp]	Lese in r/m16 ein Wort vom Stapel
POPF	9D	Lese Statusbits vom Stapel

Fortsetzung Tabelle 3 auf Seite 84

Assemblercode	Maschinencode	Anweisung	Assemblercode	Maschinencode	Anweisung
PUSH reg16	01010reg16	Lege Register reg16 auf den Stapel	SBB r/m1, r/m2	000110dw, mod reg r/m [addr/disp]	Subtrahiere r/m2 und das Übertrag-Statusbit von r/m1
PUSH sr	000sr110	Lege Segmentregister sr auf den Stapel	SCAS	101011w	Durchsuche Zeichenkette ab DI nach AL/AX
PUSH r/m16	FF, mod 110 r/m [addr/disp]	Lege r/m16 auf den Stapel	SCASB	AE	Durchsuche Zeichenkette ab DI byteweise nach AL
PUSHF	9C	Lege Statusbits auf den Stapel	SCASW	AF	Durchsuche Zeichenkette ab DI wortweise nach AX
RCL r/m [CL]	110100cw, mod 010 r/m [addr/disp]	Rotiere r/m um 1 [CL] nach links und durch das Übertrag-Statusbit	SEG	001sr110	Definiere sr als gültiges Segmentregister für den nächsten Befehl
RCR r/m [CL]	110100cw, mod 011 r/m [addr/disp]	Rotiere r/m um 1 [CL] nach rechts und durch das Übertrag-Statusbit	SHL r/m [CL]	110100cw, mod 100 r/m [addr/disp]	Schiebe r/m um 1 [CL] logisch nach links (entspricht SAL)
REP	F2 oder F3	Wiederhole Zeichenketten-Operation bis CX=0 (CX wird jeweils um 1 dekrementiert)	SHR r/m [CL]	110100cw, mod 101 r/m [addr/disp]	Schiebe r/m um 1 [CL] logisch nach rechts
REPE	F3 (nur vor CMPS/SCAS)	Wiederhole Zeichenketten-Operation bis CX=0 oder Z=0 (CX wird jeweils um 1 dekrementiert)	STC	F9	Setze Übertrag-Statusbit
REPNE	F2 (nur vor CMPS/SCAS)	Wiederhole Zeichenketten-Operation bis CX=0 oder Z=1 (CX wird jeweils um 1 dekrementiert)	STD	FD	Setze Richtung-Statusbit
REPZ	F2 (nur vor CMPS/SCAS)	Wiederhole Zeichenketten-Operation bis CX=0 oder Z=1 (CX wird jeweils um 1 dekrementiert)	STI	FB	Setze Unterbrechung-Statusbit
REPZ	F3 (nur vor CMPS/SCAS)	Wiederhole Zeichenketten-Operation bis CX=0 oder Z=0 (CX wird jeweils um 1 dekrementiert)	STOS	1010101w	Speichere AL/AX in Zeichenkette ab DI
RET	CB	Kehre vom Unterprogramm zurück, lese CS vom Stapel	STOSB	AA	Speichere AL byteweise in Zeichenkette ab DI
RET	C3	Kehre vom Unterprogramm zurück, lasse CS unverändert	STOSW	AB	Speichere AX wortweise in Zeichenkette ab DI
RET disp16	CA, disp16	Kehre vom Unterprogramm zurück, lese CS vom Stapel, addiere disp16 zu SP	SUB AL/AX, data	0010110w, data	Subtrahiere data von AL/AX
RET disp16	C2, disp16	Kehre vom Unterprogramm zurück, lasse CS unverändert, addiere disp16 zu SP	SUB r/m, data	100000sw, mod 101 r/m [addr/disp], data	Subtrahiere data von r/m
ROL r/m [CL]	110100cw, mod 000 r/m [addr/disp]	Rotiere r/m um 1 [CL] nach links	SUB r/m1, r/m2	001010dw, mod reg r/m [addr/disp]	Subtrahiere r/m2 von r/m1
ROR r/m [CL]	110100cw, mod 001 r/m [addr/disp]	Rotiere r/m um 1 [CL] nach rechts	TEST AL/AX, data	1010100w, data	Vergleiche data mit AL/AX über die And-Funktion
SAHF	9E	Lade AH in die unteren 8 Statusbits	TEST r/m, data	1111011w, mod 000 r/m [addr/disp], data	Vergleiche data mit r/m über die And-Funktion
SAL r/m [CL]	110100cw, mod 100 r/m [addr/disp]	Schiebe r/m um 1 [CL] arithmetisch nach links (entspricht SHL)	TEST r/m1, r/m2	1000010w, mod reg r/m [addr/disp]	Vergleiche r/m2 mit r/m1 über die And-Funktion
SAR r/m [CL]	110100cw, mod 111 r/m [addr/disp]	Schiebe r/m um 1 [CL] arithmetisch nach rechts	WAIT	9B	Warte auf aktives Signal am Test-Eingang des Prozessors
SBB AL/AX, data	0001110w, data	Subtrahiere data und das Übertrag-Statusbit von AL/AX	XCHG AX, reg16	10010 reg16	Tausche Register reg16 gegen AX aus
SBB r/m, data	100000sw, mod 011 r/m [addr/disp], data	Subtrahiere data und das Übertrag-Statusbit von AL/AX	XCHG r/m1, r/m2	1000011w, mod reg r/m [addr/disp]	Tausche r/m2 gegen r/m1 aus
			XLAT	D7	Übertrage den Inhalt der durch AL+BX+DS adressierten Speicherzelle in AL
			XOR AL/AX, data	0011010w, data	Verknüpfe data mit AL/AX über die logische XOR-Funktion
			XOR r/m, data	1000000w, mod 110 r/m [addr/disp], data	Verknüpfe data mit r/m über die logische XOR-Funktion
			XOR r/m1, r/m2	001100dw, mod reg r/m [addr/disp]	Verknüpfe r/m2 mit r/m1 über die logische XOR-Funktion

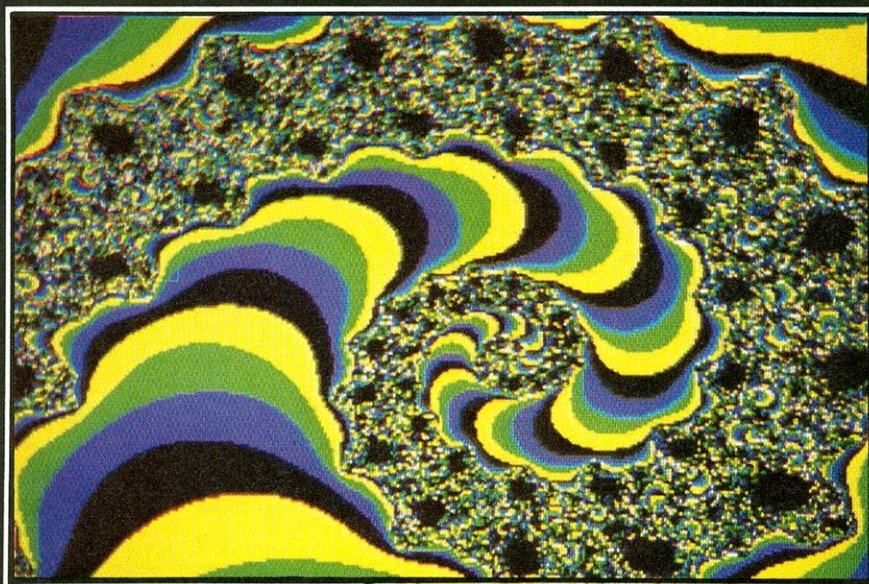
Tabelle 3. Die Assemblerbefehle des 8086/8088 in ihren Variationen mit Maschinencode und exakter Bedeutung

Die Befehle des 8086/8088-Prozessors in numerischer Reihenfolge									
Maschinencode	Befehl	1. Parameter	2. Parameter	Bemerkung	Maschinencode	Befehl	1. Parameter	2. Parameter	Bemerkung
00, mod reg r/m	ADD	r/m8,	reg8		10, mod reg r/m	ADC	r/m8,	reg8	
01, mod reg r/m	ADD	r/m16,	reg16		11, mod reg r/m	ADC	r/m16,	reg16	
02, mod reg r/m	ADD	reg8,	r/m8		12, mod reg r/m	ADC	reg8,	r/m8	
03, mod reg r/m	ADD	reg16,	r/m16		13, mod reg r/m	ADC	reg16,	r/m16	
04, data8	ADD	AL,	data8		14, data8	ADC	AL,	data8	
05, data16	ADD	AX,	data16		15, data16	ADC	AX,	data16	
06	PUSH	ES			16	PUSH	SS		
07	POP	ES			17	POP	SS		
08, mod reg r/m	OR	r/m8,	reg8		18, mod reg r/m	SBB	r/m8,	reg8	
09, mod reg r/m	OR	r/m16,	reg16		19, mod reg r/m	SBB	r/m16,	reg16	
0A, mod reg r/m	OR	reg8,	r/m8		1A, mod reg r/m	SBB	reg8,	r/m8	
0B, mod reg r/m	OR	reg16,	r/m16		1B, mod reg r/m	SBB	reg16,	r/m16	
0C, data8	OR	AL,	data8		1C, data8	SBB	AL,	data8	
0D, data16	OR	AX,	data16		1D, data16	SBB	AX,	data16	
0E	PUSH	CS			1E	PUSH	DS		
0F	POP	CS			1F	POP	DS		

Fortsetzung Tabelle 4 auf Seite 86

TEXTVERARBEITUNG, TABELLENKALKULATION LOHNBUCHHALTUNG-STATISTIK.....

Überarbeitet? Gönnen Sie sich eine Urlaubsreise in das Land der Fraktale, genießen Sie die phantastische Farb-Fähigkeit Ihres PC.



FRAKTAL GENERATOR

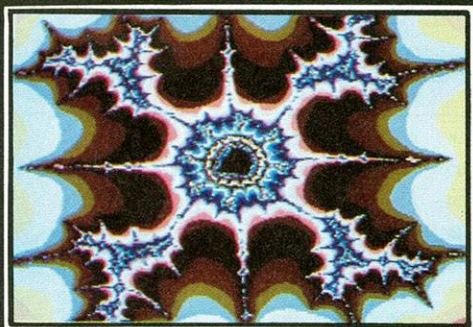
Autor: Matthias Uphoff

– eine neue Dimension für die Freunde fraktaler Grafiken: Über 100 (!) Farben stellt Ihnen dieses Programm bei einer Auflösung von 640×200 Punkten zur Verfügung. Und das berühmte »Apfelmännchen« braucht dank ausgefeilter Algorithmen nur noch ca. 3 Minuten für seine Entstehung – keine stundenlangen Wartezeiten hindern Sie mehr daran, eine märchenhafte Welt voller Farben und Formen zu entdecken.

Und dazu der Bedienungskomfort:

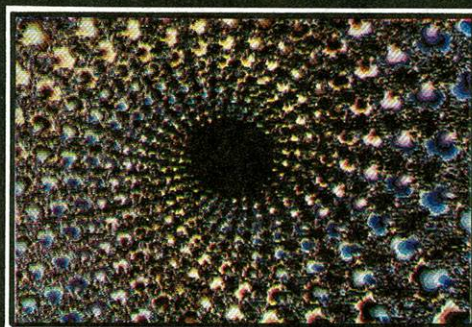
- Bedienung per Maus und Pulldown-Menues
- Hardcopy auf Knopfdruck
- Speichern von Bildern auf Diskette

- Umschalten zwischen verschiedenen Bildern
- Nachträgliches Ändern der Farben
- Vergrößerte Ausschnitte druck
- Auswahlrahmen
- Rechentiefe bis 9999



FRAKTAL GENERATOR

- Schneider PC 1512 /PC 1640
- Atari PC
- PC's mit EGA-Karte & Microsoftkompatibler Maus.



FRAKTAL GENERATOR

- ein CGX-Programm von DMV

nur

49,–

— Bitte Coupon auf Seite 99 benutzen —

DMV Daten & Medien Verlagsgesellschaft mbH · Postfach 250 · Fuldaer Str. 6 · 3440 Eschwege

Händleranfragen erwünscht

Maschinencode	Befehl	1. Parameter	2. Parameter	Bemerkung	Maschinencode	Befehl	1. Parameter	2. Parameter	Bemerkung
20, mod reg r/m	AND	r/m8,	reg8		58	POP	AX		
21, mod reg r/m	AND	r/m16,	reg16		59	POP	CX		
22, mod reg r/m	AND	reg8,	r/m8		5A	POP	DX		
23, mod reg r/m	AND	reg16,	r/m16		5B	POP	BX		
24, data 8	AND	AL,	data8		5C	POP	SP		
25, data16	AND	AX,	data16		5D	POP	BP		
26	SEG	ES			5E	POP	SI		
27	DAA				5F	POP	DI		
28, mod reg r/m	SUB	r/m8,	reg8		60	—			
29, mod reg r/m	SUB	r/m16,	reg16		61	—			
2A, mod reg r/m	SUB	reg8,	r/m8		62	—			
2B, mod reg r/m	SUB	reg16,	r/m16		63	—			
2C, data8	SUB	AL,	data8		64	—			
2D, data16	SUB	AX,	data16		65	—			
2E	SEG	CS			66	—			
2F	DAS				67	—			
					68	—			
30, mod reg r/m	XOR	r/m8,	reg8		69	—			
31, mod reg r/m	XOR	r/m16,	reg16		6A	—			
32, mod reg r/m	XOR	reg8,	r/m8		6B	—			
33, mod reg r/m	XOR	reg16,	r/m16		6C	—			
34, data8	XOR	AL,	data8		6D	—			
35, data16	XOR	AX,	data16		6E	—			
36	SEG	SS			6F	—			
37	AAA								
					70, disp8	JO	disp8		
38, mod reg r/m	CMP	r/m8,	reg8		71, disp8	JNO	disp8		
39, mod reg r/m	CMP	r/m16,	reg16		72, disp8	JB/JNAE	disp8		
3A, mod reg r/m	CMP	reg8,	r/m8		73, disp8	JAE/JNB	disp8		
3B, mod reg r/m	CMP	reg16,	r/m16		74, disp8	JE/JZ	disp8		
3C, data8	CMP	AL,	data8		75, disp8	JNE/JNZ	disp8		
3D, data16	CMP	AX,	data16		76, disp8	JBE/JNA	disp8		
3E	SEG	DS			77, disp8	JA/JNBE	disp8		
3F	AAS								
					78, disp8	JS	disp8		
40	INC	AX			79, disp8	JNS	disp8		
41	INC	CX			7A, disp8	JP/JPE	disp8		
42	INC	DX			7B, disp8	JPO/JNP	disp8		
43	INC	BX			7C, disp8	JL/JNGE	disp8		
44	INC	SP			7D, disp8	JGE/JNL	disp8		
45	INC	BP			7E, disp8	JLE/JNG	disp8		
46	INC	SI			7F, disp8	JG/JNLE	disp8		
47	INC	DI							
					80, mod 000 r/m	ADD	r/m8,	data8	
48	DEC	AX			80, mod 001 r/m	OR	r/m8,	data8	
49	DEC	CX			80, mod 010 r/m	ADC	r/m8,	data8	
4A	DEC	DX			80, mod 011 r/m	SBB	r/m8,	data8	
4B	DEC	BX			80, mod 100 r/m	AND	r/m8,	data8	
4C	DEC	SP			80, mod 101 r/m	SUB	r/m8,	data8	
4D	DEC	BP			80, mod 110 r/m	XOR	r/m8,	data8	
4E	DEC	SI			80, mod 111 r/m	CMP	r/m8,	data8	
4F	DEC	DI							
					81, mod 000 r/m	ADD	r/m16,	data16	
50	PUSH	AX			81, mod 001 r/m	OR	r/m16,	data16	
51	PUSH	CX			81, mod 010 r/m	ADC	r/m16,	data16	
52	PUSH	DX			81, mod 011 r/m	SBB	r/m16,	data16	
53	PUSH	BX			81, mod 100 r/m	AND	r/m16,	data16	
54	PUSH	SP			81, mod 101 r/m	SUB	r/m16,	data16	
55	PUSH	BP			81, mod 110 r/m	XOR	r/m16,	data16	
56	PUSH	SI			81, mod 111 r/m	CMP	r/m16,	data16	
57	PUSH	DI							

Fortsetzung Tabelle 4 Seite 87

Maschinencode	Befehl	1. Parameter	2. Parameter	Bemerkung	Maschinencode	Befehl	1. Parameter	2. Parameter	Bemerkung
82, mod 000 r/m	ADD	r/m8,	data8		A8, data8	TEST AL,	data8		
82, mod 001 r/m	OR	r/m8,	data8		A9, data16	TEST AX,	data16		
82, mod 010 r/m	ADC	r/m8,	data8		AA	STOSB			
82, mod 011 r/m	SBB	r/m8,	data8		AB	STOSW			
82, mod 100 r/m	AND	r/m8,	data8		AC	LODSB			
82, mod 101 r/m	SUB	r/m8,	data8		AD	LODSW			
82, mod 110 r/m	XOR	r/m8,	data8		AE	SCASB			
82, mod 111 r/m	CMP	r/m8,	data8		AF	SCASW			
83, mod 000 r/m	ADD	r/m16,	data8		B0, data8	MOV	AL,	data8	
83, mod 001 r/m	OR	r/m16,	data8		B1, data8	MOV	CL,	data8	
83, mod 010 r/m	ADC	r/m16,	data8		B2, data8	MOV	DL,	data8	
83, mod 011 r/m	SBB	r/m16,	data8		B3, data8	MOV	BL,	data8	
83, mod 100 r/m	AND	r/m16,	data8		B4, data8	MOV	AH,	data8	
83, mod 101 r/m	SUB	r/m16,	data8		B5, data8	MOV	CH,	data8	
83, mod 110 r/m	XOR	r/m16,	data8		B6, data8	MOV	DH,	data8	
83, mod 111 r/m	CMP	r/m16,	data8		B7, data8	MOV	BH,	data8	
84, mod reg r/m	TEST	r/m8,	r/m8		B8, data16	MOV	AX,	data16	
85, mod reg r/m	TEST	r/m16,	r/m16		B9, data16	MOV	CX,	data16	
86, mod reg r/m	XCHG	r/m8,	r/m8		BA, data16	MOV	DX,	data16	
87, mod reg r/m	XCHG	r/m16,	r/m16		BB, data16	MOV	BX,	data16	
88, mod reg r/m	MOV	r/m8,	reg8		BC, data16	MOV	SP,	data16	
89, mod reg r/m	MOV	r/m16,	reg16		BD, data16	MOV	BP,	data16	
8A, mod reg r/m	MOV	reg8,	r/m8		BE, data16	MOV	SI,	data16	
8B, mod reg r/m	MOV	reg16,	r/m16		BF, data16	MOV	DI,	data16	
8C, mod 0sr r/m	MOV	r/m16,	sr		C0	—			
8D, mod reg16 r/m	LEA	reg16,	mem		C1	—			
8E, mod 0sr r/m	MOV	sr,	r/m16		C2, disp16	RET	disp16		CS unverändert (Intrasegment)
8F, mod 000 r/m	POP	r/m16			C3	RET	CS unverändert (Intrasegment)		
90	NOP				C4, mod reg16 r/m	LES	reg16,	mem	
91	XCHG	AX,	CX		C5, mod reg16 r/m	LDS	reg16,	mem	
92	XCHG	AX,	DX		C6, mod 000 r/m	MOV	r/m8,	data8	
93	XCHG	AX,	BX		C7, mod 000 r/m	MOV	r/m16,	data16	
94	XCHG	AX,	SP		C8	—			
95	XCHG	AX,	BP		C9	—			
96	XCHG	AX,	SI		CA, disp16	RET	disp16		CS geändert (Intersegment)
97	XCHG	AX,	DI		CB	RET			CS geändert (Intersegment)
98	CBW				CC	INT	3		Abbruchpunkt (Breakpoint)
99	CWD				CD, data8	INT	data8		CS geändert (Intersegment)
9A, addr1, addr2	CALL	address	IP=addr1, CS=addr2		CE	INTO			CS geändert (Intersegment)
9B	WAIT				CF	IRET			CS geändert (Intersegment)
9C	PUSHF				D0, mod 000 r/m	ROL	r/m8		
9D	POPF				D0, mod 001 r/m	ROR	r/m8		
9E	SAHF				D0, mod 010 r/m	RCL	r/m8		
9F	LAHF				D0, mod 011 r/m	RCR	r/m8		
A0, addr	MOV	AL,	mem		D0, mod 100 r/m	SAL/SHL	r/m8		
A1, addr	MOV	AX,	mem		D0, mod 101 r/m	SHR	r/m8		
A2, addr	MOV	mem,	AL		D0, mod 110 r/m	—			
A3, addr	MOV	mem,	AX		D0, mod 111 r/m	SAR	r/m8		
A4	MOVS								
A5	MOVSW								
A6	CMPSB								
A7	CMPSW								

Fortsetzung Tabelle 4 Seite 88

Maschinencode	Befehl	1. Parameter	2. Parameter	Bemerkung	Maschinencode	Befehl	1. Parameter	2. Parameter	Bemerkung
D1, mod 000 r/m	ROL	r/m16			F0	LOCK			
D1, mod 001 r/m	ROR	r/m16			F1	—			
D1, mod 010 r/m	RCL	r/m16			F2	REP/REPNE/ REPNZ			
D1, mod 011 r/m	RCR	r/m16			F3	REPE/REPZ			
D1, mod 100 r/m	SAL/SHL	r/m16			F4	HLT			
D1, mod 101 r/m	SHR	r/m16			F5	CMC			
D1, mod 110 r/m	—								
D1, mod 111 r/m	SAR	r/m16							
D2, mod 000 r/m	ROL	r/m8,	CL		F6, mod 000 r/m	TEST	r/m8,	data8	
D2, mod 001 r/m	ROR	r/m8,	CL		F6, mod 001 r/m	—			
D2, mod 010 r/m	RCL	r/m8,	CL		F6, mod 010 r/m	NOT	r/m8		
D2, mod 011 r/m	RCR	r/m8,	CL		F6, mod 011 r/m	NEG	r/m8		
D2, mod 100 r/m	SAL/SHL	r/m8,	CL		F6, mod 100 r/m	MUL	r/m8		
D2, mod 101 r/m	SHR	r/m8,	CL		F6, mod 101 r/m	IMUL	r/m8		
D2, mod 110 r/m	—				F6, mod 110 r/m	DIV	r/m8		
D2, mod 111 r/m	SAR	r/m8,	CL		F6, mod 111 r/m	IDIV	r/m8		
D3, mod 000 r/m	ROL	r/m16,	CL		F7, mod 000 r/m	TEST	r/m16,	data16	
D3, mod 001 r/m	ROR	r/m16,	CL		F7, mod 001 r/m	—			
D3, mod 010 r/m	RCL	r/m16,	CL		F7, mod 010 r/m	NOT	r/m16		
D3, mod 011 r/m	RCR	r/m16,	CL		F7, mod 011 r/m	NEG	r/m16		
D3, mod 100 r/m	SAL/SHL	r/m16,	CL		F7, mod 100 r/m	MUL	r/m16		
D3, mod 101 r/m	SHR	r/m16,	CL		F7, mod 101 r/m	IMUL	r/m16		
D3, mod 110 r/m	—				F7, mod 110 r/m	DIV	r/m16		
D3, mod 111 r/m	SAR	r/m16,	CL		F7, mod 111 r/m	IDIV	r/m16		
D4, 0A	AAM								
D5, 0A	AAD								
D6	—								
D7	XLAT								
D8, mod xxx r/m	ESC				F8	CLC			
D9, mod xxx r/m	ESC				F9	STC			
DA, mod xxx r/m	ESC				FA	CLI			
DB, mod xxx r/m	ESC				FB	STI			
DC, mod xxx r/m	ESC				FC	CLD			
DD, mod xxx r/m	ESC				FD	STD			
DE, mod xxx r/m	ESC				FE, mod 000 r/m	INC	r/m8		
DF, mod xxx r/m	ESC				FE, mod 001 r/m	DEC	r/m8		
E0, disp8	LOOPNE/ LOOPNZ	disp8			FE, mod 010 r/m	—			
E1, disp8	LOOPE/ LOOPZ	disp8			FE, mod 011 r/m	—			
E2, disp8	LOOP	disp8			FE, mod 100 r/m	—			
E3, disp8	JCXZ	disp8			FE, mod 101 r/m	—			
E4, port	IN	AL,	port		FE, mod 110 r/m	—			
E5, port	IN	AX,	port		FE, mod 111 r/m	—			
E6, port	OUT	port,	AL						
E7, port	OUT	port,	AX		FF, mod 000 r/m	INC	r/m16		
E8, disp16	CALL disp16		CS unverändert (Intrasegment)		FF, mod 001 r/m	DEC	r/m16		
E9, disp16	JMP	disp16		CS unverändert (Intrasegment)	FF, mod 010 r/m	CALL	r/m16		CS unverändert (Intrasegment)
EA, addr1, addr2	JMP	address		IP=addr1, CS=addr2	FF, mod 011 r/m	CALL	mem		CS geändert (Intersegment)
EB, disp8	JMP	disp8		CS unverändert (Intrasegment)	FF, mod 100 r/m	JMP	r/m16		CS unverändert (Intrasegment)
EC	IN	AL,	(DX)		FF, mod 101 r/m	JMP	mem		CS geändert (Intersegment)
ED	IN	AX,	(DX)		FF, mod 110 r/m	PUSH	r/m16		
EE	OUT	(DX),	AL		FF, mod 111 r/m	—			
EF	OUT	(DX),	AX						

Tabelle 4. Alle Assemblerbefehle numerisch nach dem Maschinencode sortiert

Der Parameter »mod« bestimmt die Art des Befehlsaufbaus

mod	1. Byte	2. Byte	3. Byte	4. Byte
00	Befehlscode	mod reg mem	—	—
00	Befehlscode	mod reg 110	addr (LSB)	addr (MSB)
01	Befehlscode	mod reg mem	disp8	—
10	Befehlscode	mod reg mem	disp16 (LSB)	disp16 (MSB)
11	Befehlscode	mod reg reg	—	—

Tabelle 5. Der Parameter »mod« bestimmt in einem Assemblerbefehl, wie dieser Befehl aufgebaut ist

Die Kombination von »reg« und »w« definiert ein Register

reg	w=0	w=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Tabelle 6. Der Parameter »reg« wählt in Zusammenarbeit mit »w« eines von 16 Registern aus

Der Parameter »reg16« definiert ein 16-Bit-Register

reg16	Register
000	AX
001	CX
010	DX
011	BX
100	SP
101	BP
110	SI
111	DI

Tabelle 7. Der Parameter »reg16« definiert eines der acht 16-Bit-Register

Der Parameter »sr« definiert ein Segmentregister

sr	Segmentregister
00	ES
01	CS
10	SS
11	DS

Tabelle 8. Der Parameter »sr« bestimmt für einen Assemblerbefehl ein Segmentregister

Die Kombination von »r/m« und »mod« definiert eine Adressierungsart

r/m	mod=00	mod=01	mod=10	mod=11	
				w=0	w=1
000	BX+SI	BX+SI+disp8	BX+SI+disp16	AL	AX
001	BX+DI	BX+DI+disp8	BX+DI+disp16	CL	CX
010	BP+SI	BP+SI+disp8	BP+SI+disp16	DL	DX
011	BP+DI	BP+DI+disp8	BP+DI+disp16	BL	BX
100	SI	SI+disp8	SI+disp16	AH	SP
101	DI	DI+disp8	DI+disp16	CH	BP
110	addr	BP+disp8	BP+disp16	DH	SI
111	BX	BX+disp8	BX+disp16	BH	DI

Tabelle 9. Mit den beiden Parametern »r/m« und »mod« läßt sich die Adressierungsart für einen Assemblerbefehl festlegen

Spezielle Regeln für die Tabellen 3 und 4:

Der optionale Ausdruck [addr/displ] bezieht sich auf die durch mem angegebene Adresse einer Speicherzelle. Bei Befehlen in der Art »OPCODE r/m1, r/m2« muß mindestens einer der beiden Operanden in einem Register stehen. Die Angaben »darüber« und »darunter« in den bedingten Sprungbefehlen beziehen sich auf den Vergleich vorzeichenloser Werte. Die Angaben »größer« und »kleiner« in den bedingten Sprungbefehlen beziehen sich auf den Vergleich von Werten mit Vorzeichen.

Die Register und Statusbits des 8086/8088-Prozessors

Register/ Statusbit	Funktion
AH	Obere 8 Bit des Akkumulators
AL	Untere 8 Bit des Akkumulators
AX	Akkumulator (accumulator)
BP	Basiszeiger (base pointer)
BH	Obere 8 Bit des Basisregisters
BL	Untere 8 Bit des Basisregisters
BX	Basisregister (base register)
C	Übertrag-Statusbit (carry flag)
CH	Obere 8 Bit des Programmregisters
CL	Untere 8 Bit des Programmregisters
CS	Programmsegment (code segment)
CX	Zählerregister (count register)
DH	Obere 8 Bit des Datenregisters
DI	Zielregister (destination index)
DL	Untere 8 Bit des Datenregisters
DS	Datensegment (data segment)
DX	Datenregister (data register)
ES	Extrasegment (extra segment)
FLAGS	Statusbits (status flags)
IP	Befehlszeiger (instruction pointer)
O	Überlauf-Statusbit (overflow flag)
P	Parität-Statusbit (parity flag)
S	Vorzeichen-Statusbit (sign flag)
SI	Quellregister (source index)
SP	Stapelzeiger (stack pointer)
SS	Stapelsegment (stack segment)
Z	Null-Statusbit (zero flag)

Tabelle 10. Die Bedeutung der Register und Statusbits, die in Tabelle 3 und 4 verwendet werden

Die Parameter der Assemblerbefehle des 8086/8088

Parameter	Bedeutung
addr	16-Bit-Adresse einer Speicherzelle (niederwertiges Byte vor höherwertigem Byte)
address	20-Bit-Adresse einer Speicherzelle, die sich aus einer 16-Bit-Adresse und einer nachfolgenden 16-Bit-Segmentadresse (für CS) zusammensetzt
AL/AX	w=0: Register AL angesprochen w=1: Register AX angesprochen
c	c=0: Operand um eine Bitposition verschieben c=1: Operand um die in CL angegebene Anzahl von Positionen verschieben
d	d=0: Ergebnis steht im durch mod und r/m adressierten Operanden d=1: Ergebnis steht im durch den Code reg bezeichneten Register
data	8-Bit-Datenwert oder 16-Bit-Datenwert
data8	8-Bit-Datenwert
disp	8-Bit-Adresse oder 16-Bit-Adresse, die zu der dem aktuellen Befehl folgenden Adresse naddr addiert wird
disp8	8-Bit-Adresse, die zu der dem aktuellen Befehl folgenden Adresse naddr addiert wird
disp16	16-Bit-Adresse, die zu der dem aktuellen Befehl folgenden Adresse naddr addiert wird
mod	Code für eine Adressierungsart (siehe Tabelle)
mem	Inhalt der durch addr oder disp adressierten Speicherzelle(n)
naddr	Nächste gültige Adresse hinter dem aktuellen Befehl
port	8-Bit-Adresse eines Ports
reg	Code für ein Register (siehe Tabelle)
reg16	Code für ein 16-Bit-Register (siehe Tabelle)
r/m	Register oder Speicherzelleninhalt
r/m8	8-Bit-Register oder Inhalt einer Speicherzelle
r/m16	16-Bit-Register oder Inhalt zweier aufeinanderfolgenden Speicherzellen
sr	Code für ein Segmentregister (siehe Tabelle)
sw	sw=00: data ist ein 8-Bit-Operand sw=01: data ist ein 16-Bit-Operand sw=10: data ist ein 8-Bit-Operand sw=11: data ist ein 8-Bit-Operand, der auf 16 Bit erweitert werden muß
w	w=0: Bytebefehl w=1: Wortbefehl
x	Wahlweise gesetztes oder gelöscht Bit

Tabelle 11. Diese Parameter werden ebenfalls in Tabelle 3 und 4 verwendet

Der Arithmetikprozessor – das Rechengenie

Fast alle PCs verfügen über einen freien Sockel für den Arithmetikprozessor 8087, doch die wenigsten PC-Besitzer nennen auch den zugehörigen Prozessor ihr Eigen. Lesen Sie, wie der 8087 funktioniert, und ob sich die Investition von 300 bis 500 Mark wirklich lohnt.

Der Arithmetikprozessor 8087 wurde von Intel zur Unterstützung des 8086/8088-Prozessors bei mathematischen Operationen entwickelt. So sind im Arithmetikprozessor 8087 viele mathematischen Funktionen (Grundrechenarten, Wurzelberechnung, Trigonometrie, Logarithmen) fest verdrahtet und können in einem einzigen Programmschritt ausgeführt werden. Zudem rechnet der Arithmetikprozessor prinzipiell mit 80 Bit breiten numerischen Werten, so daß höchste Genauigkeit und ein großer Wertebereich gegeben sind.

Die Ansteuerung des Arithmetikprozessors 8087 erfolgt vom 8086/8088-Prozessor über Befehlsfolgen, die durch den Assembler-Befehl »ESC« eingeleitet werden. Der Registersatz des Arithmetikprozessors 8087 (Bild 1) besteht im Wesentlichen aus einem Registerstapel von acht 80-Bit-Registern und einem Statusregister, das die Statusbits zur Anzeige von Ausnahmebehandlungen beinhaltet. Alle numerischen Werte werden in den Registern des Registerstapels als temporäre Fließkommazahlen abgelegt. Wenn die Werte nicht von vornherein in diesem Format vorliegen,

werden sie vom Arithmetikprozessor automatisch in das genannte Format umgewandelt.

Tabelle 1 zeigt die numerischen Formate, die der Arithmetikprozessor verarbeiten kann, und Bild 2 zeigt, wie der Arithmetikprozessor numerische Werte in verschiedenen Formaten im Arbeitsspeicher des PC ablegt.

Das Statusregister des 8087 enthält auch den Stapelzeiger für den 80-Bit-Registerstapel

Das Statusregister des Arithmetikprozessors setzt sich aus acht Statusbits, einem 4-Bit-Bedingungscode und dem 3-Bit-Stapelzeiger zusammen:

Bit 0: Ungültig-Statusbit IE (invalid operation exception)

Bit 1: Unmaskiert-Statusbit DE (denormalized operand exception)

Bit 2: Nulldivision-Statusbit ZE (zerodivide exception)

Bit 3: Überlauf-Statusbit OE (overflow exception)

Bit 4: Unterlauf-Statusbit UE (underflow exception)

Bit 5: Genauigkeit-Statusbit PE (precision exception)

Bit 6: Reserviert

Bit 7: Interrupt-Statusbit IR (interrupt request)

Bit 8: Bedingungscode (Bit 0)

Bit 9: Bedingungscode (Bit 1)

Bit A: Bedingungscode (Bit 2)

Bit B: Stapelzeiger (Bit 0)

Bit C: Stapelzeiger (Bit 1)

Bit D: Stapelzeiger (Bit 2)

Bit E: Bedingungscode (Bit 3)

Bit F: Beschäftigt-Statusbit B (busy)

Der Stapelzeiger in Bit B bis D markiert mit seinem Inhalt jeweils das Register, das die Spitze des Registerstapels darstellt. Dieses Register trägt in den Assemblerbefehlen die Bezeichnung st(0), das darunter liegende Register die Bezeichnung st(7) und so weiter.

Wird beispielsweise der Wert des Stapelzeigers um 1 erhöht, so wird das über st(0) liegende Register zum neuen Register st(0) und das alte st(0) zum neuen Register st(7). Tabelle 2 zeigt die Assemblerbefehle des Arithmetikprozessors 8087 nach Funktionsgruppen geordnet und Tabelle 3 die genaue Bedeutung aller 8087-Assemblerbefehle.

(ma)

Die numerischen Formate für den 8087

Typ	Länge	Wertebereich
word integer	2 Byte	-32768 bis +32767
short integer	4 Byte	-2,14x10 ⁺⁹ bis +2,14x10 ⁺⁹
long integer	8 Byte	-9,22x10 ⁺¹⁸ bis +9,22x10 ⁺¹⁸
packed decimal	10 Byte	-9999999999999999 bis +9999999999999999
short real	4 Byte	+/- 8,43x10 ^{+/-37} bis +/- 3,37x10 ⁺³⁸
long real	8 Byte	+/- 4,19x10 ^{+/-307} bis +/- 1,67x10 ⁺³⁰⁸
temporary real	10 Byte	+/- 3,4x10 ⁻⁴⁹³² bis +/- 1,2x10 ⁺⁴⁹³²

Tabelle 1. Diese sieben numerischen Formate verarbeitet der Arithmetikprozessor 8087

Der Registersatz des Arithmetikprozessors 8087

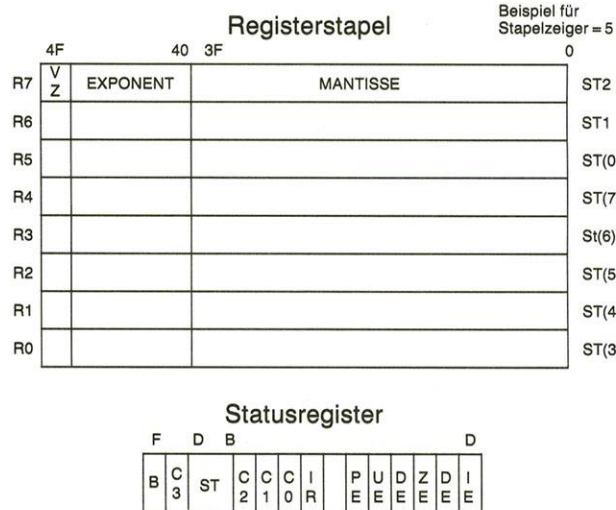


Bild 1. Der Registersatz des Arithmetikprozessors 8087 besteht hauptsächlich aus einem 80-Bit-Registerstapel und dem Statusregister

Speicherbelegung für numerische Formate

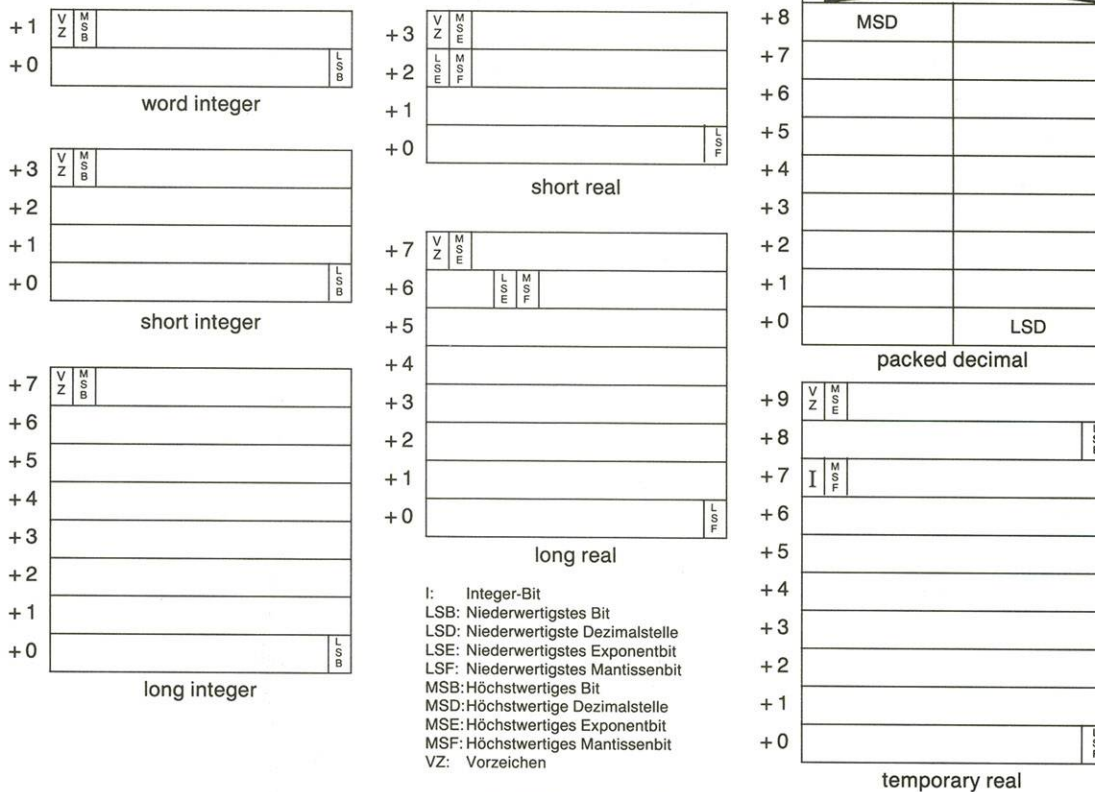


Bild 2. Auf diese Weise werden numerische Werte in verschiedenen Formaten im Arbeitsspeicher des PC abgelegt

Die Befehle des Arithmetikprozessors 8087 nach Funktionsgruppen geordnet

Bezeichnung	Funktion
Steuerbefehle	
FINIT	Arithmetikprozessor initialisieren
FFREE	Register als leer kennzeichnen
FCLEX	Exception-Statusbits, Unterbrechungsanforderung-Statusbit und Busy-Statusbit löschen
FINCSTP	Stapelzeiger inkrementieren
FDECSTP	Stapelzeiger dekrementieren
FDISI	Unterbrechungsanforderungen durch Arithmetikprozessor sperren
FENI	Unterbrechungsanforderungen durch Arithmetikprozessor freigeben
FLDCW	Steuerwort aus dem Arbeitsspeicher in das Steuerwort-Register lesen
FSTCW	Steuerwort aus dem Steuerwort-Register in den Arbeitsspeicher schreiben
FSTSW	Statuswort aus dem Statuswort-Register in den Arbeitsspeicher schreiben
FLDENV	Steuerwort, Statuswort, Tag-Feld, Befehlszeiger und Datenzeiger aus dem Arbeitsspeicher lesen
FSTENV	Steuerwort, Statuswort, Tag-Feld, Befehlszeiger und Datenzeiger in den Arbeitsspeicher schreiben
FRSTOR	Sämtliche Register des Arithmetikprozessors aus dem Arbeitsspeicher lesen
FSAVE	Sämtliche Register des Arithmetikprozessors in den Arbeitsspeicher schreiben
FNOP	Keine Operation ausführen
FWAIT	Auf Testsignal warten
Transferbefehle	
FLD	Fließkommazahl auf den Registerstapel legen
FST	Fließkommazahl vom Registerstapel lesen (Stapelzeiger bleibt unverändert)
FSTP	Fließkommazahl vom Registerstapel lesen
FXCH	Fließkommazahl des obersten Registers auf dem Stapel gegen Fließkommazahl eines anderen Registers austauschen
FILD	Ganzzahligen Wert aus dem Arbeitsspeicher auf den Registerstapel legen
FIST	Ganzzahligen Wert vom Registerstapel in den Arbeitsspeicher schreiben (Stapelzeiger bleibt unverändert)

Bezeichnung	Funktion
Transferbefehle	
FISTP	Ganzzahligen Wert vom Registerstapel in den Arbeitsspeicher schreiben
FBLD	Gepackte Dezimalzahl im BCD-Format aus dem Arbeitsspeicher auf den Registerstapel legen
FBSTP	Gepackte Dezimalzahl im BCD-Format vom Registerstapel in den Arbeitsspeicher schreiben
Addition	
FADD	Fließkommazahlen addieren
FADDP	Fließkommazahl im obersten Register des Stapels zur Fließkommazahl eines anderen Registers addieren
FIADD	Ganzzahligen Wert aus dem Arbeitsspeicher zur Fließkommazahl des obersten Registers auf dem Stapel addieren (Stapelzeiger bleibt unverändert)
Subtraktion	
FSUB	Fließkommazahlen subtrahieren
FSUBP	Fließkommazahl im obersten Register des Stapels von der Fließkommazahl eines anderen Register subtrahieren
FISUB	Ganzzahligen Wert aus dem Arbeitsspeicher von der Fließkommazahl im obersten Register des Stapels subtrahieren (Stapelzeiger bleibt unverändert)
FSUBR	Fließkommazahlen umgekehrt subtrahieren
FSUBRP	Fließkommazahl eines Registers von der Fließkommazahl im obersten Register des Stapels subtrahieren
FISUBR	Fließkommazahl im obersten Register des Stapels von einem ganzzahligen Wert im Arbeitsspeicher subtrahieren (Stapelzeiger bleibt unverändert)
Multiplikation	
FMUL	Fließkommazahlen multiplizieren
FMULP	Fließkommazahl im obersten Register des Stapels mit Fließkommazahl eines anderen Registers multiplizieren
FIMUL	Ganzzahligen Wert aus dem Arbeitsspeicher mit Fließkommazahl im obersten Register des Stapels multiplizieren (Stapelzeiger bleibt unverändert)

Die Befehle des Arithmetikprozessors 8087 nach Funktionsgruppen geordnet

Bezeichnung	Funktion	Bezeichnung	Funktion
Division		Transzendente Befehle	
FDIV	Fließkommazahlen dividieren	FYL2XP1	Fließkommazahl nach der Funktion $z = yx \log_2 (x + 1)$ berechnen
FDIVP	Fließkommazahl eines Registers durch Fließkommazahl im obersten Register des Stapels dividieren	Konstanten schreiben	
FIDIV	Fließkommazahl im obersten Register des Stapels durch ganzzahligen Wert aus dem Arbeitsspeicher dividieren (Stapelzeiger bleibt unverändert)	FLD1	Konstante 1 in das oberste Register des Stapels schreiben
FDIVR	Fließkommazahlen umgekehrt dividieren	FLDZ	Konstante 0 in das oberste Register des Stapels schreiben
FDIVRP	Fließkommazahl im obersten Register des Stapels durch Fließkommazahl eines anderen Registers dividieren	FLDPI	Konstante π in das oberste Register des Stapels schreiben
FIDIVR	Ganzzahligen Wert im Arbeitsspeicher durch Fließkommazahl im obersten Register des Stapels dividieren (Stapelzeiger bleibt unverändert)	FLDL2E	Konstante $\log_2 e$ in das oberste Register des Stapels schreiben
Weitere arithmetische Befehle		FLDL2T	Konstante $\log_2 10$ in das oberste Register des Stapels schreiben
FABS	Absolutwert der Fließkommazahl im obersten Register des Stapels berechnen	FLDLG2	Konstante $\log_{10} 2$ in das oberste Register des Stapels schreiben
FCHS	Vorzeichen der Fließkommazahl im obersten Register des Stapels komplementieren	FLDLN2	Konstante $\log_e 2$ in das oberste Register des Stapels schreiben
FPREM	Rest bei der Division von zwei Fließkommazahlen bestimmen	Vergleichsbefehle	
FRNDINT	Fließkommazahl im obersten Register des Stapels in ganzzahligen Wert umwandeln	FCOM	Fließkommazahl im obersten Register des Stapels mit einer anderen Fließkommazahl vergleichen (Stapelzeiger bleibt unverändert)
FSCALE	Fließkommazahl nach der Funktion $z = yx 2^x$ berechnen	FCOMP	Fließkommazahl im obersten Register des Stapels mit einer anderen Fließkommazahl vergleichen
FSQRT	Quadratwurzel aus der Fließkommazahl im obersten Register des Stapels berechnen	FCOMPP	Fließkommazahl im obersten Register des Stapels mit Fließkommazahl im nachfolgenden Register vergleichen, Stapelzeiger doppelt erhöhen
FXTRACT	Fließkommazahl im obersten Register des Stapels in Exponent und Mantisse aufteilen	FICOM	Ganzzahligen Wert im Arbeitsspeicher mit Fließkommazahl im obersten Register des Stapels vergleichen (Stapelzeiger bleibt unverändert)
Transzendente Befehle		FICOMP	Ganzzahligen Wert im Arbeitsspeicher mit Fließkommazahl im obersten Register des Stapels vergleichen
FPATAN	Arcustangens einer Fließkommazahl berechnen	FTST	Fließkommazahl im obersten Register des Stapels mit 0 vergleichen
FPTAN	Tangens einer Fließkommazahl berechnen	FXAM	Fließkommazahl im obersten Register des Stapels untersuchen
F2XM1	Fließkommazahl nach der Funktion $z = 2^x - 1$ berechnen		
FYL2X	Fließkommazahl nach der Funktion $z = yx \log_2 x$ berechnen		

Tabelle 2. Alle Assemblerbefehle des Arithmetikprozessors 8087 nach Funktionsgruppen geordnet

Die für den Arithmetikprozessor 8087 gültigen Assemblerbefehle

Befehl	Parameter	Bedeutung	Befehl	Parameter	Bedeutung
Steuerbefehle			Transferbefehle		
FINIT	–	Initialisiere Arithmetikprozessor	FIELD	mem	Dekrementiere Stapelzeiger, schreibe ganzzahligen Wert aus mem in st(0)
FFREE	st(i)	Kennzeichne Register st(i) als leer	FIST	mem	Schreibe Wert aus st(0) als ganzzahligen Wert in mem
FCLEX	–	Lösche Exception-Statusbits, Unterbrechungsanforderungs-Statusbit und Busy-Statusbit	FISTP	mem	Schreibe Wert aus st(0) als ganzzahligen Wert in mem, inkrementiere Stapelzeiger
FINCSTP	–	Inkrementiere Stapelzeiger	FBLD	mem	Dekrementiere Stapelzeiger, schreibe Dezimalzahl im BCD-Format aus mem als Fließkommazahl in st(0)
FDECSTP	–	Dekrementiere Stapelzeiger	FBSTP	mem	Schreibe Wert aus st(0) als gepackte Dezimalzahl im BCD-Format in mem, inkrementiere Stapelzeiger
FDISI	–	Setze Unterbrechungs-Statusbit (Unterbrechungsanforderungen durch Arithmetikprozessor gesperrt)	Addition		
FENI	–	Lösche Unterbrechungs-Statusbit (Unterbrechungsanforderungen durch Arithmetikprozessor freigegeben)	FADD	st, st(i)	Addiere Wert aus st(i) mit st(0), schreibe Ergebnis in st(0)
FLDCW	mem	Lese Steuerwort aus mem in Steuerwort-Register	FADD	st(i), st	Addiere Wert aus st(0) mit st(i), schreibe Ergebnis in st(i)
FSTCW	mem	Schreibe Steuerwort aus Steuerwort-Register in mem	FADD	mem	Addiere Fließkommazahl aus mem mit st(0), schreibe Ergebnis in st(0)
FSTSW	mem	Schreibe Statuswort aus Statuswort-Register in mem	FADD	–	Addiere Wert aus st(0) mit st(1), schreibe Ergebnis in st(1), inkrementiere Stapelzeiger
FLDENV	mem	Lese Steuerwort, Statuswort, Tag-Feld, Befehlszeiger und Datenzeiger ab mem (0E Byte)	FADDP	st(i), st	Addiere Wert aus st(0) mit st(i), schreibe Ergebnis in st(i), inkrementiere Stapelzeiger
FSTENV	mem	Schreibe Steuerwort, Statuswort, Tag-Feld, Befehlszeiger und Datenzeiger ab mem (0E Byte)	FIADD	mem	Addiere ganzzahligen Wert aus mem mit st(0), schreibe Ergebnis in st(0)
FRSTOR	mem	Lese sämtliche Register des Arithmetikprozessors ab mem (5E Byte)	Subtraktion		
FSAVE	mem	Schreibe sämtliche Register des Arithmetikprozessors ab mem (5E Byte)	FSUB	st, st(i)	Subtrahiere Wert aus st(i) von st(0), schreibe Ergebnis in st(0)
FNOP	–	Führe keine Operation aus	FSUB	st(i), st	Subtrahiere Wert aus st(0) von st(i), schreibe Ergebnis in st(i)
FWAIT	–	Warte auf aktives Signal am Testeingang	FSUB	mem	Subtrahiere Fließkommazahl aus mem von st(0), schreibe Ergebnis in st(0)
Transferbefehle			FSUB	–	Subtrahiere Wert aus st(0) von st(1), schreibe Ergebnis in st(1), inkrementiere Stapelzeiger
FLD	st(i)	Dekrementiere Stapelzeiger, schreibe Wert aus st(i) in st(0)	FSUBP	st(i), st	Subtrahiere Wert aus st(0) von st(i), schreibe Ergebnis in st(i), inkrementiere Stapelzeiger
FLD	mem	Dekrementiere Stapelzeiger, schreibe Fließkommazahl aus mem in st(0)	FISUB	mem	Subtrahiere ganzzahligen Wert aus mem von st(0), schreibe Ergebnis in st(0)
FST	st(i)	Schreibe Wert aus st(0) in st(i)			
FST	mem	Schreibe Wert aus st(0) in mem			
FSTP	st(i)	Schreibe Wert aus st(0) in st(i), inkrementiere Stapelzeiger			
FSTP	mem	Schreibe Wert aus st(0) in mem, inkrementiere Stapelzeiger			
FXCH	st(i)	Tausche Wert aus st(0) gegen Wert in st(i) aus			
FXCH	–	Tausche Wert aus st(0) gegen Wert in st(i) aus			

Die für den Arithmetikprozessor 8087 gültigen Assemblerbefehle

Befehl	Parameter	Bedeutung	Befehl	Parameter	Bedeutung
Subtraktion			Weitere arithmetische Befehle		
FSUBR	st,st(i)	Subtrahiere Wert aus st(0) von st(i), schreibe Ergebnis in st(0)	FSQRT	–	Berechne Quadratwurzel von Wert aus st(0), schreibe Ergebnis in st(0)
FSUBR	st(i),st	Subtrahiere Wert aus st(i) von st(0), schreibe Ergebnis in st(i)	FXTRACT	–	Teile Fließkommazahl aus st(0) in Exponent und Mantisse auf, schreibe Exponent in st(0), dekrementiere Stapelzeiger, schreibe Mantisse in st(0)
FSUBR	mem	Subtrahiere Wert aus st(0) von Fließkommazahl in mem, schreibe Ergebnis in st(0)	Transzendente Befehle		
FSUBR	–	Subtrahiere Wert aus st(1) von st(0), schreibe Ergebnis in st(1), inkrementiere Stapelzeiger	FPATAN	–	Berechne Arcustangens aus dem Bruch in st(1) (Zähler) sowie st(0) (Nenner), inkrementiere Stapelzeiger, schreibe Ergebnis in st(0)
FSUBRP	st(i),st	Subtrahiere Wert aus st(i) von st(0), schreibe Ergebnis in st(i), inkrementiere Stapelzeiger	FPTAN	–	Berechne Tangens als Bruch aus dem Wert von st(0), schreibe Zähler in st(0), dekrementiere Stapelzeiger, schreibe Nenner in st(0)
FISUBR	mem	Subtrahiere Wert aus st(0) vom ganzzahligen Wert in mem, schreibe Ergebnis in st(0)	F2XM1	–	Berechne Funktion $y = 2^{st(0)} - 1$, schreibe Ergebnis in st(0)
Multiplikation			FYL2X	–	Berechne Funktion $y = st(1) \times \log_2 st(0)$, inkrementiere Stapelzeiger, schreibe Ergebnis in st(0)
FMUL	st,st(i)	Multipliziere Wert aus st(i) mit st(0), schreibe Ergebnis in st(0)	FYL2XP1	–	Berechne Funktion $y = st(1) \times \log_2 (st(0) + 1)$, inkrementiere Stapelzeiger, schreibe Ergebnis in st(0)
FMUL	st(i),st	Multipliziere Wert aus st(0) mit st(i), schreibe Ergebnis in st(i)	Konstanten schreiben		
FMUL	mem	Multipliziere Fließkommazahl aus mem mit st(0), schreibe Ergebnis in st(0)	FLD1	–	Dekrementiere Stapelzeiger, schreibe die Konstante 1 in st(0)
FMUL	–	Multipliziere Wert aus st(0) mit st(1), schreibe Ergebnis in st(1), inkrementiere Stapelzeiger	FLDZ	–	Dekrementiere Stapelzeiger, schreibe die Konstante 0 in st(0)
FMULP	st(i),st	Multipliziere Wert aus st(0) mit st(i), schreibe Ergebnis in st(i), inkrementiere Stapelzeiger	FLDPI	–	Dekrementiere Stapelzeiger, schreibe die Konstante π in st(0)
FIMUL	mem	Multipliziere ganzzahligen Wert aus mem mit st(0), schreibe Ergebnis in st(0)	FLDL2E	–	Dekrementiere Stapelzeiger, schreibe die Konstante $\log_2 e$ in st(0)
Division			FLDL2T	–	Dekrementiere Stapelzeiger, schreibe die Konstante $\log_2 10$ in st(0)
FDIV	st,st(i)	Dividiere Wert aus st(0) durch st(i), schreibe Ergebnis in st(0)	FLDLG2	–	Dekrementiere Stapelzeiger, schreibe die Konstante $\log_{10} 2$ in st(0)
FDIV	st(i),st	Dividiere Wert aus st(i) durch st(0), schreibe Ergebnis in st(i)	FLDLN2	–	Dekrementiere Stapelzeiger, schreibe die Konstante $\log_e 2$ in st(0)
FDIV	mem	Dividiere Wert aus st(0) durch Fließkommazahl in mem, schreibe Ergebnis in st(0)	Vergleichsbefehle		
FDIV	–	Dividiere Wert in st(1) durch st(0), schreibe Ergebnis in st(1), inkrementiere Stapelzeiger	FCOM	st(i)	Vergleiche Wert aus st(i) mit st(0), setze entsprechend die Statusbits
FDIVP	st(i),st	Dividiere Wert aus st(i) durch st(0), schreibe Ergebnis in st(i), inkrementiere Stapelzeiger	FCOM	mem	Vergleiche Fließkommazahl aus mem mit st(0), setze entsprechend die Statusbits
FIDIV	mem	Dividiere Wert aus st(0) durch ganzzahligen Wert in mem, schreibe Ergebnis in st(0)	FCOM	–	Vergleiche Wert aus st(1) mit st(0), setze entsprechend die Statusbits
FDIVR	st,st(i)	Dividiere Wert aus st(i) durch st(0), schreibe Ergebnis in st(0)	FCOMP	st(i)	Vergleiche Wert aus st(i) mit st(0), setze entsprechend die Statusbits, inkrementiere Stapelzeiger
FDIVR	st(i),st	Dividiere Wert aus st(0) durch st(i), schreibe Ergebnis in st(i)	FCOMP	mem	Vergleiche Fließkommazahl aus mem mit st(0), setze entsprechend die Statusbits, inkrementiere Stapelzeiger
FDIVR	mem	Dividiere Fließkommazahl aus mem durch st(0), schreibe Ergebnis in st(0)	FCOMP	–	Vergleiche Wert aus st(1) mit st(0), setze entsprechend die Statusbits, inkrementiere Stapelzeiger
FDIVR	–	Dividiere Wert aus st(0) durch st(1), schreibe Ergebnis in st(1), inkrementiere Stapelzeiger	FCOMPP	–	Vergleiche Wert aus st(1) mit st(0), setze entsprechend die Statusbits, inkrementiere Stapelzeiger
FDIVRP	st(i),st	Dividiere Wert aus st(0) durch st(i), schreibe Ergebnis in st(i), inkrementiere Stapelzeiger	FICOM	mem	Vergleiche ganzzahligen Wert aus mem mit st(0), setze entsprechend die Statusbits
FIDIVR	mem	Dividiere ganzzahligen Wert aus mem durch st(0), schreibe Ergebnis in st(0)	FICOMP	mem	Vergleiche ganzzahligen Wert aus mem mit st(0), setze entsprechend die Statusbits, inkrementiere Stapelzeiger
Weitere arithmetische Befehle			FTST	–	Vergleiche Wert in st(0) mit 0, setze entsprechend die Statusbits
FABS	–	Berechne Absolutwert von Fließkommazahl aus st(0), schreibe Ergebnis in st(0)	FXAM	–	Untersuche Wert in st(0), setze entsprechend die Statusbits
FCHS	–	Komplementiere Vorzeichen der Fließkommazahl aus st(0), schreibe Ergebnis in st(0)			
FPREM	–	Dividiere Wert aus st(0) durch st(1), schreibe Rest der Division in st(0)			
FRNDINT	–	Wandle Fließkommazahl aus st(0) in ganzzahligen Wert, schreibe Ergebnis in st(0)			
FSCALE	–	Berechne Funktion $y = st(0) \times 2^{st(1)}$, schreibe Ergebnis in st(0)			

Tabelle 3. Alle Assemblerbefehle des Arithmetikprozessors 8087 mit ihrer exakten Bedeutung

Die Parameter der für den Arithmetikprozessor 8087 gültigen Assemblerbefehle

Parameter	Bedeutung	Parameter	Bedeutung
mem	Speicherstelle im Arbeitsspeicher des PC	st(0)	Register aus dem Registerstapel des Arithmetikprozessors, auf das der Stapelzeiger gegenwärtig zeigt
st	Kurzform für st(0)	st(1)	Register aus dem Registerstapel des Arithmetikprozessors, das dem Register st(0) unmittelbar folgt
st(i)	Register aus dem Registerstapel des Arithmetikprozessors (Wertebereich für i: 0 bis 7)		

Umrechnungen ganz einfach

Die Umrechnung von Hexadezimalzahlen in dezimale Werte und zurück kann zu einer echten Plage werden. Unsere Tabellen helfen Ihnen, solche Umrechnungen künftig blitzschnell vorzunehmen.

Wenn Sie sich bereits mit der Assembler- oder Maschinensprache-Programmierung befaßt haben, wissen Sie, daß numerische Werte in diesen Sprachen nicht dezimal, sondern hexadezimal notiert werden. Dies liegt daran, daß sich die Binärzahlen, mit denen der Computer rechnet, in hexadezimalen Werten besser zusammenfassen lassen.

Um Ihnen die Arbeit bei der Umrechnung von hexadezimal in dezimal und von dezimal in hexadezimal zu erleichtern, haben wir für Sie einige Tabellen zusammengestellt. Tabelle 1 dient zur Umrechnung von zweistelligen Hexadezimalzahlen in dezimale Werte. Die erste Ziffer der hexadezimalen Zahl ist spaltenweise und die zweite Ziffer zeilenweise angegeben. Tabelle 2 erleichtert allgemein die Umrechnung von bis zu sechsstelligen Hexadezimalzahlen in dezimale Werte. Die Umrechnung geschieht durch Aufteilung der Hexadezimalzahl und einfache Addition der dezimalen Gegenwerte. Das Beispiel am Ende der Tabelle erläutert die Vorgehensweise.

Tabelle 3 dient zur Umrechnung von Dezimalzahlen in hexadezimale Werte. Die Tabelle ist ähnlich wie Tabelle 1 aufgebaut. Das GW-Basic-Programm schließlich formt Dezimalzahlen bis 16777215 in hexadezimale Werte um.

(ma)

Umrechnung von hexadezimal in dezimal für zweistellige hexadezimale Werte

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Tabelle 1. Mit dieser Tabelle lassen sich zweistellige hexadezimale Zahlen in dezimale Werte umrechnen

Umrechnung von hexadezimal in dezimal für bis zu sechsstellige hexadezimale Werte:

hex	dez	hex	dez	hex	dez	hex	dez	hex	dez	hex	dez
0	0	0	0	0	0	0	0	0	0	0	0
100000	1048576	10000	65536	1000	4096	100	256	10	16	1	1
200000	2097152	20000	131072	2000	8192	200	512	20	32	2	2
300000	3145728	30000	196608	3000	12288	300	768	30	48	3	3
400000	4194304	40000	262144	4000	16384	400	1024	40	64	4	4
500000	5242880	50000	327680	5000	20480	500	1280	50	80	5	5
600000	6291456	60000	393216	6000	24576	600	1536	60	96	6	6
700000	7340032	70000	458752	7000	28672	700	1792	70	112	7	7
800000	8388608	80000	524288	8000	32768	800	2048	80	128	8	8
900000	9437184	90000	589824	9000	36864	900	2304	90	144	9	9
A00000	10485760	A0000	655360	A000	40960	A00	2560	A0	160	A	10
B00000	11534336	B0000	720896	B000	45056	B00	2816	B0	176	B	11
C00000	12582912	C0000	786432	C000	49152	C00	3072	C0	192	C	12
D00000	13631488	D0000	851968	D000	53248	D00	3328	D0	208	D	13
E00000	14680064	E0000	917504	E000	57344	E00	3584	E0	224	E	14
F00000	15728640	F0000	983040	F000	61440	F00	3840	F0	240	F	15

Ein Beispiel:

ABCDEF (hex) = A00000 + B0000 + C000 + D00 + E0 + F (hex)
 = 10485760 + 720896 + 49152 + 3328 + 224 + 15 (dez)
 = 11259375 (dez)

Tabelle 2. Hexadezimalzahlen mit einer Länge von bis zu sechs Stellen können nach diesem Schema umgerechnet werden

Umrechnung von dezimaler in hexadezimale Darstellung für dezimale Werte bis 255:

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	A	B	C	D	E	F	10	11	12	13
2	14	15	16	17	18	19	1A	1B	1C	1D
3	1E	1F	20	21	22	23	24	25	26	27
4	28	29	2A	2B	2C	2D	2E	2F	30	31
5	32	33	34	35	36	37	38	39	3A	3B
6	3C	3D	3E	3F	40	41	42	43	44	45
7	46	47	48	49	4A	4B	4C	4D	4E	4F
8	50	51	52	53	54	55	56	57	58	59
9	5A	5B	5C	5D	5E	5F	60	61	62	63
10	64	65	66	67	68	69	6A	6B	6C	6D
11	6E	6F	70	71	72	73	74	75	76	77
12	78	79	7A	7B	7C	7D	7E	7F	80	81
13	82	83	84	85	86	87	88	89	8A	8B
14	8C	8D	8E	8F	90	91	92	93	94	95
15	96	97	98	99	9A	9B	9C	9D	9E	9F
16	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9
17	AA	AB	AC	AD	AE	AF	B0	B1	B2	B3
18	BA	BB	BC	BD	BE	BF	C0	C1	C2	C3

	0	1	2	3	4	5	6	7	8	9
19	BE	BF	C0	C1	C2	C3	C4	C5	C6	C7
20	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1
21	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB
22	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5
23	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
24	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
25	FA	FB	FC	FD	FE	FF	100	101	102	103

Listing. Dieses GW-Basic-Programm rechnet Dezimalzahlen bis zum Wert von 16777215 um

```
100 α$=""
110 INPUT "Dezimaler Wert";dw
120 IF dw>16777215 THEN 110
130 FOR i=5 TO 0 STEP -1
140 z=INT(dw/16^i)
150 dw=dw-z*16^i
160 IF z>10 THEN α$=α$+STR$(z) ELSE α$=α$+CHR$(z+55)
170 NEXT i
180 PRINT "Hexadezimaler Wert:"α$
190 GOTO 100
```

Tabelle 3. Diese Tabelle erlaubt die Umrechnung von dezimalen Werten in Hexadezimalzahlen

"BYTE ME" PC-Software

TEAM 548,-- DM

**Mitgliederverwaltung
für Vereine, Verbände und Organisationen**

- bis 65.000 Mitglieder
- bis 50 Abteilungen
- bis je 50 Unterabteilungen
- Beitragsverbuchung
- Mahnwesen
- vielfacher Listendruck
- Etikettendruck
- Statistik
- Korrespondenz
- Serienbriefe
- Überweisungsträgerdruck
- Abschlüsse

TEAM-DATA: Lastschriftinzug per Diskette - Aufpreis 150,- DM

FIBU II 548,-- DM

mandantenfähige Finanzbuchhaltung

- bis 65.000 Konten
- bis 65.000 Buchungen
- Sach-, Debitoren- und Kreditorenkonten
- Summen- und Saldenlisten
- Umsatzsteuervoranmeldung
- Kontenblätterdruck
- Kontenauskunft am Monitor
- Journalausdruck
- betriebswirtschaftliche Auswertung (BWA)
- G&V-Rechnung
- Monats- und Jahresabschluß
- Schnittstelle für Datenübernahme

HERKULES

348,- DM

Multifunktionales
Adreßverwaltungsprogramm mit
Terminverwaltung
Schnittstelle zu vielen Textverarbeitungs-
programmen

HD-MASTER

98,-- DM

Festplattenverwaltungsprogramm,
einfache Bedienung, verwaltet bis 26
Programme, Installierung von Batch-
Dateien möglich, ständige Anzeige von
Datum, Uhrzeit und freiem Speicherplatz

"BYTE ME"
COMPUTERSYSTEME

Wilhelmstr. 7
5240 Betzdorf/Sieg
Tel. (02741) 23537

**Info kostenlos, -
bitte unverbindlich
anfordern!**



Nichts geht
ohne sie:

Die Software-Interrupts im PC

Sämtliche Funktionen, die BIOS und MS-DOS bieten, lassen sich über Software-Interrupts aufrufen. Der Assembler-Programmierer kann sich auf diese Weise viel Arbeit ersparen – wenn er weiß, welche Interrupts welche Funktion auslösen und welche Parameter in welchen Registern eine Rolle spielen.

Alle grundlegenden Funktionen eines Computers sind beim PC als feste Software-Routinen im BIOS implementiert. Dazu zählen beispielsweise die Unterprogramme zur Tastatureingabe, Bildschirm- und Druckerausgabe, Funktionen für Disketten- und Festplattenzugriffe sowie Routinen zur Kommunikation über eine serielle Schnittstelle.

Die MS-DOS-spezifischen Funktionen, wie zum Beispiel Programm-, Datei- und Datenträgerverwaltung sind ebenfalls als feste Routinen, diesmal allerdings im Betriebssystemkern von »io.sys« und »msdos.sys«, enthalten. All diese BIOS- und MS-DOS-Funktionen lassen sich nun aus Assembler-Programmen heraus über die sogenannten Software-Interrupts aufrufen und für eigene Programme nutzen. Für den Aufruf der Funktionen ist lediglich ein einziger Befehl, nämlich der INT-Befehl nötig. So ruft beispielsweise der Befehl »INT 21« eine Funktion des MS-DOS-Interrupts 21 auf.

Der 8086/8088-Prozessor erlaubt 100hex verschiedene Interrupts, die beim PC folgendermaßen aufgeteilt sind:

00 bis 04: Prozessor
05 : Hardcopy
06 und 07: Nicht verwendet
08 bis 0F: Interrupt-Controller
10 bis 1A: BIOS
1B und 1C: Anwender
1D bis 1F: BIOS-Parameter
20 bis 27: MS-DOS
28 bis 3F: Für MS-DOS reserviert
40 bis 5F: Für Erweiterungen reserviert
60 bis 67: Für Anwender reserviert
68 bis 7F: Nicht verwendet
80 bis 85: Für Basic reserviert
86 bis F0: Basic-Interpreter
F1 bis FF: Nicht verwendet

Die Tabelle listet alle Interrupts des PC auf. Wie Sie sehen, sind viele Interrupts für mehr als eine Funktion zuständig. In diesem Fall müssen Sie zusätzlich im AH-Register die Nummer der gewünschten Funktion eines Interrupts angeben, um die entsprechende Funktion auszulösen.

Folgende Interrupts sind für mehrere Funktionen zuständig:

Interrupt 10: Grafik
Interrupt 13: Diskette/Festplatte
Interrupt 14: Serielle Schnittstelle
Interrupt 15: Kassette
Interrupt 16: Tastatur
Interrupt 17: Drucker
Interrupt 1A: Systemuhr
Interrupt 21: MS-DOS

Zu jedem dieser Interrupts finden Sie in dieser Ausgabe einen gesonderten Beitrag, der sämtliche Funktionen mit Parametern und Bedeutung behandelt.

(ma)

Interrupt	Bedeutung
Prozessor-Interrupts	
00	Division durch 0
01	Einzelschrittmodus
02	Nicht sperrbarer Interrupt (NMI), der bei Paritätsfehler aufgerufen wird
03	Abbruchpunkt (Breakpoint)
04	Überlauf bei INTO-Aufruf
05	Hardcopy drucken (PrtSc-Taste), (00500) = 0: Korrekt beendet, (00500) = 1: Hardcopy wird gedruckt, (00500) = FF: Fehler aufgetreten
06	Nicht verwendet (Schneider PC: Aufruf einer Maus-Funktion)
07	Nicht verwendet
Interrupt-Controller	
08 (IRQ 0)	Interrupt 18,2 mal pro Sekunde durch die Systemuhr des Zeitgebers 8253 (Zeit (32 Bit) in Speicherzelle 0046C)
09 (IRQ 1)	Interrupt durch die Tastatur
0A (IRQ 2)	Interrupt durch die EGA-Karte
0B (IRQ 3)	Interrupt durch die zweite serielle Schnittstelle (COM2), SDLC-Schnittstelle oder BSC-Controller
0C (IRQ 4)	Interrupt durch die erste serielle Schnittstelle (COM1), SDLC-Schnittstelle oder BSC-Controller
0D (IRQ 5)	Interrupt durch den Festplatten-Controller
0E (IRQ 6)	Interrupt durch den Disketten-Controller
0F (IRQ 7)	Interrupt durch den Drucker (PRN, LPT1 oder LPT2)
BIOS-Interrupts	
10	Aufruf einer Bildschirm-Funktion (Funktionsnummer wird in AH übergeben)
11	Überprüfe Systemkonfiguration und übergebe Ergebnis in AX Bit 0 = 1: Diskettenlaufwerk vorhanden Bit 1 = 1: Arithmetikprozessor 8087 vorhanden Bit 2 und 3: Speicher auf der Hauptplatine in 16 KByte-Einheiten Bit 5 = 0 und Bit 4 = 0: – Bit 5 = 0 und Bit 4 = 1: 40x25 Zeichen (Farbgrafikkarte) Bit 5 = 1 und Bit 4 = 0: 80x25 Zeichen (Farbgrafikkarte) Bit 5 = 1 und Bit 4 = 1: 80 x 25 Zeichen (Monochromadapter) Bit 6 und 7: Anzahl der Diskettenlaufwerke (00 = 1, 01 = 2 und so weiter) Bit 8: – Bit 9 bis B: Anzahl der seriellen RS 232C-Schnittstellen Bit C = 1: Spieladapter vorhanden Bit D: – Bit E und F: Anzahl der angeschlossenen Drucker
12	Überprüfe Größe des Computerspeichers und übergebe Ergebnis (in KByte) in AX
13	Aufruf einer Disketten-/Festplatten-Funktion (Funktionsnummer wird in AH übergeben)
14	Aufruf einer Funktion der seriellen Schnittstelle (Funktionsnummer wird in AH übergeben)
15	Aufruf einer Kassettenrecorder-Funktion (Funktionsnummer wird in AH übergeben) Schneider PC: Erweiterte BIOS-Funktionen (batteriegepufferter RAM (NVR) und Grafik-Bitplanes für Grafikmodus 2)
16	Aufruf einer Tastatur-Funktion (Funktionsnummer wird in AH übergeben)
17	Aufruf einer Drucker-Funktion (Funktionsnummer wird in AH übergeben)
18	Rufe residentes Kassetten-Basic im ROM ab Adresse F6000 auf (sofern vorhanden)
19	Lade Betriebssystem von Sektor 0 auf Spur 0 nach Adresse 07C00 und starte es
1A	Aufruf einer Systemuhr-Funktion (Funktionsnummer wird in AH übergeben)
Anwender-Interrupts	
1B	Abbruchanforderung über die Tastatur ([Ctrl-C] oder [Ctrl-Break])
1C	Interrupt (wird 18,2 mal pro Sekunde aufgerufen)
BIOS-Parameter	
1D	Initialisierung des Bildschirms
1E	Definition der Disketten-Parameter
1F	Definition von Grafikzeichen für Farbgrafikkarte
MS-DOS-Interrupts	
20	Beendigung des laufenden Programmes
21	Aufruf einer MS-DOS-Funktion (Funktionsnummer wird in AH übergeben)
22	Adresse, die bei Beendigung des laufenden Programms aufgerufen wird
23	Adresse, zu der beim Auftreten von [Ctrl-C] verzweigt wird
24	Adresse, zu der bei Behandlung kritischer Fehler verzweigt wird
25	Diskettensektoren wahlfrei lesen AL = 0: Laufwerk A, AL = 1: Laufwerk B und so weiter CX = Zahl der Sektoren DX = erster Sektor (DS:BX) = Lesebuffer (DTA)
26	Diskettensektoren wahlfrei schreiben AL = 0: Laufwerk A, AL = 1: Laufwerk B und so weiter CX = Zahl der Sektoren DX = erster Sektor (DS:BX) = Schreibbuffer (DTA)
27	Programm beenden und für Neustart im Arbeitsspeicher behalten CS:DX = Zeiger auf Programmende + 1
28 bis 3F	Für MS-DOS reserviert
40	Für Erweiterungen reserviert
41	Definition der Festplatten-Parameter
42 bis 5F	Für Erweiterungen reserviert
60 bis 67	Für Anwender-Interrupts reserviert
68 bis 7F	Nicht verwendet
80 bis 85	Für Basic reserviert
86 bis F0	Basic-Interpreter-Interrupts
F1 bis FF	Nicht verwendet

Tabelle. Alle Interrupts des PC im Überblick

Der Video-Interrupt 10

Der Software-Interrupt mit der Nummer 10 ist für den Assembler-Programmierer das Tor zum Bildschirm. 16 Funktionen erlauben den Zugriff auf einen Monochromadapter oder eine CGA-Farbgrafikkarte.

Für den Assembler-Programmierer gibt es zwei Möglichkeiten, um etwas auf den Bildschirm auszugeben. Als erstes kann er die Ausgabefunktionen des MS-DOS-Interrupt 21 benutzen.

Die Möglichkeiten dieses Interrupts sind in dieser Beziehung jedoch recht spärlich, weil alle Funktionen zeichenorientiert arbeiten, so daß die Funktionen nur die Zeichen des ASCII-Zeichensatzes ausgeben können.

Weitaus leistungsfähiger und vielfältiger in ihren Anwendungsmöglichkeiten sind die 16 Funktionen des Video-Interrupt 10.

Diese Funktionen können auf eine CGA-Karte auch Grafik ausgeben, Bildschirmbereiche auf- oder abwärts scrollen und einzelne Bildschirmpositionen auf Textzeichen oder Grafikpunkte abfragen.

Tabelle 1 zeigt alle Funktionen des Video-Interrupt 10 mit den erforderlichen Ein- und Ausgabe-Parametern. Der Aufbau des Videomodus, der für die Funktionen 00 und 0F erforderlich ist, wird in Tabelle 2 dargestellt.

Für den Monochromadapter gilt der Videomodus 07.

Der Aufbau des Monochrom-Attributs beim Monochromadapter wird in Tabelle 3 und die Farbverteilung der CGA-Karte in Tabelle 4 erklärt.

Zum Monochrom-Attribut ist noch zu sagen, daß es sich im Bildschirmspeicher jeweils direkt hinter dem zugehörigen Zeichen befindet, so daß jedes Zeichen im Bildschirmspeicher des Monochromadapters zwei Byte an Speicherplatz belegt.

(ma)

Der Videomodus für Interrupt 10, Funktion 00 und 0F				
AL	Modus	Grafikauflös.	Textauflös.	Farben
00	Text	320x200	40x25	Monochrom
01	Text	320x200	40x25	16
02	Text	640x200	80x25	Monochrom
03	Text	640x200	80x25	16
04	Grafik	320x200	40x25	Monochrom
05	Grafik	320x200	40x25	16
06	Grafik	640x200	80x25	Monochrom
07	Text	720x350	80x25	Monochrom

Tab. 2. Der Videomodus gibt die Auflösung und maximale Farbenzahl an

Das Monochrom-Attribut d. Monochromadapters	
Code	Funktion
0	Unsichtbar
1	Unterstrichen
7	Normal
9	Leuchtend und unterstrichen
15	Leuchtend
112	Invers
129	Blinkend und unterstrichen
135	Blinkend
137	Blinkend, leuchtend und unterstrichen
143	Blinkend und leuchtend
240	Blinkend und invers

Tab. 3. gültige Werte f. d. Monochrom-Attribut

Die Farben der CGA-Grafikkarte	
Nummer	Farbe
0	Schwarz
1	Blau
2	Grün
3	Kobaltblau (blau und grün)
4	Rot
5	Violett (blau und rot)
6	Braun (grün und rot)
7	Weiß (blau, grün und rot)
8	Grau
9	Hellblau
A	Hellgrün
B	Hellkobaltblau (hellblau und hellgrün)
C	Hellrot
D	Hellviolet (hellblau und hellrot)
E	Gelb (hellgrün und hellrot)
F	Hellweiß (hellblau, hellgrün und hellrot)

Tab. 4. Die CGA-Grafikkarte kann 16 verschiedene Farben darstellen

Die Funktionen des Video-Interrupt 10

AH-Register	Eingabe	Ausgabe	Bedeutung
00	AL=Videomodus AL=2 für Monochromadapter AL=7 für Hercules-kompatibel	..	Definiere Videomodus (Textmodus oder Grafikmodus)
01	CH=00000000 (00000000=Startzeile von 0 bis 1F) CL=00000000 (00000000=Endzeile von 0 bis 1F)	..	Definiere Cursorgröße
02	DH=Zeile (0 bis 24) DL=Spalte (0 bis 79) BH=Bildschirmseite (BH=0 für Grafikmodus)	..	Positioniere Cursor (0,0=oben links, 39/79,24=unten rechts)
03	BH=Bildschirmseite (BH=0 für Grafikmodus)	DH=Zeile, DL=Spalte CH=Cursor-Startzeile CL=Cursor-Endzeile	Lese Cursorposition (0,0=oben links, 39/79,24=unten rechts)
04	..	AH=1: Lichtgriffel aktiv AH=0: Lichtgriffel inaktiv DH=Zeile (0 bis 24) DL=Spalte (0 bis 79) CH=Grafikzeile (0 bis 199) BX=Grafikspalte (0 bis 639)	Lese Lichtgriffel Position (0,0=oben links, 39/79,24=unten rechts)
05	AL=Bildschirmseite	..	Wähle aktive Bildschirmseite
06	AL=Anzahl der Zeilen (frei werdende Zeilen am Fensterende werden gelöscht) (AL=0: Lösche ganzes Fenster) CH=Obere Zeile CL=Linke Spalte DH=Untere Zeile DL=Rechte Spalte BH=Attribut für gelöschte Zeilen	..	Scrolle Bildschirm aufwärts
07	AL=Anzahl der Zeilen (frei werdende Zeilen am Fensterbeginn werden gelöscht) (AL=0: Lösche ganzes Fenster) CH=Obere Zeile CL=Linke Spalte DH=Untere Zeile DL=Rechte Spalte BH=Attribut für gelöschte Zeilen	..	Scrolle Bildschirm abwärts

AH-Register	Eingabe	Ausgabe	Bedeutung
08	BH=Bildschirmseite (BH=0 für Grafikmodus)	AL=Zeichen AH=Monochrom-Attribut	Lese Zeichen und Monochrom-Attribut von Cursorposition
09	BH=Bildschirmseite (BH=0 für Grafikmodus) AL=Zeichen BL=Monochrom-Attribut BL=Farbe (nur Grafikmodus) CX=Anzahl der Zeichen	..	Schreibe Zeichen und Monochrom-Attribut auf Cursorposition
0A	BH=Bildschirmseite (BH=0 für Grafikmodus) AL=Zeichen BL=Farbe (nur Grafikmodus) CX=Anzahl der Zeichen	..	Schreibe Zeichen auf Cursorposition
0B	Grafikmodus: BH=0: BL enthält Vordergrundfarbe BH=1: BL enthält Farbpalette BL=Vordergrundfarbe/Farbpalette (0 oder 1) Textmodus: BH=0: BL enthält Randfarbe BL=Randfarbe (0 bis 1F)	..	Wähle Farbpalette oder Farbe
0C	DX=Zeile (0 bis 199) CX=Spalte (0 bis 639) AL=Farbe (Bit 7=1: XOR-Modus)	..	Setze Grafikpunkt
0D	DX=Zeile (0 bis 199) CX=Spalte (0 bis 639)	AL=Farbe des Punktes	Überprüfe Grafikpunkt
0E	AL=Zeichen, BL=Vordergrundfarbe (nur Grafikmodus) BH=Bildschirmseite (BH=0 für Grafikmodus)	..	Schreibe Zeichen auf Bildschirm und bewege Cursor vorwärts (Teletype-Emulation)
0F	..	AH=Zeichen pro Zeile AL=Videomodus BH=Bildschirmseite (BH=0: Grafikmodus oder Modus 7)	Ermittle aktuellen Bildschirmstatus

BX, CX, DX und die Segmentregister werden gesichert. Alle Text- und Grafik-Koordinatenangaben sind dezimal.

Tabelle 1. 16 Funktionen stellt der Video-Interrupt 10 dem Assembler-Programmierer zur Verfügung

Der Interrupt 13 für Diskette und Festplatte

Über den Interrupt 13 können Sie ohne den Umweg über MS-DOS direkt auf eine Diskette oder Festplatte zugreifen – nicht nur für Kopierschutz-Knacker ein interessantes Thema!

Zur Verwaltung von Dateien und Programmen bieten zweifellos die Funktionen des MS-DOS-Interrupt 21 den größeren Komfort. Doch wenn Sie beispielsweise einen ganz bestimmten Sektor von Ihrer Diskette oder Festplatte lesen möchten, wenn Sie einzelne Sektoren beschreiben oder überprüfen wollen, wenn Sie spurweise formatieren oder defekte Sektoren kennzeichnen möchten, dann ist der Interrupt 13 genau das Richtige für Sie. Tabelle 1 zeigt die wichtigsten Funktionen des Interrupt 13 und Tabelle 2 die zu den Fehlercodes passenden Fehlermeldungen. Doch bevor Sie nun anfangen, Ihre Diskette über den Interrupt 13 zu untersuchen, müssen wir Sie warnen: Schon ein kleiner Fehler beim Programmieren kann die Daten auf Ihrer Diskette teilweise oder ganz zerstören. Das gleiche gilt natürlich auch für die Festplatte. An eine Festplatte sollten Sie sich über den Interrupt 13 wirklich nur dann wagen, wenn Sie weitreichende Kenntnisse vom Datenaufbau einer Festplatte besitzen und über einige Erfahrung im Umgang mit den Funktionen des Interrupt 13 verfügen!

(ma)

Der Fehlercode des Interrupt 13	
AH	Fehlermeldung
00	Kein Fehler aufgetreten
01	Ungültiger Befehl
02	Adreßmarke des angesprochenen Sektors ist defekt
03	Nur Diskettenlaufwerk: Datenträger ist schreibgeschützt
04	Sektor nicht gefunden
05	Nur Festplatte: erfolgloser Reset-Versuch
07	Nur Festplatte: Fehler bei der Initialisierung
08	Nur Diskettenlaufwerk: Überlauf im DMA-Controller (Datenübertragung zu schnell)
09	Ungültiger Versuch, die 64-KByte-Grenze des DMA-Controllers zu überschreiten
10	Diskettenlaufwerk: Ungültiger CRC (Prüfsummenfehler beim Lesen oder Verifizieren)
11	Festplatte: nicht korrigierbarer Lesefehler, Daten nicht gelesen
20	Nur Festplatte: Daten trotz Prüfsummenfehler gelesen
40	Fehler im Controller (zum Beispiel Spur 0 nicht gefunden)
80	Spur nicht gefunden
88	Laufwerk reagiert nicht (Wartezeit überschritten)
BB	Nur Festplatte: Fehlermeldung vom BIOS (zum Beispiel falscher Interleave-Wert)
FF	Nur Festplatte: Nicht aufschlüsselbarer Fehler aufgetreten

Tabelle 2.

Der Fehlercode steht im AH-Register

Der Aufbau des Formatpuffers:	
Byte	Inhalt
Byte 00	Spur für den ersten Sektor
Byte 01	Kopf für den ersten Sektor
Byte 02	Nummer für den ersten Sektor
Byte 03	Anzahl der Bytes im ersten Sektor (0=128, 1=256, 2=512, 3=1024)
Byte 04	Spur für den zweiten Sektor
Byte 05	Kopf für den zweiten Sektor
Byte 06	Nummer für den zweiten Sektor
Byte 07	Anzahl der Bytes im zweiten Sektor
Byte 08	Spur für den dritten Sektor
und so weiter	

Die Funktionen des Interrupt 13

AH-Register	Eingabe	Ausgabe	Bedeutung
00	[DL=80: Erste Festplatte] [DL=81: Zweite Festplatte]	..	Initialisiere Diskettensystem [und erste/zweite Festplatte]
01	..	AL=Fehlercode	Lesen Fehlercode (Nur Festplatte: Setze Fehlercode anschließend auf 0)
02	AL=Anzahl der Sektoren (ES:BX)=Lesebuffer Diskettenlaufwerk: DL=Laufwerk (0 bis 3) DH=Kopf (0 oder 1) CH=Spur (0 bis 27) CL=Sektor (1 bis 9) Festplattenlaufwerk: DL=Laufwerk (80 bis 87) DH=Kopf (0 bis 7) CH=Spur (0 bis 3FF) Bit 6,7 v. CL=Bit 8,9 v. CH Bit 0 bis 4 von CL=Sektor 0 bis 11)	AL=Anzahl der gelesenen Sektoren C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Lesen Sektor(en)
03	AL=Anzahl der Sektoren (ES:BX)=Schreibpuffer Diskettenlaufwerk: DL=Laufwerk (0 bis 3) DH=Kopf (0 oder 1) CH=Spur (0 bis 27) CL=Sektor (1 bis 9) Festplattenlaufwerk: DL=Laufwerk (80 bis 87) DH=Kopf (0 bis 7) CH=Spur (0 bis 3FF) Bit 6,7 v. CL=Bit 8,9 v. CH Bit 0 bis 4 von CL=Sektor (0 bis 11)	AL=Anzahl der geschriebenen Sektoren C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Schreibe Sektor(en)
04	AL=Anzahl der Sektoren (ES:BX)=Formatpuffer Diskettenlaufwerk: DL=Laufwerk (0 bis 3) DH=Kopf (0 oder 1) CH=Spur (0 bis 27) CL=Sektor (1 bis 9) Festplattenlaufwerk: DL=Laufwerk (80 bis 87) DH=Kopf (0 bis 7) CH=Spur (0 bis 3FF) Bit 6,7 v. CL=Bit 8,9 v. CH Bit 0 bis 4 von CL=Sektor (0 bis 11)	AL=Anzahl der überprüften Sektoren C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Überprüfe Sektor(en)
05	(ES:BX)=Formatpuffer Diskettenlaufwerk: DL=Laufwerk (0 bis 3) DH=Kopf (0 oder 1) CH=Spur (0 bis 27) Festplattenlaufwerk: DL=Laufwerk (80 bis 87) DH=Kopf (0 bis 7) CH=Spur (0 bis 3FF) Bit 6,7 v. CL=Bit 8,9 v. CH	C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Formatiere Spur
06	DL=Laufwerk (80 bis 87) DH=Kopf (0 bis 7) CH=Spur (0 bis 3FF) Bit 6,7 v. CL=Bit 8,9 v. CH	C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Kennzeichne defekte Spur mit einem entsprechenden Statusbit (nur Festplatte)
07	DL=Laufwerk (80 bis 87) DH=Kopf (0 bis 7) CH=Spur (0 bis 1023) Bit 6,7 v. CL=Bit 8,9 v. CH	C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Formatiere Festplatte ab Spur CH (nur Festplatte)
08	DL=Laufwerk (80 bis 87)	DL=Anzahl d. angeschlossenen Festplatten DH=Anzahl der Köpfe CH=Anzahl der Spuren Bit 6,7 v. CL=Bit 8,9 v. CH CL=Anzahl der Sektoren C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Ermittle Laufwerksdaten (nur Festplatte)
11	DL=Laufwerk (80 bis 87)	C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Parke Schreib-/Leseköpfe auf Spur 0 (nur Festplatte)
13	DL=Laufwerk (80 bis 87)	C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Prüfe Lesbarkeit von Sektor 0 auf jeder Spur (nur Festplatte)
14	DL=Laufwerk (80 bis 87)	C=0: Kein Fehler C=1: Fehler aufgetreten, AH=Fehlercode	Führe Selbsttest durch (nur Festplatte)

BX, CX, DX, CS, SI, DI und BP werden gesichert.

Tabelle 1. Die Funktionen des Interrupt 13

DOS DATABOX

— Das ist die Software zur DOS EXTRA Nr. 1 1987/88 —

DATABOX:

- mehr als der übliche Softwareservice
- bringt alle Programme der DOS EXTRA auf Datenträger

Für alle PCs
unter MS-DOS
als 5 1/4" oder 3 1/2"
Diskette



Die DOS-Databox bietet unter anderem:

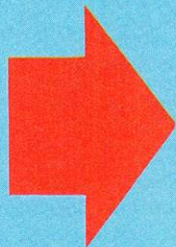
- SAM, der DOS-Assembler
- Clock, Uhr-Utilities
- Pascal – Include-Datei zum Aufruf von BIOS-Funktionen unter Turbo Pascal
- Tips und Tricks für jeden Zweck

und des weiteren die lauffähigen Dateien aller im Heft abgedruckten Listings

Diskette: **24, – DM**

Coupon ausschneiden, auf Postkarte aufkleben oder per Brief senden an:

DMV-Verlag
Postfach 250
3440 Eschwege



DOS-Bestellservice

Ich mache von Ihrem Angebot Gebrauch und bestelle hiermit:

224	_____ Stck.	CGX – Das Grafikwunder für den PC Color Grafik Extension für Schneider PC 1512/1640 bzw. IBM-kompatible	DM 199, –
225	_____ Stck.	Fraktal Generator Die neue Dimension für die Freunde fraktaler Grafiken für Schneider PC 1512/1640, Atari PC und PC's mit EGA-Karte Microsoft kompatibler Maus	DM 49, –
226	_____ Stck.	BCI Pascal/i86 für MS-DOS PC	DM 249, –
	_____ Stck.	DOS-EXTRA-Databox 5 1/4"	DM 24, –
	_____ Stck.	DOS-EXTRA-Databox 3 1/2"	DM 24, –

Gesamtbetrag _____

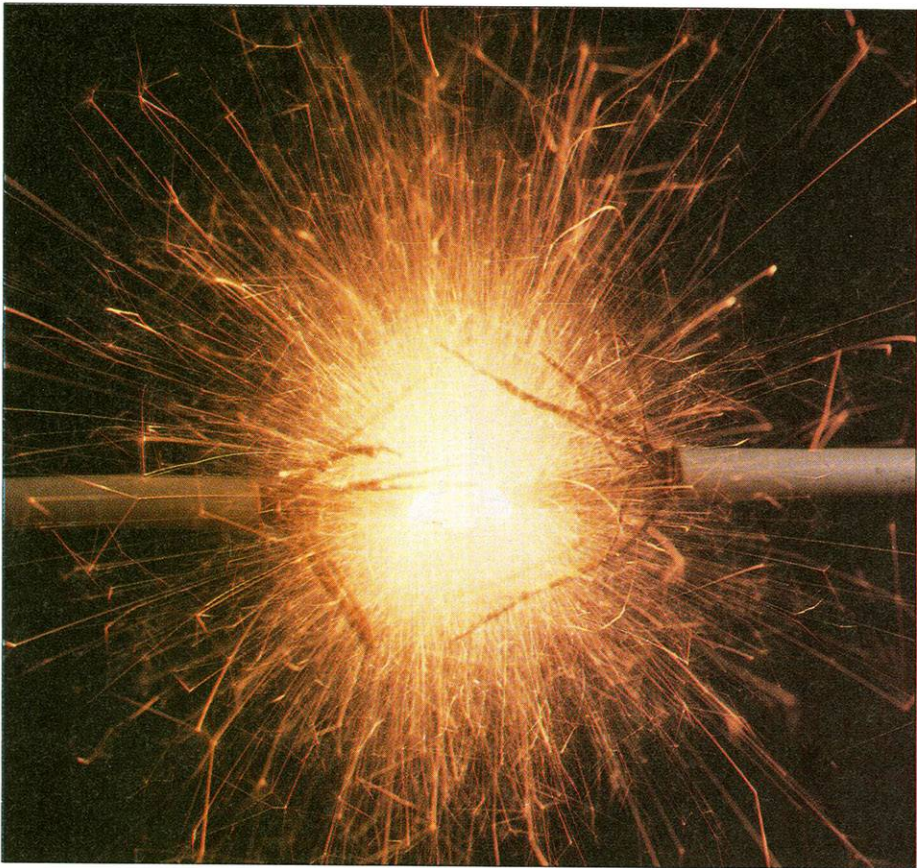
zzgl. DM 3,– Porto/Verpackung. Per Nachnahme zzgl. Nachnahmegebühr.
(In das Ausland ist Nachnahme nicht möglich)

- ☐ Den Betrag bezahle ich mittels des beigefügten Verrechnungsschecks.
☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD)

Datum _____

Unterschrift (bei Minderjährigen des gesetzl. Vertreters) _____

Der Interrupt 14 für serielle Schnittstellen



Wenn Sie in Assembler-Programmen Daten übertragen möchten, kommen Sie am Interrupt 14 nicht vorbei, denn dieser Interrupt ist für die Kommunikation mit einer seriellen Schnittstelle zuständig.

Daß die Programmierung einer seriellen Schnittstelle nicht ganz einfach ist, zeigt der Interrupt 14.

Zwar ist dieser Interrupt mit nur vier Funktionen sehr übersichtlich, doch die Parameter dieser Funktionen haben es in sich.

Die Initialisierung der Schnittstelle über die Funktion 00 durch das Konfigurationsbyte ist noch relativ leicht zu bewerkstelligen, doch die Auswertung des Zeilen- und Modem-Statusregisters während der Datenübertragung ist schon etwas kniffliger.

Aber für DFÜ-Experten wird dieser Signal-Dschungel keine Hürde darstellen, und wer in der Materie nicht so sattelfest ist, kann auf ein fertiges Programm (zum Beispiel »Procomm« aus der Public Domain) zurückgreifen.

(ma)

Die Funktionen des Interrupt 14			
AH-Register	Eingabe	Ausgabe	Bedeutung
00	DX = Schnittstelle (0 oder 1) AL = Konfiguration	AH = Zeilen-Statusregister AL = Modem-Statusregister	Initialisiere serielle Schnittstelle
01	AL = Zu sendendes Zeichen DX = Schnittstelle (0 oder 1)	AH = Zeilen-Statusregister (Bit 7 = 1: Wartezeitfehler)	Sende Zeichen
02	DX = Schnittstelle (0 oder 1)	AL = Empfangenes Zeichen AH = Zeilen-Statusregister (Bit 7 = 1: Wartezeitfehler)	Empfange Zeichen
03	DX = Schnittstelle (0 oder 1)	AH = Zeilen-Statusregister AL = Modem-Statusregister	Lese Schnittstellen-Status
Nur AX, DX und die Statusbits werden beeinflusst.			

Tabelle 1. Nur vier Funktionen bietet der Interrupt 14 dem Assembler-Programmierer an

Das Konfigurationsbyte:	
Bit	Bedeutung
Bit 0 = 0	Sieben Datenbit
Bit 0 = 1	Acht Datenbit
Bit 1 = 1	(Konstant)
Bit 2 = 0	Ein Stopbit
Bit 2 = 1	Zwei Stopbit
Bit 3 = 0	Parität gesperrt
Bit 3 = 1	Parität freigegeben
Bit 4 = 0	Ungerade Parität
Bit 4 = 1	Gerade Parität
Bit 5 bis Bit 7	Baudcode

Tabelle 2. Das Konfigurationsbyte definiert die Parameter zur Datenübertragung

Kodierung der Baudrate im Konfigurationsbyte	
Baudcode	Baudrate
0	110
1	150
2	300
3	600
4	1200
5	2400
6	4800
7	9600

Tabelle 3. 3 Bit im Konfigurationsbyte bestimmen die Datenübertragungsgeschwindigkeit

Der Aufbau des Zeilen-Statusregisters	
Bit	Funktion
0	Empfänger-Daten bereit (DR)
1	Overrun-Fehler (OE)
2	Paritäts-Fehler (PE)
3	Framing-Fehler (FE)
4	Break erkannt (BI)
5	Transmit-Holding-Register frei (THRE)
6	Transmit-Shift-Register frei (TSRE)
7	Time-Out-Fehler (TOE)

Tabelle 4. Das Zeilen-Statusregister zeigt Fehler und Bereitschaftszustände beim Zerlegen und Zusammensetzen eines Bytes an

Der Aufbau des Modem-Statusregisters	
Bit	Funktion
0	Delta klar zum Senden (DCTS)
1	Delta-Daten bereit gesetzt (DDSR)
2	Trailing-Edge-Ringindikator (TREI)
3	Delta-Daten-Carrier-Signal erkannt (DDCD)
4	Klar zum Senden (CTS)
5	Daten bereit gesetzt (DSR)
6	Ringindikator (RI)
7	Daten-Carrier-Signal erkannt (DCD)

Tabelle 5. Das Modem-Statusregister gibt den Zustand der Schnittstellensignale an

**nicht nur
für PS/2**

ADI



Bundesweit im Fachhandel erhältlich.

bäumer foto-design

Kulenkampff & Konitzky · Kohlhöckerstraße 19 · 2800 Bremen 1 · Tel. (0421) 3676-1 · Telex 244246

Der Interrupt 15 für Kassette

Sie werden es vielleicht nicht glauben, doch die ersten IBM-PCs verfügten über einen Kassettenrecorder-Anschluß. Für diesen Fall war der Interrupt 15 vorgesehen, der auch in unserer Aufstellung nicht fehlen soll.

Der Interrupt 15 ist wohl der am seltensten benutzte Interrupt im PC, denn die

wenigsten PC-Besitzer werden einen PC mit Kassettenrecorder-Anschluß und zugehörigem Kassettenrecorder besitzen.

Sollten Sie dennoch Eigentümer eines dieser betagten Geräte sein und auch tatsächlich einen Kassettenrecorder besitzen, der sich am PC anschließen läßt,

dann haben Sie über die vier Funktionen des Interrupt 15 endlich die Gelegenheit, Ihren Recorder auch in Assemblerprogrammen zu nutzen.

Vielleicht möchten Sie den Recorder aber auch nur abwechselnd ein- und ausschalten, zum Beispiel zur Beruhigung der durch die Geschwindigkeit Ihrer Festplatte gestreßten Nerven oder als Geräuschkunterhaltung für Computerspiele.

(ma)

Die Funktionen des Interrupt 15

AH-Register Eingabe		Ausgabe	Bedeutung
00	--	--	Schalte Kassettenrecordermotor ein
01	--	--	Schalte Kassettenrecordermotor aus
02	CX=Anzahl der Bytes (ES:BX)=Lesebuffer	DX=Anzahl der gelesenen Bytes AH=Kassettenstatus	Schalte Kassettenrecordermotor ein, lese Datenblock von Kassette, und schalte Kassettenrecordermotor wieder aus
03	CX=Anzahl der Bytes (ES:BX)=Schreibbuffer	AH=Kassettenstatus	Schalte Kassettenrecordermotor ein, schreibe Datenblock auf Kassette, und schalte Kassettenrecordermotor wieder aus

Schneider PC: Erweiterte BIOS-Funktionen (batteriegepufferter RAM (NVR) und Grafik-Bitplanes für Grafikmodus 2)

Tabelle 1. Mit diesen vier Funktionen läßt sich ein am PC angeschlossener Kassettenrecorder benutzen

Der Kassettenstatus

AH	Fehlermeldung
00	Kein Fehler aufgetreten
01	Ungültiger CRC (Prüfsummenfehler beim Lesen oder Verifizieren)
02	Keine Datenübergänge gefunden
04	Kein Leitsatz (Header) gefunden
80	Ungültiger Befehl

Tabelle 2. Der Kassettenstatus zeigt, ob ein Lese- oder Schreibversuch von Erfolg gekrönt ist

Der Tastatur-Interrupt 16

Wenn Sie ganz genau wissen wollen, welche Tasten gleichzeitig betätigt wurden, beziehungsweise gedrückt sind, dann hilft der Interrupt 16, dieses festzustellen.

Während die Funktionen des MS-DOS-Interrupt 21 nur ASCII-Zeichen

einlesen können, bietet der Interrupt 16 einiges mehr.

Mit ihm läßt sich beispielsweise feststellen, ob eine Taste zusammen mit [Shift], [Ctrl] oder [Alt] gedrückt wurde – und welche Sondertaste gegenwärtig aktiv ist.

Tabelle 1 zeigt die drei Funktionen des Interrupt 16.

Den Tastaturcode und dessen Bedeutung listet Tabelle 2 auf und Tabelle 3 zeigt den erweiterten Tastaturcode.

(ma)

Die Funktionen des Tastatur-Interrupt 16			
AH-Register	Eingabe	Ausgabe	Bedeutung
00	--	AX = Tastaturcode	Lese Tastaturcode aus Puffer; warte auf Taste, wenn Puffer leer ist
01	--	Z = 0: AX = Tastaturcode (Pufferzeiger unverändert) Z = 1: Puffer leer	Prüfe Tastaturcode auf der Pufferzeiger-Position und prüfe, ob Puffer leer ist
02	--	AL = Tastaturstatus	Lese Tastaturstatus

Nur AX und die Statusbits werden beeinflusst.

Tabelle 1. Drei Funktionen bietet der Tastatur-Interrupt 16.

Der Tastaturstatus:

Bit 0 = 1: Rechte Shift-Taste gedrückt
 Bit 1 = 1: Linke Shift-Taste gedrückt
 Bit 2 = 1: Ctrl-Taste gedrückt
 Bit 3 = 1: Alt-Taste gedrückt
 Bit 4 = 1: [Scroll Lock] gedrückt
 Bit 5 = 1: [Num Lock] gedrückt
 Bit 6 = 1: [Caps Lock] gedrückt
 Bit 7 = 1: [Ins] gedrückt

Die Tastaturcodes des PC (1. Teil)				
AL	Zeichen	[Shift] gedrückt	[Ctrl] gedrückt	[Alt] gedrückt
01	Esc (1B)	Esc (1B)	Esc (1B)	Esc (1B)
02	1	!	*	**
03	2	"	*	**
04	3	#	*	**
05	4	\$	*	**
06	5	%	*	**
07	6	&	*	**
08	7	/	*	**
09	8	(	*	**
0A	9	)	*	**
0B	0	=	*	**
0C	.	?	*	**
0D	.	.	*	**
0E	BS (08)	BS (08)	DEL (7F)	*
0F	HT (09)	HT back	*	*
10	q	Q	DC1 (11)	**
11	w	W	ETB (17)	**
12	e	E	ENQ (05)	**
13	r	R	DC2 (12)	**
14	t	T	DC4 (14)	**
15	z	Z	SUB (1A)	**
16	u	U	NAK (15)	**
17	i	I	HT (09)	**
18	o	O	SI (0F)	**
19	p	P	DLE (10)	**
1A	ü	Ü	*	*
1B	+	*	*	*
1C	CR (0D)	CR (0D)	LF (0A)	*
1D	[Ctrl]	*	*	*
1E	a	A	SOH (01)	**
1F	s	S	DC3 (13)	**
20	d	D	EOT (04)	**
21	f	F	ACK (06)	**
22	g	G	BEL (07)	**
23	h	H	BS (08)	**
24	j	J	LF (0A)	**
25	k	K	VT (0B)	**
26	l	L	FF (0C)	**
27	ö	Ö	*	*
28	ä	Ä	*	*
29	#	↑	*	*
2A	[Shift]	*	*	*
2B	<	>	*	*
2C	y	Y	EM (19)	**

AL	Zeichen	[Shift] gedrückt	[Ctrl] gedrückt	[Alt] gedrückt
2D	x	X	CAN (18)	**
2E	!	C	ETX (03)	**
2F	v	V	SYN (16)	**
30	b	B	STX (02)	**
31	n	N	SO (0E)	**
32	m	M	CR (0D)	**
33	.	*	*	*
34	:	*	*	*
35	-	—	US (1F)	*
36	[Shift]	*	*	*
37	[Print Screen]	[Print Screen]	**	*
38	[Alt]	*	*	*
39	SP (20)	SP (20)	SP (20)	SP (20)
3A	[Caps Lock]	*	*	*
45	[Num Lock]	*	*	*
46	[Scroll Lock]	*	*	*

AL	Zeichen	[Num Lock] gedrückt	[Ctrl] gedrückt	[Alt] gedrückt
47	[Home]	7	Bildschirm löschen	**
48	[Cursor Up]	*	*	**
49	[Page Up]	9	Textanfang + [Home]	**
4A	.	*	*	*
4B	[Cursor Left]	4	Ein Wort zurück	**
4C	.	5	*	*
4D	[Cursor Right]	6	Ein Wort vor	**
4E	+	+	*	**
4F	[End]	1	Bis Zeilenende löschen	**
50	[Cursor Down]	2	*	**
51	[Page Down]	3	Bis Bildschirmende löschen	**
52	[Insert]	0	*	*
53	[Delete]	.	*	*

*: Keine Funktion **: Erweiterter Tastaturcode in AX

Tabelle 2. Der Tastaturcode zeigt an, welche Taste beziehungsweise welche Tastenkombination gedrückt wurde.

Die Tastaturcodes des PC (2. Teil)							
AX	Funktion	AX	Funktion	AX	Funktion	AX	Funktion
0300	NUL (00)	2400	[Alt-J]	4300	[F9]	6C00	[Alt-F5]
0F00	TAB back	2500	[Alt-K]	4400	[F10]	6D00	[Alt-F6]
1000	[Alt-Q]	2600	[Alt-L]	4700	[Home]	6E00	[Alt-F7]
1100	[Alt-W]	2C00	[Alt-Y]	4800	[Cursor Up]	6F00	[Alt-F8]
1200	[Alt-E]	2D00	[Alt-X]	4900	[Page Up]	7000	[Alt-F9]
1300	[Alt-R]	2E00	[Alt-C]	4B00	[Cursor Left]	7100	[Alt-F10]
1400	[Alt-T]	2F00	[Alt-V]	4D00	[Cursor Right]	7200	[Ctrl-Print Screen]
1500	[Alt-Z]	3000	[Alt-B]	4F00	[End]		(Hardcopy ein/aus)
1600	[Alt-U]	3100	[Alt-N]	5000	[Cursor Down]	7300	[Ctrl-Cursor Left]
1700	[Alt-I]	3200	[Alt-M]	5100	[Page Down]	7400	[Ctrl-Cursor Right]
1800	[Alt-O]	3800	[F1]	5200	[Insert]	7500	[Ctrl-End]
1900	[Alt-P]	3C00	[F2]	5300	[Delete]	7600	[Ctrl-Page Down]
1E00	[Alt-A]	3D00	[F3]	5400	[Shift-F1]	7700	[Ctrl-Home]
1F00	[Alt-S]	3E00	[F4]	5500	[Shift-F2]	7800	[Alt-1]
2000	[Alt-D]	3F00	[F5]	5600	[Shift-F3]	7900	[Alt-2]
2100	[Alt-F]	4000	[F6]	5700	[Shift-F4]	7A00	[Alt-3]
2200	[Alt-G]	4100	[F7]	5800	[Shift-F5]	7B00	[Alt-4]
2300	[Alt-H]	4200	[F8]	5900	[Shift-F6]	7C00	[Alt-5]

Tabelle 3. Der erweiterte Tastaturcode belegt das gesamte AX-Register.

Der Drucker-Interrupt 17

Auch der Drucker lässt sich beim PC über einen Software-Interrupt ansteuern.

Dazu ist der Drucker-Interrupt 17 vorgesehen.

Wenn Sie in Assembler-Programmen Daten auf den Drucker ausgeben möchten, empfiehlt sich der Drucker-Interrupt 17. Dieser Interrupt stellt dem Programmierer drei Funktionen

zur Verfügung. Eine Funktion initialisiert den Drucker, eine weitere gibt ein einzelnes Zeichen aus und die dritte Funktion überprüft den Druckerstatus. Die Tabelle zeigt die drei Funktionen und der nebenstehende Kasten den Aufbau des Druckerstatusbyte. (ma)

Der Druckerstatus:

Bit 0 = 1:	Drucker reagiert nicht (Wartezeit überschritten)
Bit 1 = 0	(Konstant)
Bit 2 = 0	(Konstant)
Bit 3 = 1:	Ein-/Ausgabefehler (error)
Bit 4 = 1:	Drucker ausgewählt (selected)
Bit 5 = 1:	Papierende erreicht (paper out)
Bit 6 = 1:	Bestätigung (acknowledge)
Bit 7 = 1:	Drucker beschäftigt (busy)

Die Funktionen des Drucker-Interrupt 17

AH-Register Eingabe		Ausgabe	Bedeutung
00	AL = Zeichen DX = Drucker (0 bis 2)	AH = Druckerstatus	Drucke Zeichen auf Drucker
01	DX = Drucker (0 bis 2)	AH = Druckerstatus	Initialisiere Drucker
02	DX = Drucker (0 bis 2)	AH = Druckerstatus	Lese Druckerstatus

Nur AH wird verändert.

Tabelle. Der Drucker-Interrupt 17 stellt drei Funktionen zur Verfügung

Der Systemuhr-Interrupt 1A



Über den Systemuhr-Interrupt 1A lässt sich direkt der Zeitzähler des PC ansprechen. Aus diesem Zähler wird beispielsweise die Uhrzeit für den MS-DOS-Befehl »time« gebildet.

Während sich über die Funktion 2D des MS-DOS-Interrupt 21 die Uhrzeit des PC in Stunden, Minuten und Sekunden setzen lässt, bietet der Systemuhr-Interrupt 1A weniger Komfort. Hier bestimmt lediglich ein 32-Bit-Wert die Zeit. Doch dafür erhält man über den Interrupt 1A direkten Zugriff auf den Zeitzähler des PC, von dem aus die Uhrzeit erst berechnet wird.

Der Zeitzähler arbeitet mit dem Takt des Ticker-Interrupts im PC, das heißt, er erhöht den Zählerwert genau 18,2 Mal in der Sekunde. 24-Stunden-Intervalle werden gesondert im AL-Register registriert. Die Tabelle zeigt die beiden Funktionen des Systemuhr-Interrupt.

Die Funktionen des Systemuhr-Interrupt 1A

AH-Register Eingabe		Ausgabe	Bedeutung
00	--	AL = Anzahl der vergangenen 24-Stunden-Intervalle CX = Obere Hälfte des 32-Bit-Zeit Zählers DX = Untere Hälfte des 32-Bit-Zeit Zählers	Lese Zeitzähler (Zeitähler/18,2 = Sekunden)
01	CX = Obere Hälfte des 32-Bit-Zeit Zählers DX = Untere Hälfte des 32-Bit-Zeit Zählers	--	Setze Zeitzähler (Zeitähler/18,2 = Sekunden)

Tabelle. Die Funktion 00 liest und die Funktion 01 setzt den Zeitzähler

(ma)

So ist ein PC aufgebaut



Die Elektronik eines PC besteht aus der Mutterplatine und den Steckkarten. Während sich die Steckkarten nach Belieben austauschen lassen, bleibt der PC-Besitzer an seine Mutterplatine gebunden. Sie stellt nämlich die Basiseinheit des PC dar und bestimmt in entscheidendem Maße die Leistungsfähigkeit des Computersystems.

Die Mutterplatine des PC ist zwar nur eine Computer-Baugruppe unter vielen, doch als Basiseinheit stellt sie die wichtigste und oft auch komplexeste Komponente des PC dar. Das Herz der Mutterplatine ist der Mikroprozessor 8086 oder 8088 von Intel. Beide Prozessoren sind 16-Bit-Prozessoren, doch der 8088 arbeitet im Unterschied zum 8086 mit einem externen 8-Bit-Datenbus.

Der 8086/8088-Prozessor bearbeitet die Befehle eines Programms, reagiert auf die Unterbrechungsanforderungen (Interrupts) der Peripherie und verwaltet den Speicher sowie weitere Computerbausteine. Der Prozessor wird oft auch als CPU bezeichnet, was für »central processing unit«, das heißt »zentrale Verarbeitungseinheit« steht.

Bei den mathematischen Befehlen unterstützt den Prozessor der Arithmetikprozessor 8087, sofern dieser auf der Mutterplatine vorhanden ist. Fast alle Mutterplatinen besitzen zwar einen Sockel für den Arithmetikprozessor, doch nur in den wenigsten Fällen ist er auch serienmäßig im Computer eingebaut.

Dies liegt einerseits im verhältnismäßig hohen Preis des Arithmetikprozessors begründet (300 bis 500 Mark, je nach Ausführung), andererseits unterstützen bislang nur wenige Programme den Arithmetikprozessor, so daß die Leistung dieses Prozessors brachliegt. Mittlerweile sind aber

mehr und mehr Programme und Programmiersprachen in der Lage, die Kapazität des 8087 auszunutzen.

Mikroprozessor und Arithmetikprozessor benötigen zur Synchronisation mit dem Speicher und der Peripherie einen externen Takt, für den ein spezieller Taktgenerator zuständig ist. Im PC erzeugt der Taktgenerator vom Typ 8284 mit Hilfe eines Quarzes zwei verschiedene Taktfrequenzen. Der Quartz im PC schwingt mit einer Frequenz von 14,318 MHz. Dieser Systemtakt wird vom Taktgenerator 8284 einerseits durch drei geteilt, um den CPU-Takt von 4,77 MHz für den Prozessor und die Systemuhr des Timer-Bausteins 8253 zu erzeugen, zum anderen teilt der Taktgenerator den Systemtakt durch vier, um den Bus-Takt mit einer Frequenz von 3,58 MHz zu erhalten. Diesen Takt benötigt beispielsweise die CGA-Farbgrafikkarte für den Bildschirmaufbau.

Der Speicherbereich des PC unterliegt einer festen Aufteilung

Der 8086/88-Prozessor kann 1 MByte Speicher (das sind 1048576 Speicherzellen) adressieren. In diesem Bereich liegt zu Beginn der Arbeitsspeicher (RAM), der maximal 640 KByte groß sein darf. Darauf folgt der Bildschirmspeicher, der je nach Grafikkarte über eine Speicherkapazität von 4 KByte (Monochromadapter) bis 256

KByte (EGA-Karte) verfügt. An den Bildschirmspeicher schließt sich ein für weitere Steckkarten und Peripheriegeräte reservierter Speicherbereich an (zum Beispiel für das Festplatten-BIOS). Den Abschluß des 1-MByte-Speicherbereichs bildet ein Festwertspeicher (RAM), der als BIOS (basic input output system) bezeichnet wird und die zur Verwaltung der PC-Hardware nötigen Betriebssystem-Routinen enthält (Bildschirmsteuerung, Tastaturabfrage und so weiter).

Nach dem Einschalten oder nach einem Reset ruft der PC das BIOS auf. Das Startprogramm im BIOS führt einen Speichertest durch, überprüft die weitere Hardware des PC und initialisiert das Computersystem, indem es die Register der verschiedenen Computerbausteine programmiert. Dadurch startet beispielsweise der Timer-Baustein die Systemuhr des PC und sorgt für das regelmäßige Auffrischen der dynamischen Speicherbausteine des Arbeitsspeichers.

Beim IBM-PC enthält der Festwertspeicher neben dem BIOS auch noch die Programmiersprache Basica, die zum Basic-Dialekt GW-Basic von Microsoft weitgehend kompatibel ist und als ROM-gestütztes Programm sofort nach dem Einschalten des Computers zur Verfügung steht.

Der Arbeitsspeicher des PC dient zur Aufnahme des Betriebssystems und zum Speichern von Daten und Programmen. Je nach Ausführung befin-

den sich 256 bis 640 KByte RAM auf der Mutterplatine. Zur Erhöhung der Datensicherheit ist jedes Byte nicht acht, sondern neun Bit breit, wobei das neunte Bit als sogenanntes Paritätsbit benutzt wird. Hier überprüft im PC ein Paritätsgenerator vom Typ 74S280, ob das neunte Bit für die acht zugehörigen Datenbits die korrekte Parität anzeigt. Das neunte Bit ist damit eine Art Prüfsumme, die eventuelle Übertragungs- und Speicherfehler anzeigt. Hat der Paritätsprüfgenerator einen Paritätsfehler erkannt, so erzeugt er den nichtsperrbaren Interrupt 2 (NMI), der dem Prozessor einen schwerwiegenden Speicherfehler anzeigt und die Arbeit unterbricht.

Die Mutterplatine der meisten PCs besitzt im Gegensatz zu der Mutterplatine eines AT noch zwei bis drei Schalterblöcke, bestehend aus je acht Dipschaltern (dual in package). Über diese Dipschalter läßt sich die Systemkonfiguration des PC, beispielsweise die Größe des Arbeitsspeichers, die Anzahl der Diskettenlaufwerke und das Vorhandensein eines Arithmetikprozessors einstellen.

Die Stellung dieser Dipschalter fragt der Parallel-Portbaustein 8255 ab und kann von einem Programm über den

Software-Interrupt 17 eingelesen werden. Der Parallel-Portbaustein ist auch für die Abfrage der Tastatur (die sich über den Interrupt 9 bemerkbar macht) und für die Freigabe des Tons zuständig.

Der Sound des PC besteht aus den Rechteckschwingungen eines Uhrenzählers

Den Ton selbst erzeugt wiederum der Timer-Baustein 8253. Dieser Baustein enthält drei interne Zähler. Der erste Zähler erzeugt über Kanal 0 18,2 mal pro Sekunde den Interrupt 8, der die Systemuhr des PC weiterschaltet. Der zweite Zähler sorgt über Kanal 1 für das regelmäßige Auffrischen der dynamischen Speicherbausteine (andernfalls würde der Speicherinhalt des Arbeitsspeichers verlorengehen), und der dritte Zähler erzeugt über den Kanal 2 die Rechteckschwingungen für den Lautsprecher. Dieser Zähler war einst auch für den Kassettenrecorderanschluß des PC zuständig.

Die Interrupt-Anforderungen, die von den einzelnen Bausteinen des PC ausgehen, laufen abgesehen von dem nichtsperrbaren NMI-Interrupt im Interrupt-Controller zusammen. Der In-

terrupt-Controller besitzt acht Eingänge für Interrupt-Anforderungen. Er wertet die Interrupts nach ihrer Priorität, die die jeweilige Interruptnummer bestimmt, aus und entscheidet, ob die Interrupt-Anforderungen tatsächlich den Prozessor unterbrechen dürfen.

Der Interrupt des Timer-Bausteins 8253 besitzt die niedrigste Nummer und hat damit die höchste Priorität. Darauf folgt die Interrupt-Anforderung der Tastatur und am Schluß die Anforderungen der Steckkarten (Grafikkarte, serielle und parallele Schnittstelle, sowie Disketten- und Festplatten-Controller).

Der DMA-Controller 8237 ist neben den beiden Prozessoren der komplexeste Computerbaustein auf der Mutterplatine des PC. Seine Aufgabe ist die Übertragung der Datenblöcke von Diskette oder Festplatte in den Arbeitsspeicher und zurück, sowie das Verschieben von Datenblöcken innerhalb des Arbeitsspeichers.

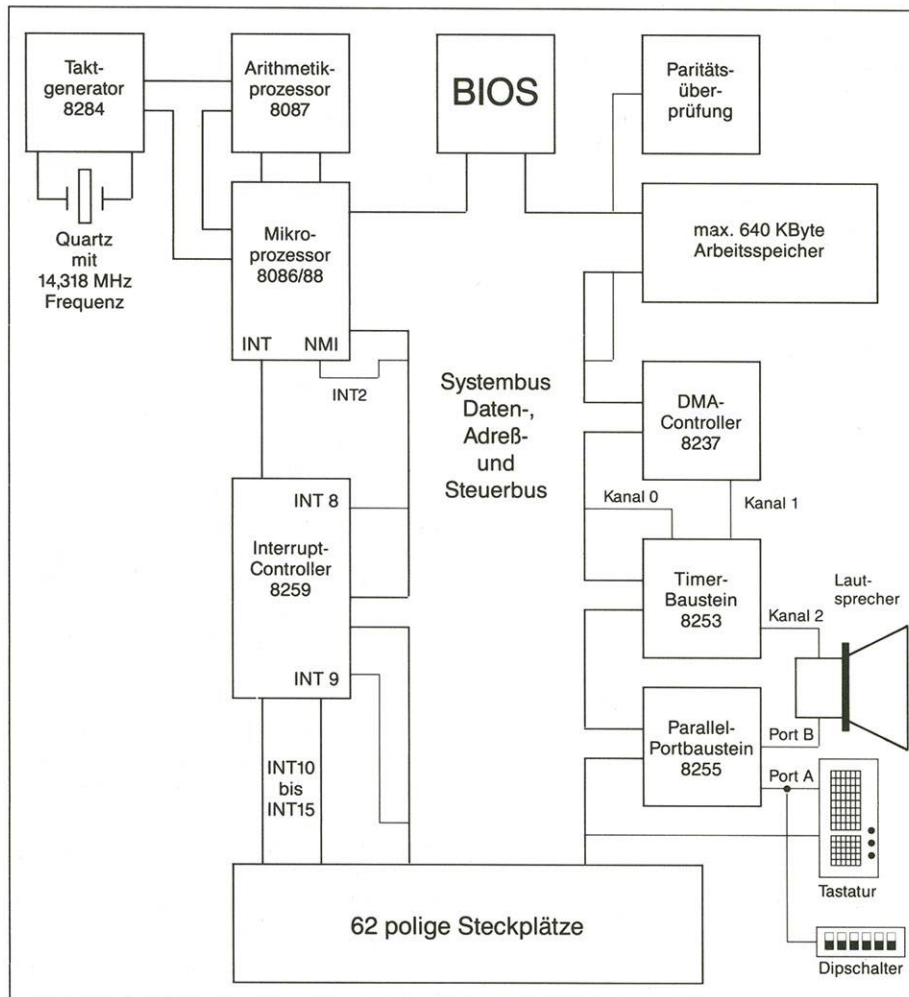
Diese Aufgabe könnte auch der 8086/8088-Prozessor übernehmen, doch der DMA-Controller ist speziell für die Datenübertragung konstruiert und arbeitet dadurch wesentlich schneller als herkömmliche Mikroprozessoren.

Der DMA-Controller 8237 besitzt vier DMA-Kanäle, die unabhängig voneinander Daten übertragen können. Drei dieser DMA-Kanäle lassen sich beliebig, beispielsweise auch von Erweiterungskarten nutzen. Der vierte Kanal dagegen ist für das Refresh reserviert und übernimmt in Zusammenarbeit mit Kanal 1 des Timer-Bausteins das periodische Auffrischen der einzelnen Speicherbausteine.

Die letzte, doch bei weitem nicht unwichtigste Komponente der PC-Mutterplatine sind die 62poligen Steckplätze, an die der komplette Systembus des PC, das heißt der Daten-, Adreß- und Steuerbus geführt wird. Hinzu kommen noch einige Interrupt-Leitungen des Interrupt-Controllers und die Steuersignale für die drei frei verfügbaren Kanäle des DMA-Controllers.

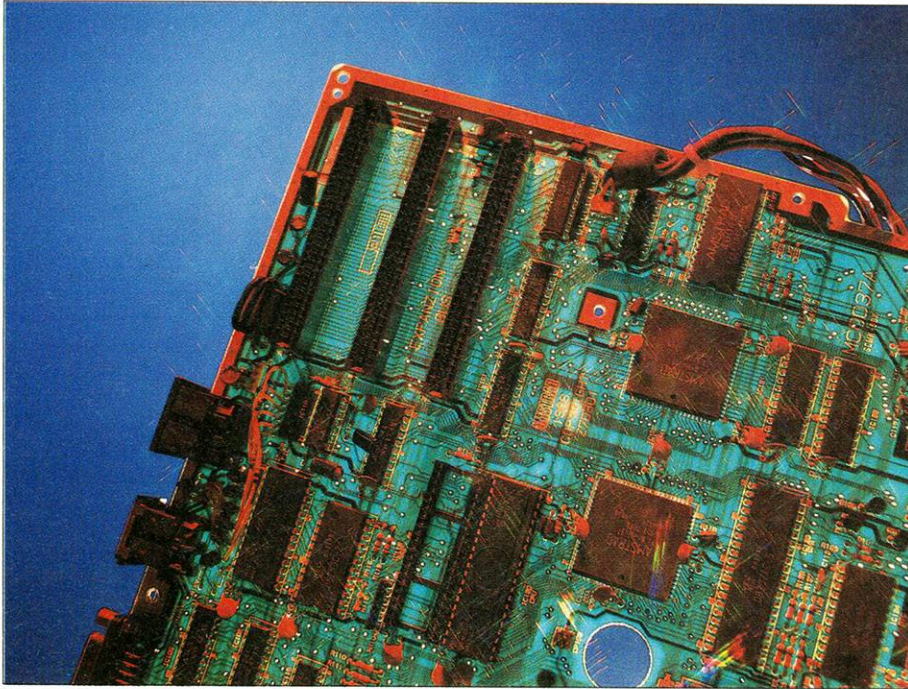
Für den ordnungsgemäßen Betrieb eines PC ist neben der Mutterplatine zumindest ein Monochromadapter für die Darstellung von Text auf dem Bildschirm und ein Disketten-Controller für den Anschluß von ein oder zwei Disketten-Laufwerken erforderlich. Damit Sie Ihre soeben erworbenen Kenntnisse vertiefen können, werden der Video-Controller 6845 des Monochromadapters, der Disketten-Controller UPD 765 des Disketten-Controllers sowie die in diesem Beitrag vorgestellten Computerbausteine in einzelnen Beiträgen dieser Ausgabe ausführlich vorgestellt. Bitte blättern Sie um!

(Martin Hepp/ma)



Das Blockschaltbild einer Standard-Mutterplatine für PCs

Ausbau leicht gemacht – die Steckplätze des PC



Die Steckplätze eines PC sind neben dem stabilen Gehäuse und der abgesetzten Tastatur der entscheidende Unterschied zu einem gewöhnlichen Heimcomputer. Während ein Heimcomputer alle zwei Jahre technisch veraltet ist, kann der PC durch die Erweiterungskarten bei der technischen Entwicklung mithalten.

Während noch vor wenigen Jahren bei einem PC für jede Speichererweiterung, Schnittstelle oder Grafikkarte eine neue Steckkarte erforderlich war, verfügen PCs neueren Datums bereits in der Grundausstattung über 640 KByte Arbeitsspeicher und je eine parallele und serielle Schnittstelle auf der Hauptplatine. Im Schneider PC 1640 ist sogar zusätzlich die Elektronik einer kompletten CGA- beziehungsweise EGA-Grafikkarte auf die Hauptplatine gepackt worden.

Trotzdem sind damit die Steckplätze im PC nicht überflüssig geworden, denn die technische Entwicklung hat uns attraktive neue Karten beschert, die jeder PC-Besitzer gern in seinem Computer sehen würde. Dazu zählen Festplatten-Controller oder ganze Festplatten zum Einstecken ebenso wie EGA- und VGA-Grafikkarten oder 286er-Karten, die einen PC durch die Verwendung eines 80286-Prozessors deutlich beschleunigen. Und auch dann, wenn die Funktionen dieser Steckkarten einmal auf der Hauptplatine eines PC verfügbar sein werden, wird es neue Karten mit viel

leistungsfähigeren Funktionen geben, die wieder einen Standard-PC in ein Wunderwerk der Technik verzauern. Dies beweist beispielsweise die Entwicklung auf dem AT-Markt. Hier werden die 386er-Karten, die einen

AT für 4000 Mark zum 386er-Computer aufrüsten, den Händlern fast aus den Händen gerissen. Der Steckplatz wird also seinen hohen Stellenwert im PC auch in Zukunft behalten.

Technisch bezeichnet man einen Steckplatz oder Slot auch als E/A-Kanal (Ein-/Ausgabe-Kanal), weil über diesen Anschluß Daten von der Hauptplatine zur Steckkarte und von dort eventuell weiter zur Peripherie gelangen. Umgekehrt kann die Peripherie Informationen über die Steckkarten an die Hauptplatine des PC weitergeben.

In einem PC sind in der Regel fünf bis acht Steckplätze vorhanden. Jeder dieser Steckplätze bildet eine vollwertige Erweiterung für den Prozessorbus des 8086 oder 8088, so daß für den Prozessor zwischen den Bausteinen auf der Hauptplatine und den Bausteinen auf einer Steckkarte praktisch kein Unterschied besteht. Die Elektronik auf der Hauptplatine eines PC läßt sich folglich über die Steckkarten beliebig erweitern. Nur die Anzahl der Steckplätze setzt dem Ausbau Grenzen.

Für den Kontakt zwischen Steckplatz und Karte ist auf der Hauptplatine an jedem Steckplatz eine 62polige Steckleiste aufgelötet, in die der dazu passende Platinezapfen einer Steckkarte gedrückt wird. Die 62 Anschlüsse der Steckleiste sind komplett belegt, so daß an jedem Steckplatz 62 Leitungen zur Verfügung stehen. Diese Leitungen sind bis auf das Kartenprüfsignal (I/O CH CK) auf

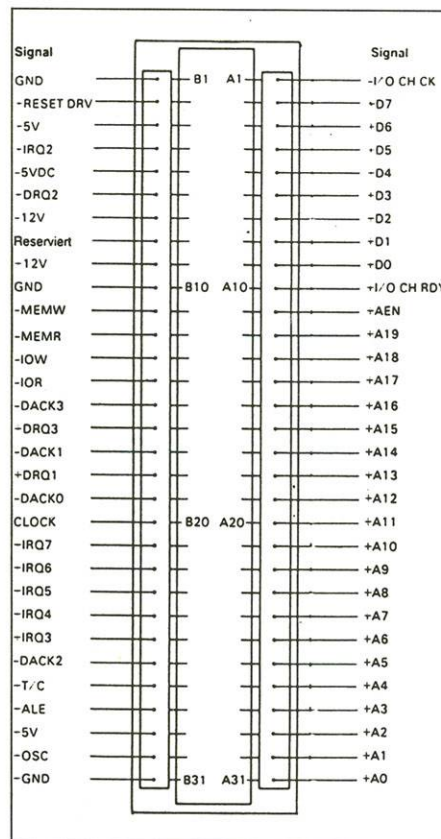


Bild. Die Pinbelegung eines Steckplatzes

jedem Steckplatz identisch und alle Steckplätze sind gleichberechtigt. Alle Signalleitungen führen TTL-Pegel (+ 5 Volt oder 0 Volt), so daß keine Spannungsumsetzung zwischen der Hauptplatine und den Steckkarten erforderlich ist.

Jeder Steckplatz stellt alle 20 Adreßleitungen, die acht Datenleitungen D0 bis D7, sechs Interruptsignale, sechs DMA-Signale sowie alle für die Speicherverwaltung und Adressierung notwendigen Signale zur Verfügung. Weiterhin sind Taktsignale, Spannungsleitungen und einige weitere Prüf- und Steuersignale vorhanden.

Damit der Systembus des PC nicht überlastet wird, sind alle Leitungen durch integrierte Treiberbausteine gepuffert und dadurch in der Lage, mehrere Eingänge zugleich zu bedienen. Einige Anschlüsse sind direkt mit dem Netzteil verbunden und führen Spannungen von + 12 Volt, + 5 Volt, -5 Volt oder -12 Volt. Hier läßt sich im begrenzten Maße Spannung abgreifen, doch sollte der Stromfluß abhängig von der Leiterbahnbreite und Ausgangsleistung des Netzteils wenige bis einige hundert Milliampere nicht überschreiten.

Die Steuerung der Kommunikation zwischen Hauptplatine und Steckkarte erfolgt über sechs Leitungen für Interrupt-Anforderungen (IRQ2 bis IRQ7), die den Interrupt-Controller 8259 anlaufen, und über drei Leitungen für DMA-Anforderungen (DRQ1 bis DRQ3), die direkt den DMA-Controller 8237 ansprechen. Diese Anforderungen werden durch die drei DMA-Bestätigungssignale DACK1 bis DACK3 beantwortet. Die Tabelle zeigt Ihnen detailliert den Aufbau eines Steckplatzes und das Bild zeigt die Pinbelegung.

Das Zusammenspiel von Steckkarte und Hauptplatine ist gut durchdacht

Besonders wichtig für eine korrekte Steuerung der Steckkarten mit den daran angeschlossenen Peripheriegeräten ist die richtige Synchronisation mit dem Computer. Dazu läuft sowohl der Systemtakt als auch der Takt des 14,318-MHz-Quarzes über jeden Steckplatz. Über eine sogenannte »Bereit«-Leitung (I/O CK RDY) kann eine Steckkarte dem Computer anzeigen, daß ihr die Systemgeschwindigkeit zu hoch ist. In diesem Fall fügt der Taktgenerator 8284 Wartezyklen (Waitstates) in den Arbeitsablauf ein, um der langsameren Karte den Zugriff auf die Hauptplatine zu gestatten.

* Alle Signale mit einem Stern vor der Bezeichnung arbeiten mit negativer Logik, daß heißt sie sind nullaktiv.

Signal	E = Eingang A = Ausgang	Pin	Signalbeschreibung
*I/O CH CK	E	A1	Ein-/Ausgang-Steckplatzüberprüfung. Liegt diese Leitung auf 0, so signalisiert dieser Pegel für den Prozessor einen Adreßfehler oder Paritätsfehler auf einer der Steckkarten.
D0 bis D7	E/A	A2 bis A9	Datenleitungen 0 bis 7. Sie liefern die Daten für Prozessor, Speicher und Steckkarten. D0 bezeichnet das niederwertigste und D7 das höchstwertige Datenbit.
I/O CK RDY	E	A10	Steckkarte ist bereit. Im Normalfall liegt diese Leitung immer auf 1, das heißt die Steckkarte ist zur Arbeit mit der normalen Systemgeschwindigkeit bereit. Langsamere Karten legen diesen Anschluß auf 0. Dadurch verlängert der Computer die Zyklen für Datenein- und -ausgabe, um auch der langsamen Karte den Zugriff auf das Computersystem zu ermöglichen. Die Maschinenzyklen werden in dieser Betriebsart um ein Vielfaches von 210 ns (ein Taktzyklus) verlängert. Die Leitung darf maximal zehn Taktzyklen lang auf 0 liegen.
AEN	A	A11	Mit Hilfe dieser Leitung kann der Computer über eine Karte auch dann noch DMA-Übertragungen durchführen, nachdem die Karte vom Prozessor und anderen Ein-/Ausgabe-Einheiten getrennt wurde. Bei aktiver Leitung kontrolliert die DMA-Steuerung den Adreß- und Datenbus sowie die Schreib- und Leseleitungen des Speichers.
A0 bis A19	A	A12 bis A31	Adreßleitungen 0 bis 19. Dies sind die zur Adressierung des Speichers und der Ein-/Ausgabe-Einheiten zuständigen Leitungen. Sie erlauben die Adressierung von 1 MByte Speicher und werden entweder vom Prozessor oder dem DMA-Controller erzeugt. A0 bezeichnet das niederwertigste und A19 das höchstwertige Adreßbit Masse. Dieser Anschluß führt den 0-Volt-Pegel des Computers.
GND	—	B1	
RESET	—	B1	
DRV	A	B2	Über diesen Anschluß erfolgt die Neu-Initialisierung des Computersystems
+ 5V	—	B3	Positive Versorgungsspannung vom Netzteil.
IRQ2	E	B4	Interruptanforderung an den Prozessor durch eine Karte. IRQ0 hat die höchste, IRQ7 die niedrigste Priorität. Nur die Systemuhr und die Tastatur des PC haben über die Leitungen IRQ0 und IRQ1 eine höhere Interrupt-Priorität, als die IRQ-Signale der Steckkarten. Bei einer Interruptanforderung geht der Anschluß auf 1 und verbleibt bis zur Quittierung durch den Prozessor (Acknowledge) in diesem Zustand.
-5V	—	B5	Negative Versorgungsspannung vom Netzteil.
DRQ2	E	B6	DMA-Anforderung. Asynchrone Anforderung eines Peripheriegerätes für den Zugriff auf eine DMA-Funktion. DRQ1 hat die höchste, DRQ3 die niedrigste Priorität. Bei einer Anforderung wechselt dieser Anschluß auf 1 und verbleibt bis zur Aktivierung der zugehörigen DACK-Leitung in diesem Zustand.
-12V	—	B7	Negative Spannung vom Netzteil
Reserviert	—	B8	Für spezielle Signale einiger exotischen Karten reserviert
+ 12V	—	B9	Positive Spannung vom Netzteil
GND	—	B10	
*MEMW	A	B11	Masse. Dieser Anschluß führt den 0-Volt-Pegel des Computers. Speicher-Schreibbefehl. Dieses Signal veranlaßt den Arbeitsspeicher des PC, die auf dem Datenbus anliegenden Daten zu speichern. Das Signal wird vom Prozessor oder der DMA-Steuerung erzeugt.
*MEMR	A	B12	Speicher-Lesebefehl. Dieses Signal veranlaßt den Speicher des PC, Daten auf den Datenbus zu legen. Ansteuerung wie bei MEMW.
*IOW	A	B13	Steckkarten-Schreibbefehl. Durch diesen Befehl weist der Computer eine Karte an, Daten vom Datenbus zu lesen. Ansteuerung wie bei MEMW.
*IOR	A	B14	Steckkarten-Lesebefehl. Über diese Leitung wird eine Karte angewiesen, Daten auf den Datenbus zu legen. Ansteuerung wie bei MEMW.
*DACK3	A	B15	DMA-Bestätigung. Dieses Signal quittiert die über das DRQ-Signal gestellte DMA-Anforderung.
DRQ3	E	B16	Siehe DRQ2
*DACK1	A	B17	Siehe DACK3
DRQ1	E	B18	Siehe DRQ2
*DACK0	A	B19	DMA-Bestätigung. Dieses Signal sorgt im PC für einen Refreshzyklus.
CLK	A	B20	Systemtakt des Computers. Er beträgt 4,77 MHz und hat eine Zykluszeit von 210ns.
IRQ7	E	B21	Siehe IRQ2
IRQ6	E	B22	Siehe IRQ2
IRQ5	E	B23	Siehe IRQ2
IRQ4	E	B24	Siehe IRQ2
IRQ3	E	B25	Siehe IRQ2
*DACK2	A	B26	Siehe DACK3
T/C	A	B27	Terminalzähler. Dieses Signal liefert einen Impuls, wenn der Terminalzählerstand für einen beliebigen DMA-Kanal erreicht ist.
ALE	E	B28	Adressenverriegelung. Mit Hilfe dieser Leitung lassen sich die derzeit gültigen Adressen gegenüber dem Prozessor verriegeln. Die Verriegelung erfolgt durch den Bus-Controller 8288. In Verbindung mit dem Signal AEN steht ALE auf dem Steckplatzbus als Hinweis für eine gültige Prozessoradresse zur Verfügung.
+ 5V	—	B29	Positive Versorgungsspannung vom Netzteil.
OSC	A	B30	Oszillatortakt. An diesem Anschluß läßt sich die Quarz-Frequenz von 14,318 MHz direkt abgreifen.
GND	—	B31	Masse. Dieser Anschluß führt den 0-Volt-Pegel des Computers.

Tab. Diese Signale stehen am Steckplatz eines PC für die Erweiterungskarte zur Verfügung

Komplexere Erweiterungen erledigen diese Aufgabe von sich aus und geben entsprechende Signale direkt an den DMA-Controller. In diesem Fall bekommt der Prozessor lediglich über einen der acht DMA-Kanäle die Meldung, daß eine bestimmte Steckkarte langsamer arbeiten muß.

Im Normalfall dauert eine Schreib-/Leseoperation des Prozessors bei einem Speicherzugriff vier Takte, wovon jeder eine Länge von 210 ns hat (eine Taktfrequenz von 4,77 MHz vorausgesetzt). Demnach benötigt der Computer zur Verarbeitung eines Byte 840 ns (Nanosekunden). Im Unterschied hierzu benötigt der Prozessor für einen Schreib-/Lese-Zyklus bei der Daten-Ein-/Ausgabe fünf Takte, das heißt 1050 ns pro übertragene Byte.

Die zur Datenerhaltung nötigen Refresh-Zyklen für die dynamischen Speicherbausteine generiert das System ungefähr alle 1500 ns, also jeden 72. Takt. Dabei beansprucht ein Refresh-Zyklus insgesamt vier Systemtakte.

Um die Steckplätze anzusprechen, verwendet der PC sogenannte Adreßplatzabbilder. Dies sind Bilder des gesamten für die Erweiterungsplätze zur Verfügung stehenden, adressierbaren Speicherplatzes, die im Spei-

cherplatz des Computers liegen. Aus diesem Grund hat man den zur Adressierung der Steckplätze zuständigen DMA-Kanal so ausgelegt, daß 512 Adressen für Ein-/Ausgabeeinheiten zur Verfügung stehen.

Um Fehler in der Adressierung oder auf den Steckkarten feststellen zu können, existiert für jeden Steckplatz eine eigene Prüfleitung (I/O CH CK). Mit Hilfe dieser Leitung kann ein Peripheriegerät oder eine Steckkarte, sofern Sie dafür ausgelegt ist, Adressierungsfehler selbstständig erkennen. So werden fehlerhafte Informationen vom Prozessor ferngehalten. Bekäme der Prozessor falsche Informationen über einen Steckplatz, so hätte jedes Ansprechen der entsprechenden Adresse unweigerlich einen Systemabsturz zur Folge.

Tritt ein Fehler auf einer Steckkarte auf und aktiviert dadurch die Prüfleitung, so führt das zu dem nichtsperrbaren Interrupt NMI, der dem Software-Interrupt 2 entspricht und eine entsprechenden Fehlermeldung für den Anwender auf dem Bildschirm ausgibt. Speichererweiterungskarten melden dem Computer über die Prüfleitung, wenn ein Paritätsfehler aufgetreten ist. Für den Anwender läßt sich mit diesen Fehlermeldungen schnell und sicher eine falsche Spei-

cherkodierung finden und beheben.

Um alle Steckkarten ausreichend mit Spannung versorgen zu können, muß das Netzteil des PC entsprechend dimensioniert sein. Die Stromversorgung für jede Steckkarte ist in der Regel so ausgelegt, daß pro Leitung zwei TTL-Lasteinheiten angesteuert werden können. Das bedeutet, daß jede Leitung eines Steckplatzes in der Lage ist, ohne Schaden zwei Gatter der TTL-Familie zu steuern. Diese Angabe ist besonders für eigene Entwicklungen von Bedeutung, um die DMA-Bausteine des Computers vor Beschädigungen zu schützen.

Besteht die Gefahr einer Leitungsüberlastung, so müssen Sie die anzusteuern Schaltung mit Treiberbausteinen ausstatten, die eine Überlastung der Bausteine im Computer verhindern. Achten Sie jedoch darauf, daß Sie hinreichend schnelle Treiberbausteine verwenden. So vermeiden Sie von vornherein Zeitprobleme mit den Taktsignalen. Die im Handel erhältlichen Zusatzkarten und Adapter für den PC belasten die Steckplätze gewöhnlich nur mit je einer TTL-Last, so daß keine Gefahr der Überlastung besteht.

(Alfred Kurz/ma)

Leistungsstarke Software für professionelle Anwender

AUFTRAGSBEARBEITUNG

Geschäftssoftwarepaket mit modulare Aufbau. Die einzelnen Module sind untereinander voll kompatibel und erlauben so den Aufbau einer individuell angepaßten Lösung. Durch die logisch aufgebaute Bedienung ist eine kurze Einarbeitungsphase gewährleistet. Ständig wiederkehrende Funktionen, z. B. Anreden, Mengeneinheiten, etc. werden durch Tabellen verwaltet.

Kurzreferenz:

- Anwendersystemdaten:
 - Anrede:tabellen
 - Mengeneinheiten
 - Kenntnisse für die Zugriffsberechtigung
 - Mehrwertsteuersätze/Zahlungsbedingungen
 - Firmendaten für Rechnungskopf etc.
 - Numerikreife für Rechnungen/Gutschriften

Kundenverwaltung:

- Stetige Kundennummer
- Umsatz, letztes Umsatzdatum, letztes Werbedatum
- Zahlungsbedingungen
- Stammdatenliste/Telefonliste/Umsatzliste
- Etikettendruck

Artikelverwaltung:

- 12stellige Artikelnummer (alphanum.)
- Bestand, Mengeneinheit, fakturierte Menge
- VK-Preis/MwSt.-Schlüssel
- Bestandslisten, Preislisten
- Etikettendruck

Fakturierung:

- Direktfakturierung
- automatische Vergabe von Rechnungs-/Gutschriftennr.
- Rechnungsausgangsjournal, Jahresabschluß
- automatische Übernahme neuer Artikel und Kunden
- Druckwiederholung/Bestandskontrolle
- Übersicht der erfaßten Posten

Preis: 398,-

Folgende Erweiterungsmodul sind lieferbar:

Stapelfakturierung	98,-
Warengruppenverwaltung	98,-
Mandantenfähigkeit	798,-
Lieferscheinbeschreibung	98,-
Lieferantenverwaltung	98,-
passende Fibu	498,-

Händleranfragen erwünscht!

Alle Preise zzgl. Versandkostenanteil.
Lieferung per Nachnahme oder Vorauskasse (2% Skonto)

GENERIC-CADD - Benutzerfreundliches u. leistungsfähiges CAD-Programm

Dieses Programm läßt sich sowohl mit der Tastatur als auch mit Maus, Digitalisieretafel oder Light-Pen steuern. Es bietet die geometrischen Grundfunktionen wie Gerade, Rechteck, regelmäßiges Viereck, Kreis, Ellipse, Kurve, Bogen, etc.

- 7 Zeichensätze sind vorgegeben, weitere können leicht definiert werden
- komfortable Editierfunktionen wie Verschieben, Ändern, Löschen, Trennen können auf die Grundelemente, Objekte, Ebene, Fenster, Zeichnung angewandt werden
- bis zu 256 Ebenen können einzeln editiert werden
- Zoom- und Panfunktion vorhanden
- Zeichnungsgröße nur vom Hauptspeicher abhängig
- Zeichnungsvorgang kann auch extern gesteuert werden
- Bemessungsfunktionen können abgefragt werden
- Eingabe in absoluten, relativen oder Polarkoordinaten möglich
- Bei Ausgabe der Zeichnungen auf Matrixdruckern wird eine sehr gute Auflösung erzielt

Preis: Generic-Cadd
DotPlot

398,-
99,-

EINNAHME-ÜBERSCHUSSRECHNUNG

Diese extrem schnelle und einfach zu bedienende Buchführung ist für alle diejenigen geeignet, welche nach § 4 (3) EStG nicht zur doppelten Buchführung verpflichtet sind. Das Programm ist durchgehend mit Plausibilitätsprüfungen versehen und liefert alle notwendigen Auswertungen und Informationen.

- jederzeit abrufbare ONLINE-Hilfefunktion
- der vorgegebene Kontenrahmen kann beliebig verändert werden
- auch Privatentnahmen/-umbuchungen etc. werden berücksichtigt
- beim Buchen werden Datum und Belegnummer automatisch vorgegeben (beliebig veränderbar!)
- Buchungen werden nach Datum und Belegnummern sortiert, d. h. auch nachträgliches Buchen ist möglich
- jederzeit klare Übersicht über alle betr. Finanzkonten
- sofortiger Kontendruck (einzeln oder Stapel), Umsatzsteuerabrechnung
- Überschuberechnung, Bestandsverzeichnis, Kostenstellenübersicht, Finanzübersicht
- BWA
- Abschluß nach beliebigen Perioden
- immer alle Buchungen des gesamten Jahres verfügbar

Preis: 298,-

Mandantenfähige Version für Steuerberater, etc.

648,-

Demo-Version

35,-

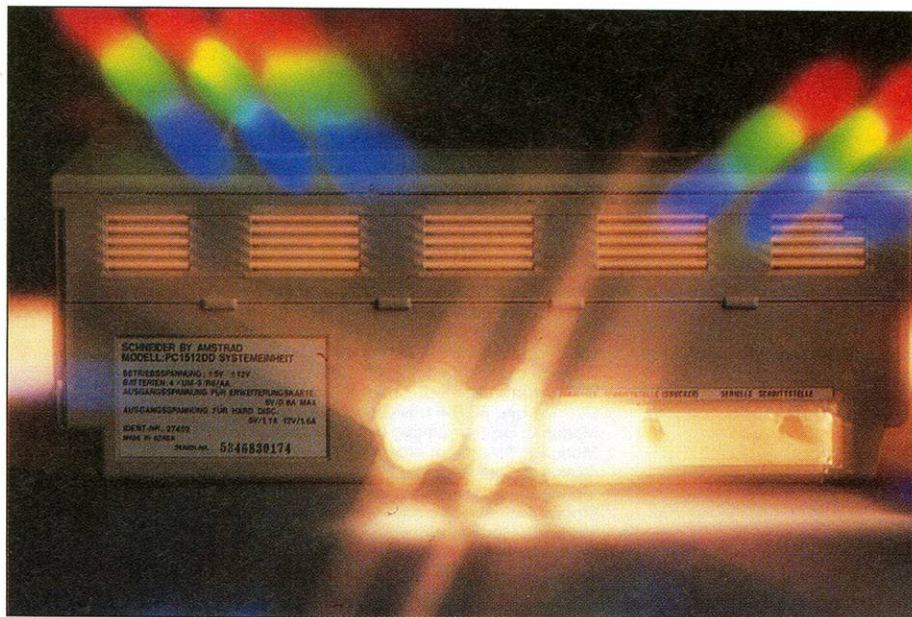
LOHN- UND EINKOMMENSSTEUERBERECHNUNG

Dieses Programm stellt eine große Hilfe für die Erstellung Ihrer Lohn- und Einkommenssteuererklärung dar. Die übersichtliche und klare Bedienungsführung macht lange Einarbeitungsphasen überflüssig. Das Programm ermittelt entsprechend den Steuervorschriften die Abzugsbeträge, das zu versteuernde Einkommen, die zu zahlende Einkommenssteuer sowie die voraussichtliche Rückerstattung bzw. Nachzahlung einschl. Kirchensteuer. Dabei werden Höchst-, Frei- und Pauschalbeträge automatisch berechnet. Es berücksichtigt ferner die besondere Vorsorgepauschale (für nicht rentenversicherungspflichtige Arbeitnehmer), Altersentlastung, Härteausgleich, Sparzulagen (vermögenswirksame Leistungen), außerordentliche Einkünfte (§ 34 EStG) und unter Progressionsvorbehalt stehende Einnahmen. Die Ergebnisse werden in Form einer Steuerübersicht komprimiert ausgedruckt; weiterhin werden Aufstellungen zu einzelnen Positionen erstellt. (Formulardruck in Vorbereitung!)

Vor Ablauf des Steuerjahres können durch Berechnen der tatsächlichen bzw. geschätzten steuerlichen Daten steuerliche und finanzielle Dispositionsmöglichkeiten aufgezeigt werden. Desweiteren werden ungenutzte Absetzungen für Versicherungen- und Bausparbeiträge bei den Sonderausgaben aufgezeigt. Auch die Berechnung des Kindergeldzuschlags nach § 11 BKGG ist in das Programm integriert!!! Der Updateservice garantiert die Anpassung an jeweilige Gesetzesänderungen.

Preis: 149,-
Mandantenfähige Version für Lohnsteuerhilfevereine, etc. 648,-
Demo-Version 35,-

TG-Soft · **Offersdorf 5** · **8491 Rimbach** · **Tel. 099 41 / 37 65**
Thomas Gnach



Die Portadressen des PC

In der Regel kennt sie nur der Assembler-Programmierer: die Portadressen im PC, über die sich die einzelnen Bausteine des Computers direkt programmieren lassen.

Die Hardware eines PC ist hauptsächlich aus den Bausteinen der 80xx-Serie von Intel aufgebaut, sofern nicht Gate Arrays die Funktionen einiger Bausteine übernehmen (zum Beispiel im Schneider PC). Tabelle 1 zeigt die Intel-Bausteine und andere VLSI-Bausteine (Very Large Scale Integration), die in einem PC zum Einsatz kommen.

Zusätzlich befinden sich auf der Hauptplatine eines PC diverse TTL-Bausteine der 74xx-Serie, die die Dekodierung von Portadressen, das Ver-

stärken und Zwischenspeichern von Daten und Adressen, sowie einige andere Aufgaben übernehmen. In der Regel werden die TTL-Bausteine in Tabelle 2 im PC verwendet.

Die einzelnen Bausteine des PC lassen sich über Portadressen ansprechen. Der 8086/8088-Prozessor erlaubt 400hex verschiedene Portadressen, von denen die ersten 200hex Adressen ursprünglich für die Bausteine auf der Hauptplatine vorgesehen und die Portadressen ab 200hex für die Elektronik der Erweiterungs-

karten reserviert waren. Durch die fortschreitende Integration in der Mikroelektronik befinden sich jedoch mittlerweile auf den Hauptplatinen moderner PCs auch Bausteine, die einst auf Erweiterungskarten steckten. Deshalb muß man die Trennung zwischen internen und externen Portadressen immer unter dem »historischen« Aspekt betrachten.

Die Erweiterungskarten für einen PC lassen sich in die fünf bis acht 62poligen 8-Bit-Steckplätze des Computers einstecken und erweitern die Hardware des PC beispielsweise um einen Festplatten-Controller, eine Schnittstellen-Karte oder eine Grafikkarte. Tabelle 3 listet alle Portadressen der Hauptplatine des PC auf und Tabelle 4 zeigt die reservierten Portadressen für Erweiterungskarten.

Ein Großteil von Tabelle 4 widmet sich den Portadressen des Monochromadapters und der CGA-Grafikkarte. Dies sind (noch) die gebräuchlichsten Grafikadapter bei PCs. Der Monochromadapter stellt 25 Zeilen zu je 80 Zeichen aus dem Zeichensatz des PC dar und arbeitet pro Zeichenposition mit einer Matrix von 9x14 Bildpunkten.

Die CGA-Farbgrafikkarte stellt wahlweise 25 Zeilen zu je 40 oder 80 Zeichen dar und arbeitet pro Zeichenposition mit einer Matrix von 8x8 Bildpunkten. Jeder Bildpunkt läßt sich auch einzeln ansprechen, so daß eine niedrige Grafikauflösung von 320x200 Bildpunkten und eine hohe Auflösung von 640x200 Bildpunkten zur Verfügung steht. In der niedrigen Auflösung läßt sich jeder Bildpunkt in 4 aus 16 Farben darstellen, während in der hohen Auflösung für jeden Bildpunkt nur die Auswahl von 2 aus 16 Farben möglich ist.

(ma)

Die hochintegrierten Bausteine im PC

Typ	Baustein	Aufgabe/Zuständigkeit
8048	Prozessor	Tastaturdekodierung
8086 oder 8088	Mikroprozessor	Zentraleinheit
8087	Mikroprozessor	Zentraleinheit
8237	Arithmetik-Prozessor	Mathematische Operationen
	DMA-Controller	Speicher-, Disketten- und Festplattenzugriff, Refresh
8253	Timer	Systemuhr, Refresh, Sound
8255	Parallel-Portbaustein	Tastatur, Konfigurationsschalter
8259	Interrupt-Controller	Interrupts 08 bis 0F
8284	Taktgenerator	Erzeugung der Takt- und Wartezyklen
8288	Bus-Controller	Dekodierung der Bussignale
6845	Video-Controller	Bildaufbau
0765	Floppy-Controller	Disketten-Controller

Tabelle 1. Ein knappes Dutzend komplexer Bausteine findet sich in einem konventionell aufgebauten PC

Die TTL-Bausteine im PC

Typ	Baustein	Aufgabe/Zuständigkeit
74LS138	Decoder	Portadressen
74LS158	Multiplexer	Speicheradressen
74LS244	8-Bit-Treiber (unidirektional)	Adreßtreiber
74LS245	8-Bit-Treiber (bidirektional)	Datentreiber
74LS322	Schieberegister	Tastaturcode
74LS373	8-Bit-Register	DMA-Adreßpuffer
74LS670	16-Bit-Register	DMA-Seitenregister (4xA16 bis A19)

Tabelle 2. Die TTL-Bausteine im PC erledigen auch anspruchsvolle Aufgaben

Die internen Portadressen des PC

Adresse Hauptplatine

DMA-Controller 8237

000-00F	Register des 4-Kanal-DMA-Controllers 8237
008	Steuerregister des 8237 (nur schreiben) Bit 0 = 1: Speicher-zu-Speicher-Übertragung freigeben Bit 1 = 1: Adresse für Kanal 0 merken Bit 2 = 1: DMA-Controller sperren Bit 3 = 1: Komprimiertes Timing verwenden Bit 4 = 1: Rotierende Priorität Bit 5 = 1: Erweiterte Schreibauswahl Bit 6: Bedeutung des DREQ-Signals Bit 7: Bedeutung des DACK-Signals
008	Statusregister des 8237 (nur lesen) Bit 0 = 1: Kanal 0 hat TC (Terminal Count) erreicht (Zählerüberlauf) Bit 1 = 1: Kanal 1 hat TC (Terminal Count) erreicht (Zählerüberlauf) Bit 2 = 1: Kanal 2 hat TC (Terminal Count) erreicht (Zählerüberlauf) Bit 3 = 1: Kanal 3 hat TC (Terminal Count) erreicht (Zählerüberlauf) Bit 4 = 1: Anfrage von Kanal 0 Bit 5 = 1: Anfrage von Kanal 1 Bit 6 = 1: Anfrage von Kanal 2 Bit 7 = 1: Anfrage von Kanal 3
009	Anforderungsregister des 8237 (nur Schreiben) Bit 1 = 0 und Bit 0 = 0: Anforderungsbit von Kanal 0 ausgewählt Bit 1 = 0 und Bit 0 = 1: Anforderungsbit von Kanal 1 ausgewählt Bit 1 = 1 und Bit 0 = 0: Anforderungsbit von Kanal 2 ausgewählt Bit 1 = 1 und Bit 0 = 1: Anforderungsbit von Kanal 3 ausgewählt Bit 2 = 0: Anforderungsbit löschen Bit 2 = 1: Anforderungsbit setzen Bit 3 bis Bit 7: Nicht verwendet
00A	Masken-Setz/Lösch-Register des 8237 (nur Schreiben) Bit 1 = 0 und Bit 0 = 0: Maskierungsbit von Kanal 0 ausgewählt Bit 1 = 0 und Bit 0 = 1: Maskierungsbit von Kanal 1 ausgewählt Bit 1 = 1 und Bit 0 = 0: Maskierungsbit von Kanal 2 ausgewählt Bit 1 = 1 und Bit 0 = 1: Maskierungsbit von Kanal 3 ausgewählt Bit 2 = 0: Maskierungsbit löschen Bit 2 = 1: Maskierungsbit setzen Bit 3 bis Bit 7: Nicht verwendet
00B	Modus-Auswahlregister des 8237 (nur Schreiben) Bit 1 = 0 und Bit 0 = 0: Kanal 0 ausgewählt Bit 1 = 0 und Bit 0 = 1: Kanal 1 ausgewählt Bit 1 = 1 und Bit 0 = 0: Kanal 2 ausgewählt Bit 1 = 1 und Bit 0 = 1: Kanal 3 ausgewählt Bit 3 = 0 und Bit 2 = 0: Überprüfungs-Übertragung (Verify) Bit 3 = 0 und Bit 2 = 1: Schreib-Übertragung Bit 3 = 1 und Bit 2 = 0: Lese-Übertragung Bit 3 = 1 und Bit 2 = 1: Nicht verwendet Bit 4 = 1: Auto-Initialisierung freigeben Bit 5 = 0: Adressen inkrementieren Bit 5 = 1: Adressen dekrementieren Bit 7 = 0 und Bit 6 = 0: Nachfragemodus ausgewählt Bit 7 = 0 und Bit 6 = 1: Einzelmodus ausgewählt Bit 7 = 1 und Bit 6 = 0: Blockmodus ausgewählt Bit 7 = 1 und Bit 6 = 1: Kaskadenmodus ausgewählt
00F	Masken-Schreib-Register des 8237 (nur Schreiben) Bit 0 = 0: Maskierungsbit von Kanal 0 löschen Bit 0 = 1: Maskierungsbit von Kanal 0 setzen Bit 1 = 0: Maskierungsbit von Kanal 1 löschen Bit 1 = 1: Maskierungsbit von Kanal 1 setzen Bit 2 = 0: Maskierungsbit von Kanal 2 löschen Bit 2 = 1: Maskierungsbit von Kanal 2 setzen Bit 3 = 0: Maskierungsbit von Kanal 3 löschen Bit 3 = 1: Maskierungsbit von Kanal 3 setzen Bit 4 bis Bit 7: Nicht verwendet
010-01F	Redundante Dekodierung des 8237

Interrupt-Controller 8259

020-021	Register des 8-Kanal-Interrupt-Controllers 8259 Kanal 0: Takt für Systemuhr Kanal 1: Unterbrechungs-Anforderung der Tastatur Kanal 2: Reserviert für EGA-Karte etc. Kanal 3: Zweite serielle Schnittstelle oder SDLC Kanal 4: Erste serielle Schnittstelle Kanal 5: Festplatten-Controller Kanal 6: Disketten-Controller Kanal 7: Parallele Schnittstelle
022-03F	Redundante Dekodierung des 8259

Timer-Baustein 8253

040-043	Registeradressen des 3-Kanal-Timer-Bausteins 8253
040	Register des Zählers 0 (Lesen und Schreiben): Systemuhr (Zeit und Datum)
041	Register des Zählers 1 (Lesen und Schreiben): DRAM-Refresh-Signal für DMA-Kanal 0
042	Register des Zählers 2 (Lesen und Schreiben): Rechteck-Signal für Lautsprecher und Kassettenrecorder
043	Steuerregister des 8253 (nur Schreiben) Bit 0 = 0: 16-Bit-Binärlöser ausgewählt Bit 0 = 1: BCD-Zähler ausgewählt

Adresse Hauptplatine

Timer-Baustein 8253

Bit 3 = 0, Bit 2 = 0 und Bit 1 = 0: Modus 0 ausgewählt Bit 3 = 0, Bit 2 = 0 und Bit 1 = 1: Modus 1 ausgewählt Bit 2 = 1 und Bit 1 = 0: Modus 2 ausgewählt Bit 2 = 1 und Bit 1 = 1: Modus 3 ausgewählt Bit 3 = 1, Bit 2 = 0 und Bit 1 = 0: Modus 4 ausgewählt Bit 3 = 1, Bit 2 = 0 und Bit 1 = 1: Modus 5 ausgewählt Bit 5 = 0 und Bit 4 = 0: Momentanen Zählerstand zum Lesen übernehmen Bit 5 = 0 und Bit 4 = 1: Nur LSB lesen/laden Bit 5 = 1 und Bit 4 = 0: Nur MSB lesen/laden Bit 5 = 1 und Bit 4 = 1: Erst LSB, dann MSB lesen/laden Bit 7 = 0 und Bit 6 = 0: Zähler 0 ausgewählt Bit 7 = 0 und Bit 6 = 1: Zähler 1 ausgewählt Bit 7 = 1 und Bit 6 = 0: Zähler 2 ausgewählt Bit 7 = 1 und Bit 6 = 1: Nicht verwendet Modus 0: Unterbrechung bei TC (Zählerüberlauf) Modus 1: Programmierbarer Pulsgenerator (Monoflop) Modus 2: Taktgenerator Modus 3: Rechtecksignal-Generator Modus 4: Software-gesteuerte Auslösung Modus 5: Hardware-gesteuerte Auslösung	
044-05F	Redundante Dekodierung des 8253

Parallel-Portbaustein 8255

060-063	Registeradressen des Parallel-Portbausteins 8255
060	Datenregister für Port A des 8255 (nur Lesen) Port A: 8 Bit Eingabe Bit 7 = 0: Byte von Tastatur einlesen Bit 7 = 1: Schalter SW1 (Konfiguration) einlesen
061	Datenregister für Port B des 8255 (nur Schreiben) Port B: 8 Bit Ausgabe Bit 0 = 1: Kanal 2 von 8253 generiert Rechtecksignal Bit 1 = 1: Wellenzüge für Lautsprecher generieren Bit 2 = 0: Bit 5 von SW2 über C0 lesen Bit 2 = 1: Untere 4 Bit von SW2 über C0 bis C3 lesen Bit 3 = 0: Recordermotor einschalten Bit 3 = 1: Recordermotor ausschalten Bit 4 = 0: Unterbrechungsanforderung (NMI) bei Parity-Fehler Bit 4 = 1: Kein Parity-Check Bit 5 = 0: Unterbrechungsanforderung (NMI) bei Fehler auf Erweiterungskarte Bit 5 = 1: Kein NMI bei Fehler auf Erweiterungskarte Bit 6 = 0: Tastaturtakt abschalten Bit 6 = 1: Tastaturtakt anschalten Bit 7: Siehe Kanal A
062	Datenregister für Port C des 8255 (nur Lesen) Port C: 8 Bit Eingabe Bit 0 bis 3: Anzahl der 32 KByte-Speicherbereiche über 256 KByte Bit 4: Datenbit vom Kassettenrecorder Bit 5: Ausgang von Kanal 2 des 8253 Bit 6 = 1: Unterbrechungsanforderung (NMI) wegen Fehler auf Erweiterungskarte Bit 7 = 1: Unterbrechungsanforderung (NMI) wegen Paritätsfehler
063	Steuerregister des 8255 (nur Schreiben) Bit 0 = 0: Kanal C (Bit 0 bis Bit 3) ist Ausgabekanal Bit 0 = 1: Kanal C (Bit 0 bis Bit 3) ist Eingabekanal Bit 1 = 0: Kanal B ist Ausgabekanal Bit 1 = 1: Kanal B ist Eingabekanal Bit 2 = 0: Modus 0 Bit 2 = 1: Modus 1 Bit 3 = 0: Kanal C (Bit 4 bis Bit 7) ist Ausgabekanal Bit 3 = 1: Kanal C (Bit 4 bis Bit 7) ist Eingabekanal Bit 4 = 0: Kanal A ist Ausgabekanal Bit 4 = 1: Kanal A ist Eingabekanal Bit 5 = 0 und Bit 6 = 0: Modus 0 Bit 5 = 1 und Bit 6 = 0: Modus 1 Bit 6 = 1: Modus 2 Bit 7 = 1 Modus 0: Unidirektionale Ein-/Ausgabe ohne Quittierungssignale (kein Handshake) Modus 1: Unidirektionale Ein-/Ausgabe mit Quittierungssignalen (Handshake-Betrieb) Modus 2: Bidirektionale Ein-/Ausgabe für Kanal A mit Quittierungssignalen
064-07F	Redundante Dekodierung des 8255

DMA-Seitenregister 74LS670

080-083	Adressen der vier DMA-Seitenregister 74LS670
081	DMA-Seitenregister für Kanal 2 (Disketten-Controller 765)
082	DMA-Seitenregister für Kanal 3 (Festplatten-Controller)
083	DMA-Seitenregister für Kanal 0 (DRAM-Refreshzyklus) und 1 (SDLC)
084-09F	Redundante Dekodierung des 74LS670
0A0-0AF	Bit 7: Statusbit zum Freigeben und Sperren des NMI-Interrupts
0B0-0BF	Redundante Dekodierung des NMI-Statusbit
0C0-0FF	Reserviert
100-1FF	Nicht verwendet (redundante Dekodierung)

Tabelle 3. Durch die unvollständige (redundante) Dekodierung auf der Hauptplatine, sind alle Bausteine auch durch um 10 hex und 100 hex höher liegende Portadressen ansprechbar.

Die externen Portadressen des PC

Adresse	Erweiterungskarten
200-20F	Joystick-Karte
201	Ausgaberegister (nur Schreiben) Das Schreiben eines beliebigen Wertes startet die vier Kippstufen
201	Eingaberegister für Joystickschalter (nur Lesen) Bit 0 = 1: Monostabile Kippstufe (AX) aktiv (Impulsdauer bestimmt die Joystick-Stellung) Bit 1 = 1: Monostabile Kippstufe (AY) aktiv (Impulsdauer bestimmt die Joystick-Stellung) Bit 2 = 1: Monostabile Kippstufe (BX) aktiv (Impulsdauer bestimmt die Joystick-Stellung) Bit 3 = 1: Monostabile Kippstufe (BY) aktiv (Impulsdauer bestimmt die Joystick-Stellung) Bit 4 = 0: Taste gedrückt (A1) Bit 5 = 0: Taste gedrückt (A2) Bit 6 = 0: Taste gedrückt (B1) Bit 7 = 0: Taste gedrückt (B2)
210-213	Bus-Erweiterungskarte
214-216	Empfängerkarte
220-24F	Reserviert
250-277	Nicht verwendet
278-27F	Dritte parallele Druckerschnittstelle (LPT3)
278	Datenregister für Druckerschnittstelle (Lesen und Schreiben)
279	Drucker-Statusregister (nur Lesen) Bit 0 = 0: Jumper 1 eingesetzt Bit 1 = 0: Jumper 2 eingesetzt Bit 2 = 0: Jumper 3 eingesetzt Bit 3 = 0: Druckerfehler aufgetreten Bit 4 = 1: Drucker ausgewählt Bit 5 = 1: Kein Papier vorhanden Bit 6 = 0: ACK-Signal aktiv Bit 7 = 0: Drucker beschäftigt
27A	Drucker-Steuerregister (Lesen und Schreiben) Bit 0 = 1: STROBE-Signal aktiv Bit 1 = 1: Automatischer Zeilenvorschub Bit 2 = 0: Druckerinitialisierung Bit 3 = 1: Druckerauswahl Bit 4 = 1: Unterbrechungsfreigabe durch ACK-Signal
280-2AF	Nicht verwendet
2B0-2DF	Zweiter EGA-Grafik-Controller
2E0-2EF	Nicht verwendet
2F0-2F7	Reserviert
2F8-2FF	Zweite serielle UART-Schnittstelle (COM2)
300-31F	Prototypenkarte (Entwicklungskarte)
320-32F	Festplatten-Controller
320	Datenregister des Festplatten-Controllers (Lesen und Schreiben)
321	Statusregister des Festplatten-Controllers (nur Lesen)
322	Steuerregister des Festplatten-Controllers (nur Schreiben)
330-377	Nicht verwendet
378-37F	Zweite parallele Druckerschnittstelle (LPT2)
378	Datenregister für Druckerschnittstelle (Lesen und Schreiben)
379	Drucker-Statusregister (nur Lesen) Bit 0 = 0: Jumper 1 eingesetzt Bit 1 = 0: Jumper 2 eingesetzt Bit 2 = 0: Jumper 3 eingesetzt Bit 3 = 0: Druckerfehler aufgetreten Bit 4 = 1: Drucker ausgewählt Bit 5 = 1: Kein Papier vorhanden Bit 6 = 0: ACK-Signal aktiv Bit 7 = 0: Drucker beschäftigt
37A	Drucker-Steuerregister (Lesen und Schreiben) Bit 0 = 1: STROBE-Signal aktiv Bit 1 = 1: Automatischer Zeilenvorschub Bit 2 = 0: Druckerinitialisierung Bit 3 = 1: Druckerauswahl Bit 4 = 1: Unterbrechungsfreigabe durch ACK-Signal
380-38F	Zweite serielle SDLC-Schnittstelle oder BSC-Controller
390-393	Cluster-Controller
394-39F	Nicht verwendet
3A0-3AF	Erste serielle SDLC-Schnittstelle oder BSC-Controller

Adresse	Erweiterungskarten
3B0-3BB	Monochromadapter
3B0	Nicht verwendet
3B1	Nicht verwendet
3B2	Nicht verwendet
3B3	Nicht verwendet
3B4	Adreßregister des Video-Controllers 6845 (nur Schreiben)
3B5	Über Adreßregister ausgewähltes Datenregister des 6845

Register	Bit	Voreinstellung	Funktion
00	0 bis 7	61	Zahl der (teilweise unsichtbaren) Textzeilen pro Textzeile -1 (für HSYNC-Signal)
01	0 bis 7	50	Zahl der tatsächlich angezeigten Textzeilen pro Textzeile
02	0 bis 7	52	Rasterzeilen-Position des HSYNC-Signals (bestimmt horizontale Verschiebung)
03	0 bis 3 4 bis 7	F 0	Länge des HSYNC-Signals Länge des VSYNC-Signals (0 = 16 Zeiteinheiten)
04	0 bis 6	19	Zahl der (teilweise unsichtbaren) Textzeilen -1
05	0 bis 4	06	Feineinstellung für Register 4 in Rasterzeilen
06	0 bis 6	19	Zahl der tatsächlich angezeigten Textzeilen
07	0 bis 6	19	Zeichenposition des VSYNC-Signals -1 (bestimmt senkrechte Verschiebung)
08	0 u. 1	2	Zeilensprungmodus (Interlace) ein (1 und 3) oder aus (0 und 2)
09	0 bis 4	0D	Höhe eines Zeichens in Rasterzeilen -1
0A	0 bis 3 5 u. 6	B 0	Erste Rasterzeile des Cursors (maximal 0C) Bit 6 = 0 und Bit 5 = 0: Kein Blinken Bit 6 = 0 und Bit 5 = 1: Kein Cursor Bit 6 = 1 und Bit 5 = 0: Blinken Bit 6 = 1 und Bit 5 = 1: Langsam blinkender Cursor
0B	0 bis 3	C	Letzte Rasterzeile des Cursors (maximal 0C)
0C	0 bis 5 6 u. 7 0 bis 7	- 0 -	Bit 8 bis Bit D der Startadresse des Bildschirmspeichers - Bit 0 bis Bit 7 der Startadresse des Bildschirmspeichers
0E	0 bis 5 6 u. 7 0 bis 7	- 0 -	Bit 8 bis Bit D der Adresse des Cursors (maximal 07CF) - Bit 0 bis Bit 7 der Adresse des Cursors (maximal 07CF)
10	0 bis 5 6 u. 7 0 bis 7	- 0 -	Bit 8 bis Bit D der Bildschirmadresse, bei der ein Lichtgriffelimpuls auftrat - Bit 0 bis Bit 7 der Bildschirmadresse, bei der ein Lichtgriffelimpuls auftrat

(Nur die Register 0C bis 11 können auch gelesen werden.)

3B6	Nicht verwendet
3B7	Nicht verwendet
3B8	Steuerregister des Video-Controllers (nur Schreiben) Bit 0 = 1: Hohe Auflösung Bit 3 = 1: Anzeige aktivieren Bit 5 = 1: Blinken aktivieren
3B9	Reserviert
3BA	Statusregister des Video-Controllers (nur Lesen) Bit 0 = 1: Horizontales Synchronisationssignal aktiv Bit 3 = 1: Schwarz/Weiß-Anzeige
3BB	Reserviert

3BC-3BF	Erste parallele Druckerschnittstelle (LPT1)
3BC	Datenregister für Druckerschnittstelle (Lesen und Schreiben)
3BD	Drucker-Statusregister (nur Lesen) Bit 0 = 0: Jumper 1 eingesetzt Bit 1 = 0: Jumper 2 eingesetzt Bit 2 = 0: Jumper 3 eingesetzt Bit 3 = 0: Druckerfehler aufgetreten Bit 4 = 1: Drucker ausgewählt Bit 5 = 1: Kein Papier vorhanden Bit 6 = 0: ACK-Signal aktiv Bit 7 = 0: Drucker beschäftigt
3BE	Drucker-Steuerregister (Lesen und Schreiben) Bit 0 = 1: STROBE-Signal aktiv Bit 1 = 1: Automatischer Zeilenvorschub Bit 2 = 0: Druckerinitialisierung Bit 3 = 1: Druckerauswahl Bit 4 = 1: Unterbrechungsfreigabe durch ACK-Signal
3BF	Nicht verwendet

3C0-3CF	Erster EGA-Grafik-Controller
---------	------------------------------

Adresse	Erweiterungskarten
3D0-3DF	CGA-Farbgrafikkarte
3D0	Nicht verwendet
3D1	Nicht verwendet
3D2	Nicht verwendet
3D3	Nicht verwendet
3D4	Adreßregister des Video-Controllers 6845 (nur Schreiben)
3D5	Über Adreßregister ausgewähltes Datenregister des 6845

Regi-ster	Bit	Voreinstellung Textmodus (80), Textmodus (40), Grafikmodus	Funktion
00	0 bis 7	38, 71, 38	Zahl der (teilweise unsichtbaren) Textzeichen pro Textzeile -1 (für HSYNC-Signal)
01	0 bis 7	28, 50, 28	Zahl der tatsächlich angezeigten Textzeichen pro Textzeile
02	0 bis 7	2D, 5A, 2D	Rasterzeilen-Position des HSYNC-Signals (bestimmt horizontale Verschiebung)
03	0 bis 3 4 bis 7	A, A, A 0, 0, 0	Länge des HSYNC-Signals Länge des VSYNC-Signals (0 = 16 Zeiteinheiten)
04	0 bis 6	1F, 1F, 7F	Zahl der (teilweise unsichtbaren) Textzeilen -1
05	0 bis 4	06, 06, 06	Feineinstellung für Register 4 in Rasterzeilen
06	0 bis 6	19, 19, 64	Zahl der tatsächlich angezeigten Textzeilen
07	0 bis 6	1C, 1C, 70	Zeichenposition des VSYNC-Signals -1 (bestimmt senkrechte Verschiebung)
08	0 u. 1	2, 2, 2	Zeilensprungmodus (Interlace) ein (1 und 3) oder aus (0 und 2)
09	0 bis 4	07, 07, 01	Höhe eines Zeichens in Rasterzeilen -1
0A	0 bis 3 5 u. 6	6, 6, 6 0, 0, 0	Erste Rasterzeile des Cursors (maximal 0C) Bit 6 = 0 und Bit 5 = 0: Kein Blinken Bit 6 = 0 und Bit 5 = 1: Kein Cursor Bit 6 = 1 und Bit 5 = 0: Blinken Bit 6 = 1 und Bit 5 = 1: Langsam blinkender Cursor
0B	0 bis 3	7, 7, 7	Letzte Rasterzeile des Cursors (maximal 0C)
0C	0 bis 5 6 u. 7	- 0, 0, 0	Bit 8 bis Bit D der Startadresse des Bildschirmspeichers
0D	0 bis 7	-	Bit 0 bis Bit 7 der Startadresse des Bildschirmspeichers
0E	0 bis 5 6 u. 7	- 0, 0, 0	Bit 8 bis Bit D der Adresse des Cursors (maximal 07CF)
0F	0 bis 7	-	Bit 0 bis Bit 7 der Adresse des Cursors (maximal 07CF)
10	0 bis 5 6 u. 7	- 0, 0, 0	Bit 8 bis Bit D der Bildschirmadresse, bei der ein Lichtgriffelimpuls auftrat
11	0 bis 7	-	Bit 0 bis Bit 7 der Bildschirmadresse, bei der ein Lichtgriffelimpuls auftrat

(Nur die Register 0C bis 11 können auch gelesen werden.)

3D6	Nicht verwendet
3D7	Nicht verwendet
3D8	Modus-Auswahlregister (nur Schreiben)
	Bit 0 = 0: 40-Zeichen-Modus
	Bit 0 = 1: 80-Zeichen-Modus
	Bit 1 = 0: Textmodus
	Bit 1 = 1: Grafikmodus
	Bit 2 = 0: Farbmodus
	Bit 2 = 1: Schwarz-Weiß-Modus
	Bit 3 = 1: Gib Bildschirmbild frei
	Bit 4 = 0: Wähle Auflösung 320x200 aus, wenn Bit 1 = 1
	Bit 4 = 1: Wähle Auflösung 640x200 aus, wenn Bit 1 = 1
	Bit 5 = 0: Verhindere blinkende Zeichen
	Bit 5 = 1: Erlaube blinkende Zeichen
	Bit 6: Nicht verwendet
	Bit 7: Nicht verwendet

Adresse	Erweiterungskarten		
3D9	Farb-Auswahlregister (nur Schreiben)		
Bit	Textmodus (40x25)	Grafikmodus (320x200)	Grafikmodus (640x200)
Bit 0 = 1	Rahmen blau	Rahmen und Hintergrund blau	Vordergrund blau
Bit 1 = 1	Rahmen grün	Rahmen und Hintergrund grün	Vordergrund grün
Bit 2 = 1	Rahmen rot	Rahmen und Hintergrund rot	Vordergrund rot
Bit 3 = 1	Rahmen hell	Rahmen und Hintergrund hell	Vordergrund hell
Bit 4 = 1	Hintergrundfarben statt Rahmen	Alternativer, heller Farbensatz	Alternativer, heller Farbensatz
Bit 5 = 0		Wähle Farbpalette 0 aus	
Bit 5 = 1		Wähle Farbpalette 1 aus	
Bit 6	-	-	-
Bit 7	-	-	-

3DA	Grafik-Statusregister (nur Lesen)
	Bit 0 = 1: Zugriff auf Regenerierungs-Pufferspeicher (Anzeige inaktiv)
	Bit 1 = 1: Lichtgriffel-Statusbit gesetzt (Impuls aufgetreten)
	Bit 2 = 1: Lichtgriffel ist ausgeschaltet
	Bit 3 = 1: Elektronenstrahl läuft an Ausgangspunkt (Geeigneter Zeitpunkt für Bildschirm-puffer-Zugriff)
	Bit 4 bis Bit 7: Nicht verwendet
3DB	Lichtgriffel-Register löschen
3DC	Lichtgriffel-Register setzen

Die folgenden drei Register gelten nur für den Schneider PC 1512 im Grafikmodus 2 (640x200 Bildpunkte in 16 Farben)

3DD	Farb-Schreib-Register (nur Schreiben)
	Bit 0 = 1: Speicherbereich (Bitplane) für Farbe Blau angesprochen
	Bit 1 = 1: Speicherbereich (Bitplane) für Farbe Grün angesprochen
	Bit 2 = 1: Speicherbereich (Bitplane) für Farbe Rot angesprochen
	Bit 3 = 1: Speicherbereich (Bitplane) für helle Darstellung angesprochen
	Bit 4 bis Bit 7: Nicht verwendet
3DE	Farb-Lese-Register (nur Schreiben)
	Bit 1 = 0 und Bit 0 = 0: Speicherbereich für Farbe Blau kann gelesen werden
	Bit 1 = 0 und Bit 0 = 1: Speicherbereich für Farbe Grün kann gelesen werden
	Bit 1 = 1 und Bit 0 = 0: Speicherbereich für Farbe Rot kann gelesen werden
	Bit 1 = 1 und Bit 0 = 1: Speicherbereich für helle Darstellung kann gelesen werden
	Bit 2 bis Bit 7: Nicht verwendet
3DF	Rahmen-Register (nur Schreiben)
	Bit 0 = 1: Blauanteil des Rahmens aktiviert
	Bit 1 = 1: Grünanteil des Rahmens aktiviert
	Bit 2 = 1: Rotanteil des Rahmens aktiviert
	Bit 3 = 1: Hellanteil des Rahmens aktiviert
	Bit 4 bis Bit 7: Nicht verwendet

3E0-3EF	Reserviert
---------	------------

3F0-3F7	Disketten-Controller
3F2	Laufwerk-Auswahlregister des Disketten-Controllers (nur Schreiben)
	Bit 1 = 0 und Bit 0 = 0: Laufwerk A
	Bit 1 = 0 und Bit 0 = 1: Laufwerk B
	Bit 1 = 1 und Bit 0 = 0: Laufwerk C
	Bit 1 = 1 und Bit 0 = 1: Laufwerk D
	Bit 2 = 1: Diskettensteuerung arbeitet
	Bit 3 = 1: Disketten-Interrupt wird weitergeleitet
	Bit 4 = 1: Motor von Laufwerk A läuft
	Bit 5 = 1: Motor von Laufwerk B läuft
	Bit 6 = 1: Motor von Laufwerk C läuft
	Bit 7 = 1: Motor von Laufwerk D läuft
3F4	Haupt-Statusregister des Disketten-Controllers 765 (nur Lesen)
	Bit 0 = 1: Laufwerk A ist aktiv
	Bit 1 = 1: Laufwerk B ist aktiv
	Bit 2 = 1: Laufwerk C ist aktiv
	Bit 3 = 1: Laufwerk D ist aktiv
	Bit 4 = 1: Diskettensteuerung arbeitet
	Bit 5 = 1: Diskettensteuerung befindet sich nicht im DMA-Modus
	Bit 6 = 0: Datenübertragung vom Prozessor zum Datenregister
	Bit 6 = 1: Datenübertragung vom Datenregister zum Prozessor
	Bit 7 = 1: Datenregister ist zum Senden/Empfangen von Daten bereit
3F5	Datenregister des Disketten-Controllers 765 (Lesen und Schreiben)

3F8-3FF	Erste serielle UART-Schnittstelle (COM1)
---------	--

Tabelle 4. Im Bereich von 200 bis 3FFhex sind Portadressen für Grafikkarten, Schnittstellen, Joysticks und andere Erweiterungen reserviert.

Anschluß für den Drucker:

Die Centronics-Schnittstelle

Die bekannteste parallele Schnittstelle ist zweifellos die Centronics-Schnittstelle zum Anschluß eines Druckers an den Computer. Doch wie funktioniert diese Schnittstelle, wie ist sie aufgebaut und wie läßt sie sich programmieren?

Für die Verbindung eines Computers mit seinen Peripheriegeräten haben sich abhängig von der Art der Daten und den notwendigen Übertragungsgeschwindigkeiten unterschiedliche Standards entwickelt. Für die Ansteuerung von Druckern setzte sich dabei die Schnittstelle des Druckerherstellers Centronics durch, die nach und nach auch von den anderen Druckerherstellern übernommen wurde.

Die Centronics-Schnittstelle arbeitet mit TTL-Pegeln (+ 5 Volt und 0 Volt). Die Länge der Leitung zwischen Computer und Drucker sollte nicht mehr als zwei Meter betragen. Bei Leitungslängen über 0,8 Meter hat sich die sogenannte Twisted-Pair-Verdrahtung durchgesetzt. Hier ist jede einzelne Signalleitung mit einer eigenen Masseleitung verdreht, woraus eine hohe Störsicherheit resultiert. Die Twisted-Pair-Verdrahtung wird schon durch die Steckerbelegung der Centronics-Schnittstelle unterstützt, denn hier ist jeder Datenleitung und den wichtigsten Steuerleitungen eine eigene Masseleitung zugeteilt. Die Signal- und Masseleitungen sind zudem so angeordnet, daß sie sich am Stecker beziehungsweise an der Buchse direkt gegenüberliegen.

Der Anschluß des Centronics-Kabels am Drucker erfolgt über einen Amphenol-Stecker (Bild 1). Dieser Stecker läßt sich mit zwei Haltebügeln an der Centronics-Buchse des Druckers befestigen, so daß sich eine mechanisch stabile Verbindung herstellen läßt, die auch größeren Zugbelastungen widersteht. Die Tabelle zeigt die Pinbelegung des Amphenol-Steckers. Die Signale mit einem Stern vor der Bezeichnung arbeiten mit negativer Logik, das heißt, sie liegen im aktiven Zustand auf Masse.

Das höchstwertige Datenbit 7 der Centronics-Schnittstelle wird von einigen Druckern als Dateninformation ignoriert, weil der einfache ASCII-Zeichensatz mit sieben Bit (Bit 0 bis 6) auskommt. Zum Teil benutzen diese Drucker das siebte Bit zu Steuerzwecken, um über diese Leitung Betriebsarten umzuschalten und ähnliches. Erzeugt der Computer diese Steuersignale nicht, so muß man das

Datenbit 7 auf Masse legen, weil es sonst zu Fehlfunktionen kommen kann. Bei vielen Druckern läßt sich das siebte Bit auch softwaremäßig ausschalten.

Die Signale der Centronics-Schnittstelle im Detail sind:

Data Strobe

Wenn der Computer Daten an den Drucker übergeben möchte, aktiviert er für 1 bis 5 Mikrosekunden diese Leitung als Zeichen für den Drucker, daß Daten zur Übergabe bereitstehen. Data Strobe liegt im aktiven Zustand auf Masse.

Data 0 bis Data 7

Über diese acht Datenleitungen liest der Drucker nach einem Data Strobe-Signal die anstehenden Daten ein.

Acknowledge

Sobald der Drucker die eingelesenen Daten verarbeitet hat, legt er auf die Acknowledge-Leitung einen Masseimpuls von maximal 10 Mikrosekunden Länge, um dem Computer mitzuteilen, daß er neue Daten übergeben kann.

Busy

Während der Drucker Daten einliest oder ausdruckt, ist das Busy-Signal aktiv und liegt auf + 5 Volt. Auch bei der Initialisierung, im Off-Line-Zustand und beim Auftreten eines Fehlers (siehe Error-Signal) ist das Busy-Signal gesetzt. Bild 2 zeigt das Zusammenspiel von Data Strobe, Acknowledge und Busy bei der Datenübertragung.

Paper Out

Dieses Signal zeigt an, daß das Papierende erreicht ist. Um den Druckvorgang fortzusetzen, muß neues Papier zugeführt werden.

Select

Über dieses Signal zeigt der Drucker dem Computer an, daß er ausgewählt und aktiv ist. Das Signal liegt bei vielen Druckern über einen Pull-up-Widerstand konstant auf + 5 Volt.

Autofeed

Über diesen Anschluß läßt sich der Zeilenvorschub steuern. Liegt dieses Signal auf Masse, so führt der Drucker zum Ende jeder Druckzeile einen automatischen Zeilenvorschub

aus. Liegt das Autofeed-Signal dagegen auf + 5 Volt, so erfolgt die Steuerung des Zeilenvorschubs über ein Programm, beziehungsweise den Druckertreiber.

No Connection

Diese Leitung ist offen und damit unbenutzt.

Signal Ground

Über diese Leitung liegt der Massepegel des Computersystems an. Der Drucker kann sich somit an das Massepotential des Computers anpassen.

Chassis Ground

Dieses Signal führt den Massepegel des Druckers. Es ist zu beachten, daß die Masse des Druckers und des Computers voneinander getrennt sind. Sie dürfen diese beiden Leitungen nicht miteinander verbinden.

+ 5 Volt

Über diese Leitung kann der Drucker vom Computer maximal 100 Milliampere Strom aufnehmen.

Ground

Dies sind die Massepegel zu den Datensignalen und einigen Steuersignalen.

Initial

Über diesen Ausgang kann der Computer den Drucker neu initialisieren, das heißt, in den Einschaltzustand zurückversetzen. Dazu muß das Signal für mindestens 50 bis 100 Mikrosekunden auf Masse liegen.

Error

Dieses Signal liegt auf Masse, sobald der Drucker Off-Line geschaltet, das Papier zu Ende, oder ein Fehler während des Druckvorganges (zum Beispiel ein falscher Steuercode) aufgetreten ist.

Clock

Über die Clock-Leitung kann der Computer dem Drucker einen Takt übermitteln, der den Rhythmus der Datenübergabe angibt.

Test

Mit der Testleitung läßt sich das Vorhandensein des Druckers überprüfen. Diese Leitung liegt im Drucker in der Regel über einen Pull-up-Widerstand konstant auf + 5 Volt.

Select Input

Über einen Massepegel auf der Select Input-Leitung erfolgt die Auswahl (Selektierung) des Druckers. Die bei-

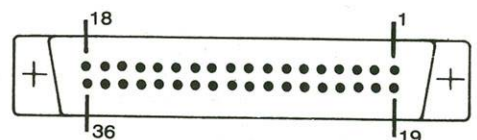


Bild 1: Der 36polige Amphenol-Stecker führt die Signale der Centronics-Schnittstelle an den Drucker

den Auswahlcodes Decice Control 1 (DC1) und Device Control 3 (DC3) funktionieren nur, wenn die Select-Leitung inaktiv ist, das heißt, auf + 5 Volt liegt. Im anderen Fall läßt sich der Drucker über DC1 und DC3 nicht an- oder abwählen.

Für einen funktionierenden Druckerbetrieb ist es nicht notwendig alle Schnittstellensignale zu erzeugen oder auszuwerten. Normalerweise reichen die sieben oder acht Datenleitungen, sowie das Data Strobe- und das Busy-Signal aus, um die Funktion der Centronics-Schnittstelle zu gewährleisten. Die übrigen Anschlüsse bleiben dann unbeschaltet.

Mit nur zwei Steuerleitungen läßt sich eine Centronics-Schnittstelle betreiben

Der Ablauf der Datenübertragung vom Computer zum Drucker sieht in solch einem Fall wie folgt aus: Zuerst liefert der Computer einen Data Strobe-Impuls an den Drucker. Sobald dieser Impuls beendet ist, übernimmt der Drucker die auf den Leitungen D0 bis D7 anliegenden Daten. Während er diese Daten einliest, ist das Busy-Signal aktiv. Hat der Drucker das Einlesen der Daten beendet, so ändert auch das Busy-Signal seinen Zustand auf inaktiv. Dies informiert den Computer darüber, daß der Drucker zur Übernahme von neuen Daten bereit ist. Also folgt der zweite Data Strobe-Impuls und der Drucker liest das zweite Datenbyte ein.

Während des Einlesens der Daten beginnt der Drucker bereits mit dem Ausdruck. Weil er aber nicht so schnell drucken wie einlesen kann, ist irgendwann der interne Datenpuffer des Druckers gefüllt. Jetzt kann der Drucker immer nur dann ein neues Datenbyte einlesen, wenn er ein Zeichen aus dem Druckpuffer ausgedruckt hat und damit ein Pufferbyte frei geworden ist.

Der PC kennt die drei parallelen Schnittstellen LPT 1 bis LPT 3. Die erste Schnittstelle LPT 1 läßt sich über die Portadressen 3BC bis 3BEhex, die zweite Schnittstelle LPT2 über die Portadressen 378 bis 37Ahex und die dritte Schnittstelle LPT3 über die Portadressen 278 bis 27Ahex erreichen.

Die erste Portadresse (278, 378 oder 3BC) gibt dabei jeweils das Datenregister der Druckerschnittstelle an, in das 8-Bit-Werte geschrieben und aus dem 8-Bit-Werte gelesen werden können. Die zweite Portadresse (279, 379 oder 3BD) bestimmt das Statusregister. Die Bits 3 bis 7 des Statusregi-

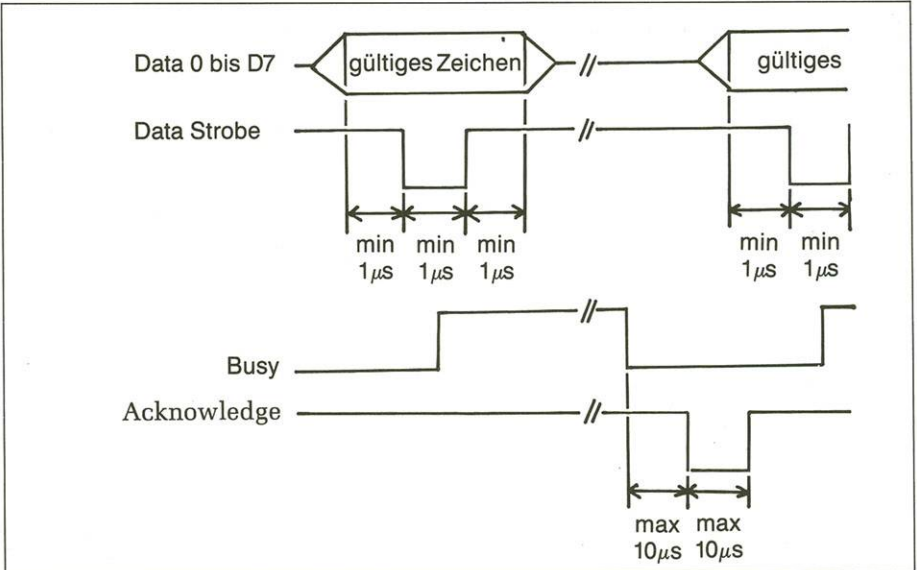


Bild 2: Das Zusammenspiel von Data Strobe, Busy und Acknowledge bei der Datenübertragung vom Computer an den Drucker

Die Signale der Centronics-Schnittstelle				
Pin	Masse	Computer	Signal	
1	19	Ausgang	*Data Strobe (STB)	
2	20	Ausgang	Data 0 (D0)	
3	21	Ausgang	Data 1 (D1)	
4	22	Ausgang	Data 2 (D2)	
5	23	Ausgang	Data 3 (D3)	
6	24	Ausgang	Data 4 (D4)	
7	25	Ausgang	Data 5 (D5)	
8	26	Ausgang	Data 6 (D6)	
9	27	Ausgang	Data 7 (D7)	
10	28	Eingang	*Acknowledge (ACK)	
11	29	Eingang	Busy	
12	30	Eingang	Paper Out	
13	-	Eingang	Select	
14	-	Ausgang	*Autofeed	
15	-	-	No Connection (NC)	
16	-	-	Signal Ground	
17	-	-	Chassis Ground	
18	-	Ausgang	+ 5 Volt	
19-30	-	-	Ground (GND)	
31	-	Ausgang	*Initial	
32	-	Eingang	*Error	
33	-	-	Ground (GND)	
34	-	Ausgang	Clock (CLK)	
35	-	Eingang	Test	
36	-	Ausgang	*Select Input	

Tabelle: Der normgerechte Aufbau der 36poligen Centronics-Schnittstelle

sters repräsentieren die Zustände einiger Signale der Centronics-Schnittstelle:

- Bit 3 = 0: Error (Fehler aufgetreten)
- Bit 4 = 1: Select (Drucker ausgewählt)
- Bit 5 = 1: Paper Out (kein Papier vorhanden)
- Bit 6 = 0: Acknowledge (fertig)
- Bit 7 = 0: Busy (beschäftigt)

Ein Programm, das das Statusregister regelmäßig ausliest, ist immer über den Betriebszustand der Centronics-Schnittstelle informiert.

Unter der dritten Portadresse der Druckerschnittstelle (27A, 37A oder 3BE) findet sich schließlich das Steuerregister, das wie das Datenregister

zum Lesen und Schreiben geeignet ist. Folgende Funktionen lassen sich mit dem Steuerregister beeinflussen:

- Bit 0 = 0: Data Strobe-Signal aktivieren
- Bit 1 = 0: Autofeed einschalten
- Bit 2 = 0: Drucker initialisieren
- Bit 3 = 1: Drucker auswählen
- Bit 4 = 1: Acknowledge bewirkt Interrupt 0F

Mit diesen Informationen dürfte es Ihnen nun leicht fallen, die Druckerschnittstellen des PC selbst zu programmieren, um sie beispielsweise zur Ansteuerung von selbstgebauten Schaltungen und anderen Hardware-Erweiterungen zu verwenden.

(Alfred Kurz/ma)

Disketten unter Kontrolle

Bei der Elektronik des Disketten-Controllers und Diskettenlaufwerks fehlt oft auch den fortgeschrittenen PC-Anwendern der Durchblick, weil die nötigen technischen Informationen fehlen. Dabei ist alles ganz einfach.

Disketten sind nach wie vor das wichtigste Speichermedium für den PC, auch wenn die Festplatte und das CD-ROM dem ehrwürdigen Datenträger Konkurrenz machen. Durch die Einführung der robusten 3 1/2-Zoll-Disketten, die 720 KByte oder 1,44 MByte an Daten aufnehmen können, ist die Attraktivität dieses Speichermediums sogar noch aufgewertet worden.

Wenn man sich vorstellt, daß sich auf den handlichen Datenscheiben bis weit über eine Million Zeichen speichern lassen, kann man erahnen, welche aufwendige Mechanik und Elektronik dahintersteckt. Um die Mechanik sollen Sie sich aber keine Gedanken machen, denn hier gibt es nichts zu basteln oder zu programmieren. Ganz im Gegenteil! Schon ein kleiner Eingriff in die Mechanik eines Diskettenlaufwerks kann wichtige Justierungen verstellen, so daß das Laufwerk hinterher nicht mehr korrekt arbeitet.

Anders dagegen bei der Elektronik. Hier lassen sich die Bausteine nach Herzenslust programmieren und manipulieren. Allerdings sollten Sie für Ihre ersten Versuche nur Disketten mit nicht mehr benötigten Daten verwenden – oder zumindest vorher Sicherheitskopien herstellen.

Doch bevor Sie sich nun in die Elektronik von Disketten-Controller und Laufwerk stürzen, eine kurze Definition für die Bezeichnung Disketten-Controller: Der Disketten-Controller ist eine elektronische Schaltung, die ein Interface (Verbindungsglied) zwischen Diskettenlaufwerk und dem Computer darstellt. Bild 1 zeigt als Blockschaltbild, wie der Disketten-Controller und das Diskettenlaufwerk zusammenarbeiten. Auf den ersten Blick ist die Darstellung vielleicht etwas verwirrend, aber am Ende dieses Beitrags wird für Sie alles glasklar sein.

Der Controller-Teil des Blockschaltbildes besteht am linken Bildrand beginnend, zunächst einmal aus dem Systembus des PC, der Adreß-, Daten- und Steuerleitungen vereint. Diese Leitungen werden über einen Puffer an den Controller-Baustein UPD 765 und das Laufwerk-Auswahlregister geführt. Das Laufwerk-Aus-

wahlregister spricht über einen zwischengeschalteten Dekoder eines von vier Diskettenlaufwerken an, während der Controller-Baustein über vier Steuerleitungen (Ausgänge) und drei Statusleitungen (Eingänge) direkt mit dem Diskettenlaufwerk verbunden ist.

Im (seriellen) Daten-Schreibkanal des Disketten-Controllers ist zwischen Controller-Baustein und Diskettenlaufwerk zusätzlich eine Schreibschaltung eingefügt und im (seriellen) Daten-Lese Kanal befindet sich entsprechend eine Leseschaltung. Beide Schaltungen sind auch mit einem Takt- und Zeitgeber verbunden, der für das richtige Timing bei der seriellen Datenübertragung sorgt.

Beim Schreiben von Daten auf Diskette erzeugt die Schreibschaltung aus den zu schreibenden Daten (Out) und einem Taktsignal WCLK das endgültige Schreibsignal für den Schreib-/Lesekopf des Diskettenlaufwerks. Die Leseschaltung stellt dann beim Einlesen der Signale vom Schreib-/Lesekopf mit Hilfe des Taktsignals RCLK wieder die ursprünglichen Daten (Inp) her, so daß der Controller-Baustein exakt die Daten erhält, die er einst ausgelesen hat.

Auf der Laufwerksseite des Blockschaltbildes befindet sich die Diskette mit Laufwerksmechanik (als Zylinder dargestellt). Für die Pegelanpassung der Signale vom und zum Dis-

ketten-Controller sind Schreib- und Leseverstärker für die Schreib-/Lesekopf-Signale und zwei Verstärker für die Dekodersignale zum Einschalten des Laufwerksmotors und zum Auswählen des Laufwerks vorhanden. Die Schreib- und Leseverstärker sind besonders wichtig, weil der Disketten-Controller mit TTL-Pegeln (+ 5 Volt und 0 Volt) arbeitet, während die Signale des Schreib-/Lesekopfes schwankende Amplituden haben und geringere Spannungspegel aufweisen.

Die Programmierung des Disketten-Controllers erfolgt wie bei den anderen Bausteinen des PC auch über die Leitungen des Adreß-, Daten- und Steuerbusses. Zuerst muß eine Einheit des Disketten-Controllers über eine Portadresse angesprochen werden, dann zeigen die Signale des Steuerbusses an, ob Daten gelesen oder geschrieben werden sollen. Über den Datenbus schließlich werden die Daten übertragen.

Zum Ansprechen durch eine Portadresse stehen auf dem Disketten-Controller das Laufwerk-Auswahlregister unter der Portadresse 3F2 und der Controller-Baustein unter den beiden Portadressen 3F4 und 3F5 zur Verfügung.

Das Laufwerk-Auswahlregister besteht aus einem 8-Bit-Register und läßt sich nur beschreiben. Wie der Name schon sagt, wählt das Lauf-

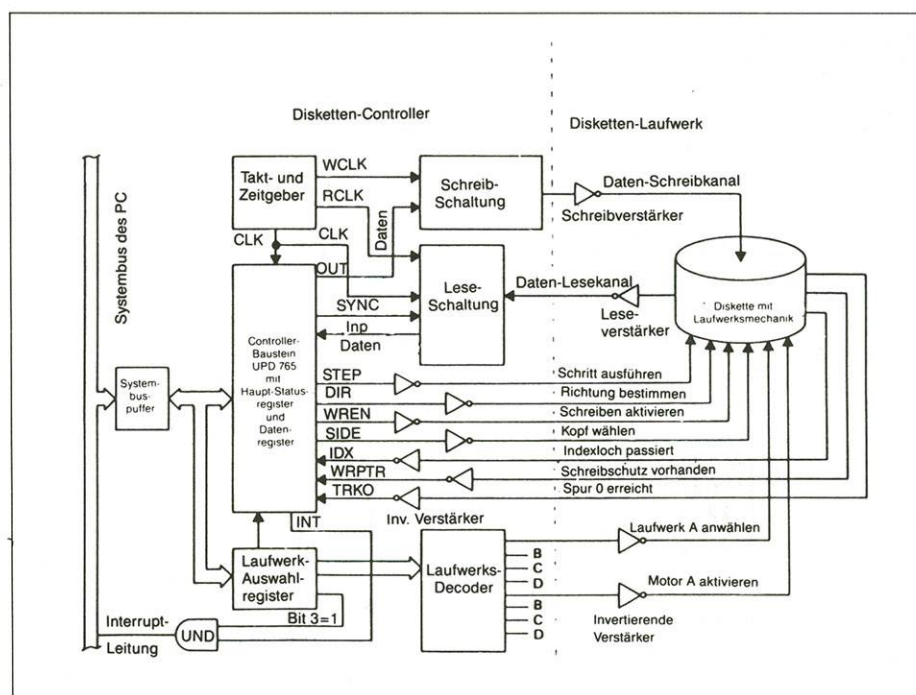


Bild 1. Das Blockschaltbild zeigt den Aufbau eines Disketten-Controllers mit Diskettenlaufwerk

Die Bits des Laufwerk-Auswahlregisters	
Bit	Funktion
Bit 1 = 0 und Bit 0 = 0	Laufwerk A ausgewählt
Bit 1 = 0 und Bit 0 = 1	Laufwerk B ausgewählt
Bit 1 = 1 und Bit 0 = 0	Laufwerk C ausgewählt
Bit 1 = 1 und Bit 0 = 1	Laufwerk D ausgewählt
Bit 2 = 0	Diskettensteuerung zurückgesetzt
Bit 2 = 1	Diskettensteuerung aktiviert
Bit 3 = 0	Software-Interrupt 0E durch Disketten-Controller gesperrt
Bit 3 = 1	Software-Interrupt 0E durch Disketten-Controller freigegeben
Bit 4 = 1	Motor von Laufwerk A arbeitet, Laufwerk kann ausgewählt werden
Bit 5 = 1	Motor von Laufwerk B arbeitet, Laufwerk kann ausgewählt werden
Bit 6 = 1	Motor von Laufwerk C arbeitet, Laufwerk kann ausgewählt werden
Bit 7 = 1	Motor von Laufwerk D arbeitet, Laufwerk kann ausgewählt werden

Tabelle 1. Über das Laufwerk-Auswahlregister wird eines von vier Laufwerken aktiviert

werk-Auswahlregister eines von maximal vier Laufwerken aus. Die Bits 0 und 1 definieren, welches Laufwerk ausgewählt ist, und die Bits 4 bis 7 bestimmen, welche Laufwerksmotoren eingeschaltet sind. Die restlichen Bits 2 und 3 sind Steuersignale. Tabelle 1 zeigt die Bedeutung der Bits des Laufwerk-Auswahlregisters im Detail.

Im Controller-Baustein läßt sich über die Portadresse 3F4 das Haupt-Statusregister und über die Portadresse 3F5 das 8-Bit-Datenregister ansprechen. Der Inhalt des Haupt-Statusregisters kann nur ausgelesen werden und gibt über aktive Laufwerke und Betriebsmodi Auskunft. Tabelle 2 zeigt im einzelnen die Bedeutung der Bits des Haupt-Statusregisters.

Das Datenregister des Controller-Bausteins besteht eigentlich aus mehreren Registern, die einen Registerstapel bilden und von denen jeweils nur ein Register am Datenbus des PC anliegt. In den Datenregisterstapel lassen sich Daten, Befehle und Parameter schreiben. Das Auslesen dieses Registerstapels ist auch möglich.

Dann liefern die Register die Daten, die von der Diskette gelesen wurden, oder Statusinformationen über die angeschlossenen Diskettenlaufwerke. Das Blockschaltbild in Bild 1 zeigt deutlich, wie das Laufwerk-Auswahlregister einen Dekoder ansteuert, der wiederum das gewünschte Laufwerk (A, B, C oder D) anspricht. Aus Platzgründen ist hier nur das Laufwerk A dargestellt.

Das Interruptsignal für die Interruptanforderung des Disketten-Controllers wird über eine UND-Verknüpfung aus Bit 3 des Laufwerk-Auswahlregisters und dem INT-Ausgang des Controller-Bausteins gebildet.

Vom Controller-Baustein führen vier Steuersignale direkt zum Diskettenlaufwerk und drei Statusleitungen vom Laufwerk zum Controller zurück. Die Signalpegel der Controller- und Laufwerkssignale werden über invertierende Verstärker (dargestellt als Dreiecke mit kleinen Kreisen an der Spitze) aneinander angepaßt. Invertierend bedeutet, daß die logischen Pegel der Signale vertauscht werden, das heißt aus logisch 0 wird logisch 1, und aus logisch 1 wird logisch 0. Die Invertierung ist für das korrekte Zusammenspiel von Controller- und Laufwerkssignalen erforderlich, weil die Laufwerkssignale ausschließlich mit negativer Logik arbeiten.

Die vier Signale, die vom Controller zum Laufwerk führen, sind STEP, DIR, WREN und SIDE. Die Bedeutung dieser Signale ist mit einigen Englischkenntnissen bereits dem Namen zu entnehmen.

Das STEP-Signal bewegt den Schreib-/Lesekopf des Diskettenlaufwerks um einen Schritt (eine Spur) weiter. Jeder Impuls auf dieser Leitung bewirkt einen einzelnen Schritt. Damit das Diskettenlaufwerk auch erfährt, ob der Schreib-/Lesekopf nach innen oder außen bewegt werden soll, gibt das DIR-Signal (direction) die Richtung der Kopfbewegung an. Ist das DIR-Signal gesetzt, so bewegt sich der Kopf in Richtung Diskettenmitte, und liegt die DIR-Leitung auf 0, so wandert der Schreib-/Lesekopf zum Diskettenrand.

Über das Signal WREN (write enable) wird das Schreiben von Daten auf Diskette freigegeben. Das SIDE-Signal gibt schließlich die Nummer des Schreib-/Lesekopfes an, der die Daten schreiben oder lesen soll. Liegt die SIDE-Leitung auf 0, so ist der Kopf 0 angesprochen, ist das SIDE-Signal auf 1 gesetzt, so ist entsprechend Schreib-/Lesekopf 1 an der Reihe. Sie sehen schon, die ganze Geschichte ist gar nicht so kompliziert, wie es zuerst den Anschein hatte.

Die drei Leitungen, die vom Laufwerk zum Controller führen, sind die Signale IDX, WRPRT und TRK0. Hier ist die Identifizierung der Signalbe-

deutung schon etwas schwieriger, weil die Abkürzungen kryptischer geworden sind.

Über das IDX-Signal zeigt das Diskettenlaufwerk an, daß das Indexloch der Diskette die interne Index-Lichtschranke passiert hat. Durch das Indexloch wird die Lichtschranke nämlich kurz freigegeben und liefert dadurch einen Impuls über die IDX-Leitung an den Disketten-Controller.

Durch die Leitung WRPRT (write protected) teilt das Diskettenlaufwerk dem Controller mit, daß die aktuell im Laufwerk befindliche Diskette schreibgeschützt ist. Jede Diskette besitzt oben rechts eine Kerbe, die sich mit einem Aufkleber verschließen läßt. Dort, wo sich an der Diskette die Kerbe befindet, sitzt im Diskettenlaufwerk eine Schreibschutz-Lichtschranke. Verdeckt nun ein Aufkleber diese Kerbe, so ist die Schreibschutz-Lichtschranke im Laufwerk gesperrt, was über das WRPRT-Signal angezeigt wird.

Das dritte und letzte Signal lautet TRK0 (track 0). Über die TRK0-Leitung bedeutet das Diskettenlaufwerk dem Controller, daß sich der Schreib-/Lesekopf auf Spur 0 der Diskette befindet. Hier muß sich das Inhaltsverzeichnis der Diskette und bei Systemdisketten auch der Bootsektor für das Betriebssystem befinden. Die Spur 0 ist deshalb für den Disketten-Controller von besonderer Bedeutung.

Wie Sie dem Blockschaltbild in Bild 1 bereits entnehmen konnten, ist in den Daten-Schreib- und -Lesekanal je eine Schreib- beziehungsweise Leseschaltung eingefügt. Diese beiden Schaltungen sind erforderlich, weil auf einer Diskette nicht die reinen Daten des Controllers aufgezeichnet werden können. Vielmehr ist noch ein spezieller Takt notwendig, den beim Disketten-Controller ein Takt- und Zeitgeber liefert.

Im Blockschaltbild ist der Takt zum Schreiben der Daten durch das Signal WCLK gekennzeichnet und der Takt zum Lesen der Daten durch das Signal RCLK. Zusätzlich benötigt die Leseschaltung noch den Originaltakt CLK, mit dem auch der Controller-Baustein arbeitet. Über die SYNC-

Die Bits des Haupt-Statusregisters	
Bit	Funktion
Bit 0 = 1	Der Schreib-/Lesekopf von Laufwerk A wird positioniert
Bit 1 = 1	Der Schreib-/Lesekopf von Laufwerk B wird positioniert
Bit 2 = 1	Der Schreib-/Lesekopf von Laufwerk C wird positioniert
Bit 3 = 1	Der Schreib-/Lesekopf von Laufwerk D wird positioniert
Bit 4 = 1	Ein Schreib- oder Lesebefehl wird verarbeitet
Bit 5 = 1	Die Diskettensteuerung befindet sich nicht im DMA-Modus
Bit 6 = 0	Datenübertragung vom Prozessor zum Datenregister des Disketten-Controllers
Bit 6 = 1	Datenübertragung vom Datenregister des Disketten-Controllers zum Prozessor
Bit 7 = 1	Datenregister zum Senden/Empfangen von Daten bereit

Tabelle 2. Das Haupt-Statusregister zeigt die Betriebsarten des Disketten-Controllers an

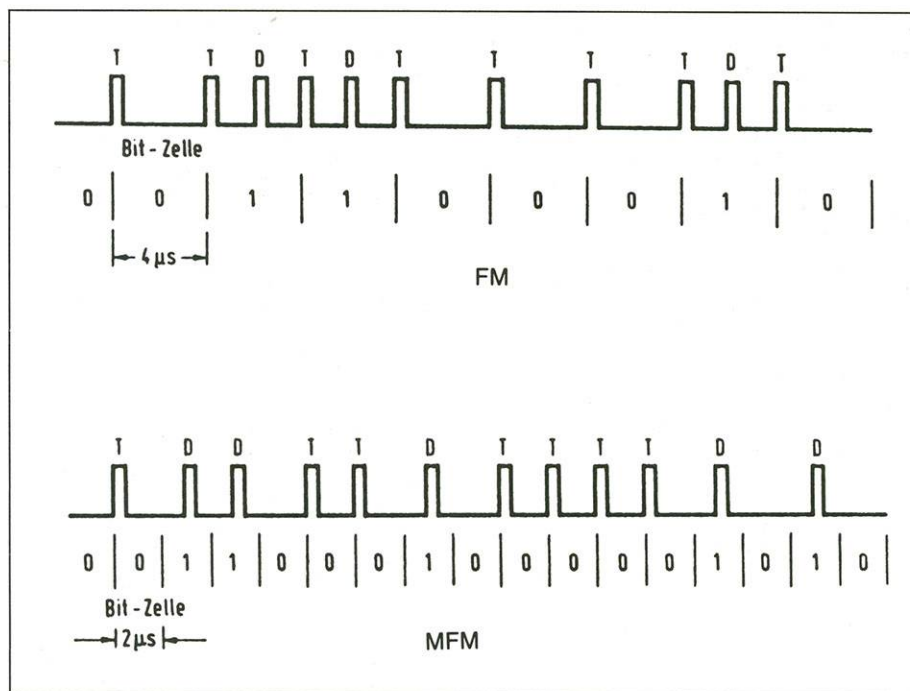


Bild 2. Der Vergleich zwischen dem FM- und MFM-Aufzeichnungsformat zeigt die Überlegenheit des MFM-Verfahrens

Leitung synchronisiert der Controller-Baustein die Datenübertragung mit der Leseschaltung.

Der Aufbau der Schreib- und Leseschaltung hängt vom verwendeten Verfahren für die Aufzeichnung der Daten auf Diskette ab. Die bekanntesten Aufzeichnungsarten sind das FM-Verfahren (Frequenzmodulation) und das MFM-Verfahren (modifizierte Frequenzmodulation). Während das FM-Verfahren nur bei der Datenaufzeichnung auf 8-Zoll-Disketten eine Rolle spielt, hat sich das MFM-Verfahren inzwischen für 5 1/4- und 3 1/2-Zoll-Disketten weltweit durchgesetzt.

Die modifizierte Frequenzmodulation MFM arbeitet gegenüber dem FM-Verfahren mit einer größeren Genauigkeit, so daß sich die Speicherkapazität einer Diskette mit diesem Verfahren verdoppeln läßt.

Jede Bit-Information auf der Diskette besitzt beim MFM-Verfahren eine Länge von nur 2 Mikrosekunden. Besitzt ein Bit den Wert 1, so erfolgt die Aufzeichnung eines Impulses in der physikalischen Mitte der Bitzelle. Besitzt das Bit dagegen den Wert 0, dann erfolgt die Aufzeichnung des Impulses am Anfang der Bitzelle. Dies geschieht jedoch nur, wenn das vorherige Bit nicht den Wert 1 hatte. Besteht eine Impulsfolge nur aus 0-Werten, so entspricht dies dem Aufzeichnungs-Taktsignal T.

Zur Rückgewinnung der Daten aus dem auf der Diskette aufgezeichneten Signal kann beim MFM-Verfahren nicht mehr wie beim FM-Verfahren ein einfaches Monoflop benutzt wer-

den. Vielmehr wird eine PLL-Schaltung (Phase-Locked-Loop) benötigt. Die Daten-Rückgewinnung über eine PLL-Schaltung ist zwar zeitkritischer als die Methode bei der FM-Aufzeichnung, doch bei der heutigen Präzision der Disketten und Laufwerke bleiben die Rotationsschwankungen innerhalb der von der PLL-Schaltung geforderten Grenzwerte. Daher setzt man mittlerweile fast ausschließlich das MFM-Verfahren ein. Bild 2 zeigt den Unterschied in der Aufzeichnungsart und Datendichte zwischen dem alten FM- und dem modernen MFM-Verfahren. Das T gibt jeweils das Taktsignal an.

Um ein Diskettenlaufwerk an den Disketten-Controller eines PC an-

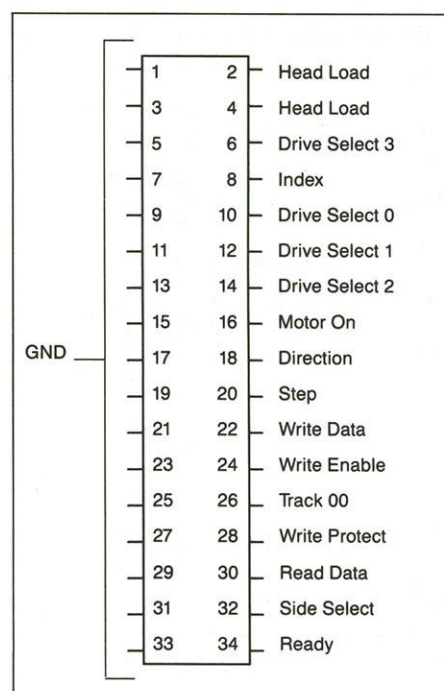


Bild 3. Die Pinbelegung am 34poligen Stecker des Disketten-Controllers

schließen zu können, haben sich die Controller- und Laufwerkhersteller auf einen einheitlichen Standard geeinigt. An jedem Diskettenlaufwerk befindet sich eine 34polige Steckleiste in Form eines Platinensteckers. Der Disketten-Controller wird nun über ein 34adriges Flachbandkabel mit der passenden Buchse angeschlossen. Bild 3 zeigt die Belegung des Steckers und Tabelle 3 gibt die Bedeutung aller Anschlüsse aus der Sicht des Controllers an. Sämtliche Signale arbeiten aufgrund der invertierenden Verstärker mit negativer Logik, das heißt ein Massepegel kennzeichnet den aktiven Zustand dieser Signale.

(Alfred Kurz/ma)

Die Signale des Controller-Steckers		
Signal	A = Ausgang E = Eingang	Bedeutung
Head Load	A	Schreib-/Lesekopf auf Diskette absenken
Drive Select	A	Auswahl eines angeschlossenen Laufwerks
Index	E	Indexloch hat Lichtschranke passiert
Motor On	A	Laufwerksmotor einschalten
Direction	A	Richtungsangabe für Step-Signal
Step	A	Schreib-/Lesekopf eine Spur weiterbewegen
Write Data	A	Daten für Diskette
Write Enable	A	Schreiben auf Diskette freigeben
Track 00	E	Lesekopf hat Spur 0 erreicht
Write Protect	E	Diskette schreibgeschützt
Read Data	E	Daten von Diskette
Side Select	A	Auswahl der Diskettenseite
Ready	E	Laufwerk ist bereit

Tabelle 3. Die Signale des Controller-Steckers arbeiten ausschließlich mit negativer Logik

Bit für Bit:

Die serielle Schnittstelle RS 232C

Die serielle Schnittstelle macht dem Anwender die weite Welt der Datenfernübertragung per Computer zugänglich. Doch für viele PC-Besitzer ist diese Schnittstelle immer noch ein Buch mit sieben Siegeln.

Die serielle Schnittstelle RS 232C ist heute die bekannteste und am weitesten verbreitete Schnittstelle für Mikrocomputer, obwohl es inzwischen weitaus leistungsfähigere Versionen (zum Beispiel RS 422 und RS 423) gibt.

Die RS 232C wurde ursprünglich für die Datenübertragung zwischen einem Hauptrechner und daran angeschlossenen Terminals konzipiert. Schnell kamen jedoch neue Anwendungsgebiete hinzu, wie zum Beispiel die Kommunikation zwischen Computern (Datenendgeräten) und Modems oder Akustikkopplern (Datenübertragungsgeräte).

Für einen korrekten Send- und Empfangsbetrieb besitzt die RS 232C einen Kanal zum Senden und Empfangen von Daten sowie einen Rückkanal (Hilfskanal). Im einzelnen bestehen die Signale der RS 232C aus Send- und Empfangsleitungen, Taktleitungen, Status- und einigen Steuersignalen. Die Tabelle zeigt die nach der EIA-Norm (Electronic Industries Association) festgelegten Signale der RS 232C-Schnittstelle.

Von dieser Vielzahl an Signalen werden jedoch im Computerbetrieb gewöhnlich nur neun Leitungen benutzt (TxD, RxD, RTS, CTS, DSR, DTR, DCD, GND und RI), so daß die RS 232C-Schnittstelle oft auch als 9polige Schnittstelle ausgeführt ist. Die vollausgebaute RS 232C-Schnittstelle arbeitet dagegen mit 25-poligen D-Subminiaturbuchsen und -steckern.

Die RS 232C-Schnittstelle ist für Übertragungsgeschwindigkeiten von bis zu 38400 Baud (Bit pro Sekunde) ausgelegt. Dabei dürfen die Übertragungsleitungen maximal 15 bis 20 Meter lang sein. Die Standardübertragungsraten für die RS 232C sind 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200 und 38400 Baud. Die gebräuchlichsten Datenübertragungsraten sind 300 Baud für den Betrieb von Akustikkopplern und 1200 Baud für die weniger stör anfälligeren Modems. Einige Standleitungen arbeiten mit Datenübertragungsraten von bis zu 9600 Baud.

Die Definition der Signalpegel für eine serielle Schnittstelle unterscheidet sich von den üblichen Logikpegeln einer Computerschaltung. Führt eine Leitung der RS 232C-Schnittstelle eine Spannung von weniger als -3 Volt, so entspricht dies dem logischen Zustand 1. Liegt der Signalpegel dagegen über +3 Volt, so gilt der logische Zustand 0. Sobald sich ein Signal zwischen der Spannung von -3 und +3 Volt befindet, ist es nicht mehr definiert und wird ignoriert.

Die maximale Spannung eines Signals darf bei der seriellen Übertragung +/-15 Volt betragen. Damit steht dem Anwender ein Spannungsbereich von insgesamt 30 Volt zur Verfügung. Die Zuordnung der logischen Pegel zu den Spannungspegeln

der Status- und Steuerleitungen ist umgekehrt. Eine negative Spannung entspricht hier dem logischen Zustand 0 und eine positive Spannung dem Zustand 1.

Im folgenden finden Sie einen kleinen Überblick zu dem Zusammenspiel der wichtigsten Signale einer RS 232C-Schnittstelle. Das Datenendgerät repräsentiert dabei in der Regel der PC, während Modem oder Akustikkoppler das Datenübertragungsgerät darstellen:

Über die Leitung TxD laufen die Sendedaten vom Datenendgerät zum Datenübertragungsgerät und über die Leitung RxD die Empfangsdaten vom Datenübertragungsgerät zum Datenendgerät. Im Ruhezustand liegen beide Signale auf logisch 1, das heißt -3 bis -15 Volt.

Die Datenübertragung ist nur dann möglich, wenn die beiden Statussignale DSR und DTR auf 1 liegen. Mit der DTR-Leitung zeigt das Datenendgerät dem Datenübertragungsgerät an, daß es betriebsbereit ist. Das Datenübertragungsgerät zeigt wiederum durch das DSR-Signal seine Betriebsbereitschaft an. Das Datenendgerät kann nun durch das RTS-Signal dem Datenübertragungsgerät mitteilen, daß es Daten senden möchte. Wenn das Datenübertragungsgerät zur Entgegennahme

der Daten vom Datenendgerät bereit ist, aktiviert es die CTS-Leitung.

Ist RTS dagegen inaktiv und liegt auch CTS auf 0, so kann das Datenübertragungsgerät durch sein DSR-Signal dem Datenendgerät mitteilen, daß es nun seinerseits senden möchte. Über die DCD-Leitung kann das Datenendgerät während des Empfangs feststellen, ob das Datenübertragungsgerät von der Gegenstation die Daten in ausreichender Stärke empfängt. Wird das DCD-Signal inaktiv, dann schaltet der Empfänger des Datenendgerätes ab und unterbricht damit automatisch die Datenübertragung.

Ein Gegenstück zur RS 232C-Schnittstelle bildet die 20-Milliampere-Stromschnittstelle, die im Prinzip nur aus einem Transistor und zwei Optokopplern aufgebaut ist (Bild). Diese Schnittstelle arbeitet nicht mit Spannungen, sondern mit definierten Strömen (0 mA und 20 mA) als Signalpegel. Dieses Stromschleifenprinzip wendet man überall dort an, wo hohe Störspannungsspitzen zu erwarten sind. Indem man für die Darstellung von Daten Ströme statt Spannungen benutzt, erreicht man bei der Datenfernübertragung eine äußerst hohe Störsicherheit.

(Alfred Kurz/ma)

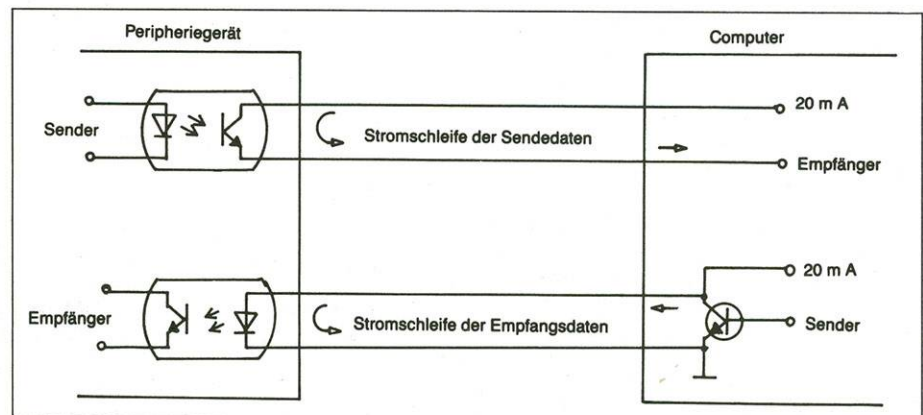
Die Pinbelegung der RS-232C-Schnittstelle

Pin	Kurzbezeichnung	Englische Bezeichnung	Datenendgerät	Bedeutung
1	CHASSIS GND	Protective ground	—	Schutzerde
2	TxD	Transmitted data	Ausgang	Sendedaten eines Datenendgerätes
3	RxD	Received data	Eingang	Empfangsdaten für ein Datenendgerät
4	RTS	Request to send	Ausgang	Anforderung des Datenendgerätes zum Senden
5	CTS	Clear to send	Eingang	Datenübertragungsgerät frei zum Übertragen an Gegenstation
6	DSR	Data set ready	Eingang	Datenübertragungsgerät betriebsbereit
7	GND	Signal ground	—	Signalerde
8	DCD	Data carrier detect	Eingang	Empfangssignalpegel von Gegenstation für Datenübertragungsgerät
9	—	+ Test voltage	—	Positive Prüfspannung
10	—	- Test voltage	—	Negative Prüfspannung
11	—	Select transmit frequency	Ausgang	Sendefrequenzwahl
12	2. DCD	Secondary carrier detect	Eingang	DCD für Rückkanal
13	2. CTS	Secondary clear to send	Eingang	CTS für Rückkanal
14	2. TxD	Secondary transmitted data	Ausgang	TxD für Rückkanal
15	TxC	Transmitter signal clock	Eingang	Sendetakt von DCE
16	2. RxD	Secondary received data	Eingang	RxD für Rückkanal
17	RxC	Receiver signal clock	Eingang	Empfangstakt von DCE
18	—	—	—	—
19	2. RTS	Secondary request to send	Ausgang	RTS für Rückkanal
20	DTR	Data terminal ready	Ausgang	Datenendgerät betriebsbereit
21	—	Signal quality detect	Eingang	Empfangsqualität von Gegenstation für Datenübertragungseinheit
22	RI	Ring indicator	Eingang	Anforderung von Gegenstation über Datenübertragungseinheit
23	DRS	Data rate select	Ausgang/ Eingang	Übertragungsrate für Datenübertragungsgerät/Datenendgerät
24	TxC	Transmitter signal clock	—	Sendetakt von DTE
25	BUSY	Line busy	Ausgang	Datenendgerät beschäftigt

Datenendgerät: DTE (Data terminal equipment)

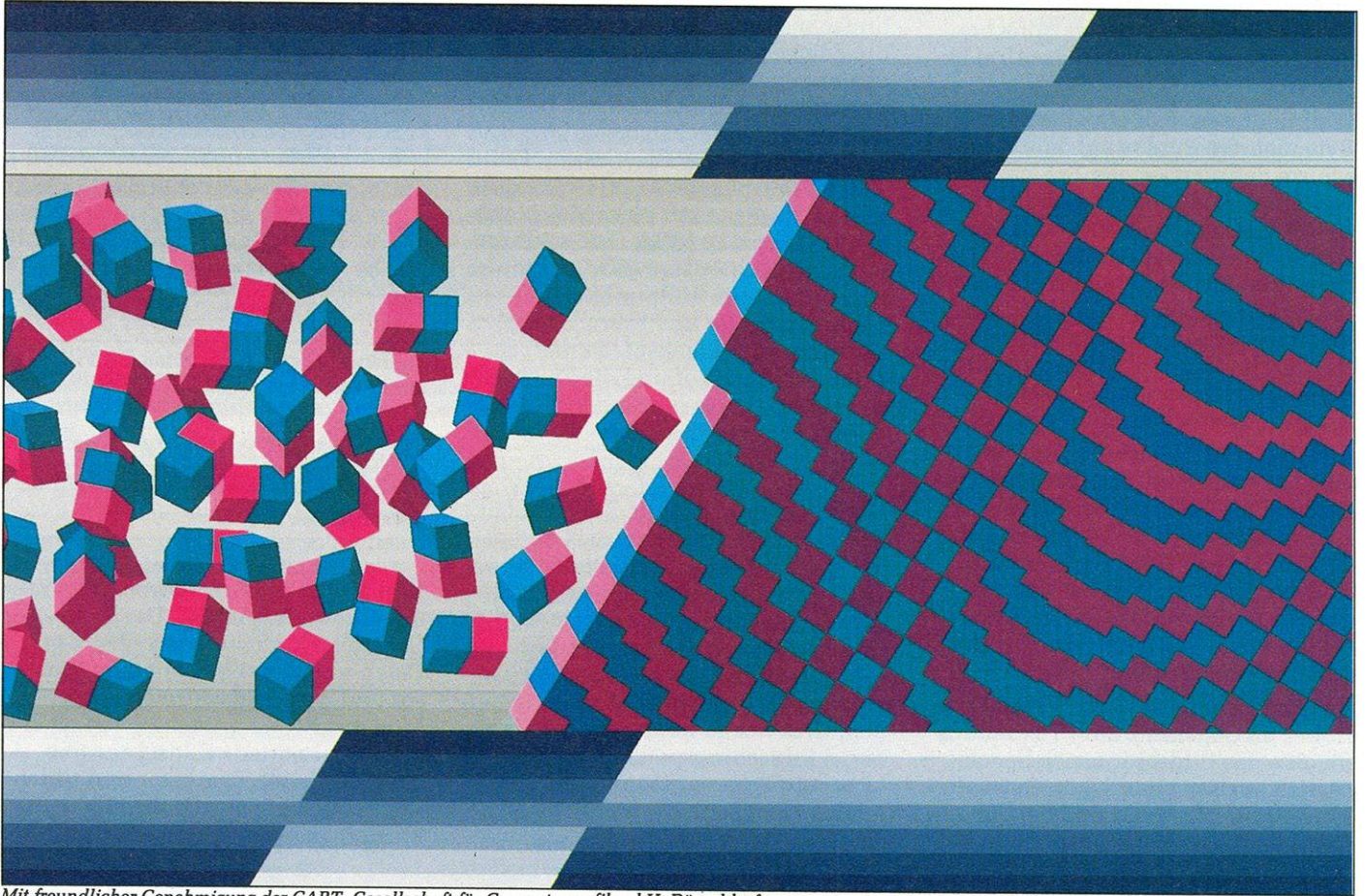
Datenübertragungsgerät: DCE (Data communication equipment)

Tabelle. Diese Schnittstellensignale sind für die RS 232C durch eine EIA-Norm festgelegt



Das Funktionsprinzip einer 20-Milliampere-Stromschnittstelle

Wie aus Daten Bilder werden



Mit freundlicher Genehmigung der CART, Gesellschaft für Computergrafik mbH, Düsseldorf

Wie gelingt es einer Grafikkarte, aus langweiligen Bytes und Bits Bilder, Grafiken und Texte für den Bildschirm hervorzuzaubern? Hier lesen Sie, wie eine Grafikkarte dieses kleine Wunder bewirkt und auf welche Weise sich eine Grafikkarte programmieren läßt.

Für den PC existieren viele verschiedene Grafikkarten, doch alle haben eines gemeinsam: Sie besitzen einen Video-Controller, der die Daten aus dem Bildschirmspeicher in elektrische Signale für das Bild auf dem Monitor umwandelt. Dieser Video-Controller ist das Herz jeder Grafikkarte.

Doch zunächst ein allgemeiner Überblick über die Arbeitsweise einer Grafikkarte. Daten, die im PC für den Monitor bestimmt sind, werden im

Bildschirmspeicher, der sich auf der jeweiligen Grafikkarte befindet, abgelegt. Die Bildschirmspeicher der verschiedenen Grafikkarten belegen unterschiedliche Speicherbereiche. Tabelle 1 zeigt die Speicherbereiche der wichtigsten Grafikkarten im Vergleich.

Um die Daten im Bildschirmspeicher des PC in Punkte, Zeichen oder Grafiken auf dem Bildschirm umzuwandeln, liest der Video-Controller diese Daten ein und formt daraus die elek-

trischen Signale für den Monitor, der dann den gewünschten Text oder die Grafik anzeigt.

Die Daten im Bildschirmspeicher des PC können auf zweierlei Art vorliegen. Zum einen kann jedes Byte des Bildschirmspeichers den Code für ein ASCII-Zeichen angeben, zum anderen können die Bits im Speicher direkt einzelne Grafikpunkte für Grafiken auf dem Bildschirm darstellen.

Deshalb unterscheidet der PC zwischen Textmodus und Grafikmodus.

Grafikkarte	Speicherbereich
Monochromadapter	B0000 bis B0FFFhex
Hercules-Karte	B0000 bis BFFFFhex
CGA-Farbgrafikkarte	B8000 bis BBFFFhex
EGA-Karte	A0000 bis BFFFFhex

Tabelle 1. Die Bildschirmspeicher verschiedener Grafikkarten belegen unterschiedliche Speicherbereiche

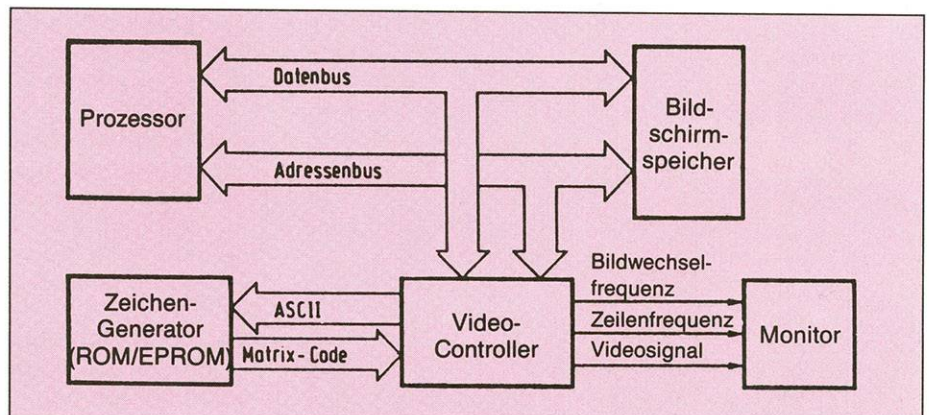


Bild 1. Der Video-Controller kommuniziert im PC mit Prozessor, Bildschirmspeicher, Zeichengenerator und Monitor

Im Textmodus interpretiert der Video-Controller die Bytes im Bildschirmspeicher als ASCII-Code und ordnet Ihnen eine Bildpunktmatrix zu, die den Aufbau des entsprechenden Zeichens auf dem Bildschirm beschreibt.

Weil in diesem Fall ein einzelnes Byte mehrere Punkte auf dem Bildschirm definiert, ist der Speicherbedarf eines Bildes im Textmodus wesentlich geringer als im Grafikmodus. Um den dadurch zusätzlich zur Verfügung stehenden Speicherplatz zu nutzen, arbeitet der PC im Textmodus mit sogenannten Bildschirmseiten, das heißt im Bildschirmspeicher können sich mehrere komplette Bilder gleichzeitig befinden.

So arbeitet beispielsweise die CGA-Farbgrafikkarte im Textmodus mit 80 Zeichen pro Zeile mit vier Bildschirmseiten und im Textmodus mit 40 Zeichen pro Zeile sogar mit acht Bildschirmseiten. Nur beim Monochromadapter, der keine Grafik darstellen kann, läßt sich nicht mit Bildschirmseiten arbeiten.

Im Grafikmodus muß der Video-Controller im Gegensatz zum Textmodus die Bits des Bildschirmspeichers lediglich auslesen und kann sie bei der Hercules-Karte direkt und bei der CGA- und EGA-Karte unter Berücksichtigung

sichtigung der Farbinformation als Bildpunkt auf dem Bildschirm umsetzen.

Einfache Controller verwenden für die Zeichendarstellung im Textmodus nur eine Matrix von 5x7 Bildpunkten, so daß sich Zeichen mit Unterlängen (die Buchstaben g, j, p, q und y) nicht exakt darstellen lassen. Der Video-Controller einer PC-Grafikkarte arbeitet dagegen mit einer Matrix von 8x8 Bildpunkten (CGA-Farbgrafikkarte), 14x8 Bildpunkten (EGA-Karte) oder sogar 14x9 Bildpunkten (Monochromadapter und Hercules-Karte). Dadurch treten bei Unterlängen keine Probleme auf und die Zeichen gewinnen mit zunehmender Bildpunktauflösung der Zeichenmatrix an Schärfe.

Natürlich ließe sich die Bildpunktauflösung der Zeichenmatrix und damit die Wiedergabequalität der einzelnen Buchstaben noch weiter steigern, doch dann müßte sich auch die Grafikaufklärung der Karte verbessern, damit sich noch 80x25 Zeichen auf dem Bildschirm darstellen ließen. Dies würde wiederum einen größeren Bildschirmspeicher und einen leistungsfähigeren Monitor erfordern – also erhebliche Kosten verursachen.

Die Funktionen des Video-Interrupt 10

Funktion	Bedeutung
00	Text- beziehungsweise Grafikmodus definieren
01	Cursorgöße einstellen
02	Cursor positionieren
03	Cursorposition lesen
04	Lichtgriffel-Position lesen
05	Aktive Bildschirmseite auswählen
06	Bildschirm aufwärts scrollen
07	Bildschirm abwärts scrollen
08	Zeichen und Attribut auf Cursorposition lesen
09	Zeichen und Attribut auf Cursorposition schreiben
0A	Zeichen auf Cursorposition schreiben
0B	Farbpalette oder Farbe auswählen
0C	Grafikpunkt setzen
0D	Grafikpunkt lesen
0E	Zeichen schreiben und Cursor bewegen
0F	Bildschirmstatus ermitteln

Tabelle 2. Die 16 Funktionen des Video-Interrupt 10

Der Video-Controller hat auf der Grafikkarte die Fäden in der Hand

Bild 1 zeigt die Arbeitsweise des Video-Controllers im PC. Der Video-Controller kommuniziert über den Daten- und Adreßbus mit dem Prozessor und dem Bildschirmspeicher. Moderne EGA-Karten besitzen oft einen »dual-ported« Bildschirmspeicher, das heißt, Prozessor und Video-Controller können gleichzeitig, jeder über einen eigenen Port auf den Bild-

Der Registersatz des Video-Controllers 6845

Register	Bits	MDA	CGA-Textmodus mit 80 Zeichen/Zeile	Voreinstellung CGA-Textmodus mit 40 Zeichen/Zeile	CGA-Grafikmodus	Bedeutung
0	0 bis 7	97	113	56	56	Zahl der (teilweise unsichtbaren) Textzeichen pro Textzeile -1 (für HSYNC-Signal)
1	0 bis 7	80	80	40	40	Zahl der tatsächlich angezeigten Textzeichen pro Textzeile
2	0 bis 7	82	90	45	45	Rasterzeilen-Position des HSYNC-Signals (bestimmt horizontale Verschiebung)
3	0 bis 3	15	10	10	10	Länge des HSYNC-Signals
	4 bis 7	0	0	0	0	Länge des VSYNC-Signals (0 = 16 Zeiteinheiten)
4	0 bis 6	25	31	31	127	Zahl der (teilweise unsichtbaren) Textzeilen -1
5	0 bis 4	6	6	6	6	Feineinstellung für Register 4 in Rasterzeilen
6	0 bis 6	25	25	25	100	Zahl der tatsächlich angezeigten Textzeilen
7	0 bis 6	25	28	28	112	Zeichenposition des VSYNC-Signals -1 (bestimmt senkrechte Verschiebung)
8	0 und 1	2	2	2	2	Zeilensprungmodus (Interlace) ein (1 und 3) oder aus (0 und 2)
9	0 bis 4	13	7	7	1	Höhe eines Zeichens in Rasterzeilen -1
10	0 bis 3	11	6	6	6	Erste Rasterzeile des Cursors (maximal 12)
	5 und 6	0	0	0	0	Bit 6 = 0 und Bit 5 = 0: Kein Blinken Bit 6 = 0 und Bit 5 = 1: Kein Cursor Bit 6 = 1 und Bit 5 = 0: Blinken Bit 6 = 1 und Bit 5 = 1: Langsam blinkender Cursor
11	0 bis 3	12	7	7	7	Letzte Rasterzeile des Cursors (maximal 12)
12	0 bis 5	—	—	—	—	Bit 8 bis Bit 13 der Startadresse des Bildschirmspeichers
	6 und 7	0	0	0	0	—
13	0 bis 7	—	—	—	—	Bit 0 bis Bit 7 der Startadresse des Bildschirmspeichers
14	0 bis 5	—	—	—	—	Bit 8 bis Bit 13 der Adresse des Cursors (maximal 1999)
	6 und 7	0	0	0	0	—
15	0 bis 7	—	—	—	—	Bit 0 bis Bit 7 der Adresse des Cursors (maximal 1999)
16	0 bis 5	—	—	—	—	Bit 8 bis Bit D der Bildschirmadresse, bei der ein Lichtgriffelimpuls auftrat
	6 und 7	0	0	0	0	—
17	0 bis 7	—	—	—	—	Bit 0 bis Bit 7 der Bildschirmadresse, bei der ein Lichtgriffelimpuls auftrat

Tabelle 3. Über die 18 Register des Video-Controllers 6845 läßt sich alles programmieren, was mit dem Bildschirmaufbau zu tun hat

schirmspeicher zugreifen. Dadurch läßt sich der lästige »Schneefall« vermeiden, der auftritt, wenn der Prozessor Daten in den Bildschirmspeicher schreiben möchte, während der Video-Controller Daten ausliest.

Der Video-Controller liest periodisch (50, 60 oder 70 Mal pro Sekunde) den Inhalt einer Bildschirmseite (Textmodus) oder den gesamten Bildschirmspeicher (Monochromadapter oder Grafikmodus) aus und formt ihn in Signale für einen Monitor um.

Arbeitet der PC im Textmodus, so übergibt der Video-Controller die eingelesenen Bytes aus dem Bildschirmspeicher als ASCII-Wert an den Zeichengenerator. Jeder ASCII-Wert gibt eine Adresse an, unter der im Zeichengenerator die Zeichenmatrix zu dem entsprechenden ASCII-Zeichen abgelegt ist. Diese Zeichenmatrix liest der Video-Controller und erzeugt daraus über ein sehr schnelles Schieberegister das serielle Videosignal.

Befindet sich der PC dagegen im Grafikmodus, so kann der Video-Controller die aus dem Bildschirmspeicher eingelesenen Daten direkt über das Schieberegister in das serielle Videosignal umformen.

Doch das Videosignal allein reicht nicht aus, um ein Bild auf dem angeschlossenen Monitor entstehen zu lassen. Vielmehr muß der Video-Controller auch zwei Steuersignale erzeugen, die den Bildaufbau regeln. Das eine Steuersignal ist die Bildwechselfrequenz, die für die vertikale Synchronisation zuständig ist, das andere die Zeilenfrequenz, die die horizontale Synchronisation übernimmt.

Die Bildröhre des Monitors baut das Bild aus horizontalen Zeilen auf, indem der Elektronenstrahl der Bildröhre von links nach rechts auf den Bildschirm schreibt. Beim Rücklauf des Strahls vom Zeilenende zum Anfang der nächsten Zeile wird der Strahl dunkel geschaltet (ausgetastet), damit er unsichtbar bleibt.

Der Monitor bringt die einzelnen Zeilen von oben nach unten auf den Bildschirm und kehrt am Ende der letzten Zeile wieder diagonal zum Anfang der ersten Zeile zurück. Auch bei diesem Rücklauf wird der Strahl ausgetastet. Diese Methode des Bildaufbaus auf dem Bildschirm ist bei allen Monitoren identisch.

Ein solches Bild besteht nach deutscher Norm beim Fernsehgerät aus 625 Bildschirmzeilen und wird 25 Mal pro Sekunde aufgebaut. Die Trägheit des menschlichen Auges, also die Flimmergrenze, liegt jedoch bei 30 bis 40 Hz. Dadurch entsteht

bei einer Bildwechselfrequenz von 25 Hz beim Betrachter der Eindruck, daß das Bild flimmert.

Um diesen Effekt zu verhindern, ließ man sich einen einfachen, aber wirkungsvollen Trick einfallen. Er besteht darin, daß man das sogenannte Vollbild in zwei Halbbilder aufteilt. Das erste Halbbild besteht aus den Bildschirmzeilen mit den ungeraden, das zweite Halbbild aus den Bildschirmzeilen mit den geraden Zeilennummern. Der Elektronenstrahl schreibt nun zuerst die Zeilen des ersten Halbbildes und darauf die Zeilen des zweiten Halbbildes, so daß im Endeffekt 50 Mal pro Sekunde ein Halbbild erzeugt wird.

Dieses Halbbildverfahren hat den Vorteil, daß das Bild für das menschliche Auge nicht mehr flimmert und die Video-Bandbreite durch diese Methode nicht erhöht werden muß (Kostenersparnis).

Bei billigen Heimcomputern wendet der Video-Controller ebenfalls das Halbbildverfahren zum Bildaufbau an und speist sein auf einen hochfrequenten Träger aufmoduliertes Videosignal in die Eingangsbuchse des Fernsehgerätes.

Dieses Verfahren ist jedoch im Grunde genommen für Computer ungeeignet, weil das Computerbild oft statisch und dadurch das Flimmern trotz einer Bildwechselfrequenz von 50 Hz deutlich zu erkennen ist. Insbesondere bei Textverarbeitungs-

Das Attribut des Monochromadapters

Wert	Bedeutung
0	Unsichtbar
1	Unterstrichen
7	Normale Darstellung
9	Leuchtend und unterstrichen
15	Leuchtend
112	Invertiert
129	Blinkend und unterstrichen
135	Blinkend
137	Blinkend, leuchtend und unterstrichen
143	Blinkend und leuchtend
240	Blinkend und invertiert

Tabelle 4. Das Attribut des Monochromadapters und seine Bedeutungen

Datenbank- und Tabellenkalkulations-Programmen bekommt der Anwender diesen Nachteil zu spüren.

Deshalb hat man sich beim PC erst gar nicht auf das unprofessionelle Halbbildverfahren eingelassen. Die Grafikkarten des PC stellen die Bilder mit einer echten Bildwechselfrequenz von 50 Hz (Monochromadapter und Hercules-Karte), 60 Hz (CGA- und EGA-Karte) oder sogar 70 Hz (VGA-Grafik) dar. »Echt« heißt in diesem Fall, daß die Grafikkarten 50, 60 oder 70 Vollbilder pro Sekunde aufbauen.

Zur Programmierung des Video-Controllers stellt das BIOS des PC den Video-Interrupt 10 mit den Funktionen 00 bis 0Fhex zur Verfügung. Tabelle 2 zeigt die Bedeutung der 16 Funktionen. Diese Funktionen und die zusätzlichen Funktionen einer EGA-

3B8hex Steuerregister (nur schreiben)

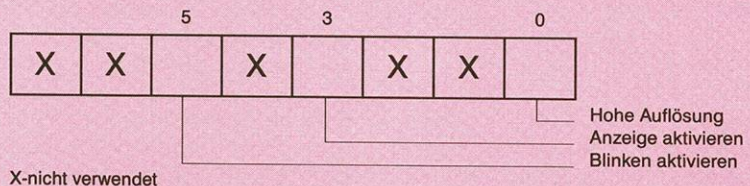


Bild 2. Das Steuerregister des 6845 kann nur beschrieben werden

3BAhex Statusregister (nur lesen)

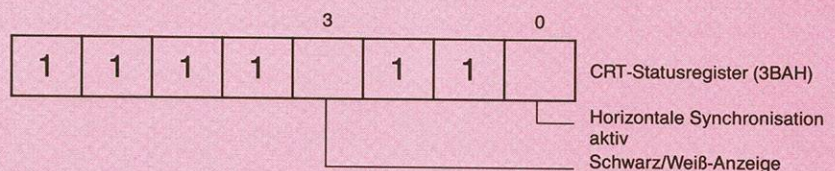


Bild 3. Aus dem Statusregister des 6845 läßt sich nur lesen

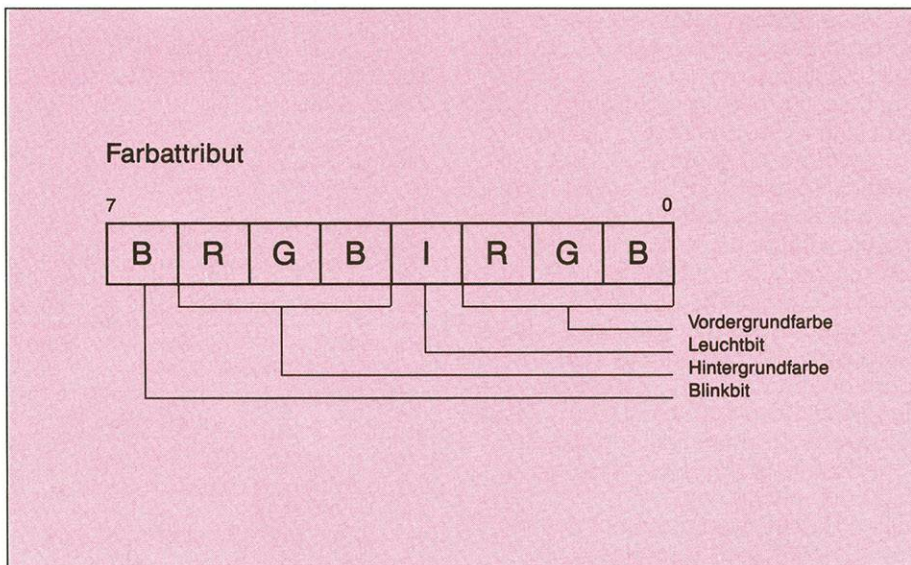


Bild 4. Das Farbattribut erlaubt im Textmodus der CGA-Farbgrafikkarte 128 verschiedene Farbkombinationen und blinkende Darstellung

Karte werden in verschiedenen Beiträgen in dieser Ausgabe detailliert erklärt.

Der Monochromadapter (MDA) und die CGA-Farbgrafikkarte arbeiten beide mit dem Video-Controller 6845. Dieser Video-Controller läßt sich über seinen internen Registersatz von 18 Registern (0 bis 17) direkt programmieren.

Allerdings sollten Sie dabei sehr sorgsam vorgehen und die bestehenden Parameter nur in kleinen Schritten ändern, weil sonst Ihr Monitor Schaden nehmen könnte.

Die internen Register des Video-Controllers 6845 lassen sich über das Adreßregister unter der Portadresse 3B4 (MDA) oder 3D4 (CGA) auswählen und über das Datenregister unter der Portadresse 3B5 (MDA) oder 3D5 (CGA) beschreiben (nur die Register 00 bis 0Fhex) und auslesen (nur die Register 10 und 11). Tabelle 3 zeigt den kompletten Registersatz des 6845 mit den Voreinstellungen für verschiedene Videomodi und den Bedeutungen der Register.

Für gefahrloses Experimentieren sind die Register 10, 11, 14 und 15 geeignet. Die Register 10 und 11 geben die Start- und Endzeile des Bildschirmcursors an. Beim Monochromadapter lautet die Startzeile 11 und die Endzeile 12, während bei der CGA-Farbgrafikkarte der Cursor von Zeile 6 bis 7 reicht. In beiden Fällen stellt der Video-Controller einen zweizeiligen Cursor dar. Ändern Sie die Startzeile beispielsweise in 0 um, so erhalten Sie einen Blockcursor.

Die Register 14 und 15 bestimmen die Cursorposition auf dem Bildschirm. Als Positionen sind die Werte 0 bis 1999 erlaubt, wobei 0 die linke obere Bildschirmecke und 1999 die rechte untere Bildschirmecke bezeichnet. Die Blinkfrequenz des Cursors läßt sich softwaremäßig leider nicht verändern, weil deren Steuerung durch die Hardware erfolgt.

Der Video-Controller 6845 besitzt einen reichhaltigen und leistungsfähigen Registersatz

Neben den 18 internen Registern sowie dem Adreß- und Datenregister besitzt der Video-Controller noch unter der Portadresse 3B8 ein Steuerregister (Bild 2) und unter der Portadresse 3BA ein Statusregister (Bild 3). Über Bit 3 des Steuerregisters wird beispielsweise der gesamte Bildschirm dunkel geschaltet und Bit 5 läßt den gesamten Bildschirm blinken.

Bit 0 des Statusregisters ist wichtig, wenn Sie Daten ohne Umweg über den Video-Interrupt 10 direkt in den Bildschirmspeicher schreiben möchten. Ist dieses Bit gesetzt, so ist das Signal zur horizontalen Synchronisation (Strahlrücklauf) aktiv, und Sie können ein Byte in den Bildschirmspeicher schreiben, ohne »Bildschirmschnee« durch den gleichzeitigen Speicherzugriff des Video-Controllers befürchten zu müssen.

Beim Monochromadapter und in den beiden Textmodi der CGA-Farbgrafikkarte entspricht jede Zeichenposi-

tion auf dem Bildschirm einem 2-Byte-Wert im Arbeitsspeicher. Das erste Byte kennzeichnet den ASCII-Wert des jeweiligen Zeichens, und das zweite Byte gibt dessen Attribut an.

Tabelle 4 zeigt die Bedeutungen der Attributwerte beim Monochromadapter und Bild 4 gibt Auskunft über den Aufbau des Farbattributs, das in den beiden Textmodi der CGA-Farbgrafikkarte Vorder- und Hintergrundfarbe, die Intensität der Vordergrundfarbe und die blinkende Darstellung bestimmt.

Die erste Speicherstelle im Bildschirmspeicher des Monochromadapters oder der CGA-Farbgrafikkarte ist auch mit dem ersten Zeichen auf dem Bildschirm belegt. Die nachfolgenden Speicherstellen sind in der gleichen Reihenfolge belegt, wie die Zeichen auf dem Monitor erscheinen.

Die Speicheradresse eines Zeichens läßt sich beim Monochromadapter wie folgt berechnen:

$$720896 + 2 \times (80 \times \text{zeile} + \text{spalte})$$

Bei der CGA-Farbgrafikkarte lautet die Formel im Textmodus mit 80 Zeichen pro Zeile

$$753664 + 2 \times (80 \times \text{zeile} + \text{spalte})$$

und im Textmodus mit 40 Zeichen pro Zeile entsprechend

$$753664 + 2 \times (40 \times \text{zeile} + \text{spalte})$$

Der Parameter »zeile« darf dabei jeweils Werte von 0 bis 24 und der Parameter »spalte« Werte von 0 bis 79 beziehungsweise von 0 bis 39 annehmen. Wenn Sie die Adresse eines Zeichens um 1 erhöhen, erhalten Sie die Speicheradresse des zugehörigen Attributs.

Um bei der CGA-Farbgrafikkarte nicht nur die erste, sondern auch die folgenden Bildschirmseiten ansprechen zu können, müssen Sie das Ergebnis der zweiten Formel für jede weitere Seite um den Wert 4096 und das Ergebnis der dritten Formel für jede zusätzliche Seite um 2048 erhöhen.

Jetzt dürfte es Ihnen nicht mehr schwerfallen, die Funktion des Video-Controllers im PC zu verstehen und ihn sowie den Bildschirmspeicher direkt zu programmieren. Ihre selbstgeschriebenen Programme werden dadurch in Zukunft mit überraschenden Effekten aufwarten.

(Alfred Kurz/ma)

Schnelle Speicherverwaltung mit dem DMA-Controller

Ohne einen Arbeitsspeicher ist kein Computer zu gebrauchen. Im PC sorgt ein eigener Baustein dafür, daß zwischen Arbeitsspeicher und dem Rest des Computers alles glatt läuft. »DMA-Controller 8237« heißt der Chip von Intel, der den Speicher des PC verwaltet.

Für den Datenfluß zwischen Speicher und Mikroprozessor verwendet man in modernen Computern DMA-Bausteine, die speziell für diese Zwecke konzipiert wurden. Der in den PCs eingebaute 8237 ist ein solcher DMA-Baustein. DMA bedeutet »direkter Speicherzugriff« oder »direct memory access«.

Der 8237 nimmt dem Prozessor des Computers alle Verwaltungsarbeiten in Bezug auf den Speicherbereich ab und entlastet damit das gesamte System. Der Erfolg liegt klar auf der Hand: Der Computer arbeitet deutlich schneller.

Der 8237 kann bis zu 64 KByte große Datenbereiche von einem Speicherbereich zum anderen transportieren. Der 8237 beginnt mit der Arbeit, sobald auf seiner Leitung DREQ (DMA-Anforderung) ein Signal anliegt. Sofort sendet er dem Peripheriegerät über die Leitung DACK (DMA-Bestätigung) ein Auswahlsignal. Dies bewirkt, daß über den Eingangskanal ein Byte auf dem Datenbus liegt. Gleichzeitig stellt das System freien Speicherplatz zur Verfügung und sendet ein Signal, das die Übernahmebereitschaft für die Daten kennzeichnet.

Ein solcher kompletter Zyklus dauert nur 5 Takte, während der Prozessor 29 Takte benötigen würde. Weil der 8237 vier getrennte Kanäle besitzt, ist er in der Lage, vier solcher Funktionen gleichzeitig durchzuführen.

Überträgt der Computer Daten zu einem Peripheriegerät, so aktiviert der DMA-Controller die Leitungen MEMWR, IOW und DACK. Da sich dadurch und durch die zusätzlich anliegende Adresse bedingt auch andere Ein-/Ausgabegeräte angesprochen fühlen können, wählt das System mit dem Signal AEN das entsprechende Gerät aus. Adreßdekoder verhindern während dieser Zeit die Aktivierung weiterer Ein-/Ausgabegeräte.

Für diese Datentransfers benötigt man jeweils nur einen DMA-Kanal. Anders sieht es dagegen bei einer

Verschiebung innerhalb des Speichers aus. Hierzu benötigt der 8237 zwei Datenkanäle. Da sich keine Speicherzelle mit Hilfe einer DACK-Leitung anwählen läßt, muß die Speicher-zu-Speicher-Übertragung sequentiell erfolgen. Dabei liegt die Quelladresse über Kanal 0 und die Zieladresse über Kanal 1 an.

Die Übertragung innerhalb des Speichers legen Sie unter der Portadresse 008 fest. Ist das erste Bit (0) gleich 1, so ist die Speicher-zu-Speicher-Übertragung freigegeben. Mit Hilfe der über die Portadresse 00A ansprechbaren Bits 0 und 1 läßt sich festlegen, ob die Quelladresse inkrementiert oder dekrementiert wird, oder konstant bleibt. Es ist zu beachten, daß der Prozessor während einer DMA-Übertragung kein Programm ausführen kann und auf die Beendigung der Übertragung wartet.

Die Programmierung des 8237 erfolgt mit Hilfe zahlreicher Befehle und Adressen. Zur Steuerung eines jeden einzelnen Kanals steht dem Programmierer ein 6 Bit langes Modus-Auswahlregister zur Verfügung, das unter der Portadresse 00B zur Erreichung ist. Tabelle 1 zeigt den Aufbau des Modus-Auswahlregisters.

Abhängig von Bit 5 inkrementiert oder dekrementiert der 8237 die aktuelle Adresse nach der Übertragung eines Bytes und dekrementiert gleichzeitig den aktuellen Zähler. Ist dieser Zähler auf Null gestellt oder die Leitung EOP aktiviert, so endet die Übertragung. Darauf erhalten die Basisregister die Anfangswerte der aktuellen Register, so daß diese eine neue Information aufnehmen können. Das ist der Fall, sobald der Zähler auf Null steht und Bit 4 den Wert 1 hat.

Durch die beiden Bits 2 und 3 geben Sie entweder einen Schreib- oder Lesebefehl und mit den Bits 6 und 7 stellen Sie die verschiedenen Übertragungsarten wie Anforderungs-, Einzel-, Block- oder Kaskadierungsmodus ein. Dabei hat es sich als günstig herausgestellt, bei Übertragungsgeschwindigkeiten von weniger als einem Byte pro 4 Taktzyklen im Einzelübertragungsmodus zu arbeiten, denn hierbei überträgt das System mit jeder Aktivierung des Signals DREQ ein Byte. Diesen Modus verwendet man ebenfalls zum Auffrischen der dynamischen Speicherbau-

steine und für Übertragungen von oder zum Diskettenlaufwerk, die mit 32 Mikrosekunden pro Byte vonstatten gehen.

Natürlich können Sie diese Operation auch mit dem Prozessor selbst durchführen. Allerdings müssen Sie dann, im Gegensatz zur DMA-Operation,

Das Modus-Auswahlregister

Zustand	Bedeutung
Bit 0, Bit 1 = 00	Maskierung von Kanal 0 aktiv
Bit 0, Bit 1 = 11	Maskierung von Kanal 1 aktiv
Bit 0, Bit 1 = 10	Maskierung von Kanal 2 aktiv
Bit 0, Bit 1 = 11	Maskierung von Kanal 3 aktiv
Bit 2, Bit 3 = 01	Verifizieren
Bit 2, Bit 3 = 01	Schreiben
Bit 2, Bit 3 = 10	Lesen
Bit 4 = 1	Freigabe der Auto-Initialisierung
Bit 5 = 0	Adressen jeweils um 1 erhöhen
Bit 5 = 1	Adressen jeweils um 1 vermindern
Bit 6, Bit 7 = 00	Anforderung
Bit 6, Bit 7 = 01	Einzelmodus
Bit 6, Bit 7 = 10	Blockmodus
Bit 6, Bit 7 = 11	Kaskadierung

Tabelle 1. Der Aufbau des Modus-Auswahlregisters

Das Kanalanforderungs-Register

Zustand	Bedeutung
Bit 0, Bit 1 = 00	Anforderung von Kanal 0
Bit 0, Bit 1 = 01	Anforderung von Kanal 1
Bit 0, Bit 1 = 10	Anforderung von Kanal 2
Bit 0, Bit 1 = 11	Anforderung von Kanal 3
Bit 2 = 0	Anforderungsbit löschen
Bit 2 = 1	Anforderungsbit setzen

Tabelle 2. Der Aufbau des Kanalanforderungs-Registers

Das Kanalmaskierungs-Register

Zustand	Bedeutung
Bit 0, Bit 1 = 00	Maskierung für Kanal 0
Bit 0, Bit 1 = 01	Maskierung für Kanal 1
Bit 0, Bit 1 = 10	Maskierung für Kanal 2
Bit 0, Bit 1 = 11	Maskierung für Kanal 3
Bit 2 = 0	Maskierung löschen
Bit 2 = 1	Maskierung setzen

Tabelle 3. Der Aufbau des Kanalmaskierungs-Registers

Das Hauptsteuerregister

Zustand	Bedeutung
Bit 0 = 1	Speicherübertragung freigeben
Bit 1 = 1	Adresse von Kanal 0 speichern
Bit 2 = 1	Controller sperren
Bit 3 = 1	Komprimierte Zeitabfrage
Bit 4 = 1	Rotierende Priorität
Bit 5 = 1	Schreibbefehlserweiterung
Bit 6	Festlegen der Bedeutung des Signals DREQ
Bit 7	Festlegen der Bedeutung des Signals DACK

Tabelle 4. Der Aufbau des Hauptsteuerregisters

die Echtzeituhr sowie die Tastaturinterrupts abschalten. Eine undankbare Aufgabe, die zudem den Prozessor für andere Aufgaben blockiert.

Ist ein angeschlossenes Datengerät ebenso schnell oder sogar schneller als der 8237 selbst, so verwenden Sie sinnvollerweise den Blocktransfer. Im anderen Fall, das heißt, wenn das Datengerät langsamer ist, besteht für das Peripheriegerät die Möglichkeit, den DMA-Controller mit Hilfe der Leitung Ready zum Warten zu veranlassen. Der Controller fügt dann automatisch, ähnlich wie der Prozessor, Wartezyklen zur künstlichen Verlängerung der Zugriffszeit ein.

Der 8237 besitzt drei weitere Kontrollregister von denen Sie nachstehend eine Aufstellung der einzelnen Bedeutungen finden. Mit Hilfe des Kanal Anforderungs-Registers können sie ein Programm dazu veranlassen, eine DMA-Anforderung zu stellen. Die Portadresse hierfür ist 009. Tabelle 2 zeigt den Aufbau des Kanal Anforderungs-Registers.

Mit einer Kombination der ersten beiden Bits aktivieren Sie einen der vier DMA-Kanäle und ordnen ihn einer anfordernden Einheit zu. Das Bit 2 ist in diesem Zusammenhang das wichtigste, denn abhängig von dessen Zustand können Sie die Anforderungsleitung freigeben oder sperren.

Das zweite steuernde Register ist das Kanalmaskierungs-Register. Mit diesem Register können Sie alle DMA-Kanäle einzeln an- und abschalten. Die Portadresse für das Kanalmaskierungs-Register lautet 00A. Tabelle 3 zeigt den Aufbau dieses Registers.

Wie schon beim Kanal Anforderungs-Register, so erfolgt auch beim Kanalmaskierungs-Register die Manipulierung der einzelnen Kanäle mit Hilfe der ersten beiden Bits. Somit ist ein einfaches Ein- und Ausblenden der Kanäle durchführbar, ohne sie gleich inaktiv schalten zu müssen – einer der wesentlichen Vorzüge des 8237.

Das Bit 2 sorgt auch hier wieder für die Aktivierung oder Deaktivierung der Maskierungsprozeduren.

Das letzte wichtige Steuerregister des 8237 ist das Hauptsteuerregister, das Sie über die Portadresse 008 ansteuern. In diesem Register legen Sie die Betriebsarten des 8237 fest und nehmen damit die Grobsteuerung für die anderen schon beschriebenen Befehlsregister vor. Tabelle 4 zeigt den Aufbau des Hauptsteuerregisters.

Das erste Bit dient zur Freigabe der Speicher-zu-Speicher-Übertragung. Wie bereits erwähnt, kann der 8237 maximal 64 KByte große Blöcke auf einmal verschieben. Mit dem zweiten Bit können sie den Controller veranlassen, die aktuelle Adresse des ersten Übertra-

gungskanals zu speichern und festzuhalten. Während intensiver Datentransfers gestattet eine solche Programmierung einen schnellen Zugriff auf diesen Kanal, ohne viel hin- und herschalten zu müssen.

Setzen Sie das Bit 2, so ist der Controller solange gesperrt, bis sich der Zustand dieser Bits wieder ändert. Damit ist eine Kontrolle über den Baustein aus einem laufenden Programm heraus kein Problem.

Bit 3 gestattet eine häufigere Abfrage der einzelnen Kanäle, was jedoch bei Diskettenzugriffen zu Zeitproblemen führen kann. Die rotierende Priorität, die mit Bit 4 bestimmt wird, gestattet den schnellen Wechsel einzelner Prioritätsstufen, was bei Interruptbehandlungen dienlich ist. Die restlichen drei Bits haben wir schon weiter oben erläutert.

Mit diesen Hinweisen sind Sie in der Lage, Ihre ersten Schritte bei der Programmierung des DMA-Controllers 8237 zu wagen. Durch entsprechende Routinen werden Sie diesen vielseitigen Baustein noch effektiver nutzen können. Denken Sie beispielsweise an die Verschiebung von Textblöcken innerhalb eines größeren Textes, die sich über den 8237 deutlich beschleunigen lassen. Sicher fallen Ihnen mit der Zeit viele weitere Anwendungsmöglichkeiten für den DMA-Controller ein.

(Alfred Kurz/ma)

Die Zeit im Griff mit dem Timerbaustein 8253

Der Timerbaustein 8253 fungiert im PC als Mädchen für alles. Ob Uhrzeit, RAM-Refresh oder Tonausgabe, ohne den Timerbaustein läuft hier nichts. In alten Tagen war er sogar für den Kassettenspieler zuständig. Doch wenigstens das bleibt ihm heutzutage erspart.

Die Bedeutung des Timerbausteins 8253 wird auch von Hardware-Freaks oft unterschätzt. Zugegeben, im PC arbeitet der 8253 hauptsächlich für andere Bausteine, indem er für sie auf- und abwärts zählt, doch nichtsdestotrotz sind es gerade diese grundlegenden Arbeiten, deren Ausbleiben den Computer sofort hoffnungslos abstürzen läßt. Unterbleibt beispielsweise auch nur kurzzeitig der Refresh für den Arbeitsspeicher, so ist der berühmt-berüchtigte Paritätsfehler nicht mehr weit.

Der 8253 ist strenggenommen ein 16-Bit Zähler, der drei voneinander unabhängige Kanäle besitzt. Jeder der drei Kanäle ist in der Lage, einen Eingangstakt im Bereich von 0 bis 2 MHz zu übernehmen. Dieser Takt läßt sich durch einen beliebigen 16-Bit-Wert teilen, so daß am Ausgang eines Kanals praktisch jeder

Das Steuerregister des Timerbausteins

Zustand	Funktion
Bit 0 = 0	16-Bit Binärzähler ist eingeschaltet
Bit 0 = 1	BCD-Zähler ist eingeschaltet
Bit 1, 2, 3 = 000	Modus 0 ist eingestellt
Bit 1, 2, 3 = 001	Modus 1 ist eingestellt
Bit 1, 2 = 10	Modus 2 ist eingestellt
Bit 1, 2 = 11	Modus 3 ist eingestellt
Bit 1, 2, 3 = 100	Modus 4 ist eingestellt
Bit 1, 2, 3 = 101	Modus 5 ist eingestellt
Bit 4, 5 = 00	Augenblicklichen Zählerstand zum Lesen übernehmen
Bit 4, 5 = 01	Lese oder lade das niederwertige Byte (LSB)
Bit 4, 5 = 10	Lese oder lade das höherwertige Byte (MSB)
Bit 4, 5 = 11	Zuerst das LSB, dann das MSB lesen oder laden
Bit 6, 7 = 00	Zähler 0 ist angewählt
Bit 6, 7 = 01	Zähler 1 ist angewählt
Bit 6, 7 = 10	Zähler 2 ist angewählt
Bit 6, 7 = 11	Konfiguration ist unbenutzt

Tabelle. Der Aufbau des Steuerregisters im Timerbaustein 8253

Takt bis zu einer Frequenz von 2 MHz erzeugt werden kann. Am Takteingang des 8253 liegt im PC fest eine Frequenz von 1,93 MHz (4,77 MHz/4) an.

Der Ausgang des ersten Kanals 0 des 8253 steht im PC für die Steuerung der internen Systemuhr zur Verfügung. Der Ausgang von Kanal 1 dient als Refresh-Signal für den DMA-Controller, um diesen zum Auffrischen der dynamischen Speicherbausteine zu veranlassen. Der Ausgang von Kanal 2 ist direkt mit dem eingebauten Lautsprecher verbunden, so daß dieser Kanal für die bescheidenen Sound-Fähigkeiten des PC verantwortlich ist.

Im PC werden die Fähigkeiten des 8253 nur zum Erzeugen eines Ausgangssignals, das dem durch eine 16-Bit-Zahl geteilten Wert des Systemtaktes entspricht, verwendet. Dabei ist jeder einzelne Kanal auch in der Lage, in einer von sechs möglichen Betriebsarten zu arbeiten, die im folgenden aufgeführt sind:

Modus 0: Erzeugen eines Interrupts nachdem eine bestimmte Anzahl von Impulsen gezählt wurde oder nach einem Zählerüberlauf.

Modus 1: Programmierbarer Monoflop

Modus 2: Taktgenerator

Modus 3: Rechtecksignalgenerator

Modus 4: Softwaremäßige Triggerung des Ausgangs

Modus 5: Hardwaremäßige Triggerung des Ausgangs

Im Modus 0 liegt die Leitung OUT (Ausgang) für eine bestimmte programmierbare Anzahl von Impulsen, die über die CLK-Leitung (Takt) einkommen, auf 0, um nach Erreichen des vorprogrammierten Zählerstandes wieder auf 1 zu gehen.

Mit Hilfe dieser beiden Übergänge (erst von 1 auf 0 und anschließend wieder von 0 auf 1) läßt sich programmtechnisch die Arbeit des Prozessors für eine bestimmte Zeitdauer unterbrechen.

Ähnlich arbeitet ein Kanal im Modus 1. Hierbei handelt es sich, wie auch im Modus 0, um die Programmierung eines Monoflops (monostabile Kippstufe).

Allerdings startet hier die Zählfunktion nicht durch einen Befehl, sondern durch einen positiven Flankenwechsel an der Steuerleitung GATE (Tor). Man spricht in diesem Fall

auch von einem zustandsgesteuerten Monoflop.

Modus 2 stellt einen Teiler durch eine beliebige 16-Bit große Zahl dar. Der Ausgang bleibt während der festgelegten Zeit auf 1 und geht nach Erreichen dieser Zeit für eine Taktperiode auf 0.

Mit diesem Verhalten ergibt sich für den Ausgang ein asymmetrisches Taktverhältnis, dessen Periode vom Eingangstakt, geteilt durch den 16-Bit-Wert, bestimmt wird. Eine praktische Anwendung für diese Arbeitsweise ist in Verbindung mit dem Kanal 1 das Erzeugen der Refresh-Zyklen für die dynamischen Speicherbausteine im PC.

Möchten Sie ein symmetrisches Signal am Ausgang eines Zählers erhalten, so müssen Sie den Modus 3 verwenden.

Hier läßt sich ein symmetrischer, durch den 16-Bit-Wert geteilter Zähler aufbauen. Sein Kennzeichen ist ein Ausgangssignal, dessen 0-Pegel und 1-Pegel gleich lang sind. Dieser Modus, der einem symmetrischen Taktgenerator gleicht, wird im PC zur Kontrolle der Uhrzeit und zum Erzeugen der Töne für den Lautsprecher verwendet.

Die letzten beiden zur Verfügung stehenden Modi 4 und 5 ähneln in ihrer Funktion den Betriebsarten 0 oder 1.

Allerdings ist die Auswirkung auf die Zustände der Signalleitungen verschieden, denn in Modus 4 und 5 liegt die OUT-Leitung für die Dauer der vorgegebenen Impulse auf 1 und wechselt dann für eine ganze Periode auf 0, um anschließend wieder auf 1 zu wechseln.

Die Besonderheit der beiden über den GATE-Eingang steuerbaren Modi 1 und 5 ist, daß sich ihr Zustand vor Beendigung auffrischen und anschließend neustarten (retriggern) läßt. Dabei bleibt in diesem Fall der Ausgang OUT nach der letzten positiven Flanke auf dem Gate-Eingang während der gesamten Zähldauer in seinem vorgegebenen Zustand.

Im Modus 1 entspricht dies einem 0-Pegel und in Modus 5 einem 1-Pegel.

Dies bedeutet, daß jede neu ankommende Signalfanke den Zählvorgang von neuem startet. In allen Modi, ausgenommen dem Modus 1, kann man den Zähler solange arbeiten lassen, wie der Eingang GATE auf 1 liegt.

Der Timerbaustein 8253 läßt sich auf verschiedene Arten programmieren

Die Programmierung der drei Zählerbausteine im PC ist relativ einfach, weil die Registerstruktur sehr übersichtlich ist und nicht zu viele unterschiedliche Funktionen pro Kanal zuläßt.

Die Portadressen für die Programmierung des 8253 liegen bei 040 bis 043hex.

Das Register von Kanal 0 hat die Portadresse 040. Er ist für die Zeit und Einstellung des Datums der Systemuhr zuständig. Das unter der Adresse 041 anzusteuern Register des Kanals 1 sorgt für das Refresh-Signal zum DMA-Kanal 0.

Die Steuerung des Rechtecksignals für den Lautsprecher erfolgt schließlich über den Kanal 2, dessen Register über die Portadresse 042 zu erreichen ist.

Die Tabelle gibt Ihnen eine Übersicht zu den verschiedenen Funktionen des Steuerregisters, das über die Portadresse 043 angesprochen wird.

Zur Programmierung des 8253 muß man diesen Baustein zuerst initialisieren, indem man für jeden zu steuernden Kanal ein Moduskontrollbyte in seine Register schreibt.

Es hat sich als sinnvoll erwiesen, vor dem Lesen der beiden Byte eines 16-Bit-Zählers zuvor einen Befehl zum Zwischenspeichern zu geben, den Sie mit dem Wert 0 in den Bits 4 und 5 des Steuerregisters erreichen. Unterlassen Sie diese Maßnahme, so kann es unter Umständen sein, daß der Zählerstand beim Lesen des nieder- und höherwertigen Bytes unterschiedlich ist, weil der Zähler während dem Lesevorgang weiterarbeitet.

Möchten Sie, daß der Lautsprecher Ihres PC mit Hilfe der im Zähler von Kanal 2 erzeugten Rechteckwellen einen Ton von sich gibt, so steuern Sie den Kanal B des 8255 an und setzen dessen Bits PB0 und PB1 für die gewünschte Dauer des Tons auf 1.

Wie Sie sehen, ist die Programmierung eines Bausteins im PC nicht so kompliziert, wie es auf den ersten Blick scheint.

Experimentieren Sie ein wenig mit Kanal 2, um verschiedene Töne auszugeben oder verstellen Sie die Uhrzeit über Kanal 0. Nur von Kanal 1 sollten Sie die Finger lassen, denn ein fehlerhaftes Refresh-Signal führt unweigerlich zum Paritätsfehler beziehungsweise zum Absturz des Computers.

(Alfred Kurz/ma)

Unterbrechungen unter Kontrolle mit dem Interrupt-Controller 8259

Trotz der hohen Verarbeitungsgeschwindigkeit ist die Verwaltung der vielen Peripherieeinheiten für den Prozessor eine fast unlösbare Aufgabe. Hier hilft der Interrupt-Controller 8259, der die Unterbrechungsanforderungen der Peripherie abfängt, bearbeitet und für den Prozessor sortiert.

Aus verständlichen Gründen können die verschiedenen Bausteine und Peripheriegeräte des PC nicht alle zugleich auf den Prozessor zugreifen und mit ihm kommunizieren. Vielmehr müssen diese Einheiten zuerst eine Anforderung an den Prozessor stellen, damit er seine gegenwärtige Tätigkeit unterbricht. Bei diesem Vorgang handelt es sich um eine sogenannte Interrupt-Anforderung, die über ein Interruptsignal ausgelöst wird.

Der Interrupt-Controller 8259 fängt im PC diese Interruptsignale ab. Er ist praktisch der Sekretär des Prozessors, der seinen Chef nur bei wichtigen Angelegenheiten stört. Der 8259 untersucht zuerst die Priorität des Interruptsignals, die von Stufe 0 bis Stufe 7 reichen kann. Für jede Stufe besitzt der 8259 einen eigenen Anschluß (IR0 bis IR7). Jedem Interruptsignal ist also die Stufe zugeordnet, die der zugehörige Controller-Eingang aufweist. Ein Signal am Eingang IR0 besitzt die höchste und ein Signal am Eingang IR7 die niedrigste Priorität. In dieser Reihenfolge teilt der 8259 nun den Geräten, die eine Interruptanforderung stellen, den Zugriff auf den Systembus und damit auf den Prozessor zu. Dieses Verfahren hat zur Folge, daß der Prozessor des PC die angeschlossene Peripherie nicht ständig abfragen muß, und sich ganz der Bearbeitung des laufenden Programms widmen kann.

Der 8259 behält für den Prozessor das Computersystem »im Auge« und entscheidet, zu welchen Zeitpunkten ein anderer Baustein oder ein Gerät Zugang zum Prozessor erhält. Dies ist immer dann der Fall, wenn der Controller durch eine Anweisung des Prozessors weiß, daß er einen gerade laufenden Prozeß für kurze Zeit unterbrechen darf.

Der Zugriff auf den 8259 selbst erfolgt über eine Interruptmaske, die in ein speziell für diese Zwecke reserviertes Register abgelegt wird. Jedes Bit dieser Maske ist entsprechend seiner Wertigkeit für einen Interrupteingang zuständig. So maskiert das Bit 0 den Eingang IR0 und Bit 7 den Eingang IR7. Sobald ein Bit der Inter-

ruptmaske auf 1 gesetzt ist, wird der zugehörige Interrupteingang gesperrt und der angeschlossene Baustein oder das angeschlossene Gerät kann keine Interruptanforderung mehr stellen. Man sagt dann, der Controller ist »maskiert«.

Sobald der 8259 eine Anforderung zuläßt, legt er die Eingangsleitung INT (Interrupt) des Prozessors auf 1 und macht dadurch den Prozessor auf die Anforderung eines Bausteins oder Gerätes aufmerksam. Läßt der Prozessor die Anfrage zu, so veranlaßt er den Buskontroller 8288 die Leitung INTA (Interrupt-Erkennung) zu aktivieren, womit der Interrupt-Controller die Bestätigung zur Freigabe der Interrupt-Anforderung erhält. Erst jetzt legt der 8259 einen 8-Bit-Interruptvektor auf den Datenbus, der bei der Initialisierung des Computers speziell für einen Interrupt dieser Priorität vom Prozessor in den Controller geschrieben wurde.

Der Prozessor greift den Interruptvektor auf, multipliziert ihn mit 4 und liest unter der Adresse, die sich aus dieser Multiplikation ergibt, einen 4 Byte langen Interruptvektor aus dem unteren KByte des Arbeitsspeichers. Darauf sichert der Prozessor das Statusbit-Register auf dem Stapel und ruft über eine Intersegment-Adressierung die in dem Interruptvektor angegebene Adresse auf. Hier befindet sich in aller Regel eine Interrupt-Serviceroutine, die die Interrupt-Anforderung bedient und beispielsweise Zeichen an einen Drucker sendet oder Werte von der Tastatur übernimmt.

Ein solch komplexer Hardware-Interrupt ist knapp ausgedrückt der asynchrone Aufruf eines Unterprogramms. Man nennt den Aufruf asynchron, weil er zu jeder beliebigen Zeit erfolgen kann und nicht von einem festen Taktschema abhängig ist. Die Interrupt-Serviceroutine läßt sich als Unterprogramm bezeichnen, weil sich der Prozessor nach korrekter Beendigung einer Interrupt-Serviceroutine wieder an der Position in einem Programm befindet, an der er das Programm verlassen hat. Programmtechnisch besteht damit zwischen einem Unterprogramm und einer Interrupt-Serviceroutine kein Unterschied.

Um den Interrupt-Controller 8259 direkt steuern zu können, stehen dem Programmierer unter den Portadressen 020hex und 021hex die Einsprunghstellen zu den einzelnen Registern zur Verfügung. Weil die Programmierung der einzelnen Register

sehr komplex ist, sei der interessierte Leser für detaillierte Informationen auf das Datenblatt zum 8259 verwiesen.

Die acht Interrupteingänge IR0 bis IR7 des 8259 sind im PC wie folgt verteilt:

IR0 : Takt der Systemuhr
IR1 : Tastatur-Interrupt
IR2 : Grafikkarte
IR3 : Zweite serielle Schnittstelle
IR4 : Erste serielle Schnittstelle
IR5 : Festplatten-Controller
IR6 : Disketten-Controller
IR7 : Parallele Schnittstelle

Der erste Eingang des 8259 ist mit der Systemuhr beschaltet. 18,2 Mal pro Sekunde erzeugt die Systemuhr des PC ein Taktsignal, das wiederum eine Interrupt-Anforderung stellt. Dieses Taktsignal hat die höchste Priorität, denn davon leitet sich die Uhrzeit ab, und die Bedienung des Ticker-Interrupts 1C erfolgt ebenfalls über dieses Signal. Würde der Takt der Systemuhr auch nur ab und zu durch einen gesperrten Interrupt »verschluckt«, so könnte die Uhrzeit nicht mehr zuverlässig angezeigt werden.

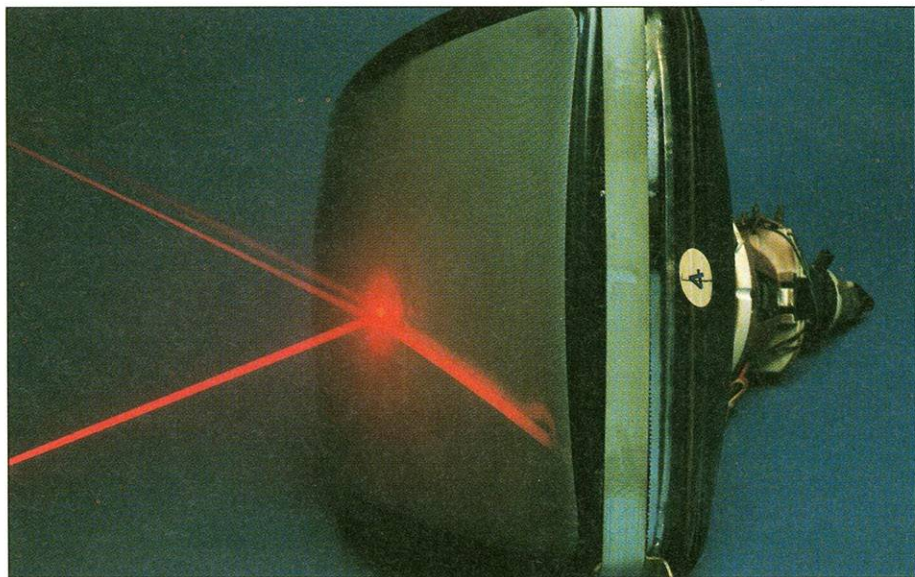
Am zweiten Eingang des Interrupt-Controllers ist die Tastatur angeschlossen. Sobald Sie auch nur eine einzige Taste kurz drücken, erzeugt die Tastatur eine Interruptanforderung, um dem Prozessor mitzuteilen, daß eine Taste gedrückt wurde und diese ausgewertet werden muß. Der dritte Eingang des 8259 ist für die Interruptanforderung einer Grafikkarte reserviert.

Über den vierten Eingang des Interrupt-Controllers kann die zweite serielle Schnittstelle des PC – sofern vorhanden – eine Interruptanforderung stellen und damit beispielsweise mitteilen, daß ein angeschlossenes Modem Daten senden möchte. Der fünfte Eingang des 8259 ist für die erste serielle Schnittstelle vorgesehen. Auch diese kann über eine Interruptanforderung anzeigen, daß Daten empfangen werden oder Sendebereitschaft besteht.

Eingang sechs und sieben des Interrupt-Controllers sind für den Anschluß eines Disketten- und Festplatten-Controllers vorgesehen, so daß auch diese Einheiten über eine Interruptanforderung auf sich aufmerksam machen können. Der achte und letzte Eingang ist schließlich für die parallele Druckerschnittstelle reserviert, die beispielsweise über eine Interrupt-Anforderung anzeigen kann, daß der angeschlossene Drucker den Ausdruck des Textes beendet hat und der nächste Text bearbeitet werden kann.

(Alfred Kurz/ma)

Grundlagen von Grafik und Text beim PC



Was wäre der PC ohne Bildschirm? Diese – zugegebenermaßen etwas »dumme« Frage – macht klar, wie wichtig die optische Anzeige ist. Hier lesen Sie, wie die Ausgabe auf dem Bildschirm grundsätzlich funktioniert, welche Hardware- und Software-Komponenten daran beteiligt sind und wie diese Komponenten zusammenarbeiten. Nebenbei erfahren Sie noch einiges über das oft übersehene DOS-Zusatzprogramm »ansi.sys«.

Es sind drei wichtige Teile des PC, die für die Bilderzeugung zuständig sind: der Monitor, die Steuerlogik und der Bildschirmspeicher. Diese Teile sind im Bild 1 schematisch dargestellt. Die Beziehungen zwischen den drei Hardwarekomponenten und deren Beziehung zur Software sind im Bild als Pfeile zu sehen.

Das Bild verdeutlicht, daß ein Programm grundsätzlich zwei verschiedene Arten der Bildschirmausgabe kennt. Entweder wird es direkt in den Bildschirmspeicher geschrieben oder ein Kommando zur Steuerlogik geschickt. Die Steuerlogik benutzt die Daten im Bildschirmspeicher, um das Signal für den Monitor zu erzeugen. Am Ende des Pfades wandelt der Monitor das elektrische Signal in Licht um.

Der Monitor ist zwar nicht programmierbar, aber trotzdem eine wichtige Komponente der Bildschirmhardware. Erstens bestimmt er zwei unübersehbare Merkmale des Bildes, die Schärfe und die Ein- oder Mehrfarbigkeit, zweitens ist die Qualität des Monitors maßgebend für die Qualität des Bildes. Die schönste Grafik sieht auf einem schlechten Monitor miserabel aus. Deshalb ist die Wahl des Monitors eine sehr wichtige Entscheidung. Leider sagt der Preis des Monitors relativ wenig über seine Qualität aus. Es gibt sehr günstige, sehr gute und auch teure, schlechte Monitore. Der sicherste Weg heißt »erst ansehen, dann kaufen.«

Die Bildschirmsteuerlogik und der Bildschirmspeicher sind üblicherweise zusammen auf dem Bildschirm-Adapter untergebracht. Im Bild 2 sehen Sie ein Beispiel für den

Aufbau der Steuerlogik. Ihre Aufgabe ist es, die digitalen Daten aus dem Bildschirmspeicher zu lesen, die Zeichen in Punktmuster zu übersetzen und die digitalen Punkte in ein analoges Signal für den Monitor umzuwandeln. Der zentrale Baustein, genannt Video-Controller, bestimmt die Fähigkeiten der Steuerlogik. Der Video-Controller holt die Daten aus dem Bildschirmspeicher und steuert die Arbeiten der anderen Komponenten der Steuerlogik. Der Zeichengenerator wandelt ein Zeichen aus dem ASCII-Format in ein Punktmuster um, das Schiebe-Register gibt die Punkte in der korrekten Reihenfolge an den Signal-Generator weiter und der Signal-Generator bekommt Synchronisierungs-Information vom Video-Controller und Punkte vom Schiebe-Register und wandelt diese ins Monitorsignal um.

Das alles passiert mit einer beeindruckenden Geschwindigkeit. Der übliche Bildschirm setzt sich aus 640×200 oder insgesamt 128000 Punkten zusammen. Das ganze Bild wird (mindestens) 50 Mal in der Sekunde ausgegeben, was 128000×50 oder 6,4 Millionen Punkte pro Sekunde ergibt. Nicht eingerechnet sind die Zwischenzeiten, während denen der Bildschirm kurzfristig abgeschaltet wird, weil der Elektronenstrahl neu ausgerichtet werden muß. Alles in allem werden die Punkte von der Steuerlogik zum Bildschirm mit einer Geschwindigkeit von über 10 Millionen Punkten pro Sekunde übertragen!

Die letzte Hardwarekomponente der Bilderzeugung ist der Bildschirmspeicher. Wie bereits erwähnt, wer-

den die digitalen Bilddaten hier abgelegt. Der Bildschirmspeicher wird wie der normale Datenspeicher angesprochen, das heißt, ein Programm kann in diesen Speicherbereichen ohne weiteres lesen und schreiben. Um Verwechslungen auszuschließen, liegt der Bildschirmspeicher an einer Adresse, die völlig vom normalen Programm- und Datenspeicher getrennt ist. Die meisten Bildschirm-Adapter verwenden den Bildschirmspeicher in zwei unterschiedlichen Arten, je nachdem ob Text- oder Grafikdarstellung gefordert ist. In beiden Fällen ist der Inhalt des Bildschirmspeichers eine einfache Abbildung des Bildschirms, nur die Form der Abbildung wird verändert. Text wird im Bildschirmspeicher als Folge einzelner ASCII-Zeichen abgelegt, wobei hinter jedem Zeichen noch ein sogenanntes Attribut-Byte folgt, das besondere Zeichenattribute wie Farbe oder blinkende Darstellung angibt.

Die Position eines Zeichens im Bildschirmspeicher bestimmt die Position des Zeichens auf dem Bildschirm, wie im Bild 3 deutlich wird. Die Steuerlogik wird so programmiert, daß die ASCII-Zeichen durch den sogenannten »Zeichengenerator« in Punktmuster umgewandelt werden. Im Bildschirmspeicher werden auch die Attribute der Zeichen abgelegt.

Diese Attribute bestimmen die Farbe und andere Eigenschaften eines Zeichens. Bild 4 zeigt schematisch die Abbildung des Schirms im Bildschirmspeicher bei der Grafikdarstellung. Statt Zeichen werden nun einzelne Punkte im Bildschirmspeicher abgelegt. Wie beim Text bestimmt die

Lage eines Punkts im Bildschirmspeicher dessen Position auf dem Bild. In dieser etwas vereinfachten Darstellung wird abhängig von der Anzahl Farben, die gleichzeitig gezeigt werden können, ein Punkt in einem, zwei oder mehreren Bits gespeichert. Die Reihenfolge der Punkte bleibt aber in allen Fällen gleich.

Der »weiche« Teil der optischen Wiedergabe – Software steuert den Bildschirm

Fast alle Programme geben Zeichen auf den Bildschirm aus. Zur Bildschirmausgabe stehen vier Wege zur Verfügung. Diese Möglichkeiten werden jetzt kurz beschrieben und verglichen, detaillierte Beschreibungen finden Sie bei den folgenden Artikeln über die einzelnen Bildschirm-Adapter, da die Funktionen, Adressen etc. von Adapter zu Adapter etwas differieren.

Der einfachste Weg, ein Zeichen auszugeben, ist die Verwendung einer entsprechenden Funktion einer höheren Programmiersprache – sofern Sie mit einer solchen Sprache arbeiten. Die Ausgabe auf dieser Ebene bietet einige Vorteile. Erstens bleibt die Ausgabe – soweit die Sprache auf anderen Rechnern implementiert ist – portabel. Zweitens ist eine Funktion einer höheren Programmiersprache leicht verständlich, was sowohl das ursprüngliche Schreiben als auch das spätere Ändern des Programms wesentlich erleichtert. Drittens fängt das Übersetzungsprogramm für die Sprache (zum Beispiel der Turbo Pascal-Compiler oder der Basic-Interpreter) alle gravierenden Fehler in der Ausgabe ab.

Die höheren Sprachen weisen aber auch zwei gravierende Nachteile auf. Einmal steht hier die mangelnde Geschwindigkeit. Insbesondere bei der Ausgabe von Grafiken sind die Funktionen der meisten höheren Programmiersprachen wesentlich langsamer als andere Methoden. Der zweite Nachteil ist die unvollständige Abbildung der Grafikfähigkeiten des PC, die die meisten höheren Sprachen anbieten. Funktionen, die sich auf einer tieferen Ebene höchst einfach gestalten, sind manchmal von einer höheren Sprache aus schwierig oder völlig unmöglich.

Eine andere Art der Zeichenausgabe bietet MS-DOS. Die MS-DOS-Ausgabe-Funktionen sind leider sehr primitiv und dazu sehr langsam. Man kann nur einfachen Text ausgeben, auf Farben oder irgendwelche anderen Attribute muß man hierbei verzichten. Es ist nicht einmal möglich, den

Cursor auf eine gewünschte Stelle zu positionieren. Die MS-DOS-Funktionen werden hauptsächlich in Utilities verwendet. Dies verdeutlicht auch den Vorteil dieser Funktionen: Wo MS-DOS ist, sind auch diese Funktionen dabei. Sie sind Hardware- und Sprachen-unabhängig, weshalb sich auch die verschiedensten kommerziellen Programme ihrer bedienen. Für die eigene Programmierung sind sie jedoch uninteressant.

Die letzte Softwarekomponente, das BIOS, bietet schon etwas mehr. Text kann man über das BIOS mit Attributen ausgeben. Außerdem läßt sich der Bildschirm-Modus bestimmen und abfragen, der Cursor positionieren und einige andere Hilfsfunktionen mehr stehen hierbei zur Verfügung. Zusätzlich gibt es Funktionen zum Setzen und Abfragen eines einzelnen Punktes, das heißt demnach: Das BIOS ist grafikfähig, wenn auch nur auf einer relativ primitiven Ebene. Die Routinen sind verhältnismäßig schnell und stellen alle notwendigen Funktionen zur Verfügung. Zudem sind Programme, die den Bildschirm ausschließlich über das BIOS steuern, immer noch Hardware-unabhängig. Aber auch die BIOS-Routinen haben zwei Nachteile. Erstens sind die Aufrufe von der Syntax her etwas zu umständlich und schwierig, um sie von einer höheren Sprache aus leicht zu benutzen. Zweitens sind wirklich nur die Basisfunktionen vorhanden. Das heißt, komplexe Aufgaben wie zum Beispiel eine Linie zu ziehen oder einen Kreis zu zeichnen muß man selbst programmieren.

Der letzte Weg zum Bildschirm ist bei weitem der schnellste. Wie oben kurz erwähnt wurde, kann man auch direkt in den Bildschirmspeicher

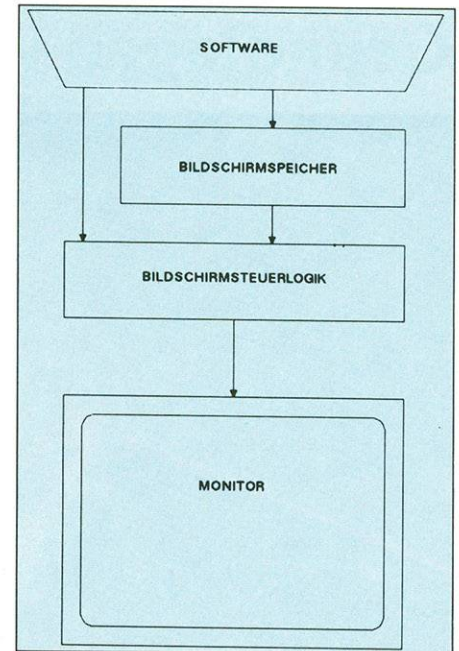


Bild 1: Hardware-Komponenten der Bildschirmsteuerung

schreiben. Hier ist jedoch Eigeninitiative gefragt – man muß nämlich alles selbst machen: die Adresse im Bildschirmspeicher berechnen, das Bit-Muster für ein Zeichen oder einen Punkt zusammenstellen und so weiter. Außerdem gibt es keine Garantie, daß ein anderer oder neuerer PC den gleichen Speicher in der gleichen Weise verwenden wird. Diese Methode ist also nicht unabhängig von der Hardware. Die Vorteile sind jedoch schlagend: Das Schreiben in den Speicher läuft erstens sehr schnell ab und zweitens hat man mit dieser Methode die absolute Kontrolle über die Ausgabe.

Die verschiedenen Methoden der Bildschirmausgabe und deren Vor- und Nachteile sind in Tabelle 1 zusammengefaßt.

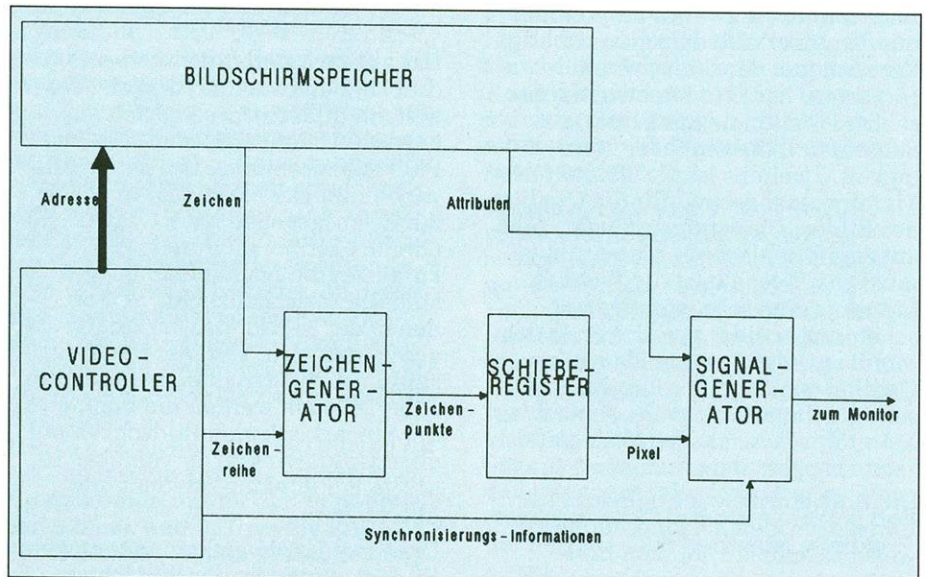


Bild 2. Die Bildschirmsteuerlogik

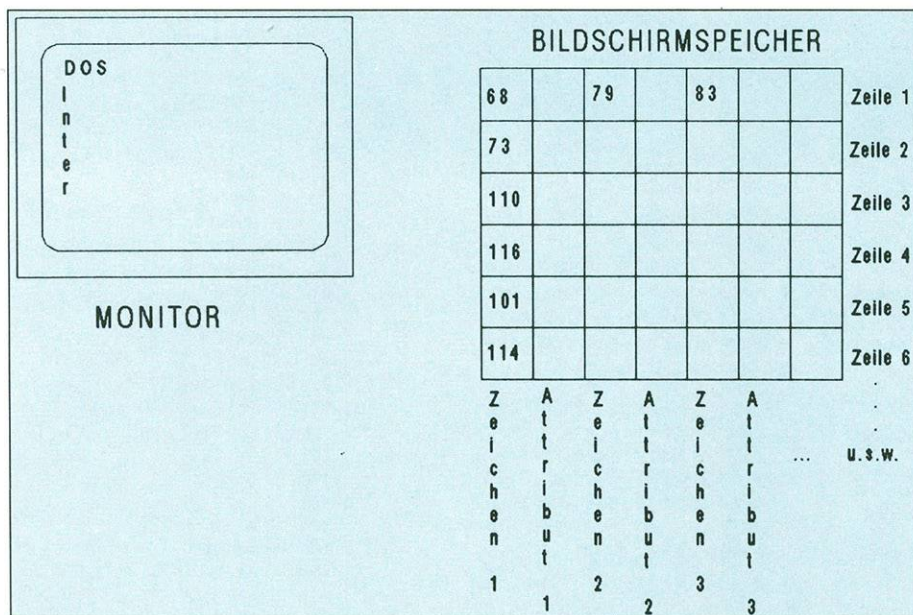


Bild 3: Beziehung zwischen Speicher und Bildschirm im Text-Modus

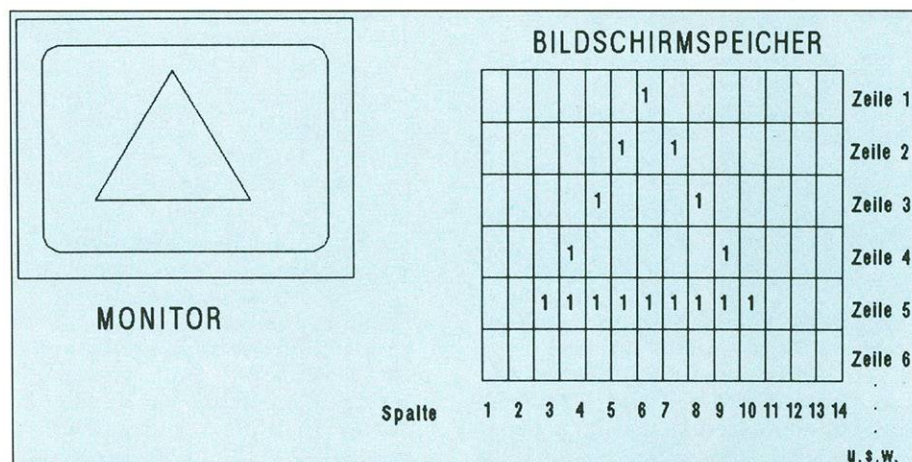


Bild 4: Beziehung zwischen Speicher und Bildschirm im Grafik-Modus.

Da MS-DOS selbst nicht viel im Bereich Bildschirmsteuerung bietet, haben die Erfinder des DOS ein zusätzliches Programm entwickelt, das viele der fehlenden Funktionen enthält. Dieses Programm heißt »ansi.sys« und wird mit jeder MS-DOS-Version ab 2.0 mitgeliefert. »ansi.sys« ist ein sogenannter Gerätetreiber, also ein Programm, das ein Ein- oder Ausgabegerät steuert. Dieses Programm erledigt beides, sowohl die Eingabe von der Tastatur als auch die Ausgabe zum Bildschirm. »ansi.sys« stellt nicht alle im BIOS möglichen Bildschirmfunktionen zur Verfügung, aber immerhin bringt es etwa 15 neue Steuerfunktionen mit. Bevor man »ansi.sys« benutzen kann, muß das Programm geladen und MS-DOS bekanntgegeben werden. Dies geschieht automatisch beim Starten des PC, wenn die Datei »config.sys« die Zeile

device=ansi.sys

beinhaltet. Bei einem älteren PC ist dies überhaupt der einzige Weg, um

den Ansi-Treiber im Betrieb zu nehmen. Manche neuere PCs bieten auch ein DOS-Kommando »ansi« an, womit der Ansi-Treiber nachträglich geladen werden kann.

Gleichgültig wie das Laden stattfindet, nachdem »ansi« einmal installiert wurde, bleibt es immer im Speicher und überwacht ständig die Ta-

statureingabe und die Bildschirmausgabe. »ansi.sys« arbeitet mit speziellen Zeichenfolgen, die nicht ausgegeben, sondern als Bildschirm-Steuerbefehle ausgeführt werden. Die Zeichenfolgen heißen »Escape-Sequenzen«, weil das erste Zeichen jeder Folge das Escape-Zeichen (ESC, ASCII-Wert 27) ist. Um die Folgen eindeutig zu identifizieren, ist das zweite Zeichen immer die eckige Klammer »[« (ASCII-Wert 91). Die restlichen Zeichen bestimmen den spezifischen Bildschirm-Steuerbefehl. Wichtig ist, daß in den Befehlen zwischen Groß- und Kleinbuchstaben unterschieden wird.

Die einzelnen Funktionen des Ansi-Treibers lassen sich in vier Gruppen aufteilen. Diese vier funktionellen Gruppen sind Löschen, Farbgestaltung/Attribute, Cursor-Steuerung und schließlich die Tastatur-Steuerung. Tabelle 2 gibt einen Überblick über die möglichen Funktionen. Beachten Sie, daß »ESC« in dieser Tabelle für das Esc-Zeichen (ASCII-Wert 27) steht und nicht die drei Zeichen »E«, »S« und »C« gemeint sind. Die Parameter, die in der Tabelle symbolisch (»#«) angegeben sind, werden in den folgenden Absätzen erklärt. Es existieren zwei Löschbefehle im Befehlssatz von »ansi.sys«. Der erste Befehl löscht den ganzen Bildschirm und bringt den Cursor in die obere, linke Ecke des Bildschirms. Dieser Befehl besteht aus den beiden Vorspannzeichen »ESC« und »[« und den beiden weiteren Zeichen »2« und »J«. Der zweite Befehl dieser Gruppe löscht eine Zeile von der aktuellen Cursorposition an bis ans Ende der Zeile. Dieser Befehl besteht aus dem Vorspann plus dem Buchstaben »K«.

In der Gruppe »Farbe und Attribute« gibt es auch zwei Befehle. Der erste setzt den Bildschirm-Modus, der wiederum die Eigenschaften der Bildschirmanzeige bestimmt. Weil dieser Befehl weitgehend dem MS-DOS-Befehl »Mode« entspricht, gehen wir

Software-Komponente	Vorteile	Nachteile
Höhere Sprache	portabel Hardware-unabhängig Fehler-Prüfung leicht verständlich viele Funktionen	langsam nicht vollständig umständlich
DOS-Routinen	immer verfügbar Hardware-unabhängig Fehler-Prüfung	primitiv langsam sehr wenig Funktionen keine Grafik
BIOS-Routinen	vollständig schnell Hardware-unabhängig	schwierig zu benutzen wenig Funktionen
Direkte Zugriffe	sehr schnell sehr flexibel	Hardware-abhängig keine Funktionen

Tabelle 1. Vor- und Nachteile der verschiedenen Software-Komponenten der Bildschirmsteuerung

Befehlssequenz	Funktion
ESC [2J ESC [K	Bildschirm löschen Von Cursor-Position bis Zeilenende löschen
ESC [= #h ESC [#;...;#m	Bildschirm-Modus setzen (siehe unten) Farben und Attribute setzen (siehe unten)
ESC [A ESC [B ESC [C ESC [D ESC [#;#H ESC [s ESC [u ESC [6n	Cursor eine Zeile nach oben Cursor eine Zeile nach unten Cursor eine Spalte nach rechts Cursor eine Spalte nach links Cursorposition setzen (siehe unten) Cursorposition speichern Cursor auf gespeicherter Position setzen Cursorposition abfragen
ESC [#;#;"String";#p	Taste neu definieren (siehe unten)
Parameter für die einzelnen Befehle:	
Befehlssequenz	Parameter
ESC [#h	# ist der Bildschirm-Modus: 0: Text, 40 x 25 Zeichen, schwarz/weiß 1: Text, 40 x 25 Zeichen, farbig 2: Text, 80 x 25 Zeichen, schwarz/weiß 3: Text, 80 x 25 Zeichen, farbig 4: Grafik, 320 x 200 Punkte, farbig 5: Grafik, 320 x 200 Punkte, schwarz/weiß 6: Grafik, 640 x 200 Punkte, schwarz/weiß
ESC [#;...;#m	# ist die Farbe/das Attribut: 0: Attribute aus (normal schwarz/weiß) 1: Aufgehell 5: Blinkend 7: Reverse-Video 8: Unsichtbar 30: Vordergrund schwarz 31: Vordergrund rot 32: Vordergrund grün 33: Vordergrund gelb 34: Vordergrund blau 35: Vordergrund magenta 36: Vordergrund zyan 37: Vordergrund weiß 40: Hintergrund schwarz 41: Hintergrund rot 42: Hintergrund grün 43: Hintergrund gelb 44: Hintergrund blau 45: Hintergrund magenta 46: Hintergrund zyan 47: Hintergrund weiß
ESC [#;#H	erstes # ist die Zeilennummer zweites # ist die Spaltennummer
ESC [#;#;"String";#p	#;# ist der Tastenschlüssel (siehe MS-DOS-Manual) »String« ist die neue Zeichenfolge

Tabelle 2. Die Befehle des Ansi-Treibers

Code	Sondertaste	Code	Sondertaste
3	Ctrl-2	85	Shift-F2
15	Shift-Tab	86	Shift-F3
16	Alt-Q	87	Shift-F4
17	Alt-W	88	Shift-F5
18	Alt-E	89	Shift-F6
19	Alt-R	90	Shift-F7
20	Alt-T	91	Shift-F8
21	Alt-Y	92	Shift-F9
22	Alt-U	93	Shift-F10
23	Alt-I	94	Ctrl-F1
24	Alt-O	95	Ctrl-F2
25	Alt-P	96	Ctrl-F3
30	Alt-A	97	Ctrl-F4
31	Alt-S	98	Ctrl-F5
32	Alt-D	99	Ctrl-F6
33	Alt-F	100	Ctrl-F7
34	Alt-G	101	Ctrl-F8
35	Alt-H	102	Ctrl-F9
36	Alt-J	103	Ctrl-F10
37	Alt-K	104	Alt-F1
38	Alt-L	105	Alt-F2
44	Alt-Z	106	Alt-F3
45	Alt-X	107	Alt-F4
46	Alt-C	108	Alt-F5
47	Alt-V	109	Alt-F6
48	Alt-B	110	Alt-F7
49	Alt-N	111	Alt-F8
50	Alt-M	112	Alt-F9
59	F1	113	Alt-F10
60	F2	114	Ctrl-PrtSc
61	F3	115	Ctrl-Cursor-links
62	F4	116	Ctrl-Cursor-rechts
63	F5	117	Ctrl-End
64	F6	118	Ctrl-PgDn
65	F7	119	Ctrl-Home
66	F8	120	Alt-1
67	F9	121	Alt-2
68	F10	122	Alt-3
71	Home	123	Alt-4
72	Cursor oben	124	Alt-5
73	PgUp	125	Alt-6
75	Cursor links	126	Alt-7
77	Cursor rechts	127	Alt-8
79	End	128	Alt-9
80	Cursor unten	129	Alt-0
81	PgDn	130	Alt-B
82	Ins	131	Alt-'
83	Del	132	Ctrl-PgUp
84	Shift-F1		

Tabelle 3. Die Sondertasten-Codes der PCs

hier nicht weiter ins Detail. Die notwendigen Informationen können Sie der Tabelle entnehmen. Der zweite Befehl dieser Gruppe ist vielleicht der nützlichste von allen »ansi.sys«-Befehlen. Damit lassen sich die Farben des Vorder- und Hintergrunds und die anderen Attribute eines Zeichens bestimmen. Nach dem Vorspann »ESC [« steht eine Liste von Attributen, getrennt durch Strichpunkte. Diese Attribute werden als Zahlen verschlüsselt, wie im zweiten Teil der Tabelle zu sehen ist. Man darf die Attribute in einem Befehl kombinieren: Zum Beispiel bewirkt die Folge

ESC (1;5;32m

daß die Zeichen grün (Attribut 32), aufgehellt (Attribut 1) und blinkend (Attribut 5) gezeigt werden. Die Hintergrundfarbe ändert sich durch diese

Folge nicht. Die Eigenschaften der Anzeige, die dieser Befehl aktiviert, bleiben solange gültig, bis ein neuer Befehl die Eigenschaften ändert. Zum Beispiel wird die Sequenz

ESC (1;37;44m

jede weitere Ausgabe in weiß auf hellblau anzeigen. Wenn zu einem späteren Zeitpunkt der Befehl

ESC (46m

erfolgt, wird der Hintergrund auf zyan umgestellt, jedoch bleiben die anderen Eigenschaften unverändert. Die größte Gruppe der Ansi-Befehle steuert den Cursor. Die ersten vier bewegen den Cursor um je eine Stelle in die vier verschiedenen Richtungen oben und unten, links und rechts. Wenn der Cursor bereits am entsprechenden Bildschirmrand steht, wird er in allen vier Fällen nicht weiter bewegt. Der nächsten Befehl versetzt

den Cursor an eine beliebige Stelle auf dem Bildschirm. In der Tabelle sieht man, daß dieser Befehl zwei Parameter (Zeile und Spalte) benötigt. Die obere, linke Ecke des Bildschirms ist Zeile 1, Spalte 1. Die beiden Parameter sind einfach als ASCII-Ziffern (getrennt durch einen Strichpunkt) zu übergeben. Zum Beispiel setzt

ESC (13;8H

den Cursor in Zeile 13, Spalte 8. Die nächsten beiden Befehle werden als Paar benutzt. Der erste speichert die Cursorposition und der zweite setzt den Cursor wieder auf die bereits gespeicherte Position. Damit kann man die aktuelle Cursorposition speichern, den Cursor versetzen, eine Nachricht ausgeben und anschließend den Cursor auf die ursprüngliche Stelle zurückstellen. Zum Beispiel können Sie den MS-DOS-

Prompt verwenden, um den aktuellen Pfad auszugeben. Da der Name des Pfades manchmal recht lang ist, lassen Sie ihn doch einfach in der obersten Bildschirmzeile erscheinen mit der Befehlssequenz

```
prompt $e($e(1;30H$p$e(u$g
```

Dieser Befehl ist wie folgt zu verstehen: In einem Prompt-Befehl vertritt die Zeichenfolge »\$e« das ESC-Zeichen. Die ersten vier Zeichen »\$e[s« stehen dann für den Ansi-Befehl »speichere Cursorposition«. Die Folge »\$e[1;30H« setzt den Cursor in Zeile 1, Spalte 30, wo Sie den Namen des Pfades ausgeben wollen. »\$p« bewirkt, daß MS-DOS den Pfadnamen als Teil des System-Prompts ausgibt, und zwar dort, wohin Sie gerade den Cursor gesetzt haben. Die Folge »\$e[u« bringt den Cursor zurück an seine frühere Position und »\$g« schließlich veranlaßt, daß MS-DOS das bekannte »>«-Zeichen dort ausgibt.

Der letzte Ansi-Befehl der Cursor-Gruppe bewirkt, daß die aktuelle Cursorposition als eine Zeichenfolge von der Tastatur gelesen werden kann. Gibt man den Befehl

```
ESC (6n
aus, so werden die Zeichen
ESC [#;#R
```

als die nächsten Zeichen von der Tastatur gelesen. Dabei wird das erste #-Zeichen durch die Zeilennummer und das zweite durch die Spaltennummer ersetzt. Natürlich braucht der Anwender dafür keine Taste zu

betätigen, aber das laufende Programm kann so durch Lesen der Tastatur-Einheit die aktuelle Cursorposition feststellen.

Die letzte Gruppe der Ansi-Befehle besteht nur aus einem einzigen, allerdings sehr leistungsfähigen Befehl. Wie oben bereits erwähnt wurde, übernimmt »ansi.sys« die Eingabe von der Tastatur. Normalerweise unterscheidet sich die Eingabe über »ansi« nicht von der normalen Eingabe. Mit diesem Befehl aber kann man veranlassen, daß ein Tastendruck eine beliebige Zeichenfolge erzeugt. Im Listing finden Sie als Beispiel zehn solcher Ansi-Folgen in einer Stapeldatei zusammengefaßt, wobei jeweils Prompt-Befehle angewendet werden. Wenn Sie den Ansi-Treiber installiert haben, können Sie diese Stapeldatei dazu verwenden, die zehn Funktions-Tasten Ihres PC wie folgt umzuprogrammieren:

```
F1 -- dir/w
F2 -- more <
F3 -- dir
F4 -- type
F5 -- cd
F6 -- cd ..
F7 -- copy
F8 -- rename
F9 -- erase
F10 -- format
```

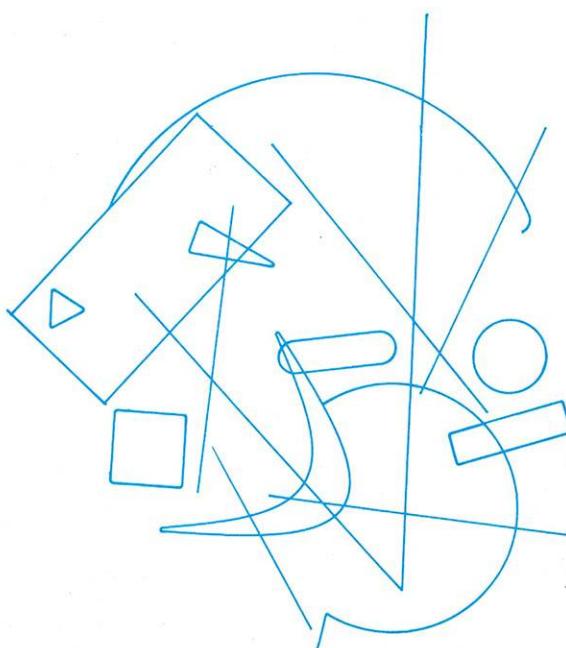
Natürlich können Sie die Befehle je nach Geschmack ändern. Dazu dient der erste Prompt-Befehl in Listing 1 als Beispiel. Dieser Befehl programmiert die Taste F1 um. Wie alle Ansi-Befehle beginnt er mit der Sequenz

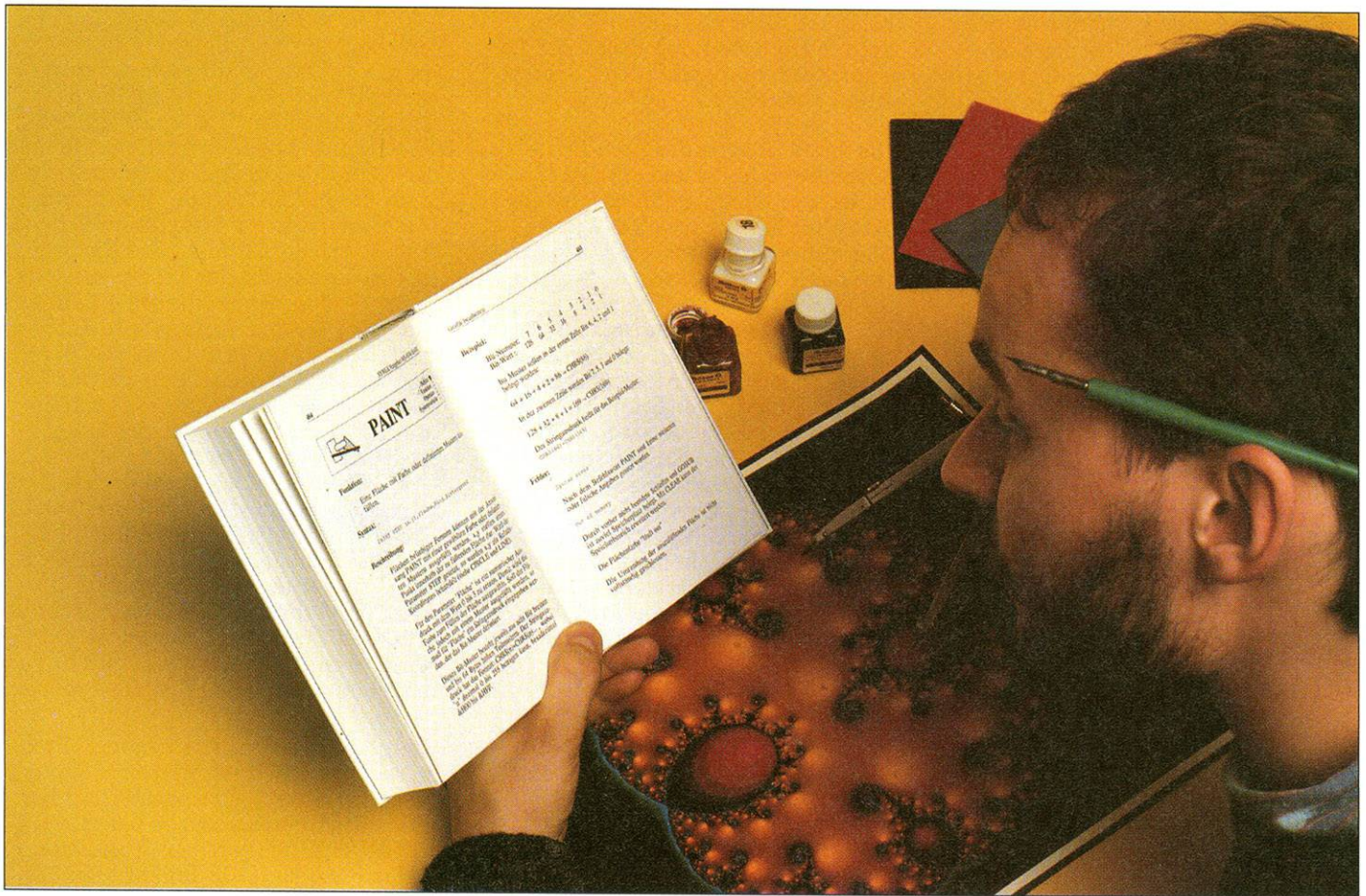
»ESC [«, wobei das ESC-Zeichen hier wie beim Prompt-Befehl üblich, durch die Zeichenfolge »\$e« repräsentiert wird. Danach folgt der Tastenschlüssel der Taste, die umdefiniert werden soll. Bei den normalen Schreibmaschinentasten ist dies einfach der ASCII-Wert des jeweiligen Zeichens, den Sie bitte Ihrem PC-Handbuch entnehmen. Für die Sondertasten besteht der Tastencode aus zwei Zahlen, wobei die erste Zahl immer eine Null und die zweite Zahl der sogenannte Scan-Code der jeweiligen Sondertaste ist, den Sie der Tabelle 3 entnehmen können. Im Fall der F1-Taste besteht der Schlüssel also aus den beiden Zahlen »0« und »59«. Dann folgt die gewünschte Zeichenfolge, die in Anführungszeichen stehen muß. Den Schluß bildet der Befehlsbuchstabe »p«. So ein Befehl braucht nur einmal ausgegeben werden, er bleibt dann bis zum Abschalten des PC wirksam. Der letzte Prompt-Befehl im Listing stellt einfach einen normalen System-Prompt wieder her.

Damit haben wir unsere beiden Ziele erreicht. Zum einen wissen Sie nun über die Arbeitsweise des PC im Bereich Bildschirmausgabe bescheid. Dieses Wissen dient als Grundlage für das Verständnis der anderen Beiträge in diesem Sonderheft, die sich mit spezifischen Grafiktechniken und Grafikkarten befassen. Zum anderen beherrschen Sie es jetzt, die Bildschirmausgabe auch von MS-DOS aus ein bißchen schöner und komfortabler zu gestalten.

(Thomas Little/ev)

```
0: rem Funktionstastenbelegung
1: rem ** DOS International **
2: echo off
3: prompt $e[0;59;"dir/w";13p
4: prompt $e[0;60;"more <"p
5: prompt $e[0;61;"dir "p
6: prompt $e[0;62;"type "p
7: prompt $e[0;63;"cd "p
8: prompt $e[0;64;"cd ..";13p
9: prompt $e[0;65;"copy "p
10: prompt $e[0;66;"rename "p
11: prompt $e[0;67;"erase "p
12: prompt $e[0;68;"format "p
13: prompt $p$g
```





Grafik-Programmierung in GW-Basic

GW-Basic verfügt über einen leistungsfähigen Grafikbefehlssatz, mit dem sich sehr einfach umfangreiche und komfortable Grafikanwendungen realisieren lassen. In diesem Beitrag erfahren Sie alles Wissenswerte über die grafischen Fähigkeiten der Programmiersprache GW-Basic.

Für die Bildschirmausgabe sind bei GW-Basic drei Grafikmodi vorgesehen. Der erste Modus (Modus 0) ist ausschließlich der Textausgabe vorbehalten. In diesem Modus werden Buchstaben, Zahlen und alle Sonderzeichen am Bildschirm dargestellt. Macht man sich die Mühe, mit den Grafiksymbolen des PC-Zeichensatzes eine Blockgrafik zu erzeugen, so kann man auch diesen Modus als Grafikmodus betrachten.

Standardmäßig erfolgt im Modus 0 die Textausgabe in weißer Schrift auf schwarzem Untergrund. Verfügen Sie über einen Farbbildschirm, so können Sie weitere 14 Farben (also insgesamt 16) benutzen. Zur Ansteuerung dieser 16 Farben verfügt GW-Basic über den Color-Befehl. Die-

ser Befehl hat folgende Syntax:
color zeichenfarbe,hintergrundfarbe,rahmenfarbe

Tabelle 1 zeigt die 16 Farben, die dem Benutzer bei diesem Befehl zu Verfügung stehen. Auf einem Monochrom-Monitor lassen sich natürlich nur die Helligkeitsstufen Dunkel, Hell und Leuchtend darstellen. Bei Verwendung der Farbnummern 1 und 9 werden die betreffenden Zeichen unterstrichen.

Der Parameter »zeichenfarbe« darf einen Wert von 0 bis 31 annehmen. Die Farbnummern 0 bis 15 entsprechen den Farben aus Tabelle 1. Wird der Wert einer Farbnummer um 16 erhöht, so erscheint das Zeichen in der ursprünglichen Farbe, jedoch blinkend.

Der Parameter »hintergrund« bestimmt die Hintergrundfarbe eines Zeichens und darf nur Werte zwischen 0 und 7 annehmen. Blinkmodus und die hellen Farben sind also nicht zugelassen. Bei einem Monochrom-Monitor sorgen die Farbnummern 0 bis 6 für schwarzen Hintergrund. Ist als Parameter der Wert 7 angegeben, so ist der Hintergrund weiß.

Der letzte Parameter des Color-Befehls definiert die Rahmenfarbe des Bildschirms. In diesem Fall sind wieder alle 16 Farben zugelassen. Der Blinkmodus entfällt. Es ist darauf zu achten, daß Zeichen- und Hintergrundfarbe nicht identisch sein dürfen, weil sonst alle gedruckten Zeichen unsichtbar bleiben. Wird beim Color-Befehl ein Parameter ausgelassen, so bleibt die voreingestellte Farbe bestehen.

Im Textmodus sind zwei Auflösungsstufen möglich. Abhängig vom Befehl »width« werden 40 oder 80 Zeichen pro Bildschirmzeile dargestellt. Die Position eines Textzeichens legen jeweils zwei Koordinaten fest. Die Zählweise dieser Koordinaten beginnt in der linken oberen Ecke (0,0) und endet in der rechten unteren Ecke (24,39 beziehungsweise 24,79) des Textbildschirms.

Wird der Bildschirm mittels Print- oder Write-Befehl beschrieben, so erfolgt die Textausgabe von oben nach unten. Erreicht der Cursor dabei das Ende der vorletzten Zeile, wird der Bildschirm nach oben gescrollt. Die letzte Zeile ist davon jedoch nicht betroffen. Diese Zeile behält jederzeit

ihre Position bei. In GW-Basic wird deshalb in der untersten Zeile die Belegung der Funktionstasten angezeigt.

Sollen die Funktionstasten nicht angezeigt werden, aber trotzdem aktiv bleiben, so läßt sich die Anzeige über den Befehl »key off« deaktivieren. Nun steht auch die letzte Zeile für den Programmierer zur Verfügung. Weil der Cursor jedoch nicht ohne weiteres in die letzte Zeile springen kann (ab der vorletzten Zeile wird nach oben weg gescrollt), muß der Cursor mit dem Locate-Befehl in die letzte Zeile gesetzt werden. Der Befehl

locate 25,1

setzt den Cursor in die erste Spalte der 25. Zeile. Nun läßt sich diese Zeile über den Print-Befehl beschreiben. Tabelle 2 gibt einen Einblick in die wichtigsten GW-Basic-Befehle für den Datenaustausch im Textmodus. Wenn Sie einen Farbmonitor besitzen, können Sie bei der Darstellung von 40 Zeichen pro Zeile 8 Bildschirmseiten benutzen. Im 80-Zeichen-Modus verringert sich die Anzahl auf 4 Seiten. Die einzelnen Bildschirmseiten lassen sich über den Screen-Befehl auswählen. Dieser Befehl hat folgende Syntax:

screen modus, farbe, aktuelle Seite, sichtbare Seite

Der Parameter »modus« (Tabelle 3) steht für den aktuellen Videomodus. Im Textmodus muß dieser Parameter 0 lauten. Der zweite Parameter »farbe« hat den Charakter einer booleschen Variablen. Ist »farbe« gleich 1 (logisch wahr), so ist eine Farbänderung zugelassen, das heißt, der Text darf farbig dargestellt werden. Wenn »farbe« dagegen gleich 0 (logisch falsch) ist, sind nur die Zeichenfarben Schwarz oder Weiß erlaubt.

Mit dem Parameter »aktuelle Seite« wird die zu bearbeitende Bildschirmseite ausgewählt. Diese muß nicht unbedingt die gerade sichtbare Bildschirmseite sein. Die sichtbare Seite bestimmt die letzte Parameter. Je nach Textmodus dürfen die letzten zwei Parameter nur Werte von 0 bis 7 (bei 40 Zeichen pro Zeile) oder von 0 bis 3 (bei 80 Zeichen pro Zeile) annehmen.

Die Grafik unter GW-Basic arbeitet mit einem mittelauflösenden und einem hochauflösenden Grafikmodus

Wenn über den Screen-Befehl die mittlere Grafikauflösung (Modus 1) ausgewählt wurde, stehen dem Programmierer 64000 Bildpunkte zur Verfügung. Dies entspricht einer Bildschirmauflösung von 320x200 Punkten (Bild 1). Die Zählweise der

Farbnummer	Farb-Monitor	Monochrom-Monitor
0	Schwarz	Dunkel
1	Blau	Unterstrichen
2	Grün	Hell
3	Kobaltblau	Hell
4	Rot	Hell
5	Violett	Hell
6	Braun	Hell
7	Grau	Hell
8	Grau	Dunkel
9	Hellblau	Leuchtend und unterstrichen
10	Hellgrün	Leuchtend
11	Hellkobaltblau	Leuchtend
12	Hellrot	Leuchtend
13	Hellviolett	Leuchtend
14	Gelb	Leuchtend
15	Weiß	Leuchtend

Tabelle 1. Die Farbnummern bestimmen die Farbe oder Helligkeit auf dem Bildschirm

Befehl	Erklärung
cls	Löscht den Textbildschirm (Bildschirmfenster)
color	Setzt die Farben für Zeichen, Hintergrund und Rahmen
csrlin	Gibt aktuelle Cursor-Zeile wieder
locate	Positioniert Cursor an beliebiger Stelle
pos	Gibt aktuelle Cursor-Spalte an
print	Gibt Zeichen am Bildschirm aus
screen	Setzt Attribute für Bildschirm beziehungsweise gibt Attribute zurück
spc	Gibt eine bestimmte Anzahl von Leerzeichen aus
tab	Setzt Cursor an Tabulator-Position
width	Schaltet auf 40- oder 80-Zeichen-Modus
write	Gibt Zeichen am Bildschirm aus
view print	Wählt Bildschirmfenster für Textausgabe

Tabelle 2. Die wichtigsten GW-Basic-Befehle für den Textmodus

Modus	Bedeutung
0	Textmodus mit 40 beziehungsweise 80 Zeichen pro Zeile
1	Grafikmodus mit einer Auflösung von 320 * 200 Bildpunkte (maximal 4 Farben)
2	Grafikmodus mit einer Auflösung von 640 * 200 Bildpunkte (maximal 2 Farben)

Tabelle 3. Die drei Ausgabemodi von GW-Basic

Kennziffer	Palette 0	Palette 1
0	Beliebige Hintergrundfarbe	
1	Grün	Kobaltblau
2	Rot	Violett
3	Braun	Weiß

Tabelle 4. Die beiden Farbpaletten des Color-Befehls

Zeichenkommando	Funktion
U n	Zeichnet n Bildpunkte nach oben
D n	Zeichnet n Bildpunkte nach unten
L n	Zeichnet n Bildpunkte nach links
R n	Zeichnet n Bildpunkte nach rechts
E n	Zeichnet n Bildpunkte nach rechts oben
F n	Zeichnet n Bildpunkte nach rechts unten
G n	Zeichnet n Bildpunkte nach links unten
H n	Zeichnet n Bildpunkte nach links oben
M x,y	Setzt Cursor auf Koordinate x,y
A n	Setzt einen rechten Winkel (0, 90, 180 oder 270)
TA n	Setzt beliebigen Winkel (-360 bis + 360)
C n	Setzt die Farbe n
S x	Der Skalenfaktor wird zwischen 1/4 und 255/4 festgelegt
X v\$	Eine weitere Zeichenkette wird ausgeführt
P n,m	Ein Objekt wird mit einer Farbe ausgefüllt

Tabelle 5. Der Zeichenkommandosatz des Draw-Befehls ist reichhaltig

Modus	Bedeutung
pset	Das Shape wird originalgetreu dargestellt
preset	Das Shape wird invertiert dargestellt
and	Das Shape wird mit dem Hintergrund über die Und-Funktion logisch verknüpft
or	Das Shape wird mit dem Hintergrund über die Oder-Funktion logisch verknüpft
xor	Das Shape wird mit dem Hintergrund über die Exklusiv-Oder-Funktion logisch verknüpft

Tabelle 6. Die Wiedergabe-Modi für ein Shape im Put-Befehl

Bildpunkte entspricht der des Textmodus. Der Bildpunkt in der linken oberen Ecke besitzt die Koordinate 0,0 und der rechte untere Bildpunkt die Koordinate 319,199.

Verwendet man den Color-Befehl im Grafikmodus, so hat dieser eine andere Syntax als im Textmodus:

color hintergrundfarbe, palette

Bei der Farbwahl mittels Color-Befehl läßt sich dem Hintergrund eine beliebige Farbnummer aus Tabelle 1 zuordnen. Der Parameter »palette« darf zwischen 0 und 255 liegen. Entscheidend sind jedoch nur zwei Zustände. Ist der Parameter eine gerade Zahl, so arbeitet die Grafik mit der Farbpalette 0, ist der Parameter ungerade, mit der Farbpalette 1. Jede Farbpalette enthält 3 Farben, die über die Kennziffern 1, 2 und 3 aufzurufen sind. Als Kennziffern können jedoch Werte von 0 bis 3 übergeben werden, wobei 0 die Hintergrundfarbe bezeichnet. Tabelle 4 zeigt, aus welchen Farben sich die beiden Farbpaletten zusammensetzen.

Da es nicht immer wünschenswert ist, ausschließlich mit den voreingestellten Farben der beiden Paletten zu arbeiten, besitzt GW-Basic den Befehl »palette«. Mit diesem Befehl lassen sich beliebige Farben einer Farbpalette, sowie die Hintergrundfarbe verändern. Der Palette-Befehl besitzt folgende Syntax:

palette kennziffer, farbnummer

Den Parameter »kennziffer« entnehmen Sie der Tabelle 4. Ist der Parameter gleich 0, so ändert der Palette-Befehl die Hintergrundfarbe. Der Parameter »farbnummer« darf -1 bis 15 betragen. Die Zuordnung zwischen Parameter und Farbe zeigt wieder die Tabelle 1.

Beträgt der Wert des Parameters »farbnummer« -1, dann wird die aktuelle Farbe nicht geändert. Sollen alle vier Farben einer Palette gleichzeitig geändert werden, muß dem Befehl »palette« das Schlüsselwort »using« angehängt werden. Die Syntax von »using« lautet:

palette using farbe

Bei dieser Version des Palette-Befehls muß der Parameter »farbe« eine Integer-Feldvariable sein, die die aufeinanderfolgenden Werte der vier Farben enthält. Das folgende kleine Programm ändert die aktuelle Farbpalette auf die Farben Rot, Grün und Blau und die Hintergrundfarbe auf Weiß:

```
10 data 7,4,2,1
20 for i=0 TO 3
30 read farbe%(i)
40 next i
50 palette using farbe%(0)
```

Wenn Sie eine Farbe aus der Palette oder der Hintergrundfarbe ändern, wechseln auch die zu dieser Farbe korrespondierenden Bildpunkte entsprechend ihre Farbe. Es können also niemals mehr als vier Farben gleich-

zeitig auf dem Bildschirm dargestellt werden.

Im Grafikmodus läßt sich auch Text darstellen, bei mittlerer Auflösung 40 Zeichen pro Zeile und bei hoher Auflösung 80 Zeichen pro Zeile.

Die hohe Grafikauflösung aktiviert der Wert 2 im Screen-Befehl. Dieser Grafikmodus arbeitet mit insgesamt 128000 Bildpunkten, was einer Bildschirmauflösung von 640x200 Bildpunkten entspricht (Bild 2).

Im hochauflösenden Grafikmodus lassen sich nur noch zwei Farben darstellen. Die Farbnummer 0 steht für Schwarz und Farbnummer 1 für Weiß. Die Ausgabe von Text erfolgt im hochauflösenden Grafikmodus mit 80 Zeichen pro Zeile. Kaum ein Programm kommt ganz ohne Grafik aus. Deshalb zeigen wir Ihnen im folgenden einige Verfahren zur Ausgabe von Grafiken.

Sehr häufig müssen in Anwendungsprogrammen Daten als Diagramme dargestellt werden. Wie sich diese Aufgabe mit wenigen Befehlen lösen läßt, zeigt Listing 1. In diesem Beispiel werden zehn numerische Werte in Prozentanteile umgerechnet und anschließend als Tortendiagramm dargestellt. Die Grafikausgabe erfolgt in Zeile 460 mit dem Befehl »circle«. Dieser Befehl ist sehr flexibel und eignet sich hervorragend zur Darstellung von Tortendiagrammen. Der Circle-Befehl hat folgende Syntax:

circle (x,y),radius,farbe,anfang,ende,verhältnis
Die Parameter »x« und »y« geben die Mittelpunktordinate des Kreises (beziehungsweise der Ellipse) an. Der Radius wird mit dem gleichnamigen Parameter übergeben. Im mittelaufösenden Grafikmodus läßt sich mit dem Parameter »farbe« eine Kennziffer von 0 bis 3 für eine Farbpalette angeben, und im hochauflösenden Modus über 0 oder 1 die Farbe Schwarz oder Weiß auswählen.

Die Parameter »anfang« und »ende« sind numerische Variablen, die den Anfangs- und Endwinkel des Kreisbogens festlegen. Beide Werte müssen im Bodenmaß angegeben werden. Die Umrechnung von Grad in Bodenmaß erfolgt durch die Formel $\text{Bogenmaß} = \text{Grad} * \pi / 180$

Wenn Sie negative Winkelangaben verwenden, verbindet der Circle-Befehl die Kreissegmente mit dem Mittelpunkt. Aus diesem Grund wurde auch der Anfangswert in unserem Beispielprogramm auf »-2 PI« gesetzt. Der letzte Parameter des Circle-Befehls gibt das Verhältnis der Koordinatenachsen an. Bei mittlerer Auflösung ist der Wert (5/6), bei hoher Auflösung der Wert (5/12). Durch diese Angaben werden die Ellipsen als Kreis am Bildschirm dargestellt. Eine Änderung der Parameter verzerrt den Kreis zu einer Ellipse. Zum Löschen des Grafikbildschirms

besitzt GW-Basic den Befehl »cls«. Dieser Befehl löscht jedoch nur den gerade sichtbaren Grafikbildschirm. Dabei wird der Cursor in die sogenannte »Home-Position« zurückgesetzt. Im Textmodus befindet sich diese Position in der linken oberen Bildschirmcke, im Grafikmodus liegt sie in der Mitte des Bildschirms (Koordinate 160/100 bei mittlerer Auflösung und Koordinate 320/100 bei hoher Auflösung).

Der Basic-Befehl »draw« ist aufgrund seiner zahlreichen Parameter sehr leistungsfähig

Der mit Abstand leistungsfähigste Grafikbefehl von GW-Basic ist »draw«. Mit diesem Befehl lassen sich auf einfache Weise komplexe Objekte zeichnen. Der Befehl setzt sich aus der Draw-Anweisung und einer Zeichenkette zusammen. Die dem Draw-Befehl folgende Zeichenkette versteht sogenannte Zeichenkommandos, ähnlich der Turtle-Grafik von Logo. Durch spezielle Befehle können alle wichtigen Grafik-Funktionen mit einer einzigen Zeichenkette ausgeführt werden. Tabelle 5 erklärt die wichtigsten Zeichenkommandos des Draw-Befehls.

Wird den neun Bewegungsbefehlen (»U«, »D«, »L«, »R«, »E«, »F«, »G«, »H« und »M«) ein »B« vorangestellt, so wird während der Cursorbewegung nicht gezeichnet. Durch Vorstellen eines »N« bleibt die Ausgangsposition der Bewegung trotz eines verschobenen Cursors erhalten. Bei der Eingabe eines rechten Winkels steht der Parameter »n« für folgende Werte:

n	Winkel
0	0 Grad
1	90 Grad
2	180 Grad
3	270 Grad

Das Zeichenkommando »M x,y« kennt zwei Ausführungsvariationen. Wird den Parametern »x« und »y« kein Vorzeichen vorangestellt, so beziehen sich die Angaben auf die absolute Koordinatenposition. Bei Angabe eines Vorzeichens werden die Punkte dagegen relativ zum letzten Endpunkt verschoben. In Listing 2 zeichnet der Draw-Befehl einen Quader. Durch Angabe der Schrittweite wird der Quader einmal um den Bild-Mittelpunkt gedreht. Bei einer Schrittweite größer als 360 wird der Quader nur ein einziges Mal dargestellt. Ein weiterer, leistungsfähiger Grafikbefehl ist »line«. Dieser Befehl verbindet beliebige Punkte auf dem Bildschirm mit einer geraden Linie. Durch Angabe des Parameters »B«, zeichnet »line« auch ein Viereck über die angegebenen Eckpunkte. Wird der Parameter »F« hinzugefügt, so füllt GW-Basic dieses Rechteck mit

der aktuellen Zeichenfarbe aus. Die Syntax von »line« lautet:
 line (x1,y1)-(x2,y2),farbe,BF,maske
 Die Parameter x1, x2, y1 und y2 bezeichnen Start- und Endpunkt der Linie und haben abhängig vom Vorzeichen absoluten oder relativen Charakter. Wird eine der beiden Koordinaten nicht angegeben, so setzt GW-Basic automatisch die letzte Cursor-Position ein. Der Parameter »farbe« ist identisch mit dem gleichlautenden Parameter im Circle-Befehl und bedarf deshalb keiner weiteren Erklärung. Leere oder gefüllte Vierecke lassen sich durch Angabe der Kennungen »B« beziehungsweise »BF« erzwingen.

Interessant ist auch der letzte Parameter »maske«. Dieser Parameter ist eine 16-Bit-Maske und erlaubt die Definition des Linienmusters. Listing 3 zeigt ein Beispiel für die Handhabung des Line-Befehls. Liegt dem Anwender eine bereits fertiggestellte Grafik vor, so ist es oft wünschenswert, das Attribut eines bestimmten Bildpunktes abzufragen. Hierfür bietet GW-Basic den Befehl »point« mit folgender Syntax an:

point (x,y)
 beziehungsweise

point (n)
 Mit »point (x,y)« wird zunächst der gewünschte Bildpunkt ausgewählt. Anschließend läßt sich über »point (n)« ein beliebiges Attribut abfragen. Zum Setzen eines einfachen Bildpunktes stehen die Befehle »preset« und »pset« zur Verfügung. Die Syntax dieser beiden Befehle lautet:

preset (x,y),farbe
 und

pset (x,y),farbe

Eine genauere Erläuterung dieses Befehls erübrigt sich, weil alle Parameter aus den bereits beschriebenen Befehlen bekannt sind.

Die Programmierung von Shapes ist eine besondere Fähigkeit von GW-Basic

GW-Basic erlaubt auch die Programmierung sogenannter Shapes. Shapes sind Grafikausschnitte, die einer vollständigen Grafik entnommen und an anderer Stelle oder anderen Stellen des Bildschirms wieder eingefügt werden. Mit Shapes läßt sich beispielsweise bedingt Animation simulieren.

Der Basic-Befehl »get« dient zur Übergabe eines Bildschirmausschnitts an eine Feldvariable. Die Syntax dieses Befehls lautet:
 get (x1,y1)-(x2,y2),feldvariable
 Die Koordinaten x1/y1 und x2/y2 geben die zwei gegenüberliegenden Eckpunkte des Bildschirmausschnittes an.

Interessant ist die Berechnung des Platzbedarfs eines Shape, wenn seine relativen Ausmaße bekannt sind:

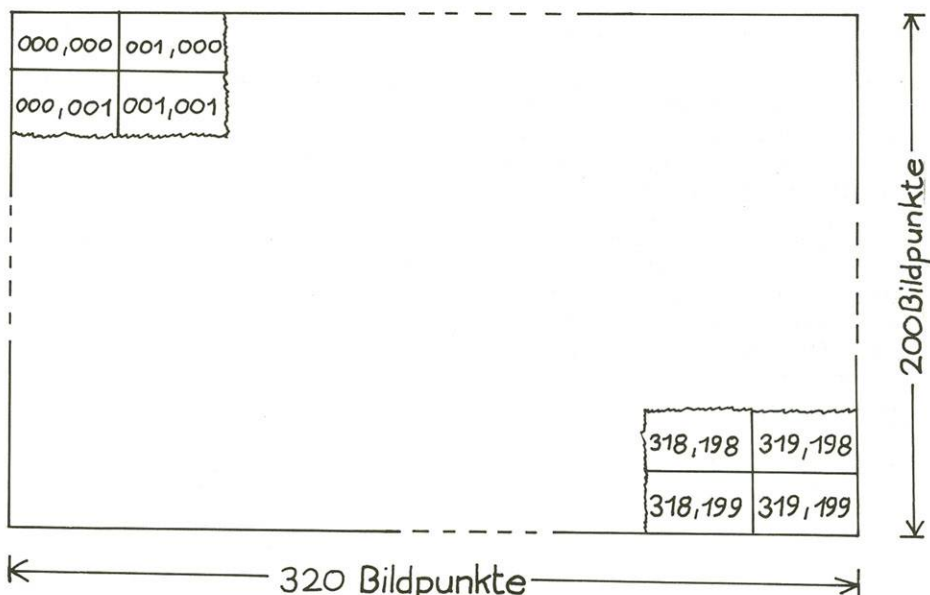


Bild 1. Die mittlere CGA-Grafikauflösung mit 320x200 Bildpunkten

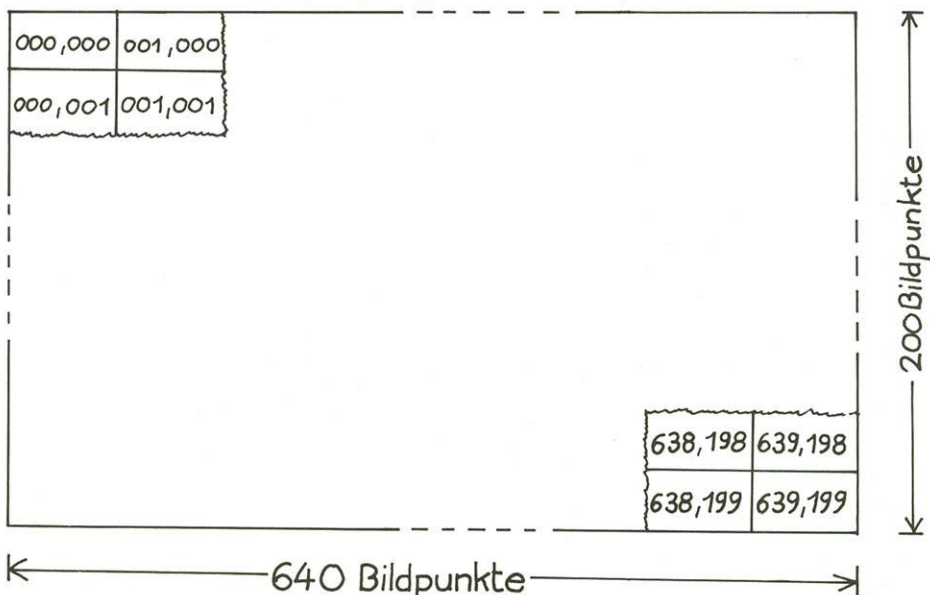


Bild 2. Die hohe CGA-Grafikauflösung mit 640x200 Bildpunkten

Bytes = int((bitwertigkeit*x+7)/8)*y
 Bei der hohen Grafikauflösung beträgt der Wert für die Bitwertigkeit 1 und bei der mittleren Grafikauflösung 2.

Die Wiedergabe des in der Feldvariablen gespeicherten Bildschirmausschnitts erfolgt durch den Befehl »put«. Die Syntax:

put (x,y),feldvariable,modus

Bei dieser Syntax ist der Parameter »modus« neu. Der Parameter »modus« beschreibt die Wiedergabeart des Shape. Tabelle 6 zeigt die Angaben, die für »modus« gültig sind und was sie bedeuten.

Soll die Ausgabe des Shape auf dem Bildschirm nur in einem bestimmten Bereich erfolgen, so kann dieser Bereich über den Befehl »view« definiert werden. Die vollständige Syntax des View-Befehls lautet:

view screen (x1,y1)-(x2,y2),farbe,randfarbe

Die Eckpunkte (x1/y1) und (x2/y2) des Bildschirmbereichs lassen sich

sowohl absolut als auch relativ angeben. Praktisch ist der Parameter »randfarbe«, der es dem Programmierer erlaubt, den Bildschirmbereich mit einer beliebigen Farbe zu umranden. Hinsichtlich der Anzahl von Bildschirmfenstern sind programmtechnisch keine Grenzen gesetzt. Achten Sie jedoch darauf, daß der Befehl »cls« immer nur im gerade aktiven Bildschirmausschnitt funktioniert.

Wie Sie sehen, ist GW-Basic mit erstaunlich vielfältigen und leistungsfähigen Grafikbefehlen ausgerüstet. Die Programmiersprache deckt damit fast alle erdenklichen Grafik-Anwendungen zufriedenstellend ab. Wenn Sie nun auf den Geschmack gekommen sind, empfehlen wir Ihnen, die beschriebenen Befehle auszuprobieren und die Beispiele auszubauen, um durch »learning by doing« das Maximum an Grafikpower aus Ihrem PC herauszukitzeln. (sp)


```

100 REM *****
110 REM * Tortendiagramm (sp190987) *
120 REM *****
130 :
140 SCREEN 2,1 : 'Hohe Aufloesung, Farbe = Blau
150 CLS : 'Bildschirm loeschen
160 LOCATE 5,20 : PRINT "Kuchendiagramm"
170 LOCATE 6,20 : PRINT "*****"
180 PRINT
190 PRINT " Bitte 10 Werte eingeben:"
200 PRINT
210 :
220 REM *** Zehn Zahlenwerte eingeben ***
230 :
240 FOR I = 1 TO 10
250 PRINT " Bitte Wert";I;"eingeben";
260 INPUT KUCHEN (I)
270 MAX = MAX + KUCHEN (I) : 'Maximalwert berechnen
280 NEXT I
290 :
300 REM *** Prozentwerte berechnen und anzeigen ***
310 :
320 LOCATE 8,35 : PRINT "PROZENT"
330 LOCATE 9,35 : PRINT "*****"
340 FOR I = 1 TO 10
350 LOCATE 9+I,35 : PRINT KUCHEN (I) / MAX * 100 ; "%" : 'Prozentangabe
360 WINKEL (I) = KUCHEN (I) / MAX * 2 * 3.141593 : 'Bogenmass berechnen
370 NEXT I
380 :
390 REM *** Kuchendiagramm darstellen ***
400 :
410 ANFANG = -6.283186 : 'Anfang bei - 2 PI
420 LOCATE 2,60 : PRINT "Grafik"
430 LOCATE 3,60 : PRINT "*****"
440 FOR I = 1 TO 10
450 ENDE = ANFANG + WINKEL (I) : 'Endwert berechnen
460 CIRCLE (500,100),120,,ENDE,ANFANG: 'Segment zeichnen
470 ANFANG=ENDE : 'Neuer Anfang = altes En
480 NEXT I
490 LOCATE 20,1

```

Listing 1. So entsteht ein Kuchendiagramm mit einfachen Mitteln

```

100 REM *****
110 REM * DRAW-Demo (sp190987) *
120 REM *****
130 :
140 SCREEN 2,1 : 'Hohe Aufloesung, Farbe = Blau
150 CLS : 'Bildschirm loeschen
160 INPUT "Schrittweite";SCHRITT
170 FOR I=0 TO 360 STEP SCHRITT
180 DRAW "d50r50e25u50l50g25r50d50u50e25d50g25l50u50": 'Quader zeichnen
190 DRAW "ta=i;" : 'Drehwinkel in Grad
200 NEXT I
210 A$=INKEY$ : IF A$="" THEN 210 : 'Auf Tastendruck warten
220 RUN

```

Listing 2. Ein Beispielprogramm für den Draw-Befehl

```

100 REM *****
110 REM * LINE-Demo (sp190987) *
120 REM *****
130 :
140 SCREEN 2,1 : 'Hohe Aufloesung, Farbe = Blau
150 CLS : 'Bildschirm loeschen
160 FOR I=0 TO 1000
170 LINE -(RND(1)*640,RND(1)*200): 'Zufaellige Linien ziehen
180 NEXT I
190 A$=INKEY$ : IF A$="" THEN 190 : 'Auf Tastendruck warten
200 RUN

```

Listing 3. Das Line-Demo zeichnet Zufallslinien

Wissen Sie mehr von PCs als andere Leute?

Dann schreiben Sie uns. Die Redaktion DOS International sucht aktive Mitarbeiter, die über interessante Gebiete aus der »Computerei« berichten wollen. MS-DOS, DOS Plus, GEM, Drucker, Basteleien – es gibt nichts, über das nicht andere Leser unserer Zeitschrift auch etwas lesen wollen.

Ihren Einsatz lassen wir uns natürlich etwas kosten. Je nach Qualität des zugesandten Manuskripts bekommen Sie pro gedruckte Zeile zwischen 70 Pfennig bis zu 1,10 Mark. Bilder, Zeichnungen, Tabellen und Listings werden gesondert berechnet.

Doch bevor Sie sich hinsetzen und das Schreiben anfangen:

Rufen Sie uns an.

Sagen Sie uns was Sie schreiben wollen, und wir klären ob – und wann – der Artikel in unser Konzept paßt.

Der Artikel muß mit einem Textsystem erfaßt werden. Wir brauchen von Ihnen eine Diskette im MS-DOS-Format auf der der Text als reines ASCII-Feld steht.

Aber auch wenn Sie nur Vorschläge für Themen haben, dann schreiben Sie uns.

DOS International ist nämlich eine Zeitschrift von aktiven PC-Besitzern für aktive PC-Benutzer. Besonders das DOS-Forum lebt von Ihrer Mitarbeit. Auch die Redaktion kann nicht alle Antworten wissen. Fragen an uns, die nicht sofort beantwortet werden können, werden im DOS-Forum veröffentlicht. Wissen Sie eine Antwort, so wird diese veröffentlicht.

Sie sehen: DOS International ist die PC-Zeitschrift, in der das Wissen der PC-Benutzer untereinander ausgetauscht wird.

Redaktion · DOS International · Elfenstr. 40 · 8000 München 83
Tel.: (089) 60 20 66

Farbgrafikkarte und Monochomadapter

Die Farbgrafikkarte, kurz CGA (Color Graphics Adapter) genannt, gehört inzwischen zur Grundausstattung nahezu aller PCs. Grund genug also, sich die verschiedenen Betriebsarten und Darstellungsfähigkeiten dieser Karte einmal genauer anzusehen. Als Nebenprodukt fallen dabei auch einige nützliche Informationen über den Monochomadapter ab, den ältesten der Familie der IBM-kompatiblen Bildschirmadapter.

Der Monochomadapter und die Farbgrafikkarte sind die einzigen Bildschirmadapter, die vom Betriebssystem auf BIOS-Ebene direkt unterstützt werden. Für diejenigen, die es noch nicht wissen, die vier Buchstaben BIOS sind die Abkürzung für »Basic Input-Output System«, zu deutsch »Basissystem für Ein- und Ausgabe«. Das BIOS ist in der Hardware (in einem ROM) implementiert und steht deshalb immer zur Verfügung, ohne daß es geladen werden muß. Es besteht aus einer Menge Routinen, die alle notwendigen Hardware-Funktionen anbieten und wird an anderer Stelle in diesem Sonderheft ausführlich beschrieben. Für unsere Zwecke interessant und wichtig ist der sogenannte Video-Interrupt des BIOS, der Interrupt 10hex. Es existieren insgesamt 16 Unterrouinen zur Bildschirmsteuerung, die alle über den Interrupt 10 zu erreichen sind. Um die gewünschte Routine auszuwählen, muß man das AH-Register des Prozessors mit einer Zahl von 0 bis 15 laden, bevor der Interrupt ausgelöst wird. In Tabelle 1 sind die verschiedenen Routinen des Video-Interrupts aufgelistet. Einige dieser Routinen sind in Hochsprachen wie GW-Basic oder Turbo Pascal bereits implementiert und können direkt verwendet werden, häufig sogar mit vernünftigen Namen und einfacher Syntax. Die entsprechenden Basic- und Pascal-Befehle sind in der Tabelle namentlich aufgeführt.

IBM war ursprünglich der Ansicht, der PC sei nur für den geschäftlichen Einsatzbereich geeignet und deshalb wurden die ersten PCs mit dem »Monochrome Display Adapter« MDA, auf deutsch »Karte zur einfarbigen Bildschirmsteuerung«, ausgeliefert. Diese Karte bietet (beziehungsweise bot, da sie kaum mehr zu bekommen ist) ein sehr schönes Schriftbild auf einem 80 (Spalten) x 25 (Zeilen) Bildschirm, ist aber weder farb- noch grafikfähig. Diese Einschränkungen führten schnell zu einem Nachfolger, der Farbgrafikkarte CGA, die bald zum Standard avancierte. Diese Farbgrafikkarte (Color Graphic Adapter) hat sich inzwischen soweit durchge-

setzt, daß sie von jeder nachfolgenden Karte emuliert wird. Der Grund ist einfach: Es gibt zuviele Programme, die sich auf der Funktionalität der CGA stützen, um diese Karte ignorieren zu können.

Die CGA-Karte bietet, wie die MDA-Karte, einen 80x25 Text-Bildschirm, aber die Schrift ist lange nicht so schön wie bei ihrem Vorgänger. Warum sollte man dann überhaupt eine CGA-Karte anschaffen? Die Antwort ist einfach: Grafik und Farbe. Außer einem Textmodus in 16 Farben bietet die CGA-Karte zwei Grafikmodi, 320 (waagrecht) x 200 (senkrecht) Punkte in vier Farben, oder 640x200 Punkte in zwei Farben. Zusätzlich hat die CGA-Karte einen neuen Textmodus, 25 Zeilen mit je 40 (doppelgroßen) Zeichen auch in 16 Farben.

Die Farbgrafikkarte kennt sieben unterschiedliche Betriebsarten

Die CGA-Karte läßt sich in mehreren Modi betreiben, der MDA dagegen kennt jedoch nur einen einzigen Mo-

du. Der Modus setzt drei Eigenschaften der Bildschirmanzeige fest. Erstens gibt es die Unterscheidung zwischen Text und Grafik. Es gibt reine Textmodi, in denen keine Grafik dargestellt werden kann, daneben stehen die Grafikmodi, die sowohl Text als auch Grafik auf den Bildschirm bringen können. Zweitens bestimmt der Modus, wieviele Farben gleichzeitig gezeigt werden können. Drittens legt dieser Modus fest, aus wievielen Zeichen beziehungsweise Punkten eine Zeile besteht.

Tabelle 2 faßt die verschiedenen Betriebsarten der Bildschirmdaten zusammen. Die Nummer in der linken Spalte wird als Kennzeichen für den Modus verwendet, man spricht vom »Modus 6« oder »Modus 1«. Wie man in der Tabelle sieht, besitzt die CGA-Karte die Modi 0 bis 6. Der einzige Modus der MDA-Karte ist die Nummer 7, der dem CGA-Modus 2 ähnelt.

Wer sich Tabelle 2 etwas näher betrachtet, bemerkt, daß einige CGA-Modi sich sehr ähnlich sind. Es gibt drei Paare: 0 und 1, 2 und 3, und 4 und 5, die sich nur in der Wiedergabe von Farben unterscheiden. Wieso? Die Antwort liegt in der Hardware, in diesem Fall in der Grafikkarte und dem Monitor begründet. Als die CGA-Karte erstmals auf den Markt kam, waren Farbmonitore drei- bis fünfmal so teuer wie sie heute sind. Deshalb besitzt die originale CGA-Karte zwei Monitor-Ausgänge, einen für einen Farbmonitor und einen für einen Monochrom-Monitor. Wer auf Grafik Wert legte, aber die paar Tausender für einen Farbmonitor nicht erübrigen konnte, war damit auch gut bedient. Er konnte nämlich einen

Nr.	BIOS-Routine	Turbo Pascal	GW-Basic
0	Modus setzen	TextMode GraphMode, GraphColorMode, HiRes	Screen, Width
1	Cursorgröße setzen	—	Locate
2	Cursorposition setzen	GotoXY	Locate
3	Cursorposition abfragen	WhereX,WhereY	Csrlin,Pos
4	Lichtgriffelposition abfragen	—	—
5	aktive Seite setzen	—	Screen
6	Fenster nach oben rollen	—	—
7	Fenster nach unten rollen	—	—
8	Zeichen lesen	—	Screen
9	Zeichen und Attribut schreiben	Write, TextColor, TextBackground	Print, Color
10	Zeichen schreiben	Write	Print
11	Farbpalette auswählen	Palette, GraphBackground	Color
12	Pixel schreiben	Plot	PSet, PReset
13	Pixel lesen	GetDotColor	Point
14	TTY-Zeichen schreiben	Write, TextColor	Print, Color
15	Modus abfragen	—	—

Tabelle 1. Die BIOS-, Basic- und Pascal-Routinen der Bildschirmsteuerung

günstigen Monochrom-Monitor an die Farbgrafikkarte anschließen und damit die Grafikfähigkeit der CGA-Karte ausnutzen – allerdings ohne Farbe. Dies ging aber nur dann, wenn die CGA-Karte kein Farbsignal an den Monochrom-Monitor weitergab, und dies ist der Grund für die Existenz der drei »farblosen« Modi. Diese drei Betriebsarten ermöglichen eine Zusammenarbeit zwischen der Farbgrafikkarte und einem Monochrom-Monitor. Ansonsten sind sie überflüssig, es sei denn, Sie möchten keine Farben mehr sehen.

Wenn Sie mit einem Monochrom-Monitor arbeiten und ein etwas seltsames Bild erhalten, überprüfen Sie, ob Modus 1, 3 oder 4 verwendet wurde. Wenn dies der Fall ist, stellen Sie entsprechend auf Modus 0, 2 oder 5 um, und schon erzielen Sie ein besseres Ergebnis. Das Gegenteil gilt für diejenigen, die einen Farbmonitor verwenden. Falls die Farben plötzlich verschwinden (oder, bei manchen CGA-Karten fest auf drei unerwünschten Farben bleiben, auch wenn Sie andere Farben programmiert haben), prüfen Sie, ob nicht versehentlich Modus 0, 2 oder 5 gewählt wurde. Wenn das der Fall sein sollte, steigen Sie auf den entsprechenden Modus 1, 3 oder 4 um.

Die Farbgrafikkarte bietet bis zu 16 Farben gleichzeitig am Bildschirm

Die CGA-Karte kann 16 Farben darstellen. Diese Anzahl kommt nicht von Ungefähr, sondern ist eine Konsequenz der Arbeitsweise dieser Karte. Das Signal zum Monitor setzt sich aus vier Komponenten zusammen, wobei jede Komponente entweder ins Signal eingemischt wird oder aus dem Signal fortgelassen wird. Vier Komponenten mit je zwei möglichen Zuständen ergeben gerade 16 komplette Signale, die jeweils eine Farbe wiedergeben. Drei der vier Komponenten sind die Farbkomponenten Rot, Blau und Grün. Die vierte Komponente heißt »Intensität« und ist in etwa als Helligkeit zu verstehen. Wenn die Intensitäts-Komponente aktiv ist, resultiert einfach eine hellere Version der gleichen Grundfarbe. Die Farben können deshalb in die zwei Gruppen, »hell« und »dunkel«, aufgeteilt werden, wie sie in Tabelle 3 aufgeführt sind. Wie die Farben tatsächlich aussehen, hängt sehr stark vom verwendeten Monitor ab. Manche Monitore können überhaupt nichts mit dem Wert für Intensität anfangen, das heißt, die dunkle und die helle Version einer Farbe sehen

Modus	Art	Auflösung	Farben
0	Text	40x25	2 (S/W)
1	Text	40x25	16
2	Text	80x25	2 (S/W)
3	Text	80x25	16
4	Grafik	320x200	4
5	Grafik	320x200	4 (Graustufen)
6	Grafik	640x200	2 (S/W)
7	Text	80x25	2 (S/W, MDA-Modus)

Tabelle 2. Die Betriebsarten des Farbgrafikadapters (CGA)

Farbkomponenten			Intensität = 0 (dunkel)		Intensität = 1 (hell)	
Rot	Grün	Blau	Nr.	Farbe	Nr.	Farbe
0	0	0	0	Schwarz	8	Dunkelgrau
0	0	1	1	Blau	9	Hellblau
0	1	0	2	Grün	10	Hellgrün
0	1	1	3	Türkis	11	Helltürkis
1	0	0	4	Rot	12	Hellrot
1	0	1	5	Violett	13	Lila
1	1	0	6	Braun	14	Gelb
1	1	1	7	Hellgrau	15	Weiß

Tabelle 3. Die Farben der CGA-Karte

Nr.	Turbo Pascal	GW-Basic
0	TextMode (BW40);	SCREEN 0,0 : WIDTH 40
1	TextMode (C40);	SCREEN 0,1 : WIDTH 40
2	TextMode (BW80);	SCREEN 0,0 : WIDTH 80
3	TextMode (C80);	SCREEN 0,1 : WIDTH 80
4	GraphColorMode;	SCREEN 1,0
5	GraphMode;	SCREEN 1,1
6	HiRes;	SCREEN 2

Tabelle 4. Setzen des Modus mit Turbo Pascal und GW-Basic

Palette	Farbe 0	Farbe 1	Farbe 2	Farbe 3
0	(1)	Grün	Rot	Braun
1	(1)	Türkis	Magenta	Hellgrau
2	(1)	Hellgrün	Hellrot	Gelb
3	(1)	Helltürkis	Lila	Weiß

(1) Farbe 0 ist standardmäßig Schwarz, kann aber auf jede beliebige Farbe gesetzt werden.

Tabelle 5. Die Farb-Paletten der CGA-Karte

gleich aus, während andere Monitore Schwierigkeiten mit manchen Farben (typischerweise Braun, Gelb und Grau) aufweisen. Was bei Ihrem Gerät der Fall ist, stellen Sie durch Ausprobieren sehr schnell fest. Tabelle 4 zeigt die Funktionen der Sprachen Turbo Pascal und GW-Basic, die den Bildschirmmodus setzen. Hier ist kein Zugriff auf dem BIOS oder der Hardware notwendig, da alle Modi bequem und einfach von der Hochsprache aus eingestellt werden können. Jedoch bietet weder Pascal noch Basic eine einfache Methode, um den Modus abzufragen. Hier könnte man die BIOS-Routine »Modus abfragen« direkt aufrufen, jedoch gibt es eine noch einfachere Möglich-

keit. Das BIOS speichert die Nummer des Modus an einer festen Adresse, nämlich 449 hexadezimal. Mit der Pascal-Zeile

```
modus := mem ($40:$49);
oder mit den Basic-Zeilen
```

```
def seg = &H40
Modus = peek(&H49)
def seg
```

bekommt man den aktuellen Modus in der Variablen »Modus« zurück.

Der Bildschirmspeicher wird auf zwei Arten verwendet. In den Textmodi werden ASCII-Codes in den Bildschirmspeicher geschrieben, und die CGA-Karte wandelt diese Codes in Punktmuster um, die wiederum auf dem Bildschirm zu sehen sind. In

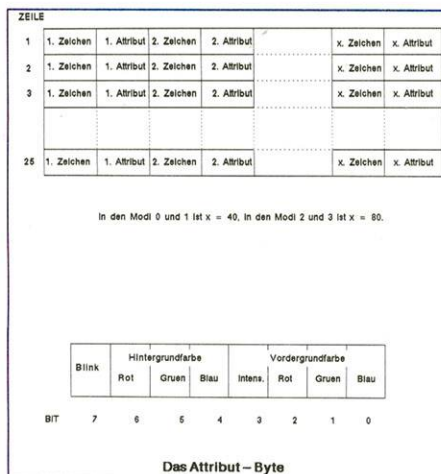


Bild 1. Der Aufbau des Bildschirmspeichers in den Textmodi

den Grafikmodi werden Punkte in Form verschlüsselter Farben in den Speicher geschrieben. Die CGA-Karte mischt das Farbsignal nach dem Speicherinhalt zusammen, muß aber keine weitere Umwandlung vornehmen.

Der Bildschirmspeicher ist ein reservierter Bereich im Gesamtspeicher des Prozessors, der sowohl vom Prozessor (sprich vom Programm) als auch von der Bildschirmsteuerlogik gelesen und geschrieben werden kann. Dieser Speicher ist physikalisch getrennt vom »normalen« Speicher, wo Programme und Daten abgelegt werden. Diese Trennung spiegelt sich auch in den Adressen wieder. Während der Hauptspeicher eines PC von »0000:0000« bis maximal »9000:FFFF« liegt, ist der Bildschirmspeicher der CGA-Karte zwischen den Adressen »B800:0000« und »B800:3FFF« angesiedelt und damit 16 KByte groß. Der Speicher der MDA-Karte ist dagegen nur 4 KByte groß und liegt unter der Adresse »B000:0000« bis »B000:0FFF«.

Die Textmodi der Farbgrafikkarte im Detail

Den Basisaufbau des Bildschirmspeichers zeigt der obere Teil von Bild 1. Wie abgebildet, besteht jedes angezeigte Zeichen aus zwei Byte. Das erste Byte ist einfach das Zeichen in ASCII-Code, während das zweite Byte das sogenannte Attribut darstellt. Die Zusammenstellung dieses Attributs findet sich im unteren Teil des Bildes. Wie man sieht, bestimmt das Attribut mit vier Bit die Farbe des Zeichens und mit drei Bit die Farbe des Hintergrunds. Das letzte Bit bestimmt, ob das Zeichen blinkend dargestellt wird oder nicht. Das kleine Basic-Programm in Listing 1 führt die verschiedenen Attribut-Bits vor. Star-

```

1 ' Initialisierung - Modus 3, lösche Bildschirm
10 SCREEN 0,1: WIDTH 80
20 CLS
21 ' Schreibe Text-Muster
30 FOR I=6 TO 21
40   FOR J=12 TO 72 STEP 4
50     LOCATE I,J: PRINT "ABC";
60   NEXT
70 NEXT
71 ' Schreibe Attribut-Byte-Werte
80 FOR I=13 TO 73 STEP 4
90   LOCATE 4,I: PRINT HEX$((I-13)*4);
100  NEXT
110 FOR I=6 TO 21
120   LOCATE I,3: PRINT HEX$(I-6);
130 NEXT
131 ' Schreibe Titel
140 LOCATE 1,25: PRINT "Attribute in den Text-Modi"
141 ' Fange mit Attribut = 0 an
150 ATTR = 0
151 ' Setze Bildschirmspeicher als POKE-Ziel
160 DEF SEG = &HB800
161 ' Schreibe die Attribute direkt in Bildschirmspeicher
170 FOR I=6 TO 21
180   FOR J=12 TO 72 STEP 4
190     ADDR = (I-1)*160 + (J-1)*2 + 1
200     POKE ADDR,ATTR: POKE ADDR+2,ATTR: POKE ADDR+4,ATTR
210     ATTR = ATTR+1
220   NEXT
230 NEXT
231 ' setze SEG zurück (vorsichtshalber...)
240 DEF SEG
250 LOCATE 24,25: PRINT "[Ctrl-Break] zu beenden";
260 GOTO 260

```

Listing 1. Dieses Basic-Programm zeigt alle Attribute

ten Sie das Programm und beachten Sie die Ausgabe, um einen schnellen Blick in die Möglichkeiten, die Ihnen über die Attribute zur Verfügung stehen, zu bekommen. Der Basisaufbau des Bildschirmspeichers ist in allen Textmodi gleich. Auf der Adresse »B800:0000« steht das erste ASCII-Zeichen, das in der oberen, linken

Ecke des Bildschirms sichtbar ist. In der nächsten Speicherzelle, Adresse »B800:0001«, steht das Attribut-Byte für dieses Zeichen. In der nächsten Zelle wiederum, also in »B800:0002«, steht schon das zweite ASCII-Zeichen, danach das zugehörnde Attribut, und so geht es auch weiter bis zum Ende der ersten Zeile. Die zwei-

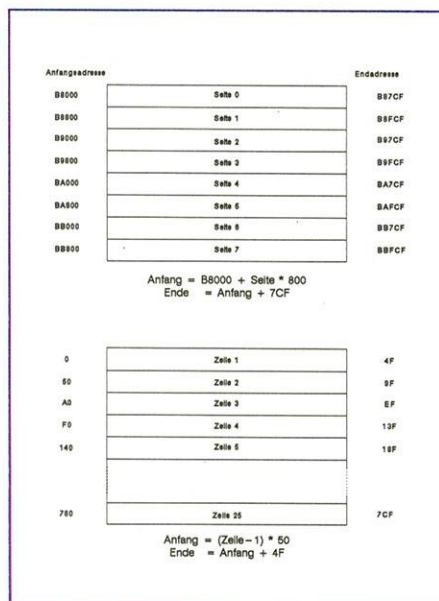


Bild 2. Zeilen und Seiten in den Textmodi 0 und 1

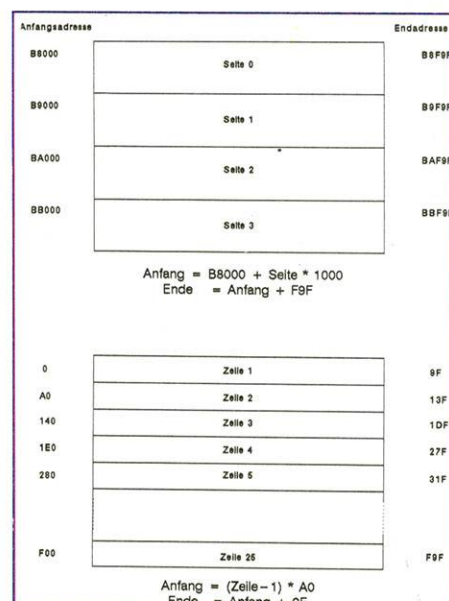


Bild 3. Zeilen und Seiten in den Textmodi 2 und 3


```

0: VAR Regs: RECORD ax,bx,cx,dx,bp,si,di,ds,es,fl: INTEGER;
1:      END;
2:
3: FUNCTION SetScreenPage (Seite: INTEGER): BOOLEAN;
4: BEGIN
5:   SetScreenPage := FALSE;
6:   IF (Seite<0) THEN EXIT;
7:   IF (Mem[$40:$49]<2) THEN BEGIN
8:     IF (Seite>7) THEN EXIT;
9:     END
10:  ELSE BEGIN
11:    IF (Seite>3) THEN EXIT;
12:    END;
13:  Regs.ax := 1280+Seite;
14:  Intr (16,Regs);
15:  SetScreenPage := TRUE;
16:  END;

```

Listing 2. Diese Routine ermöglicht die Verwendung von Bildschirmseiten in Turbo Pascal

te Zeile steht direkt nach der ersten im Speicher und ist, wie alle anderen Zeilen auch, genauso aufgebaut wie die erste Zeile, also als Kette von ASCII-Zeichen, jeweils gefolgt vom entsprechenden Attribut-Byte.

Die Anfangsadresse aller Zeilen nach der ersten hängt vom Modus ab, da es zwei unterschiedliche Längen für eine Textzeile gibt. In den CGA-Modi 0 und 1 ist eine Zeile 40 Zeichen lang, in den Modi 2 und 3 jedoch 80 Zeichen, das heißt, eine Zeile im Modus 0 oder 1 beansprucht 80 Byte (40 ASCII-Codes plus 40 Attribut-Byte), während eine Zeile im Modus 2 oder 3 160 Byte belegt. Die beiden folgenden Formeln ergeben jeweils die Verschiebungen eines Zeichens, relativ zum ersten Zeichen:

in Modus 0 oder 1: (Zeile x 80) +
(Spalte x 2)
in Modus 2 oder 3: (Zeile x 160) +
(Spalte x 2)

Die Adresse des zugehörigen Attribut-Byte ist immer die Adresse des Zeichens plus 1. Im Bild 2 ist der Speicheraufbau für den 40-Zeichen-Modus gezeigt, Bild 3 veranschaulicht den Aufbau für den 80-Zeichen-Modus. Die Farbgrafikkarte bietet in den Textmodi sogenannte Bildschirmseiten an, die jeweils einen kompletten Bildschirm darstellen. Natürlich ist nur eine Seite auf einmal sichtbar, mit ein bißchen Mühe kann jedoch zu jedem beliebigen Zeitpunkt eine beliebige Seite geschrieben werden. Da die Modi 0 und 1 etwa 2 KByte pro Bildschirmseite benötigen, werden Seiten (mit Nummern von 0 bis 7) eingerichtet, während in den Modi 2 und 3 vier Seiten (Nummern von 0 bis 3) zur Verfügung stehen.

Das BIOS bietet eine Funktion (Funktion 5) an, womit die aktive Seite gesetzt werden kann. »Aktiv« bedeutet hier, daß diese Seite auf dem Bild-

schirm erscheint. Die anderen Seiten können nach wie vor angesprochen werden, sind aber unsichtbar, bis eine neue aktive Seite gesetzt wird. Diese BIOS-Funktion ist auch in GW-Basic zu finden, als Teil des Screen-Befehls. Eine Anweisung wie

```
screen ,0,1
```

bewirkt, daß die Seite 1 auf dem Bildschirm gezeigt wird, während alle Print-Befehle ihre Ausgabe auf Seite 0 schreiben. Ausnahmsweise ist diese Funktion nicht in Turbo Pascal zu finden, und deshalb ist in Listing 2 die Turbo-Funktion »SetScreenPage« aufgelistet. Diese Routine überprüft die angegebene Seitennummer und selektiert, falls die Nummer in Ordnung ist, die neue Seite und gibt den Wert »True« zurück. Falls die Seitennummer nicht korrekt ist, gibt die Routine »False« zurück, ohne eine neue Seite zu aktivieren.

Eine Funktion, die sowohl im BIOS als auch in den höheren Programmiersprachen fehlt, ist »aktive Seite abfragen«. Glücklicherweise speichert das BIOS die Nummer der aktuellen Seite, so daß wir nur von dieser Adresse ein Byte ablesen müssen, um diese wichtige Auskunft zu bekommen. Die Adresse lautet 462hex und kann, ähnlich wie der Bildschirmmodus in Adresse »\$0040:0049« sehr einfach abgefragt werden. In

```

10 CLS
20 SCREEN ,,1.0
40 FOR I=0 TO 1999: PRINT "1";: NEXT
45 C=1
50 SCREEN ,,0.0
55 C=(C+1) MOD 8)+1: COLOR C: LOCATE 1,1
60 FOR I=0 TO 1999
70 PRINT "0";: IF INKEY$<>" " THEN 100
80 NEXT
90 GOTO 50
100 SCREEN ,,1.1
110 IF INKEY$<>" " THEN 50
120 GOTO 110

```

Listing 3. Ein Basic-Programm zeigt, wie schnell die Bildschirmseite gewechselt wird

Turbo Pascal ist lediglich eine Zeile notwendig:

```
Seite := Mem ($40:$62);
```

In GW-Basic ist die Sache etwas länger, aber genau so einfach:

```
def seg = &H40
Seite = peek(&H62)
def seg
```

Der Befehl »def seg« nach dem Peek-Befehl ist reine Höflichkeit und setzt das Segment für weitere Peek- und Poke-Befehle zurück auf das Standard-Segment des Basic-Programmes.

Jetzt möchte man vielleicht fragen:

»Wenn nur eine Bildschirmseite sichtbar ist, wozu sind die anderen gut?« Eine Antwort ist, daß ein sehr schneller Bildschirmwechsel erzielt wird, wenn das neue Bild stets auf einer anderen Seite bereits fertig steht. Man braucht lediglich die neue Seite einzuschalten, und schon ist das Bild da. Das kleine Basic-Programm in Listing 3 zeigt Ihnen, wie schnell ein Seitenwechsel erfolgt. Tippen Sie doch das Programm einfach ein und probieren Sie es aus – es wird Sie überzeugen.

Das Schreiben auf dem Bildschirm ist Bestandteil jeder Sprache. Jedoch ist es von einer Hochsprache aus umständlich, wenn man auf einmal Farbe, Position (Zeile und Spalte) und Seite setzen will. Auf der niedrigen Ebene des BIOS hingegen ist dies nicht schwierig. Um ein Zeichen zu schreiben, bietet das BIOS gleich drei verschiedene Funktionen an. Eine davon, die Nummer 14 »TTY-Schreiben«, ist im wesentlichen die bekannte Write- oder Print-Anweisung. Diese Funktion brauchen wir nicht weiter zu behandeln.

Mit den anderen beiden BIOS-Routinen kann man in einem Befehl das Zeichen, dessen Attribute und die Seite auf einmal bestimmen, beziehungsweise ein Zeichen auf einer beliebigen Seite mit den dort bestehenden Attributen schreiben. Die Funktion 9 »Zeichen und Attribut schreiben« erhält vier Parameter zugewiesen: das Zeichen, dessen Attribut,

Blink	Hintergrund	Hell / Fett	Vordergrund
-------	-------------	-------------	-------------

BIT	7	6	5	4	3	2	1	0
	Hintergrund				Vordergrund			
	0 0 0	0 0 0	nicht darstellen					
	0 0 0	0 0 1	unterstreichen					
	0 0 0	1 1 1	normal					
	1 1 1	0 0 0	umgekehrt					

Das Attribut – Byte der MDA – Karte

Bild 4. Das Attribut-Byte der MDA-Karte


```

0: VAR Regs: RECORD ax,bx,cx,dx,bp,si,di,ds,es,fi: INTEGER;
1:   END;
2:
3: PROCEDURE WriteChar (Zeichen: CHAR; Attribut, Seite, Anzahl
  : INTEGER);
4: VAR i: INTEGER;
5: BEGIN
6:   Regs.ax := 2304+Ord(Zeichen);
7:   Regs.bx := Lo(Seite)*256 + Lo(Attribut);
8:   Regs.cx := Anzahl;
9:   Intr (16,Regs);
10:  END;

```

Listing 4. Die Turbo Pascal-Routine gibt ein Zeichen und dessen Attribut über das BIOS auf den Bildschirm aus

```

0: PROCEDURE WriteChar (Zeichen: CHAR; Attribut, Zeile, Spalte
  , Seite: INTEGER);
1: VAR i: INTEGER;
2: BEGIN
3:   IF (Zeile<1) OR (Spalte<1) OR (Seite<0) OR (Zeile>25) THE
  N EXIT;
4:   i := Mem [$40:$49];
5:   IF (i<2) THEN BEGIN
6:     i := 1;
7:     IF (Spalte>40) OR (Seite>7) THEN EXIT;
8:     END
9:   ELSE BEGIN
10:    i := 2;
11:    IF (Spalte>80) OR (Seite>3) THEN EXIT;
12:    END;
13:   i := Seite*i*2048 + (Zeile-1)*i*80 + (Spalte-1)*2;
14:   Mem [$B800:i] := Ord(Zeichen);
15:   Mem [$B800:i+1] := Lo(Attribut);
16:   END;

```

Listing 5. Eine Turbo Pascal-Routine, die direkte Ausgabe mit Bildschirmspeicher-Zugriffen realisiert

```

10000 IF (Z<1) OR (SP<1) OR (S<0) OR (Z>24) THEN RETURN
10010 DEF SEG = &H40: MM=PEEK(&H49): DEF SEG: IF MM>3 THEN RETURN
10020 IF MM>1 THEN 10040
10030 MM=1: IF (S>7) OR (SP>40) THEN RETURN ELSE 10050
10040 MM=2: IF (S>3) OR (SP>80) THEN RETURN
10050 DEF SEG = &HB800
10060 MM=S*MM*2048+(Z-1)*MM*80+(SP-1)*2: POKE MM,C: POKE MM+1,A
10070 DEF SEG
10080 RETURN

```

Listing 6. Das Basic-Unterprogramm schreibt Zeichen und Attribut direkt in den Bildschirmspeicher

```

10000 C=-1: A=-1
10010 IF (Z<1) OR (SP<1) OR (S<0) OR (Z>24) THEN RETURN
10020 DEF SEG = &H40: MM=PEEK(&H49): DEF SEG: IF MM>3 THEN RETURN
10030 IF MM>1 THEN 10050
10040 MM=1: IF (S>7) OR (SP>40) THEN RETURN ELSE 10060
10050 MM=2: IF (S>3) OR (SP>80) THEN RETURN
10060 DEF SEG = &HB800
10070 MM=S*MM*2048+(Z-1)*MM*80+(SP-1)*2: C=PEEK(MM): A=PEEK(MM+1)
10080 DEF SEG
10090 RETURN

```

Listing 7. Das Gegenstück zum Listing 6 - dieses Unterprogramm liest Zeichen und Attribut

die Seite und die Anzahl der zu schreibender Zeichen. Die Funktion 8 übernimmt das bestehende Attribut und bekommt nur die anderen drei Parameter.

In Listing 4 ist die Turbo Pascal Routine »WriteChar«, die diese Funktionalität über das BIOS realisiert.

Betrachtet man aber diese BIOS-Routinen genau, wird klar, daß die beiden Routinen nicht mehr tun, als in den Bildschirmspeicher schreiben. Können wir das nicht auch? Zunächst die Frage, was passiert, wenn man ein ASCII-Zeichen direkt in den Bildschirmspeicher schreibt?

Die erfreuliche Antwort ist, wenn man die oben angegebenen Formeln beachtet, erscheint das Zeichen auf dem Bildschirm. Das gleiche gilt für Attribute: Schreibt man ein Attribut-Byte in den Bildschirmspeicher, bekommt das zugehörige Zeichen sofort die entsprechenden Eigenschaften. Also gut, aber wie macht man das in der Praxis?

Erweitern wir zuerst die beiden Formeln, die Speicheradressen aus Spalte und Zeile berechnen, so daß die Seite auch berücksichtigt wird. Dann haben wir:

in Modus 0 oder 1: (Seite x 2048) + (Zeile x 80) + (Spalte x 2)
 in Modus 2 oder 3: (Seite x 4096) + (Zeile x 160) + (Spalte x 2)

Wie zuvor ist die Adresse des Attribut-Byte stets die Adresse des Zeichens plus 1. Jetzt haben wir alle Voraussetzungen für eine allgemeine Zeichen-Ausgabe-Routine. Die beiden Routinen in Listing 5 und 6 sind Ihre Aufmerksamkeit wert: Einmal in Basic und einmal in Pascal kann man hier Zeichen und Attribute beliebig und sehr schnell ausgeben. Diese beiden Routinen funktionieren auch mit dem Monochromadapter MDA, wenn man die Adresse B800hex durch B000hex ersetzt. Das Attribut-Byte der MDA-Karte ist natürlich ein anderes als das der CGA-Karte, da die MDA-Karte nicht farbfähig ist. In Bild 4 ist der Aufbau des Attribut-Byte der MDA-Karte dargestellt.

Eine interessante Funktion ist das Lesen eines Zeichens von einer bestimmten Bildschirmposition. Hier bietet GW-Basic wieder die Screen-Funktion an. Turbo Pascal schweigt, wie bei den Bildschirmseiten. Und wie bei der Ausgabe, geht es einfacher und schneller mit direkten Zugriffen auf dem Bildschirmspeicher. Da die Routinen praktisch identisch mit den Ausgabe-Routinen sind, ist keine weitere Erklärung erforderlich. In Listing 7 ist die Basic-Routine, in Listing 8 die Pascal-Routine zu finden. Beide geben das gelesene ASCII-Zeichen in der Variablen »ZCode« und das gelesene Attribut in der Variablen »ZAttr« zurück. In Pascal müssen die beiden Variablen bereits außerhalb der Routine »ReadChar« deklariert sein. Damit haben wir die wichtigsten Funktionen in den Textmodi schon gesehen. Sie sollten wissen, daß alle Funktionen in allen Modi das erwünschte Ergebnis liefern. Passen Sie lediglich auf, daß Seite, Zeile und Spalte stimmen.

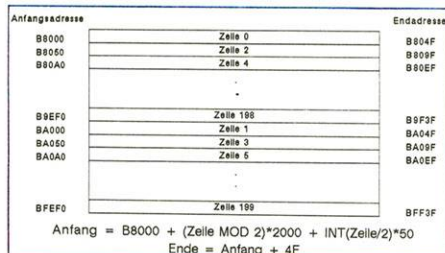


Bild 5. Der Aufbau des Bildschirmspeichers in den Grafikmodi

Wir widmen uns zuletzt einer Gruppe von Funktionen, die mit dem Cursor zu tun haben. Die erste setzt die Position des Cursors, das heißt, bewegt den Cursor zu der gewünschten Stelle, und die zweite fragt die Position des Cursors ab. Beide sind bereits in Hochsprachen zu finden: »GotoXY« in Turbo Pascal und »Locate« in GW-Basic setzen die Cursorposition, und »WhereX« und »WhereY« (Pascal), beziehungsweise »CsrLin« und »Pos« (Basic) geben die aktuelle Cursorposition zurück. Wie üblich weisen die Routinen der Hochsprachen eine Schwäche auf: Sie arbeiten nur auf der aktuellen Seite. Man muß also eine Bildschirmseite sichtbar machen, um den Cursor bewegen zu können. Natürlich geht es für den DOS-Profi auch anders, und der andere Weg ist nicht nur leistungsfähiger, sondern auch schneller.

Es geht wieder über das BIOS, das freundlicherweise die Cursorpositionen auf allen Seiten speichert. An der Adresse 450hex beginnt ein Array von 16 Byte, in acht Paaren organisiert. Jedes Paar besteht aus der Spaltenangabe im ersten Byte und der Zeilenangabe im zweiten Byte, die zusammen die aktuelle Cursorposition auf der jeweiligen Seite abbilden. Das Lesen dieser Bytes ersetzt völlig die Funktion »Cursorposition abfragen« und das Schreiben ersetzt mit einer Ausnahme die Funktion »Cursorposition setzen«. Die Ausnahme ist, daß die neu geschriebene Cursorposition erst dann in Kraft tritt, wenn auf den Bildschirm geschrieben wird, während die Funktion »Cursor setzen« sofort wirkt. Wenn die Variable »Seite« mit der Nummer der gewünschten Seite belegt ist, wird die Cursorposition in GW-Basic so gelesen:

```
def seg = &H40
Zeile = peek (&H50 + Seite*2 + 1)
Spalte = peek (&H50 + Seite*2)
def seg
In Turbo Pascal funktioniert es so:
Zeile := Mem [$40 : $50 + Seite*2 + 1];
Spalte := Mem [$40 : $50 + Seite*2];
Die letzte Routine, mit der wir uns im
```

```
0: FUNCTION ReadChar (Zeile, Spalte, Seite: INTEGER): INTEGER;
1: VAR i: INTEGER;
2: BEGIN
3: IF (Zeile<1) OR (Spalte<1) OR (Seite<0) OR (Zeile>25) THEN
  EXIT;
4: i := Mem [$40:$49];
5: IF (i<2) THEN BEGIN
6: i := 1;
7: IF (Spalte>40) OR (Seite>7) THEN EXIT;
8: END
9: ELSE BEGIN
10: i := 2;
11: IF (Spalte>80) OR (Seite>3) THEN EXIT;
12: END;
13: i := Seite*i*2048 + (Zeile-1)*i*80 + (Spalte-1)*2;
14: ReadChar := Mem [$B800:i] + Mem [$B800:i+1] * 256;
15: END;
```

Listing 8. Die Turbo Pascal-Routine, die direkt aus dem Bildschirmspeicher liest

```
0: VAR Regs: RECORD ax,bx,cx,dx,bp,si,di,ds,es,fi: INTEGER;
1: END;
2:
3: FUNCTION SetCursor (Anfang, Ende: INTEGER): BOOLEAN;
4: BEGIN
5: IF (Anfang<0) OR (Ende<0) THEN
6: Regs.cx := $2000
7: ELSE
8: Regs.cx := (Anfang * 256) + Ende;
9: Regs.ax := 256;
10: Intr (16, Regs);
11: END;
```

Listing 9. Cursorgröße in Turbo Pascal setzen

Byte 0	Byte 1	Byte 2	Byte 3	...	Byte 78	Byte 79
Pixel 0 bis 3	Pixel 4 bis 7	Pixel 8 bis 11	Pixel 12 bis 15		Pixel 312 bis 315	Pixel 316 bis 319

Pixel 0		Pixel 1		Pixel 2		Pixel 3	
Bit 1	Bit 0	Bit 1	Bit 0	Bit 1	Bit 0	Bit 1	Bit 0

Bit 1	Bit 0	Farbe
0	0	Hintergrund (0)
0	1	1
1	0	2
1	1	3

Bild 6. Zeilen und Punkte in den Grafikmodi 4 und 5

Rahmen der Textmodi beschäftigen, bestimmt die Größe des Cursors. Um diese Routine zu verwenden, muß man wissen, wie sich ein Zeichen auf dem Bildschirm aus Punkten zusammen setzt. Die Punktmatrix besteht aus acht Reihen von je acht Punkten. Die Reihen werden von oben (Reihe 0) nach unten (Reihe 7) durchnummeriert. Mit der Routine »Cursorgröße setzen« legt man fest, aus welchen Reihen der Cursor besteht. Diese Routine ist wieder in Basic zu finden, als Teil des Locate-Befehls. Zum Beispiel erzeugt der Befehl

```
locate ...,7,7
den bekannten Unterstrich als Cursor, während der Befehl
locate ...,0,7
einen Block als Cursor ergibt. Zur Abwechslung könnte man
locate ...,3,4
ausprobieren. Das Ergebnis ist ein doppelter Strich in der Mitte des Feldes. Auch in Basic kann man problemlos den Cursor ausschalten (unsichtbar machen):
locate ,,0
schaltet den Cursor ab und
locate ,,1
schaltet ihn wieder ein.
```



```

5 DIM PAL(4)
6 PAL(0)=0: PAL(1)=&H20: PAL(2)=&H10: PAL(3)=&H30
10 SCREEN 1,0
11 CLS: OUT &H3D9,0
20 LINE (0,0)-(59,59),1,BF
30 LINE (60,60)-(119,119),2,BF
40 LINE (120,120)-(179,179),3,BF
50 P=0
60 LOCATE 1,24: PRINT "Palette";P;
70 LOCATE 2,14: INPUT "<RETURN> drucken";I$
80 P=(P+1) MOD 4: OUT &H3D9,PAL(P): GOTO 60

```

Listing 10. Ein Basic-Programm verwendet die dritte und vierte Palette

```

0: VAR Regs: RECORD ax,bx,cx,dx,bp,si,di,ds,es,fl: INTEGER;
1:      END;
2:
3: FUNCTION ReadChar: CHAR;
4: BEGIN
5:   Regs.ax := $0800;
6:   Intr (16,Regs);
7:   ReadChar := Chr(Lo(Regs.ax));
8: END;

```

Listing 11. Eine Turbo Pascal-Routine kann über das BIOS Zeichen in einem Grafikmodus vom Bildschirm ablesen

Da diese Funktionen in Turbo Pascal fehlen, ist in Listing 9 die Routine »SetCursor« abgedruckt, die die Größe und Sichtbarkeit des Cursors bestimmt.

Ein weiterer Tip: Es gibt keine Funktion, die die aktuelle Größe des Cursors zurückgibt. Hier bietet DOS wieder Hilfe: In den beiden Byte ab Adresse 460hex stehen die Anfangs- und Endzeilen des Cursors. Seltsamerweise folgt erst die End- und dann die Anfangszeile, aber wenn das bekannt ist, bereitet der Umstand kein Problem mehr. Diese beiden Byte können analog zum Cursorposition-Array gelesen werden, jedoch hat das Schreiben in diese Bytes keine sofortige Auswirkung.

Die Grafikmodi der CGA-Karte im Detail

Wie oben schon erwähnt, arbeitet die CGA-Karte in einem Grafikmodus anders, als im Textmodus. Dieser Unterschied ist auch im logischen Aufbau des Bildschirmspeichers zu sehen. Im Bild 5 wird dieser Aufbau schematisch dargestellt. Zunächst sieht der Speicher ganz ordentlich aus, die ersten 80 Byte enthalten die erste Bildschirmzeile (Zeile Nummer 0). Aber dann muß man die zweite Zeile suchen... und suchen... und suchen... Die zweite Zeile (Zeile Nummer 1) steht nämlich nicht direkt nach der ersten, wie in den Textmodi, sondern am Anfang der zwei-

ten Hälfte des Bildschirmspeichers, genauer ab der Adresse »B800:2000«. Was steht dann aber nach der ersten Zeile? Na klar, die dritte Zeile! Und nach der zweiten steht die vierte, und nach der dritten die fünfte, und so weiter bis wir endlich alle 200 Zeilen im Speicher haben. Zusammengefaßt ist der Bildschirmspeicher in den Grafikmodi in zwei Hälften geteilt, die zusammen gerade ein vollständiges Bild ergeben. Die erste Hälfte beinhaltet alle geraden Zeilen (Nummern 0, 2, 4, etc.), eine nach der anderen und die zweite Hälfte alle ungeraden Zeilen (1, 3, 5, etc.) in der gleichen Reihenfolge. Die Frage, wieso der Speicher so (un)organisiert wurde, ist gestattet, und bekommt eine leider allzu übliche Antwort: Die Hardware wurde dadurch ein bißchen einfacher. Daß die Programmierung der Hardware dadurch wesentlich schwieriger und viele Programme dadurch sehr viel langsamer werden, war dem Hardware-Entwickler anscheinend nicht so wichtig. Daher müssen nun die PC-Anwender mit einer etwas verwirrenden Speicherstruktur leben.

In allen Grafikmodi ist diese Struktur gleich und zusätzlich ist eine Zeile immer 80 Byte lang. So läßt sich eine Formel für die Anfangsadresse einer Zeile leicht entwickeln:

$$(\text{Zeile MOD } 2) * 8192 + (\text{INT}(\text{Zeile} / 2) * 80)$$

Weil die anderen Eigenschaften des Speichers und die Verwendung der BIOS-Routinen stark vom jeweils gewählten Grafikmodus abhängen, schauen wir zunächst die Modi 4 und 5 (320x200 Pixel in vier Farben beziehungsweise Graustufen) an, und anschließend betrachten wir den hochauflösenden Modus 6.

Der vierfarbige Modus 4 der CGA-Karte ist der beliebteste und am meisten verwendete Grafikmodus des PC. Dieser Modus bietet eine relativ hohe Auflösung in mehreren Farben, was kein anderer Modus der CGA-Karte kann; er findet deswegen in allen Bereichen seinen Einsatz. Wie eine Zeile in diesem Modus gespeichert wird, zeigt Bild 6. Jeder Punkt auf dem Bildschirm wird in zwei Bit abgebildet, deren Wert die Farbe des Punktes bestimmt. Deshalb gibt es in diesem Modus vier mögliche Farben. Es werden genau vier Grafikpunkte in jedem Byte gespeichert: Die linken vier Punkte einer Zeile im ersten Byte, die nächsten vier im zweiten Byte und so weiter. In Bild 6 ist zu sehen, wie die vier Punkte innerhalb eines Byte abgelegt werden. Damit läßt sich die Formel für die Adresse einer Grafikzeile erweitern, um die genaue Position eines Punktes zurückzugeben. Hier reicht die Byte-Adresse nicht aus; es ist eine zusätzliche Information nötig, in welchen zwei Bit der Punkt gespeichert ist. Diese Information berechnet sich als ein Byte, worin genau zwei Bit gesetzt sind und zwar die zwei, die den gesuchten Punkt abbilden. Dieses Byte wird »Maske« genannt. Die Formeln sind:

$$\text{Adr} = (\text{Zeile MOD } 2) * 8192 + \text{int}(\text{Zeile} / 2) * 80 + \text{int}(\text{Spalte} / 4)$$

$$\text{Maske} = 192 / (4 \uparrow (\text{Spalte MOD } 4))$$

Die letzte Zeile dürfte etwas unverständlich sein. Für diejenigen, die sich im hexadezimalen Zahlenbereich zuhause fühlen, kann man es auch so ausdrücken:

$$\text{Maske} = \text{shr}(\$C0, 2 * (\text{Spalte MOD } 4))$$

Beide Formeln für die Maske sind identisch, von der Schreibweise einmal abgesehen. Mit diesen Informationen lassen sich ein paar Programmzeilen schreiben, die einen direkten Zugriff auf dem Bildschirmspeicher implementieren. Es gibt auch BIOS-Routinen, um Pixel zu schreiben und zu lesen. Aber da fast alle Sprachen diese Funktionen schon beinhalten, ist es müßig, sich an dieser Stelle weiter damit zu be-

schäftigen. Die entsprechenden Pascal-Zeilen sehen so aus:

```
Adr := (Zeile MOD 2)*8192 + Int(Zeile/2) + Int(Spalte/4);
Farbe := Shl (Farbe, 2*(3-(Spalte MOD 4)));
```

```
Maske := Shr ($C0, 2*(Spalte MOD 4));
Mem ($B800:Adr) := Mem ($B800:Adr)
AND (NOT Maske) OR Farbe;
```

Die Berechnung der Adresse und Maske sind schon bekannt und die Farbe wird analog zur Maske berechnet. Die letzte Zeile erledigt dann den Rest, nachdem in den ersten drei Zeilen die Vorbereitungen getroffen wurden. In der vierten Zeile wird das Byte aus dem Bildschirmspeicher gelesen, um die anderen drei Punkte wieder schreiben zu können. »Maske« löscht die beiden Bit, die zu dem zu schreibenden Punkt gehören, und danach werden die entsprechenden Bits gesetzt, um die gewünschte Farbe zu bekommen. Das Ganze wird sofort wieder in den Bildschirmspeicher geschrieben und schon ist die Sache erledigt.

Wenn man einen Punkt nur lesen möchte, geht man ähnlich vor. Die Adresse und die Maske werden gleich berechnet, die Farbe fällt weg – die suchen wir nämlich! Ist dann auch die Farbe gefunden, müssen lediglich die beiden Bit ganz nach rechts gebracht werden, um eine Zahl von 0 bis 3 zurückzuliefern. Das ganze geht so:

```
Adr := (Zeile mod 2)*8192 + int(Zeile/2) + int(Spalte/4);
```

```
Maske := Shr ($C0, 2*(Spalte mod 4));
```

```
Farbe := Shl (Mem($B800:Adr) and (not Maske), 2*(3-(Spalte mod 4)));
```

Diese beiden Programmteile lassen sich relativ einfach in Basic umsetzen. Nehmen Sie die Routinen von den Listings 5 und 6 als Vorbild, weil in diesen beiden Listings die Sprach-elemente der beiden Sprachen leicht zu vergleichen sind.

Eine schnelle Berechnung zeigt, daß in diesen beiden Modi der Speicher praktisch voll ausgenutzt wird. Wir haben 320x200 gleich 64000 Punkte, geteilt durch 4 Punkte pro Byte ergibt 16000 Byte, um ein Bild zu speichern. Von den 16 KByte auf der CGA-Karte sind nur noch 384 Byte übrig – und die werden nicht verwendet.

Jetzt wissen Sie, daß jeder Punkt in zwei Bit gespeichert wird und wie Sie die Bits im Bildschirmspeicher finden können. Aber wie bestimmen die Bits die Farbe des Punkts, und

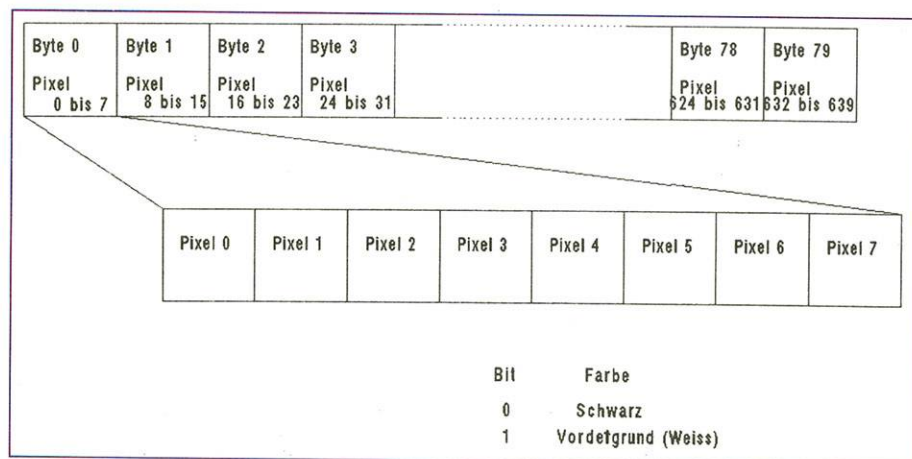


Bild 7. Zeilen und Punkte in dem Grafikmodus 6

welche vier Farben von den sechzehn möglichen werden verwendet?

Diese Fragen beantwortet ein neues Konzept, die Farbpalette. Die Palette beinhaltet, wie bei einem Maler, eine Auswahl aller zur Verfügung stehenden Farben. Bei der Original-IBM-Farbgrafikkarte gibt es zwei Paletten, die Paletten 0 und 1 aus Tabelle 5. Inzwischen können fast alle anderen CGA-Karten mit einer von vier Paletten arbeiten, wobei die beiden neuen (Paletten 2 und 3 in der Tabelle) »helle Versionen« der Original-Paletten sind. Hier hilft nur das Ausprobieren mit einem kleinen Programm, das die drei Farben auf dem Bildschirm zeigt und dann jede der vier möglichen Paletten nacheinander angezeigt. Ein Basic-Programm zu diesem Zweck finden Sie in Listing 10. Erhalten Sie vier verschiedene Ergebnisse, besitzen Sie eine moderne CGA-Karte. Dies verdeutlicht nebenbei eine interessante Eigenschaft der Paletten, daß nämlich beim Wechsel einer Palette, alle Farben auf dem Bildschirm gleichzeitig wechseln.

Eine weitere Eigenschaft aller Paletten stellt die Hintergrundfarbe dar. Diese Farbe hat immer die Nummer 0, wobei aber jede beliebige Farbe als Hintergrundfarbe deklariert werden kann. Das heißt, in jeder Palette sind drei Farben festgelegt, während die vierte frei wählbar ist. Diese freie Farbe ist aber die Hintergrundfarbe. Dies bedeutet, daß ein gelöschter Bildschirm stets diese Farbe hat.

Die Palette kann man bequem von Pascal oder Basic aus setzen. In Turbo Pascal sind dazu die Zeilen

```
Graphbackground (White);
Palette (1);
```

nötig, um die Farben im Modus 4 zu bestimmen. GW-Basic schafft das

Ganze mit einer einzigen Anweisung, nämlich

```
color 15,1
```

Allerdings lehnt Basic es grundsätzlich ab, die Paletten 2 oder 3 zu setzen. Je nach Basic-Version bekommt man eine Fehlermeldung, oder die Anweisung (zum Beispiel »color 15,2«) wird einfach verschluckt. Es gibt einen einfachen Umweg, der einen einzigen direkten Zugriff auf die Bildschirmsteuerlogik verlangt. Zunächst wählen Sie Ihre Hintergrundfarbe aus, danach die gewünschte Palette. Suchen Sie in der Tabelle die Palette und addieren Sie den nebenstehenden Wert zu der Nummer Ihrer Hintergrundfarbe. Setzen Sie das Resultat anstelle von »x« in der folgenden Basic-Anweisung ein:

```
out &H3D9,x
```

Und das war es auch. Die Farbpalette ist dadurch nach Wunsch eingestellt worden. Die Paletten und Werte sind:

Palette	Wert
0	0
1	32
2	16
3	48

Natürlich gibt es auch eine BIOS-Funktion, um die Palette zu setzen, aber die ist inzwischen überflüssig. Wenn man die aktive Palette abfragen will, sieht es düster aus. Es gibt überhaupt keine Möglichkeit, diese einfache Frage zu beantworten. Weder Turbo Pascal noch GW-Basic bietet solch eine Funktion an und sogar das BIOS schweigt an dieser Stelle. In keiner gängigen Literatur wird erwähnt, ob das BIOS oder das DOS die Palette irgendwo speichert. Jetzt könnte man denken, wenn wir oben einen Ausgabe-Befehl verwendet haben, um die Palette zu setzen, könn-

ten wir genauso den entsprechenden Eingabe-Befehl verwenden, um die Palette auszulesen. Leider stimmt das nicht – jeder lesende Zugriff auf die Port-Adresse 3D9hex gibt lediglich den Wert 255 zurück, egal was vorher geschrieben wurde. Fazit: Sie sollten sich merken, was für eine Palette gerade ausgewählt ist, da eine Abfrage hierzu nicht existiert.

Der letzte Modus der CGA-Karte ist der hochauflösende Modus 6. Hier besteht jede Zeile aus 640 Punkten, das Doppelte der Modi 4 und 5. In diesem Modus wird jeder Punkt als ein Bit gespeichert, was die gleiche Ausnutzung des Bildspeichers ergibt, wie bei den anderen beiden Grafikmodi. Der Aufbau einer Zeile ähnelt im übrigen den Modi 4 und 5 und ist im Bild 7 zu sehen. Die Zeilenadressen sind sogar völlig identisch, da eine Zeile genau so viele Bits benötigt wie vorher, jedoch beansprucht die Maske nur ein einziges Bit. Die beiden Formeln für Byte-Adresse und Bit-Maske eines Punkts in Modus 6 sind:

$\text{Adr} = (\text{Zeile mod } 2) * 8192 + \text{int}(\text{Zeile} / 2) * 80 + \text{int}(\text{Spalte} / 8)$

$\text{Maske} = 128 / (2 \uparrow (\text{Spalte mod } 8))$

Wie vorher ist die Formel für die Maske alternativ auch so zu schreiben:

$\text{Maske} = \text{shr}(\$80, (\text{Spalte mod } 8))$

Um Punkte zu lesen und zu schreiben, greift man entweder wieder zu den BIOS-Routinen oder programmiert direkte Zugriffe auf den Bildschirmspeicher. Die Hochsprachen GW-Basic und Turbo Pascal können dies bereits und erledigen es mit den gleichen Funktionen wie in den Modi 4 und 5. Da es nur ein Bit pro Punkt gibt, existieren auch nur zwei Farben in diesem Modus. Die Geschichte wird bekannt sein: Die Original-IBM CGA-Karte konnte nur Schwarz (Hintergrund) und Weiß (Vordergrund) im Modus 6 darstel-

len. Inzwischen ist es aber üblich, daß die Vordergrundfarbe frei ausgewählt werden kann. Zu diesem Zweck findet man in Turbo Pascal die Prozedur »HiResColor«, Basic und das BIOS bieten keine Möglichkeiten dazu. Falls Sie das Glück haben, im Besitz eines neueren PC zu sein, können Sie die folgende Basic-Zeile verwenden, um eine beliebige Vordergrundfarbe im Modus 6 einzusetzen: `OUT &3D9, Farbe`

»Farbe« kann jeden Wert von 0 bis 15 annehmen. Eine Umprogrammierung dieses Wertes ändert alles auf dem Bildschirm zur neuen Farbe.

Grafik ist schön und gut, aber meistens benötigt man nicht reine Grafik, sondern eine Mischung aus Grafik und Text. Wie geht das? Zunächst sollte man wissen, daß Funktionen wie »Print« in Basic oder »WriteLn« in Pascal auch im Grafikmodus funktionieren. Das heißt, man kann von einer Hochsprache aus ohne weiteres Grafik und Text mischen. Die Farbe, in der der Text ausgegeben wird, hängt vom Modus und gegebenenfalls von der Palette ab. Im Modus 4 ist die Textfarbe stets Farbe 3 der aktuellen Palette, im Modus 5 natürlich die entsprechende Graustufe. Die Textfarbe im Modus 6 ist logischerweise die Vordergrundfarbe.

Ein Problem bei der Vermischung von Text und Grafik ist, daß die Bildschirmkoordinaten für den Text und die Grafik unterschiedlich sind: Für Text verwendet man nach wie vor Reihe und Spalte in einer 40×25 beziehungsweise 80×25 Textmatrix, für eine Grafik hingegen die X- und Y-Koordinaten des Grafik-Bildschirms, also 320×200 beziehungsweise 640×200 . Die Umwandlung ist jedoch nicht sehr schwierig:

$\text{Text-Reihe} = (\text{GrafikX} / 8) + 1$

$\text{Text-Spalte} = (\text{GrafikY} / 8) + 1$

Die Formeln gelten für alle drei Grafikmodi, da ein Text in den Modi 4

und 5 doppelt so breit ist. Die letzte Funktion in der Bildschirmgruppe klingt einfach, aber es steckt viel dahinter. Inzwischen ist bekannt, wie man ein Zeichen vom Bildschirm liest, wenn ein Textmodus aktiv ist. Wie aber geht man vor, wenn der aktuelle Modus ein Grafikmodus ist? Basic macht auch hier das Leben einfach: Die Screen-Funktion gibt das Erwartete zurück, sowohl in einem Textmodus als auch in einem Grafikmodus. Hier muß man lediglich wissen, daß die Screen-Funktion auch in einem Grafikmodus Text-Koordinaten erwartet. Verwenden Sie die beiden Formeln oben, um die Grafik-Koordinaten umzuwandeln.

Turbo Pascal besitzt keine ähnliche Funktion und die Routine im Listing 8 wird nur Unsinn zurückliefern, wenn sie in einem Grafikmodus aufgerufen wird. Das BIOS beinhaltet aber eine geeignete Routine und mit der Funktion im Listing 11 steht diese Routine dem Pascal-Anwender zur Verfügung. Das Interessante an dieser Routine wird deutlich, wenn Sie daran denken, wie ein Zeichen im Bildschirmspeicher abgelegt wird, wenn ein Grafikmodus aktiviert ist. Das Zeichen wird nämlich, wie alles andere auch, als Muster aus einzelnen Punkten gespeichert. Die BIOS-Routine muß also ein Muster aus dem Bildschirm herauslesen und dieses Muster mit allen möglichen Zeichensmustern vergleichen, um das Zeichen zu identifizieren. Dies ist keine leichte Aufgabe! Das BIOS spart uns also eine Menge Arbeit an dieser Stelle. Damit ist unsere Reise durch die Bildschirmsteuerung mit der Farbgrafikkarte zu Ende. Was hier geboten wurde, sollte Ihre Arbeit erleichtern und helfen, die Grenzen des Möglichen ein bißchen weiter zu stecken – und vor allem viele Anreize für eigene Programme geben.

(Thomas Little/ev)

Eine Bitte an unsere Abonnenten

Vermerken Sie bei Schriftverkehr und Zahlungen neben der vollständigen Anschrift stets Ihre Abo-Nummer.

Sie vermeiden damit unnötige Verzögerungen bei der Bearbeitung Ihres Abonnements.

Vielen Dank

Ihre DMV-Versandabteilung

DER COMPILER DER

BCi Pascal ist ein leistungsstarkes **Programmentwicklungssystem**, das dem Programmierer die Erzeugung äußerst lesbarer, **hoch strukturierter** Programme ermöglicht.

BCi Pascal ist eine **strukturierte, system-orientierte** Programmiersprache. Die anwenderfreundlichen prozeduralen Mechanismen und die Vielfalt der Datentypen machen es ideal zum schnellen Lösen komplexer Aufgaben. Gleichzeitig ist **BCi Pascal** ein exzellentes **Entwicklungswerkzeug** für den **professionellen Anwender**.

BCi Pascals sorgfältig durchdachte **Erweiterungen** erlauben Zugriff auf die Maschine unter Erhaltung der strukturierten Eigenschaften. Dieses beinhaltet **direkten Speicherzugriff** und den Aufruf von Assembler-routinen.

Zweisprachig. **BCi Pascal** ist eine hervorragende **Lernsprache für den deutschsprachigen Anfänger**. Alle Fehlermeldungen sind auf deutsch. Deutsche Schlüsselwörter können mit ihrem Pendant der ISO-Pascalnorm vermischt werden. Dabei sind auch die **deutschen Umlaute** zulässig. Daher ist **BCi Pascal** nicht nur für didaktische Aufgaben geeignet, sondern auch für professionelle Systemprogrammierung.

BCi PASCAL

erhältlich im guten
Fachhandel oder direkt
beim **Exklusiv-**
Vertreiber:

**Händleranfragen
willkommen!**

The logo for DMV (Daten- und Medien-Verlagsgesellschaft mbH) features the letters 'DMV' in a bold, stylized font. The 'D' and 'M' are black, while the 'V' is red. The letters are slightly slanted and have a modern, geometric feel.

Große Speichermodelle. Im Gegensatz zu vielen gängigen Compilern ist **BCi Pascal** nicht auf 64 kB Code beschränkt, sondern erlaubt die volle Ausnutzung der **640 kB Adreßraum** unter MS-DOS. Das gleiche gilt für den Heap: auch dieser kann durch die Verwendung von »langen Zeigern« 64 kB überschreiten.

Außer der konventionellen Heap-Allokation (»new« und »dispose«) sind auch Libraryroutinen zur **automatischen Datenspeicherallokation** über das Standardsegment von 64 kB hinaus vorhanden. Dieses unterstützt die Verwendung des gesamten vorhandenen Speicherraums.

BCi PASCAL

Modulare Compilierung. Einzelne Module können separat zu .OBJ-Dateien compiliert und in Bibliotheken abgelegt werden. Beim Linkprozeß können damit – auch unter Einbeziehung von Modulen in **C, FORTRAN, Assembler** – Programme bis zu 640 kB erstellt werden. Die lauffähigen Programme sind im .EXE-Format.

Erweiterte Funktionen und Prozeduren. Funktionen können sowohl strukturierte als auch scalare Ergebnisse zurückgeben. Als Parametertypen für Unterprogramme wird die **volle ISO-Norm** unterstützt, dies beinhaltet **conformant arrays** sowie **Funktionen und Prozeduren als Parameter**. Darüberhinaus können Strings von variabler Länge und **externe Variablen** übergeben werden.

Bitte Coupon auf Seite 99 benutzen

DMV Daten- und Medien-Verlagsgesellschaft mbH

Postfach 250 · Fuldaer Str. 6

3440 Eschwege · Tel. (0 56 51) 87 02

90^{er} JAHRE

BCi PASCAL



Compilereigenschaften

2-phasiger, hoch optimierender Compiler, deutsche Fehler- und Systemmeldungen, erweiterter Zeichensatz (Umlaute und »ß«), 4 Speichermodelle bis 640 kB, modulare Compilierung, bedingte Compilierung, Strings variabler Länge erzeugt »native Code«, lange Zeiger (großer Heap), linkt mit verschiedenen Sprachen, Unterstützung des 8087/80287 Co-Prozessors, BCD Unterstützung, binäre, oktale, dezimale und hexadezimale Konstanten sind möglich. Das Compilat ist frei von Rechten des Herstellers (Run-Time Bibliotheken).

BCi Pascal Hotline: Jeden Dienstag von 17 – 20 Uhr unter der Rufnummer (0 56 51) 87 02

BCi Pascal/i86 Version 1.1, für den IBM PC, PC/XT und IBM AT sowie alle kompatiblen Mikrocomputer wie z.B. Schneider PC1512/1640 unter MS-DOS und PC-DOS Version 2.xx, 3.xx und höher.

Service vor und nach dem Kauf: Gegen Einsendung von 5,- DM in Briefmarken erhalten Sie das ausführliche **Info-Paket** zu BCi Pascal.
Erst prüfen – dann entscheiden!

Copyright 1987 BCi, West Germany und USA.

5 verschiedene Bibliotheken im Lieferumfang: Standard- mit Grafikroutinen, Integer-, Reell-, 8087- und BCD-Bibliothek.

Im Lieferumfang enthalten: **BCiGRAF** – Die Grafik-Bibliothek für BCi-Pascal.

Mit den in dieser Library enthaltenen Routinen können Sie auf Ihrem Rechner Programme in hochauflösenden oder Multicolor-Grafikmodus schreiben. Voraussetzung ist eine CGA- und EGA-Grafikkarte, die über das BIOS angesprochen werden kann. Die bei der Programmierung verwendeten modernen Algorithmen (Bresenham, Integer-DDA) sorgen für hohe Effizienz der Routinen. Mit den Turbo Pascal kompatiblen Prozeduren und Funktionen erhalten Sie die Möglichkeit, verschiedene Grafikmodi auszuwählen, Punkte zu setzen und zu testen, Linien und Kreise zu zeichnen und Grafik-Fenster zu definieren.

Unterstützung für den Profi durch Metakommandos. Bedingte Compilierung zur Erleichterung des Debugging und der Programmierung von portablen Programmen. Ein DOS-Errorlevelcode wird zurückgegeben, so daß der Compilierungs- und Bindeprozeß automatisiert werden kann.

Full-screen Editor mit **deutschen Fehlermeldungen**, Linker und Stapelprogramme im Preis eingeschlossen.

BCi Pascal ist didaktisch wertvoll aufgebaut und daher besonders geeignet für den **schulischen Einsatz**.

*unverbindliche Preisempfehlung

249,-DM*

Der Hercules-Karte ins Register geschaut

Die Hercules-Karte bietet neben einer Druckerschnittstelle auch einen Bildschirmanschluß, der eine monochrome Auflösung von bis zu 720x348 Punkten erlaubt. DOS International zeigt Ihnen, wie Sie die Fähigkeiten der Hercules-Karte voll nutzen.

Die Hercules-Karte erlaubt zwei Betriebsarten, den Text- und den Grafikmodus. Im Textmodus wird jedes Zeichen durch zwei Byte dargestellt. Im ersten Byte steht der ASCII-Wert des Zeichens. Da der Zeichengenerator in einem ROM auf der Karte untergebracht ist, kann der Zeichensatz im Textmodus nicht verändert werden. Das zweite Byte enthält das Attribut, wobei die Hercules-Karte hier folgende Möglichkeiten bietet:

Normale Anzeige: 07hex
Invers: 70hex
Unterstreichen: 01hex
Schwarztafeln: 00hex

Für blinkende Zeichen ist zu diesem Attributwert 80hex zu addieren, für hervorgehobene, helle Darstellung der Wert 08hex. Für ein blinkendes, hervorgehobenes und unterstrichenes Zeichen ergibt sich somit beispielsweise der Attribut-Wert 80hex + 08hex + 01hex = 89hex.

Die Adresse eines Zeichens im Bildschirmspeicher, das in Spalte X und Zeile Y steht, läßt sich mit der Formel $\text{Offset} = 160 * Y + 2 * X$ berechnen. Das Segment hierfür ist B000hex. Da die Hercules-Karte in diesem Modus zum IBM-Monochromadapter kompatibel ist, können die BIOS-Funktionen zur Textdarstellung verwendet werden. Den Grafikmodus jedoch unterstützt leider das BIOS nicht, so daß der Anwender hier auf eigene Unterprogramme zurückgreifen muß. Im Grafikmodus stehen dem Benutzer zwei Seiten zur Verfügung. Die erste Seite beginnt bei Adresse B0000hex, die zweite bei Adresse B8000hex. Spricht man die Karte von einem Assemblerprogramm aus an, so ist es zweckmäßig, diese Basisadresse durch 16 zu dividieren und in ein Segmentregister zu laden. Man kann dann die Byte-Adresse eines Punktes mit den Koordinaten X, Y folgendermaßen bestimmen:

$\text{Offset} = 90 * (Y \text{ Div } 4) + 8192 * (Y \text{ Mod } 4) + (X \text{ Div } 8)$

Die Nummer des jeweiligen Grafik-Bits innerhalb dieses Byte läßt sich mit $\text{Bit} = 7 - (X \text{ Mod } 8)$ ausrechnen. Ist dieses Bit gesetzt, so

wird der entsprechende Punkt auf dem Bildschirm hell dargestellt, wird das Bit gelöscht, so wird auch der Punkt auf dem Bildschirm gelöscht.

Die zentralen Steuerungsfunktionen zum Erzeugen des Video-Signals nimmt bei der Hercules-Karte ebenso wie beim IBM-Farb-/Grafikadapter (CGA) der Video-Controller der Motorola 6845 wahr. Der Baustein enthält ein

Adreß- und 18 Datenregister R0, R1, ..., R17. Um eines der Datenregister anzusprechen zu können, muß zunächst das Adreßregister unter der Portadresse 03B4hex mit der gewünschten Registernummer geladen werden. Auf das Datenregister kann man dann in einem zweiten Arbeitsgang über die Portadresse 03B5hex zugreifen. Die Register R0 bis R9 enthalten Formatparameter, wie zum Beispiel die Zahl der Punkte pro Zeile. Aus Hardwaregründen sind für den Grafikmodus nur die Standardeinstellungen nach der Tabelle sinnvoll.

R10 und R11 sind im Textmodus für Größe und Lage des Cursors zuständig: R10 enthält die Nummer der ersten Abtastzeile, die vom Cursor überlagert wird, R11 die Nummer der letzten. Im Grafikmodus müssen diese Register mit Null geladen werden. Die Register R12 und R13 enthalten die Startadresse des aktiven Teils des Bildspeichers. Im Normalfall werden sie mit Null initialisiert. Durch Verändern des Inhalts läßt sich ein Fenster über den Bildspeicher seitenweise, zeilenweise oder auch zeichenweise verschieben. Die Register R14 und R15 definieren die Cursorposition: Sie enthalten die durch 2 dividierte Zeichenadresse. Im Grafikmodus ist der Inhalt dieser Register unwesentlich.

Die Hercules-Karte gestattet auch den Anschluß eines Lichtgriffels. Er spricht auf die kurzen Lichtimpulse an, die der Leuchtschirm des Monitors emittiert, wenn er vom Elektronenstrahl getroffen wird. Aus dem Zeitpunkt eines solchen Lichtblitzes wird die Position des Lichtgriffels ermittelt. Wenn an Stift 1 des Lichtgriffelsteckers eine fallende Flanke auftritt, wird das »Lichtgriffel-Flip-Flop« gesetzt. Die steigende Flanke des Flip-Flop-Ausgangssignals bewirkt, daß der Video-Controller die entsprechende Bildspeicheradresse in den Registern R16 und R17 ablegt.

Neben den Registern des Videocontrollers enthält die Hercules-Karte eine Reihe weiterer Register und Flip-Flops, die unter Programmkontrolle verändert, in einem Falle auch gelesen werden können:

Das Anzeige-Steuerungs-Register hat die I/O-Adresse 03B8hex. Bit 1 entscheidet über die Betriebsart. Für Textmodus muß es gelöscht, für Grafikmodus gesetzt werden. Durch Löschen von Bit 3 wird der gesamte Bildschirm dunkel getastet, ohne daß dabei der Bildspeicherinhalt verändert wird. Bit 5 muß gesetzt sein, wenn im Textmodus das Blink-Attribut benutzt werden soll.

Ist Bit 5 gelöscht, so erhält Bit 7 des Attribut-Byte eine andere Funktion. Es erlaubt dann, einen hellen Zeichenhintergrund zu wählen. Bit 7 des Registers 3B8hex gibt die Nummer der gerade ausgegebenen Bildspeicherseite an. Die Bits 0, 2, 4 und 6 haben keine Bedeutung.

Das Konfigurations-Register mit der Portadresse 03BFhex hat nur zwei signifikante Bit: Ist Bit 0 gelöscht, kann nicht auf Grafikmodus geschaltet werden. Wenn beide Seiten des Bildspeichers benutzt werden sollen, muß Bit 1 gesetzt werden. Ist Bit 1 gelöscht, so kann auf die zweite Seite nicht zugegriffen werden. Dadurch wird der gleichzeitige Betrieb einer Hercules-Karte und eines Farb-/Grafik-Adapters möglich: Die Speicheradressen B0000hex bis B7FFFhex stehen der Hercules-Karte zur Verfügung, die Adressen B8000hex bis BBFFFhex dem Farb-/Grafik-Adapter.

Das Statusregister mit der Portadresse 03BAhex kann nur gelesen werden. Bit 3 gibt den Zustand von Bit 3 des Anzeige-Steuerungs-Registers, also das Bildschirm-Freigabe-Signal, wieder. Bit 0 ist während des Horizontal-Rücklaufs, Bit 7 während des Vertikal-Rücklaufs gesetzt. Mittels dieser Informationen kann ein Programm mit dem zeitlichen Ablauf der Bildschirmausgabe synchronisiert werden.

Immer wenn das Lichtgriffel-Flip-Flop vom Lichtgriffel gesetzt wurde, muß es vom Programm wieder gelöscht werden. Dies geschieht durch einen Schreibzugriff auf die I/O-Adresse 03BBhex. Das Flip-Flop kann auf ähnliche Weise unter Programmkontrolle gesetzt werden und zwar durch einen Schreibzugriff auf die Adresse 03B9hex.

Die wichtigsten Assembler-Unterprogramme für die Erzeugung einer Grafik zeigt Listing 1. Diese Unterprogramme schalten die Hercules-Karte in den Grafikmodus und wieder zurück in den Textmodus. Außerdem sind die Routinen zum Setzen, Löschen, Invertieren und Abprüfen eines Punktes implementiert. Die Routinen müssen mit der Assembleranweisung »include« in das Anwenderprogramm eingebunden und dann mit dem Call-Befehl aufgerufen werden. Das Demonstrationsprogramm in Listing 2 veranschaulicht die Verwendung dieser Grafikroutinen. Es erzeugt eine schräge Linie, die von links nach rechts über den Bildschirm wandert. (Frank Brendle/ev)

Register	Wert
R0	35hex
R1	2Dhex
R3	2Ehex
R4	07hex
R5	5Bhex
R6	02hex
R7	57hex
R8	57hex
R9	02hex

Tabelle. Standardeinstellungen der 6845-Register für den Grafikmodus


```

0: ; Grafiktreiber für Hercules-Karte
1: ; -----
2: ; *** DOS International, 1987 ***
3: ; -----
4:
5:
6: ; Adressen der Hercules-Karte:
7:
8: Index Equ 3B4H ; Index-Register der Hercules-Karte
9: Control Equ 3B8H ; Control-Register der Hercules-Karte
10: TextMod Equ 28H ; Wert um auf Text zu schalten
11: Grafik Equ 10 ; Wert um auf Grafik zu schalten
12: Config Equ 3BFH ; Konfigurations-Schalter
13: Page1 Equ 0B000H ; Startsegment der 1. Bildschirmseite
14: Length Equ 4000H ; Länge einer Seite in Worten
15: Video Equ 10H ; Interrupt zum Aufruf von Bildschirm-
16: ; Unterprogrammen.
17:
18: ; Videocontroller auf Grafik schalten.
19: ; 1. Seite anwählen, Bildschirm löschen.
20:
21: Graph: Mov DX,Config ; in Full-Modus schalten.
22: Mov AL,3
23: Out DX,AL
24: Mov SI,Offset GTable ; Initialisierungs-Tabelle.
25: Cld ; Richtungsflagge löschen.
26: Mov DL,Low Index ; Adresse des 6845.
27: Mov CX,12 ; Zahl der Register.
28: Push CS ; CS nach DS schaffen.
29: Pop DS
30: Graph1: LodSW ; Initialisierungswert holen
31: Out DX,AX ; und abspeichern.
32: Loop Graph1 ; Nächster Wert.
33: Mov DL,Low Control ; Control-Register.
34: Mov AL,Grafik ; 1. Seite selektieren.
35: Out DX,AL
36: Mov AX,Page1 ; Bildschirminhalt löschen.
37: Mov ES,AX ; Basisadresse nach ES.
38: Mov CX,Length ; Länge des Bildschirms.
39: Xor AX,AX ; Füllmuster in AX aufsetzen.
40: Rep Stosw ; Speicher löschen.
41: Ret
42:
43: ; Bildschirmadapter wieder in den Textmodus schalten.
44:
45: Text: Xor AX,AX ; BIOS-Unterprogramm aufrufen.
46: Int Video
47: Ret
48:
49: ; Punkt zeichnen (Y-Koordinate in AX, X-Koordinate in BX,
50: ; DL=0 --> Punkt setzen, DL=1 --> Punkt löschen, sonst
51: ; Punkt invertieren).
52:
53: Plot: Call GetPar ; Bytenummer u. Maske berechnen.
54: Or DL,DL ; Punkt setzen?
55: Jnz Plot1
56: Or Byte Ptr ES:[DI],AL ; Punkt wird gesetzt.
57: Ret
58: Plot1: Dec DL ; Punkt löschen?
59: Jnz Plot2
60: Not AL ; Punkt wird gelöscht.
61: And Byte Ptr ES:[DI],AL
62: Ret
63: Plot2: Xor Byte Ptr ES:[DI],AL ; Punkt wird invertiert.
64: Ret
65:
66: ; Abprüfen, ob ein Punkt gesetzt ist (Y-Koordinate in AX,
67: ; X-Koordinate in BX, CL=0 --> Punkt nicht gesetzt,
68: ; CL=1 --> Punkt gesetzt).
69:
70: GetCol: Call GetPar ; Bytenummer u. Maske berechnen.
71: And AL,Byte Ptr ES:[DI] ; Punkt abprüfen.
72: Jz GetCol1 ; Punkt war nicht gesetzt.
73: Mov AL,1 ; AL=1 --> Punkt war gesetzt.
74: GetCol1: Ret
75:
76: ; Bytenummer und Bitmaske berechnen. (X-Koordinate in BX,
77: ; Y-Koordinate in AX, Rückgabe der Maske in AL und Byte-
78: ; nummer in DI).
79:
80: GetPar: Mov DI,AX ; Zeilenoffset berechnen.
81: ShR AX,1 ; Offsetberechnung:
82: ShR AX,1 ; OfS = 90 * (Y Div 4)
83: Mov AH,90 ;
84: Mul AH ; 2000H * (Y Mod 4)
85: And DI,3
86: RoR DI,1
87: RoR DI,1
88: RoR DI,1
89: Add DI,AX
90: Mov AX,Page1 ; ES mit Basisadresse laden.
91: Mov ES,AX
92: Mov CL,BL ; X-Koordinate für Bitmaske.
93: ShR BX,1 ; Byte-Offset.
94: ShR BX,1
95: ShR BX,1
96: Add DI,BX ; Adresse steht jetzt in DI.
97: Mov AL,128 ; Bitmaske berechnen.
98: And CL,7 ; Bitnummer steht jetzt in CL.
99: ShR AL,CL

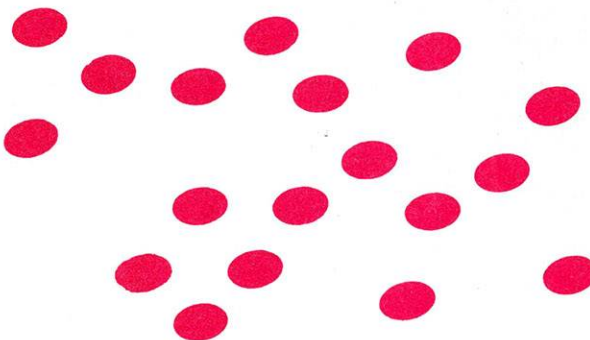
```

```

100: Ret
101:
102: ; Tabellenwerte, um auf Grafik zu schalten:
103:
104: GTable: DW 3500H, 2D01H, 2E02H, 0703H, 5B04H, 0205H,
105: DW 5706H, 5707H, 0208H, 0309H, 000AH, 000BH

```

Listing 1. Die Routinen »Punkt setzen«, »Punkt prüfen« und »Grafikmodus initialisieren« als Assembler-Unterprogramme



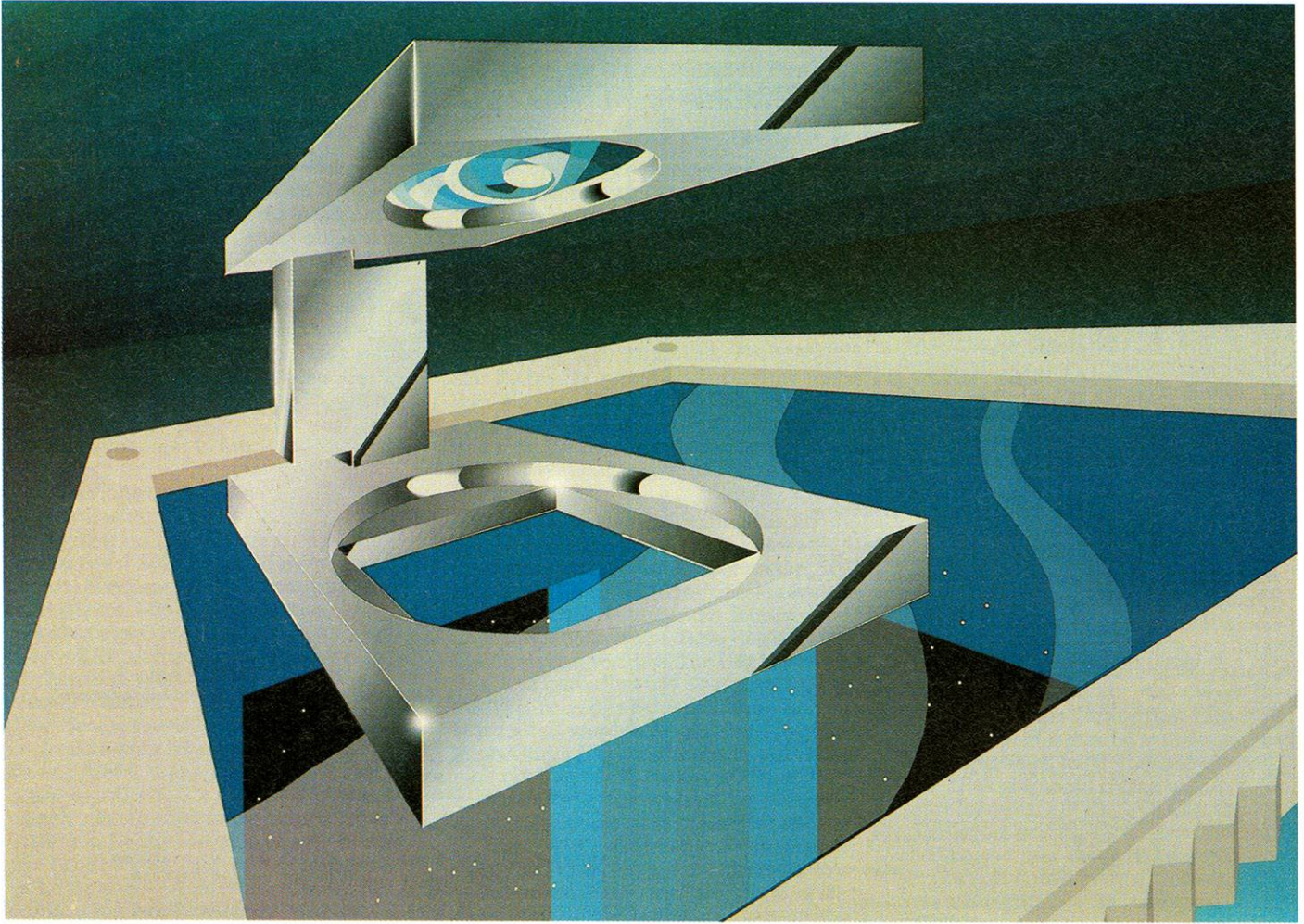
```

0: Page 60.132
1:
2: Title Beispiel für die Grafik-Unterprogramme.
3:
4: ; Funktions-Interrupts:
5:
6: Code Segment:
7: Org 100H ; .COM-File.
8: Assume CS:Code, DS:Code, ES:Code
9:
10: GrafTest Proc Near
11:
12: Start: Call Graph ; Grafikmodus einschalten.
13: Xor AX,AX ; Zähler initialisieren.
14: Xor BX,BX
15: Call Line ; Linie zeichnen.
16:
17: Start1: Inc BX ; Schritt nach rechts.
18: Call Line
19: Dec BX ; Letzte Linie wieder löschen.
20: Call Line
21: Inc BX
22: Cmp BX,719 ; Fertig?
23: Jnz Start1
24:
25: Mov AH,0 ; Auf Tastendruck warten.
26: Int 16H
27: Call Text ; Zurück in Textmodus.
28: Int 20H ; Programm beenden.
29:
30: ; Schräge Linie zeichnen.
31: Line: Push AX ; AX und BX retten.
32: Push BX
33: Loop: Mov DL,2 ; Linie invertierend zeichnen.
34: Push AX
35: Push BX
36: Call Plot ; Punkt setzen.
37: Pop BX
38: Pop AX
39: Inc BX ; Schritt nach rechts unten.
40: Inc AX
41: Cmp AX,347 ; Fertig?
42: Jnz Loop
43: Pop BX ; Register wiederherstellen.
44: Pop AX
45: Ret
46:
47: Include GrafProc.Asm ; Unterprogramme einbinden.
48:
49: GrafTest EndP
50: Code EndS
51: End Start
52: End

```

Listing 2. Ein kleines Demonstrationsprogramm, das eine wandernde Linie erzeugt.

Die EGA-Karte



Mit freundlicher Genehmigung der CART, Gesellschaft für Computergrafik mbH, Düsseldorf

klipp und klar

Durch den ständigen Preisverfall gewinnen EGA-Karten auch bei den privaten PC-Besitzern zunehmend an Verbreitung. Und weil man sich als Programmierer auf Dauer mit den EGA-Demos und Programmen von der Stange nicht zufrieden gibt, zeigen wir Ihnen, wie Sie das gute Stück selbst programmieren können.

Die EGA-Karte ist eine Grafikkarte, die die Qualität der Bildschirmausgabe Ihres PC beträchtlich erhöht. Sie besitzt zwei entscheidende Leistungsmerkmale. Zum einen arbeitet die EGA-Karte mit einer Auflösung von 640x350 Bildpunkten, so daß sie nahezu die Auflösung einer Hercules-Karte (720x348 Bildpunkte) erreicht. Einer CGA-Farbgrafikkarte (maximal 640x200 Bildpunkte) ist die

EGA-Karte sogar deutlich überlegen.

Zum anderen kann die EGA-Karte gleichzeitig 16 Farben aus einer Palette von 64 Farben darstellen. Die Hercules-Karte erlaubt dagegen lediglich die monochrome Darstellung von Grafiken, und mit der CGA-Karte lassen sich Grafiken bei einer Auflösung von 320x200 Bildpunkten nur vierfarbig und bei einer Auflösung von 640x200 Bildpunkten nur zweifarbig erzeugen.

Zudem ist die EGA-Karte CGA-kompatibel, so daß fast alle Programme mit der EGA-Karte arbeiten und keine neue Software erforderlich ist. Einzig und allein Programme, die direkt in die Register des Video-Controllers 6845 schreiben (zum Beispiel Actionspiele), sind auf einer EGA-Karte nicht lauffähig, weil diese Karte

nicht mit dem genannten Video-Controller bestückt ist.

Die Elektronik einer EGA-Karte ist nämlich so komplex, daß die Leistungen des 6845-Bausteins nicht ausreichen. Deshalb sind EGA-Karten mit modernen, hochintegrierten Gate Arrays bestückt, die speziell für die Funktionen der EGA-Karte programmiert wurden.

Der PC ruft die neuen Grafikmodi der EGA-Karte über die Funktion 0 des Video-Interrupt 10 auf. Diese Funktion übergibt einen Videomodus zur Auswahl von Text- oder Grafikmodus an die Grafikkarte. Tabelle 1 zeigt die Videomodi des PC in der Übersicht. Die Modi 0 bis 6 gelten für eine CGA-Farbgrafikkarte, der Modus 7 für einen Monochromadapter (MDA) oder eine Hercules-Karte, und die Modi 13

Die Videomodi des PC

Modus	Art	Grafikauflösung	Textauflösung	Farben
00	Text	320x200	40x25	Monochrom
01	Text	320x200	40x25	16
02	Text	640x200	80x25	Monochrom
03	Text	640x200	80x25	16
04	Grafik	320x200	40x25	Monochrom
05	Grafik	320x200	40x25	4
06	Grafik	640x200	80x25	Monochrom
07	Text	720x350	80x25	Monochrom
13	Grafik	320x200	40x25	16
14	Grafik	640x200	80x25	16
15	Grafik	640x350	80x25	Monochrom
16	Grafik	640x350	80x25	16 aus 64

Tabelle 1. Diese Videomodi stellen PC und EGA-Karte zur Verfügung

Die beiden Paletten des PC

0	1
Grün	Kobaltblau
Rot	Violett
Braun	Weiß
Hintergrund	Hintergrund
beliebig	beliebig

Tabelle 2. Die vier Farben im Grafikmodus 5 lassen sich nicht frei wählen, sondern werden durch zwei Paletten definiert

Die Screen-Befehle für die EGA-Karte

Befehl	Modus	Auflösung	Farben
SCREEN 7	13	320x200	16
SCREEN 8	14	640x200	16
SCREEN 9	16	640x350	16
SCREEN 10	15	640x350	Monochrom

Tabelle 3. Vier neue Screen-Befehle für die EGA-Karte sind bei GW-Basic 3.2 hinzugekommen

Der Aufbau des Farbattributs im Textmodus:

Bit	Funktion
0	Blau für Vordergrund
1	Grün für Vordergrund
2	Rot für Vordergrund
3	Leuchtend für Vordergrund
4	Blau für Hintergrund
5	Grün für Hintergrund
6	Rot für Hintergrund
7	Blinkend für Vordergrund

Tabelle 5. Das Farbattribut erlaubt im Textmodus 256 verschiedene Farbkombinationen von Vorder- und Hintergrundfarbe

Der Aufbau des Farbattributs im Grafikmodus

Byte	Farbe
00	Schwarz
01	Blau
02	Grün
03	Kobaltblau
04	Rot
05	Violett
06	Braun
07	Weiß
08	Grau
09	Hellblau
0A	Hellgrün
0B	Hellkobaltblau
0C	Hellrot
0D	Hellviolett
0E	Gelb
0F	Hellweiß

Tabelle 6. Im Grafikmodus bestimmen die Bits 0 bis 3 (Blau, Grün, Rot und Intensität) den Aufbau des Farbattributs

Der Video-Interrupt 10 mit EGA-Funktionen

AH-Register	Eingabe	Ausgabe	Bedeutung
00	AL=Videomodus Bit 7 von AL=1: Bildschirmspeicher wird nicht gelöscht	—	Definiere Videomodus (Textmodus oder Grafikmodus)
01	CH=000xxxx (xxxx=Startzeile von 0 bis 1F) CL=000xxxx (xxxx=Endzeile von 0 bis 1F)	—	Definiere Cursorgröße
02	DH=Zeile (0 bis 24) DL=Spalte (0 bis 79) BH=Bildschirmseite	—	Positioniere Cursor (0,0=oben links, 39/79,24/42=unten rechts)
03	BH=Bildschirmseite	DH=Zeile, DL=Spalte CH=Cursor-Startzeile CL=Cursor-Endzeile	Ermittle Cursorposition (0,0=oben links, 39/79,24/42=unten rechts)
04	—	AH=1: Lichtgriffel aktiv AH=0: Lichtgriffel inaktiv DH=Zeile (0 bis 24/42) DL=Spalte (0 bis 79) CH=Grafikzeile (0 bis 199/349) BX=Grafikspalte (0 bis 639)	Ermittle Lichtgriffel-Position (0,0=oben links, 39/79,24/42=unten rechts)
05	AL=Bildschirmseite (0 bis 7)	—	Wähle aktive Bildschirmseite
06	AL=Anzahl der Zeilen (frei werdende Zeilen am Fensterende werden gelöscht) (AL=0: Lösche ganzes Fenster) CH=Obere Zeile CL=Linke Spalte DH=Untere Zeile DL=Rechte Spalte BH=Farbattribut für gelöschte Zeilen	—	Scrolle Bildschirm aufwärts
07	AL=Anzahl der Zeilen (frei werdende Zeilen am Fensterbeginn werden gelöscht) (AL=0: Lösche ganzes Fenster) CH=Obere Zeile CL=Linke Spalte DH=Untere Zeile DL=Rechte Spalte BH=Farbattribut für gelöschte Zeilen	—	Scrolle Bildschirm abwärts
08	BH=Bildschirmseite	AL=Zeichen AH=Attribut (nur Textmodus)	Lese Zeichen und Farbattribut von Cursorposition
09	BH=Bildschirmseite AL=Zeichen BL=Farbnummer (0 bis F) BL=Attribut (nur Textmodus) CX=Anzahl der Zeichen	—	Schreibe Zeichen und Farbattribut auf Cursorposition
0A	BH=Bildschirmseite AL=Zeichen BL=Farbattribut (nur Grafikmodus) CX=Anzahl der Zeichen	—	Schreibe Zeichen auf Cursorposition
0B	Grafikmodus: BH=0: BL enthält Vordergrundfarbe BH=1: BL enthält Farbpalette BL=Vordergrundfarbe (0 bis F) Farbpalette (0 oder 1) Textmodus: BH=0: BL enthält Rahmenfarbe BL=Rahmenfarbe (0 bis 1F) BH=Bildschirmseite AL=Farbattribut (Bit 7 von AL=1: XOR-Modus) CX=Spalte (0 bis 639) DX=Zeile (0 bis 199/349)	—	Wähle Farbpalette oder Farbe für CGA-Videomodus 0 bis 5
0C	BH=Bildschirmseite AL=Farbattribut (Bit 7 von AL=1: XOR-Modus) CX=Spalte (0 bis 639) DX=Zeile (0 bis 199/349)	—	Setze Grafikpunkt im Grafikmodus
0D	BH=Bildschirmseite CX=Spalte (0 bis 639) DX=Zeile (0 bis 199/349)	AL=Farbattribut	Überprüfe Grafikpunkt im Grafikmodus
0E	AL=Zeichen, BL=Farbattribut (nur Grafikmodus) BH=Bildschirmseite	—	Schreibe Zeichen auf Bildschirm und bewege Cursor vorwärts (Teletype-Emulation)
0F	—	AH=Zeichen pro Zeile AL=Videomodus BH=Bildschirmseite (BH=0: Grafikmodus oder Modus 7) AH=Zeichen pro Zeile AL=Videomodus BH=Aktive Bildschirmseite	Ermittle aktuellen Bildschirmstatus
10	AL=Unterfunktion, weitere Register: Je nach Funktion	—	Setze Farbpaletten-Register
10.00	BH=Farbwert BL=Farbpaletten-Register	Siehe oben	Setze definiertes Farbpaletten-Register
10.01	BH=Farbwert	Siehe oben	Setze Rahmenfarbregister
10.02	DX:ES=Zeiger auf 17-Byte-Tabelle für die 17 Farbwerte	Siehe oben	Setze alle 16 Farbpaletten-Register und das Rahmenfarbregister
10.03	BL=0: Leuchtende Darstellung BL=1: Blinkende Darstellung AL=Unterfunktion (00 bis 03, 10 bis 12, 20 bis 23 und 30) Register: Je nach Funktion	Siehe oben	Wähle leuchtende oder blinkende Darstellung
11	—	CX=Aktuelle Zeichenhöhe DL=Zeilen pro Bildschirmseite BP:ES=Zeiger (Bedeutung: Je nach Funktion)	Lade Text- oder Grafikzeichensatz
12	BL=10	BH=0: Farbmonitor angeschlossen BH=0: Monochrom-Monitor angeschlossen BL=0: 64 KByte Bildschirmspeicher BL=1: 128 KByte Bildschirmspeicher BL=10: 192 KByte Bildschirmspeicher BL=11: 256 KByte Bildschirmspeicher	Ermittle Informationen zur EGA-Karte
12	BL=20	—	Aktiviere alternative Print-Screen-Routine für Zeichenhöhen ungleich 8 und 14
13	AL=00: Cursorposition bleibt erhalten AL=01: Cursor wird neu gesetzt AL=02: Zeichenkette enthält Attributbytes, Cursorposition bleibt erhalten AL=03: Zeichenkette enthält Attributbytes, Cursor wird neu gesetzt BH=Bildschirmseite BL=Attribut (wenn AL=00 oder 01) CX=Länge der Zeichenkette DX=Startposition für erstes Zeichen (Zeile, Spalte) BP:ES Zeiger auf Zeichenkette	—	Gebe Zeichenkette auf den Bildschirm aus

BX, CX, DX und die Segmentregister werden gesichert.
Alle Text- und Grafik-Koordinatangaben sind dezimal.

Tabelle 4. Die EGA-Karte erweitert den Video-Interrupt 10 um vier leistungsfähige Funktionen.

bis 16 kommen durch den Einsatz einer EGA-Karte neu hinzu.

Wie Sie sehen, kann die EGA-Karte mit den Grafikmodi 13 und 14 auch Grafiken in CGA-Auflösung in 16 Farben gleichzeitig darstellen. Volle Kompatibilität ist jedoch nur mit den Grafikmodi 4 bis 6 gewährleistet. Im Modus 5 müssen die Farben dann wie bei einer CGA-Karte aus zwei Paletten gewählt werden. Tabelle 2 zeigt die Farbzusammenstellung dieser Paletten.

Die EGA-Karte arbeitet gewöhnlich mit einer Zeichenmatrix von 8x14 Bildpunkten. Damit ist sie in der Zeichen-Auflösung einer Hercules-Karte ebenbürtig. Eine Hercules-Karte arbeitet zwar mit einer Matrix von 9x14 Bildpunkten, jedoch ist hier die erste Matrixspalte grundsätzlich unbenutzt, so daß sich ebenfalls eine effektive Auflösung von 8x14 Bildpunkten pro Zeichen ergibt. Die CGA-Farbgrafikkarte kann da bei einer Auflösung von 8x8 Bildpunkten pro Zeichen nicht mithalten.

Viele EGA-Karten können durch einen zweiten Zeichensatz ebenfalls mit einer 8x8-Matrix arbeiten. Die einzelnen Zeichen entsprechen dann in der Auflösung den Zeichen einer CGA-Karte, sind also deutlich lesbar. Doch bei der EGA-Karte lassen sich aufgrund der Gesamtauflösung von 640x350 Bildpunkten mit dem alternativen Zeichensatz nicht die üblichen 80x25 Zeichen, sondern 80x43 Zeichen gleichzeitig auf dem Bildschirm darstellen. Dies ist ein Zugewinn von 18 Textzeilen und vor allem bei Textverarbeitungs- und Tabellenkalkulations-Programmen ein bedeutender Vorteil.

Der Bildschirmspeicher der EGA-Karte ist viermal so groß wie der gesamte Speicher des Commodore 64

Die EGA-Karte besitzt einen Bildschirmspeicher von 256 KByte. Dieser Speicher ist »dual-ported«, das heißt, sowohl dem Prozessor als auch der Grafikkarte steht der Zugriff auf diesen Speicher offen. Der Speicherbereich beginnt ab der Adresse A0000hex und reicht bis zur Adresse BFFFFhex. Wenn ein Programm jedoch die CGA-Video modi 4 bis 6 anspricht, werden die Grafikdaten aus Kompatibilitätsgründen wie bei einer CGA-Karte ab der Speicheradresse B8000hex abgelegt. Bei der Auswahl von Videomodus 7 beginnt der Bild-

schirmspeicher aus dem gleichen Grund ab der Adresse B0000hex.

Wählt ein Programm die CGA-Textmodi 0 bis 3 aus, so stellt die EGA-Karte alle Zeichen automatisch in der höheren Auflösung von 320x350 beziehungsweise 640x350 Bildpunkten dar. Dies trägt entscheidend zur Lesbarkeit der Texte bei.

Die Definition der Rahmenfarbe ist bei der EGA-Karte im Gegensatz zur CGA-Farbgrafikkarte nur noch in den Grafikmodi 4 bis 6 und in den erweiterten Modi 13 und 14 möglich.

In den Textmodi 0 bis 3 und den hochauflösenden Modi 15 und 16 wird der Rahmen grundsätzlich in der Farbe dargestellt, die im Rahmenfarbregister angegeben ist.

Vielleicht haben Sie es schon selbst berechnet: Ein echtes EGA-Bild benötigt bei einer Auflösung von 640x350 Bildpunkten in 16 Farben (4 Bit) $640 \times 350 \times 4 = 896000$ Bit oder knapp 110 KByte Speicher.

Wozu sind nun die übrigen 146 KByte Speicher bei der EGA-Karte gut?

Der freie Speicherbereich einer EGA-Karte kann für die Definition von bis zu vier Zeichensätzen zu je 256 Zeichen (maximale Zeichenmatrix: 32x8 Bildpunkte) und für ein weiches Scrolling benutzt werden.

Bei dieser Art von Scrolling wird nicht der Bildinhalt im Bildschirmspeicher verschoben, sondern die Grenzen für das Bild innerhalb des Speichers werden neu festgelegt. Dadurch muß für die Videologik lediglich die Start- und Endadresse des Bildes im Speicher neu angegeben werden, während im anderen Fall aufwendige Speicherverschiebungen notwendig wären.

Das weiche Scrolling ist folglich wesentlich schneller, als das konventionelle Scrolling einer anderen Grafikkarte.

Wenn Sie GW-Basic ab der Version 2.1 besitzen, können Sie mit dieser Programmiersprache eine EGA-Karte ansprechen.

Tabelle 3 zeigt die neuen Screen-Befehle, über die sich in Basic die neuen Grafikmodi aufrufen lassen. Die ersten zwei Screen-Befehle (Screen 7 und Screen 8) funktionieren auch, wenn nur ein normaler Farbmonitor an Ihrem PC angeschlossen ist.

Im Textmodus, der über den Befehl »screen 0« aufgerufen wird, beträgt die Auflösung automatisch 640x350 Bildpunkte, wenn am PC ein EGA-Monitor angeschlossen ist.

Mit einem einfachen Poke-Befehl läßt sich über die Speicherzelle 00410hex

Der Aufbau des Farbwertes für ein Paletten- oder das Rahmenfarbregister	
Bit	Funktion
0	1. Blau
1	1. Grün
2	1. Rot
3	2. Blau
4	2. Grün
5	2. Rot
6	Unbenutzt
7	Unbenutzt

Tabelle 7. 64 verschiedene Farben lassen sich mit den 2x3 Farbbits des Farbwertes definieren

feststellen, welche Art von Monitor gegenwärtig aktiv ist:

00410 = 2D: Farbmonitor aktiv

00410 = 3D: Monochrom-Monitor aktiv

Die Speicherzelle 00488hex gibt Auskunft, ob es sich bei dem Farbmonitor um einen CGA-Monitor (handelsüblicher Farbmonitor), einen EGA-Monitor oder einen Monochrom-Monitor handelt:

00488 = F8: CGA-Monitor

00488 = F9: EGA-Monitor

00488 = FB: Monochrom-Monitor

Über die Speicherzelle 004A8hex läßt sich schließlich überprüfen, ob im PC überhaupt eine EGA-Karte vorhanden ist:

004A8 = 00: Keine EGA-Karte vorhanden

004A8 > 00: EGA-Karte vorhanden

In Maschinensprache und Assemblercode wird die EGA-Karte wie der Monochromadapter und die CGA-Farbgrafikkarte über den Software-Interrupt 10 angesprochen.

Tabelle 4 zeigt die Funktionen dieses Interrupts in einer Übersicht. Funktion 11 ist für sich allein bereits so komplex, daß wir aus Platzgründen auf eine detaillierte Darstellung verzichten haben.

Tabelle 5 und 6 zeigen, wie sich das Farbattribut im Text- und Grafikmodus zusammensetzt.

Tabelle 7 gibt Auskunft über den Aufbau des Farbwertes, der in die 16 Paletten-Register und das Rahmenfarbregister geladen wird und dadurch die 16 Bildschirmfarben sowie die Rahmenfarbe bestimmt.

Zum Schluß noch ein Hinweis für die Assembler-Programmierer. Achten Sie darauf, daß BP-Register vor dem Aufruf einer EGA-Funktion zu retten, weil die erweiterten und neuen EGA-Funktionen des Interrupt 10 dieses Register im Gegensatz zu den alten Funktionen 00 bis 0F oft verwenden und dementsprechend dessen Inhalt verändern. (ma)

Hallo PC, wie spät ist es?

Die Uhr in Ihrem PC steigert nicht nur den Komfort bei der täglichen Arbeit. Sie schützt auch Ihren Monitor vor Schäden, – vorausgesetzt man setzt sie richtig ein.

Schon seit Stunden sitzen Sie vor Ihrem Personalcomputer und haben Raum und Zeit vergessen. Damit letzteres nicht zu häufig passiert und Sie nicht eventuell wichtige Termine versäumen, brauchen Sie eine Uhr, die permanent die aktuelle Zeit auf den Bildschirm schreibt. Haben Sie vergessen Ihren Computer abzuschalten, wenn Sie den Arbeitsplatz verlassen, dann brennt der Elektrostrahl das stillstehende Bild in den Schirm ein. Würde der Bildschirm auf dunkel geschaltet, so wäre das Problem gelöst.

»clock« ist ein Dienstprogramm für genau solche Einsatzbereiche. Die Uhrzeit wird ständig auf den Bildschirm eingeblendet. Das Programm arbeitet dabei vollständig im Hintergrund, so daß es mit anderen Programmen keine Probleme gibt.

Und wenn Sie schon eine Uhr in Ihr System integrieren, dann soll diese auch gleich andere sinnvolle Aufgaben übernehmen – zum Beispiel den Monitor schützen. So ist in »clock« eine Routine eingebunden, die den Monitor ausschaltet, wenn Sie Ihren PC über einen bestimmten Zeitraum hinweg nicht benutzen. Drei Minuten nach dem letzten Tastendruck wird die Bildausgabe angehalten. Damit verhindern Sie, daß sich irgendwelche Zeichen in den Schirm einbrennen können.

Die Uhr liefert jede volle Stunde ein Stundensignal (Dreitonfolge). Ferner können Sie eine Alarmzeit setzen, an der Ihr Computer Sie durch eine dreifache Dreitonfolge an wichtige Termine erinnert.

»Clock« arbeitet menügesteuert (Bild). In dem Menü wählen Sie den Platz auf dem Bildschirm, ob und wo die Uhr eingeblendet und ob die Monitorroutine und die Alarmzeit aktiv sein soll.

Die Arbeit mit solchen Hilfsutilites ist allerdings bei manchen Anwendungsprogrammen nicht ganz ungefährlich. So arbeitet beispielsweise »Autocad« nicht mit dem Monitorschutzprogramm zusammen. Beim Aufruf des Hilfsprogramms können Sie deshalb menügesteuert einstellen, ob die Routine aktiv sein soll.

Vor dem ersten Start jeder Software müssen Sie sich deshalb immer überzeugen, ob sich die beiden Programme – die Utility »Clock« und das Programm, das Sie verwenden wollen – miteinander vertragen. Der Grund für diese Probleme liegt in der Speicherbelegung. Es ist leicht zu verstehen, daß ein Programm nicht funktionieren kann, wenn die Hilfsroutine den gleichen Speicherbereich beansprucht.

Listing 1 und Listing 2 ist für die Arbeit mit dem Microsoft Assembler (MASM) gedacht. Geben Sie beide Programme mit einem Editor als reine ASCII-Datei ein. Die Namen lauten »clock.asm« für die Routine aus Listing 1 und »clkframe.asm« für Listing 2. Übersetzen Sie nun mit dem Assembler beide Dateien:

```
masm clock
und
masm clkframe
```

Die anderen Fragen des Assemblers können Sie mit [Enter] übergehen.

Sind die Routinen fehlerfrei übersetzt, so müssen Sie mit `link clock+clkframe,clock`

das Programm »clock.exe« erzeugen. Diese Datei ist aber – trotz der Erweiterung »exe« (normalerweise werden damit nur lauffähige Programme bezeichnet) – noch nicht lauffähig. Mit

```
exe2bin clock,clock.exe
```

bekommen Sie die gewünschte Routine.

Wer keinen Assembler besitzt, der kann mit »debug« von der Systemdiskette Ihres Computers die hexadezimalen Zahlen aus Listing 3 eingeben. Diese Eingabe gestaltet sich etwas mühsam – ist aber immer noch besser als auf »clock« zu verzichten.

Starten Sie also »debug« und verändern Sie mit

```
r cx
```

das Register »cx«. Sie müssen mit

```
09d9
```

die Länge der Datei festlegen. Dann geben Sie mit

```
n clock.com
```

den Namen der Datei vor. Mit dem Kommando »e« geben Sie nun das Programm zeilenweise ein. Die erste Zeile wird mit

```
e 0100 <Werte der ersten Listingzeile>
```

eingetippt. Zwischen zwei Zahlen muß immer genau ein Leerzeichen stehen. Die zweite Zeile geben Sie mit

```
e 0110 <Werte der zweiten Listingzeile ein>
```

ein. Die vierstellige Ziffer in dem Listing bezeichnet immer die Zeilennummer (nach »e« angeben) und die 16 hexadezimalen Ziffern dahinter die Daten des Programms. Speichern Sie bei der Eingabe den Code zur Sicherheit öfters mit

```
w
```

auf Diskette.

Haben Sie sämtliche Werte eingegeben, so benutzen Sie wieder

```
w
```

um das gesamte Programm zu sichern.

Mit

```
q
```

wird der Debugger beendet und »clock.com« steht auf Ihrer Diskette. Falls Sie keinen Fehler bei der Eingabe gemacht haben, so läuft das Programm jetzt korrekt.

Zur Fehlersuche gibt es zwei verschiedene Debugbefehle.

Starten Sie als erstes den Debugger mit

```
debug clock.com
```

Bei diesem Aufruf wird die zu untersuchende Datei automatisch mit geladen.

```
d 0100
```

bringt einen bestimmten Teil der Datei als hexadezimale Zahlen auf den Bildschirm.

```
d
```

zeigt den nächsten Block an. Finden Sie einen Fehler, so können Sie die Zeile mit

```
e <Zeilennummer> <Werte der Zeile>
```

korrigieren.

Wer sich ein wenig mit dem Debugger auskennt, der kann sich mit

```
e <Speicherstelle> <Wert der Stelle>
```

einige Arbeit ersparen. Allerdings müssen Sie dazu die Speicherstelle berechnen.

Das ist aber nicht schwer. Die ersten drei Ziffern der Adresse entsprechen genau den ersten drei Werten der Zeilennummer. Die Zelle ganz links trägt an der vierten Stelle eine 0, die nächste Stelle eine 1, die dritte eine 2 und so weiter. Nach 9 geht es allerdings nicht mit der 10 weiter, sondern mit dem A. Auch die Adreßdaten werden hexadezimal angegeben. Nach diesem Schema trägt die Speicherstelle ganz rechts in einer Zeile immer an der vierten Stelle der Adresse ein F (für 15, die 16. Zahl hinter 0).

Die Arbeit mit dem Debugger ist sicher nicht besonders komfortabel, aber vielleicht bekommen Sie Lust auf Maschinensprache und legen sich einen Assembler zu. Auch

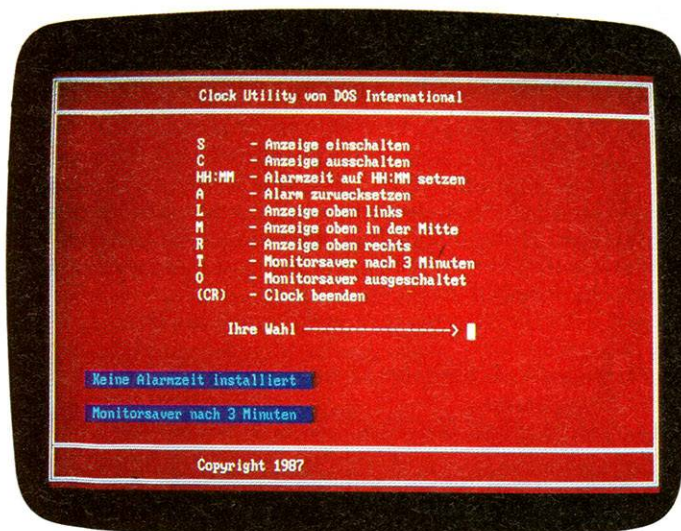


Bild 1. Mit »clock« wissen Sie immer wie spät es ist

der Assemblerkurs in diesem Heft soll Ihnen Mut und Lust auf die Maschinensprache des PC machen.

Der ebenfalls in diesem Heft abgedruckte Assembler ist leider für die Eingabe von »Clock« nicht geeignet. Der Microsoft Assembler arbeitet so komfortabel, daß er sich mit einem einfachen Programm nachbilden ließe. Und der Quellcode von »Clock« benutzt eben alle Annehmlichkeiten, die der Microsoft-Assembler zu bieten hat.

Noch ein Wort zu der Arbeit mit »Clock«. Da die Menüsteuerung beim Aufruf über eine Batch-Datei umständlich ist, müssen Sie sich in diesem Fall eine Textdatei (nur ASCII-Werte) mit den gewünschten Optionen schreiben und beim Aufruf mit übergeben. Der Aufruf lautet dann

clock < clock.set

sofern Ihre Textdatei – wie in Listing 4 – »clock.set« heißt. Wichtig ist, in der Datei am Ende ein [Enter] einzugeben (daß hier ein ASCII-Wert 13 steht), da sonst das Menü von »clock« auf dem Bildschirm stehen bleibt.

(J. Bauer/hg)

```

0: TITLE CLOCK.ASM
1: PAGE 66,132
2:
3:
4: .....
5:
6:                CLOCK.COM
7:                MASM CLKFRAME.ASM
8:                MASM CLOCK.ASM
9:                LINK CLOCK+CLKFRAME,CLOCK
10:               EXE2BIN CLOCK CLOCK.COM
11: .....
12:
13: CSEG          SEGMENT PUBLIC 'CODE'
14:                ORG 100H
15:
16: PUBLIC        MODULE2,GET,CL_COM,AS1
17: EXTRN         POS_CUR:NEAR,DLOOP:NEAR,EXIT:NEAR,MODULE1:NEAR
18: PAGE_NR
19:
20: VECTOR        EQU 1CH*4      ;Vector table offset of INT 1CH
21: ATTRIB         EQU 011BH
22: CRT_MODE       EQU 49H        ;Offset of CRT-Mode
23: ASI_SEGH       EQU 40H        ;ASI segment
24: VECTOR_SEGH    EQU 00H        ;Vector table segment
25:
26: CURSOR_MODE     EQU 60H
27: ADDR_6845       EQU 63H
28: CRT_MODE_SET    EQU 65H
29: TICK_30         EQU 1C8H
30:
31: ASSUME         CS:CSEG,DS:CSEG,ES:CSEG,SS:NOTHING
32:
33: MODULE2
34: JMP TRANSIENT
35:
36: BUSY            DB 1
37: TICKS           DB 0
38: SCP             DB 0
39: D_C_POS         DW 0
40: ALRM_TIME       DW 0ABCDH
41: ALRM_FLAG       DW 0ABCDH
42: ALRM_DISP       DW 7,0
43: ALRM_DISP       DW 2020H
44: TUBEHOUR        DW 2020H
45: TUBEMIN         DW ' '
46: TUBESEC         DW 2020H
47: TUBESEC         DW ' '
48: TUBESEC         DW 2020H
49: TUBESEC         DW 2020H
50:
51: RESIDENT:      ;Start of interrupt service routine
52:                CLI                ;Disable interrupts
53:                PUSH AX
54:                PUSH DS
55:                PUSHF
56:                PUSH CS
57:                POP DS              ;Get correct data segment
58:                PUSH BX
59:                PUSH CX
60:                PUSH DX
61:                PUSH SI
62:                PUSH DI
63:                PUSH BP
64:                PUSH ES
65:                MOV AX,ASI_SEGH
66:                MOV ES,AX
67:                CMP BYTE PTR[TICKS],12H
68:                JZ ACTIVATE          ;Jump every second
69:                INC BYTE PTR[TICKS]  ;Increment 1/18 sec
70:                ;counter
71:                JMP RECOVER_REGS     ;Jump to end of interrupt
72:                ;service
73:
74: ACTIVATE:
75:                MOV BYTE PTR[TICKS],00 ;Reset 1/18 sec
76:                ;counter
77:                CMP BYTE PTR[BUSY],0   ;Check busy flag
78:                ;(e.g. still alarm)

```

```

79:                JNZ GET_TIME
80: CHK_AL:        JMP CHECK_ALARM
81:
82: GET_TIME:
83:                MOV AH,0FH
84:                INT 10H
85:                CMP AL,3           ;Check video mode is not graphics
86:                JG CHK_AL
87:                MOV AX,WORD PTR ES:[006EH] ;Address of
88:                                     ;'DOS' seconds
89:                MOV CL,4
90:                PUSH CX
91:                SHL AX,CL
92:                MOV BX,WORD PTR ES:[006CH] ;Address of 'DOS'
93:                                     ;minutes and hours
94:                MOV DX,BX          ;Convert time in 'DOS'
95:                                     ;format to ASCII
96:                MOV CL,12          ;formatted time, for being
97:                                     ;displayed
98:                SHR DX,CL
99:                ADD DX,AX
100:               MOV AX,BX
101:               POP CX
102:               SHL AX,CL
103:               MOV BX,17478
104:               DIV BX
105:               MOV WORD PTR [TUBESEC],DX
106:               MOV BX,60
107:               XOR DX,DX
108:               DIV BX
109:               MOV WORD PTR [TUBEMIN],DX
110:               CALL CONVERT
111:               CMP AX,WORD PTR [TUBEHOUR]
112:               JNZ SIGNAL_HOUR
113:               MOV WORD PTR [TUBEHOUR],AX
114:               MOV AX,WORD PTR [TUBEMIN]
115:               CALL CONVERT
116:               MOV WORD PTR [TUBEMIN],AX
117:               MOV AX,WORD PTR [TUBESEC]
118:               MUL BX
119:               MOV BX,17478
120:               DIV BX
121:               CALL CONVERT
122:               MOV WORD PTR [TUBESEC],AX
123:               JMP DISPLAY_ON        ;Jump to display
124: SIGNAL_HOUR:
125:               PUSH AX
126:               CALL SPEAKER_ON
127:               POP AX
128:               JMP CONT
129:
130: CONVERT
131: PROC NEAR
132: AAM
133: ADD AX,3030H      ;Convert decimal to ASCII and
134: XCHG AH,AL        ;exchange high and low nibble
135: RET
136:
137: DISPLAY_ON:
138: MOV AH,3
139: XOR BH,BH
140: INT 10H           ;Fetch cursor position
141: MOV WORD PTR[SCP],DX ;Save current
142:                               ;cursor position
143: MOV DX,CS:[D_C_POS]
144: MOV AH,2
145: INT 10H           ;Position cursor
146: MOV SI,OFFSET ALRM_DISP+1 ;Address start
147:                               ;of display
148: MOV CX,10         ;Length of display
149: DILoop:          ;Load character
150: PUSH SI
151: PUSH CX
152: MOV AH,9
153: MOV CX,1
154: MOV BL,04FH       ;Attribute
155: INT 10H           ;Write character and attribute
156: INC DL            ;Increment cursor position
157: MOV AH,2
158: XOR BH,BH

```



```

158: INT 10H ;Position cursor
159: POP CX
160: POP SI
161: INC SI ;Address next character
162: LOOP D1LOOP
163: CMP BYTE PTR [ALRM_DISP],0EH
164: ;Check for alarm display
165: JNZ REC_CURS
166: MOV DX,CS:[D_C_POS]
167: XOR BH,BH
168: MOV AH,2
169: INT 10H
170: MOV AX,90EH ;Load symbol
171: MOV BL,OCFH ;Blinking attribute
172: MOV CX,1
173: INT 10H
174: REC_CURS: MOV DX,WORD PTR[SCP] ;Recover former
175: ;cursor position
176: MOV AH,2
177: XOR BH,BH
178: INT 10H
179: CHECK_ALARM:
180: CMP BYTE PTR[ALRM_FLAG],5 ;Check for
181: ;alarm finished
182: JB GIVE_ALARM ;Jump if not finished yet
183: MOV AX,WORD PTR[ALRM_TIME]
184: CMP AX,WORD PTR[TUBEHOUR] ;Check for
185: ;alarm hour reached
186: JNZ RECOVER_REGS ;Jump if not
187: MOV AX,WORD PTR[ALRM_TIME+2]
188: CMP AX,WORD PTR[TUBEHOUR] ;Check for
189: ;alarm minute reached
190: JNZ RECOVER_REGS ;Jump if not
191: MOV WORD PTR [ALRM_FLAG],0 ;Reset alarm
192: ;count flag
193:
194: GIVE_ALARM:
195: CALL BUZZER
196:
197: RECOVER_REGS
198: PROC NEAR
199: MOV AH,OFH
200: INT 10H
201: CMP AL,3
202: JG END_INT
203: CMP CS:DELAY,0
204: JZ END_INT
205: CALL CHECK_KEYPRESS
206: CMP CS:VIDEO_FLAG,0
207: JZ END_INT
208: DEC CS:COUNT
209: JNZ END_INT
210: MOV CS:COUNT,TICK_30
211: DEC CS:TIMEOUT
212: JNZ END_INT
213: MOV AL,CS:DELAY
214: MOV CS:TIMEOUT,AL
215: CALL VIDEO_OFF
216: END_INT:
217: POP ES
218: POP BP
219: POP DI
220: POP SI
221: POP DX
222: POP CX
223: POP BX
224: POP DS
225: POP AX
226: STI
227: IRET
228: RECOVER_REGS
229: ENDP
230: CHECK_KEYPRESS
231: PROC NEAR ;Check for key was pressed
232: MOV AX,WORD PTR CS:[SCP] ;Reset counters
233: ;if so
234: CMP AX,WORD PTR CS:[LAST_CUR]
235: MOV CS:[LAST_CUR],AX
236: JZ NO_KEY
237: KEYPRESSED:
238: MOV SI,ADDR_6845
239: MOV DX,WORD PTR ES:[SI]
240: ADD DX,4
241: MOV SI,CRT_MODE_SET
242: MOV AL,BYTE PTR ES:[SI]
243: OUT DX,AL
244: MOV CS:VIDEO_FLAG,1
245: MOV CS:COUNT,TICK_30
246: MOV AL,CS:DELAY
247: MOV CS:TIMEOUT,AL
248: NO_KEY_PRESSED:
249: RET
250: NO_KEY:
251: MOV AX,ES:[17H]
252: CMP AX,LAST_KB_FLAG
253: MOV CS:[LAST_KB_FLAG],AX
254: JNZ KEYPRESSED
255: MOV AX,ES:[1CH]
256: CMP AX,LAST_PTR
257: MOV CS:[LAST_PTR],AX
258: JNZ KEYPRESSED
259: JMP NO_KEY_PRESSED
260: CHECK_KEYPRESS
261: ENDP
262: VIDEO_OFF
263: PROC NEAR ;Switch beam current off
264: MOV SI,ADDR_6845
265: MOV DX,WORD PTR ES:[SI]
266: ADD DX,4
267: MOV SI,CRT_MODE_SET
268: MOV AL,BYTE PTR ES:[SI]
269: AND AL,OF7H
270: OUT DX,AL
271: MOV CS:VIDEO_FLAG,0
272: RET
273: VIDEO_OFF
274: ENDP
275: DELAY
276: DB 6 ;Data needed for tube saver

```

```

274: TIMEOUT
275: DB 2
276: COUNT
277: DW 456
278: LAST_CUR
279: DW 0
280: LAST_KB_FLAG
281: DW 0
282: LAST_PTR
283: DW 0
284: VIDEO_FLAG
285: DB 1
286:
287:
288: BUZZER
289: PROC NEAR
290: MOV WORD PTR[ALRM_TIME],0ABCDH
291: ;Clear alarm time buffer
292:
293: MOV AL,20H
294: MOV BYTE PTR [ALRM_DISP],AL ;Clear alarm
295: ;display
296:
297: SPEAKER_ON:
298: MOV AL,3H ;Set up first frequency
299: CALL SOUND
300: INC BYTE PTR[ALRM_FLAG+1] ;Set first
301: ;frequency is done
302: MOV AL,4H ;Set up second frequency
303: CALL SOUND
304: INC BYTE PTR[ALRM_FLAG+1] ;Set second
305: ;frequency is done
306: MOV AL,5H ;Set up third frequency
307: CALL SOUND
308: INC BYTE PTR[ALRM_FLAG] ;Increment alarm
309: ;count flag
310: MOV BYTE PTR[ALRM_FLAG+1],0 ;Set third
311: ;frequency is done
312: RET
313: SP_EXIT:
314:
315: SOUND:
316: PUSH CX
317: PUSH AX
318: MOV AL,0B6H
319: OUT 43H,AL
320: XOR AL,AL
321: OUT 42H,AL
322: POP AX
323: OUT 42H,AL
324: IN AL,61H
325: PUSH AX
326: OR AL,3
327: OUT 61H,AL
328: MOV CX,OFFFHH
329: MOV AL,2
330: LOOP SOUNDLOOP
331: MOV CX,OFFFHH
332: DEC AL
333: CMP AL,1
334: JZ SOUNDLOOP
335: POP AX
336: OUT 61H,AL
337: POP CX
338: RET
339: BUZZER
340: ENDP
341:
342:
343: *****
344: TRANSIENT:
345: PUSH CS
346: POP DS ;Get correct data segment
347: CALL MODULE1 ;Call the AS frame
348: MOV SI,OFFSET MSG1
349: ;Display main message
350: CALL DLOOP
351: MOV DL,60
352: CALL DLOOP
353: MOV DX,OF18H
354: CALL DLOOP
355: MOV DX,0414H
356: CALL DLOOP
357: MOV DX,0514H
358: CALL DLOOP
359: MOV DX,0614H
360: CALL DLOOP
361: MOV DX,0714H
362: CALL DLOOP
363: MOV DX,0814H
364: CALL DLOOP
365: MOV DX,0914H
366: CALL DLOOP
367: MOV DX,0A14H
368: CALL DLOOP
369: MOV DX,0B14H
370: CALL DLOOP
371: MOV DX,0C14H
372: CALL DLOOP
373: MOV DX,0D14H
374: CALL DLOOP
375: MOV AX,351CH
376: INT 21H ;Get vector of interrupt 1C
377: CMP BYTE PTR ES:[BX],OCFH ;Check for CLOCK
378: ;already being installed
379: JZ INSTALL ;Jump if not
380: CALL PARM_INP
381: CALL EXIT
382: INT 20H ;Finish transient program
383:
384: AS1:
385:
386: DB ' ' ;Copyright 1987 ',0
387: DB ' ' ;Data for displaying signon message
388:
389: INSTALL:
390: MOV BX,VECTOR_SEG
391: MOV ES,BX ;Set ES to segment of vector table
392: MOV BX,DS
393: CLI ;Disable interrupts
394: MOV SI,VECTOR+2
395: MOV WORD PTR ES:[SI],BX ;Install new
396: ;interrupt segment vector
397: DEC SI
398: DEC SI
399: MOV BX,OFFSET RESIDENT
400: MOV WORD PTR ES:[SI],BX ;Install new
401: ;interrupt offset vector

```



```

390:          STI                      ;Enable interrupts
391:          MOV AX,DS
392:          MOV ES,AX                ;Set ES to current data segment
393:          CALL PARM_INP            ;Get parameters
394:          CALL EXIT
395:          MOV DX,OFFSET TRANSIENT
396:          INC DX
397:          ;Terminate program
398:          ;and let stay resident
399:          INT 27H                  ;all coding,
;that is before the label 'TRANSIENT'
400: PARM_INP PROC NEAR
401:          MOV WORD PTR [DLOOP+1],011BH
402:          ;Set up cyan on blue
403:          CALL SHOW_ALARM_TIME
404:          CALL CL_COM
405:          CALL CL_DISP
406:          CALL GET
407:          XOR AX,AX
408:          XOR BX,BX                ;Clear registers
409:          XOR SI,SI
410:          XOR CH,CH
411:          MOV DI,80H              ;Address of parameter count
412:          MOV CL,BYTE PTR[DI]      ;Parameter count to CL
413:          MOV BP,OFFSET ALRM_HOUR-1
414:          ;Address of transient alarm time buffer
415:          CMP CL,0
416:          JNZ NEXT_PARM
417:          JMP GO_EXIT              ;Jump if no parameters
;in command line
418:
419: NEXT_PARM:
420:          INC DI                  ;Address next parameter
421:          MOV BL,BYTE PTR[DI]      ;Parameter to BL
422:          CMP BL,40H
423:          JL GO
424:          CMP BL,50H
425:          JGE LU_C1
426:          OR BL,60H
427:          JMP GO
428: LU_C1:
429:          CMP BL,60H
430:          JGE GO
431:          OR BL,70H
432:          CMP BL,'s'
433:          JNZ L_TOP
434:          MOV BYTE PTR ES:[BUSY],1
435:          MOV DX,1405H
436:          MOV SI,OFFSET MSG6
437:          CALL DLOOP
438:          JMP PARM_INP
439:
440: L_TOP:
441:          CMP BL,'1'
442:          JNZ M_TOP
443:          MOV ES:[D_C_POS],0       ;LEFT HAND DISPLAY
444:          MOV DX,1405H
445:          MOV SI,OFFSET MSG7
446:          CALL DLOOP
447:          JMP PARM_INP
448:
449: M_TOP:
450:          CMP BL,'m'
451:          JNZ R_TOP
452:          MOV ES:[D_C_POS],0023H
453:          MOV DX,1405H
454:          MOV SI,OFFSET MSG8
455:          CALL DLOOP
456:          JMP PARM_INP
457:
458: R_TOP:
459:          CMP BL,'r'
460:          JNZ TUBESAVE
461:          MOV ES:[D_C_POS],0046H   ;RIGHT HAND DISPLAY
462:          MOV DX,1405H
463:          MOV SI,OFFSET MSG9
464:          CALL DLOOP
465:          JMP PARM_INP
466:
467: TUBESAVE:
468:          CMP BL,'t'
469:          JNZ TUBES_OFF
470:          MOV ES:[DELAY],6
471:          MOV DX,1405H
472:          MOV SI,OFFSET MSG10
473:          CALL DLOOP
474:          JMP PARM_INP
475:
476: TUBES_OFF:
477:          CMP BL,'o'
478:          JNZ CANCEL
479:          MOV ES:[DELAY],0
480:          MOV DX,1405H
481:          MOV SI,OFFSET MSG3
482:          CALL DLOOP
483:          JMP PARM_INP
484:
485: CANCEL:
486:          CMP BL,'c'
487:          JNZ IS_IT_RESET
488:          MOV BYTE PTR ES:[BUSY],0 ;Reset busy flag
;in resident part
489:          MOV DX,1405H
490:          MOV SI,OFFSET MSG3
491:          CALL DLOOP
492:          JMP PARM_INP
493:
494: IS_IT_RESET:
495:          CMP BL,'a'
496:          JNZ BLANK
497:          MOV WORD PTR ES:[ALRM_TIME],0ABCDH
498:          ;Clear resident alarm buffer
499:          MOV BYTE PTR ES:[ALRM_DISP],' '
;Clear alarm display
500:          JMP PARM_INP
501:
502: GO_EXIT:
503:          JMP END_PARM
504:
505: BLANK:
506:          CMP BL,' '
507:          JNZ DPOINT_CHECK        ;Jump if not 'SPACE'
508:
509: GET_NEXT_PARM:
510:          LOOP L_N_P
511:          CMP BP,OFFSET ALRM_HOUR-1
512:          ;Check if alarm time was requested
513:          JNZ NXT
514:          JMP PARM_INP            ;Jump if not

```

```

506: L_N_P:
507: NXT:
508:          JMP NEXT_PARM
509:          CMP BP,OFFSET ALRM_MIN+1 ;Check for
;alarm time request was HH:MM
510:          JNZ ALRM_ERR            ;Jump if alarm time
;is incorrect
511:
512:          MOV AX,WORD PTR [ALRM_HOUR]
513:          XCHG AH,AL
514:          CMP AX,'23'
515:          JG ALRM_ERR            ;Jump if requested alarm hour
;is not OK
516:
517:          MOV AX,WORD PTR [ALRM_MIN]
518:          XCHG AH,AL
519:          CMP AX,'59'
520:          JNG STORE_ALARM        ;Jump if requested alarm
;minute is OK
521: ALRM_ERR:
522:          MOV SI,OFFSET ERR_MSG_1
523:          MOV DX,1405H
524:          CALL DLOOP            ;Display 'input error' message
525:          JMP PARM_INP
526:
527: STORE_ALARM:
528:          MOV BP,WORD PTR [ALRM_HOUR] ;Alarm
;hour to BP
529:          MOV WORD PTR ES:[ALRM_TIME],BP ;Alarm
;hour to resident buffer
530:
531:          MOV BP,WORD PTR [ALRM_MIN] ;Alarm
;minute to BP
532:          MOV WORD PTR ES:[ALRM_TIME+2],BP ;Alarm
;minute to resident buffer
533:          MOV BYTE PTR ES:[ALRM_DISP],0EH ;Set alarm display
534:
535:          MOV SI,OFFSET MSG4
536:          MOV DX,1405H
537:          CALL DLOOP            ;Display 'alarm stored' message
538:          JMP PARM_INP
539:
540:
541:
542: DPOINT_CHECK:
543:          CMP BL,'.'
544:          JZ DPOINT1
545:          CMP BL,' '
546:          JNZ CIPHER            ;Jump if not ' '
547:
548: DPOINT1:
549:          CMP BP,OFFSET ALRM_HOUR+1 ;Check for two digits were before ' '
550:          JNZ ALRM_ERR1
551:          ;Jump if not two digits for alarm hour
552:          CMP CX,3
553:          JB ALRM_ERR           ;Check for 3 parameters minimum
554:          INC BP                ;Jump if less parameters
555:          JMP GET_NEXT_PARM
556:
557: ALRM_ERR1:
558:          CMP BP,OFFSET ALRM_HOUR ;Check for one
;digit for hour
559:          JNZ ALRM_ERR          ;Jump if not
560:
561:          PUSH BX
562:          MOV BL,BYTE PTR[ALRM_HOUR]
563:          MOV BYTE PTR[ALRM_HOUR+1],BL
564:          MOV BYTE PTR[ALRM_HOUR],30H
565:          POP BX
566:          CMP CX,2
567:          JB ALRM_ERR
568:          ADD BP,2
569:          JMP GET_NEXT_PARM
570:
571: CIPHER:
572:          CMP BL,'0'
573:          JB ERR_MSG            ;Jump if parameter <0
574:          CMP BL,'9'
575:          JA ERR_MSG            ;Jump if parameter >9
576:          INC BP                ;Address next field of alarm buffer
577:          MOV BYTE PTR [BP],BL ;Put digit into
;alarm buffer
578:          JMP GET_NEXT_PARM
579:
580: ERR_MSG:
581:          MOV SI,OFFSET ERR_MSG_2
582:          MOV DX,1405H
583:          CALL DLOOP            ;Display 'invalid option' message
584:          JMP PARM_INP
585:
586: END_PARM:
587:          RET
588:
589: SHOW_ALARM_TIME PROC NEAR
590:          MOV AX,WORD PTR ES:[ALRM_TIME]
591:          CMP AX,0ABCDH
592:          JZ NOT_SET            ;Jump if no alarm time is set
593:          MOV WORD PTR [ALRM_HOUR],AX
594:          MOV WORD PTR ES:[ALRM_TIME+2],AX
595:          ;Put contents of resident buffer
596:          MOV WORD PTR [ALRM_MIN],AX ;into alarm
;buffer
597:          MOV DX,1205H
598:          MOV SI,OFFSET ALARM_TIME
599:          CALL DLOOP            ;Display 'alarm time'
;of resident buffer
600:          RET
601:
602: NOT_SET:
603:          MOV DX,1205H
604:          MOV SI,OFFSET MSG5
605:          CALL DLOOP            ;Display 'no alarm set' message
606:          RET
607:
608: GET
609:          PROC NEAR
610:          MOV BX,0147H
611:          MOV DX,0F37H
612:          CALL POS_CUR
613:          MOV BP,7FH
614:          MOV BYTE PTR[BP],8
615:          MOV DX,BP
616:          MOV AH,0AH
617:          INT 21H
618:          MOV WORD PTR[DLOOP+1],014FH
619:          MOV DX,1405H
620:          MOV SI,OFFSET CL_C_DAT
621:          CALL DLOOP
622:          MOV WORD PTR[DLOOP+1],011BH
623:          RET

```



```

622: GET          ENDP
623: ;
624: CL_COM        PROC NEAR          ;Clear command line
625:               MOV WORD PTR[DLOOP+1],014FH ;WHITE ON RED
626:               MOV DX,0F37H
627:               MOV SI,OFFSET CL_C_DAT+10
628:               CALL DLOOP
629:               MOV WORD PTR[DLOOP+1],011BH ;CYAN ON BLUE
630:               RET
631: CL_COM        ENDP
632: ;
633: CL_DISP       PROC NEAR          ;Clear message display
634:               MOV DX,0
635:               MOV BX,000FH
636:               CALL POS_CUR
637:               MOV CX,80
638:               MOV AX,0920H
639:               INT 10H
640:               RET
641: CL_DISP       ENDP
642: DUMHY         DB ' ',0
643: CL_C_DAT      DB ' ',0
644: MSG1          Clock Utility von DOS Inte
645:               DB ' ',0
646: MSG2          DB 'Ihre Wahl -----> ',0
647:               DB 'S - Anzeige einschalten ',0
648:               DB 'C - Anzeige ausschalten ',0
649:               DB 'HH:MM - Alarmzeit auf HH:MM setzen ',0
650:               DB 'A - Alarm zuruecksetzen ',0
651:               DB 'L - Anzeige oben links ',0
652:               DB 'M - Anzeige oben in der Mitte ',0
653:               DB 'R - Anzeige oben rechts ',0
654:               DB 'T - Monitorsaver nach 3 Minuten ',0
655:               DB 'O - Monitorsaver ausgeschaltet ',0
656:               DB '(CR) - Clock beenden ',0
657: MSG3          DB 'Clock-Anzeige ausgeschaltet ',0
658: MSG4          DB 'Neue Alarmzeit installiert ',0
659: MSG5          DB 'Keine Alarmzeit installiert ',0
660: MSG6          DB 'Clock-Anzeige eingeschaltet ',0
661: MSG7          DB 'Anzeige oben links ',0
662: MSG8          DB 'Anzeige oben in der Mitte ',0
663: MSG9          DB 'Anzeige oben rechts ',0
664: MSG10         DB 'Monitorsaver nach 3 Minuten ',0
665: MSG11         DB 'Monitorsaver ausgeschaltet ',0
666: ERR_MSG_1     DB 'Eingabe-Fehler ',0
667: ERR_MSG_2     DB 'Falsche Option, wiederholen ',0
668: CLR           DB ' ',0
669: ALARM_TIME    DB 'Alarmzeit ist '
670: ALRM_HOUR     DW 2020H
671:               DB ' ',0
672: ALRM_MIN      DW 2020H
673:               DB ' ',0
674: ;
675: MODULE2       ENDP
676: CSEG ENDS
677: END MODULE2

```

Listing 1. Das Assemblerlisting für »clock«

```

0: TITLE CLKFRAME.ASM
1: PAGE 66,132
2: ;
3: ;
4: ;This module 's got to become linked with CLOCK.
5: ;
6: ;
7: CSEG SEGMENT PUBLIC 'CODE'
8: ASSUME CS:CSEG,DS:CSEG,ES:CSEG,SS:CSEG
9: ;
10: PUBLIC POS_CUR,DLOOP,EXIT,MODULE1
11: EXTRN MODULE2:NEAR,GET:NEAR,CL_COM:NEAR,AS1:NEAR
12: ;
13: ATTR EQU 014FH ;WHITE ON RED INCREASED INTENSITY
14: ;ON DISPLAY PAGE 2
15: BLOCK EQU 010CH ;BLOCK CURSOR TOP LINE 1,
16: ;BOTTOM 12
17: HORI EQU 0CDH ;HORIZONTAL FRAMING CHARACTER
18: VERT EQU 0BAH ;VERTICAL FRAMING CHARACTER
19: TOPLEFT EQU 0C9H ;TOP LEFT FRAMING CORNER
20: BOTLEFT EQU 0C8H ;BOTTOM LEFT FRAMING CHARACTER
21: TOPRIGHT EQU 0BBH ;TOP RIGHT FRAMING CHARACTER
22: BOTRIGHT EQU 0BCH ;BOTTOM RIGHT FRAMING CHARACTER
23: LLINK EQU 0CCH ;LEFT FRAME LINKER
24: RLINK EQU 0B9H ;RIGHT FRAME LINKER
25: LINE EQU 78 ;LENGTH OF HORIZONTAL DISPLAY LINE
26: LIND EQU 80H ;ADDRESS OF LENGTH INDICATOR INPUT
27: ;
28: ;
29: MODULE1:
30: MOV BH,0 ;PAGE 1
31: MOV AH,3 ;GET CURSOR POSITION
32: INT 10H
33: MOV WORD PTR[CUR].DX
34: ;SAVE CURSOR POSITION FROM PAGE 1
35: MOV AX,0501H ;SELECT PAGE 2
36: INT 10H
37: MOV CX,BLOCK ;SET BLOCK STYLE CURSOR
38: MOV AH,1
39: INT 10H
40: MOV AX,0600H ;BLANK WINDOW
41: MOV BH,4FH ;ATTRIBUTE FOR BLANKED WINDOW
42: XOR CX,CX ;UPPER LEFT CORNER OF WINDOW IS 0,0
43: MOV DX,1850H ;LOWER RIGHT CORNER OF
44: ;WINDOW IS 24,80
45: INT 10H
46: MOV BX,ATTR ;SET UP PAGE AND ATTRIBUTE
47: MOV DX,0001H ;ROW 0 COLUMN 1
48: CALL POS_CUR
49: MOV CL,LIN ;SET UP CHARACTER COUNT
50: MOV AL,HORI ;HORIZONTAL FRAMING
51: INT 10H ;FRAME IN ROW 0
52: CALL SKIP2
53: INT 10H ;FRAME IN ROW 4
54: MOV DH,22
55: CALL POS_CUR

```

```

56: INT 10H ;FRAME IN ROW 22
57: CALL SKIP2
58: INT 10H ;FRAME IN ROW 24
59: XOR DX,DX ;ROW 0 COLUMN 0
60: CALL POS_CUR
61: MOV CL,1 ;NEW CHARACTER COUNT
62: MOV AL,TOPLEFT ;TOP LEFT CHARACTER
63: INT 10H
64: CALL NEXTLINE ;ROW 1 COLUMN 0
65: MOV AL,VERT ;VERTICAL CHARACTER
66: PUSH AX
67: INT 10H
68: CALL NEXTLINE ;ROW 2 COLUMN 0
69: MOV AL,LLINK ;LEFT LINKING CHARACTER
70: INT 10H
71: POP AX ;RESTORE VERTICAL CHARACTER
72: PUSH AX
73: CALL LOOP
74: CALL NEXTLINE ;ROW 22 COLUMN 0
75: MOV AL,LLINK
76: INT 10H
77: CALL NEXTLINE ;ROW 23 COLUMN 0
78: POP AX ;RESTORE VERTICAL CHARACTER
79: INT 10H
80: CALL NEXTLINE ;ROW 24 COLUMN 0
81: MOV AL,BOTLEFT ;BOTTOM LEFT CHARACTER
82: INT 10H
83: MOV DX,004FH ;TOP RIGHT CORNER
84: CALL POS_CUR
85: MOV AL,TOPRIGHT ;TOP RIGHT CHARACTER
86: INT 10H
87: CALL NEXTLINE ;ROW 2 COLUMN 80
88: MOV AL,VERT
89: PUSH AX
90: INT 10H
91: CALL NEXTLINE ;ROW 3 COLUMN 80
92: MOV AL,RLINK ;RIGHT LINKING CHARACTER
93: INT 10H
94: POP AX ;RESTORE VERTICAL CHARACTER
95: PUSH AX
96: CALL LOOP
97: CALL NEXTLINE ;ROW 23 COLUMN 0
98: MOV AL,RLINK ;RIGHT LINKING CHARACTER
99: INT 10H
100: CALL NEXTLINE ;ROW 24 COLUMN 80
101: POP AX ;RESTORE VERTICAL CHARACTER
102: INT 10H
103: CALL NEXTLINE ;ROW 25 COLUMN 80
104: MOV AL,BOTRIGHT ;BOTTOM RIGHT CHARACTER
105: INT 10H
106: MOV DX,1701H ;LINE 23 COLUMN 1
107: MOV SI,OFFSET AS1 ;ADDRESS MESSAGE BUFFER
108: CALL DLOOP ;DISPLAY 'Advanced Systems'
109: MOV DL,69 ;LINE 23 COLUMN 69
110: CALL DLOOP ;DISPLAY 'AS by JB'
111: MOV DX,0103H ;LAST CURSOR POSITION
112: CALL POS_CUR
113: MOV BX,ATTR
114: RET ;RETURN TO MODULE1
115: ;
116: EXIT: CALL CL_COM
117: CMP BYTE PTR CS:[LIND],0
118: JZ N_STEP
119: CALL GET
120: MOV BH,0 ;PAGE 1
121: MOV AX,0500H ;SELECT PAGE 1 AGAIN
122: INT 10H
123: MOV DX,WORD PTR[CUR]
124: ;RESTORE FORMER CURSOR POSITION
125: MOV AH,2
126: INT 10H ;SET CURSOR IN PAGE 1 AGAIN
127: RET ;RETURN TO MODULE1
128: ;
129: SKIP2: INC DH
130: NEXTLINE: INC DH
131: POS_CUR: PUSH AX
132: PUSH BX
133: PUSH CX
134: MOV AH,02H ;SET CURSOR POSITION
135: INT 10H
136: POP CX
137: POP BX
138: POP AX
139: MOV AH,09H ;WRITE CHARACTER AND ATTRIBUTE
140: RET
141: ;
142: LOOP: MOV CX,19 ;COUNT OF ROWS
143: L1: CALL NEXTLINE ;NEXT ROW
144: CALL POS_CUR
145: PUSH CX
146: MOV CL,1 ;CHARACTER COUNT
147: INT 10H
148: POP CX ;RESTORE LOOP COUNTER
149: LOOP L1
150: MOV CL,1 ;CHARACTER COUNT
151: RET
152: ;
153: DLOOP: MOV BX,ATTR
154: CALL POS_CUR
155: LODSB ;LOAD MESSAGE CHARACTER
156: OR AL,AL ;CHECK FOR TERMINATOR 00H
157: JZ ENDDL
158: MOV CX,1 ;CHARACTER COUNT
159: INT 10H
160: INC DL ;NEXT COLUMN
161: JMP DLOOP
162: ENDDL: RET
163: ;
164: ;
165: CUR: DW 0
166: ;
167: ;
168: CSEG ENDS
169: END MODULE1
170:

```

Listing 2. Der Quellcode für »clkframe«


```
0100 E9 45 02 01 00 00 00-00 CD AB CD AB 07 00 20
0110 20 20 3A 20 20 3A 20-20 20 3A 20 20 1E 3C 0E
0120 1F 53 51 52 56 57 58 59 60 61 62 63 64 65 66
0130 01 12 74 07 FE 06 04 01-E9 F8 00 C6 06 04 01
0140 80 3E 03 01 00 75 03 E9-C7 00 84 0F CD 10 3C 03
0150 7F 26 A1 6E 00 B1 04-51 D3 E0 26 06 1E 6C 00
0160 BB D3 B1 0C D3 EA 03 D0-BB C3 59 D3 BB 46 44
0170 F7 F3 89 16 17 01 BB 3C-00 33 D2 F7 F3 89 16 14
0180 01 E8 2C 00 3B 06 11 01-75 1F A3 11 01 A1 14 01
0190 E8 1D 00 A3 14 01 A1 17 01 F7 B3 46 57 F7 F3
01A0 E8 0D 00 A3 17 01 E8 10-90 58 51 01 58 E8 DA
01B0 04 0A 05 30 80 86 0E C3-B4 03 32 FF CD 10 89 16
01C0 05 01 2E BB 16 07 01 B4-02 CD 10 BE 10 01 B9 0A
01D0 00 8A 04 56 51 B4 09 B9-01 00 B3 4F CD 10 FE C2
01E0 B4 02 32 FF CD 10 59 5E-46 E2 B6 3E 0F 01 E9
01F0 75 15 2E BB 16 07 01 32-FF B4 02 CD 10 B8 0E 0E
0200 B3 CF B9 01 00 CD 10 8B-16 05 01 84 02 32 FF CD
0210 10 80 3E 05 01 05 72 1B-A1 09 01 3B 06 11 01 75
0220 12 A1 0B 01 3B 06 14 01-75 09 C7 06 00 01 00 00
0230 E8 0C 00 B4 0F CD 10 3C-03 7F 33 2E 80 3E E8 02
0240 04 28 E8 34 00 2E 80-3E F2 02 00 74 20 2E FF
0250 0E EA 02 75 19 2E C7 06-EA 02 C8 01 2E FE 0E E9
0260 02 75 0B 2E A0 E8 02 2E-A2 E9 02 E8 60 00 7D 50
0270 5F 5E 5A 59 58 9D 1F 5B-FB CF 2E A1 05 01 2E 3B
0280 0E EC 02 2E A3 EC 02 74-27 BE 63 00 26 88 14 03
0290 C2 04 BE 65 00 26 BA 04-E2 2E C6 06 F2 02 01 90
02A0 2E C7 06 EA 02 C8 01 2E-A0 E8 02 E2 A2 E9 02 C3
02B0 26 A1 17 00 3B 06 E2-2E A3 E2 02 80 C8 26 A1
02C0 1C 00 3B 06 F0 02 2E A3-F0 02 75 8D E8 1E BE 63
02D0 00 26 B8 14 83 C2 04 BE-65 00 26 BA 04 24 F7 E8
02E0 2E C6 06 F2 02 00 90 C3-06 02 C8 01 00 00 00 00
02F0 00 00 01 C7 06 09 01 CD-AB 80 20 A2 0F 01 80 03
0300 E8 1C 00 FE 0E 01 80-04 E8 13 00 FE 0E 01 81
0310 B0 05 E8 0A 00 FE 0E 00-01 C6 06 0E 01 00 C3 51
0320 50 B0 B6 64 43 C2 C0 E6-42 58 E6 42 64 61 50 C0
0330 C3 E6 61 B9 FF FF 00 02-E2 FE B9 FF 0F FE C8 3C
0340 01 74 F5 58 E6 61 59 C3-0E 1F E8 73 06 BE 4D 06
0350 E8 6F 07 B2 C8 E8 6A 07-BA 18 0F E8 64 07 BA 14
0360 04 E8 5E 07 BA 14 05 E8-58 07 BA 14 06 E8 52 07
0370 BA 14 07 E8 4C 07 BA 14-08 E8 46 07 BA 14 09 E8
0380 40 07 BA 14 0A E8 3A 07-BA 14 0B E8 3A 07 BA 14
0390 0C E8 2E 07 BA 14 0D E8-28 07 B8 1C 35 CD 21 26
03A0 80 3F CF 74 2D E8 51 00-E8 04 06 CD 20 20 20 20
03B0 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
03C0 43 6F 70 79 72 69 67 68-74 20 31 39 38 37 20 00
03D0 20 00 B8 00 00 8E C3 8C-DB FA BE 72 00 26 89 1C
03E0 4E 4E B8 1B 01 26 89 1C-FB 8C DB 8E C8 E8 09 00
03F0 E8 8C 06 BA 48 03 42 CD-27 C7 06 C3 0A 01 B1 E8
0400 A9 01 E8 FF 01 E8 12 02-E8 00 01 33 C0 33 D8 33
0410 F6 32 ED BF 80 00 BA 0D-BD A3 09 80 F9 00 75 03
0420 E9 02 00 47 8A 1D 80 FB-40 7C 13 80 FB 50 70 06
0430 80 CB 60 E8 09 80 FB-60 7D 03 C8 CB 70 80 FB
0440 73 75 11 26 C6 06 03 01-01 BA 05 14 BE 7E 08 E8
0450 70 06 E8 A5 80 FB 6C 75-12 26 C7 06 07 01 00 00
0460 BA 05 14 BE 8D 08 E8 59-06 E8 8E 80 FB 6D 75 13
0470 26 C7 06 07 01 23 0A BA-05 14 BE 8C 08 E8 42 06
0480 E9 76 FF 80 FB 72 75 13-26 C7 06 07 01 46 00 8A
0490 05 14 BE DB 08 E8 2A 06-E9 5E FF 80 FB 74 75 12
04A0 26 C6 06 E8 02 06 BA 05-14 BE FA 08 E8 13 06 E9
04B0 47 FF 80 FB 6F 75 12 26-C6 06 E8 02 00 BA 05 14
04C0 BE 19 09 E8 FC 05 E8 30 FF 80 FB 63 75 12 26 C6
04D0 06 03 01 00 BA 05 14 BE-21 08 E8 E5 05 19 FF
04E0 80 FB 61 75 13 26 C7 06-09 01 CD AB 26 C6 06 0F
04F0 01 20 E9 04 FF E9 B2 00-80 FB 20 75 58 E2 09 B1
0500 FD A3 09 75 06 E9 F1 FE-E3 18 FF 61 FD A8 09 75
0510 14 A1 A4 09 86 00 3D 33-32 7F 0A A1 A7 09 86 E0
0520 3D 39 35 7E 0C BE 38 09-8A 05 14 E8 94 05 E9 C8
0530 FE B8 2E A4 09 C6 B9 2E-09 01 B8 2E A7 09 26 B9
0540 2E 0B 01 26 C6 06 0F 01-0E BE 40 8A 05 14 E8
0550 70 05 E9 A4 FE 80 FB 3A-74 05 80 FB 2E 75 2E 81
0560 FD A5 09 75 08 83 F9 03-72 BB 45 E8 90 81 FD A4
0570 09 75 B2 53 BA 1E A4 09-8B 1E A5 09 C6 06 A4 09
0580 30 58 83 F9 02 72 9E 83-C3 02 E9 70 FF 80 FB 30
0590 72 0C 80 FB 39 77 07 45-88 5E 00 59 5F FF BE 57
05A0 09 BA 05 14 E8 18 05 E9-4F FE C3 26 A1 09 01 3D
05B0 CD AB 74 14 C7 06 06 09-3A 00 A3 A4 09 26 A1 05
05C0 01 A3 A7 09 BA 05 12 BE-95 09 E8 05 04 C3 BA 05
05D0 12 BE 5F 08 E8 E8 04 C3-BB 47 01 BA 37 0F E8 C0
05E0 04 BD 7F 00 C6 06 00 8B-88 05 B4 0A CD 21 C7 06
05F0 C3 0A 4F 01 BA 05 14 BE-2E 06 E8 C5 04 C7 06 C3
0600 0A 1B 01 C3 C7 C6 03 0A-4F 01 BA 37 0F E8 C6 FE
0610 E8 AF 04 C7 06 C3 0A 1B-01 C3 BA 00 00 BB 0F 00
0620 E8 7E 04 B9 50 00 B8 20-09 CD 10 C3 20 00 20 00
0630 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
0640 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
```

```
0650 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20 20
0660 6F 63 68 20 55 74 69 6C-69 74 79 20 76 6F 6E 20
0670 44 4F 53 20 49 6E 74 65-72 6E 61 74 69 6F 6E 61
0680 6C 20 00 20 00 49 68 72-65 20 57 61 68 6C 20 20
0690 2D 2D 2D 2D 2D 2D 2D-2D 2D 2D 2D 2D 2D 2D 2D 2D
06A0 2D 2D 3E 20 00 53 20-20 20 20 20 20 20 20 20 20
06B0 7A 65 69 67 65 20 69-6E 73 63 68 61 6C 74 65
06C0 6E 20 20 20 20 20 20-20 20 20 20 00 43 20 20 20
06D0 20 20 20 41 6E 7A 65-69 67 65 20 61 75 73 73
06E0 20 20 61 6C 74 65 6E 20 61 6C 74 65 69 74 20
06F0 00 48 48 3A 4D 4D 4D-20 20 20 20 41 6C 74 6D 7A
0700 65 69 74 20 61 75 66 20-48 48 3A 4D 4D 73 65
0710 74 7A 65 6E 20 00 41-20 20 20 20 20 20 20 20 20
0720 41 6C 61 72 6D 20 7A 75-72 75 65 63 68 73 65 74
0730 7A 65 6E 20 20 20 20-20 20 20 20 20 00 4C 20 20
0740 20 20 20 20 20 20 20-20 20 20 20 65 69 67 65 62
0750 65 6E 20 6C 69 6E 68 73-20 20 20 20 20 20 20 20
0760 20 20 00 4D 20 20 20-20 20 20 20 20 20 41 6E 7A 65
0770 69 67 65 20 6F 62 65 6E-20 69 6E 20 64 65 72 20
0780 4D 69 74 74 65 20 20-20 52 20 20 20 20 20 20 20
0790 20 20 41 6E 7A 65 69 67-65 20 6F 62 65 6E 20 72
07A0 65 63 68 74 73 20 20-20 20 20 20 20 20 20 00 54
07B0 20 20 20 20 20 20 20-20 4D 6F 6E 69 74 6F 72 73
07C0 61 76 65 72 20 6E 61 63-68 20 33 20 4D 69 6E 75
07D0 74 65 6E 20 00 4F 20-20 20 20 20 20 20 20 4D 6F
07E0 6E 69 74 6F 72 73 61 76-65 72 20 61 75 73 67 65
07F0 73 63 68 61 6C 74 65 74-20 20 20 00 28 43 52 29 20
0800 20 20 20 20 43 6C 6F 63-68 20 62 65 65 6E 64 65
0810 6E 20 20 20 20 20 20-20 20 20 20 20 20 20 20 20
0820 02 20 43 6C 6F 63 68 20-41 6E 7A 65 69 67 65 20
0830 61 75 73 67 65 73 63 68-61 6C 74 65 74 20 20 00
0840 20 4E 65 75 65 20 41 6E-61 72 6D 74 65 69 74 20
0850 69 6E 73 74 61 6C 6C 69-65 72 74 20 20 20 20 00
0860 65 69 6E 69 6E 65 20 41 6C-61 72 6D 74 65 69 74 20
0870 69 6E 73 74 61 6C 6C 69-65 72 74 20 20 00 43
0880 6C 6F 63 68 20 41 6E 7A-65 69 67 65 20 65 69 6E
0890 67 65 73 63 68 61 6C 74-65 74 20 20 20 20 41 6E
08A0 7A 65 69 67 65 20 6F 62 65 6E-20 6C 69 6E 68 73
08B0 20 20 20 20 20 20 20-20 20 20 20 00 41 6E 7A
08C0 65 69 67 65 20 6F 62 65 6E-20 69 6E 20 64 65 72
08D0 20 4D 69 74 74 65 20 20-20 20 20 20 20 41 6E 7A
08E0 69 67 65 20 6F 62 65 6E-20 72 65 63 68 74 72 20
08F0 20 20 20 20 20 20 20-20 20 20 20 00 4D 6F 6E 69 74
0900 6F 72 73 61 76 65 72 20-6E 61 63 68 20 33 20 4D
0910 69 6E 75 74 65 6E 20 20-20 20 20 20 4D 6F 6E 69 74 6F
0920 72 73 61 76 65 72 20 61-75 73 67 65 73 63 68 61
0930 6C 74 65 74 20 20 20 20-00 45 69 6E 67 61 62 65
0940 2D 46 65 68 6C 65 72 20-20 20 20 20 20 20 20 20
0950 20 20 20 20 20 20 20-20 46 61 6C 73 63 68 65 20
0960 4F 70 74 69 6F 6E 2C 20-77 65 64 65 72 6F 6F
0970 6C 65 6E 20 20 20 20-20 20 20 20 20 20 20 20 20
0980 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20 20
0990 20 20 20 20 20 20 20-20 41 6C 61 72 6D 74 65 69 74 20
09A0 69 73 74 20 20 20 3A 20-20 20 20 20 20 20 20 20
09B0 20 20 20 00 00 00 00-00 00 00 00 00 00 00 00 00
09C0 87 00 B4 03 CD 10 89 16-07 0A B8 01 05 CD 10 89
09D0 0C 01 B4 01 CD 10 B8 00-06 87 4F 33 8A 50 18
09E0 CD 10 B8 4F 01 BA 01 00-E8 B6 00 B1 4E BD CD CD
09F0 10 E8 A9 00 CD 10 B6 16-E8 A6 00 CD 10 E8 9D 00
0A00 CD 10 33 D2 E8 9A 00 B1-01 80 C9 CD 10 E8 8F 00
0A10 B0 BA 50 CD 10 E8 87 00-80 CD 10 58 50 B8 00
0A20 00 E8 7B 00 B0 CC CD 10-E8 74 00 58 CD 10 E8 6E
0A30 00 B0 C8 CD 10 BA 4F 00-E8 66 00 B0 BB CD 10 E8
0A40 50 00 B0 BA 50 CD 10 E8-55 00 B9 CD 10 58 50
0A50 E8 00 E8 49 00 B0 B9 CD 10 E8 42 00 58 50 B8
0A60 E8 3C 00 B0 CC CD 10 BA-01 17 BE AD 03 E8 52 00
0A70 B2 45 E8 4D 00 BA 03 01-E8 26 00 B8 4F 01 C3 E8
0A80 B2 FB 2E 80 00 00 74 03 E8 48 FB 87 00 B8
0A90 00 05 CD 10 B8 16 07 0A-B4 02 CD 10 C3 FE C6 FE
0AA0 C6 50 53 51 B4 02 CD 10-59 58 5B 84 09 C3 B9 13
0AB0 00 E8 E8 FF E8 EA FF 51-B1 01 CD 10 59 E2 F2 B1
0AC0 01 C3 E8 4F 01 E8 09 FF-AC 0A C0 74 09 89 01 00
0AD0 CD 10 FE C2 E8 EC C3 00-00
```

Listing 3.
Für die hexadezimale Eingabe brauchen Sie keinen Assembler
R
O
10:30
Listing 4.
Komfort beim Aufruf durch eine Batch-Datei mit »clock.lst«

Systemnahe Programmierung in Turbo Pascal

Schnelle Bildausgabe erreicht man in Turbo Pascal durch direktes Beschreiben des Bildschirmspeichers. Doch dazu muß man systemnah programmieren und die Unterschiede zwischen Inline- und External-Code kennen.

Obwohl Turbo Pascal einen der schnellsten und kompaktesten Codes aller Programmiersprachen für PCs generiert, kommt es doch zuweilen vor, daß ein Ausweichen auf direkte Systemzugriffe nötig ist. Besonders wenn eine Prozedur zeitkritisch ist, muß man auf die Maschinsprache ausweichen. Auch für Anweisungen, die in der Hochsprache nicht vorgesehen sind, brauchen Sie systemnahe Routinen.

Turbo Pascal besitzt zahlreiche Anweisungen um ohne größere Verrenkungen systemnahe Programme zu schreiben – sei es mit Hilfe des Turbo Pascal-eigenen Speicherzugriffs, der systemnahen Programmierung, dem getrennten Compilieren von Assemblerprozeduren und dem Ansprechen als »external Prozedur« oder dem Formulieren dieser Probleme als Inline-Code mit dem direkten Zugriff auf spezielle Routinen von Turbo Pascal.

Bei solchen systemnahen Programmen müssen Sie sich immer bewußt sein, daß sich eine Routine mit wohldefi-

nieren MS-DOS-Aufrufen nur schwer auf andere Computersysteme (zum Beispiel CPM-Geräte) übertragen läßt. Weiterhin können Programme, die beim direkten Speicherzugriff auf einen Commodore-PC keine Probleme machen, auf einem Olivetti oder IBM erbarmungslos abstürzen. Einzig sorgfältige Dokumentation und strikte Modularität helfen da, solche Probleme zumindest zu mindern.

Daß die Formulierung eines Problems in Inline-Code auch Platz spart, macht das Problem aus Listing 1 deutlich. Das Listing zeigt den Maschinencode für zwei Routinen, die den Cursor ein- oder ausschalten. Eingebunden in Pascal-Programme können Sie so effektive Ausgaberroutinen programmieren. Versuchen Sie einmal den Aufruf des BIOS-Interrupt 10hex mit den Anweisungen von Turbo Pascal zu programmieren (siehe auch Handbuch zu Turbo Pascal Seite 208 und Seite 214). Sie werden Maschinsprache schätzen lernen.

Für das wohl häufigste Problem, den Cursor nur an- oder auszuschalten, stellt diese Routine die effektivste Lösung dar. Will man jedoch einen eigenen Cursor programmieren, gestaltet sich das Problem schon aufwendiger. Insbesondere die zu setzenden Parameter müssen in der richti-

gen Reihenfolge an das Maschinenprogramm übergeben werden.

Da schon häufig falsche Aussagen über systemnahe Prozeduren veröffentlicht wurden und das Handbuch von Turbo Pascal sich über das Thema recht lakonisch ausschweigt, finden Sie hier die Parameterübergabe an External- oder Inline-Prozeduren näher erläutert.

Auf MS-DOS-Systemen organisiert Turbo Pascal Unterprogrammaufrufe mit Hilfe einer Stapelstruktur, dem sogenannten Stack. Wird ein Unterprogramm aufgerufen, so gelangt zunächst die Rücksprungadresse auf den Stack. Schließlich muß das Hauptprogramm wissen, wo es nach dem Ende des Unterprogramms weiter zu machen hat. Daneben werden alle Parameter, die das Unterprogramm bearbeiten soll, in der deklarierten Reihenfolge auf den Stack gelegt.

Nun könnte man diese Werte innerhalb eines Assemblerprogramms durch den Befehl »push« (hole vom Stapel) bearbeiten; allerdings nur sequentiell, daß heißt der Reihe nach, da »push« sich Wert für Wert »in den Stapel hineinfrißt«, und wenn Sie nicht aufpassen, am Ende auch noch die Rücksprungadresse löscht.

Will man beliebig auf die übergebenen Parameter zugreifen, so muß man sicherstellen, daß die Stapelstruktur erhalten bleibt. Dies ist prinzipiell ganz einfach, da nur der Stackpointer ins Basisseitenregister kopiert werden muß. Danach kann man indiziert über den Basepointer auf den Stack zugreifen.

Doch zuerst muß noch bekannt sein, ob der auf dem Stack abgelegte Wert ein Byte, Word, oder gar ein 256 Zeichen langer String ist. Wenn man ebenfalls nicht weiß, in welcher Reihenfolge die Prozedurparameter auf dem Stack liegen, wann Turbo Pascal Variableninhalte als Wert (»call by value«) oder als Adresse (»call by reference«) übergibt, dann hilft nur noch ausprobieren.

In der Regel führt das zu ganzen Serien von Systemabstürzen und Erfahrungen sind hier nur sehr mühsam und zeitaufwendig zu erwerben. Deshalb – und wenn Ihnen die Erklärung der Fakten zu kurz und damit zu unverständlich war – benutzen Sie einfach »stack.pas« aus Listing 2. Das Programm enthält einige Zeilen Inline-Code, die den Stack lesen und die letzten 20 dort abgelegten Werte mit den Indizes in hexadezimaler Schreibweise auf den Bildschirm ausgeben. Dieser Code muß aus Listing 3 einzeln abgetippt und mit dem Namen »stack.inl« auf Diskette abgelegt werden. Wenn man die Werte in der aufrufenden Zeile hexadezimal formuliert und Deklarationen von Turbo Pascal ein wenig durchprobiert, erhält man schnell eine Übersicht über die interne Arbeitsweise des Compilers.

Das Assemblerprogramm funktioniert nur als Inline-Code. Rufen Sie die Prozedur »external« auf, so stürzt das Programm ab. Der Grund ist, daß Turbo Pascal beim Einbinden eines »external« Programms nicht weiß, wieviel Parameter über den Stack übergeben werden. Den Stack so zu ordnen, daß als letzter Wert die Rücksprungadresse zum Hauptprogramm liegt, ist Aufgabe des Programmierers.

Werden zum Beispiel vier Byte an die Prozedur übergeben, darf diese nicht einfach mit »ret« verlassen, sondern es muß auch der Stapel um die vier 4 Byte erniedrigt werden (»ret 4«). Binden Sie dagegen das Maschinenprogramm »inline« ein, so übersetzt der Compiler das ab-

Rahmen für Assembler Einbindung:

```
procedure asm(l,h:integer);(external 'asm.bin')
```

»inline«	»external«
Turbos »begin«	push bp mov bp,sp
;Variable h holen mov ch,(bp)+4	;Variable h holen mov ch,(bp)+4
;Variable l holen mov cl,(bp)+6	;Variable l holen mov cl,(bp)+6
;Verarbeitungsteil ;z.B. die eingangs vorgestellte Cursor ;Umschaltung.	----- ----- ----- mov sp,bp pop bp ret 4
Turbos »end«	

Die Unterschiede zwischen Inline- und External-Code

schließende »end« automatisch mit dem passenden Return-Befehl.

Ansonsten ist das Prinzip von Inline- oder External-Prozeduren gleich. Alle Übergabeprobleme lassen sich durch »stack.inl« erforschen. Wenn der Rahmen entsprechend gestaltet wird, steht hocheffizienten Turbo Pascal-Programmen nichts mehr im Wege.

Wie man sich das in der Praxis zunutze macht, zeigt das Programm aus Listing 4, das eine schnelle Zeichenausgabe auf den Bildschirm erlaubt, da es direkt in den Bildschirmspeicher schreibt. Weiterhin wird anhand einer Stoppuhr gezeigt, wie man die systeminterne Uhr zur Laufzeitbestimmung von Pascal-Prozeduren einsetzt.

Turbo Pascal benutzt für die Zeichenausgabe den Interrupt 10hex, der über das BIOS läuft. Zwar ist so sichergestellt, daß die Zeichen wirklich in den Bildschirmspeicher geschrieben werden, die internen Überprüfungen verlangsamen die Zeichenausgabe jedoch erheblich. Das Programm »quickscreen« zeigt, wie die Bildschirmausgabe durch immer systemnähere Programmierung in vier Stufen beschleunigt wird.

Der Nachteil der selbstgeschriebenen Ausgabeprozeduren ist jedoch, daß der Bildschirmspeicher wirklich an der Adresse B000hex beginnt, und daß der Monitor im monochromen Modus sein muß, da ansonsten entweder gar nichts oder nur wirre Zeichen auf dem Bildschirm ausgegeben werden. Weiterhin kann es gelegentlich, je nach Grafikkarte, Probleme durch »Schneefall« auf dem Bildschirm geben.

Die oben beschriebenen Prozeduren laufen auf dem Commodore-PC anstandslos. Wenn auf anderen PCs die Bildschirmausgabe gestört wird, muß die in »bsout« ausgeklammerte Prozedur »retrace« eingebunden werden. Diese wartet bei der Zeichenausgabe bis das Bildschirmsynchronsignal einen Nulldurchlauf macht und daher die höhere und hier störende Priorität des Prozessors vor dem Bildschirmdirektzugriff keine Rolle spielt.

(Wilfried Radigk/hg)


```

0: procedure cursorOff;
1: begin
2:   inline
3:   ($50/          ( push ax          )
4:   $51/          ( push cx          )
5:
6:   $B4/$01/ ( mov  ah,01  Cursor-Größe definieren )
7:   $B5/$FF/ ( mov  ch,FF  Bits 0..4 CursorStartz. )
8:   $B1/$0C/ ( mov  cl,0C  Bits 0..4 CursorEndzeile )
9:   $CD/$10/ ( int   10h   Bildschirmfunktionen )
10:
11:  $59/          ( pop  cx          )
12:  $5B);         ( pop  ax          )
13: end;{-----}
14:
15: procedure cursorOn;
16: begin
17:   inline
18:   ($50/$51/$B4/$01/$B5/$0C/$B1/$0D/$CD/$10/$59/$5B);
19: end;{-----}

```

Listing 1. Platz sparen mit Inline-Code

```

0: program stack;
1: { Darstellung der Parameterübergabe an Prozeduren
2:   durch Turbo-Pascal }
3:
4: {----- Includefile STACK.INL -----}
5: clrscr;
6: writeln('Stackbelegung in HEX');
7: writeln('BP+25,' Wert');
8: inline($EB/$4D/$90/$50/$53/$51/$52/$56/$57/$8A/$D0/$B4
9: /$02/$CD/$21/$5F/$5E/$5A/$59/$5B/$58/$C3/$50/$53/$51
10: /$52/$1E/$56/$57/$33/$D2/$B1/$04/$33/$DB/$8A/$D8/$80
11: /$E3/$0F/$80/$FB/$09/$7F/$07/$80/$C3/$30/$53/$EB/$05
12: /$90/$80/$C3/$37/$53/$D3/$E8/$42/$83/$FA/$04/$75/$E3
13: /$58/$E8/$BF/$FF/$4A/$75/$F9/$5F/$5E/$1F/$5A/$59/$5B
14: /$58/$C3/$1E/$55/$BE/$18/$00/$8B/$C6/$E8/$BD/$FF/$B0
15: /$20/$E8/$A5/$FF/$8B/$02/$E8/$B3/$FF/$B0/$0A/$E8/$9B
16: /$FF/$B0/$0D/$E8/$96/$FF/$4E/$4E/$83/$FE/$00/$7F/$E0
17: /$5D/$1F);
18: ----- Ende des Includefiles -----}
19:
20: type longstring=string[20];
21: var l:longstring;
22: const cl='ABCDEFGH';
23:
24: procedure probe(lstring:longstring;a,b:integer);
25: begin
26:   ($I stack.inl)
27: end;
28:
29: procedure probel(var lstring:longstring;a,b:integer);
30: begin
31:   ($I stack.inl)
32: end;
33:
34: begin (main)
35:   probe(cl,$10,$20);
36:   writeln: readln;
37:   l:='1234567';
38:   probel(l,$10,$20);
39: end. (main)

```

Listing 2. »stack.pas« liest den Stack

```

0: writeln('Stackbelegung in HEX');
1: writeln('BP+25,' Wert');
2: inline
3: ($EB/$4D/$90/$50/$53/$51/$52/$56/$57/$8A/$D0/$B4
4: /$02/$CD/$21/$5F/$5E/$5A/$59/$5B/$58/$C3/$50/$53/$51
5: /$52/$1E/$56/$57/$33/$D2/$B1/$04/$33/$DB/$8A/$D8/$80
6: /$E3/$0F/$80/$FB/$09/$7F/$07/$80/$C3/$30/$53/$EB/$05
7: /$90/$80/$C3/$37/$53/$D3/$E8/$42/$83/$FA/$04/$75/$E3
8: /$58/$E8/$BF/$FF/$4A/$75/$F9/$5F/$5E/$1F/$5A/$59/$5B
9: /$58/$C3/$1E/$55/$BE/$18/$00/$8B/$C6/$E8/$BD/$FF/$B0
10: /$20/$E8/$A5/$FF/$8B/$02/$E8/$B3/$FF/$B0/$0A/$E8/$9B
11: /$FF/$B0/$0D/$E8/$96/$FF/$4E/$4E/$83/$FE/$00/$7F/$E0
12: /$5D/$1F);

```

Listing 3. Der Inline-Code für »stack.pas« trägt den Namen »stack.inl«

```

0: program quickscreen;
1: { Systemnahe Programmierung in Turbo-Pascal }
2: { Schnelle Bildschirmausgabe durch direktes
3:   Beschreiben des Bildschirmspeichers }
4:
5: type longstring=string[80];
6: var  xw,yw:byte;
7:      min,std:integer;
8:      zeit:array[1..4] of longstring;
9:
10: procedure Ausgabe1(lstring:longstring;y,x:integer);
11: begin
12:   gotoxy(x,y);

```

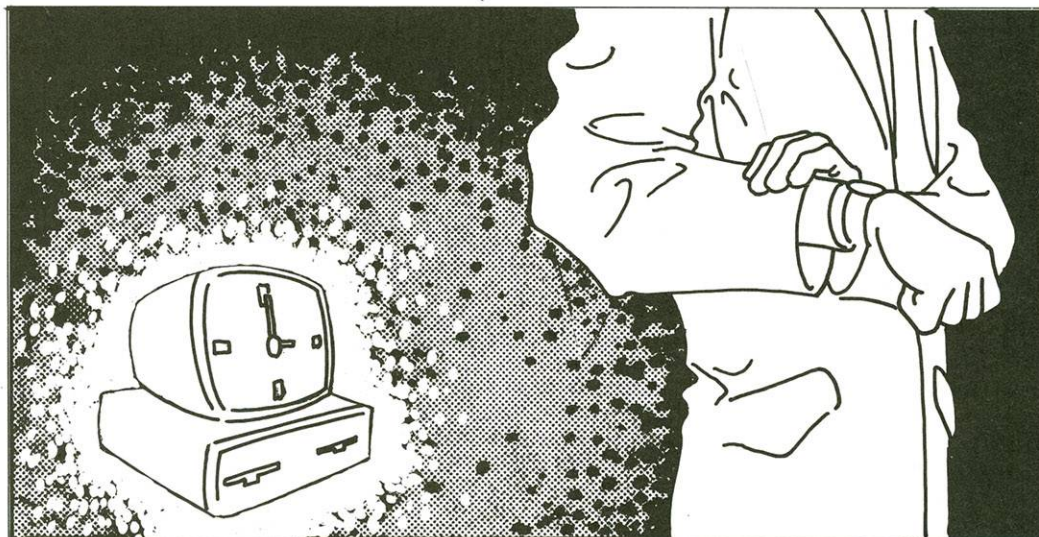
```

13:   write(lstring);
14: end;{-----}
15:
16: procedure Ausgabe2(lstring:longstring;y,x:integer);
17: var neu:byte;
18: begin
19:   for neu:=1 to length(lstring) do begin
20:     mem[$B000:(y-1)*160+(x+neu-2)*2]
21:       :=ord(copy(lstring,neu,1));
22:   end;
23: end;{-----}
24:
25: procedure Ausgabe3(lstring:longstring;y,x:integer);
26: var neu:byte;
27: begin
28:   for neu:=1 to length(lstring) do begin
29:     mem[$B000:((y-1) shl 5) * 5 + (x+neu-2) shl 1]
30:       :=ord(copy(lstring,neu,1));
31:   end;
32: end;{-----}
33:
34: procedure Ausgabe4(lstring:longstring;y,x:integer);
35: begin
36:   inline($55/$8B/$5E/$04/$4B/$D1/$E3/$8B/$46/$06/$48/$B1
37: /$05/$D3/$E0/$52/$BA/$05/$00/$F7/$E2/$5A/$03/$C3/$BE
38: /$00/$00/$B9/$00/$00/$8A/$4E/$08/$8B/$F8/$B8/$00/$B0
39: /$8E/$C0/$B4/$0F/$8A/$42/$09/$AB/$46/$3B/$F1/$75/$F5
40: /$5D);
41: end;{-----}
42:
43: procedure start;
44: begin
45:   min:=memw[$0040:$006C];
46:   std:=memw[$0040:$006E];
47:   { Dies sind die Speicherplätze der Uhr im Commodore-
48:     PC. Sollte diese Methode nicht funktionieren muß
49:     auf DOS ausgewichen werden.
50:
51:   type regs=record
52:     ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
53:   end;
54:   var reg:regs;
55:   begin
56:     reg.ax:=0000; AH=0 Zeit lesen
57:     Intr($1A,reg); DOS Interrupt Uhrzeit
58:     std:=reg.cx; High-Word = Stunden
59:     min:=reg.dx; Low-Word = min+sec in 18,2 Hz
60:   end;
61: end;{-----}
62:
63: function stop:longstring;
64: var stdl,minl,sec,tel:integer;
65: s,m,c,h:string[10];
66: begin
67:   minl:=memw[$0040:$006C];
68:   stdl:=memw[$0040:$006E];
69:   tel:=((minl-min) mod 1092) mod 18; str(tel:3,h);
70:   sec:=((minl-min) mod 1092) div 18; str(sec:3,c);
71:   min:=(minl-min) div 1092; str(min:3,m);
72:   std:=stdl-std; str(std:3,s);
73:   stop:=(s+' ':'+m+' ':'+c+' ':'+h');
74: end;{-----}
75:
76: begin {-----Hauptprogramm-----}
77:   {-----Turbo-Pascal Standardausgabe-----}
78:   clrscr;
79:   start;
80:   for yw:=1 to 24 do
81:     for xw:=1 to 70 do begin
82:       Ausgab1('Probe-1',yw,xw);
83:     end;
84:   zeit[1]:=stop;
85:   {-----Direkter Zugriff auf VideoPage-----}
86:   clrscr;
87:   start;
88:   for yw:=1 to 24 do
89:     for xw:=1 to 70 do begin
90:       Ausgabe2('PROBE-2',yw,xw);
91:     end;
92:   zeit[2]:=stop;
93:   {-----Assemblernahe Programmierung-----}
94:   clrscr;
95:   start;
96:   for yw:=1 to 24 do
97:     for xw:=1 to 70 do begin
98:       Ausgabe3('PROBE-3',yw,xw);
99:     end;
100:  zeit[3]:=stop;
101:  {-----Assemblerprogrammierung-----}
102:  clrscr;
103:  start;
104:  for yw:=1 to 24 do
105:    for xw:=1 to 70 do begin
106:      Ausgabe4('PROBE-4',yw,xw);
107:    end;
108:  zeit[4]:=stop;
109:  {-----Performance-----}
110:  clrscr;
111:  write('Dauer der Prozeduren ');
112:  writeln('STD : MIN : SEC : HUN');
113:  writeln;
114:  writeln('Turbo-Pascal braucht : ',zeit[1]);
115:  writeln('Direktzugriff auf BS : ',zeit[2]);
116:  writeln('Assemblernah : ',zeit[3]);
117:  writeln('Assembler : ',zeit[4]);
118: end.

```

Listing 4. »quickscreen« macht den Bildschirm schnell

Real-Time-Clock im AT manipulieren



Sicher hat sich der eine oder andere Anwender schon gefragt, welche Daten beim Setup-Programm auf einem AT in den batteriegepufferten Speicherbereich der Real-Time-Clock geschrieben werden. DOS International lüftet dieses Geheimnis und zeigt, wie man die Daten in der Real-Time-Clock auslesen und manipulieren kann.

Recht umständlich ist es schon, jedesmal die Diagnosediskette in das Laufwerk zu legen und ewig auf ein Ergebnis zu warten, nur um die Daten in der RTC (Real-Time-Clock) lesen zu können. Das muß nicht so sein, mit dem nötigen Wissen versehen, kann man die Informationen aus dem Baustein in eine Datei schreiben und bei Bedarf wieder einlesen. Insgesamt 64 Byte stehen batteriegepuffert zur Verfügung und enthalten bei jedem IBM-kompatiblen AT wichtige Systemparameter und Daten. Zur Übersicht über die Art der gespeicherten Informationen dienen die in diesem Beitrag enthaltenen Tabellen. Tabelle 1 gibt den Inhalt des RTC-RAM wieder, wie er für einen AT definiert ist. Es gibt auch Abweichungen von dieser Festlegung, die allerdings in der Regel nur im Bereich der 8086-PCs zu finden sind. Das bekannteste Beispiel hierfür ist wohl der Schneider PC, der eine funktionsgleiche Hardwareuhr eingebaut hat. Die Art der gespeicherten Daten unterscheidet sich aber beim Schneider PC ganz deutlich von den beim AT enthaltenen Informationen. Auch Schneider-Anwender brauchen sich nicht mit umständlichen Initialisierungen herumzuschlagen: Hier sind Dinge wie Zeit, Datum, Bildschirmmodus, Farbeinstellungen und RAM-Disk-Größe sofort nach dem Einschalten korrekt vorhanden. Tabelle 2 zeigt als Referenz die Belegung des RTC-RAM beim Schneider PC.

Der RTC-Speicher enthält unter Umständen über Jahre hinweg die gleichen Informationen

Die Daten in der RTC (Real-Time-Clock) bleiben (abgesehen von Datum und Zeit) in aller Regel unverändert, es sei denn, Erweiterungen werden eingebaut. Nun muß nicht jede Erweiterungskarte in der RTC vermerkt werden, sondern nur Hauptspeicherausbaueinheiten und Massenspeicher wie Festplatte und Diskette. Sollte man den Bildschirmadapter gegen einen besseren, möglicherweise gegen einen »Enhanced Grafik Adapter« (EGA) austauschen, so müssen

Sie diese Änderung natürlich auch dem System mitteilen. Besonders unglückliche Umstände bringen aber die gespeicherten Informationen im Chip durcheinander, wie fehlerhafte Netzteile oder auch alternde Bauteile. Es kann auch vorkommen, daß die Batterie, die den Datenerhalt sichert, schlicht und einfach leer ist. Denn mit einer Batterie werden die Daten bis zum nächsten Einschalten des Computers im RTC festgehalten.

Genau vom Zeitpunkt des Ausschaltens des Gerätes an übernimmt nämlich eine Batterie die Funktion der Netzspannung. Der Baustein und seine interne Uhr werden weiterhin mit Spannung versorgt und behalten ihre Daten

Adresse	Inhalt
0	Sekunden Uhrzeit
1	Sekunden Alarm
2	Minuten Uhrzeit
3	Minuten Alarm
4	Stunden Uhrzeit
5	Stunden Alarm
6	Wochentag
7	Tagesdatum
8	Monatsdatum
9	Jahr
10	Status-Register A
11	Status-Register B
12	Status-Register C
13	Status-Register D
14	Diagnostic-Status
15	Grund für Shutdown
16	Diskettentyp A und B
17	Reserviert
18	Festplattentyp C und D
19	Reserviert
20	Equipmentbyte
21	Low-Byte Memory
22	High-Byte Memory
23	Low-Byte Expansion
24	High-Byte Expansion
25 bis 45	Reserviert
46 bis 47	Checksum RTC
48	wie Byte 17, aktuell
49	wie Byte 18, aktuell
50	Jahrhundert
51	Flags
52 bis 63	Reserviert

Tabelle 1. RTC-RAM Motorola MC146818 (IBM-AT und Kompatible)

für ungefähr zwei Jahre. So war die Funktion von den Entwicklern geplant.

In dem Falle also, daß diese Batterie erschöpft ist, wartet der ein oder andere unglückliche Anwender eines Personalcomputers dann vergeblich auf den Start des Betriebssystems PC- oder MS-DOS. Für den Fachhändler ist oder sollte diese Unpäßlichkeit eine Angelegenheit von wenigen Minuten sein, der normale Benutzer steht nun aber vor einer fast unlösbaren Aufgabe.

Die Batterie zu erneuern ist sicher kein Problem, aber die Daten im Baustein sind verloren, oder zumindest gehörig durcheinander. Selten hat man nun genau die benötigte Software zur Hand, um die Informationen wieder in den Baustein zu schreiben. Bei einem IBM-AT (XT-286) ist es die sogenannte Diagnosediskette, die dem geplagten Anwender hier weiterhilft. Wer einen Kompatiblen sein eigen nennt, der hat hoffentlich ein sogenanntes Setup-Programm mitgeliefert bekommen. Nur mit der Software allein ist es allerdings noch nicht getan, denn wer weiß denn schon genau, welche Konfiguration für den eigenen Computer richtig ist?

Um den RTC-Baustein zu lesen und gegebenenfalls zu beschreiben, müssen erst einmal die gespeicherten Daten in den 64 Byte und deren Funktionen erklärt werden. In Tabelle 1 ist für eine erste Information die Belegung der fraglichen 64 Byte angegeben. Teilweise sind die Informationen auch bitweise organisiert, beziehungsweise müssen so interpretiert werden, wie zum Beispiel das Byte Nummer 14 sowie die in den Registern enthaltenen Informationen. Die Register A bis D beinhalten Steuerinformationen zum Betrieb des Bausteins. Zugriff hat man auf die beiden Register A und B, während C und D für die interne Verwaltung benutzt werden. Sie sollten nur gelesen, aber nicht beschrieben werden. Tabelle 3 enthält eine Übersicht über die Bit-Belegung der Kontrollregister.

Angesprochen wird der Baustein von der Hardware über die Port-Adressen 70hex und 71hex. Über die Adresse 70hex teilt man dem RTC die gewünschte Operation mit (lesen oder schreiben), während über Portadresse 71hex die entsprechenden Bytes oder Register ausgelesen und auch beschrieben werden. Das System benutzt das Kontrollregister B für die Zeitmessung, Register A bleibt unbenutzt. Es gibt, wie aus Tabelle 1 zu sehen ist, noch Speicherstellen im RTC, die für spätere Erweiterungen reserviert sind (Diskettenlaufwerke und Festplatten). Von der Speicherstelle 13 liest das System nur Bit 7. Ist es auf 1 gesetzt, so ist die Spannungsversorgung des RTC in Ordnung, hat es den Wert 0, so war kurzzeitig keine Spannung vorhanden. Speicherstelle 12 hingegen enthält interne Informationen, zu denen es keine näheren Angaben gibt.

Die Bits der Register A und B haben beide die gleiche Bedeutung, da sie zum Timer gehören. Mit gesetztem Bit 1 werden die Stunden im 24-Stunden-Modus gezählt, hat es den Wert 0, so zählt der Baustein im 12-Stunden-Modus. Das Bit 2 ist für das Format (BCD/Binär) der Zahlen verantwortlich, auf 0 gesetzt wird BCD-Format angezeigt. Das BIOS des Computers setzt über die Funktion 1Ahex beim Systemstart dieses Bit wieder auf 0, zumindest bei einem IBM-AT verhält es sich so. Mit Bit 5 wird ein eventueller Alarm aktiviert, wenn das Bit auf 1 gesetzt ist. In diesem Fall wird der Interrupt 4Ahex ausgeführt, den man auf eine selbstgeschriebene Alarmroutine setzen kann. Im BIOS steht nur ein simples »reti«, es ist also keine Funktion implementiert. Ein Interrupt, der nach jeder Sekunde stattfinden soll (welcher das ist, war nicht herauszufinden), wird bei gesetztem Bit 6 ausgeführt. Mit dem Bit 7 wird der Timer des RTC-Chip gestoppt, wenn es den Wert 1 hat und startet erst wieder, wenn es per Programm gesetzt wird. Schaltet man beispielsweise den Computer aus, nachdem man das Bit 7 gesetzt hat, so erscheint beim Booten die Meldung »Timer-chip stopped« auf dem Bild-

Adresse	Inhalt
0	Sekunden Uhrzeit
1	Sekunden Alarm
2	Minuten Uhrzeit
3	Minuten Alarm
4	Stunden Uhrzeit
5	Stunden Alarm
6	Wochentag
7	Tagesdatum
8	Monatsdatum
9	Jahr
10	Status-Register A
11	Status-Register B
12	Status-Register C
13	Status-Register D
14 bis 19	Zeit und Datum letzte Benutzung
20	Checksum RTC
21 bis 22	Tasten-Übersetzung [Enter]
23 bis 24	Tastenübersetzung [Del-Forward]
25 bis 26	Tastenübersetzung Joystick Feuer 1
27 bis 28	Tastenübersetzung Joystick Feuer 2
29 bis 30	Tastenübersetzung Mausknopf 1
31 bis 32	Tastenübersetzung Mausknopf 2
33	Maus-Skalierung X-Richtung
34	Maus-Skalierung Y-Richtung
35	Bildschirmmodus/Anzahl Laufwerke
36	Bildschirmfarbattribute
37	RAM-Disk-Größe in 2-KByte-Blöcken
38 bis 39	RS-232-Parameter
40 bis 63	unbenutzt

Tabelle 2. RTC-RAM Hitachi HD146818 (Schneider PC)

schirm; dazu noch der Hinweis, man möge das Programm Setup starten. Diese Meldungen sind aber abhängig vom verwendeten BIOS und können bei anderen ATs anders lauten oder vollkommen fehlen.

Das Bit 0 des Registers B ermöglicht eine Umrechnung von Sommer- auf Winterzeit. Hat das Bit den Wert 1, so ist die Sommerzeit eingestellt. Da es aber nur durch entsprechende Programme, zum Beispiel den Microsoft C-Compiler direkt unterstützt wird, hat es normalerweise keine Auswirkungen auf das Betriebssystem. Eine Manipulation an den Bit 3 und 4 des Registers B zeigte keine erkennbare Wirkung. Die Definition der Bits lautet in Reihenfolge »Square wave enabled« und »Update ended interrupt en-

Register	Bit	Funktion
A	0	Rate Selection Bit 0
	1	Rate Selection Bit 1
	2	Rate Selection Bit 2
	3	Rate Selection Bit 3
	4	Divider Bit 0
	5	Divider Bit 1
	6	Divider Bit 2
B	7	Update in Progress
	0	Daylight Savings Enable
	1	24/12 Hour Format
	2	Data Mode
	3	Square Wave Enable
	4	Update-ended Interrupt Enable
	5	Alarm Interrupt Enable
C	6	Periodic Interrupt Enable
	7	Set Date/Time
	0 bis 3	(immer Null)
	4	Update-ended Interrupt Flag
D	5	Alarm Interrupt Flag
	6	Periodic Interrupt Flag
	7	Interrupt Request Flag
	0 bis 6	(immer Null)
	7	Valid RAM Time Bit

Tabelle 3. Bit-Belegung der Kontrollregister (AT und Schneider PC)

abled«. Leider schweigen sich hier die Dokumentationen über den Sinn dieser Speicherstellen aus. Tabelle 4 gibt einen Überblick über weitere Bit-codierte Informationen der RTC.

Nachdem nun alle Daten und deren Form im RTC bekannt sind, fehlt nur noch die Ansteuerung der Ports 70hex und 71hex, die auch in einer bestimmten Weise erfolgen muß. Der Ablauf wiederholt sich dann für jedes Byte, das gelesen oder geschrieben wird. Über Port 70hex wird ein sogenanntes Adreßregister angesprochen, das anhand des übergebenen Wertes eine Speicherstelle im internen RAM-Bereich adressiert. Beim Videocontroller 6845 funktioniert dies übrigens auf die gleiche Art.

Ein Beispiel: Die Speicherstelle 12hex (Festplatte) im RTC soll modifiziert werden. Dazu ist zu der Speicherstelle, die adressiert werden soll, der Wert 128 (80hex) zu addieren. Durch diese Operation wird das Bit 7 gesetzt, was einen NMI (Non Maskable Interrupt) verhindert, der wiederum ein Schreiben in den RTC unterbinden könnte. Der NMI wird im Falle eines RAM-Fehlers vom System ausgelöst, was je nach Auslegung der Hardware auch vom RTC erfolgen kann. Das Bit 7 sollte also gesetzt werden, ein Versuch mit Bit 7 auf Null ergab teilweise Probleme beim Schreiben in den Baustein. Die RTC-Speicherstelle 46 bis 47 (Prüfsumme) kann hier vollkommen außer acht gelassen werden, da außer Datum und Uhrzeit keine Daten verändert werden. Auch die Zeitfunktion wird nicht mit in die Prüfsumme einbezogen, was auch unlogisch wäre. Bei einer Änderung der Konfigurationsdaten ohne die Prüfsumme neu zu berechnen und in die entsprechende Speicherstelle zu schreiben, resultiert beim nächsten Start eine Fehlermeldung.

In Basic würde man also »out &h70,&h92« eingeben, um dem Adreßregister mitzuteilen, daß die Speicherstelle 12 im RTC angesprochen werden soll. Der nächste Programmschritt, um die Adresse 12 zu lesen, ist dann »inp (&h71)«, da an Port 71hex die Informationen bereitgestellt werden. Soll nicht gelesen, sondern zum Beispiel die Festplatte C von Typ 2 auf 3 verändert werden, so ist statt

des Input-Befehls wieder ein Out-Befehl auszuführen, diesmal aber zum Port 71hex. Also lautet der entsprechende Basic-Befehl »out &h71,&h30« – weil das entsprechende RTC-Byte beide Festplatten gleichzeitig beschreibt. Würde man statt 30hex einfach 3hex auf den Port schreiben, so gäbe es keine Festplatte C mehr, dafür aber die Festplatte D Typ 3, das wäre eine Festplatte mit 31 MByte Kapazität und sechs Köpfen. Eine gewisse Vorsicht ist also durchaus geboten, wenn man im System manipuliert; diese Daten werden beim nächsten Systemstart gelesen und – außer einer Fehlermeldung passiert nichts weiter. Egal was für Daten im RTC die derzeit gültige Konfiguration beschreiben, das BIOS versucht immer erst ein Laufwerk A oder C anzusprechen, bevor es wirklich aufgibt; auch wenn angeblich keines vorhanden ist, wenn man den Bytes 16 und 20 im RTC glauben soll. Nur ist eine Fehlermeldung lästig und die volle Leistung im AT wird mangels Information nicht genutzt, wie Speicherausbau, Coprozessor und so weiter.

Das C-Programm »cmos« (Listing 1) liest einfach die Daten im RTC, (alle 64 Byte) und schreibt sie in eine Datei »cmos.rtc«. Der Name ist willkürlich gewählt und kann bei Nichtgefallen geändert werden. Die kleine Include-Datei »cursor.c« (Listing 2) muß aber mit eingebunden werden, wenn man Wert auf Cursorsteuerung legt. Andernfalls streichen Sie die Funktionen »clscr« und »gotoxy« ersatzlos aus dem Programm, beziehungsweise ersetzen Sie sie durch Funktionen für den Ansi-Treiber.

Die vom Cmos-Programm gefundene Konfiguration wird sofort auf dem Bildschirm angezeigt. Die Prüfsumme der RTC wird nicht aus der entsprechenden Speicherstelle im RTC gelesen, sondern neu berechnet. Eine Warnung, sofern eine Differenz der beiden Werte auftritt, wurde nicht mitprogrammiert, der zusätzliche Einbau sollte einem Programmierer aber keine Schwierigkeiten bereiten. Da bei einer so erzeugten Datei die Prüfsumme im RTC nicht verändert wird, kann die Datei »cmos.rtc« auch zum späteren Zeitpunkt (zum Beispiel nach Wechseln der Batterien) wieder in die 64 Byte des RTC-Chip geschrieben werden.

Adresse	Bit	Funktion
14	0	nicht benutzt
	1	nicht benutzt
	2	1 = Zeit ungültig
	3	1 = WDC-Fehler
	4	1 = RAM-Fehler
	5	1 = ungültige Konfiguration
	6	1 = RTC-Checksum-Fehler
	7	1 = Spannungsausfall
16	7 bis 4	Floppy-Typ A
	3 bis 0	Floppy-Typ B
	Typ 0	kein Laufwerk
	1	48 Tpi Double Sided (360 KByte)
	2	96 Tpi High Capacity (1,2 MByte)
18	7 bis 4	Festplatte C
	3 bis 0	Festplatte D
(Typ s. Tab. 4)		
20	0	1 = Diskette vorhanden
	1	1 = 80287 installiert
	2 bis 3	nicht benutzt
	4 bis 5	10 = Farbe 40stellig
		01 = Farbe 80stellig
		11 = Monochrom
	6	keine Funktion
	7	0 = ein Laufwerk
		1 = zwei Laufwerke
51	0 bis 5	nicht benutzt
	6	benutzt vom Setup-Programm
	7	erweiterter Speicher

Tabelle 4. Zusätzliche Bit-Informationen im RTC (IBM-AT und Kompatible)

Typ	Zylinder Tracks	Köpfe Heads	Landezone	Formatierte Kapazität in MByte
1	306	4	305	10,4
2	615	4	615	20,9
3	615	6	615	31,3
4	940	8	940	63,9
5	940	6	940	47,9
6	615	4	615	20,9
7	462	8	511	31,4
8	733	5	733	31,1
9	900	15	901	114,7
10	820	3	820	20,9
11	855	5	855	36,3
12	855	7	855	50,8
13	306	8	319	20,8
14	733	7	733	43,6
16	612	4	663	20,8
17	977	5	977	41,5
18	977	7	977	58,1
19	1024	7	1023	60,9
20	733	5	732	31,1
21	733	7	733	43,6
22	733	5	733	31,1
23	306	4	336	10,4
37	830	10	830	70,5
38	823	10	824	69,9
39	615	4	664	20,9
40	615	8	664	41,8
41	917	15	918	116,9
42	1023	15	1024	130,4

Die Tabelle erhebt keinen Anspruch auf Vollständigkeit, sie soll nur einen Anhaltspunkt bei Installationen geben.

Tabelle 5. Festplattentypen bei einem AT


```

62: printf("\n Datum in Form [TT MM JJ] eingeben
63: : ");
64: scanf("%2d%2d%2d",&tag,&mon,&jhr);
65: /*****
66: *
67: Aus dezimalen Eingaben wieder BCD-Format packed herstellen
68: *****/
69: */
70:
71: n= (6* (tag/10) + tag); daten[7] = n;
72: n= (6* (mon/10) + mon); daten[8] = n;
73: n= (6* (jhr/10) + jhr); daten[9] = n;
74: printf("Uhrzeit in Form [SS MM ss] eingeben:
75: ");
76:
77: scanf("%2d%2d%2d",&std.&min.&sek);
78: n= (6* (sek/10) + sek); daten[0] = n;
79: n= (6* (min/10) + min); daten[2] = n;
80: n= (6* (std/10) + std); daten[4] = n;
81: }
82:
83: /*****
84: Umwandlung abgeschlossen
85: *****/
86:
87: for (i=0; i < 64; i++)
88: {
89:     outp(port,base);
90:     /* Daten werden in RTC geschrieben */
91:     outp(port1,daten[i]);
92:     ++base; /* nächste Speicherstelle */
93:     /* voila, das war's */
94: }
95: showconf(); /* Zeige eingelesene Konfiguration */
96: gotoxy(20,25);
97: if (dos3)
98: printf("\tNach Systemstart Datum/Uhrzeit eingeben\n");
99: );
100: printf("Bitte jetzt per Warmstart [Ctrl-Alt-Del] ne
101: u Booten \n");
102: break;
103: default:
104:     printf("\n\t\tNur S oder L eingeben\n");
105: break;
106: )
107: exit(0);
108: }
109:
110: showconf()
111: /* zeigt die Konfiguration eines ATs anhand der Daten im Pu
112: ffer an */
113: {
114: int i,m,k;
115: clrscr();
116: gotoxy(1,1);
117: printf("\t\t RTC-MC146818-Inhalt sichern / schreiben \n");
118: printf("\n");
119: printf("\t\t\t\t\t(C)copyright M.Bormann 1986/87 \n");
120: gotoxy(9,15);
121: printf("Folgende Systemkonfiguration ist vorhanden :\n");
122:
123: /* Diskettenlaufwerke */
124:
125: i = daten[16];
126: m = i >> 4; /* 4 Bits nach rechts schieben */
127: k = i & 0017; /* 4 High-Bits löschen */
128: if (m == 1) printf("Laufwerk A: 360 KByte\n");
129: if (m == 2) printf("Laufwerk A: 1,2 MByte\n");
130: gotoxy(10,40);
131: if (k == 1) printf("Laufwerk B: 360 KByte\n");
132: if (k == 2) printf("Laufwerk B: 1,2 MByte\n");
133: if (k == 0) printf("Laufwerk B: nicht vorhanden\n");
134:
135: /* Festplattenlaufwerke */
136:
137: i = daten[18];
138: m = i >> 4; k = i & 0017; /* Ausmaskieren */
139: if (m == 15) {
140: i = (daten[25] - 15); /* Erweiterter Bereich für Drive C
141: > Typ 14 */
142: m +=i;
143: }
144: if (k == 15) {
145: i = (daten[26] - 15); /* Das gleiche für Drive D */
146: k +=i;
147: }
148: printf ("Festplatte C: ");
149: if (!m)
150: printf ("nicht vorhanden\n");
151: else
152: printf("Type %d\n",m);
153: gotoxy(11,40);
154: printf ("Festplatte D: ");
155: if (!k)
156: printf ("nicht vorhanden\n");
157: else
158: printf ("Type %d\n",k);
159:
160: /* Hauptspeicher
161: Low-Byte DATEN[21] zählt Bytes
162: High-Byte DATEN[22] zählt in 256-KByte Blöcken
163: */
164:
165: i = daten[21]; k= daten[22];
166: m = k*256 + i; /* Formel : 256*High_Byte + Low_Byte */
167: printf ("Hauptspeicher : %d KByte\n",m);
168: gotoxy(12,40);

```

```

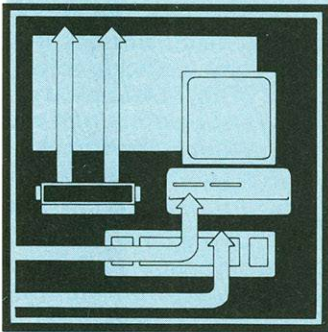
241: /* Erweiterter Speicher über 1 MByte
242:   Organisation wie Hauptspeicher
243: */
244:
245: i = daten[23]; k = daten[24];
246: m = k*256 + i;
247: printf ("Extended Memory : %d KByte\n",m);
248:
249: /* Co-Prozessor 80287 und Bildschirm
250:   nur im Equipment-Byte zu finden
251: */
252:
253: i = daten[20];
254: m = i & 0x30;
255: if (i & 0x02) printf("Co-Prozessor installiert\n");
256: gotoxy(15,0);
257: printf("Video-Adapter : ");
258: if (m == 0x30) printf("MONOCHROM\n");
259: if (m == 0x20) printf("COLOR \n");
260: if (m == 0x00) printf("EGA\n");
261:
262: /* Checksum RTC
263:   beinhaltet die Bytes 16 bis 31 im IBM-AT
264:   steht in den Bytes 46 und 47 der RTC.
265: */
266:
267: m = 0;
268: for (i = 16; i < 32; i++)
269:   m += daten[i];
270: gotoxy(15,40);
271: printf ("Checksum RTC : %x Hexadezimal \n",m);
272: }
273:

```

```

0: ***** CURSOR.C *****
1: Hier wird die Cursorsteuerung erledigt
2: Direkte Bios-Funktion über Interrupt 10
3: für Bildschirm löschen und Positionierung
4: des Cursors.
5: *****/
6:
7: #include <dos.h> /* Definitionen der Register */
8: gotoxy(x,y) /* Positioniere Reihe,Spalte */
9: int x,y;
10: { union REGS ir,or;
11:   ir.h.ah=2; ir.h.dh = x; ir.h.dl = y; ir.h.bh=0;
12:   int86 (0x10,&ir,&or);
13: }
14:
15: int getmon() /* Gibt aktuellen Videomodus zurück */
16: { union REGS ir,or;
17:   ir.h.ah=0x0F; ir.h.al = 0;
18:   int86 (0x10,&ir,&or); /* Wird in CMOS nicht benutzt */
19:   return (or.h.al);
20: }
21:
22: setmon(mode)
23: int mode; /* setzen anderen Videomodus */
24: { union REGS ir,or;
25:   ir.h.ah=0; ir.h.al = mode;
26:   int86 (0x10,&ir,&or);
27: }
28:
29: clrscr() /* Bildschirm löschen */
30: { union REGS ir,or;
31:   ir.x.ax=0x0700; ir.x.cx=0; ir.h.dh=24;
32:   ir.h.dl=79; ir.h.bh = 7;
33:   int86 (0x10,&ir,&or);
34: }

```

Die DOS-Trickkiste

Tips & Tricks sind die Würze im PC-Alltag, denn oftmals hilft ein kleiner Hinweis oder ein kurzes Programm genau da weiter, wo Handbücher und integrierte Software-Pakete aufhören. Daher bringt DOS International Monat für Monat die besten Tips und Tricks rund um den PC, und das nicht in homöopatischer Dosierung, sondern gleich seitenweise. Für diese Sonderausgabe der DOS International haben wir eine Auswahl der besten bisher erschienenen Tips und Tricks rund um den PC für Sie zusammengestellt.

Programme drucken mit dem Turbo-Editor

Besonders bei der Bearbeitung längerer und komplexerer Programme in Turbo Pascal braucht man des öfteren einen Ausdruck des gesamten Programms oder zumindest einiger Teile davon. Mit dem folgenden Trick ist das sehr einfach ohne Verlassen des Editors möglich: Markieren Sie den zu druckenden Programmbereich als Textblock. Wenn Sie das gesamte Programm drucken wollen, dann benutzen Sie die Tastenfolge [Ctrl-QR], [Ctrl-KB], [Ctrl-QC] und [Ctrl-KK]. Rufen Sie anschließend den Befehl zum Schreiben eines Blocks in eine Datei auf, nämlich [Ctrl-KW]. Turbo Pascal fragt jetzt nach einem Dateinamen zum Speichern des markierten Textblocks. Auf diese Frage antworten Sie nun wahlweise mit »lpt1« oder »prn«, um die Ausgabe zum Drucker zu lenken. Turbo Pascal, wie die meisten Computerprogramme nicht gerade mit übermäßiger Intelligenz gesegnet, erlaubt sich nun die doch recht dumme Frage: »Overwrite old lpt1.pas?«, die Sie getrost mit »Y« beantworten dürfen. Vergewissern Sie sich aber vorher, ob Ihr Drucker Online ist. Der markierte Textbereich wird nun auf den Drucker ausgegeben. Zum Schluß noch ein kleiner Hinweis: Wenn Sie Turbo Pascal gerne ein wenig irritieren möchten, dann versuchen Sie es doch einmal mit »con« statt »lpt1«...

(Hans-Jürgen Selters/ev)

Copy-Befehl als Texteditor

Nicht allen PC-Anwendern ist die Tatsache bekannt, daß der Copy-Befehl auch zur Eingabe von kurzen Texten in Dateien benutzt werden kann. Die Befehlszeile

```
copy con text.dat
```

bewirkt zum Beispiel, daß alle folgenden Eingabezeilen von der Konsole (con), also der Tastatur, in die Datei »text.dat« kopiert werden. Wenn die Datei noch nicht existiert, dann wird sie neu angelegt, andernfalls wird sie automatisch gelöscht. Damit hat man praktisch jederzeit einen kleinen, wenn auch recht primitiven, Texteditor zur Verfügung. Jede Eingabezeile kann wie unter DOS üblich editiert werden, solange die Return-Taste noch nicht betätigt wurde. Um die Eingabe in die Datei zu beenden, muß das Zeichen [Ctrl-Z] (Dateiende-Kennung) eingegeben und danach wieder die Return-Taste gedrückt werden. [Ctrl-Z] wird auch durch Drücken der Funktionstaste [F6] erreicht. Oftmals möchte man aber einen Text an eine bereits bestehende Textdatei anhängen. In diesem Falle kann man sich die Eigenschaft des Copy-Befehls zunutze machen, Dateien aus mehreren Quellen auf Wunsch miteinander zu kombinieren. Dazu geben Sie beispielsweise die beiden folgenden Befehlszeilen ein:

```
ren text.dat text.bak
```

```
copy text.bak +con text.dat
```

Die erste Zeile bewirkt, daß die Datei »text.dat« in »text.bak« umbenannt wird. In der zweiten Zeile wird die Datei dann zusammen mit der Tastatureingabe (con) in die

Datei »text.dat« gespeichert. Im Ergebnis enthält »text.bak« dann die alte Textversion, »text.dat« dagegen den ursprünglichen Text, plus die neu hinzugekommenen Tastatureingaben. Wem die Eingabe der zwei Kommandozeilen zu kompliziert ist, der kann sich die Sache über eine Stapeldatei einfacher machen. Sie erzeugen diese Stapeldatei, indem Sie folgende Zeilen eingeben:

```
copy con ed.bat
echo off
del %1.bak
ren %1.dat %1.bak
copy %1.bak +con %1.dat
type %1.dat
```

Nach der letzten Zeile beenden Sie die Eingabe durch [F6] oder [Ctrl-Z] und die Return-Taste. Sie haben damit eine Stapeldatei namens »ed.bat« erzeugt, die zum Beispiel folgendermaßen aufgerufen wird:

```
ed text
```

Auf dieses Kommando hin wird automatisch die Datendatei in eine Bak-Datei umgewandelt. Alle Tastatureingaben werden anschließend an den Inhalt dieser Datei angehängt. Zum Schluß wird die erweiterte Datei durch das Type-Kommando am Bildschirm ausgegeben. Falls die Datendatei noch nicht existiert, erscheint eine Fehlermeldung, durch die Sie sich aber nicht stören lassen sollten.

Eine weitere interessante Anwendung des Copy-Befehls ist die Form

```
copy con prn
```

Hier werden alle Zeilen, die Sie über die Tastatur eingeben, direkt zum Drucker geschickt und dort ausgedruckt. Dies geht ebenfalls so lange, bis Sie [Ctrl-Z] oder die F6-Taste betätigen.

(Jochen Kruse/ev)

Bit-Manipulationen mit Turbo Pascal

Wenig bekannt ist die Tatsache, daß Turbo Pascal die Verwendung logischer Operatoren auch mit Integer-Zahlen erlaubt. Ähnlich wie in Basic ist es somit möglich, einzelne Bits von Integer- oder Byte-Werten gezielt zu manipulieren. Die logischen Operatoren Not, And, Or und Xor wirken bei Anwendung auf Integer- oder Byte-Werte auf jedes einzelne Bit der Zahlendarstellung. Einige Beispiele mögen dies verdeutlichen:

Wert := not -1;

Nach dieser Zuweisung enthält »Wert« den Zahlenwert 0, der dadurch zustande kommt, daß auf jedes einzelne Bit der binären Darstellung von -1 der Not-Operator angewendet wird. Jedes einzelne Bit wird dadurch invertiert.

Dezimal Binär

```
-1      1111 1111 1111 1111
0       0000 0000 0000 0000
```

Nun ein Beispiel für den And-Operator:

Wert := 17 and 5;

Hier erhält »Wert« das Ergebnis aus der binären And-Verknüpfung der Dezimalzahlen 17 und 5 zugewiesen. Das Ergebnis ist allerdings nicht 22, wie bei der Addition, son-

dern 1, denn ein Bit im Ergebnis wird nur dann gesetzt, wenn das entsprechende Bit im ersten und im zweiten Operanden gesetzt ist:

```
Dezimal  Binär
17      0000 0000 0001 0001
and 4    0000 0000 0000 1001
```

```
-----
1      0000 0000 0000 0001
```

Analog funktioniert das Ganze auch mit den Operatoren Or und Xor. Die Bit-weise Verknüpfung bei den vier Operatoren ist einfach zu merken, denn die einzelnen Bits einer Integer- oder Byte-Zahl werden jedes für sich wie eine Boolean-Variable behandelt:

- Not invertiert jedes einzelne Bit; aus 0 wird 1 und umgekehrt.
- And setzt ein Bit im Ergebnis, wenn das entsprechende Bit im ersten und im zweiten Operanden gesetzt ist.
- Or setzt ein Bit im Ergebnis, wenn das entsprechende Bit im ersten oder im zweiten Operanden oder in beiden gesetzt ist.
- Xor setzt ein Bit im Ergebnis, wenn das entsprechende Bit entweder im ersten oder im zweiten Operanden gesetzt ist. Ist es in beiden Operanden gesetzt, dann wird es im Ergebnis gelöscht.

Wenn Sie Integer-Operationen mit logischen Operatoren in eigenen Programmen verwenden, dann sollten Sie sich allerdings der Tatsache bewußt sein, daß diese Vorgehensweise vom Pascal-Standard nicht abgedeckt wird. Ein solches Programm läuft also nur unter Turbo Pascal und ist in der Regel nicht auf andere Compiler portierbar.

(ev)

Tab-Funktion für Turbo Pascal

Die in Basic oft nützliche Tab-Funktion zur spaltenrichtigen Ausgabe von Daten ist in Turbo Pascal leider nicht vorhanden. Um in Basic beispielsweise drei Zahlenkolonnen nebeneinander in den Spalten 5, 15 und 25 auf den Bildschirm zu bringen, bedarf es nur einer einzigen Print-Anweisung:

```
print tab(5);a1;tab(15);a2;tab(25);a3
```

In Turbo Pascal muß hier umständlich mit Gotoxy gearbeitet werden. Die kurze Pascal-Funktion aus Listing 1 simuliert die Tab-Funktion von Basic und kann völlig analog in Write-Anweisungen verwendet werden. Mit dieser Funktion läßt sich das oben genannte Beispiel analog zu Basic in einer einzigen Anweisung unterbringen:

```
writeln(tab(5),a1,tab(15),a2,tab(25),a3);
```

```
0: program TabDemo;
1:
2: function tab(x: integer): char;
3: begin
4:   if x > 80 then x := x mod 80
5:   else if x < 1 then x := 1;
6:   GotoXY(x, WhereY);
7:   tab := chr(8);
8: end;
9:
10: var i: integer;
11:
12: begin
13:   for i := 1 to 7 do writeln(tab(i*10), 'Tab');
14: end.
```

Listing 1. Eine Tab-Funktion in Turbo Pascal

Wie Sie bei näherem Hinsehen feststellen werden, wird in dieser Funktion allerdings mit einigen kleinen Tricks gearbeitet. Sinn und Zweck der Tab-Funktion ist es ja, den Cursor während der Textausgabe auf eine bestimmte Spaltenposition am Bildschirm zu setzen. Dies geschieht durch die Gotoxy-Anweisung innerhalb der Funktion. Der von der Tab-Funktion zurückgegebene Funktionswert ist hierfür eigentlich völlig uninteressant. Das Wesentliche an der Tab-Funktion ist also nicht das Ergebnis, das sie liefert, sondern der Nebeneffekt der Cursorpositionierung.

Im Prinzip wäre es daher egal, von welchem Typ die Funktion ist. Doch die Tab-Funktion soll in einer Write-Anweisung verwendet werden, was nichts anderes bedeutet, als daß Turbo Pascal versucht, den von der Funktion gelieferten Datenwert irgendwie auszudrucken. Daher wurde für die Funktion der Datentyp Char gewählt und als Funktionswert wird konstant das ASCII-Zeichen für Tab, nämlich chr(8), zugewiesen. Dadurch wird erreicht, daß die Write-Prozedur im Falle von Tab tatsächlich nichts druckt. Wegen der Verwendung der Gotoxy-Anweisung ist die Tab-Funktion in dieser Form nur für die Bildschirmausgabe verwendbar. Versuchen Sie nicht, Druckerausgaben oder Textdateien damit zu formatieren.

(Wolfgang Kraul/ev)

Farbiger Bildschirmrand

Das Handbuch zu Turbo Pascal verschweigt geflissentlich, wie man die Farbe der nicht beschreibbaren Umrandung des Bildschirms einstellen kann. Normalerweise hat dieser Rahmen die gleiche Farbe wie der übrige Bildschirmhintergrund. In allen farbigen Textmodi kann aber die Rahmenfarbe separat geändert werden. Dies funktioniert merkwürdigerweise über die eigentlich für die grafische Betriebsart vorgesehene Prozedur »GraphBackGround«.

```
0: program BorderDemo;
1:
2: type color = 0..15;
3: var Farbe: color;
4:
5: procedure Border(Farbe: color);
6: begin
7:   GraphBackGround(Farbe);
8: end;
9:
10: begin
11:   TextMode(C80);
12:   for Farbe := 0 to 15 do
13:     begin
14:       Border(Farbe);
15:       Delay(500);
16:     end;
17:   Border(0);
18: end.
```

Listing 2. Farbe auf dem Bildschirmrand

Das Programm nach Listing 2 demonstriert den Effekt, wobei für die Wahl der Rahmenfarbe aus Gründen der besseren Lesbarkeit eine eigene Prozedur »Border« eingeführt wird. Es versteht sich von selbst, daß dieses Programm nur mit einer Farbgrafikkarte (CGA) und einem Farbmonitor den gewünschten Effekt zeigt. Auf jeden Fall sollten Sie die Border-Prozedur in Ihre Prozedur-Bibliothek aufnehmen, wenn Sie mit Farbe programmieren. Wechselnde Rahmenfarben können in Anwenderprogrammen beispielsweise bestimmte Betriebszustände kennzeichnen oder auf Eingabefehler aufmerksam machen.

(Wolfgang Kraul/ev)

Basic-Aufruf mit Parametern

Die meisten GW-Basic-Versionen erlauben wie BasicA wahlweise den Aufruf des Interpreters mit einer Anzahl von Parametern. Die allgemeine Form des Aufrufs sieht so aus:

```
gwbasic (Dateiname) (<Eingabe) (>)(> Ausgabe)
(/C:Com_Puffer) (/D) (/F:Max_Dateien) (/M:Max_Speicher)
(/S:Satzlänge) (.Max_Block)
```

Alle Parameter in eckigen Klammern können einzeln oder zu mehreren angegeben werden (natürlich ohne die eckigen Klammern). Die einzelnen Parameter haben folgende Bedeutung:

Dateiname: der Name eines Basic-Programms, das sofort geladen und gestartet wird. Beim Aufruf in dieser Form

erfolgt bei Ende des Programms automatisch die Rückkehr ins Betriebssystem.

<Eingabe: Die Standard-Eingabe (Tastatur) wird zur Datei »Eingabe« umgelenkt. Alle Tastatureingaben werden dann aus der angegebenen Eingabe-Datei geholt.

> Ausgabe: Die Standard-Ausgabe (Bildschirm) wird zur Datei »Ausgabe« umgelenkt. Alle Bildschirmausgaben werden dadurch in diese Datei geschrieben statt am Monitor angezeigt zu werden.

>> Ausgabe: Diese Option lenkt wie »> Ausgabe« die Bildschirmausgabe zur angegebenen Datei um. Allerdings wird hier keine neue Datei erzeugt, sondern die Bildschirmausgabe wird an die angegebene Datei angehängt.

/C:Com_Puffer: Hiermit wird die maximale Größe des Kommunikationspuffers für die serielle Datenübertragung festgelegt. Die Voreinstellung für Com_Puffer ist 256. Der maximal mögliche Wert ist 32767. Durch Eingabe von »/C:0« wird die Kommunikationspufferung ganz abgeschaltet.

/D: Die Angabe dieser Option bewirkt, daß die mathematischen Funktionen Atn, Cos, Exp, Log, Sin, Sqr und Tan mit doppelter Genauigkeit berechnet werden.

/F:Max_Dateien: Diese Option legt die maximale Anzahl von Dateien fest, die unter Basic gleichzeitig geöffnet sein können. Die Voreinstellung ist 3, maximal sind 15 Dateien möglich. Die fünf DOS-Standard-Dateien sind hierbei nicht eingerechnet. Der durch Max_Dateien eingestellte Wert muß daher immer um fünf unter der im DOS eingestellten maximalen Anzahl (Datei »Config.sys«) bleiben.

/M:Max_Speicher: Hierdurch wird die Größe des Basic-Datensegments in Bytes festgelegt. Die Voreinstellung ist mit 65535 (genau ein 64-KByte-Segment) bereits der maximale Wert. Im Basic-Datensegment verwaltet der Basic-Interpreter neben einer Reihe von internen Variablen auch das Anwender-Programm und die Basic-Variablen. Eine Begrenzung der Größe des Datensegments durch die M-Option ist im allgemeinen nicht sinnvoll. Eine Ausnahme liegt dann vor, wenn nur wenig Hauptspeicher vorhanden ist (128 KByte) und noch einige speicherresidente Programme neben Basic verwendet werden sollen. Dann ist es unter Umständen sinnvoll, das Basic-Datensegment auf zum Beispiel 16 oder 32 KByte zu begrenzen.

/S:Satzlänge: Dieser Parameter gibt die maximale Satzgröße für Random-Dateien in Byte an. Die Voreinstellung ist 128; möglich sind Werte zwischen 1 und 32767.

/,Max_Block: Hiermit wird der gesamte von Basic erreichbare Hauptspeicherbereich einschließlich den Bereichen für Maschinenprogramme in Paragraphen (16 Byte) eingestellt.

Zahlenangaben dürfen wie unter Basic üblich in dezimaler, hexadezimaler (&H) oder oktaler (&O) Form erfolgen. Ein Beispiel für den Aufruf des Basic-Interpreters mit Parametern:

```
gwbasic test >out.dat /S:256 /D
```

Hierdurch wird GW-Basic aufgerufen, wobei das Programm »test.bas« sofort geladen und ausgeführt wird. Alle Print-Ausgaben des Programms werden in die Datei »Out.dat« umgeleitet. Die maximale Record-Länge für Random-Dateien wird auf 256 Byte festgelegt und die Berechnung aller mathematischer Funktionen erfolgt mit doppelter Genauigkeit.

(Jochen Kruse/ev)

Caps-Lock per Programm

Bei einigen Programmen kann es wünschenswert sein, den Caps-Lock- oder den Num-Lock-Status der Tastatur per Software zu verändern. Die beiden Prozeduren »CapsLock« und »NumLock« im Demo-Programm nach Listing 3

```
0: program TastaturStatus;
1:
2: const
3:   ein = true;
4:   aus = false;
5:
6: var Zeile: string[128];
7:
8: procedure CapsLock(Status: boolean);
9: begin
10:  if Status then
11:    mem[0:$417] := mem[0:$417] or $40
12:  else
13:    mem[0:$417] := mem[0:$417] and $BF;
14: end;
15:
16: procedure NumLock(Status: boolean);
17: begin
18:  if Status then
19:    mem[0:$417] := mem[0:$417] or $20
20:  else
21:    mem[0:$417] := mem[0:$417] and $DF;
22: end;
23:
24: begin
25:  writeln('Bitte geben Sie etwas ein...');
26:  CapsLock(ein); NumLock(ein);
27:  readln(Zeile);
28:  CapsLock(aus); NumLock(aus);
29:  writeln(Zeile);
30: end.
```

Listing 3. Sondertasten einfach abgefragt

leisten dies. Der Hintergrund beider Prozeduren ist die Tatsache, daß die Speicherzelle \$417 im Segment 0 vom BIOS als Merker für den Tastaturzustand benutzt wird. Bit 6 dieser Speicherzelle entscheidet über den Caps-Lock-Status, Bit 5 über den Num-Lock-Status.

(Michael Obrut/ev)

Basic-Datensegment feststellen

Mit dem Basic-Befehl »def seg« wird unter Basica und GW-Basic das Datensegment für Peek, Poke, Call und Usgesetzt. Die genaue Syntax ist »def seg = <Adresse>«, wobei <Adresse> eine beliebige Segment-Adresse, also eine Integer-Zahl oder -Variable darstellt. Durch Aufruf der Anweisung »def seg« ohne Zahlenangabe und ohne Gleichheitszeichen wird das Datensegment wieder auf den ursprünglichen Wert zurückgesetzt. Dieser Wert entspricht dem Standard-Datensegment von Basic, in dem auch das Basic-Programm selbst und alle Variablen untergebracht sind. Wenn man jedoch die Adresse dieses Datensegmentes erfahren möchte, muß man zu einem Trick greifen. Basic merkt sich den aktuellen Wert des Datensegmentes im Speicherwort an der Speicheradresse »0000:0510« (hexadezimal). Das Basic-Programm in Listing 4 liest diesen Wert und gibt ihn in hexadezimaler Schreibweise am Bildschirm aus.

```
10 DEF SEG = 0
20 DS=PEEK(&H510)+256*PEEK(&H511)
30 PRINT "Datensegment = &H"+HEX$(DS)
40 DEF SEG
```

Listing 4. Basic-Datensegment feststellen

Der von diesem Programm angezeigte Wert ist wohlge-merkt der Standardwert des Datensegments, nicht der gerade aktuell durch »def seg« eingestellte Wert. Dieser Wert wird von Basic als Grundlage für alle Operationen mit Programm- und Variablendaten benutzt. Ändern Sie daher niemals diese Speicherzellen etwa durch Poke-Befehle, denn das würde unweigerlich zum »Absturz« des Basic-Interpreters führen.

(Wolfgang Kraul/ev)

Grafik-Hardcopy

Viele PC-Anwender verzichten aus Unkenntnis oder Bequemlichkeit darauf, die Hardcopy-Funktion Ihres PC (Taste [PrtSc] oder [Druck]) auf Grafikdruck vorzubereiten. Dabei ist die Sache ganz einfach: Sie müssen nur ein einziges Mal den externen Befehl »Graphics« von Ihrer DOS-Diskette aufrufen. Dadurch wird ein zusätzliches Treiberprogramm für Grafikdruck resident geladen. Lassen Sie sich nicht durch die Aussage in den Handbüchern abschrecken, daß Grafikdruck mit dieser Funktion nur mit IBM-Druckern möglich ist. Dies ist nichts als eine Legende. Tatsächlich verhält es sich so, daß der sogenannte IBM-Grafikdrucker praktisch vollständig Epson-kompatibel ist. Infolgedessen funktioniert das Graphics-Kommando mit allen Epson-kompatiblen Druckern (genauer: mit allen Druckern, die über ESC/P-Sequenzen gesteuert werden). Dies sind fast alle Matrixdrucker für den PC. Ein Versuch lohnt sich also in jedem Falle. Sie können dabei weder Ihren Drucker noch Ihren PC beschädigen. Schlimmstenfalls erhalten Sie kein vernünftiges Ergebnis beim Versuch, einen Grafikbildschirm als Hardcopy zu drucken.

Der Graphics-Befehl wird aufgerufen in der Form
graphics (Typ) (/R) (/B) (/F) (/C)

Die Typ-Angabe bezieht sich dabei auf den verwendeten Drucker. Folgende Angaben sind hier zulässig:

Color1: Farbdrucker mit schwarzem Farbband

Color4: Farbdrucker mit RGB-Farbband (rot, grün, blau, schwarz)

Color8: Farbdrucker mit CMY-Farbband (cyan, magenta, gelb, schwarz)

Compact: IBM-Kompaktdrucker

Colorjet: IBM-Tintenstrahlfarbdrucker

Graphics: Grafikdrucker

Wird keine Typ-Angabe gemacht, dann entspricht das dem Graphics-Typ. Dies ist der Normalfall und funktioniert mit den meisten Matrixdruckern zum PC.

Die vier möglichen Optionen zum Graphics-Befehl sind
/R: Der Hintergrund wird dunkel, Schrift und Grafik hell gedruckt. Der Ausdruck entspricht damit weitgehend dem Monitorbild, während normalerweise invertiert wird, daß heißt, die Schrift erscheint ohne diese Option dunkel, während der Hintergrund hell bleibt.

/B: Wenn diese Option angegeben ist, wird der Hintergrund farbig gedruckt, sonst bleibt er weiß. Diese Option gilt nur für die Druckertypen Color4 und Color8.

/F: Der Bildschirmausdruck wird um 90 Grad gedreht. Diese Option hat nur bei einer Grafik-Auflösung von 320x200 Punkten eine Wirkung. Im Hires-Modus (640x200 Punkte) wird der Ausdruck immer gedreht.

/C: Die Hardcopy wird zentriert, erscheint also genau in der Mitte der Druckseite.

Der Graphics-Befehl verringert den zur Verfügung stehenden Speicherplatz um einen gewissen Anteil, da der Grafiktreiber für die Hardcopy-Funktion resident geladen wird, also ständig im Speicher verbleibt. Um sich die Eingabe von »Graphics« zu sparen, integrieren Sie den Befehl am besten gleich in die Autoexec-Datei Ihrer Boot-Diskette. Dies kann mit jedem Texteditor im Programtext-Modus (ASCII-Modus) geschehen. Bis zum Ausschalten des Rechners oder einem Neustart steht Ihnen dann ständig eine Grafik-Hardcopy-Funktion über die PrtSc-Taste (Druck-Taste bei einigen Computern) zur Verfügung.

(ev)

Basic-Tricks mit dem Shell-Befehl

Mit dem Basic-Befehl »Shell« wird der Kommandoprozessor »command.com« aus dem laufenden Basic-Programm heraus aufgerufen. Damit werden einige Tricks möglich, die in reinem Basic nur schwer zu realisieren wären. Basica und GW-Basic bieten beispielsweise einen Chdir-Befehl zum wechseln in ein anderes Unterverzeichnis

(Subdirectory), aber wie wechselt man auf ein anderes Laufwerk? »chdir« verweigert jedenfalls die Mitarbeit, wenn als Parameter ein Laufwerksname angegeben wird. Hier hilft der Shell-Befehl weiter. Das folgende Basic-Kommando macht Laufwerk B zum aktiven Laufwerk:
shell "b:"

Ein weiteres Problem ist das Einlesen eines Disketten-Inhaltsverzeichnisses in eine Basic-Datei oder in ein String-Feld. Eine einfache Lösung arbeitet wieder mit dem Shell-Befehl:

shell "dir b: > dir.dat"

Diese Zeile ruft den Kommandoprozessor auf, um ein Directory von Laufwerk B aufzulisten. Die Ausgabe wird allerdings in die Datei »dir.dat« umgeleitet. Diese Datei kann man nun anschließend ganz einfach als Textdatei öffnen, lesen und bearbeiten. Listing 5 zeigt ein kleines Programmbeispiel hierzu.

```
10 REM Directory in Datei lesen
20 REM
30 SHELL "DIR B: >DIR.DAT"
40 REM
50 REM Das Directory befindet sich
60 REM nun in der Datei DIR.DAT und
70 REM kann bequem weiterverarbeitet
80 REM werden:
90 REM
100 OPEN "B:DIR.DAT" FOR INPUT AS #1
110 WHILE NOT EOF(1)
120   LINE INPUT #1,DATEI$
130   PRINT DATEI$
140 WEND
150 CLOSE 1
```

Listing 5. Directory lesen mit GW-Basic

Auch das Sortieren von Textdateien ist problemlos möglich. Statt lange Sortier-Routinen zu schreiben, sortieren Sie Ihre Textdatei doch einfach über das entsprechende DOS-Kommando:

shell "sort <dir.dat > dir.srt"

Dieser Befehl sortiert die Datei »dir.dat« alphabetisch in die Datei »dir.srt«. Natürlich muß für die Ausführung dieses Kommandos der externe DOS-Befehl »sort.com« auf dem aktuellen Laufwerk vorhanden sein. In ähnlicher Weise lassen sich auch viele andere DOS-Befehle aus Basic heraus vorteilhaft nutzen. Sicher finden Sie noch eine Menge weiterer Beispiele dafür.

(ev)

Schnelle Integer-Division

Turbo Pascal benutzt den Operator »div« für die Integer-Division. Wenn es aber um die Division durch eine Zweierpotenz geht (also etwa durch 2, 4, 8 oder 16) und es zusätzlich auf Geschwindigkeit ankommt, dann läßt sich mit Vorteil der »shr«-Operator (Shift right) von Turbo verwenden. Dieser an die Maschinensprache angelehnte Operator verschiebt das Bitmuster eines Integer-Wertes um eine oder mehrere Binärstellen nach rechts. Eine einmalige Rechtsverschiebung im Binärsystem entspricht aber gerade einer Division durch 2, eine zweimalige Verschiebung einer Division durch 4, und so fort. »shr« ist im Durchschnitt mehr als dreimal so schnell wie »div«. Die beiden folgenden Pascal-Zeilen sind ein Beispiel für die praktische Anwendung und liefern beide das gleiche Ergebnis:

a := b div 2; { langsame Version }

a := b shr 1; { schnelle Version }

Ebenso kann man natürlich auch die Integer-Multiplikation beschleunigen. Statt einer Multiplikation mit 2, 4, 8, etc. läßt sich der Operator »shl« (Shift left) verwenden. Dieser Operator wird anstelle von »shr« eingesetzt, verschiebt aber das Bitmuster eines Integer-Wertes nach links statt nach rechts. Jedes Linksverschieben einer Binärzahl entspricht aber gerade einer Multiplikation mit 2. Der »shl«-Operator ist etwa doppelt so schnell wie eine Integer-Multiplikation.

(Jochen Kruse/ev)

Impressum

Herausgeber
Christian Widuch

Chefredaktion
Andreas Hagedorn (hg), Volker Everts (ev)

Redaktion
Martin Aschoff (ma), Christian Spitzner (sp)

Schlußredaktion
Eva-Maria Hierlmeier

Freie Mitarbeiter
Martin Althaus, J. Bauer, Jürgen Beetz, Michael Bormann, Frank Brendle, Rainer W. Gerling, Martin Hepp, Alfred Kurz, Thomas Little, Andreas Mauerbrecher, Wilfried Radigk, Christian Rogge, Markus Zietlow, Uwe Damken, Jan Kayser, Hans-Joachim Kellertat, Wolfgang Kraul, Jochen Kruse, Michael Obrut, Hans-Jürgen Selters

Koordination
Stefan Ritter

Produktionsleitung
Gerd Köberich

Satz
Claudia Küllmer, Silvia Führer, Martina Siebert
Gabriela Joseph

Lektorat
Elvira Domonkos

Gestaltung
Yvonne Hendricks, Petra Biehl, Manuela Eska

Reprografie
Margarete Schenk, Helmut Skoupy

Illustration
Heinrich Stiller

Fotografie
Gerd Köberich, Christian Heckmann

Anzeigenleitung
Wolfgang Schnell

Anzeigenverkauf
Wolfgang Brill

Anschrift Redaktion:
Redaktion DOS International,
Elfenstr. 40, 8000 München 83
Tel. (089) 60 20 66

Anschrift Verlag:
DMV - Daten & Medien Verlagsges. mbH
Postfach 250, Fuldaer Straße 6
3440 Eschwege
Telefon: (0 56 51) 87 02
Telex 993 210 dmv d

Vertrieb
Verlagsunion
Friedrich-Bergius-Straße 20
6200 Wiesbaden

Druck
Druckerei Jungfer, 3420 Herzberg

Bezugspreise
DOS EXTRA erscheint unregelmäßig.
Einzelpreis 18,- DM, 145 Ös, 18,- sfr.

Bankverbindungen:
Postcheck Frankfurt/M: Kto.-Nr.: 23043-608
Raiffeisenbank Eschwege:
BLZ: 522 603 85, Kto.-Nr.: 245 7008

Manuskripte, Programme:
bitte an die Redaktion DOS International,
Elfenstr. 40, 8000 München 83 senden.

Für unverlangt eingesandte Manuskripte und Datenträger sowie Fotos übernimmt der Verlag und die Redaktion keine Haftung. Die Zustimmung zum Abdruck wird vorausgesetzt.
Das Urheberrecht für veröffentlichte Manuskripte liegt ausschließlich beim Verlag. Nachdruck sowie Vervielfältigung oder sonstige Verwertung von Texten, nur mit schriftlicher Genehmigung des Verlages.
Namentlich gekennzeichnete Fremdbeiträge geben nicht in jedem Fall die Meinung der Redaktion wieder.

**Haben Sie Anregungen,
Vorschläge oder sogar
Manuskripte für unser nächstes
DOS EXTRA
oder die monatliche Ausgabe
von DOS International?**

Dann schreiben Sie uns!

Die Anschrift der Redaktion:

**Redaktion
DOS International
Elfenstr. 40
8000 München 83**

Die Inserenten DOS EXTRA 1'87/88

A.B.C. Computer.....	49
A.S.S. Ware.....	37
Byte Me.....	95
CCP-Software.....	29
DMV.....	13,85,99,146,147,171
Kirschbaum.....	47
Kulenkampff & Konitzky.....	101
Prosoft.....	71
Rose.....	37
SOS Software-Service.....	55
SPI.....	61
Schneider Data.....	2
Star Division.....	16,17,172
TG-Soft.....	108
Vehrenberg.....	45
Vortex.....	23
Weeske.....	33

DAS GRAFIKWUNDER CGX FÜR DEN PC!

CGX (Color Graphik Extension) ist eine Turbo Pascal-Toolbox der Superlative: Eine Sammlung von 100 leistungsfähigen Prozeduren, mit denen sowohl der Heimprogrammierer als auch der Profi grafisch beeindruckende und benutzerfreundliche Turbo Pascal-Programme schreiben kann, ohne auf Spezialkenntnisse oder zeitraubende Assemblerprogrammierung angewiesen zu sein.

Mit CGX können aus ein und dem gleichen Quelltext lauffähige Programme für 3 hochauflösende PC-Grafikmodi erzeugt werden:

- Schneider PC 1512 Farbgrafik
(640x200 bei 16 Farben)
- EGA Farbgrafik (z.B. Schneider PC 1640, Atari)
(640x200 bei 16 Farben)
- CGA Hires-Grafik
(640x200 bei 2 Farben)

Damit ist es ohne weiteres möglich, auf dem Schneider PC 1512 Software für EGA-Rechner zu entwickeln – und umgekehrt!

Die CGX-Prozeduren arbeiten sehr schnell und effektiv, da sie unter Verwendung modernster Grafikalgorithmen zu einem großen Teil in kompakten 8086 Inline-Assemblercode realisiert wurden. Sie sind in mehreren Bibliotheken zusammengefaßt, die als Include-Files problemlos in eigenen Programmen verwendet werden können:

Die **KERNEL-Bibliothek** stellt den CGX-Betriebssystem-Kern dar und enthält u. a. einen neuen Bildschirmtreiber, der verschiedene Schrifttypen und -größen zur Verfügung stellt, weiterhin Unterstreichen, Fettschrift, Exponenten und Indizes, sowie die Textausgabe auf Grafikkoordinaten, Transparent- und XOR-Modus und vieles mehr.

Die **GRAPHIK-Bibliothek** enthält zahlreiche Grafikprozeduren, angefangen bei »Plot« und »Draw« bis hin zu Ellipsenbögen und Rechtecken mit runden Ecken. Weiterhin finden Sie hier eine ganze Palette von Füllprozeduren für Muster und Farben sowie Prozeduren zum Verzerren bzw. Biegen von Bildschirmbereichen.

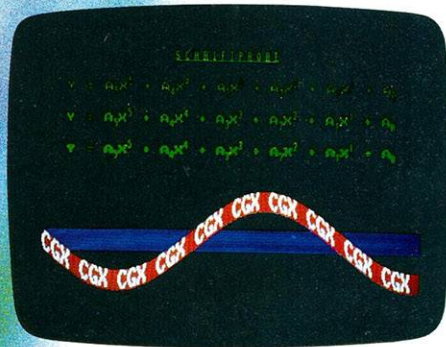
Die **MOUSE-Bibliothek** stellt die Schnittstelle zum Maustreiber dar und erlaubt neben der Abfrage und Kontrolle der Mausposition und -tasten auf komfortable Weise die Gestaltung eigener Mauszeiger. Zusätzlich gibt sie Hilfen zur »Event«-Verarbeitung: »Mausereignisse« können definiert und abgefragt werden.

Die **HARDCOPY-Bibliothek** ermöglicht den Ausdruck beliebiger Bildschirmausschnitte in frei wählbarem Format, wobei die 16 Farben durch verschiedene Muster dargestellt werden.

Die **SCREEN-Bibliothek** enthält Prozeduren mit denen Bildschirmbereiche im Speicher oder auf Diskette abgelegt und bei Bedarf wieder geladen werden.

Die **WINDOW-Bibliothek** stellt ein System zur Verwaltung von Text- und Grafikfenstern zur Verfügung – die Grundlage für eine moderne Menuetechnik.

Weiterhin umfaßt CGX neben Anwendungsbeispielen und einem ausführlichen Handbuch kommentierte Quellprogramme für einen Füllmuster- und Zeichensatzeditor – eine komfortable Programmumgebung mit »PullDown«-Menues und Mausbedienung.



Neue Gestaltungsmöglichkeiten
– auch hier
überzeugt CGX

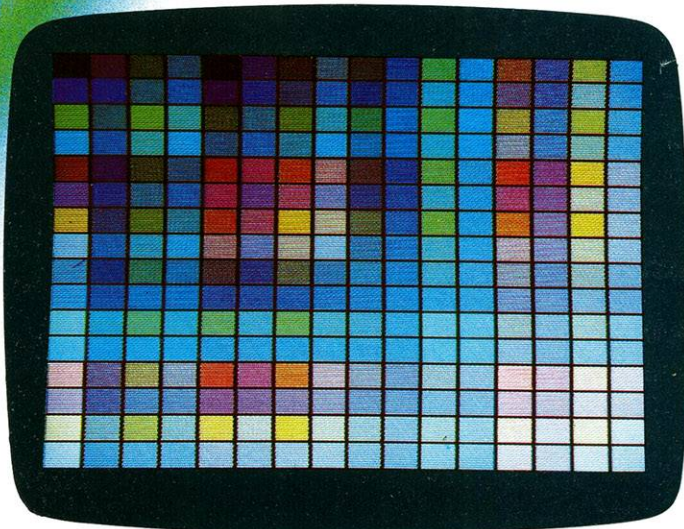
Autor: M. Uphoff

199,– DM

inkl. MwSt., zuzgl. 3,– DM Porto und Verpackung.

CGX für MS-DOS PC

Farbenvielfalt durch CGX – mehr als 100 Farben am PC!



zu beziehen
von:

DMV
Software

DMV Daten & Medien Verlagsgesellschaft mbH
Postfach 250
Fuldaer Straße 6 · 3440 Eschwege

Bitte Coupon auf Seite 99 benutzen

– Händleranfragen erwünscht –

Star-Planer^{PC}



Schwupp-di-wupp

Das deutsche,
professionelle
Kombiprogramm
für Tabellen-
kalkulation / Daten-
bank / Grafik zum
Überraschungspreis!

F Ü R I B M - K O M P A T I B L E C O M P U T E R

Star-Planer PC läuft unter PC-DOS/MS-DOS 2.11 (aufwärtskompatibel) auf jedem IBM-kompatiblen PC ab 256 kB RAM in Farbe und monochrom. Seinen hohen Komfort für den Anwender entwickelt Star-Planer PC bei vollem Speicherausbau und mit einem Grafiktreiber — Hercules, EGA oder CGA.

Vergleichen Sie gern die Leistung von Star-Planer PC auf Ihrem privaten System mit dem besten Programm, das an Ihrem beruflichen Arbeitsplatz auf einem Professional-System verfügbar ist.

Tabellenkalkulation

Mit Star-Planer PC können Sie auch zuhause und unterwegs problemlos auf die Daten vieler anderer Programme zugreifen und sie weiterverarbeiten; denn Star-Planer PC ist Files- und Makro-kompatibel zu Lotus 1-2-3, Vers. 1.1A - dem Standard in Sachen Tabellenkalkulation. Damit gehen Sie auf Nummer Sicher! Und kompatibel heißt für STAR-DIVISION - die Macher von Star-Planer PC und anderen Software-Delikatessen - auf deutsch „zusammenpassend/vereinbar“.

Star-Planer PC besitzt den Standard internationaler Spitzenprodukte — einfach top. Star-Planer PC ist die deutsche Tabellenkalkulation mit rechnender Grafik-Simultan-Anzeige und Datenbank.

Grafik

Alles*), was Sie über und mit professioneller Tabellenkalkulation gelernt haben, können Sie bei Star-Planer PC anwenden: Das Arbeitsblatt eröffnen, einfügen, löschen, radieren, benennen, fensterln, formatieren, justieren, neu-berechnen, kopieren, versetzen, transferieren, grafisch aufbereiten, verwalten, suchen, finden, analysieren, zurücksetzen. . . Alles, was Ihnen fremd geblieben ist, wird mit der deutschen Benutzerführung und einem phantastisch guten Handbuch in deutscher Sprache zur schnellen Routine.

Datenbank

*) Fast alles! Die Faxen mit Kopierschutz und Original-Startdiskette dürfen Sie getrost vergessen. Sie starten ab Floppy oder Festplatte — schwupp-di-wupp!

Star-Planer PC ist das Tabellenkalkulationsprogramm für Leute, die auch dann rechnen können, wenn's um den Anschaffungspreis für Spitzensoftware geht. Und für die Preisdifferenz zwischen Star-Planer PC und in der Leistung vergleichbarer Software erwerben sie eine Festplatte zum superschnellen Arbeiten — oder fliegen in den fälligen Urlaub. Aber verraten Sie Ihrem Chef nicht, für welchen Preis Sie Star-Planer PC erworben haben, falls er Ihnen die Heimarbeitsstunden vergütet. . .

Raubkopieren lohnt nicht — denn Star-Planer PC kostet Sie lediglich ein Lächeln und nur

DM 298,-*

* Unverbindliche Preisempfehlung



I N F O - C O U P O N



Star-Planer

Logo — ich will mehr: Informationen über Star-Planer PC und die weiteren PC-Software-Angebote von **STAR-DIVISION**



Name, Vorname

Straße, Nr.

PLZ, Ort

Bitte ausschneiden, auf eine Postkarte kleben oder in einen Briefumschlag stecken und an STAR-DIVISION GmbH, Postfach 2830 in 2120 Lüneburg abschicken

STAR-DIVISION

POSTFACH 2830 • 2120 Lüneburg • Tel.: (041 31) 70090

Hinweis: Lotus 1-2-3 ist eingetragenes Warenzeichen der Lotus-Development Corp., USA; MS-DOS der Microsoft Corporation, USA; PC-DOS der IBM Corp., USA.