

DOS EXTRA

Sonderausgabe DOS EXTRA

Nr. **21** 1992

ISSN 0934-2842
DMV-Verlag
Eine Publikation der
DOS INTERNATIONAL

zum Abtippen:

DOS-Makro- Assembler SAM 4.0

Referenz:

- ◆ alle Prozessorbefehle
- ◆ alle BIOS-Interrupts
- ◆ MS-DOS-Funktionen durchleuchtet

zum Mitmachen:

Schnelleinstieg in Assembler

Hardware im Griff:

Gerätetreiber zum Selbermachen:
Programmieren Sie Ihren
Laserdrucker

Werkstatt:

Interrupts kontrollieren
mit QBasic

**Tips & Tricks zu
Assembler und Interrupts**



Die Quelle Ihrer Kompetenz

Was Sie schon immer über Windows wissen wollten:

Zuverlässige Informationen von A – wie Aktuelles bis Z – wie Zubehör bietet Ihnen jetzt WinDOS, das monatliche Windows-Magazin der DOS International für den kompetenten Windows-Anwender.

Ab sofort im Handel!



Von Anfang an dabei

Totgesagte leben länger. Dieser Spruch hat mit der Geschichte der Maschinensprache und damit der Assembler-Sprache eine weitere Bestätigung gefunden. Denn nach der Evolution der Hochsprachen wie etwa Pascal oder C/C++ wurde die Verwendung von Maschinensprache zuerst stark verdrängt, um im Zeitalter noch schnellerer Rechner wie ein Phönix aus der Asche eine Auferstehung zu feiern. Wie sieht die Geschichte einer Programmiersprache aus, die ihre Bedeutung länger als alle anderen Sprachen bis heute behaupten konnte?

Die Urform begann mit Zuses erstem Rechner in den vierziger Jahren. Die Programmierung dieser Maschine hatte aber nur in entferntester Weise mit der späteren Form der Programmierung zu tun. Sie ähnelte eher der Bedienung eines Rechenschiebers in abgewandelter Form.

Erst mit der Entwicklung effizienterer Rechner, die in der Industrie einsetzbar waren, wurde das Verschieben und Setzen der kleinsten Informationseinheit, des Bits, mit Assembler-Befehlen symbolisiert, um die Programmierung dieser Maschinen leichter zu handhaben. Diese Symbolisierung der einzelnen Prozessoraktionen wurde daraufhin in allen folgenden Prozessor-typen übernommen.

Vom legendären Apple II und der leider zum Flop degradierten Apple-Maschine Lisa über die ersten Atari-Rechner und den ebenfalls zur Legende gewordenen C-64 bis hin zur Entwicklung und Evolution der PC-Prozessoren 80x86 haben alle Rechnertypen eine Gemeinsamkeit: die Prozessoren lassen sich durch Assembler in Aktion setzen. Natürlich gibt es für jeden Prozessor-typ eine andere Syntax, die im unterschiedlichen Aufbau der einzelnen Prozessoren begründet ist. Doch das Prinzip der Assembler-Sprache als symbolisierte Maschinensprache bleibt bei allen Prozessoren erhalten.

Leider besitzt Assembler neben den Vorteilen wie der genauen Hardwarekontrolle sowie der Produktion kleiner, effizienter Codes den Nachteil, daß der Programmierer auch die gesamte Verantwortung über alle Abläufe innerhalb der Hardware in seinen Händen hält.

Aus diesem Grund wurden Programmiersprachen wie Fortran (1957), Lisp (1960), PL/I (1964), Pascal (1971), Basic (1972) und C (1978) entwickelt. Sprachen wie Pascal und C konnten dabei ihren Siegeszug als professionelle Sprachen durch Produkte wie Borland-Pascal oder Borland-C/C++ und Microsoft-C antreten. Damit, so meinte man, wäre der Tod von Assembler im wahrsten Sinne des Wortes »vorprogrammiert«.

Doch gerade Profiprogrammierer merkten schnell, daß die Entwicklung effizienter Betriebssysteme und Compiler ohne Assembler undenkbar war. Selbst Softwareentwickler wollten die Kontrolle der Hardware lieber direkt über Assembler realisieren, um Programme klein und schnell zu halten. Kein Wunder also, daß aktuelle Produkte wie OS/2, Windows NT und Compiler nicht ohne einen Assembler-Kern auskommen.

Einziger Makel der Assembler-Sprache ist ihr schlechter Ruf. Viele Programmierer behaupten, der Oldie wäre unübersichtlich

und viel zu kompliziert. Das widerlegt diese Ausgabe der DOS Extra. Gehen Sie mit uns auf die Reise durch die wunderbare Welt der Assembler-Sprache und Interrupts. Damit Sie die Früchte Ihrer Arbeit ernten können, bieten wir Ihnen die neue Version 4.0 des DOS-Makro-Assemblers SAM an.

Viel Spaß wünschen

Ken Muhlman

R. Neal Stutz

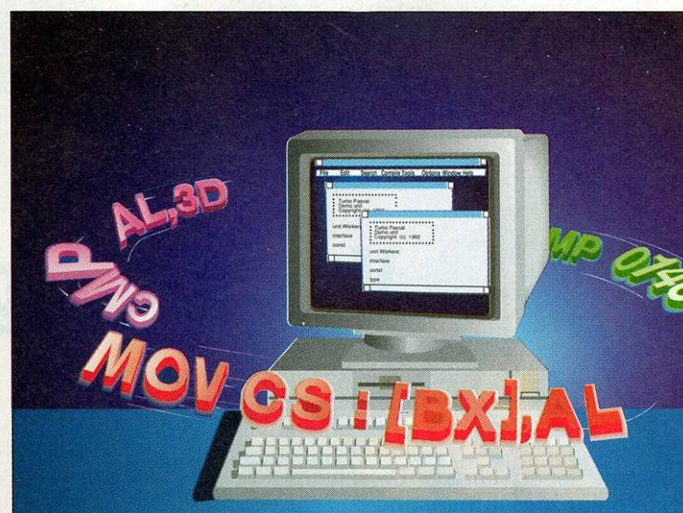


INHALT

DOS EXTRA
21/92

GRUNDLAGEN

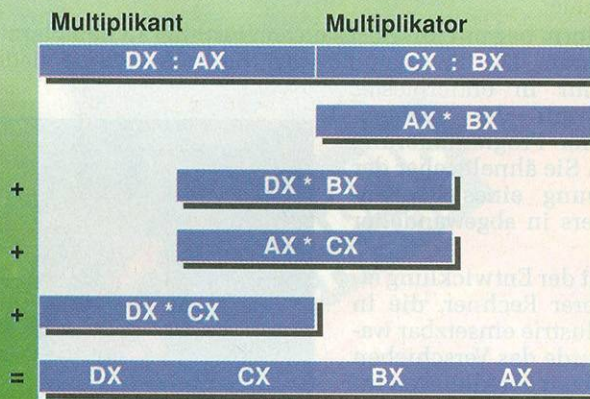
- Assembler für Einsteiger** 6
Am Anfang war das Bit
- Assembler und Mathematik** 24
Der Rechenschieber
- Assembler und Hardware** 32
Musik liegt in der Luft
- Assembler zum Abtippen** 48
Prozessor-Tuning mit SAM 4.0
- DOSsier Assembler-Befehle** 70
Das Ende von Babylon
- Befehlsreferenz** 76
Die Befehle der Coprozessoren



Viele Programmierer halten Assembler für eine der schwierigsten Programmiersprachen. Dadurch fühlen Einsteiger sich häufig verunsichert. Einen leichten Einstieg in die Assemblersprache bieten wir Ihnen ab

Seite 6

Vorzeichenlose 32-Bit-Multiplikation

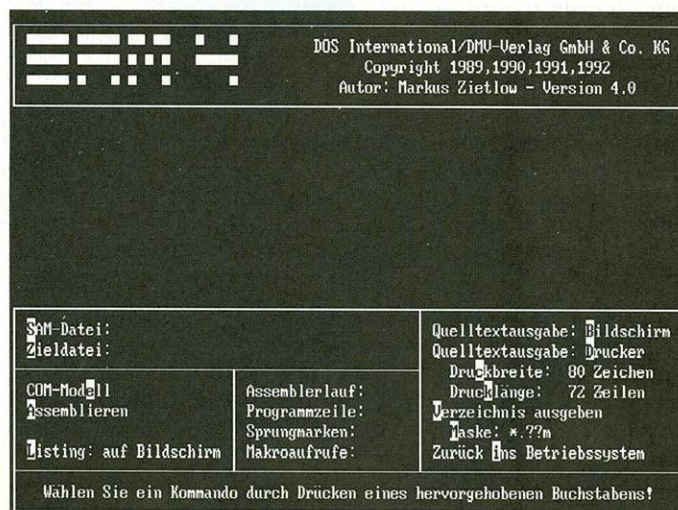


Mathematische Operationen unterliegen bestimmten Grenzen im Wertebereich, wenn sie in Hochsprachen verwirklicht werden. Assembler kennt diese Grenzen überhaupt nicht. Wie Sie diese Freiheit nutzen, lesen Sie ab

Seite 24

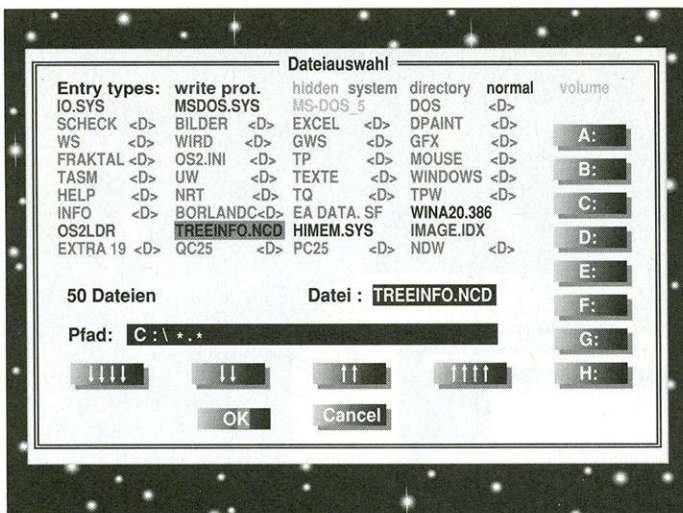
ASSEMBLER IN DER PRAXIS

- Dateiauswahlbox im Eigenbau** 78
Die Qual der Wahl
- Grundlagen** 90
Eine Rallye durch das Segment
- Interrupt 15hex** 93
Aus alt mach neu
- BIOS-Interrupts** 95
Interrupts bis zum Abwinken
- Werkstatt A86-Assembler** 105
Licht im Speicher



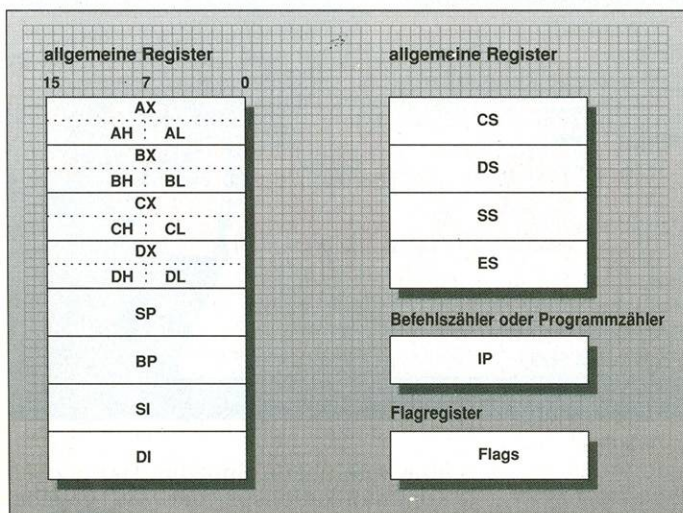
SAM 4.0 ist die Weiterentwicklung des bekannten Makro-Assemblers der DOS International. Gerade für den Einstieg lohnt es sich, SAM einzusetzen. Den Makro-Assembler SAM in der Version 4.0 zum Abtippen finden Sie ab

Seite 48



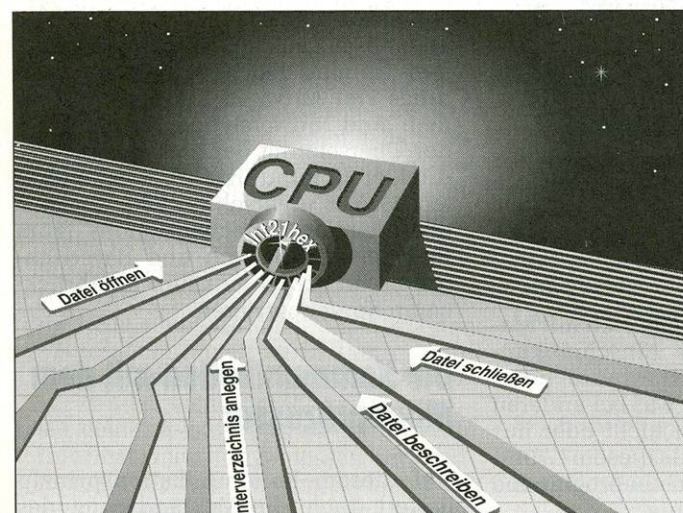
Überall dort, wo Sie auf Dateien zugreifen müssen, ist Ihnen eine Dateiauswahlbox eine große Hilfe. In Assembler programmiert bietet diese außerdem den Vorteil des geringen Speicherverbrauchs. Solch eine Box erhalten Sie ab

Seite 78



Nicht viele Programmierer wissen, was sich hinter einem Programm-Segment-Präfix (kurz: PSP) verbirgt. Daß es sich dabei um eine sehr interessante und leicht verständliche Struktur handelt, zeigen wir Ihnen ab

Seite 90



Der Interrupt 21hex ist einer der zentralen Interrupts von MS-DOS. Er besteht aus einer großen Sammlung wichtiger Funktionen und Unterfunktionen. Welche Funktionen in diesem Interrupt enthalten sind, erfahren Sie ab

Seite 125

TIPS & TRICKS

Tips & Tricks für Assembler

Schutz vor Computerviren und anderen Sabotageprogrammen	111
Paßwort in der »config.sys«	116
Ergonomische Bildfrequenz auch bei 480 Zeilen	117
Sichern des CMOS-RAM	118
Systembeschleunigung	122
Nicht bereit beim Lesen	122
Keine Chance den Boot-Sektor-Viren auf Floppies	123
Paßwortabfrage in COM-Dateien	124

— SYSTEMPROGRAMMIERUNG —

Interrupt-Funktionen	125
Ein Interrupt für alle Fälle	
Interrupt-Funktionen	134
Mit Xenix kompatibel	
Gerätetreiber	144
Individuelle Treiber	
Werkstatt QBasic	151
Interrupts unter QBasic	
DOSsier PCL-Referenz	157
Hilfe für die Laserprogrammierung	
Drucker-Programmierung	162
Vom Laser aufs Papier	

RUBRIKEN

Editorial	3
Von Anfang an dabei	
Impressum	166



Assembler für Einsteiger

Am Anfang war das Bit

Jeder Einsteiger oder Fortgeschrittene träumt davon, seinen PC bis ins kleinste Bit auszunutzen, indem er dem Mikroprozessor direkt Befehle erteilt. Das geht nur über die Maschinensprache. Doch vor den Erfolg haben die Götter den Schweiß gesetzt.

Um die Maschinensprache sinnvoll einzusetzen, sollten Sie über die Systemstrukturen des Computers Bescheid wissen. Dieser Beitrag bietet die notwendigen Grundlagen und nutzt den vorgehend vorgestellten Assembler SAM 4, der Sie in die Lage versetzt, die erworbenen Kenntnisse sofort in die Praxis umzusetzen.

Maschinensprache ist die schnellste aller Programmiersprachen, weil der Computer sie versteht, ohne sie umset-

zen zu müssen. Auch lässt sich mit Maschinensprache ohne Einschränkungen direkt auf die Hardware eines Computers zugreifen, was bei anderen Programmiersprachen nicht oder nur bedingt der Fall ist. Aus diesem Grund sind Maschinenspracheprogramme besonders leistungsfähig.

Menschen verständigen sich auf die gleiche Weise mit Computern, wie sie untereinander kommunizieren: durch

eine vereinbarte Sprache. Wenn zwei Kommunikationspartner die gleiche Sprache benutzen, so stellt die Verständigung kein Problem dar. Sprechen sie aber unterschiedliche Sprachen, so benötigen sie einen Dolmetscher, der die Worte des einen Gesprächspartners in die Sprache des anderen übersetzt. Das Gespräch über den Dolmetscher beansprucht allerdings viel Aufwand und Zeit. Außerdem sind Mißverständnisse aufgrund der Übersetzung denkbar, die bei der direkten Kommunikation nicht auftreten. Selbst unterschiedliche Dialekte des gleichen Sprachenstamms (zum Beispiel Plattdeutsch und Bayerisch) führen oft schon zu kaum überbrückbaren Sprachbarrieren.

Die Kommunikation mit dem Mikroprozessor

Auf diesem Kommunikationsprinzip basiert auch die Verständigung des Programmiers mit dem Computer. Die Mehrheit der Programmierer verwendet eine höhere Programmiersprache wie Basic, Pascal oder C. Jede die-

ser Hochsprachen übersetzt das als Quelltext vorliegende Programm in einen Code, den der Computer unmittelbar verstehen und ausführen kann: die Maschinensprache.

Bei den höheren Programmiersprachen gibt es zwei verschiedene Verfahren, Hochsprachenbefehle in Maschinensprache zu übersetzen: Interpreter und Compiler.

● **Der Interpreter:** Ein Interpreter übersetzt ein Hochsprachenprogramm während der Ausführung Befehl für Befehl in Maschinensprache. Weil jedoch beim Ausführen eines Befehls neben der Übersetzung und den gesamten Fehlerabfingroutinen auch die vom jeweiligen Kommando benötigten Systemroutinen aufgerufen werden müssen, ist die erreichbare Verarbeitungsgeschwindigkeit in den meisten Fällen vergleichsweise gering.

● **Der Compiler:** Ein Compiler hingegen übersetzt einen Quelltext nach erfolgreicher Fehlerbeseitigung in einen maschinenspracheähnlichen Code (Objektcode) oder sogar direkt in Maschinensprache. Das in dieser Weise übersetzte Programm erreicht natürlich eine wesentlich höhere Geschwindigkeit als ein Interpreter.

● **Die Maschinensprache:** Ein Maschinenspracheprogramm besteht aus einzelnen Maschinensprachebefehlen, die der Prozessor des Computers direkt ausführen kann. In einem PC arbeitet der Intel-Prozessor 8086 oder 8088 und in einem AT der abwärtskompatible 80286-, 80386- oder 80486-Prozessor, der sich im vom MS-DOS gewählten Modus wie ein 8086 verhält.

Der 8088 verfügt über einen 8-Bit-Datenbus, der 8086 über einen 16 Bit breiten Datenbus. Dies vereinfacht zwar den elektrischen Aufbau eines PCs, doch ist der 8088 weniger leistungsfähig, weil er 16-Bit-Daten nicht parallel übertragen kann. Diese müssen hierbei in zwei 8-Bit-Einheiten hintereinander übertragen werden.

Der 8088 besitzt nur einen 4-Byte-Befehlspuffer (Instruction Fetch Queue) für Maschinensprachebefehle, während der Befehlspuffer beim 8086 über sechs Byte verfügt, so daß dieser »voranschauender« arbeiten kann.

Um Ihnen den Einstieg zu erleichtern und Sie nicht durch die zahlreichen Prozessorvarianten zu irritieren, wird im folgenden der Begriff »8086/8088« verwendet, wenn der PC-Mikroprozessor gemeint ist. Wenn weiterführende Informationen angemessen erscheinen, finden Sie die vom Standard abweichenden Prozessorbezeichnungen im Text.

Der Befehlssatz des 8086/8088 umfaßt 120 Maschinensprachebefehle, die mit keinem, einem oder zwei Operanden

arbeiten. Jeder dieser Befehle wird in der vom Prozessor ausführbaren Form durch einen numerischen Wert – 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 Maschinenspracheprogrammen 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 Maschinensprachebefehle des 8086/8088 von ein bis sechs Byte.

Moderne Prozessoren weichen in diesen Konventionen von ihren Vorgängern ab. Dennoch sind sie alle in der Lage, mit den 8086/8088-Kommandos zu arbeiten.

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 sind zum Beispiel die beiden aufeinanderfolgenden Zahlen 210 und 0, die in zwei hintereinanderliegenden Speicherstellen abgelegt sind, gültige Maschinensprachebefehle. Sie weisen den Prozessor an, den Wert 0 in das Register AH zu laden.

Ein Register ist ein schneller Zwischenspeicher, der sich direkt im Prozessor befindet und damit einen zeit- aufwendigen Zugriff auf den Arbeitsspeicher überflüssig macht.

Assembler-Befehle ersetzen den Maschinencode

Weil die Zahlenkolonnen von längeren Maschinenspracheprogrammen auf die Dauer ziemlich unübersichtlich sind, hat sich Intel für seine Prozessoren anstelle der Zahlenfolgen sinnvolle Kurzbezeichnungen einfallen lassen. Diese bezeichnet man als Assembler-Befehle oder »Opcodes« (Operation Codes).

Um den 8086/8088 zu veranlassen, den Wert 0 in das AH-Register zu schreiben, gibt der Programmierer somit nicht mehr die Zahlen 210 und 0 an, sondern schreibt statt dessen

```
mov ah,0
```

Die Kurzbezeichnung »mov« steht für das englische Wort »move« (bewegen). Der Assembler-Befehl »move ah,0« veranlaßt also den Prozessor, den Wert 0 in das Register AH zu bewegen.

Sie wenden jetzt vielleicht ein, daß

durch die Assembler-Befehle doch wieder ein Übersetzungsvorgang nötig ist, nämlich von Assembler-Sprache in die Maschinensprache. Ist es auch, doch im Gegensatz zu Hochsprachenbefehlen lassen sich die Assembler-Befehle 1:1 in Maschinensprache übersetzen, so daß keine Übersetzungsverluste auftreten.

Ein Programm, das die symbolischen Assembler-Befehle in den numerischen Maschinencode übersetzt, heißt Assembler.

Die Assembler-Sprache – und damit auch die Maschinensprache – greift direkt auf den Speicher des Computers zu. Nun kann dieser Speicher nur zwischen zwei elektrischen Zuständen unterscheiden. 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 kommen Sie beim Erlernen der Assembler-Sprache nicht darum herum, sich mit einer vom Dezimalsystem abweichenden Methode der Zahlendarstellung zu beschäftigen.

234 im Binärsystem

```
234 = 128
+ 64
+ 32
+ 8
+ 2
= 1x27
+ 1x26
+ 1x25
+ 0x24
+ 1x23
+ 0x22
+ 1x21
+ 0x20
= 11101010bin
```

● **Das Dezimal- und das Binärsystem**
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 234 setzt sich beispielsweise wie folgt zusammen:

```
234 = 200
+ 30
+ 4
= 2x102
+ 3x101
+ 4x100
= 234
```

Das Binärsystem arbeitet nach dem gleichen Prinzip, doch können hier die Stellen einer Zahl nur zwei verschiedene Werte (0 und 1) annehmen. Jede Stelle stellt somit das Vielfache ei-

ner bestimmten Zweierpotenz dar. Die letzte Ziffer zeigt immer das Vielfache von 2^0 an, die vorletzte Ziffer das Vielfache von 2^1 und so weiter.

Ein Beispiel für die Dezimalzahl 234 sehen Sie im Textkasten »234 im Binärsystem«.

Zum besseren Verständnis sehen Sie die Zahlen 0 bis 10 als Binärzahlen im Kasten »Zahlen 0 bis 10 als Binärzahlen«.

Versuchen Sie einmal zur Probe, die Zahlen 5, 36 und 158 ins Binärsystem und die Zahlen 01001001, 10101010 und 10110111 ins Dezimalsystem umzuwandeln. Die Lösung finden Sie am Ende des Beitrags in dem letzten Textkasten. 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 234 mit seinem binären Gegenwert 11101010 vergleichen, stellen Sie fest, daß Binärwerte mit zunehmender Größe immer länger, also schnell unübersichtlich werden.

Abhilfe schaffen das Oktal- und das Hexadezimalsystem.

● **Das Oktalsystem:** Das Oktalsystem verbindet drei Binärziffern zu einer oktalen Ziffer. Jede Stelle einer Oktalzahl kann mit $2^3 = 8$ verschiedenen Werten von 0 bis 7 belegt sein. Die Dezimalzahl 185 erhält auf diese Weise die kompakte Zuordnung 271okt.

Um das Zahlensystem kennenzulernen, wandeln Sie die dezimalen Zahlen 18, 47 und 916 in oktale Werte um. Die Lösung finden Sie ebenfalls am Ende des Beitrags.

● **Das Hexadezimalsystem:** Mit Hexadezimalzahlen lassen sich binäre Werte bequem und platzsparend zusammenfassen. Es faßt vier Binärziffern zu einer hexadezimalen Ziffer zusammen, indem jede Stelle einer Hexadezimalzahl $2^4 = 16$ verschiedene Zustände (0 bis 9 und A bis F) annehmen kann. Dadurch werden 8-Bit-Werte mit zwei und 16-Bit-Werte mit nur vier hexadezimalen Stellen dargestellt. Die dezimale Zahl 234 wird beispielsweise zu dem kompakten Wert EAhex. Tabelle 1 zeigt eine Gegenüberstellung der 20 ersten Dezimal-, Binär-, Hexadezimal- und Oktalzahlen.

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.

Jetzt dürfte es Ihnen auch nicht mehr schwerfallen, die Dezimalzahlen 255, 616 und 7654 in ihr hexadezimalen Äquivalent umzurechnen. Die Lösung finden Sie auch in diesem Fall am Ende des Beitrags.

Der Aufbau des 8086/8088 bestimmt die Leistungsfähigkeit der Befehle sowie die Anzahl der Register und Adressierungsarten für Speicherzellen, die dem Programmierer beim Schreiben seiner Assembler-Programme zur Verfügung stehen.

Zahlen 0 bis 10 als 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

● **Datenregister:** Der 8086/8088 besitzt die acht 16-Bit-Datenregister AX, BX, CX, DX, SI, DI, SP und BP. Die Register AX bis DX können Sie zusätzlich in je zwei 8-Bit-Register AL und AH, BL und BH, CL und CH sowie DL und DH aufteilen.

● **Segmentregister:** Für die Segmentadressierung (eine besondere Form der Adressierung beim 8086/8088) stehen die vier 16-Bit-Segmentregister CS, DS, ES und SS bereit.

● **Statusregister:** Das 16-Bit-Statusregister PSW (Processor Status Word) zeigt den Zustand der Statusbits an, und der 16-Bit-Befehlszeiger (Instruction Pointer) deutet auf den nächsten auszuführenden Befehl im Speicher des Computers. Bild 1 zeigt den Registersatz des 8086/8088-Prozessors. Der Registersatz der aktuelleren Prozessorvarianten verfügt über einen erheblich

erweiterten Registersatz, der für eine 32-Bit-Programmierung unerlässlich ist. Ab dem 80386 arbeitet der Mikroprozessor mit dem Registersatz (Bild 2). Fast alle Prozessoren bearbeiten ein Programm, indem sie einen Befehl aus dem Speicher lesen und ihn ausführen. Daraufhin wird der nächste Befehl gelesen, ausgeführt und so weiter. Diese Vorgehensweise bedingt aber einige Verzögerungen, weil der Prozessor jedesmal warten muß, bis ein neuer Befehl aus dem Speicher gelesen wird, bevor er ihn dann ausführen kann.

Von Registersätzen und Befehlspuffern

● Der Befehlspuffer des 8086/8088:

Der 8086/8088 umgeht einen Teil dieses zeitaufwendigen Vorgangs, indem er den Aufgaben »Befehl lesen« und »Befehl ausführen« zwei separate Funktionseinheiten auf dem Baustein zuordnet.

Die erste Funktionseinheit hat ausschließlich die Aufgabe, Befehle aus dem Speicher zu lesen und sie in den bereits erwähnten Befehlspuffer zu schieben. Dort stehen diese Befehle für die zweite Funktionseinheit, die sie ausführt, zur Verfügung. Weil die erste Funktionseinheit immer aufeinander folgende Adressen liest, muß die zweite Funktionseinheit nur noch dann eingreifen und selbst einen Befehl lesen, wenn die Programmausführung durch einen Sprung an einer anderen Speicheradresse fortgeführt wird.

Bild 1 zeigt deutlich, daß alle Register des 8086/8088 16 Bit breit sind. Mit einem 16-Bit-Wert lassen sich jedoch nur $2^{16} = 65\,536$ Speicherstellen adres-

Drei Zahlensysteme in der Gegenüberstellung

Dezimal	Binär	Hexadezimal	Oktal
0	00000	00	00
1	00001	01	01
2	00010	02	02
3	00011	03	03
4	00100	04	04
5	00101	05	05
6	00110	06	06
7	00111	07	07
8	01000	08	10
9	01001	09	11
10	01010	0A	12
11	01011	0B	13
12	01100	0C	14
13	01101	0D	15
14	01110	0E	16
15	01111	0F	17
16	10000	10	20
17	10001	11	21
18	10010	12	22
19	10011	13	23

Tabelle 1. Die ersten 20 Zahlen des Dezimal-, Binär-, Hexadezimal- und Oktalsystems

Die Register der 8086-Familie

Bei 8088/8086/80286:



Hinweis: Der 80286 enthält außerdem die GDT-, IDT-, und TSS-Register (siehe 80386/80486-Register)

Der Registersatz des 8086/8088 /80286-Prozessors besteht aus vierzehn 16-Bit-Registern.

sieren (64 KByte). Dagegen beträgt der Speicherbereich des 8086/8088 aufgrund des 20 Bit breiten Datenbus 1 MByte. Wie adressiert nun der 8086/8088 aber den gesamten Speicherbereich?

Dazu benutzt der 8086/8088 die Segmentregister, die die Startadresse (Basisadresse) für verschiedene Speichersegmente enthalten. Die 16-Bit-Adressen der normalen Register geben dagegen nur die Speicherpositionen innerhalb dieser Segmente an und werden deshalb auch als »Offsets« bezeichnet.

Bei der Programmierung in Assembler-Sprache übernimmt der Assembler selbst die korrekte Adressierung durch die Segmentregister. Der Pro-

Die Befehle des 8086/8088-Prozessors und ihre Bedeutung

Bezeichnung Bedeutung

aaa	Ergebnis nach ASCII-Addition anpassen
aad	Ergebnis nach ASCII-Division anpassen
aam	Ergebnis nach ASCII-Multiplikation anpassen
aas	Ergebnis nach ASCII-Subtraktion anpassen
adc	addiere mit Übertrag
add	addiere ohne Übertrag
and	führe logische AND-Verknüpfung aus
call	rufe Unterprogramm auf
cbw	wandle Byte in Wort um
clc	lösche Übertrag-Statusbit
cld	lösche Richtung-Statusbit
cli	lösche Unterbrechung-Statusbit
cmc	invertiere Übertrag-Statusbit
cmp	führe Vergleich aus
cmps	vergleiche Zeichenketten
mpsB	vergleiche Zeichenketten byteweise
cmpsw	vergleiche Zeichenketten wortweise
cwd	wandle Wort in Doppelwort um
daa	Ergebnis nach dezimaler Addition anpassen
das	Ergebnis nach dezimaler Subtraktion anpassen
dec	um »1« vermindern
div	führe vorzeichenlose Division aus
esc	übergib an externe Einheit
hlt	halte Prozessor an
idiv	führe vorzeichenbehaftete Division aus
imul	führe vorzeichenbehaftete Multiplikation aus
in	lies über Port ein
inc	erhöhe um »1«
int	unterbreche
into	unterbreche bei Überlauf
iret	kehre von Unterbrechungsroutine zurück
jmp	verzweige
lahf	lade untere acht Statusbits in AH-Register
lds	lade DS-Register und weiteres Register
lea	berechne und lade effektive Adresse
les	lade ES-Register und weiteres Register
lock	aktiviere Lock-Signal für nachfolgenden Befehl
lods	lade Zeichenkette
lodsb	lade Zeichenkette byteweise
lodsw	lade Zeichenkette wortweise
loop	wiederhole Schleife, bis Zähler CX gleich 0
loope	wiederhole Schleife, bis ungleich 0 oder Zähler CX gleich 0
loopne	wiederhole Schleife, bis gleich 0 oder Zähler CX gleich 0
loopnz	wiederhole Schleife, bis gleich 0 oder Zähler CX gleich 0
loopz	wiederhole Schleife, bis ungleich 0 oder Zähler CX gleich 0
mov	übertrage Zeichen
movs	übertrage Zeichenkette

Bezeichnung Bedeutung

movesb	übertrage Zeichenkette byteweise
movesw	übertrage Zeichenkette wortweise
mul	führe vorzeichenlose Multiplikation aus
neg	negiere (2er-Komplement bilden)
nop	keine Operation ausführen
not	invertiere (1er-Komplement bilden)
or	führe logische OR-Verknüpfung aus
out	gib über Port aus
pop	lies Wort vom Stack
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
rep	wiederhole Zeichenkettenoperation, bis Zähler CX gleich 0
repe	wiederhole Zeichenkettenoperation, bis ungleich 0 oder Zähler CX gleich 0
repne	wiederhole Zeichenkettenoperation, bis gleich 0 oder Zähler CX gleich 0
repnz	wiederhole Zeichenkettenoperation, bis gleich 0 oder Zähler CX gleich 0
repz	wiederhole Zeichenkettenoperation, bis ungleich 0 oder Zähler CX gleich 0
ret	vom Unterprogramm zurückkehren
rol	rotiere nach links
ror	rotiere nach rechts
sahf	lade AH-Register in untere acht Statusbits
sal	schiebe arithmetisch nach links (entspricht SHL)
sar	schiebe arithmetisch nach rechts
sbb	subtrahiere mit Übertrag
scas	durchsuche Zeichenkette
scasb	durchsuche Zeichenkette byteweise
scasw	durchsuche Zeichenkette wortweise
seg	definiere gültiges Segmentregister für nachfolgenden Befehl
shl	schiebe logisch nach links (entspricht SAL)
shr	schiebe logisch nach rechts
stc	setze Übertrag-Statusbit
std	setze Richtung-Statusbit
sti	setze Unterbrechung-Statusbit
stos	speichere Zeichenkette
stosb	speichere Zeichenkette byteweise
stosw	speichere Zeichenkette wortweise
sub	subtrahiere ohne Übertrag
test	vergleiche über AND-Verknüpfung
wait	warte auf Testsignal
xchg	tausche Registerwerte (und Speicherinhalt) aus
xlat	adressiere Tabelle und lade Zeichen
xor	führe logische Xor-Verknüpfung aus

Tabelle 2. Die Befehle des 8086/8088-Prozessors geben Aufschluß über dessen Leistungsfähigkeit.

grammierer muß lediglich definieren, welcher Teil eines Programms zu einem Segment gehört. Die Berechnung der Adressen für die Segmentregister erledigt der Assembler.

Das erste Programm in Maschinensprache

Beim Schreiben Ihres ersten Assembler-Programms verwenden Sie als Hilfsmittel am besten den MS-DOS-Debugger Debug, der sich auf jeder MS-DOS-Systemdiskette befindet.

Der Debugger: Ein Debugger ist ein Programm, mit dem sich einzelne Speicherbereiche des Computers anzeigen und ändern lassen. Debug enthält sogar einen kleinen Assembler, der es Ihnen erlaubt, Maschinensprachebefehle auch in Assembler-Form in den Speicher zu schreiben.

Ihr erstes Programm gibt auf dem Bildschirm den Buchstaben »A« aus. In der MS-DOS-Kommandoebene starten Sie dazu den Debugger durch die Eingabe von »debug«. Nach kurzer Zeit erscheint auf dem Bildschirm ein Minuszeichen, das Bereitschaftssymbol (Prompt) des Debuggers. Das Prompt zeigt dem Benutzer an, daß der Debugger auf eine Eingabe wartet.

Wenn Sie nun den Assembler-Befehl

a0100

eingeben, speichert Debug alle Assembler-Befehle, die Sie im folgenden schreiben, als ausführbaren Maschinencode ab der Offsetadresse 0100hex im Arbeitsspeicher des PCs. Beachten Sie dabei, daß Debug alle Werte ohne Suffix als hexadezimal behandelt.

Geben Sie das folgende Assembler-Listing ein und schließen Sie die Eingabe durch das zweimalige Betätigen der Enter-Taste ab:

```
mov dl,41
mov ah,6
int 21
mov ah,4c
int 21
```

Dieses kleine Programm lädt mit dem Ihnen bereits bekannten Mov-Befehl den ASCII-Wert von »A« (41hex) in das Register DL, schreibt in das AH-Register die Zahl 6 als Nummer für eine MS-DOS-Funktion und ruft über den Software-Interrupt 21 die MS-DOS-Funktion 6 als Unterprogramm auf.

Die MS-DOS-Funktion 6 gibt den Wert im DL-Register als ASCII-Zeichen auf dem Bildschirm aus. Abschließend wird die MS-DOS-Funktion 4Chex über den Software-Interrupt 21 aufgerufen, die das Programm ordnungsgemäß beendet.

Mit Software-Interrupts und MS-DOS-Funktionen rufen Sie die Betriebssystemroutinen des PCs auf.

Mit dem Go-Befehl

g=0100

legen Sie die Startadresse fest, an der das Programm beginnt.

Nach erfolgter Eingabe starten Sie Ihr Programm mit

g

und überprüfen, ob es korrekt funktioniert.

Nach diesem ersten Erfolgserlebnis kann es sein, daß Sie Debug erneut starten müssen, um fortzufahren.

Um das Programm als ausführbare Datei auf Diskette oder Festplatte zu speichern, bestimmen Sie zunächst dessen Länge. Schauen Sie sich dazu mit dem Unassembler-Befehl

u0100

seinen Maschinencode im Speicher an. Dieser Befehl disassembliert gleichzeitig den Maschinencode in die Assembler-Befehle zurück, so daß Sie in aller Ruhe den hexadezimalen Maschinencode mit den Assembler-Befehlen vergleichen können. Suchen Sie den zweiten Interrupt-Befehl, der auch den letzten Befehl des Programms darstellt, und zählen Sie bis zu diesem Punkt die Bytes des Maschinencodes ab (jeweils zwei hexadezimale Ziffern sind ein Byte). Es müssen insgesamt zehn Bytes sein (B2, 41, B4, 06, CD, 21, B4, 4C, CD und 21).

Die Länge des Programms teilen Sie dem Debugger mit, indem Sie das CX-

Register auf diesen Wert setzen. Mit »r cx« rufen Sie die Wertzuweisung für das Register CX auf und geben hinter dem Doppelpunkt den hexadezimalen Wert 0Ahex, der dem dezimalen Wert 10 entspricht, ein.

Einen Namen geben Sie Ihrem Programm mit dem Name-Befehl.

n democom

weist beispielsweise Ihrem Programm den Namen »democom« zu. Wichtig ist die Namensweiterung »com«, weil das Programm sich sonst nicht korrekt starten läßt. Mit dem Write-Befehl »w« speichern Sie schließlich die Datei.

Wenn Sie danach mit »q« für »quit« den Debugger verlassen, läßt sich das Programm durch die Eingabe von »demo« aus der MS-DOS-Kommandoebene starten, so daß der Buchstabe »A« unter dem MS-DOS-Prompt erscheint. Entsprechend dem Wert, den Sie in das DL-Register schreiben, erscheint ein anderes ASCII-Zeichen am Bildschirm.

Die Vorteile eines Assemblers

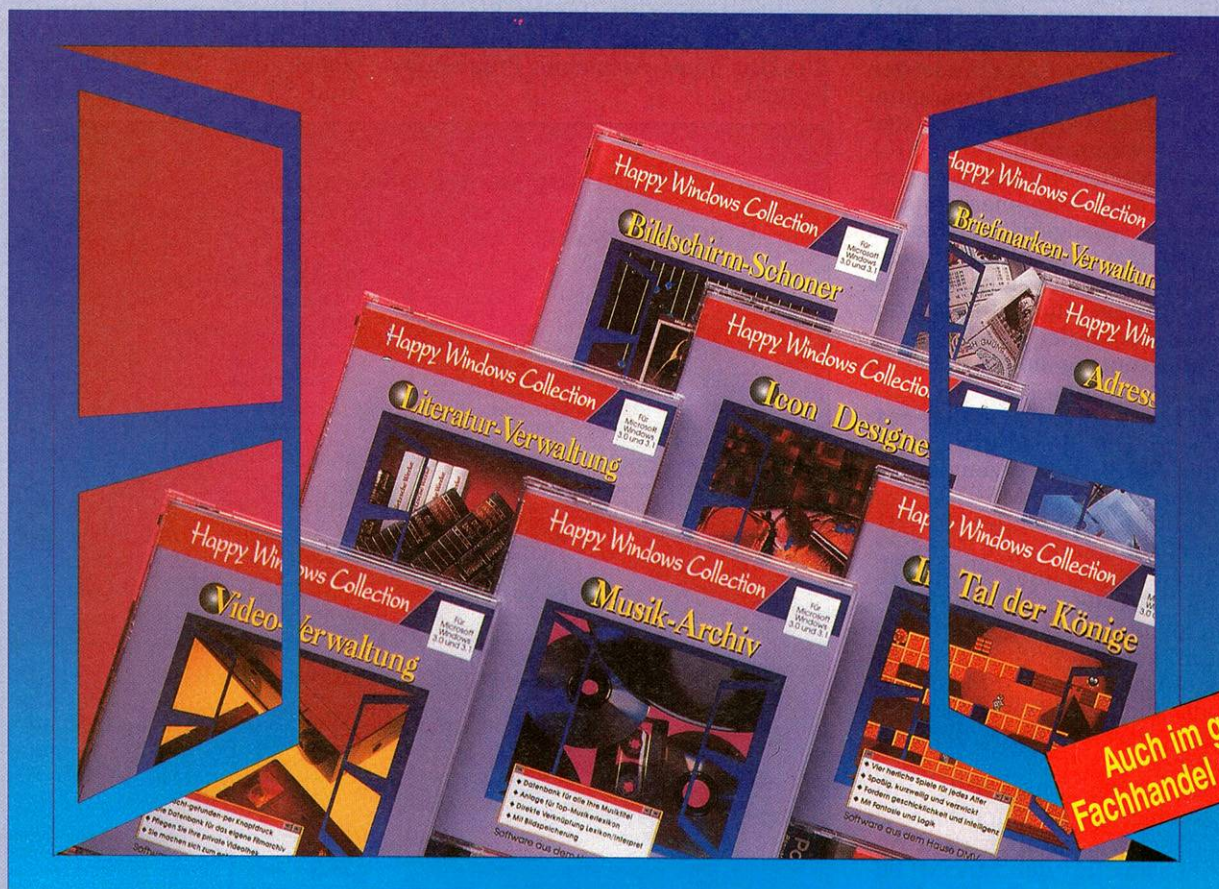
Bei kurzen Maschinenspracheprogrammen eignet sich der Debugger zur Be-

Die bedingten Sprungbefehle der Intel-Prozessoren

Befehl	englisch	entspricht	Befehl	Bedeutung
ja	jump if above	jnb	jump if not below	verzweige, wenn darüber
jae	jump if above or equal	jnb	jump if not below	verzweige, wenn darüber oder gleich
jb	jump if below	jnae	jump if not above or equal	verzweige, wenn darunter
jbe	jump if below or equal	jna	jump if not above	verzweige, wenn darunter oder gleich
jcxz	jump if CX equal zero	-	-	verzweige, wenn Inhalt von CX gleich 0
je	jump if equal	jz	jump if zero	verzweige, wenn gleich
jg	jump if greater	jnl	jump if not less	verzweige, wenn größer
jge	jump if greater or equal	jnl	jump if not less	verzweige, wenn größer oder gleich
jl	jump if less	jnge	jump if not greater or equal	verzweige, wenn kleiner
jle	jump if less or equal	jng	jump if not greater	verzweige, wenn kleiner oder gleich
jna	jump if not above	jbe	jump if below or equal	verzweige, wenn nicht darüber
jnae	jump if not above and not equal	jb	jump if below	verzweige, wenn nicht darüber und nicht gleich
jnb	jump if not below	jae	jump if above or equal	verzweige, wenn nicht darunter
jnb	jump if not below and not equal	ja	jump if above	verzweige, wenn nicht darunter und nicht gleich
jne	jump if not equal	jnz	jump if not zero	verzweige, wenn nicht gleich
jng	jump if not greater	jle	jump if less or equal	verzweige, wenn nicht größer
jnge	jump if not greater and not equal	jl	jump if less	verzweige, wenn nicht größer und nicht gleich
jnl	jump if not less	jge	jump if greater or equal	verzweige, wenn nicht kleiner
jnl	jump if not less and not equal	jg	jump if greater	verzweige, wenn nicht kleiner und nicht gleich
jno	jump if not overflow	-	-	verzweige, wenn kein Überlauf
jnp	jump if not parity	jpo	jump if positive overflow	verzweige, wenn keine Parität
jns	jump if not sign	-	-	verzweige, wenn positiv
jnz	jump if not zero	jne	jump if not equal	verzweige, wenn nicht 0
jo	jump if overflow	-	-	verzweige, wenn Überlauf
jp	jump if parity	jpe	jump if parity even	verzweige, wenn Parität
jpe	jump if parity even	jp	jump if parity	verzweige, wenn Parität gerade
jpo	jump if parity odd	jnp	jump if not parity	verzweige, wenn Parität ungerade
js	jump if sign	-	-	verzweige, wenn negativ
jz	jump if zero	je	jump if equal	verzweige, wenn 0

Tabelle 3. Die bedingten Sprungbefehle gehören zu den wichtigsten Leistungsmerkmalen der 8086/8088-Prozessoren.

Wir haben die Fenster zu Windows geöffnet



Im Tal der Könige

4 herrliche Windows-Spiele für jedes Alter: spannend, kurzweilig und verwickelt. Sie erproben die Geschicklichkeit und Intelligenz, die Fantasie und Logik. "Bobby im Tal der Könige", "Bobby im Fantasia-Land", "Twixt" und "Mini-Scrabble" – ein Spiele-Set für die ganze Familie.

Icon Designer

Für das persönliche Outfit Ihrer Windows-Oberfläche: Icons gestalten und einbinden. Und das gibt's nur hier: Mit der Maus fangen Sie einfach ein Motiv von der Oberfläche ein und bearbeiten es im Icon Designer nach!! Das ist der einfachste Weg, Windows und die Icons zu verschönern.

Bildschirm-Schoner

Was macht Ihr Bildschirm in der Pause? "Dark Fader" macht mit 29 lustigen Filmchen jede Pause zum doppelten Genuß. Sie können verschiedene Voreinstellungen vornehmen und ausgewählte Schonprogramme in Serie schalten. So verlängern Sie die Lebensdauer Ihres Monitors und schützen die Bildröhre vor Einbrennen.

Adressen-Manager

Legen Sie Wert darauf, ein Profi-System zum Low-Cost-Preis zu erwerben? Z.B. per Knopfdruck sortieren, praxisorientierte Suchfunktionen, Einkopieren in Arbeitstext, Datenexport für Serienbriefe, schneller Etikettendruck, unbegrenzte Datenmenge, Datenschutz, Zusatztext, Freitext-Recherche, vorläufiges Löschen, Wiederherstellen, Schriftzeileinstellung beim Druck u.v.a.m.

Briefmarken-Verwaltung

Kataloge enthalten immer nur Standardwerte. Jetzt haben Sie die Möglichkeit, jede Ihrer wertvollen Marken mit einem extra Datenblatt zu versehen. Sie halten Infos fest, die nur Sie kennen und beschreiben die spezifischen Merkmale Ihres Unikates. So wird Ihre Wertanlage transparent und kalkulierbar. Mit jedem Datenblatt können Sie sogar ein Bild verknüpfen.

Musik-Archiv

Kennen Sie noch alle Hitparaden-Stürmer und die Goldies der Oldies, oder wissen Sie noch, wann Karajan seine letzte Symphonie aufgenommen hat? Machen Sie sich doch zum Musikexperten, indem Sie ein für alle Mal Ihr persönliches Archiv anlegen. Sie verwalten damit Ihren gesamten Plattenbestand und können ein extra Musiklexikon anlegen.

Literatur-Verwaltung

Jetzt haben auch "Leseratten" Spaß am Computer! Diese neue Datenbank ist geeignet, Schriftstücke jeglicher Art systematisch zu verwalten. Katalogisieren Sie Ihren gesamten Lesestoff! Recherchieren Sie schnell und zuverlässig. Bibliographieren Sie nach modernen Gesichtspunkten. Zum schnellen Wiederfinden genügt ein Tastendruck.

Video-Verwaltung

Sie haben tolle Home-Videos, Kino-Klassiker und Dokumentarfilme – fragt sich nur, wo sich die guten Stücke befinden und wer wann wo mitgespielt hat. Hier haben Sie die Datenbank für das private Filmarchiv: ausfüllen, auswählen, sortieren, filtern, suchen und finden, drucken, exportieren und Bilder verknüpfen! Überblick im Handumdrehen.

Jedes Programm

Händleranfragen erwünscht

nur **49,80 DM***

(unverbindliche Preisempfehlung)

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,- bzw. für das Ausland DM 6,- Porto und Verpackung.

Bitte benutzen Sie die Bestellkarte.

DMV-Verlag • Postfach 250 • W-3440 Eschwege

Tel. 05651/809-0, Fax 05651/809-430

DMV
Daten- und
Medienverlag

fehlseingabe recht gut, doch für längere Programme ist das beschriebene Verfahren zu umständlich. Hier ist es wesentlich einfacher, zuerst den gesamten Quelltext einzugeben, um ihn dann komplett von einem Assembler übersetzen zu lassen.

In diesem Fall können Sie auch mit symbolischen Sprung- und Adreßmarken arbeiten, so daß Sie bei einer Verzweigung im Programm oder für ein Datenfeld nicht jedesmal umständlich die absolute Adresse berechnen müssen. Zudem würden sich die meisten Adressen bei jeder kleinen Programmänderung verschieben, so daß sie neu zu berechnen wären.

Ein Assembler erlaubt Ihnen dagegen, für Sprung- und Adreßmarken statt Zahlenwerten selbstgewählte Namen zu benutzen, die wesentlich aussagekräftiger und bequemer zu handhaben sind. Die Sprung- und Adreßmarken rechnet der Assembler erst beim Assemblierungsvorgang automatisch in absolute Adressen um.

Für die Eingabe eines Assembler-Programms benötigen Sie lediglich einen Editor, der ASCII-Texte erzeugt. Dies sind beispielsweise die MS-DOS-Programme Edit und Edlin oder ein Textverarbeitungsprogramm wie Word.

Listing 1 zeigt ein kleines Assembler-Programm, das über die MS-DOS-Funktion 09hex einen Text auf den Bildschirm ausgibt und sowohl für den Makro-Assembler MASM von Microsoft als auch für den Assembler SAM 4 aus dieser Ausgabe der DOS Extra gedacht ist.

Tippen Sie jedoch zunächst das Programm Buchstabe für Buchstabe ab und speichern Sie es unter dem Namen »print.asm« (für den MASM) beziehungsweise »print.sam« (für SAM 4) auf Diskette oder Festplatte.

Die Struktur eines Assembler-Programms

Beim Übersetzen von Programmen unterscheiden Assembler zwischen zwei Arten von Befehlen: den »echten« Maschinensprachebefehle in Assembler-Form (»jmp«, »mov«, »int« und so weiter), auch Opcodes genannt, und den Pseudobefehlen (»title«, »assume«, »org« und ähnliche), auch Pseudo-Opcodes genannt.

Während der Assembler die Maschinensprachebefehle direkt in den vom Prozessor ausführbaren Maschinencode umwandelt, ist ein Pseudobefehl kein Maschinensprachebefehl des Prozessors, sondern stellt vielmehr eine Anweisung für den Assembler dar.

● **Pseudobefehle:** Ein Pseudobefehl bestimmt beispielsweise, ab welcher Speicheradresse ein Programm beginnt (»org«), definiert die Grenzen

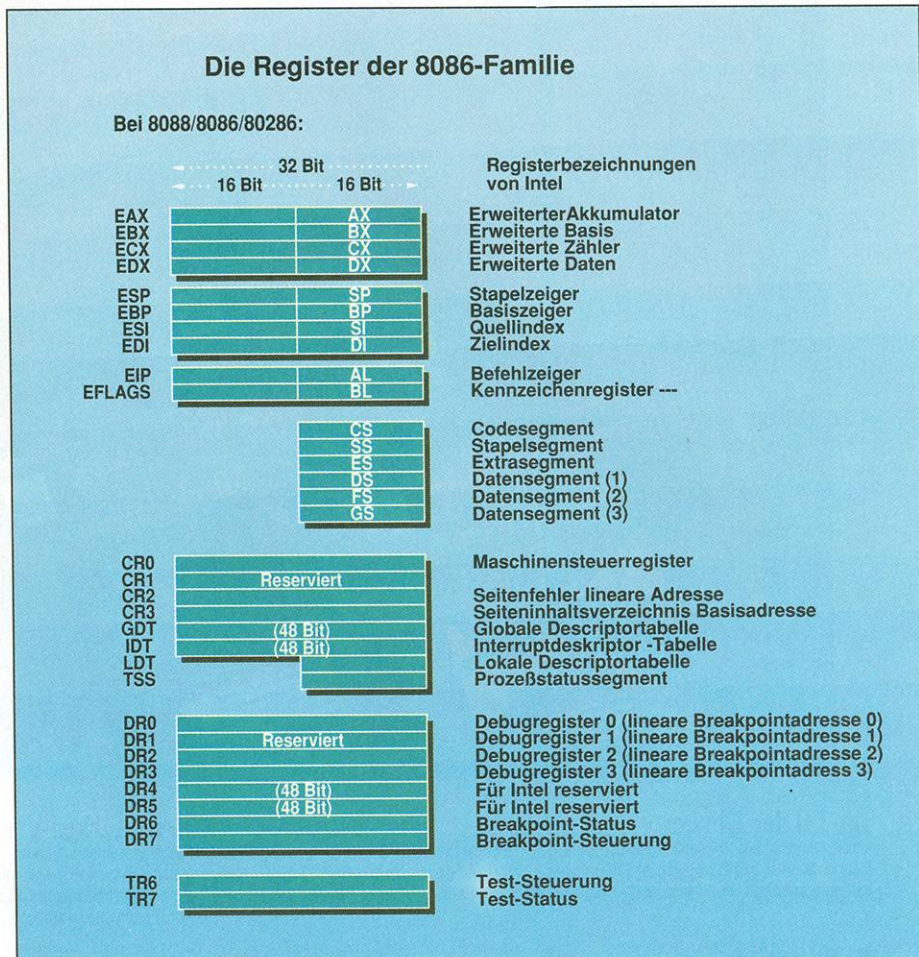
von Speichersegmenten (»segment« und »ends«) oder legt das logische Programmende fest (»end«).

Die Anzahl der Pseudobefehle eines Assemblers bestimmen dessen Leistungsfähigkeit und Komfort. Auf diesem Gebiet ist der MASM von Microsoft bisher ungeschlagen, aber auch SAM 4 braucht sich nicht zu verstecken.

Weil das Programm ein typisches Assembler-Listing ist, lernen Sie dabei auch gleichzeitig die allgemeine Struktur eines Assembler-Programms kennen.

gekennzeichnet sind und die der Assembler ignoriert. Diese Kommentare dürfen auch am Anfang einer Zeile stehen, sofern sie mit einem Semikolon beginnen.

In der zweiten Zeile des Listings definiert der Pseudobefehl »title« den Programmnamen »print«. Darauf folgen mehrere Kommentarzeilen, die die Aufgabe des Programms erläutern. Mit dem Pseudobefehl »assume« in Zeile 9 wird den Segmentregistern CS und DS das (einzige) Speichersegment »code« zugeordnet. Die Segmentregister ES und SS kommen im Programm Print



Der Registersatz des 80386/80486-Prozessors enthält den vollständigen Registersatz seiner Vorläufer.

Allgemein besteht der Text von Assembler-Programmen aus bis zu vier Spalten. Die erste Spalte enthält die Namen von Sprungmarken (englisch »labels«) und Variablen. Marken muß – normalerweise – immer ein Doppelpunkt folgen.

In der zweiten Spalte befinden sich die Assembler-Befehle und Pseudobefehle, und in der dritten Spalte des Programmtextes stehen die Bezeichnungen von Registern und Operanden für die Assembler-beziehungsweise Pseudobefehle.

Die vierte und letzte Spalte enthält Kommentare, die durch ein Semikolon

nicht zur Anwendung und fehlen deshalb. Beachten Sie bitte, daß SAM 4 die Segmentzuweisungen selbsttätig übernimmt und aus diesem Grund Assume-Anweisungen ignoriert.

Der Pseudobefehl »segment« in Zeile 12 definiert den Segmentstart von »code«, und mit dem Pseudobefehl »org« in Zeile 15 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.

Das Listing ist als Com-Datei angelegt, weil dies den Programmaufbau in As-

sembler-Sprache vereinfacht und die Programmlänge minimiert.

● **Die MS-DOS-Funktion 09hex:** »start:« in Zeile 17 ist die erste Marke im Programm. Sie bezieht sich auf den Pseudobefehl »end« und markiert den Startpunkt des Programms. Der einem End-Befehl folgende Name definiert nämlich jeweils die Sprungmarke, die beim Start eines Programms aufgerufen wird.

Der erste echte Assembler-Befehl des Programms ist der Sprungbefehl »jmp«, der beim SAM-Assembler »jmps« (für »jump short«) heißen muß. Dieser Befehl überspringt den Datenbereich im Quellcode, der den auszugebenden Text enthält, und ruft den eigentlichen Anfang des Programms auf. Der Text ist im Datenbereich ab Zeile 21 der Variablen »text« über den Pseudobefehl »db« (»define byte«) zugeordnet.

Selbstverständlich können Sie hier auch einen anderen Text einsetzen. Wichtig ist nur, daß der Text mit einem Dollarzeichen abschließt, das für die MS-DOS-Funktion 09hex bei der Ausgabe auf dem Bildschirm die Endbegrenzung darstellt. 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 Assembler-Programm. Der Pseudobefehl »offset« bewirkt, daß die Adresse des Speicherbereichs der Variablen »text«, die den auszugebenden Text enthält, in das DX-Register geschrieben wird. Fehlt der Pseudobefehl »offset«, so würde nicht die Adresse, sondern das erste Zeichen der Variablen »text« selbst in das DX-Register geladen.

Anschließend wird durch den Wert 9 im AH-Register und über den Software-Interrupt 21hex die MS-DOS-Funktion 09hex 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 des Textes.

Auch das Programmende führt eine MS-DOS-Funktion des Software-Interrupts 21hex herbei. Diesmal wird die Funktion 4Chex aufgerufen. Über diese Funktion kann das Programm problemlos in die MS-DOS-Kommandoebene zurückkehren. Der Pseudobefehl »ends« gibt schließlich das Ende des Segments »code« an, und der Befehl »end« markiert das Programmende.

● **Das Assemblieren:** Wenn Sie das Programm nun assemblieren möchten, rufen Sie den MASM oder SAM 4 auf, indem Sie auf der MS-DOS-Kommandoebene den Dateinamen »masm«

Zahlenkette (in hex)	Assembler-Befehl (Zahlen in hex)
B4 30	mov ah,30
CD 21	int 21
3D 05 00	cmp ax,0005

Tabelle 4. Hier sehen Sie, welche Zeichenkette im Beispiel welchem Assembler-Befehl entspricht.

oder »sam4« eingeben und den Assembler mit [Enter] starten. Alle weiteren Schritte ergeben sich aus den Angaben, die der Assembler benötigt. Wenn Sie schon mit SAM 4 arbeiten, geben Sie lediglich den Dateinamen »print« ein, wählen das Com-Modell aus und starten die Übersetzung. Mit MASM ist das etwas komplizierter. Damit Sie sich beim MASM nicht erst in dessen Bedienung einarbeiten müssen, schreiben Sie am einfachsten eine Stapeldatei, die ein Assembler-Programm automatisch übersetzt. Geben Sie dazu auf der Kommandoebene folgendes ein:

```
copy con asm.bat
masm %1 ;
link %1 ;
exe2bin %1 %1com
```

Anschließend betätigen Sie die Enter-Taste, die Taste F6 und nochmals die Enter-Taste. Damit besitzen Sie eine Stapeldatei, die Ihre Assembler-Listings auf Befehl übersetzt.

Schneller arbeiten mit Stapeldateien

Sorgen Sie dafür, daß sich die oben beschriebene Stapeldatei, der MASM, der Linker »link« des MASM, der MS-DOS-Befehl »exe2bin« und in diesem Fall das Assembler-Programm »print.asm« auf der gleichen Diskette beziehungsweise im gleichen Unterverzeichnis befinden.

Wenn Sie das Programm »print.asm« mit »asm print« übersetzen, erzeugt der MASM aus dem Assembler-Programm zuerst ein Objektmodul, der Linker formt 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 »printexe« und »printcom«. Die Datei »printexe« kön-

nen Sie löschen, weil sie nur eine Zwischenstufe darstellt und nicht korrekt funktioniert. »printcom« ist dagegen das Endergebnis Ihrer Bemühungen und läßt sich durch Eingabe von

print

starten. Sollte die Textausgabe wider Erwarten nicht funktionieren, dann vollziehen Sie die oben beschriebenen Schritte nochmals sorgfältig nach. Haben Sie das Listing richtig abgetippt und unter dem Namen »print.asm« auf Diskette gespeichert? Stimmt der Inhalt von »asm.bat« mit den oben beschriebenen Befehlen überein? Sind alle notwendigen Dateien auf der Diskette vorhanden?

Der Einstieg in die Maschinenspracheprogrammierung ist nicht einfach, doch zusätzlich mit den folgenden zwei Beiträgen (»Der Rechenschieber« und »Musik liegt in der Luft« wird für Sie das Thema »Programmieren in Maschinensprache« immer verständlicher.

Übersicht über die folgenden Themen

Maschinensprache zu verstehen, ist nicht leicht; sie zu programmieren, noch komplexer. Damit Sie immer wiederkehrende Funktionen nicht erneut eingeben müssen, bieten Assembler wie MASM oder SAM 4 ein hilfreiches Verfahren: die Makrotechnik. Erlernen Sie die trickreiche Methode zusammen mit den wichtigsten Assembler-Befehlen und realisieren Sie ihren Einsatz unter den verschiedenen Speichermodellen.

Tabelle 2 bietet eine kleine Übersicht über die Befehle des 8086/8088-Prozessors. Eine komplette Referenz der Assembler-Befehle bis zum 80486-Prozessor finden Sie auch als DOSsier in dieser Ausgabe der DOS Extra.

Nachdem Sie die Grundlagen der Programmierung in Maschinensprache kennengelernt haben, steigen Sie jetzt tiefer in die Geheimnisse der Assembler-Programmierung ein und erfahren einige Tricks, die auch Profis beim Schreiben ihrer Assembler-Programme verwenden.

Sie kennen nun den Unterschied zwischen Maschinen- und Assembler-Sprache und wissen, wie die verschiedenen Zahlensysteme funktionieren

Zahlen und Versionen		
MS-DOS-Version	Assembler-Befehl	Zahlenkette
3.20	cmp ax,2003	3D 03 20
3.30	cmp ax,3003	3D 03 30
4.00	cmp ax,0004	3D 04 00
4.01	cmp ax,0004 (!)	3D 04 00 (!)
5.00	cmp ax,0005	3D 05 00
6.00	cmp ax,0006	3D 06 00

Tabelle 5. Alle Versionsnummern ab MS-DOS 3.2 und die entsprechende Zahlenkette bei der Versionsabfrage

und wie ein typisches Assembler-Programm aufgebaut ist.

● **Erinnerung an die Register:** Sicher erinnern Sie sich 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 $2^{16} = 65\,536$ Speicherzellen (64 KByte) adressieren lassen, während der maximale Speicherbereich des Prozessors 1 MByte beträgt. Deshalb geben die Segmentregister des 8086/8088 jeweils die Startadresse (»Basis«) für ein Speichersegment an, und die Adresse in einem der normalen 16-Bit-Register (»Offset«) zeigt auf eine einzelne Speicherzelle innerhalb dieses Segments. Weil ein 16-Bit-Register höchstens 64 KByte Speicherbereich adressieren kann, ist auch die Länge eines Speichersegments auf maximal 64 KByte begrenzt. Ganz konkret wird die effektive Adresse einer Speicherzelle berechnet, indem die Segmentadresse in einem Segmentregister mit 16 multipliziert und zum Wert der Adresse eines normalen 16-Bit-Registers addiert wird. Enthält beispielsweise das Segmentregister DS die Adresse A000hex und das Register DX die Adresse 4000hex, so beträgt die effektive Adresse dieser beiden Register

A000hex $\times$ 10hex
+ 4000hex
= A4000hex

Mit dem DX-Register läßt sich damit der Speicherbereich von A0000hex (DX = 0000hex) bis AFFFFhex (DX = FFFFhex) adressieren.

Jedes Segmentregister des 8086/8088 hat seinen eigenen Zuständigkeitsbereich. Das Segmentregister CS (Code Segment) ist für den Befehlszeiger IP zuständig, SS (Stack Segment) gibt die Segmentadresse für den Stapelzeiger SP und den Basiszeiger BP an, und ES (Extra Segment) bestimmt das Speichersegment für das Zielregister DI bei Zeichenkettenoperationen. 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 Assembler-Programmen müssen Sie sich um diese Feinheiten nicht kümmern, wenn Ihre Programme nicht länger als 64 KByte sind – und welches Assembler-Programm ist das schon.

Code und Daten in einem Segment

Dann können Sie nämlich das Speichermodell für Dateien mit der Endung »com« verwenden, das mit einem einzigen Speichersegment arbeitet.

In diesem Speichersegment sind Programmcode und Daten gemeinsam

enthalten. Der Programmcode beginnt ab der Adresse 0100hex mit einem Sprungbefehl auf den logischen Programmstart, unmittelbar gefolgt von den Daten des Programms. Daran schließt sich der eigentliche Programmcode an. Für den Stapel (englisch »stack«) reserviert MS-DOS automatisch 100hex Byte am oberen Ende des Speichersegments.

Wenn nun die Daten und der Programmcode zusammen weniger als 64 KByte bis 100hex Byte belegen und die Stapelgröße den Betrag von 100hex Byte tatsächlich nicht überschreitet, funktioniert das Verfahren einwandfrei. Überschneiden sich dagegen die Bytes von Programm, Daten und Stapel, so führt dies in der Regel zum Absturz des Programms.

● **Das PSP:** Der Speicherbereich von Adresse 0000hex bis 00FFhex innerhalb des Speichersegments einer Datei mit der Endung »com« wird für das Kennzeichen dieses Dateityps benötigt: das PSP (Program Segment Prefix). Das PSP enthält Daten, die für die Verwaltung der Com-Datei durch das Betriebssystem erforderlich sind. Dieser Programmvorspann wird beim Aufruf einer Com-Datei aus der MS-DOS-Kommandoebene von MS-DOS selbsttätig angelegt.

Das Speichermodell für Com-Dateien wurde bereits in den beiden vorherigen Assembler-Programmen verwendet.

Listing 2 zeigt zusammenfassend den beschriebenen Aufbau eines Assembler-Programms nach dem Speichermodell der Com-Dateien für den MASM von Microsoft und für SAM 4 aus Ausgabe 21 der DOS Extra. Wenn Sie Ihre Assembler-Programme mit diesem im folgenden näher erläuterten Grundgerüst aufbauen, das heißt, anstelle der Kommentare und Punkte Daten und Befehle einfügen, können Sie am Ende die erzeugte Com-Datei problemlos aus der MS-DOS-Kommandoebene aufrufen.

● **Programmgrüst von Com-Dateien:** Das Programmgrüst für Dateien, die mit dem Speichermodell der Com-Dateien arbeiten, sieht immer gleich aus. Zunächst weisen Sie den Segmentregistern CS und DS mit dem Pseudobefehl »assume« ein gemeinsames Speichersegment zu.

Wenn das Programm keine Zeichenkettenoperationen enthält, muß das Segmentregister ES nicht initialisiert werden. Eine Zuweisung für das Stapel-Segmentregister SS ist sogar überflüssig, weil es automatisch initialisiert wird. Zu diesem Zweck reserviert das Betriebssystem MS-DOS am oberen Ende des Speichersegments 100hex Byte und legt als Kennzeichen für die Stapelspitze (»top of stack«) in den Speicherzellen FFFhex und FFFFhex die 16-Bit-Adresse 0000hex ab. Diese Adresse weist auf die erste Adresse im PSP, die die Rücksprungadresse 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 zugewiesen 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 »link« des MASM (Makro-Assembler) warnt in diesem Fall zwar mit

Warning: no stack segment

davor, daß kein Segment für den Stapel definiert wurde. Doch diese Fehlermeldung können Sie ignorieren, wenn Sie vorhaben, die vom Linker erzeugte Exe-Datei mit dem MS-DOS-Befehl »exe2bin« in eine Com-Datei umzuwandeln. SAM 4 ist auf das Assume-Kommando nicht angewiesen. Der DOS-Assembler verwaltet die Segmentzuweisungen automatisch.

Das Kennzeichenregister

Bitnummern der 8086-kompatiblen Flags *

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	NT	IO	PL	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF
NT = nested task				IF = interrupt flag				AF = auxiliary carry							
IOPL = I/O privilege level				TF = trap flag				PF = parity flag							
OF = overflow flag				SF = sign flag				CF = carry flag							
DF = direction				ZF = zero flag											

Bitnummern der zusätzlichen 80386-Flags

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Nur für Intel reserviert														VM	RF
VM = virtueller 8086 - Modus RF = Resume-Flag															

(Bit 31 = Paging-Freigabe, Bit 4 = Koprozessor-Erweiterungstyp, Bit 3 = Prozeßwechsel, Bit 2 = emulierte Koprozessor, Bit 1 = überwachte Koprozessor, Bit 0 = Schutzfreigabe)

(alle 32-Bit werden für Adressierung verwendet)

(Bit 12-31 bilden Basisregister des Seiteninhaltsverzeichnisses, übrige Bits reserviert)

Bild 3. Das Kennzeichenregister erlaubt dem Assembler-Programmierer den Einsatz bedingter Operationen.

POWER HITS

Jetzt auch
im guten
Fachhandel

Für IBM
Kompatible

Starke Software zum Spitzenpreis

PC-Spiel des Wissens 1

Das beliebte Fragepiel für die ganze Familie.

- Freizeitpaß mit kniffligen Fragen
- Fordert Allgemeinbildung und Flexibilität
- Ausgesuchte Wissensgebiete
- Geografie und Reisen
- Natur und Sport
- Medizin und Technik
- Für 1 bis 4 Personen
- Erweiterbar durch eigene Fragen, Themen und Antworten.

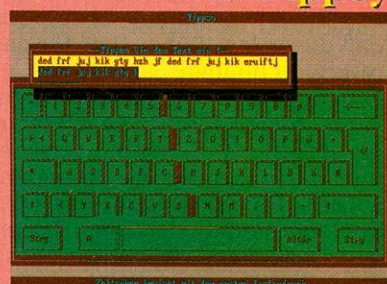
(SA: 512 KB RAM)



Tippsy

Maschinenschreiben leichtgemacht:

- Top-Training des Zehnfingersystems
- Schritt für Schritt der Bestform nähern
- Erfolge binnen kürzester Zeit
- Automatische Leistungskontrolle für die persönliche Motivation
- objektive Ergebnisbeurteilung
- Ideal auch für Ausbilder
- Veränderbare Lektionen



Text-Compiler



Der Text-Compiler wandelt Texte in Programme, die man starten kann wie jede andere Anwendung.

- Texte in eleganter Form präsentieren
- Elektronische Briefe verpacken
- "Readme-Files" compilieren
- Meldungen "in letzter Sekunde" verschicken
- Mitteilungen vor Eingriffen schützen, Informationen sicher übertragen
- Texte resident laden, um sie in laufenden Programmen per Hotkey abzurufen
- Lesen und Drucken ohne Textverarbeitung

MultiCopy



MultiCopy dupliziert Disketten in einem Rutsch - einfach und schnell!

- Sogar bei HD-Disketten entfällt das ständige Wechseln zwischen Quell- und Zieldiskette
- Komplettes Einlesen der Disk (RAM, EMS oder Festplatte)
- Auch späteres Kopieren ohne Originaldisketten möglich
- Bester Bedienungskomfort durch SAA-Oberfläche

Terminplaner

Nie mehr zu spät, nichts mehr vergessen: Der Terminplaner erinnert Sie pünktlich an Ihre Aufgaben.

- Automatische Weckfunktion
- Unabhängig von der laufenden Anwendung
- Leichte Bedienung
- Macht auch dem EDV-Laien Freude
- Speicher sparen - dennoch informiert: auf Abruf einblendbare Terminliste
- Für private Anliegen und geschäftliche Bereiche



Dieser Fraktal-Generator macht mehr aus dem Spiel mit den wundersamen Zahlen, Formen und Farben, nämlich Objekte und Landschaften, die Sie in 3D-Qualität durchwandern können!

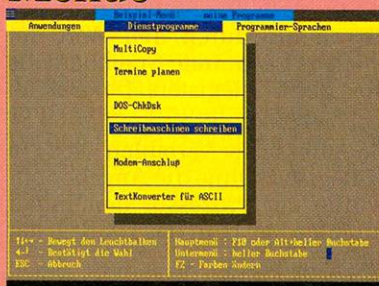
- Freie Wahl des Betrachterstandorts
- Unendliche Ausschnittvergrößerung
- Atemberaubender Aufbau
- Rekordverdächtige Bildberechnung
- Keine mathematischen Kenntnisse erforderlich

(SA: EGA/VGA)

Fraktal 3D



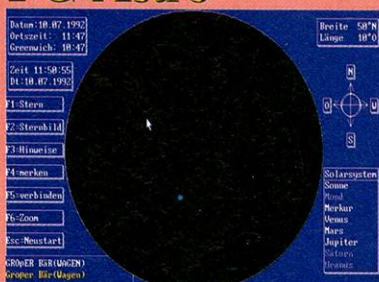
Menue



Menue ist die Benutzeroberfläche für Ihre individuelle Programmauswahl.

- Abgestimmt auf Ihren Computer, auf Ihre Anwendungen und auf Ihre Wünsche
- Startet Programme per Knopfdruck
- Lädt das gewünschte Dokument gleich mit
- Gibt Speicher frei
- Stellt den gesamten RAM einer Anwendung zur Verfügung

PC-Astro



PC-Astro macht Ihren Computer zum Space-Lab für Reisen durch Raum und Zeit.

- Karte des Sternenhimmels mit über 1200 Objekten
- Umfangreiche Berechnungsmöglichkeiten
- Freie Wahl des Beobachtungsortes und des Zeitpunktes
- Simulation von Zeitabläufen
- Ausschnittvergrößerungen
- Suche nach Objekten
- Lexikalische Info-Texte

(SA: 640 KB RAM, EGA/VGA)

Bitte benutzen Sie die Bestellkarte im Heft.

Händleranfragen erwünscht

Jedes Programm nur DM

29,80 *

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung.

DMV-Verlag • Postfach 250 • W-3440 Eschwege

DMV
Daten- und
Medienverlag

Der nächste Befehl im Programm muß mit

```
code segment para 'code'
```

den Start des Speichersegments definieren. In diesem Beispiel weist er gleichzeitig dem Segment den Namen »code« zu.

Der sich anschließende Pseudobefehl

```
org 0100h
```

gibt dem Assembler die Anweisung, das Programm ab der Adresse 0100hex im Speichersegment abzulegen, so daß die ersten 100hex Byte für das PSP reserviert bleiben.

Nun muß unbedingt der Sprungbefehl »jmp« folgen, der mit der Marke, die im Befehl »end« steht, versehen ist und den eigentlichen Programmstart aufruft. Ein Programm, das mit dem Speichermodell der Com-Dateien arbeitet, darf aus Strukturierungsgründen 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 gestartet wird.

● **Der Weg zum Programmende:** Um das Programm elegant zu beenden, empfiehlt sich die Funktion 4Ch des MS-DOS-Interrupts 21hex, die eine Rückkehr in die MS-DOS-Kommandoebene bewirkt. Zwar schließt auch der MS-DOS-Interrupt 20hex ein Programm ab, doch ist dieser Interrupt veraltet, so daß er bei Dateien mit der Endung »exe« fehlerhaft arbeitet und aus Gründen der Aufwärtskompatibilität nicht benutzt werden sollte.

```
mov ah,4Ch  
int 21h
```

lautet die korrekte Befehlsfolge zum Beenden des Programms.

Abschließend legen Sie mit den beiden Pseudobefehlen

```
code ends  
und  
end start
```

das Segmentende von »code« sowie das Programmende fest. Die Angabe von »start« im Befehl »end« markiert die Zeile, in der die Sprungmarke »start« steht, als Einstiegspunkt beim Programmstart.

Exe – Mehr Platz für Code und Daten

Für Dateien mit der Endung »exe« existiert ein eigenes Speichermodell. Beim Speichermodell der Exe-Dateien definieren Sie im Unterschied zum Speichermodell für Com-Dateien beliebig viele Speichersegmente. Listing 3 zeigt ein Beispiel, das mit einem Pro-

grammsegment, 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 ebenfalls deutlich hervor.

● **Initialisierung der Segmentregister:** Interessant ist, daß bei diesem Speichermodell die Segmentregister DS, ES und SS einzeln initialisiert werden müssen. Während dies für das Segmentregister CS beim Programmstart automatisch geschieht (CS gibt schließlich mit dem Befehlszeiger IP die Startadresse des Programms an), muß der Programmierer die Initialisierung der übrigen Segmentregister vornehmen.

Zuerst wird jeweils das AX-Register mit der Segmentadresse des entsprechenden Speichersegments geladen und anschließend dieser Wert in DS geschrieben. Ein direkter Befehl in der Art

```
mov ds,segment
```

ist beim 8086/8088-Prozessor nicht vorgesehen, so daß der Umweg über das AX-Register notwendig ist.

Wie Sie sehen, ist beim Speichermodell für Exe-Dateien der Umgang mit den Segmenten und Segmentregistern etwas komplizierter, so daß sich zumindest für den Anfang das Speichermodell der Com-Dateien empfiehlt.

● **Exe zu Com mit Exe2bin:** Arbeiten Sie mit dem MASM-Assembler, sollten Sie unbedingt die vom Linker erzeugten Exe-Dateien mit dem MS-DOS-Befehl »exe2bin« in Com-Dateien umwandeln. Exe-Dateien, die nach dem Speichermodell der Com-Dateien aufgebaut sind, funktionieren nämlich meist nicht korrekt. Umgekehrt lassen sich Exe-Dateien, die mit mehreren Speichersegmenten arbeiten, von vornherein nicht in Com-Dateien konvertieren.

Listing 3 zeigt neben dem Aufbau des Speichermodells für Exe-Dateien auch den Pseudobefehl »dw« (»define word«), den sowohl MASM als auch SAM 4 verarbeiten. Dieser Befehl ähnelt dem vorher vorgestellten Befehl »db«, nur definiert er nicht ein oder mehrere Bytes, sondern ein Wort (word) beziehungsweise mehrere Wörter (words). Während ein Byte als 8-Bit-Wert nur eine Speicherzelle belegt, ist das Word ein 16-Bit-Wert und nimmt demzufolge zwei unmittelbar aufeinanderfolgende Speicherzellen ein. Dabei wird das niederwertige vor dem höherwertigen Byte im Speicher abgelegt. Dieses Verfahren ist als Intel-Format bekannt.

Listing 3 enthält auch eine neue Variante der Befehle »db« und »dw«. Hier bekommen die vor den Befehlen ste-

henden Variablen keine absoluten Werte zugewiesen, sondern es werden nur Speicherbereiche initialisiert. Diese Initialisierung erfolgt 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.

Die Dup-Zuweisung ist nur für den MASM bestimmt. Bei SAM 4 ersetzen Sie diese Zuweisung durch den vergleichbaren DS-Befehl. Das bedeutet beispielsweise, daß Sie

```
stapel db 200h dup(0)
```

gegen

```
stapel ds 200h,0
```

austauschen müssen, damit das Programm fehlerfrei funktioniert.

Ein Überblick über die wichtigsten Assembler-Befehle verhilft Ihnen dazu, auch die Bedeutung der MS-DOS-Funktion besser zu begreifen.

Tabelle 1 zeigt den gesamten Befehlsatz des 8086/8088-Prozessors in alphabetischer Reihenfolge.

Befehle, die der Prozessor versteht

● **Der Mov-Befehl:** Von diesen Befehlen ist der Befehl »mov« zweifellos der wichtigste. Mit »mov« lassen sich Register und Speicherzellen auslesen und laden. Kein Programm kommt ohne den Mov-Befehl aus. Der Befehl

```
mov bx,1
```

setzt beispielsweise das BX-Register auf 1. Der Befehl

```
mov flag,c1
```

schreibt in die durch den Wert der Marke »flag« bestimmte Speicherzelle den Wert des CL-Registers. Der Befehl

```
mov dx,offset(text)
```

lädt die Adresse der Variablen »text« in das DX-Register. Ohne die Angabe »offset« würde der Prozessor den Inhalt der Variablen »text« in das DX-Register schreiben.

● **Die Befehle »call« und »ret«:** 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«) kehren Sie aus dem Unter-

programm 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 in das Hauptprogramm verwenden kann. Mit

```
...
call sub
...
end
;Unterprogramm
sub: xor ax,ax
xor bx,bx
xor cx,cx
xor dx,dx
ret
```

setzt das Unterprogramm »sub« durch eine schnelle Exklusiv-Oder-Verknüpfung die Register AX, BX, CX und DX auf 0 zurück.

● **Der Jmp-Befehl:** Selten kommt es vor, daß ein Programm ausschließlich linear verläuft. Immer wieder sind im Programmfluß Verzweigungen notwendig. Hierfür existieren beim 8086/8088 die Sprungbefehle (Tabelle 3). Der Befehl »jmp« (englisch »jump« = springen), den Sie bereits aus den Programmbeispielen kennen, ist ein unbedingter (»absoluter«) Sprungbefehl und entspricht dem Goto-Befehl in Basic-Programmen. Dem Befehl »jmp« muß eine Sprungmarke (»label«) folgen, damit der Assembler weiß, auf welche Adresse sich dieser Befehl bezieht. Die Zieladresse des Sprungbefehls im Assembler-Programm muß in der ersten Spalte mit dieser Sprungmarke und einem angehängten Doppelpunkt versehen sein. Mit

```
jmp weiter
text db "Dummy-Text"
weiter: mov ax,offset(text)
```

überspringt das Programm die Definition der Variablen »text« und fährt mit dem Mov-Befehl fort.

● **Bedingte Sprünge:** 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 Angabe des Sprungziels erfolgt bei den bedingten Sprungbefehlen wie beim Befehl »jmp« über Sprungmarken.

Arithmetische Kommandos, Vergleiche und einige andere Befehle bestimmen die Voraussetzungen für die bedingten Verzweigungen, indem Sie das Ergebnis Ihrer Aktion in den Statusbits (»flags«) des sogenannten Kennzeichenregisters (Bild 3) ablegen. Bild 1 zeigt das Standard-Kennzeichenregister mit den 80386-Erweiterungen. Die Operationen setzen die Statusbits dem Zustand des Prozessors entsprechend. Bild 2 listet die Bedeutung der einzelnen Statusbits auf.

Die Statusbits melden die folgenden Prozessorzustände:

- Einzelschrittmodus ist aktiv/nicht aktiv (»trace flag«)
- Ergebnis lautet 0/nicht 0 (»zero flag«)
- Parität ist gerade/ungerade (»parity flag«)
- Richtung ist vorwärts/rückwärts (»direction flag«)
- Überlauf ist aufgetreten/nicht aufgetreten (»overflow flag«)
- Übertrag ist aufgetreten/nicht aufgetreten (»carry flag« und »auxiliary carry flag«)
- Unterbrechung ist aufgetreten/nicht aufgetreten (»interrupt flag«)
- Vorzeichen ist negativ/positiv (»sign flag«)

Für Vergleiche, die die Statusbits (Bild 4) beeinflussen, ist der Befehl »cmp« (englisch »compare« = vergleichen) vorgesehen, der zwei Register oder ein

jnb: verzweigt den Programmablauf, wenn bei einem Vergleich der Art »cmp x,y« x größer oder gleich y ist.

je: verzweigt den Programmablauf, wenn bei einem Vergleich der Art »cmp x,y« x gleich y ist.

jne: verzweigt den Programmablauf, wenn bei einem Vergleich der Art »cmp x,y« x ungleich y ist.

jo: verzweigt, wenn vorher das Overflow-Flag (Überlauf) durch eine andere Operation gesetzt worden ist.

jno: verzweigt, wenn das Overflow-Flag (Überlauf) nicht gesetzt ist.

In Tabelle 3 bekommen Sie einen Überblick über die verschiedenen Sprungbefehle, die in Assembler definiert sind. Mit diesen Anweisungen halten Sie mächtige Werkzeuge zur alternativen Fortentwicklung eines Programms in Händen.

So manches Hilfsprogramm aus alten MS-DOS-Versionen würde man gerne

Die Flags im Kennzeichenregister		
Bit	Abkürzung	Name
0	CF	Carry-Kennzeichen (Übertrag) (Flag)
1		Reserviert
2	PF	Paritäts-Kennzeichen
3		Reserviert
4	AF	Hilfs-Carry-Kennzeichen
5		Reserviert
6	ZF	Null-Kennzeichen
7	SF	Vorzeichen-Kennzeichen
8	TF	Trap-Kennzeichen (Einzelschrittmodus)
9	IF	Interrupt-Freigabe (Unterbrechung freigeben)
10	DF	Richtungs-Kennzeichen (Zeichenkettenbefehle)
11	OF	Überlauf
12-13	IOPL	E/A-Privilegierungsebene (Level)
14	NT	Prozeßschachtelungs-Kennzeichen
15		Reserviert
16	RF	Resume-Kennzeichen (nur 80386)
17	VM	Virtueller 8086-Modus (nur 80386)
18-31		Reserviert

Bild 4. Der 80386 weist gegenüber seinen Vorläufern einen erweiterten Statusbitsatz auf.

Register und einen Operanden miteinander vergleicht. Die Befehlsfolge

```
cmp bl,0
jz fehler
div bl
```

vergleicht beispielsweise den Inhalt des BL-Registers mit 0. Ist der Inhalt 0, so wird die Routine »Fehler« aufgerufen, im anderen Fall erfolgt die Division von AX durch BL.

Weitere Alternativen der bedingten Sprunganweisungen sind unter anderem:

ja, jb, je, jo, js

und ihre jeweiligen Umkehrungen

jna, jnb, jne, jno, jns.

Dabei besitzen diese Sprungbefehle folgende Bedeutung:

ja: verzweigt den Programmablauf, wenn bei einem Vergleich der Art »cmp x,y« x größer als y ist.

jna: verzweigt den Programmablauf, wenn bei einem Vergleich der Art »cmp x,y« x kleiner oder gleich y ist.

jb: verzweigt den Programmablauf, wenn bei einem Vergleich der Art »cmp x,y« x kleiner als y ist.

in die neuere Version übernehmen. Rufen Sie es jedoch unter der neuen MS-DOS-Version auf, so kommt die Fehlermeldung »Falsche MS-DOS-Version«.

Wie Sie MS-DOS einfach überlisten

Ihr Programm kann während des Programmablaufs je nach Ergebnis vorheriger Operationen den Sprung in andere Routinen ausführen.

Ein Beispiel dazu liefert die Versionsabfrage in MS-DOS. Gehen Sie davon aus, daß Ihr Utility unter MS-DOS 5.0 arbeitet und Sie es in das neue MS-DOS 6.0 übernehmen wollen. Starten Sie das Programm unter MS-DOS 6.0, geschieht folgendes. Zuerst nimmt das Programm die übliche Versionsabfrage vor:

```
mov ah, 30h
int 21h.
```

Die Funktionsnummer 30hex wird mit dem Mov-Befehl in das AH-Register

abgelegt und der Interrupt 21hex mit »int 21h« aufgerufen. Die Funktion 30hex des Interrupts 21hex hat den Zweck, Ihnen die DOS-Version zurückzugeben. Hinter dem Aufruf von Funktionen und Interrupts verbirgt sich also kein großes Geheimnis. Nach dem Aufruf der Funktion 30hex des Interrupts 21hex finden Sie das Ergebnis im Register AX. Dabei wird das Ergebnis folgendermaßen abgelegt: Im AH-Register wird die Versionszahl nach dem Komma abgelegt – AH hat also den Wert 00 –, und im AL-Register finden Sie die Versionszahl, die vor dem Komma steht, das heißt, daß AL den Wert 6 hat.

Nun folgt ein Vergleich, mit dem das kleine Utility das Ergebnis überprüft:

```
cmp ax,0005.
```

Hier wird mit dem Cmp-Befehl der Inhalt des AX-Registers – in unserem Beispiel die Zahl 0006, also Version 06.00 – mit der Zahl 0005, also Version 05.00, verglichen, da unser Programm schließlich aus dem Paket von MS-DOS 5 stammt.

Das Ergebnis ist in diesem Falle natürlich nicht gleich (»not equal«).

Die englische Übersetzung erleichtert Ihnen das Verständnis des nächsten Schritts: den bedingten Sprung.

Das kleine Programm wird jetzt nämlich entscheiden, ob es unter dieser anderen DOS-Version laufen will oder nicht:

```
jne XXXX
```

XXXX steht dabei für eine beliebige Sprungadresse. Mit diesem Befehl sagt Ihnen das Hilfsprogramm:

Springe, wenn nicht gleich nach XXXX.

Auf englisch: Jump if Not Equal to XXXX.

In unserem Beispiel springt das Hilfsprogramm an eine Adresse, an der die Fehlermeldung »Falsche DOS-Version« auf dem Bildschirm ausgegeben wird und das Programm abbricht.

Dieses Verhalten ändern Sie mit einem Hexeditor, wie er bei den Norton-Utilities oder bei PC-Tools mitgeliefert wird:

Sie suchen in dem Hilfsprogramm einfach nach der Hexadezimal-Zahlenkette

```
B4 30 CD 21 3D 05 00
```

Dies sind die Zahlen, die ein Assembler aus den Assembler-Programmen macht. Einen Überblick dazu gibt Ihnen Tabelle 4.

Ändern Sie jetzt diese Zahlenkette in die Kette

```
B4 30 CD 21 3D 06 00
```

um und speichern Sie das Programm. Mit dieser Änderung führt das Programm beim nächsten Aufruf statt

```
cmp ax,0005
```

den Vergleich

```
cmp ax,0006
```

aus. Ihr Hilfsprogramm startet ab jetzt (nur) unter MS-DOS 6.0.

Möchten Sie Ihr Programm auf eine andere Version von MS-DOS exportieren, passen Sie diese Zahlenkette einfach an die jeweilige MS-DOS-Version an. Welche Zahlenkette welcher MS-DOS-Version entspricht, beschreibt auszugsweise Tabelle 5. Abweichungen tauchen allerdings bei einigen OEM-Versionen auf, sind aber nicht sehr häufig. Zu beachten ist auch die Besonderheit, daß MS-DOS 4.01 sich als Version 4.00 zu erkennen gibt.

Aus den Kinderschuhen wachsen

Der Überblick über die wichtigsten Assembler-Routinen bildet die Grundlage, die Sie zum Verständnis der Arbeitsweise von MS-DOS-Funktionen benötigen. MS-DOS-Funktionen und die sogenannten BIOS-Funktionen (Basic Input Output System) ersparen dem Programmierer viel Arbeit. Sie erfüllen Aufgaben, deren Programmierung viel Zeit beanspruchen würde. Wenn Sie zum Beispiel eine Zeichenkette auf dem Bildschirm ausgeben wollen, müssen Sie nicht umständlich und arbeitsintensiv die Chips der Bildschirmkarte ansteuern. Sie laden einfach ein Register mit der Adresse der Speicherstelle, an der die auszugebende Zeichenkette abgelegt ist, stimmen mit dem AX-Register die Nummer der Ausgabefunktion und rufen die Funktion auf.

Aufruf von Interrupts und Funktionen

Alle MS-DOS-Funktionen (und auch die BIOS-Funktionen) lassen sich über programmierbare Unterbrechungen, die als Software-Interrupts bezeichnet werden, aufrufen. Der 8086/8088-Prozessor kann maximal 256 verschiedene Software-Interrupts mit den Nummern 00hex bis FFhex erzeugen. Für die MS-DOS-Funktionen ist der Software-Interrupt 21hex zuständig. Der Assembler-Befehl zum Aufruf dieses Interrupts lautet

```
int 21h
```

Der Int-Befehl spielt beim PC eine äußerst wichtige Rolle, weil diese Funktionen direkt auf die Routinen des Betriebssystems und des BIOS zugreifen und deren Fähigkeiten nutzen.

Durch das Konzept der Software-Interrupts müssen Sie diese Routinen nicht über absolute Adressen ansprechen (die sich dann von MS-DOS- zu MS-DOS-Version und von BIOS- zu BIOS-Version ändern würden), sondern Sie rufen sie über Software-Interrupts mit standardisierten Nummern auf.

Mit dem Befehl »int« wird ein Wert im Bereich von 00hex bis FFhex übergeben, der die Nummer des gewünschten Software-Interrupts bezeichnet. Zu jedem dieser Interrupts befindet sich im ersten KByte des PC-Speichers eine 4 Byte lange Adresse. Der Prozessor sucht sich selbständig die zu dem Interrupt passende Adresse (auch »Interrupt-Vektor« genannt) und ruft diese Adresse und damit die zu dem Interrupt gehörende Serviceroutine auf. Wenn der Software-Interrupt beendet ist, kehrt der Prozessor automatisch zu dem Befehl im aufrufenden Programm zurück, der dem Int-Befehl folgt.

● **Aufruf einzelner Funktionen:** Die meisten Software-Interrupts sind für einen bestimmten Bereich (Bildschirm, Datenträger, Tastatur und so weiter) zuständig. Weil nun zu jedem dieser Bereiche mehrere Betriebssystem- und BIOS-Routinen existieren, der Anwender jedoch immer nur eine bestimmte Routine oder Routinengruppe aufrufen möchte, sind diese einfach bei 0 beginnend durchnummeriert. Jede Routine oder Routinengruppe, der eine eigene Nummer zugewiesen ist, wird als Funktion bezeichnet. Um nun eine bestimmte Funktion aufzurufen, muß vor dem Int-Befehl die Nummer der gewünschten Funktion in das AH-Register geschrieben werden.

Damit auch die einzelnen Routinen einer Routinengruppe angesprochen werden können, sind diese oft ebenfalls bei 0 beginnend durchnummeriert. Um die Routine einer Routinengruppe, auch Unterfunktion genannt, aufzurufen, schreiben Sie vor dem Int-Befehl zusätzlich die Nummer der Unterfunktion in das AL-Register.

Benötigt eine Funktion oder Unterfunktion die Eingabe von Parametern oder gibt sie Parameter zurück (eines von beiden ist fast immer der Fall), so sind für diese Parameter bestimmte Register speziell reserviert. In vielen Fachbüchern wie beispielsweise der »PC-Referenz für Programmierer« aus dem Systema-Verlag sind die Software-Interrupts des PCs mit allen Funktionen und Unterfunktionen sowie den für die Parameter reservierten Registern aufgeführt. Auch die Aufgabe und Bedeutung der Parameter wird erklärt.

Beispiele zu Interrupt-Aufrufen

Ein typisches Beispiel für die Anwendung der Betriebssystem- und BIOS-Routinen ist der Aufruf der Funktion 00hex des Video-Interrupts 10hex. Diese Funktion stellt den Videomodus (Text- oder Grafikmodus) des verwen-

Wettbewerb

Monat für Monat 1024 Mark für die besten 1024 Byte

Programmieren Sie in Assembler, Basic, C oder Pascal? Haben Sie ein oder mehrere Programme geschrieben, die das eine oder andere Problem auf geniale Weise lösen? Dann zögern Sie nicht, und senden Sie den Quelltext und die lauffähige Version auf 5 1/4- oder 3 1/2-Zoll Diskette, zusammen mit Ihrer kompletten

Anschrift und Angaben über den verwendeten Compiler, an die

**Redaktion DOS International
»1024«
Gruberstr. 46 c
8011 Poing bei München**

Jeden Monat prämiere wir das beste eingegangene Listing mit 1024 Mark. Die einzige Bedingung: Der Quelltext darf nicht länger als 1024 Byte (1 KByte) sein. Nicht prämierte Listings werden bei Veröffentlichung nach den üblichen Sätzen honoriert.

Per Anhalter durch die Fraktale

Noch nie war es so aufregend, fraktale Grafiken zu erzeugen. Noch nie war es so einfach, spannende Computer-Animationen zu produzieren. Mit vollautomatischer Berechnung fliegen Sie durch phantastische dreidimensionale Landschaften. Stellen Sie mit dem integrierten Slideshow-Generator herrliche Bilder zu einer eleganten Non-Stop-Filmpräsentation zusammen – der Publikumserfolg ist Ihnen sicher.

Fraktal-Animator 3D

69,^{DM*}

- Rasend schnelle Bildberechnung
- Zwei Animationsverfahren
- 3D- und Kugelprojektion
- Bis zu 50 Bilder/Sekunde!!
- 32-Bit- und Coprozessor-fähig
- Bildgröße bis 800 x 600 Punkte
- PCX-Bildexport
- Diashow- und Filmerstellung

Für PCs mit Festplatte und EGA/VGA-Karte

Navigator 4.5

Der Lotse für den MS-Flugsimulator III und IV

69,— DM*

Händleranfragen erwünscht.

DMV-Verlag • Postfach 250 • W-3440 Eschwege

Telefon: (0 56 51) 809-0 • Telefax: (0 56 51) 809-430



* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung.

Bitte benutzen Sie die Bestellkarte im Heft.

deten Bildschirmadapters ein. So ruft beispielsweise die Befehlsfolge

```
mov al,07h
mov ah,00h
int 10h
```

den Videomodus 7 des Monochromadapters (MDA = Monitor Display Adapter) auf. In diesem Beispiel ist der Videomodus ein Eingabeparameter und wird im AL-Register übergeben.

Um Platz zu sparen, kann man das Beispiel auch kürzer schreiben, weil sich die Register AL und AH zum AX-Register zusammenfassen lassen:

```
mov ax,0007h
int 10h
```

Welche Version Sie wählen, bleibt Ihnen überlassen. Beachten Sie allerdings, daß Assembler-Programme aufgrund der komplexen Befehlsfolgen gut strukturiert sein sollten. Deswegen erscheint es ratsam, die erste Variante zu wählen.

● **Ausgabe von Zeichen:** Ein weiteres Beispiel für die Nutzung einer Betriebssystemroutine ist der Aufruf der MS-DOS-Funktion 02hex über den Software-Interrupt 21hex. Die MS-DOS-Funktion 02hex gibt ein einzelnes ASCII-Zeichen auf dem Bildschirm beziehungsweise dem aktuellen Ausgabegerät aus. Der Code des ASCII-Zeichens muß sich als Eingabeparameter im DL-Register befinden. Die folgende Befehlsfolge gibt den Buchstaben »A« auf den Bildschirm aus:

```
mov dl,41h
mov ah,02h
int 21h
```

Die Funktionen der Software-Interrupts des PCs reichen zum Programmieren vieler 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.

● **Nachteile von Interrupts:** Wenn Sie allerdings die Geschwindigkeit der MS-DOS- und BIOS-Funktionen nicht als ausreichend ansehen (beispielsweise bei der Bildschirmausgabe), ist es sinnvoll, eigene Routinen zu schreiben, die direkt auf Bildschirmspeicher, Tastatur oder Schnittstellen zugreifen. Dieses Verfahren kostet allerdings Zeit und setzt fundierte Kenntnisse der Computerhardware und der Arbeitsweise von Betriebssystem und BIOS voraus.

● **Einlesen von Text über Tastatur:** Bereits erfahren haben Sie, wie sich in Assembler-Sprache Text auf den Bildschirm ausgeben läßt; lernen Sie nun, wie Sie Text über die Tastatur einlesen.

Ihr nächstes Assembler-Programm soll dem Basic-Befehl

```
input "Bitte Namen eingeben: ";name$
```

entsprechen, also zuerst den Text »Bitte Namen eingeben: « ausgeben und daraufhin einen Text von der Tastatur lesen, der in einer Variablen gespeichert wird.

1. Schritt: Um den Text auszugeben, benutzen Sie, wie schon besprochen, den Software-Interrupt 21hex mit der MS-DOS-Funktion 09hex. Im DX-Register übergeben Sie die Startadresse der auszugebenden Zeichenkette. Das Ende der Zeichenkette markieren Sie mit einem Dollarzeichen. Im Programmbereich muß also

```
mov dx,offset(text)
mov ah,09h
int 21h
```

und im Datenbereich

```
text db "Bitte Namen eingeben: "$
```

stehen.

2. Schritt: Für die Eingabe von Text eignet sich die Funktion 0Ahex des Interrupts 21hex. Diese Funktion läßt sich wie die Funktion 09hex aufrufen, nur enthält das DX-Register diesmal die Adresse eines Puffers, in dem der eingelesene Text abgelegt wird. Die Funktion 0Ahex speichert alle über die Tastatur eingegebenen Zeichen ab der durch das DX-Register definierten Adresse. Das Drücken der Eingabetaste beendet die Eingabe, [Strg-C] bricht sie ab.

Ein großer Vorteil der Funktion 0Ahex ist, daß alle eingegebenen Zeichen auf dem Bildschirm erscheinen und die Zeichenkette bis zum Drücken der Eingabetaste oder [Strg-C] frei editiert werden kann.

Der Eingabepuffer für die Funktion 0Ahex weist folgende Struktur auf:

1. Byte: Anzahl der maximal folgenden Zeichen (legt die maximale Länge der Zeichenkette -1 fest)
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 das zweite Byte bereits für die Anzahl der tatsächlich eingegebenen Zeichen reserviert ist.

Listing 4 zeigt das komplette Assembler-Programm »Name«. Das Programm arbeitet mit dem Speichermodell für Com-Dateien. Beachten Sie dabei, daß Sie in dem Beispielprogramm die Programmzeile

```
db 21 dup (?)
```

durch

```
ds 21,0
```

ersetzen, wenn Sie es mit SAM 4 assemblieren.

Ihnen ist sicherlich aufgefallen, daß die Befehlsfolge

```
mov dx,offset(adresse)
mov ah,nummer
int 21h
```

zum Aufruf einer Funktion des Interrupts 21hex zweimal im Programm »Name« auftaucht. Der Unterschied besteht allein darin, daß für die Zeichenkettenadresse »adresse« und die Funktionsnummer »nummer« unterschiedliche Werte eingesetzt sind.

Makros – die Hilfe für Programmierer

In längeren Programmen kommt diese Befehlsfolge noch weitaus häufiger zum Einsatz, doch muß man sie deshalb nicht jedesmal neu eingeben. Sowohl MASM, der Microsoft-Makro-Assembler, als auch SAM 4 sind in der Lage, Makros zu verwalten. Makros sind immer wiederkehrende Befehlsfolgen, die der Programmierer bei der Arbeit mit einem Makro-Assembler als Makro definieren kann und dadurch nur ein einziges Mal schreiben muß. Wenn er die in einem Makro enthaltenen Befehle an einer bestimmten Stelle in seinem Programm benötigt, setzt er lediglich den Namen des Makros ein. Der Makro-Assembler fügt dann beim Übersetzen des Programms in

Auflösung:

dezimal 5	= binär 101
dezimal 36	= binär 100100
dezimal 158	= binär 10011110
binär 01001001	= dezimal 73
binär 10101010	= dezimal 170
binär 10110111	= dezimal 183
dezimal 18	= oktal 22
dezimal 47	= oktal 57
dezimal 916	= oktal 1624
dezimal 255	= hexadezimal FF
dezimal 616	= hexadezimal 268
dezimal 7654	= hexadezimal 1DE6

Maschinencode den Inhalt des Makros automatisch an den entsprechenden Stellen ein.

Anstelle eines Makros könnten Sie auch mit einem Unterprogramm arbeiten. Doch im Gegensatz zum Makro muß das Unterprogramm jedesmal zeitraubend über den Befehl »call« aufgerufen und mit dem Befehl »ret« beendet werden, während das Makro direkt an der gewünschten Stelle im Programm vorhanden ist und den Programmlauf nicht bremst.

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. Bei langen Makros,

die oft verwendet werden, kann dadurch ein Programm um ein ganzes Stück anwachsen.

● **Vor- und Nachteile:** Legen Sie also Wert auf Geschwindigkeit, sind Makros geeignet. Wenn Sie dagegen Speicherplatz sparen wollen und die Ausführungszeit eine untergeordnete Rolle spielt, greifen Sie besser auf Unterprogramme zurück.

● **Makros beim MASM:** Makroaufbau und -aufruf unterscheiden sich bei MASM und bei SAM 4. Das Makro für den oben angegebenen Aufruf einer Funktion des Interrupts 21hex lautet beim MASM:

```
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.

● **Makros beim SAM 4:** SAM 4 erwartet eine etwas strenger definierte Form des Makros. Als Parameter sind nur die Zahlen 1 bis 9 mit vorangestelltem Prozentzeichen erlaubt (»%«).

Das oben angegebene Makro lautet unter SAM 4:

```
funktion macro %1,%2
mov dx,offset(%2)
mov ah,%1
int 21h
endm
```

Wenn Sie das Makro zum Beginn von Programm »Name« definieren, können Sie für den Aufruf der Funktion 09hex in Listing 4 unter MASM

```
funktion 09h,text
und für den Aufruf der Funktion 0Ah,max
funktion 0Ah,max
schreiben.
```

Für SAM 4 lauten die Kommandos

```
do funktion,09h,text
do funktion,0Ah,max
```

Die Assembler fügen beim Übersetzen des Programms in Maschinencode anstelle der beiden aufgeführten Zeilen jeweils das Makro »funktion« ein und ordnen im ersten Fall dem Parameter »nummer« den Wert 09hex und dem Pa-

rameter »adresse« die Variable »text« zu. Im zweiten Fall wird »nummer« auf »0Ah« 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 gleichen Reihenfolge übergeben werden.

Listing 5 zeigt, wie Sie das Programm »Name« für den Einsatz des beschriebenen Makros unter MASM ändern müssen; Listing 6 stellt die SAM-4-Version dar.

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 neuer Programme viel Zeit sparen helfen (ein Beispiel finden Sie im nächsten Beitrag »Der Rechenschieber«). Wenn Sie umfangreiche Makro-Bibliotheken benutzen, wird das Programmieren in Assembler sogar fast so komfortabel wie das Schreiben von Programmen in Hochsprachen.

(Markus Zietlow/st)

```
[ZL3] 1: ; Programmname: print
[NA2] 2: ; Sprache: SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[7D2] 6: ; title print
[Y26] 7: ; -Dieses Programm gibt auf dem Bildschirm
[NP6] 8: ; eine Zeichenkette aus und kehrt darauf
[R46] 9: ; in die MS-DOS-Kommandoebene zurück
[P30] 10: ;
[P30] 11: ;
[PM5] 12: ; -Segment code ist für die Segment-
[R75] 13: ; register CS und DS vorgesehen
[EZ3] 14: ; assume cs:code,ds:code
[P30] 15: ;
[P30] 16: ;
[084] 17: ; -Start von Segment code
[X34] 18: ; code segment para 'code'
[LW5] 19: ; -Programm startet ab 100hex (für COM-
[9V3] 20: ; Dateien erforderlich)
[WP1] 21: ; org 100h
[9X4] 22: ; -Sprung zum Programmstart
[ZB3] 23: ; start: jmp anfang
[P30] 24: ;
[P30] 25: ;
[UB2] 26: ; -Datenbereich
[P30] 27: ;
[P30] 28: ;
[186] 29: ; text db 13,10,'Ein Assemblerprogramm '
[YV4] 30: ; db 'für MASM und SAM-4',13,10,'$'
[P30] 31: ;
[P30] 32: ;
[3Y2] 33: ; -Programmbereich
[P30] 34: ;
[P30] 35: ;
[3B4] 36: ; -Adresse der Zeichenkette
[P30] 37: ;
[FR4] 38: ; anfang: mov dx,offset(text)
[P30] 39: ;
[P30] 40: ;
[P66] 41: ; -Zeichenkette über Interrupt 21hex,
[SU5] 42: ; Funktion 09hex auf dem Bildschirm
[ML1] 43: ; ausgeben
[P30] 44: ;
[Z12] 45: ; mov ah,09h
[NK1] 46: ; int 21h
[P30] 47: ;
[P30] 48: ;
[WG5] 49: ; -Programm über Interrupt 21hex,
[8W3] 50: ; Funktion 4ch beenden
[P30] 51: ;
[L32] 52: ; mov ah,4ch
[NK1] 53: ; int 21h
```

```
[P30] 54: ;
[P30] 55: ;
[SM3] 56: ; -Ende von Segment code
[P30] 57: ;
[TT1] 58: ; code ends
[P30] 59: ;
[P30] 60: ;
[HT4] 61: ; -Programmende und Definition
[043] 62: ; des Startpunktes
[C72] 63: ; end start
```

Listing 1. Das Beispielprogramm »print.sam« läßt sich sowohl mit SAM 4 als auch mit MASM assemblieren.

```
[EA7] 1: ; Programm: Programmgerüst für Com-Dateien
[NA2] 2: ; Sprache: SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[596] 6: ; -Segmentzuweisungen für CS und DS
[P30] 7: ;
[EZ3] 8: ; assume cs:code,ds:code
[P30] 9: ;
[P30] 10: ;
[MZ4] 11: ; -Start des Speichersegments
[P30] 12: ;
[X34] 13: ; code segment para 'code'
[P30] 14: ;
[P30] 15: ;
[9D4] 16: ; -Programm startet ab 100hex
[P30] 17: ;
[WP1] 18: ; org 100h
[P30] 19: ;
[P30] 20: ;
[9X4] 21: ; -Sprung zum Programmstart
[P30] 22: ;
[ZB3] 23: ; start: jmp anfang
[P30] 24: ;
[P30] 25: ;
[UB2] 26: ; -Datenbereich
```

Listing 2. Den grundlegenden Aufbau von Assembler-Programmen, die mit dem Speichermodell der Com-Dateien arbeiten, zeigt »com.sam«


```
[631] 27: ; ...
[P30] 28: ;
[P30] 29: ;
[3Y2] 30: ;--Programmbereich
[2L4] 31: ;--Erster Befehl im Programm
[P30] 32: ;
[4T1] 33: anfang: ;...
[6F0] 34: ;...
[5U4] 35: ;--Letzter Befehl im Programm
[6F0] 36: ;...
[P30] 37: ;
[P30] 38: ;
[WG5] 39: ;--Programm über Interrupt 21hex,
[8W3] 40: ; Funktion 4chex beenden
[P30] 41: ;
[L32] 42: mov ah,4ch
[NK1] 43: int 21h
[P30] 44: ;
[P30] 45: ;
[WK4] 46: ;--Ende des Speichersegments
[P30] 47: ;
[TT1] 48: code ends
[P30] 49: ;
[P30] 50: ;
[HT4] 51: ;--Programmende und Definition
[043] 52: ; des Startpunktes
[P30] 53: ;
[C72] 54: end start
```

Listing 2. Den grundlegenden Aufbau von Assembler-Programmen, die mit dem Speichermodell der Com-Dateien arbeiten, zeigt »com.sam«

```
[AA7] 1: ; Programm: Programmgerüst für Exe-Dateien
[NA2] 2: ; Sprache: SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[N66] 5: ;--Für MASM müssen Sie die Anweisungen
[CR4] 6: ; "ds xxx,0" durch "db xxx dup (?)"
[FP1] 7: ; ersetzen
[P30] 8: ;
[P30] 9: ;
[ZT1] 10: title exe
[P30] 11: ;
[P30] 12: ;
[ZM3] 13: ;--Segmentzuweisungen
[P30] 14: ;
[TV6] 15: assume cs:code,ds:data0,es:extra,ss:stack
[P30] 16: ;
[P30] 17: ;
[MK3] 18: ;--Erstes Datensegment
[P30] 19: ;
[P54] 20: data0 segment para 'data'
[GT0] 21: ;-a=10
[7Z1] 22: a db 10
[RT0] 23: ;-b=11
[B21] 24: b db 11
[YT0] 25: ;-c=12
[FZ1] 26: c db 12
[DV1] 27: data0 ends
[P30] 28: ;
[P30] 29: ;
[ZV3] 30: ;--Zweites Datensegment
[P30] 31: ;
[S54] 32: data1 segment para 'data'
[SU0] 33: ;-d=13
[LZ1] 34: d db 13
[CU0] 35: ;-e=14
[RZ1] 36: e db 14
[LU0] 37: ;-f=15
[VZ1] 38: f db 15
[FV1] 39: data1 ends
[P30] 40: ;
[P30] 41: ;
[2Y2] 42: ;--Extrasegment
[P30] 43: ;
[G94] 44: extra segment para 'data'
[CL3] 45: ;--Platz für drei
[G13] 46: ; maximal 256 Bytes
[FL3] 47: ; lange Zeichenketten
[973] 48: string1 db 100h,0
[D73] 49: string2 db 100h,0
[H73] 50: string3 db 100h,0
[7Z1] 51: extra ends
[P30] 52: ;
[P30] 53: ;
[R13] 54: ;--Stapelsegment
[P30] 55: ;
[FR4] 56: stack segment para 'stack'
```

```
[W45] 57: ;--Ein KByte für Systemstapel
[082] 58: ; reservieren
[5X2] 59: tos db 400h,0
[DE5] 60: ;--Ein KByte für Programmstapel
[082] 61: ; reservieren
[1V2] 62: tob db 400h,0
[GX1] 63: stack ends
[P30] 64: ;
[P30] 65: ;
[P30] 66: ;
[DT2] 67: ; --Codesegment
[P30] 68: ;
[X34] 69: code segment para 'code'
[P30] 70: ;
[P30] 71: ;
[7F3] 72: ;--DS mit Datensegment
[FM3] 73: ; data0 initialisieren
[P30] 74: ;
[SR3] 75: start: mov ax,data0
[X52] 76: mov ds,ax
[P30] 77: ;
[KB3] 78: ;--ES initialisieren
[XL2] 79: mov ax,extra
[162] 80: mov es,ax
[P30] 81: ;
[AD3] 82: ;--SS initialisieren
[4M2] 83: mov ax,stack
[T72] 84: mov ss,ax
[P30] 85: ;
[7D3] 86: ;--SP initialisieren
[NS3] 87: mov sp,offset(tos)
[P30] 88: ;
[3B3] 89: ;--BP initialisieren
[DM3] 90: mov bp,offset(tob)
[6F0] 91: ;...
[US2] 92: ;--Weitere Befehle
[6F0] 93: ;...
[P30] 94: ;
[P30] 95: ;
[WG5] 96: ;--Programm über Interrupt 21hex,
[8W3] 97: ; Funktion 4chex beenden
[P30] 98: ;
[L32] 99: mov ah,4ch
[NK1] 100: int 21h
[P30] 101: ;
[P30] 102: ;
[A54] 103: ;--Ende von Segment mcode'
[P30] 104: ;
[TT1] 105: code ends
[P30] 106: ;
[P30] 107: ;
[HT4] 108: ;--Programmende und Definition
[043] 109: ; des Startpunktes
[P30] 110: ;
[C72] 111: end start
```

Listing 3. Assembler-Programme, die mit dem Speichermodell für Exe-Dateien wie »exe.sam« arbeiten, sind im Aufbau komplexer als Com-Dateien.

```
[NM6] 1: ; Programm: Dieses Programm gibt auf dem
[F76] 2: ; Bildschirm eine Zeichenkette aus und
[VV4] 3: ; liest eine Zeichenkette ein
[NA2] 4: ; Sprache: SAM 4
[R86] 5: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 6: ;
[MZ5] 7: ;--Für MASM müssen Sie die Anweisung
[6S5] 8: ; "ds 21,0" durch "db 21 dup (?)" ersetzen
[P30] 9: ;
[P30] 10: ;
[T93] 11: ;--Programmname name
[8D2] 12: title inout
[P30] 13: ;
[P30] 14: ;
[Z44] 15: ;--Segment code ist für die
[8L6] 16: ; Segmentregister CS und DS vorgesehen
[P30] 17: ;
[EZ3] 18: assume cs:code,ds:code
[P30] 19: ;
[P30] 20: ;
[084] 21: ;--Start von Segment code
[P30] 22: ;
[X34] 23: code segment para 'code'
[P30] 24: ;
[P30] 25: ;
[R55] 26: ;--Programm startet ab 100hex (für
[5F4] 27: ; COM-Dateien erforderlich)
[P30] 28: ;
[WP1] 29: org 100h
[P30] 30: ;
[P30] 31: ;
[9X4] 32: ;--Sprung zum Programmanfang
```



```
[P30] 33: ;
[ZB3] 34: start: jmp anfang
[P30] 35: ;
[P30] 36: ;
[UB2] 37: ;-Datenbereich
[P30] 38: ;
[UA5] 39: text db 13,10,'Bitte geben Sie '
[1W3] 40: db 'Ihren Namen ein: $'
[P30] 41: ;
[DC5] 42: ;-Erstes Byte des Eingabepuffers
[EN2] 43: ; (Pufferlänge)
[842] 44: max db 21
[P30] 45: ;
[8P2] 46: ;-Eingabepuffer
[671] 47: db 21,0
[P30] 48: ;
[3Y2] 49: ;-Programmbereich
[3B4] 50: ;-Adresse der Zeichenkette
[FR4] 51: anfang: mov dx,offset(text)
[P30] 52: ;
[P66] 53: ;-Zeichenkette über Interrupt 21hex,
[SU5] 54: ; Funktion 09hex auf dem Bildschirm
[ML1] 55: ; ausgeben
[Z12] 56: mov ah,09h
[NK1] 57: int 21h
[P30] 58: ;
[VC3] 59: ;-Adresse des Puffers
[L83] 60: mov dx,offset(max)
[P30] 61: ;
[914] 62: ;-Zeichenkette einlesen
[Z22] 63: mov ah,0ah
[NK1] 64: int 21h
[P30] 65: ;
[WG5] 66: ;-Programm über Interrupt 21hex,
[8W3] 67: ; Funktion 4chex beenden
[L32] 68: mov ah,4ch
[NK1] 69: int 21h
[P30] 70: ;
[SM3] 71: ;-Ende von Segment code
[TT1] 72: code ends
[P30] 73: ;
[P30] 74: ;
[HT4] 75: ;-Programmende und Definition
[043] 76: ; des Startpunktes
[C72] 77: end start
```

Listing 4. Das Assembler-Programm »inout.sam« gibt eine Zeichenkette aus und liest eine Zeichenkette ein.

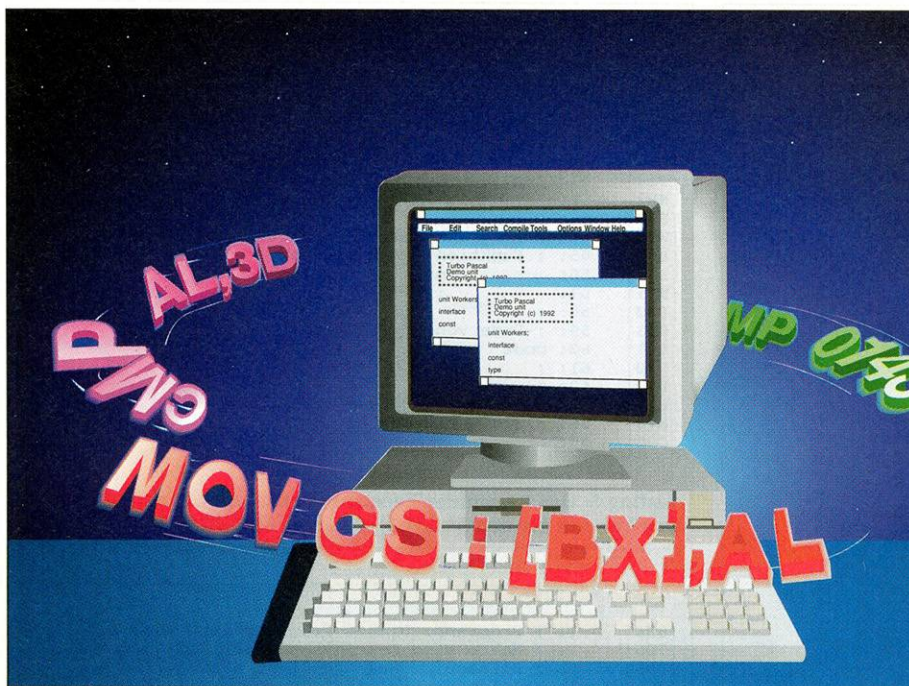
```
[3Z6] 1: ; Programm: Dieses Programm gibt über ein
[7D5] 2: ; Makro eine Zeichenkette auf dem
[385] 3: ; Bildschirm aus und liest eine
[0Z2] 4: ; Zeichenkette ein
[1F2] 5: ; Sprache: MASM
[R86] 6: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 7: ;
[P30] 8: ;
[6D2] 9: ;-MASM-Version
[3M2] 10: title makrout
[P30] 11: ;
[P30] 12: ;
[R43] 13: ;-Makrodefinition:
[5X5] 14: funktion macro funktion,adresse
[EC4] 15: mov dx,offset(adresse)
[S43] 16: mov ah,funktion
[NK1] 17: int 21h
[HP0] 18: endm
[P30] 19: ;
[P30] 20: ;
[Z44] 21: ;-Segment code ist für die
[8L6] 22: ; Segmentregister CS und DS vorgesehen
[EZ3] 23: assume cs:code,ds:code
[P30] 24: ;
[084] 25: ;-Start von Segment code
[X34] 26: code segment para 'code'
[P30] 27: ;
[R55] 28: ;-Programm startet ab 100hex (für
[5F4] 29: ; COM-Dateien erforderlich)
[WP1] 30: org 100h
[P30] 31: ;
[9X4] 32: ;-Sprung zum Programmstart
[ZB3] 33: start: jmp anfang
[P30] 34: ;
[P30] 35: ;
[UB2] 36: ;-Datenbereich
[UA5] 37: text db 13,10,'Bitte geben Sie '
[1W3] 38: db 'Ihren Namen ein: $'
[P30] 39: ;
[DC5] 40: ;-Erstes Byte des Eingabepuffers
[EN2] 41: ; (Pufferlänge)
[842] 42: max db 21
[P30] 43: ;
```

```
[8P2] 44: ;-Eingabepuffer
[SW1] 45: db 21 dup (?)
[P30] 46: ;
[3Y2] 47: ;-Programmbereich
[P04] 48: ;-Zeichenkette ausgeben
[LE4] 49: anfang: funktion 09h,text
[P30] 50: ;
[914] 51: ;-Zeichenkette einlesen
[NR2] 52: funktion 0ah,max
[P30] 53: ;
[WG5] 54: ;-Programm über Interrupt 21hex,
[8W3] 55: ; Funktion 4chex beenden
[L32] 56: mov ah,4ch
[NK1] 57: int 21h
[P30] 58: ;
[SM3] 59: ;-Ende von Segment code
[TT1] 60: code ends
[P30] 61: ;
[P30] 62: ;
[HT4] 63: ;-Programmende und Definition
[043] 64: ; des Startpunktes
[C72] 65: end start
```

Listing 5. Das Assembler-Programm »makrout.asm« ist die Makro-Version für den Makro-Assembler MASM vom Microsoft

```
[3Z6] 1: ; Programm: Dieses Programm gibt über ein
[7D5] 2: ; Makro eine Zeichenkette auf dem
[385] 3: ; Bildschirm aus und liest eine
[0Z2] 4: ; Zeichenkette ein
[NA2] 5: ; Sprache: SAM 4
[R86] 6: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 7: ;
[P30] 8: ;
[NL2] 9: ;-SAM-4-Version
[2W2] 10: title smakrout
[P30] 11: ;
[R43] 12: ;-Makrodefinition:
[BD3] 13: funktion macro %1,%2
[VA3] 14: mov dx,offset(%2)
[PX1] 15: mov ah,%1
[NK1] 16: int 21h
[HP0] 17: endm
[P30] 18: ;
[Z44] 19: ;-Segment code ist für die
[8L6] 20: ; Segmentregister CS und DS vorgesehen
[EZ3] 21: assume cs:code,ds:code
[P30] 22: ;
[084] 23: ;-Start von Segment code
[X34] 24: code segment para 'code'
[P30] 25: ;
[R55] 26: ;-Programm startet ab 100hex (für
[5F4] 27: ; COM-Dateien erforderlich)
[WP1] 28: org 100h
[P30] 29: ;
[9X4] 30: ;-Sprung zum Programmstart
[ZB3] 31: start: jmp anfang
[P30] 32: ;
[P30] 33: ;
[UB2] 34: ;-Datenbereich
[UA5] 35: text db 13,10,'Bitte geben Sie '
[1W3] 36: db 'Ihren Namen ein: $'
[P30] 37: ;
[DC5] 38: ;-Erstes Byte des Eingabepuffers
[EN2] 39: ; (Pufferlänge)
[842] 40: max db 21
[P30] 41: ;
[8P2] 42: ;-Eingabepuffer
[T81] 43: ds 21,0
[P30] 44: ;
[3Y2] 45: ;-Programmbereich
[P04] 46: ;-Zeichenkette ausgeben
[X35] 47: anfang: do funktion,09h,text
[P30] 48: ;
[914] 49: ;-Zeichenkette einlesen
[YM3] 50: do funktion,0ah,max
[P30] 51: ;
[WG5] 52: ;-Programm über Interrupt 21hex,
[8W3] 53: ; Funktion 4chex beenden
[L32] 54: mov ah,4ch
[NK1] 55: int 21h
[P30] 56: ;
[SM3] 57: ;-Ende von Segment code
[TT1] 58: code ends
[P30] 59: ;
[P30] 60: ;
[HT4] 61: ;-Programmende und Definition
[043] 62: ; des Startpunktes
[C72] 63: end start
```

Listing 6. Das Assembler-Programm »smakrout.sam« ist die Makro-Version für den Assembler SAM 4 aus dieser Ausgabe der DOS Extra.



Assembler und Mathematik

Der Rechenschieber

Wenn sich Programmierer mit den Prozessorbefehlen zufriedengeben müßten, würde die Realisierung längerer Assembler-Routinen in eine Qual ausarten. Um das Umsetzen von Standardfunktionen, wie der Wiederholung von Programmteilen, zu vereinfachen, stellen gute Assembler wie TASM, MASM oder SAM 4 zusätzliche Kommandos bereit, die sogenannten Pseudobefehle.

Da Sie die grundsätzliche Art der Programmentwicklung in Assembler und die Struktur eines Assembler-Programms kennengelernt haben, ist das Thema dieser Folge der Pseudobefehlssatz des Makro-Assemblers MASM von Microsoft und des DOS-Assemblers SAM 4.

Wie Sie bereits wissen, sind die Assembler-Befehle eine symbolische Darstellung der Maschinensprachebefehle des 8086/8088-Prozessors. Im Gegensatz dazu weisen die Pseudobefehle den Assembler an, wie er die bereitgestellten Daten verarbeiten und ausgeben soll. Das Präfix »Pseudo« (griechisch für »Schein«) wurde gewählt, weil es sich bei den Befehlen dieser Gruppe nicht um echte Prozessorbefehle, sondern um von Assembler-Programmierern entwickelte Hilfskommandos handelt.

Pseudobefehle werden hauptsächlich dazu benutzt, um Symbole zu definieren, Speicherbereiche zuzuweisen oder andere Arbeiten durchzuführen, die die Programmkonzeption und -umsetzung erleichtern. Im Gegensatz zu Assembler-Befehlen erzeugen Pseudobefehle jedoch in der Regel keinen Maschinencode.

Eine Anweisung, die einen Pseudobefehl enthält, besteht wie ein Assembler-Befehl aus einem Marken-, einem Befehls-, einem Operanden- und einem Kommentarfeld. Bis auf das Befehlsfeld sind alle Felder optional und müssen deshalb nicht belegt sein.

Genau wie bei Assembler-Befehlen können sich die Felder der Pseudobefehle an jeder Position innerhalb einer Zeile befinden. Sie müssen jedoch mindestens durch ein Leerzeichen

oder einen Tabulator voneinander getrennt sein.

Der MASM von Microsoft verfügt über mehr als 50 verschiedene Pseudobefehle, SAM 4 enthält eine Untermenge davon. Tabelle 4 zeigt die Pseudobefehle der beiden Assembler und deren unterschiedliche Form im Überblick.

● **Die Befehle »equ« und »=«:** Mit den Pseudobefehlen für die Symboldefinition sind Sie in der Lage, mathematischen Ausdrücken oder absoluten Zahlen symbolische Namen zuzuweisen. Die zu diesem Zweck benutzten Befehle »equ« und »=« (Tabelle 1) ähneln sich sehr. Der Unterschied liegt darin, daß die mit dem Gleichheitszeichen festgelegten Symbole undefiniert werden können. Zulässige Formen für die Symboldefinition sind beispielsweise:

```
akku equ ax
index equ [bp]
kbyte = 1024
```

● **Daten- und Konstantendefinitionen:** Die am häufigsten benötigten Pseudobefehle sind die der Datendefinition (Tabelle 2). Fast alle Assembler-Programme verwenden bestimmte Speicherstellen, um Variablen aufzubewahren oder Platz für einen späteren Zugriff zu reservieren. Der MASM verfügt über eine Reihe von Pseudobefehlen, die einer Variablen einen bestimmten Wert oder Speicherbereich zuordnen. Ein Pseudobefehl für die Datendefinition muß nach folgendem Schema aufgebaut sein:

```
[Variable] Definitionsbehl Operand
[,Operand [,Operand [...]]]
```

Die Angaben in den eckigen Klammern sind optional. Für den oder die Operanden sind verschiedene Formate (Bitfeld, dezimal, hexadezimal und so weiter) zulässig.

Es ist vollkommen unterschiedlich, mit welchem Wert jeder Operand höchstens belegt werden darf, denn der Höchstwert richtet sich nach dem verwendeten Pseudobefehl. Der Pseudobefehl »db« erlaubt es Ihnen sogar, neben Zahlen auch Zeichenketten direkt in den Speicher zu schreiben.

Die folgenden Beispiele zeigen Ihnen, wie Sie mit Hilfe der Pseudobefehle Konstanten definieren:

```
plz dw 8013
name db 'DOS International'
prozessor dd 80386
```

● **Definition von Tabellen:** Die Definitions-Pseudobefehle erlauben Ihnen nicht nur, Daten und Konstanten zu definieren, sondern auch Tabellen aufzubauen, indem Sie zwei oder mehrere Operanden durch Kommata trennen. Das folgende Beispiel demonstriert solch einen Tabellaufbau für MASM und SAM 4:


```
adresse db 'Redaktion '
db 'DOS International$'
db 'Gruberstraße 46c$'
db '8011 Poing$'
tabelle db 80, 83, 47, 50
db 45, 77, 111, 100
db 101, 108, 108, 101
```

● **Der Befehl »puffer«:** Wichtig ist auch die Funktion der Definitions-Pseudobefehle zum Reservieren von Speicherbereichen. Hier unterscheiden sich MASM und SAM 4. Um beispielsweise 255 Byte für einen Puffer zu reservieren, dessen Inhalt zu Beginn nicht feststeht, müssen Sie bei MASM

```
puffer db 255 dup (?)
```

schreiben. Bei Sam 4 vereinfacht sich diese Befehlszeile zu

```
puffer ds 255,0
```

Beachten Sie, daß SAM 4 den Speicherbereich mit dem Wert 0 initialisiert.

MASM stellt für die Programmierer größerer Softwareprojekte, die sich beispielsweise für unterschiedliche Hardwarekonfigurationen assemblieren lassen müssen, eine Gruppe von bedingten Pseudobefehlen zur Verfügung. Bedingte Pseudobefehle veranlassen den Assembler, einen bestimmten Teil eines Quellprogramms zu assemblieren oder ihn zu ignorieren – je nachdem, ob eine Bedingung während des Übersetzungsvorgangs logisch wahr oder falsch ist. Durch dieses wahlweise Assemblieren bestimmter Quelltextteile können Sie ein Programm leicht an verschiedene Computersysteme oder Peripherieeinheiten anpassen.

● **Die Befehle »if«, »else« und »endif«:** Falls Sie einen Teil Ihres Programms für die bedingte Assemblierung vorgesehen haben, so leiten Sie die gewünschten Befehlszeilen mit einem der in Tabelle 3 aufgeführten If-Befehle ein und schließen sie mit dem Befehl »endif« ab. Erkennt MASM einen dieser Pseudobefehle, dann assembliert er die eingeschlossenen Anweisungen, falls der Ausdruck logisch wahr ist. Andernfalls ignoriert der Assembler die in dieser Weise gekennzeichneten Befehlszeilen und unterbricht den Übersetzungsvorgang, bis der Programmteil auftritt, der mit »else« beginnt. Ist kein Else-Befehl aufgeführt, fährt er bei der Anweisung fort, die auf das »endif« folgt. Das folgende Beispiel demonstriert den Aufbau einer If-Konstruktion:

```
if angeschlossen
drucken 13h,10h ;Makro-Aufruf Drucker
else
ausgabe,13h,10h ;Makro-Aufruf Bild-
;schirm
endif
```

Bei jedem Übersetzungsvorgang, zu dem die vorher definierte Variable »an-

geschlossen« einen Wert ungleich 0 enthält, wird das Makro »drucken« in den Quelltext eingefügt. Ist hingegen »angeschlossen« gleich 0, so wird das Makro »ausgabe« übersetzt.

Pseudos in der Makro- und Prozedurprogrammierung

● **Die Befehle »macro« und »endm«:** Die Makro-Pseudobefehle (Tabelle 4) gehören zu den sinnvollsten Erweiterungen eines Assemblers. Die Hauptbefehle »macro« und »endm«, die den Anfang und das Ende eines Makros definieren, erlauben dem Assembler-Programmierer den Aufbau eigener Makrobibliotheken, die das ständige Neuprogrammieren von Standardroutinen überflüssig machen. So können Sie zum Beispiel ständig wiederkehrende Routinen mit unterschiedlichen Parametern als Makro definieren. Dieses Verfahren macht Assembler-Programme übersichtlich und besser lesbar.

● **Makrodefinitionen bei MASM:** Allerdings unterscheiden sich die Art der Definition und die Form des Makroaufrufs bei MASM und SAM 4 ein wenig. Bei MASM geben Sie ein Makro mit der Übergabe von zwei Parametern zum Beispiel wie folgt ein:

```
funktion macro variable,nummer
mov dx,variable
mov ah,nummer
int 21h
endm
```

Zum Aufruf genügt die Angabe der Makrobezeichnung:

```
funktion text,09h
```

MASM ersetzt beim Assemblieren den Makroaufruf durch das Makro selbst, wobei er »variable« gegen »text« und »nummer« gegen den Wert 09hex austauscht.

● **Makrodefinitionen bei SAM 4:** SAM 4 behandelt Makrodefinition und -aufruf etwas anders. Jeder Parameter wird bis zu einer Höchstzahl von acht Angaben durch eine Kombination aus dem Prozentzeichen und einer Ziffer repräsentiert. Das oben beschriebene Beispiel für den MASM lautet bei SAM 4 folgendermaßen:

```
funktion macro %1,%2
mov dx,%1
mov ah,%2
int 21h
endm
```

Der Makroaufruf erfolgt bei SAM 4 ebenfalls etwas anders als bei MASM:

```
do funktion,text,09h
```

MASM besitzt darüber hinaus eine Reihe weiterer, leistungsfähiger Makrobefehle, die in vielen Situationen sehr hilfreich sind. Beispielsweise können Sie mit »exitm« ein Makro vorzeitig verlassen, wenn Sie den Befehl im Zusammenhang mit der bedingten Assemblierung benutzen.

● **Lokale und globale Marken:** Falls ein Makro Sprungmarken enthält, ist es wünschenswert, daß der Assembler diese Marken von denen des Hauptprogramms unterscheidet. Andernfalls träte in jedem Fall der Fehler »Symbol mehrfach definiert« auf. Um diese Fehlermeldung zu vermeiden, teilen Sie MASM mit dem Pseudobefehl »local« (Tabelle 4) mit, welche Sprungmarken ausschließlich lokal zu

Symboldefinition

MASM	SAM 4	Erklärung
equ	equ	weist einem symbolischen Namen einen konstanten Wert zu
=	=	Entspricht dem Pseudobefehl »equ«

Tabelle 1. Die Pseudobefehl der Symboldefinition für MASM und SAM 4 im Überblick

Datendefinition

MASM	SAM 4	Erklärung
db	db	definiert Byte-Speicherbereich für Daten (8 Bit); es lassen sich mehrere Werte zuweisen.
dd	–	definiert Doppelwort-Speicherbereich für Daten (32 Bit); es lassen sich mehrere Werte zuweisen.
dq	–	definiert Vierfachwort-Speicherbereich für Daten (64 Bit); es lassen sich mehrere Werte zuweisen.
–	DS	reserviert die angegebene Anzahl von Bytes für die spätere Verwendung (entspricht dem Masm-Pseudobefehl »db anzahl dup (?)«).
dt	–	weist 10 Byte zu (wird hauptsächlich in der Fließkomma-Arithmetik benutzt).
dw	dw	definiert Wort-Speicherbereich für Daten (16 Bit); es lassen sich mehrere Werte zuweisen.

Tabelle 2. Für die Daten- und Konstantendefinition stehen Ihnen diese Pseudobefehle zur Verfügung.

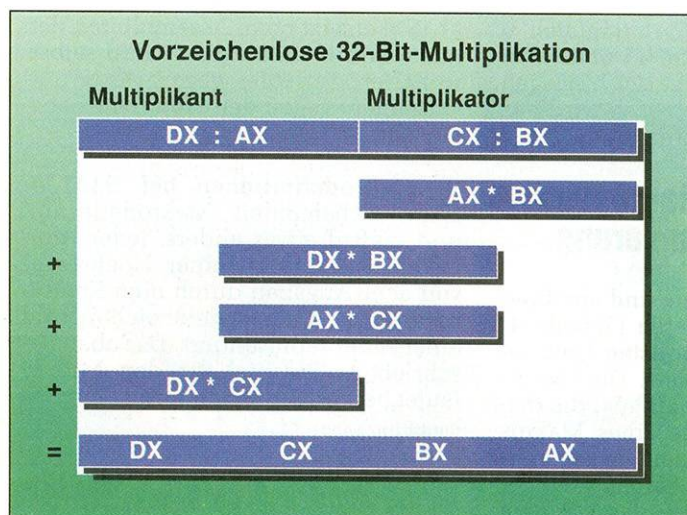


Bild 1. Die Routine in Listing 1 multipliziert zwei 32-Bit-Zahlen, indem sie die zugehörigen Register einzeln miteinander multipliziert.

verwendet sind. Dabei ist es wichtig, daß »local« am Anfang eines Makros steht. Die Befehlsfolge

```
befehl macro eins,zwei
local schleife
schleife: mov ax,eins
inc zwei
jpe schleife
endm
;
schleife: befehl drucken,ax
ist damit durchaus legitim.
```

MASM verarbeitet zusätzlich die drei Makro-Pseudobefehle »rept«, »irp« und »irpc« (Tabelle 4). Sie veranlassen den Assembler, eine Folge von Anweisungen mehrmals zu wiederholen. Bei jedem dieser Befehle wird die Anzahl der Wiederholungen im Operandenfeld angegeben. Der Operand des Pseudobefehls »rept« muß ein mathematischer Ausdruck oder ein Symbol sein, aus dem sich die Anzahl der Wiederholungen ermitteln läßt. Beim Pseudobefehl »irp« können Sie zusätzlich zu einem anzugebenden Hilfsparame- ter eine Liste bestimmen. Die Anweisung wird dann für jedes Element dieser Liste wiederholt.

Das folgende Beispiel erläutert die Funktion dieser Pseudobefehle anhand des Befehls »rept«. Das kurze Listing gibt alle ASCII-Zeichen aus:

```
ascii macro
code = 0
rept 256
call ausgabe
code = code + 1
endm
```

● **Der Befehl »purge«:** Der letzte Makrobefehl, der bei MASM Verwendung findet, ist »purge« (Tabelle 4). Mit diesem Pseudobefehl löschen Sie Makros aus den internen Tabellen des Assemblers, so daß der von ihnen belegte Arbeitsspeicher wieder frei wird. Der Pseudobefehl löscht aber keineswegs die Makroquelle oder das Objektprogramm auf Diskette oder Festplatte. Wenn Ihnen eine Zusammenstellung mehrerer Makros (beispielsweise eine

Makrobibliothek) als eigenständige Datei vorliegt, die Sie in einem anderen Programm verwenden möchten, müssen Sie natürlich dafür sorgen, daß der jeweilige Assembler beim Übersetzen Ihres Programms diese externen Dateien auch in den Assembler-Quelltext einbindet.

● **Der Befehl »include«:** Das geschieht mit dem Pseudobefehl »include«, der sowohl MASM als auch SAM 4 veranlaßt, die benötigten Dateien einzulesen und zu verwerten. Dazu führen Sie hinter dem Include-Befehl nur den Namen der Datei auf, die die gewünschten Makros enthält. Sollen mehrere Makrodateien verwendet werden, so benutzen Sie den Befehl mehrmals im Programm. Der Pseudobefehl »include« (Tabelle 5) dient aber ebenso dazu, schon vorhandene und auf Lauffähigkeit hin getestete Quelltexte in das Hauptprogramm einzubinden. Die Syntax bleibt

in diesem Fall gleich. Möchten Sie beispielsweise eine Ausgaberroutine namens »drucker.asm«, die sich auf Diskette im Laufwerk A befindet, in Ihr Hauptprogramm »ausgabe.asm« einbinden, so setzen Sie die Befehlszeile

```
include a:drucker.asm
```

zu Beginn von »ausgabe.asm« ein. Wenn der Assembler dann das Hauptprogramm übersetzt, liest er alle Befehlszeilen aus der Datei »drucker.asm« ein und behandelt sie, als ständen sie direkt an der Position des zugehörigen Include-Befehls im Quelltext.

● **Die Befehle »public« und »extrn«:** Der Pseudobefehl »include« gehört wie die MASM-Befehle »extrn« und »public« (Tabelle 5) zur Gruppe der Pseudobefehle für externe Verweise. Mit »public« werden die angegebenen Symbole auch für andere Assembler-Module zugänglich, die eventuell zum aktuellen Modul hinzugeladen werden. Der Pseudobefehl »extrn« hingegen indiziert die Symbole, die in anderen Modulen definiert und als »public« deklariert wurden.

Wollen Sie beispielsweise die Variable »zaehler« in zwei verschiedenen Modulen verwenden, so muß der Teil, in dem »zaehler« definiert wird, folgende Struktur aufweisen:

```
public zaehler
zaehler db 0 ; Zähler auf Null setzen
```

Um »zaehler« in dem anderen Modul zu verwenden, geben Sie folgende Anweisung ein, die auch den Variablentyp (»byte«, »word«, »dword«, »far« oder »near«) bestimmt:

```
extrn zaehler:byte
```

● **Die Befehle »proc« und »endp«:** Anstelle eines Makros läßt sich in einem Assembler-Programm auch eine

Bedingte Assemblierung

MASM SAM 4 Erklärung

else	–	gibt an, daß der folgende Befehlsblock ausgeführt werden soll, wenn der vorangegangene Vergleich logisch falsch war.
endif	–	gibt das Ende einer bedingten Assemblierung an.
if	–	ergibt den logischen Wert »wahr«, wenn der angegebene Ausdruck ungleich 0 ist.
ifb	–	ergibt den logischen Wert »wahr«, wenn das Argument leer ist.
ifdef	–	ergibt den logischen Wert »wahr«, wenn das spezifizierte Symbol in- oder extern definiert wurde.
ife	–	ergibt den logischen Wert »wahr«, wenn der Ausdruck gleich 0 ist.
ifidn	–	ergibt den logischen Wert »wahr«, wenn die beiden angegebenen Zeichenketten identisch sind.
ifidf	–	ergibt den logischen Wert »wahr«, wenn sich die beiden angegebenen Zeichenketten unterscheiden.
ifnb	–	ergibt den logischen Wert »wahr«, wenn das Argument nicht leer ist.
ifndef	–	ergibt den logischen Wert »wahr«, wenn das spezifizierte Symbol in- oder extern nicht definiert wurde.
if1	–	ergibt den logischen Wert »wahr«, wenn der Assembler den ersten Durchlauf vornimmt.
if2	–	ergibt den logischen Wert »wahr«, wenn sich der Assembler im zweiten Durchgang befindet.

Tabelle 3. Für die bedingte Assemblierung können Sie diese Pseudobefehle benutzen.

Prozedur einsetzen, die bei MASM und SAM 4 durch den Pseudobefehl »proc« eingeleitet und mit »endp« beendet wird (Tabelle 6). Bei jedem Aufruf einer solchen Prozedur führt der Prozessor die Befehle aus, die durch »proc« und »endp« eingegrenzt sind, und kehrt daraufhin zu dem Befehl zurück, der dem Prozeduraufruf folgt.

Die Programmierung von Prozeduren

Eine typische Prozedur ist wie folgt aufgebaut:

```
ausgabe proc
...
ret
ausgabe endp
```

Im Gegensatz zu einem Makro muß eine Prozedur jedesmal über den Befehl »call« aufgerufen und mit »ret« beendet werden. Das kostet Zeit, und die ist – vor allem für den Assembler-Programmierer – äußerst kostbar.

Dagegen erscheint nach dem Assemblieren der Maschinencode einer Prozedur nur ein einziges Mal im Programm, während der Maschinencode eines Makros überall dort vorhanden ist, wo das Makro benötigt wurde. Damit begnügt sich eine Prozedur mit erheblich weniger Speicherplatz als ein Makro – besonders dann, wenn das Makro recht lang ist und oft benutzt wird. Kurz gesagt, eine Prozedur spart also Speicherplatz, ein Makro dagegen Zeit.

• Die Befehle »page« und »title«:

Eine weitere Gruppe von Pseudobefehlen steuert die Ausgabe (Tabelle 7). Da diese sehr zahlreich sind, werden im folgenden nur die drei wichtigsten erläutert.

Der Pseudobefehl »page« hat den Aufbau

```
page [zeilen] [,spalten]
```

und legt das Druckformat bei der Ausgabe des übersetzten Quelltextes auf einem Drucker fest. Die Parameter »zeilen« und »spalten« definieren Länge und Breite der Seiten.

Der Pseudobefehl »title« versieht jede zweite Zeile einer ausgegebenen Seite mit einer linksbündigen Überschrift. In den meisten Fällen enthält diese Überschrift den Dateinamen; Sie können sie aber auch frei wählen.

Mit »subttl« wird schließlich in der dritten Zeile jeder Seite ein linksbündiger Untertitel ausgegeben. Auch er ist frei wählbar und beschreibt im allgemeinen den Inhalt der Seite. Die folgenden Pseudobefehle definieren eine typische Kopfzeile:

```
page 66,80
title framework.asm
subttl Die Textverarbeitung
```

• **Der Befehl »end«:** Die letzte Gruppe der Pseudobefehle steuert einen Assembler direkt an (Tabelle 8). Der wichtigste Befehl, der garantiert in jedem Assembler-Programm auftritt, ist »end«. Dieser Pseudobefehl markiert das Ende eines Quelltextes und teilt dem Assembler mit, an welcher Stelle er die Assemblierung beenden kann.

MASM verarbeitet darüber hinaus die Befehlsform

```
end marke
```

die mit der Sprungmarke »marke« die Startadresse für den assemblierten Maschinencode angibt. SAM 4 nimmt hingegen an, daß der Anfang des Codesegments der Startadresse des Programms entspricht.

• **Der Befehl »org«:** Der Pseudobefehl »org« setzt bei einem Assembler den Adreßzähler neu, der bestimmt, unter welchen Adressen die Befehle und Daten im Arbeitsspeicher abgelegt werden. Beim PC übernimmt zwar normalerweise das Betriebssystem MS-DOS diese Entscheidung, aber in gewissen Fällen ist es notwendig, diese zu beeinflussen. Diese Notwendigkeit tritt dann ein, wenn ein Maschinenspra-

cheprogramm als Datei mit der Endung »com« gespeichert werden soll. In diesem Fall muß vor dem ersten Befehl der Adreßzähler mit

```
org 100h
```

auf den Wert 100hex gesetzt werden, weil MS-DOS in den ersten 100hex Byte wichtige Systeminformationen ablegt.

Der Assembler-Code läßt sich auch relativ zum aktuellen Adreßzähler übersetzen, weil der aktuelle Zustand des Adreßzählers durch das Dollarzeichen repräsentiert wird. Das bedeutet, daß beispielsweise die Anweisung

```
org $ + 1024
```

bei MASM den Adreßzähler um den Wert 1024 erhöht.

• **Der Befehl »even«:** In engem Zusammenhang mit »org« steht der Pseudobefehl »even«. Er erzwingt, daß der Adreßzähler auf eine gerade Adresse gesetzt wird. Weist der Adreßzähler bereits einen geraden Wert auf, so passiert gar nichts. Ist die Adresse dagegen ungerade, so fügt der Assembler beim Übersetzen den Dummy-Assembler-Befehl »nop« ein. Dieser sorgt dafür, daß der Adreßzähler wieder auf

Makrodefinition

MASM	SAM 4	Erklärung
–	do	Aufruf eines zuvor definierten Makros. Der folgende Parameter gibt den Makronamen an, die weiteren Parameter werden in das Makro übernommen.
endm	endm	schließt eine Makrodefinition ab und kennzeichnet das Makroende.
exitm	–	beendet die Ausführung eines Makros aufgrund eines vorangegangenen Vergleichs.
irp	–	wiederholt die Anweisungen zwischen »irp« und »endm« für jedes Argument in der angegebenen Argumentliste mit den einzelnen Elementen.
irpc	–	entspricht grundsätzlich »irp«; der Unterschied besteht darin, daß statt der Argumentliste eine Zeichenkette übergeben wird.
local	–	bewirkt, daß der Assembler jede mit diesem Befehl definierte Marke als lokal auffaßt; das bedeutet, daß in diesem Fall das Hauptprogramm die gleichen Marken enthalten kann.
macro	macro	definiert die vorangehende Marke als Namen eines Makros, das wiederum durch die nachstehenden Parameter definiert wird.
purge	–	löscht das angegebene Makro aus dem Arbeitsspeicher des Computers, so daß der benutzte Speicher wieder verwendet werden kann.
rept	–	wiederholt alle Anweisungen, die sich zwischen »rept« und »endm« befinden, so lange, bis die angegebene Anzahl erreicht ist.

Tabelle 4. Diese Pseudobefehle benötigen Sie für die Makrodefinition.

Externe Informationen

MASM	SAM 4	Erklärung
extrn	–	gibt Symbole an, die in anderen Modulen definiert wurden; so können Sie auf bereits überprüfte Assembler-Quelltexte mit Querverweisen zugreifen.
include	include	fügt den Quelltext mit dem angegebenen Dateinamen in das Listing ein; auf diese Weise binden Sie bequem Bibliotheken ein.
public	–	kennzeichnet die Symbole, die andere Assembler-Programme verwenden können, nachdem sie verknüpft worden sind.

Tabelle 5. Für externe Informationen brauchen Sie diese Sammlung von Pseudobefehlen.

eine gerade Adresse zeigt. Der Einsatz des Pseudobefehls »even« ist vor allem dann sinnvoll, wenn Programme speziell für den 8086-Prozessor entwickelt werden. Da dieser einen 16 Bit breiten Datenbus besitzt, kann er auch 16 Datenbits gleichzeitig übertragen. Dieser Vorgang dauert aber länger, wenn die übertragenen Daten auf einer ungeraden Adresse liegen. So profitieren vor allem zeitkritische Programme vom Einsatz des Even-Befehls.

● **Der Befehl »radix«:** Der letzte Pseudobefehl dieser Kursfolge ist der Befehl »radix«. Wie Sie bereits wissen, lassen sich Daten binär, dezimal und hexadezimal eingeben. Um die einzelnen Zahlensysteme voneinander zu unterscheiden, gibt ein Identifikationsbuchstabe die jeweilige Zahlenbasis an, wenn sie von der dezimalen Standardbasis 10 abweicht.

Tritt jedoch der Fall ein, daß Sie mit vielen nichtdezimalen Zahlen arbeiten müssen, empfiehlt es sich, die Standardbasis Ihren Erfordernissen anzupassen. Genau dazu dient der Pseudobefehl »radix«. Sie stellen damit bei MASM die Standardbasis von 10 auf jede andere Basis von 2 bis 16 um. Die Befehlszeile

```
.radix 16
```

legt beispielsweise die Zahlenbasis auf 16 fest, so daß Sie fortan mit Hexadezimalzahlen arbeiten können, ohne an jede Zahl den Buchstaben »h« anhängen zu müssen.

Die Anzahl der verfügbaren und sinnvollen Pseudobefehle ist ein grundlegender Maßstab für die Qualität eines Assemblers. Je mehr Pseudobefehle er beherrscht, desto einfacher wird die Programmgestaltung und deren Umsetzung in Maschinensprache.

Ein rasend schneller Abakus

Das folgende Thema bereitet selbst fortgeschrittenen Assembler-Programmierern gelegentlich Schwierigkeiten: Arithmetik auf Maschinensprachebene. Damit es Ihnen nicht ebenso ergeht, lernen Sie im folgenden die grundlegenden Techniken der Assembler-Programmierung für Arithmetikroutinen mit dem Schwerpunkt Integer-Arithmetik. Sie erfahren den Aufbau und die Funktion der Arithmetikbefehle des 8086/8088-Prozessors und lernen die grundlegenden mathematischen Operationen für ganze Zahlen kennen, die den Umfang der Arithmetikbefehle des Prozessors übersteigen. Arithmetikroutinen bestehen ausschließlich aus Befehlen für die Grundrechenarten Addition, Subtraktion, Multiplikation und Division, weil der 8086/8088-Prozessor nur Be-

Unterprogrammangabe

MASM	SAM 4	Erklärung
endp	endp	markiert das Ende eines Unterprogramms.
proc	proc	kennzeichnet den Beginn eines Unterprogramms.

Tabelle 6. Um Prozeduren zu definieren, benutzen Sie diese Pseudobefehle.

Ausgabesteuerung

MASM	SAM 4	Erklärung
page	—	stellt die Länge und Breite einer Druckseite ein.
subttl	subttl	definiert den Untertitel eines Quelltextausschnitts.
titel	titel	definiert den Titel eines Listings.
.cref	—	Gegenstück zu »xcref«
.lall	—	bewirkt bei der Ausgabe des Assembler-Listings, daß alle Makrotexte des Haupttextes und jeder Erweiterung ausgegeben werden.
.lfcond	—	listet alle bedingten Blöcke auf, die als logisch falsch ausgewertet wurden.
.list	—	Gegenstück zu »xlist«
.out	—	veranlaßt die Ausgabe von Systemmeldungen auf dem Bildschirm während des Assemblierens.
.sall	—	unterdrückt die Ausgabe der Makros.
.sfcond	—	listet als falsch ausgewertete bedingte Blöcke nicht auf.
.tfcond	—	kehrt die Einstellung, ob die als falsch ausgewerteten Programmblöcke ausgegeben werden, um.
.xall	—	gibt in bezug auf Makros ausschließlich die Zeilen aus, die Objektcode erzeugen.
.xcref	—	unterdrückt die Ausgabe der Querverweisliste (»Cross Reference«) von der aktuellen Position des Pseudobefehls bis zum nächsten »cref«-Befehl.
.xlist	—	unterdrückt die Ausgabe des Assembler-Listings von der aktuellen Position des Pseudobefehls bis zum nächsten »list«.

Tabelle 7. Die Pseudobefehle des MASM und SAM 4 für die Ausgabesteuerung im Überblick

Weitere Befehle

MASM	SAM 4	Erklärung
end	end	gibt das Ende des Assembler-Programms an.
even	even	setzt den Befehlszähler auf eine gerade Adresse.
org	org	setzt den Befehlszähler auf die angegebene Adresse (beispielsweise für Com-Dateien notwendig).
.radix	—	ändert die Standardbasis bei der Angabe von Zahlen auf jede beliebige Basis von 2 bis 16.

Tabelle 8. Weitere Pseudobefehle, die bei MASM und SAM 4 nützlich sind

fehle für diese mathematischen Operationen besitzt. Alle anderen arithmetischen Operationen, so auch die trigonometrischen Funktionen, werden von diesen Befehlen abgeleitet.

● **Das Intel-Format:** Die Routinen, die im folgenden vorgestellt werden, arbeiten mit ganzzahligen Werten. Der 8086/8088-Prozessor legt diese Werte in umgekehrter Reihenfolge im Arbeitsspeicher ab. Wenn der Prozessor beispielsweise die hexadezimale Zahl 4782hex in der Speicherzelle »data« speichert, steht der Wert 82hex in »data« und der Wert 47hex in »data + 1«.

Dieses Verfahren der umgekehrten Speicherung, auch als Intel-Format bekannt, müssen Sie bei allen Operationen berücksichtigen, die mit Werten

arbeiten, die aus mehreren Bytes oder Worten bestehen.

Der 8086/8088-Prozessor führt arithmetische Operationen mit binären und dezimalen Werten ohne Vorzeichen durch. Zu den wichtigsten Befehlen gehören die Additionsbefehle, die 8- und 16-Bit-Operanden zusammenzählen.

● **Die Befehle »add« und »adc«:** Der Befehl

```
add AX, BX
```

addiert beispielsweise die Inhalte der beiden 16-Bit-Register AX und BX und legt das Ergebnis in AX ab. Allerdings taucht ein Problem auf, wenn der Zieloperand, in diesem Fall das AX-Register, die Summe der beiden Werte nicht aufnehmen kann. Falls Sie beispielsweise zwei 8-Bit-Register zusam-

menzählen, ist folgende Situation denkbar:

AL: 10101010
+ BL: 11010101

= 10111111

Weil für die Darstellung dieses Ergebnisses ein 8-Bit-Register nicht ausreicht (das Ergebnis hat neun Bit-Stellen), schreibt der Prozessor die niederwertigen acht Bit in das Zielregister und legt das neunte Bit als Übertrag im Carry-Statusbit ab.

Befinden sich nun in den Registern AL und BL jeweils nur die unteren Hälften der beiden zu addierenden Werte und in den Registern AH und BH die oberen Hälften, so können Sie diese mit dem Befehl »adc« zusammenzählen, der den Übertrag im Carry-Statusbit berücksichtigt.

Wenn Sie also Werte addieren möchten, die größer als zwei Byte sind, hat das Verfahren mit Übertrag und dem Additionsbefehl »adc« für alle höherwertigen Bytes wieder Sinn.

● **Die Befehle »sub« und »sbb«:** Analog zu den Additionsbefehlen gibt es auch bei den Subtraktionskommandos zwei Varianten. Der Befehl »sub« zieht einen Subtrahenden von einem Minuenden ab und legt das Ergebnis in dem Register ab, in dem sich der Minuend befindet. So vermindert beispielsweise der Befehl

sub AX, BX

den Inhalt von AX um den Inhalt von BX und legt das Ergebnis im AX-Register ab.

Das Kommando »sbb« berücksichtigt bei der Subtraktion zusätzlich einen eventuell vorhandenen Übertrag im Carry-Statusbit und bezieht diesen in die Berechnung mit ein.

Bei der internen Verarbeitung der Subtraktionsbefehle ist zu beachten, daß der 8086/8088-Prozessor keine Einheit zum Subtrahieren von Zahlen besitzt. Er subtrahiert vielmehr Zahlen, indem er das Negative des Subtrahenden addiert. Um Ihnen diese Technik näherzubringen, hier ein Beispiel: Sie subtrahieren die Zahl 8 von der Zahl 21.

Im allgemeingültigen Format schreiben Sie

21 - 8

Diesen Ausdruck können Sie auch folgendermaßen formulieren:

21 + (-8)

Der 8086/8088-Prozessor führt die Subtraktion in zwei Schritten durch: Zuerst negiert (»komplementiert«) er den Subtrahenden (die zweite Zahl). Danach addiert er den Minuenden (die erste Zahl) zu dem Komplement, um das Resultat zu bilden. Da der 8086/8088 mit Zahlen zur Basis 2 arbeitet, nennt man das Ergebnis dieses Vorgangs »Zweierkomplement«.

Programmblockangabe

MASM	SAM 4	Erklärung
group	-	faßt mehrere Segmente unter einem Namen zusammen, so daß sie alle in einem 64-KByte-Bereich innerhalb des Arbeitsspeichers liegen.
name	-	weist einem Modul den vorangehenden Namen zu.

Tabelle 9. Die Pseudobefehle für die Programmblockangabe im Überblick

Segmentangaben

MASM	SAM 4	Erklärung
assume	assume-	teilt dem Assembler mit, welches Segment den angegebenen Segmentregistern zugeordnet ist.
ends	ends	markiert das Ende eines Segments.
segment	segment	kennzeichnet den Beginn eines Speichersegments.

Tabelle 10. Für Segmentangaben benutzen Sie diese Pseudobefehle.

Befehl Funktion

aaa	Ergebnis nach einer ASCII-Addition anpassen
aad	Ergebnis nach einer ASCII-Division anpassen
aam	Ergebnis nach einer ASCII-Multiplikation anpassen
aas	Ergebnis nach einer ASCII-Subtraktion anpassen
adc	unter Berücksichtigung des Übertrags addieren
add	ohne Übertrag addieren
cbw	Umwandlung eines Bytes in ein Wort
cmp	einen Vergleich der angegebenen Operanden ausführen
cwd	Umwandlung eines Worts in ein Doppelwort
daa	Ergebnis nach einer dezimalen Addition anpassen
das	Ergebnis nach einer dezimalen Subtraktion anpassen
dec	angegebenen Operanden um den Wert 1 vermindern
div	vorzeichenlose Division ausführen
idiv	Operanden unter Berücksichtigung des Vorzeichens dividieren
imul	mit Vorzeichen versehene Operanden multiplizieren
inc	angegebenen Operanden um den Wert 1 erhöhen
mul	vorzeichenlose Multiplikation ausführen
neg	negiert Operanden beziehungsweise bildet dessen Zweierkomplement
sbb	unter Berücksichtigung des Übertrags subtrahieren
sub	ohne Übertrag subtrahieren

Tabelle 11. Mit den Arithmetikbefehlen des 8086/8088-Prozessors läßt sich jede mathematische Aufgabe lösen.

Um das Zweierkomplement einer negativen Zahl zu erhalten, invertiert man jedes Bit der positiven Form der Zahl und addiert anschließend 1. Im Beispiel (-8) ist das positive Bitmuster

00001000

Invertiert führt das zu

11110111

Nach der Addition von 1 ergibt das

11111000

Die Addition der beiden Zahlen führt zu folgendem Resultat:

00010101

+ 11111000

= 00001101

Das Bitmuster des Ergebnisses lautet dezimal 13. Die Subtraktion ist also korrekt verlaufen.

● **Die Befehle »mul« und »imul«:** Auch für die Multiplikation besitzt der 8086/8088-Prozessor zwei verschiedene Befehle. Sie haben das Format

mul Operand
und

imul Operand

und multiplizieren den angegebenen Operanden bei Byte-Operationen (8 Bit) mit dem AL- und bei Wort-Operationen (16 Bit) mit dem AX-Register. Im ersten Fall wird das Ergebnis in AX, im zweiten Fall in DX und AX zurückgegeben, wobei das DX-Register das höherwertige Wort enthält.

Die beiden Multiplikationsbefehle unterscheiden sich darin, daß der Befehl »imul« im Gegensatz zu »mul« bei der Ausführung ein eventuell vorhandenes Vorzeichen berücksichtigt.

● **Die Befehle »div« und »idiv«:** Den beiden Multiplikationsbefehlen entsprechend, verarbeitet der 8086/8088-Prozessor auch zwei verschiedene Divisionsbefehle. Sie haben das Format

div Divisor

und

idiv Divisor

Der Befehl »div« teilt zwei Zahlen ohne, »idiv« mit Berücksichtigung des Vorzeichens. Sie können allerdings jeweils nur den Divisor (die Zahl, durch die die andere geteilt wird) spezifizieren. Als Dividenden (die Zahl, die durch den Divisor geteilt werden soll) verwendet der 8086/8088 automatisch das Register AX bei Byte- sowie DX und AX bei Wort-Operationen.

Liegt der Divisor im Byte-Format vor, so wird das Ergebnis der Division im AL- und der Rest im AH-Register zurückgegeben. Ist der Divisor dagegen 16 Bit groß, so speichert der Prozessor das Ergebnis in AX und den Rest in DX.

● **Der Interrupt 0hex:** Bitte beachten Sie bei allen Additions-, Subtraktions-, Multiplikations- und Divisionsbefehlen, daß sich weder zwei Zahlen direkt noch zwei Werte im Speicher unmittelbar miteinander verknüpfen lassen. Vielmehr muß sich zumindest einer der beiden Werte in einem Register befinden, in das er über einen Mov-Befehl geladen wird. Bei den Multiplikations- und Divisionsbefehlen sind sogar ganz bestimmte Register vorgeschrieben.

Bei den Divisionsbefehlen tritt eine weitere Besonderheit auf. Falls nämlich das Ergebnis einer Division die Kapazität des Zielregisters überschreitet oder der Divisor den Wert 0 enthält, ruft der Prozessor automatisch den Interrupt 0 (»Division durch 0«) auf. Wenn also ab der Speicheradresse 0 ein Interrupt-Vektor, bestehend aus einer 16-Bit-Offset- und Segmentadresse, abgelegt ist, der auf eine Fehlerbehandlungsroutine zeigt, wird diese Routine automatisch aufgerufen, sobald bei der Division einer der beiden erwähnten Fehler auftritt.

Neben den vorgestellten grundlegenden Arithmetikbefehlen gibt es einige weitere, die nur in speziellen Fällen Anwendung finden. Die Funktion dieser und der bisher behandelten Arithmetikbefehle entnehmen Sie Tabelle 11.

● **Addition und Subtraktion:** Die Addition ist die einfachste Grundrechen-

art. Wichtig ist bei der Addition vor allem, daß Sie sich überlegen, ob bei der Addition zweier Zahlen ein Übertrag auftreten kann. Ist dies der Fall, so wird das Übertrag-Statusbit gesetzt und muß entsprechend berücksichtigt werden. Wenn Sie Zahlen addieren, die aus mehreren Bytes bestehen, werden zuerst die beiden niederwertigsten Bytes mit »add« und alle weiteren Bytes mit »adc« addiert, damit bei den höherwertigen Bytes jeweils der Übertrag berücksichtigt wird.

Der Aufbau der Arithmetikroutinen

Die Subtraktion von Zahlen, die aus mehreren Bytes bestehen, entspricht dem Vorgang bei der Addition. Hier wird beim Subtrahieren der beiden niederwertigsten Bytes der Befehl »sub« und bei den höherwertigen Bytes jeweils der Befehl »sbb« benutzt, um einen eventuell vorhandenen Übertrag im Carry-Statusbit zu berücksichtigen.

● **Multiplikation und Division:** Bei der Multiplikation von Werten größer als 16 Bit wird die Berechnung schon komplizierter. Wie Sie bereits wissen, können Sie mit Hilfe des Befehls »mul« nur Werte bis zu einer maximalen Länge von 16 Bit miteinander multiplizieren. Bei den meisten praktischen Anwendungen muß man aber Zahlen multiplizieren, die 32 Bit oder länger sind. Das bedeutet, daß die vorhandene Genauigkeit nicht ausreicht. Listing 1 zeigt deshalb eine Assembler-Routine, die zwei 32-Bit-Werte miteinander multipliziert, ohne die Vorzeichen zu beachten. Sie läuft sowohl unter MASM als auch unter SAM 4. Beachten Sie jedoch, daß die Routine kein eigenständiges Programm darstellt sondern als Prozedur gedacht ist. Wenn Sie das grundlegende Prinzip dieser Routine erkannt haben, steht einer Erweiterung auf 48, 64 oder gar 80 Bit nichts mehr im Weg.

Die beiden Operanden für die Multiplikationsroutine sind jeweils in den

Registerpaaren CX:BX und DX:AX abgelegt. In dieser Notation kennzeichnet das Register vor dem Doppelpunkt das höherwertige und das Register nach dem Doppelpunkt das niederwertige Wort.

Das Ergebnis der Multiplikation wird in dieselben Register geschrieben, und zwar so, daß das DX-Register das höchstwertige und das AX-Register das niederwertigste Wort enthält. Das Bild zeigt, nach welchem Verfahren die Multiplikationsroutine die Inhalte der vier Register zu einem 64-Bit-Wert multipliziert.

Beachten Sie bitte, daß die Routine keine Datenregister sichert. Dies muß deshalb das aufrufende Hauptprogramm erledigen, oder Sie müssen die Multiplikationsroutine erweitern. Mit diesen Ergänzungen ist die Routine ein wichtiges Unterprogramm, das die Grundlage für weitere Multiplikations- und Potenzfunktionen bildet.

Listing 2 zeigt eine Divisionsroutine für SAM. Die Divisionsroutinen demonstrieren, wie ein Überlauf abgefangen wird. Er tritt bei Divisionen auf, wenn der Dividend mindestens 65 536 mal so groß wie der Divisor ist. Dazu wird eine neue Routine für den Interrupt 0 (»Division durch 0«) installiert und der Dividend mit einer einfachen, aber etwas zeitaufwendigeren Prozedur durch den Divisor geteilt.

Die Routine dividiert einen 32-Bit-Wert im Registerpaar DX:AX durch einen 16-Bit-Wert in BX. Am Ende der Routine enthält das Registerpaar BX:AX das Ergebnis und DX den Rest der Division. Beachten Sie bei der Divisionsroutine auch den Sonderfall, daß der Divisor 0 ist. In diesem Fall wird nämlich die alte Routine des Interrupts 0 aufgerufen, weil die neue Routine zu diesem Zeitpunkt noch nicht installiert ist.

Die Anregungen zur Implementierung neuer Routinen für die vier Grundrechenarten zeigen, daß man mit wenig Aufwand zu Berechnungen mit erhöhter Leistung und Genauigkeit gelangt.

(Markus Zietlow/st)

```
[WW2] 1: ; Programm: Multi
[1F2] 2: ; Sprache: Masm
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[7R5] 6: ;--Multiplikationsroutine (nur als
[2V4] 7: ; Unterprogramm einsetzbar)
[P30] 8: ;
[P30] 9: ;
[V45] 10: ;--Definition des Datensegments
[P44] 11: data segment para 'data'
[P30] 12: ;
[9N2] 13: ;--Erster Faktor
[EF2] 14: faktor1 dw 0
[P30] 15: ;
[LT2] 16: ;--Zweiter Faktor
[KF2] 17: faktor2 dw 0
[P30] 18: ;
[GD4] 19: ;--Erster Zwischenspeicher
[0M2] 20: speicher1 dw 0
[P30] 21: ;
[1N4] 22: ;--Zweiter Zwischenspeicher
```

```
[2M2] 23: speicher2 dw 0
[P30] 24: ;
[DL4] 25: ;--Dritter Zwischenspeicher
[4M2] 26: speicher3 dw 0
[P30] 27: ;
[VM4] 28: ;--Vierter Zwischenspeicher
[6M2] 29: speicher4 dw 0
[P30] 30: ;
[XM4] 31: ;--Fünfter Zwischenspeicher
[8M2] 32: speicher5 dw 0
[P30] 33: ;
[5V4] 34: ;--Sechster Zwischenspeicher
[AM2] 35: speicher6 dw 0
[P30] 36: ;
[2L4] 37: ;--Siebter Zwischenspeicher
[CM2] 38: speicher7 dw 0
[P30] 39: ;
[VB4] 40: ;--Achter Zwischenspeicher
[EM2] 41: speicher8 dw 0
[P30] 42: ;
[PW3] 43: ;--Ende des Datensegments
[ZW1] 44: data ends
```



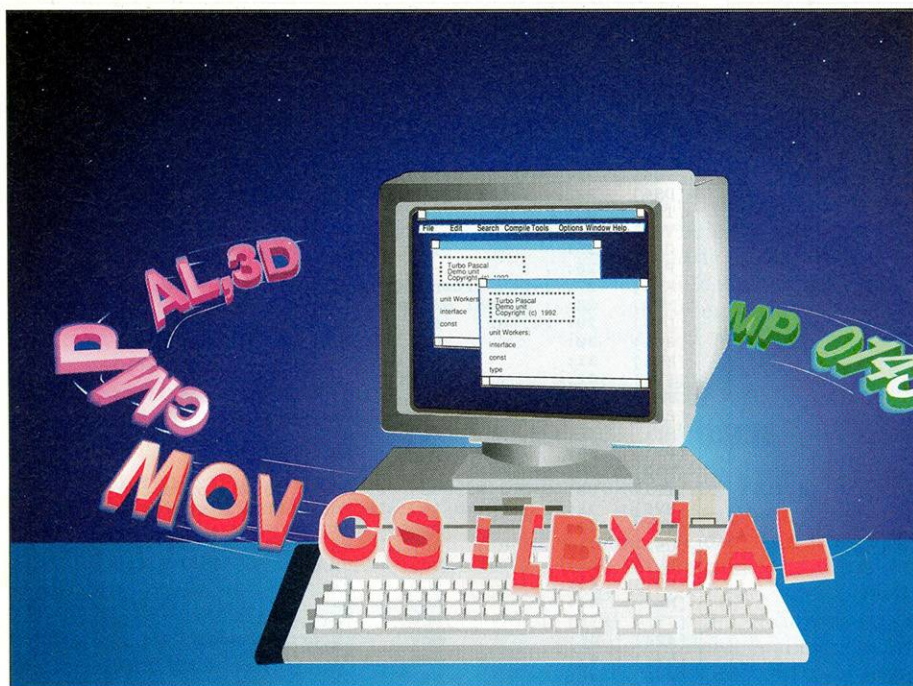
```
[P30] 45: ;
[P30] 46: ;
[X25] 47: ; -Definition des Code-Segments
[X54] 48: code segment para 'code'
[P30] 49: ;
[P30] 50: ;
[5D5] 51: ; -Festlegen des Unterprogramms
[603] 52: multi32 proc far
[P30] 53: ;
[2C3] 54: ; -Segmente festlegen
[7Z3] 55: assume cs:code, ds:data
[P30] 56: ;
[005] 57: ; -Datensegment initialisieren
[HH5] 58: ; -Bei SAM 4 müssen Sie anstatt der
[NC6] 59: ; folgenden Zeile "mov ax,data" schreiben
[HV2] 60: mov ax,seg data
[1V1] 61: mov ds,ax
[P30] 62: ;
[HW4] 63: ; -Faktoren zwischenspeichern
[RF3] 64: ; und multiplizieren
[2T2] 65: mov faktor1,dx
[0T2] 66: mov faktor2,ax
[ND1] 67: mul bx
[P30] 68: ;
[6C6] 69: ; -Erstes Zwischenprodukt abspeichern
[423] 70: mov speicher1,dx
[U13] 71: mov speicher2,ax
[P30] 72: ;
[SA6] 73: ; -Nächstes Zwischenprodukt bilden und
[P14] 74: ; ebenfalls abspeichern
[SR2] 75: mov ax,faktor1
[ND1] 76: mul bx
[823] 77: mov speicher3,dx
[Y13] 78: mov speicher4,ax
[P30] 79: ;
[3Y6] 80: ; -Weiteres Zwischenprodukt errechnen und
[LB3] 81: ; im Speicher ablegen
[VR2] 82: mov ax,faktor2
[RD1] 83: mul cx
[C23] 84: mov speicher5,dx
[223] 85: mov speicher6,ax
[P30] 86: ;
[LZ6] 87: ; -Letztes Zwischenprodukt berechnen und
[8T4] 88: ; im Arbeitsspeicher sichern
[SR2] 89: mov ax,faktor1
[RD1] 90: mul cx
[G23] 91: mov speicher7,dx
[623] 92: mov speicher8,ax
[P30] 93: ;
[E77] 94: ; -Die untere Hälfte des 64-Bit-Ergebnisses
[8M4] 95: ; aus dem ersten und zweiten
[674] 96: ; Zwischenspeicher bilden
[V33] 97: mov ax,speicher2
[W33] 98: mov bx,speicher1
[Y22] 99: add bx,speicher4
[WZ2] 100: adc bx,speicher6
[P30] 101: ;
[L96] 102: ; -Die gleiche Operation für die obere
[N86] 103: ; Hälfte des Ergebnisses durchführen
[043] 104: mov cx,speicher3
[103] 105: add cx,speicher5
[003] 106: adc cx,speicher8
[643] 107: mov dx,speicher7
[P30] 108: ;
[GX4] 109: ; -Carry-Flag berücksichtigen
[AH1] 110: adc dx,0
[P30] 111: ;
[P30] 112: ;
[TV4] 113: ; -Rückkehr ins Hauptprogramm
[3P0] 114: ret
[P30] 115: ;
[R63] 116: ; -Ende der Prozedur
[A52] 117: multi32 endp
[P30] 118: ;
[P30] 119: ;
[0V3] 120: ; -Ende des Code-Segments
[BW1] 121: code ends
[P30] 122: ;
[P30] 123: ;
[W53] 124: ; -Ende des Programms
[M72] 125: end multi32
```

Listing 1. Diese Assembler-Routine demonstriert, wie 32-Bit-Werte ohne Berücksichtigung des Vorzeichens miteinander multipliziert werden.

```
[KF3] 1: ; Programm: Sdivisor
[NA2] 2: ; Sprache: SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[376] 6: ; -Divisionsroutine mit Änderung des
[F95] 7: ; Interrupts 0 (Division durch 0)
[NL2] 8: ; -SAM-4-Version
[P30] 9: ;
[P30] 10: ;
[X25] 11: ; -Definition des Code-Segments
[X54] 12: code segment para 'code'
```

```
[P30] 13: ;
[P30] 14: ;
[7D6] 15: ; -Festlegen des Unterprogrammnamens
[K03] 16: divint proc far
[P30] 17: ;
[PB3] 18: ; -Segment festlegen
[DK2] 19: assume cs:code
[P30] 20: ;
[VU6] 21: ; -Falls der Divisor gleich 0 ist, dann den
[DS6] 22: ; Interrupt 0 (Division durch 0) aufrufen
[TL1] 23: cmp bx,0
[V52] 24: jnz weiter
[C41] 25: int 0
[P30] 26: ;
[K56] 27: ; -Den alten Interrupt-Vektor auf dem
[XP2] 28: ; Stapel sichern
[D13] 29: weiter: mov di,0
[8U1] 30: mov es,di
[MB1] 31: seg es
[PV1] 32: push [di]
[BA1] 33: inc di
[BA1] 34: inc di
[MB1] 35: seg es
[PV1] 36: push [di]
[P30] 37: ;
[KF7] 38: ; -Die Adresse der neuen Division durch 0-
[BV6] 39: ; Routine im Interruptvektor 0 speichern
[XF1] 40: push ax
[U92] 41: mov ax,code
[EV1] 42: mov cx,ax
[RD1] 43: pop ax
[MB1] 44: seg es
[3E2] 45: mov [di],cx
[424] 46: mov cx,offset(neuint0)
[691] 47: dec di
[691] 48: dec di
[MB1] 49: seg es
[3E2] 50: mov [di],cx
[P30] 51: ;
[KZ3] 52: ; -Division durchführen
[7D1] 53: div bx
[P30] 54: ;
[V86] 55: ; -Falls kein Fehler aufgetreten ist,
[VB4] 56: ; dann BX-Register löschen
[LV1] 57: xor bx,bx
[BA1] 58: inc di
[BA1] 59: inc di
[P30] 60: ;
[BC7] 61: ; -Alten Interrupt-Vektor wiederherstellen
[5W2] 62: intback: seg es
[GT1] 63: pop [di]
[691] 64: dec di
[691] 65: dec di
[MB1] 66: seg es
[GT1] 67: pop [di]
[P30] 68: ;
[U06] 69: ; -Obere 16 Bit des Restwertes löschen
[PV1] 70: xor cx,cx
[P30] 71: ;
[TV4] 72: ; -Rückkehr ins Hauptprogramm
[3P0] 73: ret
[P30] 74: ;
[8X5] 75: ; -Rückkehradresse vom Stapel holen
[DV2] 76: neuint0: pop cx
[P30] 77: ;
[GZ4] 78: ; -Neue Rückkehradresse auf dem
[0L2] 79: ; Stapel ablegen
[L04] 80: mov cx,offset(intback)
[LGL] 81: push cx
[P30] 82: ;
[G45] 83: ; -Wenn der Dividend mehr als das
[6E6] 84: ; 65535fache des Divisors beträgt, dann
[NW4] 85: ; wird das Ergebnis auf diesem
[V54] 86: ; einfachen Weg berechnet
[XF1] 87: push ax
[DV1] 88: mov ax,dx
[TV1] 89: xor dx,dx
[7D1] 90: div bx
[VD1] 91: pop cx
[XF1] 92: push ax
[CV1] 93: mov ax,cx
[7D1] 94: div bx
[TD1] 95: pop bx
[P30] 96: ;
[Y56] 97: ; -Rückkehr von der Interrupt-Routine
[GR0] 98: iret
[P30] 99: ;
[R63] 100: ; -Ende der Prozedur
[P52] 101: divint endp
[P30] 102: ;
[P30] 103: ;
[0V3] 104: ; -Ende des Code-Segments
[BW1] 105: code ends
[P30] 106: ;
[P30] 107: ;
[W53] 108: ; -Ende des Programms
[042] 109: end divint
```

Listing 2. Die SAM-4-Fassung der Divisionsroutine unterscheidet sich beispielsweise in der indizierten Adressierung von der MASM-Variante



Assembler und Hardware

Musik liegt in der Luft

Zu den interessantesten Aufgaben der PC-Programmierung gehört die Ausgabe von Grafik und Musik, die im anbrechenden Zeitalter der Multimedia-Anwendungen immer größere Bedeutung gewinnt. Die Musikeigenschaften des PCs sind aufgrund der in dieser Hinsicht kargen Hardwareausstattung begrenzt. Trotzdem erzielen Sie unter Assembler mit wenigen Mitteln überraschend gute Ergebnisse.

Dieser Beitrag beschäftigt sich mit der Programmierung von Grafik und Musik in Maschinensprache. Der Umgang mit diesen beiden Themengebieten ist gar nicht so schwer, wie es auf den ersten Blick erscheint.

Sie müssen sich bei der Grafikdarstellung nur auf die Sonderzeichen des PC-Zeichensatzes beschränken und zum Erzeugen von Tönen die Register des Timer-Bausteins 8253 programmieren. Sie erfahren unter anderem, wie einfach es ist, Ihren Computer dazu zu bringen, daß er Ihnen ein Lied vorspielt, und wie Sie mit den Sonderzeichen des PC-Zeichensatzes und durch direktes Ansprechen des Bildschirmspeichers Grafiken programmieren und animieren. Viele höhere Programmiersprachen enthalten in ih-

rem Befehlssatz Kommandos, mit denen sich über den PC-Lautsprecher Töne ausgeben lassen. Bei Assembler-Sprachen ist dies jedoch nicht der Fall, so daß Sie selbst Routinen zur Tonausgabe schreiben müssen. Sobald Sie aber einmal verstanden haben, wie ein Ton prinzipiell erzeugt wird, ist dies nicht weiter schwer.

Das Grundprinzip der Tonerzeugung

Töne (und auch Geräusche) sind nichts anderes als Schwingungen in der Luft, das heißt rasch wechselnde Druckunterschiede. Sie entstehen beispielsweise dann, wenn sich die Membran eines Lautsprechers vor- und zu-

rückbewegt. Wenn mehrere Schwingungen in kurzen Zeitabständen aufeinander folgen, entstehen bei diesem Vorgang Töne.

● **Töne und Schwingungen:** Die Anzahl der Schwingungen innerhalb einer bestimmten Zeitspanne wird als Frequenz bezeichnet und in der Einheit Hertz (Schwingungen pro Sekunde) gemessen (Bild 1). Je mehr Schwingungen ein Lautsprecher in einem festen Zeitraum erzeugt, desto höher sind der dabei entstehende Ton und die Frequenz (Bild 2).

Man unterscheidet zwischen akustischen und musikalischen Tönen. Während akustische Töne beliebige Frequenzen besitzen können, sind die Frequenzen für die Töne einer musikalischen Oktave genau festgelegt.

● **Die musikalischen Töne:** Die Tabellen 1 bis 6 zeigen die Frequenzen für die Töne der sechs unteren Oktaven. Unterhalb der Oktave, die mit der Frequenz von 16,35 Hz beginnt, existiert keine weitere Oktave, weil der Mensch erst Schwingungen ab einer Frequenz von 16 Hz als Töne wahrnimmt. Alle Frequenzen unterhalb dieses Werts werden nur als Brummen oder Dröhnen empfunden.

Der Bezugspunkt für alle Töne in den Tabellen ist der Ton A in der fünften Oktave mit einer Frequenz von exakt 440 Hz. Dieser Ton wird auch als Kammerton A bezeichnet. Weil die Frequenz des Tons A in der benachbarten Oktave stets doppelt oder halb so groß ist wie die ursprüngliche Frequenz und eine Oktave aus 12 Halbtönen besteht, ist der Unterschied zwischen zwei benachbarten Halbtönen stets ein Faktor mit dem Wert der zwölften Wurzel aus 2. Das entspricht in etwa dem Wert 1,0594631.

● **Der Halbtone:** Um nun von der Frequenz eines Halbtons ausgehend die Frequenz des unmittelbar folgenden Halbtons zu berechnen, müssen Sie nur die Frequenz des ersten Halbtons mit 1,0594631 multiplizieren.

Tonerzeugung auf dem Rechner

Doch wie lassen sich jetzt mit den Informationen aus den Tabellen Töne erzeugen? Wenn der PC-Lautsprecher beispielsweise den Kammerton A erzeugen soll, müssen Sie den Timer-Baustein 8253, der im PC die Impulse für den Lautsprecher liefert, so programmieren, daß er 440mal in der Sekunde ein elektrisches Signal ausgibt, das die Lautsprechermembran zum Schwingen anregt. Zu diesem Zweck schreiben Sie in den 8253 über die

Ton	Frequenz Oktave 1
C	16,35
Cis	17,32
D	18,35
Dis	19,45
E	20,6
F	21,83
Fis	23,12
G	24,5
Gis	25,96
A	27,5
Ais	29,14
H	30,87

Tabelle 1. Die Töne der unteren Oktave 1 mit den zugehörigen Frequenzen

Portadresse 42hex einen Wert, der die gewünschte Frequenz angibt.

Doch im PC ist nicht nur der 8253 für die Ansteuerung des Lautsprechers zuständig. Fast alle Ein- und Ausgabeoperationen steuert der programmierbare Peripherie-Baustein 8255, der über drei parallele 8-Bit-Schnittstellen verfügt. Zwei dieser Schnittstellen sind für die Eingabe und eine ist für die Ausgabe von Daten zuständig.

Alle drei Schnittstellen des 8255 sind mit 8-Bit-Registern gepuffert. Die Register für die beiden Eingabeschnittstellen besitzen die Portadressen 60 und 62hex, und das Register für die Ausgabeschnittstelle wird über die Portadresse 61hex angesprochen. Die unteren beiden Bit des letzteren Registers steuern den PC-Lautsprecher.

● **Signal per Bit:** Ist das Bit 0 dieses Registers auf 1 gesetzt, so wird der zweite Kanal des Timer-Bausteins 8253 dazu veranlaßt, ein Rechtecksignal zu erzeugen, dessen Frequenz Sie über die 8253-Register definieren.

Ist auch Bit 1 des Registers unter der Portadresse 61hex auf 1 gesetzt, so wird das Rechtecksignal des 8253 an den Lautsprecher geführt, und dessen Membran erzeugt einen Ton. Die Frequenz dieses Tons entspricht der Frequenz des Rechtecksignals. Sobald Bit 1 auf 0 zurückgesetzt wird, ist auch das Rechtecksignal des 8253 wieder vom Lautsprecher getrennt, und dieser verstummt augenblicklich.

Der Ihnen vertrauteste Ton Ihres PCs ist wahrscheinlich das Piepsen, das nach dem Einschalten und dem folgenden Selbsttest des Computers ertönt. Auch dieses Piepsen wird über den Timer-Baustein 8253 und den Peripherie-Baustein 8255 gesteuert. Listing 1 demonstriert, wie man solch einen Ton auf dem PC erklingen läßt. Das Programm ist auf MASM und SAM assemblierbar.

Listing 2, das auf dem ersten Listing aufbaut, zeigt konkret, wie Sie mit dem PC-Lautsprecher Töne erzeugen und sogar eine ganze Melodie spielen

können. Es läuft ebenfalls unter MASM und SAM 4. Damit nicht die Tonlänge und -höhe von der Taktfrequenz des verwendeten Computers abhängt und störende Interrupts die Noten der Melodie verzerren, wird der Lautsprecher nicht direkt, sondern wie bereits beschrieben über den Timer-Baustein 8253 angesteuert.

Dieser erzeugt das Rechtecksignal für den Lautsprecher, ohne daß sich der Prozessor des PCs darum kümmern muß. Aus diesem Grund kann auch der weitere Programmablauf über den Prozessor die Tonerzeugung nicht beeinflussen.

● **Der Frequenzteiler:** Um dem 8253 einen bestimmten Ton zu entlocken, schreiben Sie in das Register mit der Portadresse 42hex einen 16-Bit-Wert, durch den die interne Taktfrequenz des 8253 geteilt wird. Das Ergebnis dieser Division gibt die Frequenz für das zu erzeugende Rechtecksignal an, das die Lautsprechermembran zum Schwingen bringt. Vor der Übergabe des Frequenzteilers laden Sie jedoch noch das Steuerregister des 8253 über die Portadresse 43hex mit dem Wert 182, um den Baustein für den Lautsprecherbetrieb zu initialisieren.

Die interne Taktfrequenz des 8253 beträgt 1 193 180 Hz. Das ist ein Zwölftel des PC-Systemtakts von 14,31818 MHz, aus dem beispielsweise auch der 4,77-MHz-Takt für den Prozessor gewonnen wird. Auch in PCs, die mit höheren Taktfrequenzen arbeiten, verwendet der 8253 intern einen Takt von 1 193 180 Hz, so daß dieser Wert allgemeingültig ist.

Möchten Sie beispielsweise mit dem 8253 den Kammerton A erzeugen, so schreiben Sie den Wert 2712 in das Register mit der Portadresse 42hex, weil 1 193 180 geteilt durch 2712 den Frequenzwert 440 ergibt. In Listing 2 wird der Frequenzteiler für das 8253-Register berechnet, indem die interne Taktfrequenz von 1 193 180 Hz durch die

Ton	Frequenz Oktave 2
C	32,7
Cis	34,65
D	36,71
Dis	38,89
E	41,2
F	43,65
Fis	46,25
G	49
Gis	51,91
A	55
Ais	58,27
H	61,74

Tabelle 2. Die Töne der unteren Oktave 2 mit den zugehörigen Frequenzen

Frequenz des gewünschten Tons geteilt wird. Dies läßt sich mit der Befehlsfolge

```
mov dx,12h
mov ax,34DCh
div di
```

realisieren. In diesem Beispiel wird die interne Taktfrequenz des 8253 als Hexadezimalwert 1234DChex in das Registerpaar DX:AX geschrieben und durch den Frequenzwert in DI geteilt. Das Ergebnis dieser Division entnehmen Sie dem Register AX.

● **Die Tonlänge:** Um für die verschiedenen Töne eine konstante Länge zu garantieren, ist in Listing 2 eine Zeiterschleife notwendig. Eine Zeitverzögerung von 10 ms erreichen Sie bei einem PC mit einer Taktfrequenz von 4,772727 MHz über folgende Schleife:

```
milli: mov cx,2808
schleif: loop schleif
```

Der Wert 2808 errechnet sich aus der Taktfrequenz des PCs und den Ausführungszeiten für die Assembler-Befehle. Der 8086/8088-Prozessor benötigt für den Mov-Befehl vier Taktzyklen und für die Loop-Funktion entweder 17 Takte (wenn eine Verzweigung stattfindet) oder fünf Takte. Weil die Schleife immer verzweigt, bis CX den

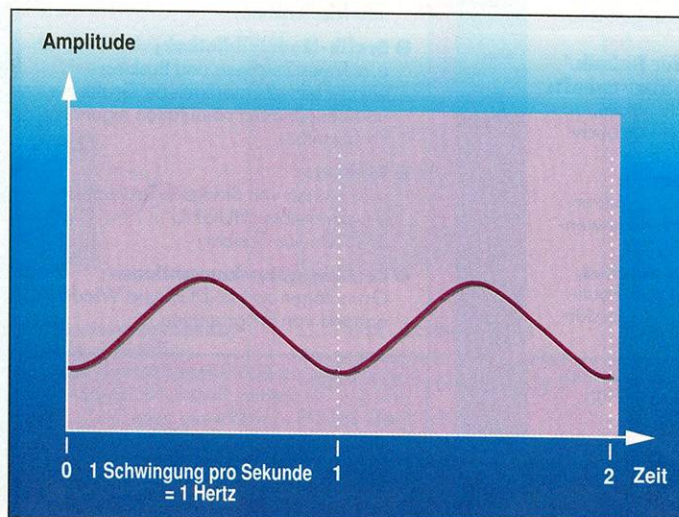


Bild 1. Hier sehen Sie eine Schwingung, die in einer Sekunde vollzogen wird; dies ist gleichbedeutend mit der Einheit 1 Hz.

Jetzt ist Ihr Erfolg vorprogrammiert!



Musterprogramme, Utilities und Modulbibliotheken in Turbo-Pascal

Dieses neue Nachschlagewerk bietet Ihnen u. a.:

- **detailliertes Know-how zum Turbo-Pascal-System;**
- **Betriebssystem-Routinen:** u. a. ROM-BIOS und IBM-Hardware (BIOS-Variable, Adapter für asynchrone Datenübertragung, Druckeransteuerung);
- **hardwarenahe Programmierung:** u. a. Disk-Optimizer, Floppy-Tuning, Prefix-Segment und Environment, Expanded Memory (EMS);
- **raffinierte Utilities:** **Schnittstellenutilities** (u. a. Turbo-Inliner, Zugriff auf dBASE-Dateien), **Grafikutilities** (Ausdruck von Farbgrafiken als Schwarz/Weiß-Muster), **Datenschutz** (u. a. Verschlüsseln von Dateien);
- **grafische Benutzeroberfläche SISGRAF:** GEM-ähnliche Benutzeroberfläche zur Einbindung in eigene Grafikprogramme oder zur Programmierung von Systemsteuerungen à la WINDOWS;
- **Anwenderprogramme für Technik/Mathematik und Betriebswirtschaft:** u. a. statistische Qualitätskontrolle, Schrittmotor-Simulation, Finanzbuchhaltung SOBUCH;
- **Entwicklungsprogramme:** Editoren (Texteditor in Pascal, Vektor-Zeichensatzeditor), Datenbankanwendungen (INTERDAT);
- **Turbo-Pascal-Modulbibliotheken,** u. a. Datum-, String- und DOS-Routinen, Mausfunktionen, EMS-Unterstützung und virtuelle Arrays.

Grundwerk ca. 600 Seiten, DIN A4, inkl. drei 5 1/4"-Disketten, Bestell-Nr.: 2500, Preis: DM 98,-, wahlweise zwei 3 1/2"-Disketten (gegen DM 4,- Aufpreis)



Komfortable C-Musterlösungen

Für den Programmierer, der sich Schritt für Schritt die Sprache C aneignen will, wie auch für den versierten C-Praktiker.

Aus dem Inhalt:

- **Das C-System im Detail:** Sprachkonzept, Programmierungselemente, wichtige Bibliotheksfunktionen von Microsoft- und Turbo-C sowie C++-Grundlagen;
- **Kurse zur Dateiverwaltung:** u. a. verschiedene Dateiformate, Kopieren von Dateien, MS-Word-Dateiformat, indexsequentielle Dateien für schnelles Auffinden von Datensätzen, Zugriff auf Fremddateien am Beispiel von dBase;
- **Hardwarenahe Programmieren:** u. a. Interruptfunktionen für die serielle Schnittstelle, Port Ein- und Ausgabe, BIOS-Funktionen, File-Transfer und Termalemulation, TSR-Programmierung ...;
- **Tools:** z. B. Font-Editor für die VGA-Karte, Maus-Funktionen, PRINT Spooler Interfacefunktion;
- **Grafik-Modulbibliothek:** u. a. freies Zeichnen und Business-Grafik, bildschirmorientierte Grafik zur überzeugenden Präsentation eigener Programme;
- **Utilities:** u. a. Suchen von Strings in Textdateien, Drucktreiber „HUGHU“, DOS-Device-Treiber;
- **Betriebssystemkonventionen:** Grundlagen zu MS-DOS und Windows anhand von Programmen.

Grundwerk ca. 800 Seiten, DIN A4, inkl. drei 5 1/4"-Disketten, Bestell-Nr.: 3600, Preis: DM 98,-, wahlweise zwei 3 1/2"-Disketten (gegen DM 4,- Aufpreis)



Schöpfen Sie BASIC voll aus!

Dieses speziell für den Programmierer entwickelte Nachschlagewerk zeigt Ihnen, wie Sie schnell und sicher in Basic programmieren.

Sie erhalten u. a.:

- **eine detaillierte Befehls- und Funktionsreferenz** zu allen wichtigen Basic-Dialekten, u. a. GW-, Power-, Quick-, Q- und Visual-Basic.
- **praxisorientierte Musterprogramme,** u. a.: **Datenbanksystem INBASE:** Sie bearbeiten Datensätze, editieren Texte, schreiben Serienbriefe. Zusätzlich erhalten Sie fertige Datenbank-Lösungen wie z. B. Adreßverwaltung, Etikettendruckprogramm; **Personalzeitsystem INPERS:** komfortable Verwaltung von bis zu 60 Mitarbeitern;
- **Basic-Programmierkurs:** für den sicheren Umgang mit Datentypen, Kontrollstrukturen, Einsatz der Interrupt-Programmierung ...;
- **Konvertierungsprogramm CONBAS:** zur Portierung Ihrer Programme von GW-Basic nach Quick- oder Visual-Basic;
- **Visual-Basic unter Windows:** mit umfassenden Befehlsübersichten und VB-Musterlösungen erstellen Sie Ihre eigenen Windows-Programme!
- **raffinierte Tools und Utilities,** z. B. einen Ausgabemasken-Interpreter, mit dem Sie ASCII-Bildschirmmasken bei Laufzeit der Basic-Applikation interpretieren und anzeigen können, komplette Directory-Verwaltung, komfortable Mausemulation u. v. m.

Grundwerk ca. 450 Seiten, DIN A5, inkl. zwei 5 1/4"-Disketten, Bestell-Nr.: 7200, Preis: DM 98,-, wahlweise 3 1/2"-Diskette (gegen DM 2,- Aufpreis)

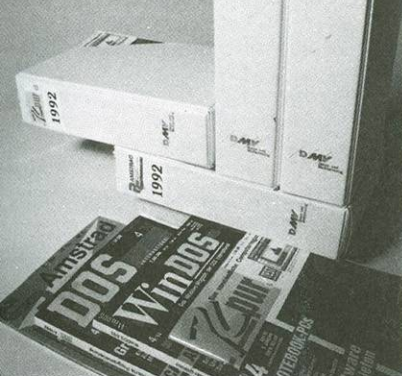
INTEREST-VERLAG
Fachverlag für Special Interest
Publikationen und Anwendersoftware
Römerstraße 16
W-8901 Kissing
Tel. 0 82 33 / 211-0
Fax 0 82 33 / 211-299



Zu jedem dieser Werke erhalten Sie alle
2-3 Monate Erweiterungsausgaben mit
je ca. 130 Seiten (bei Bestell-Nr. 7200:
100 Seiten), inkl. Diskette, zum Preis von
DM 59,- (Abbestellung jederzeit möglich).

Für Ihre Anforderung
verwenden Sie bitte neben-
stehende Bestellkarte!

Zum
Sammeln
unentbehrlich



Die DMV-Sammelordner

DOS INTERNATIONAL

Vorsprung

für

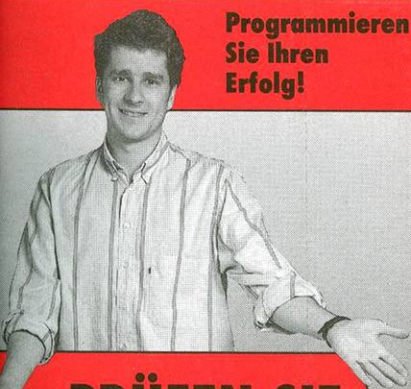
PC-Personalcomputer

mit

DOS EXTRA

im

Abonnement



**Programmieren
Sie Ihren
Erfolg!**

PRÜFEN SIE

Ihr Nachschlagewerk
in Ruhe zu Hause.
Wenn Sie nicht überzeugt
sind, schicken Sie dieses
mit den Disketten inner-
halb von 10 Tagen an
uns zurück. Damit ist die
Sache für Sie erledigt.

Sonderhefte • Bücher • Sammelordner

Ausgabe

DOS EXTRA
DM 18,-

☐ 356	Nr. 3
☐ 381	Nr. 4
☐ 382	Nr. 5
☐ 383	Nr. 6
☐ 384	Nr. 7
☐ 385	Nr. 8
☐ 387	Nr. 10
☐ 389	Nr. 12
☐ 391	Nr. 13
☐ 392	Nr. 14
☐ 393	Nr. 15
☐ 394	Nr. 16
☐ 395	Nr. 17
☐ 396	Nr. 18
☐ 397	Nr. 19
☐ 398	Nr. 20
☐ 399	Nr. 21

DATABOX

2 Stk. 5 1/4"-Disk
DM 35,-

☐ 357	
☐ 3811	
☐ 3813	
☐ 3815	
☐ 3817	
☐ 3819	
☐ 3823	
☐ 3827	
☐ 3829	
☐ 3831	
☐ 3833	
☐ 3835	
☐ 3837	
☐ 3839	
☐ 3841	
☐ 3843	

1 Stk. 3 1/2"-Disk
DM 35,-

☐ 358	
☐ 3812	
☐ 3814	
☐ 3816	
☐ 3818	
☐ 3820	
☐ 3824	
☐ 3828	
☐ 3830	
☐ 3832	
☐ 3834	
☐ 3836	
☐ 3838	
☐ 3840	
☐ 3842	
☐ 3844	

WinDOS-Sonderhefte
DM 18,- mit Diskette im Heft

☐ 30200	Nr. 1	☐ 30201	Nr. 2
--------------------------------	-------	--------------------------------	-------

DMV-ComputerWissen
DM 18,-

☐ 3511	Band 1, Basic Toolbox
☐ 3514	Band 2, Textverarbeitung
☐ 3517	Band 3, Tips und Tricks
☐ 3520	Band 4, Datenbanken

5 1/4"-Diskette
DM 35,-

☐ 3512	
☐ 3515	
☐ 3518	
☐ 3521	

3 1/2"-Diskette
DM 35,-

☐ 3513	
☐ 3516	
☐ 3519	
☐ 3522	

DOS toolbox DM 28,- mit Diskette im Heft

☐ 30300	Heft Nr. 1	☐ 30301	Heft Nr. 2
☐ 30302	Heft Nr. 3	☐ 30303	Heft Nr. 4

Bücher

Best.-Nr.

DM

☐ 60500	DOS-Taschenbuch Band 1	29,80
☐ 60510	DOS-Taschenbuch Band 2	29,80
☐ 60520	DOS-Taschenbuch Band 3	29,80
☐ 60530	DOS-Taschenbuch Band 4	29,80
☐ 40200	PC Kompakt - ConText 4.0	19,80
☐ 40210	PC Kompakt - Windows 3.0	19,80

DOS-
Taschenbücher

5 1/4"-Diskette
DM 24,-

3 1/2"-Diskette
DM 24,-

BAND 1	☐ 60501	☐ 60502
BAND 3	☐ 60521	☐ 60522
BAND 4	☐ 60531	☐ 60532

Sammelordner

2 Stk. 15,80

☐ 530 DOS	☐ 531 CPC International
☐ 535 PCpur	☐ 529 WinDOS

Zwischensumme + Porto und Verpackung
(Inland 4,- DM, Ausland 6,- DM)

Gesamtbetrag: _____ DM

Dauerbezug DOS EXTRA DATABOX

☐ Hiermit bestelle ich »DOS EXTRA DATABOX« im Dauerauftrag mit Kündigungsmöglichkeit zum vier-
teljährlichen Erscheinungstermin. **Achtung:** Nur im Bankabbuchungsverfahren möglich!

Preise BRD:

je 39,- DM

Auslandspreise Europa:

je 41,- DM

Außereuropäisches Ausland:

je 42,- DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Diskettenformat: ☐ IBM 5 1/4" ☐ IBM 3 1/2"

☐ **Bequem und bargeldlos durch Bankabbuchung**

Bankleitzahl

Konto-Nr./Inhaber

Geldinstitut

☐ **Gegen Rechnung** – zahlbar innerhalb zwei Wochen nach Erhalt.
(Bitte keine Vorauszahlung leisten – Rechnung abwarten.)

Dieses Abonnement verlängert sich automatisch um 6 bzw. 12 Ausgaben, wenn es nicht sechs Wochen vor Ablauf schriftlich
gekündigt wird.

Datum 1. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte unbedingt auf der Vorderseite eine zweite Unterschrift leisten!

Ja, senden Sie mir sofort:

Bitte abtrennen oder ausschneiden!

(Zutreffendes bitte ankreuzen)



☐ Expl. „Musterprogramme,
Utilities und Modulbiblio-
theken für raffinierte Lösun-
gen in Turbo-Pascal“

Praktischer Ringbuchordner,
DIN A4, ca. 600 Seiten, inkl. drei
5 1/4"-Disketten, Bestell-Nr.: 2500,
Preis: DM 98,-, wahlweise zwei
3 1/2"-Disketten.

Disketten-Service

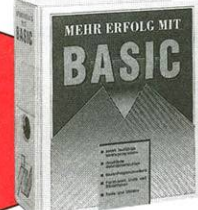
Ich wünsche die Disketten im
☐ 3 1/2"-Format (gegen DM 2,- Auf-
preis pro Diskette)



☐ Expl. „Erfolgreiches
Programmieren mit komfor-
tablen Musterlösungen in C“

Stabiler Ringbuchordner,
DIN A4, ca. 800 Seiten, inkl. drei
5 1/4"-Disketten, Bestell-Nr.: 3600,
Preis: DM 98,-, wahlweise zwei
3 1/2"-Disketten.

Zu diesen Werken erhalten Sie alle
2-3 Monate Erweiterungsausgaben mit
je ca. 130 Seiten (bei Bestell-Nr. 7200:
100 Seiten), inkl. Diskette, zum Preis von
DM 59,- (Abbestellung jederzeit möglich).



☐ Expl. „Mehr Erfolg
mit Basic“

Praktischer Ringbuchordner,
DIN A5, ca. 450 Seiten, inkl. zwei
5 1/4"-Disketten, Bestell-Nr.: 7200,
Preis: DM 98,-, wahlweise
3 1/2"-Diskette.

Nicht vergessen!

Geben Sie bitte umgehend Ihre
vollständige Adresse an.



PC-Sonderhefte

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/Ort

- ☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.
☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

W-3440 Eschwege

Bitte
ausreichend
frankieren

Informieren

DOS-Taschenbücher und DOS-EXTRA-Sonderhefte sind eine zuverlässige Informationsquelle für jeden PC-Besitzer. Nutzen auch Sie dieses Wissen, und vertiefen Sie es mit den dazu angebotenen Programmen aus dem Disketten-Service. Den PC beherrschen mit

- DOS-EXTRA-Sonderheften
- DOS-Taschenbüchern
- DOS-Disketten-Service

Zur Erinnerung:

Meine DOS-Sonderhefte, DOS-Taschenbücher, DOS-Disketten wurden bestellt am

DOS EXTRA DATABOX

Abonnement

Rechnungsanschrift

Vorname, Name

Straße, Nr./Postfach

PLZ/Ort

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, W-3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung meines Widerrufs Schreibens zur Fristwahrung ausreicht. Ich bestätige das durch meine zweite Unterschrift.

Datum 2. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

W-3440 Eschwege

Bitte
ausreichend
frankieren

Abonnieren

Jeden Monat neu, vermittelt Ihnen die DOS EXTRA DATABOX wichtige Informationen zu Ihrem Personal-Computer. Nutzen Sie dieses Wissen mit einem Abonnement zu Ihrem ganz persönlichen

- Preisvorteil
- Informationsvorsprung
- Zustellkomfort

Zur Erinnerung:

Mein persönliches Abonnement wurde bestellt am

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, W-3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung des Widerrufs Schreibens zur Fristwahrung ausreicht.

Meine Anschrift

Name

Vorname

Straße, Haus-Nr.

PLZ, Ort

Unterschreiben Sie hier bitte Ihre Bestellung!

Bei Minderjährigen ist die Unterschrift eines gesetzlichen Vertreters erforderlich. Ohne Ihre Unterschrift kann die Bestellung nicht bearbeitet werden.

Datum Unterschrift

Bitte unterschreiben Sie auch Ihre Sicherheitsgarantie,

mit der Sie folgendes zur Kenntnis nehmen: Sie haben das Recht, Ihr angefordertes Werk innerhalb von 10 Tagen ab Lieferung an den INTEREST-VERLAG GmbH, Römerstraße 16, W-8901 Kissing, zurückzusenden, wobei für die Fristwahrung das Datum der Absendung genügt. Sie kommen dadurch von allen Verpflichtungen aus der Bestellung frei.

Datum Unterschrift

IN 110 421

60 Pfennig,
die sich
lohlen!

Postkarte/Antwort

INTEREST-VERLAG

Fachverlag für Special Interest
Publikationen und Anwendersoftware
z. Hd. Herrn Michel

Römerstraße 16

W-8901 Kissing

Verlagsgarantie

- Sie erhalten von mir PC-Fachinformationen, von absoluten Spezialisten für Sie geschrieben.
- Ihr Nachschlagewerk können Sie in aller Ruhe 10 Tage zu Hause prüfen und bei Nichtgefallen innerhalb dieser Frist zurücksenden. Der INTEREST-VERLAG gewährt nach Zahlung des Rechnungsbetrages das Recht, die beiliegende Software zu testen und zeitlich unbeschränkt zu nutzen.
- Ihr Nachschlagewerk ist immer aktuell. Dafür sorgt unser Erweiterungsservice.
- Den Erweiterungsservice können Sie jederzeit kündigen. Sie gehen also kein Risiko ein. Darauf gebe ich Ihnen mein Wort.

Ihr INTEREST-VERLAG
Fachverlag für Special Interest
Publikationen und Anwendersoftware

W. Michel
Wolfgang Michel
Verlagsleiter

Kompetenz

für
PC-Personalcomputer
im
Abonnement
deutlich
günstiger

Information

über
PC-Personalcomputer
sammeln Sie mit
• DOS-Einzelheften
• DATABOX
mit mehr als 700 KByte
Software

Sichern Sie sich heute noch
Ihre Datenbank für morgen:

Ergo-Base für Windows

Einfach die Bestellkarte
ausfüllen und einsenden!

Abo-Order DOS International

Hiermit bestelle ich »DOS International« für mindestens

- ☐ 12 Ausgaben
Preise: BRD
Auslandspreise: Europa
Außereuropäisches Ausland
Lieferung erfolgt ab der nächsterreichbaren Ausgabe.
- ☐ 6 Ausgaben
12 Ausgaben 88,50 DM
6 Ausgaben 45,- DM
12 Ausgaben 120,- DM
6 Ausgaben 60,- DM
12 Ausgaben 160,- DM
6 Ausgaben 80,- DM

Lieferanschrift

☐ Geschäft ☐ Privat

Vorname, Name

Firma (nur wenn Lieferadresse)

Straße, Nr./Postfach

PLZ/Ort (Bitte genaue Anschrift angeben!)

Dieses Abonnement verlängert sich automatisch um 6 bzw. 12 Ausgaben, wenn es nicht sechs Wochen vor Ablauf schriftlich gekündigt wird.

Datum 1. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Abo-Order DOS DATABOX

Hiermit bestelle ich »DOS DATABOX« für mindestens

- ☐ 12 Ausgaben
Diskettenformat:
Inlandspreise:
Europa:
Außereurop. Ausland:
Lieferung erfolgt ab der nächsterreichbaren Ausgabe.
- ☐ 6 Ausgaben
☐ IBM 5 1/4" ☐ IBM 3 1/2"
- 12 Ausgaben 300,- DM, 6 Ausgaben 150,- DM
12 Ausgaben 320,- DM, 6 Ausgaben 160,- DM
12 Ausg. 360,- DM, 6 Ausg. 180,- DM

☐ Bequem und bargeldlos durch Bankabbuchung

Bankleitzahl

Konto-Nr./Inhaber

Geldinstitut

☐ Gegen Rechnung – zahlbar innerhalb zwei Wochen nach Erhalt. (Bitte keine Vorauszahlung leisten – Rechnung abwarten.)

Bitte unbedingt auf der Vorderseite
eine zweite Unterschrift leisten!

DOS-Einzelhefte und DATABOX

Einzelheft Ausgabe 7,50 DM/Stck.	DATABOX 5 1/4"-Diskette 24,- DM/Stck.	DATABOX 3 1/2"-Diskette 24,- DM/Stck.	Einzelheft Ausgabe 7,50 DM/Stck.	DATABOX 5 1/4" 2 Disketten 24,- DM/Stck.	DATABOX 3 1/2"-Diskette 24,- DM/Stck.
☐ 2/90	☐	☐	☐ 1/92	☐	☐
☐ 3/90	☐	☐	☐ 2/92	☐	☐
☐ 4/90	☐	☐	☐ 3/92	☐	☐
☐ 6/90	☐	☐	☐ 4/92	☐	☐
☐ 7/90	☐	☐	☐ 5/92	☐	☐
☐ 8/90	☐	☐	☐ 6/92	☐	☐
☐ 10/90	☐	☐	☐ 7/92	☐	☐
☐ 11/90	☐	☐	☐ 8/92	☐	☐
☐ 12/90	☐	☐	Einzelheft Ausgabe 8,- DM/Stck.	DATABOX 5 1/4"-Diskette 24,- DM/Stck.	DATABOX 3 1/2"-Diskette 24,- DM/Stck.
☐ 1/91	☐	☐	☐ 9/92	☐	☐
☐ 2/91	☐	☐	☐ 10/92	☐	☐
☐ 3/91	☐	☐	☐ 11/92	☐	☐
☐ 4/91	☐	☐	☐ 12/92	☐	☐
☐ 5/91	☐	☐	☐ Gratis-Sammelordner bei Bestellung von 6 Einzelheften		
☐ 6/91	☐	☐			
☐ 7/91	☐	☐			
☐ 8/91	☐	☐			
☐ 10/91	☐	☐			
☐ 11/91	☐	☐			
☐ 12/91	☐	☐			

+ Porto/Verpackung (Inland 4,- DM, Ausland 6,- DM) DM Gesamtbetrag DM

PC-Software

5 1/4"	3 1/2"		DM
23013 ☐	23012 ☐	Fraktal Animator 3D	69,-
Power Hits			
20614 ☐	20615 ☐	Text Compiler	29,80
20600 ☐	20601 ☐	Multy Copy	29,80
20612 ☐	20613 ☐	PC-Astro	29,80
20608 ☐	20609 ☐	Menue	29,80
20618 ☐	20619 ☐	PC-Spiel des Wissens 1	29,80
20616 ☐	20617 ☐	Fraktal 3D	29,80
20610 ☐	20611 ☐	Terminplaner	29,80
20606 ☐	20607 ☐	Tippsy	29,80
Happy Windows Collection			
20800 ☐	20801 ☐	Adressen Manager	49,80
20802 ☐	20803 ☐	Icon Designer	49,80
20804 ☐	20805 ☐	Literatur-Verwaltung	49,80
20806 ☐	20807 ☐	Briefmarken-Verwaltung	49,80
20808 ☐	20809 ☐	Musik-Archiv	49,80
20810 ☐	20811 ☐	Video-Verwaltung	49,80
20812 ☐	20813 ☐	Bobby Im Tal der Könige (4 Spiele)	49,80
20814 ☐	20815 ☐	Bildschirm-Schoner (Dark Fader)	49,80

+ Porto/Verpackung (Inland 4,- DM, Ausland 6,- DM) DM

60 Pfennig,
die sich
lohnern

Antwortkarte

DMV-Verlag
Versandservice
Postfach 250

W-3440 Eschwege



Abonnement

Rechnungsanschrift

Vorname, Name

Straße, Nr./Postfach

PLZ/Ort

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, W-3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung meines Widerrufs Schreibens zur Fristwahrung ausreicht. Ich bestätige das durch meine zweite Unterschrift.

Datum 2. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte
ausreichend
frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

W-3440 Eschwege

1. Abonnieren

Jeden Monat neu, vermitteln Ihnen die DOS International und DOS EXTRA wichtige Informationen zu Ihrem Personal-Computer. Nutzen Sie dieses Wissen mit einem Abonnement zu Ihrem ganz persönlichen

- Preisvorteil
- Informationsvorsprung
- Zustellkomfort

Zur Erinnerung:

Mein persönliches Abonnement wurde bestellt am

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, W-3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung des Widerrufs Schreibens zur Fristwahrung ausreicht.



Einzelhefte

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/Ort

- ☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.
☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte
ausreichend
frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

W-3440 Eschwege

2. Sammeln

DOS-International-Leser sind immer top-informiert. Daher sollte Ihnen kein Heft oder Sonderheft in Ihrer Sammlung fehlen. Das Eintippen von Listings und Programmen ersparen Sie sich durch unseren günstigen Disketten-Service. Immer gut beraten sind Sie mit DOS International

- Einzelheften
- Sonderheften
- Disketten-Service

Zur Erinnerung:

Meine Hefte und Disketten wurden bestellt am

Ja, senden Sie mir
Expl. **ERGO-Base für Windows**
zum Einzelpreis von 199,95 DM
zuzüglich Porto und Verpackung
(Inland 4,— DM, Ausland 6,— DM)

- ausführliches Benutzerhandbuch

- Programm-Disketten im Format

■ 5,25 Zoll

■ 3,5 Zoll

Bestell-Nr. 21010

Bestell-Nr. 21011

Bitte kreuzen Sie die gewünschte Zahlungsweise an:

- beigefügter Verrechnungsscheck
- per Nachnahme
- nach Erhalt der Rechnung

**Die fantastische Datenbank
mit Bild und Ton**

Meine Anschrift:

Name

Vorname

Straße, Hausnummer

PLZ/Ort

Telefon für eventuelle Rückfragen

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Sichern Sie sich heute noch
Ihre Datenbank für morgen:

Ergo-Base für Windows

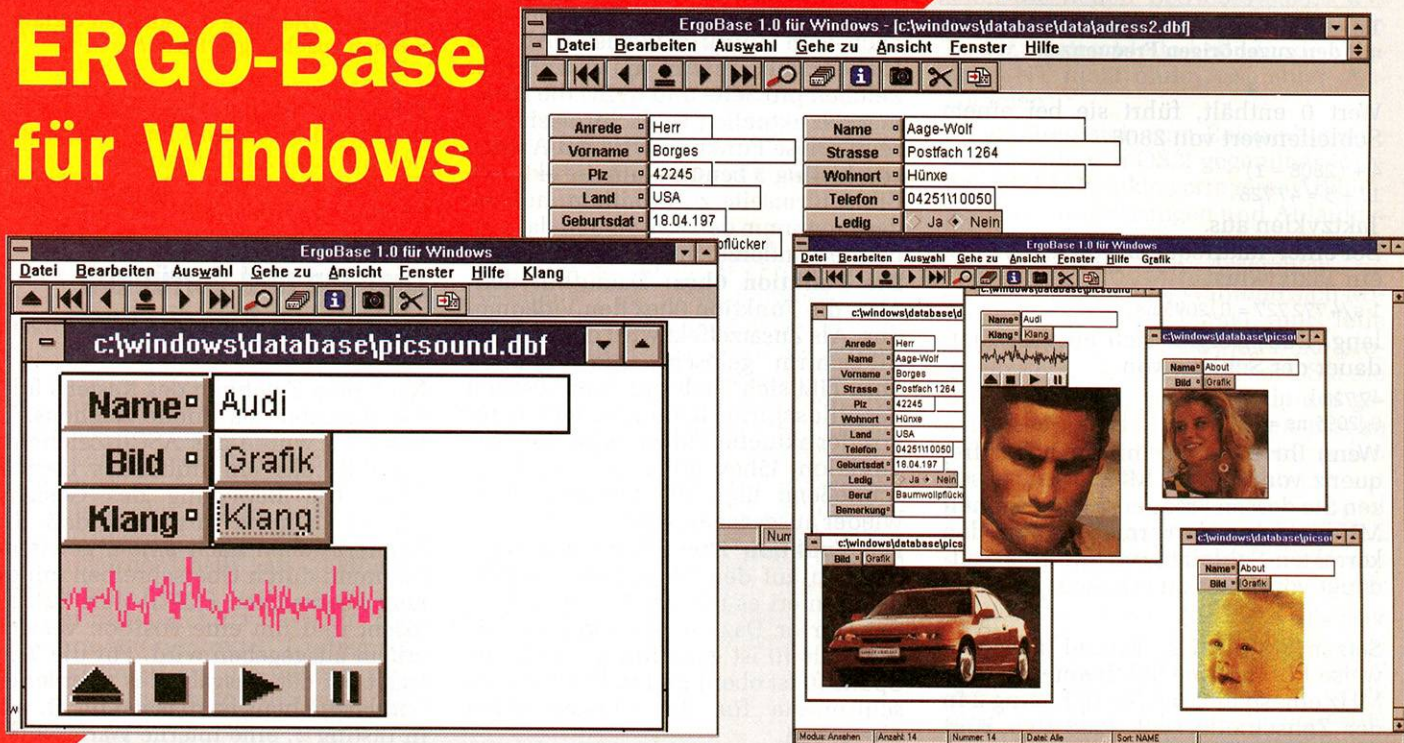
Einfach die Bestellkarte
ausfüllen und einsenden!

Unglaublich!

Datenbank mit Bild und Ton

Fantastische Windows-Leistung zum **Superpreis**

ERGO-Base für Windows



Ton und Bild machen Ihre Datenbank "dreidimensional"

Jetzt ist es endlich möglich: Ihre Adressen, Briefmarken, Videos oder CDs in einer Datenbank mit integrierten Bildern und Ton. Ihre Daten erreichen bisher unmöglich geglaubte Aussagekraft. **Praktikables Multimedia zum erschwinglichen Preis.**

ERGO-Base für Windows bietet aber auch das Leistungsspektrum dBase-kompatibler Datenbanken für den professionellen Einsatz, wie zum Beispiel als Personaldatenbank mit Paßbildern und Sprechprobe. ERGO-Base hat durch "C-Baum-Technik" auch die "Power" für Datenbanken, die Hunderte von Megabyte groß sind.

Sicherheit und leichte Einarbeitung

ERGO-Base ist übersichtlich, weil es Sie immer nur mit der Funktionsauswahl "behehlt", die in Ihrer aktuellen Arbeitssituation auch sinnvoll ist. Möchten Sie z.B. mit Grafiken arbeiten, werden Ihnen die entsprechenden Menübefehle zur Verfügung gestellt. Wenn Sie zur normalen Datenbearbeitung zurückwechseln, verschwindet das überflüssig gewordene Grafikmenü. ERGO-Base hält Ihnen den Blick auf das für Sie Wesentliche frei.

Individuelle Gestaltung und intuitive Bedienung

ERGO-Base verschont Sie mit Fachbegriffen wie "Eingabemodus" oder "Maskenmodus". Die Gestaltung und das Aussehen Ihrer Datenbank können Sie jederzeit ändern, unabhängig davon, was Sie gerade mit der Datenbank machen. Einfaches "Anfassen" und "Verschieben" mit der Maus genügt – ERGO-Base für Windows macht "Einschalten und Loslegen" wahr!

Händleranfragen erwünscht

Etiketten, Listendruck und Serienbriefe par excellence

Mit ERGO-Base sind Sie im Handumdrehen in der Lage, sowohl ansprechende Listen (Reports) zu erstellen als auch Etiketten zu generieren. Zur Gestaltung der Etiketten steht Ihnen ein komplettes Programm-Modul **mit DTP-Funktionen** zur Verfügung, mit dem Sie auch Geburtstagskarten oder Miniposter erstellen können. Durch die Serienbriefanbindung werden ERGO-Base und Ihre Textverarbeitung zu einem unschlagbaren Team.

Schon eingebaut: Unterstützung für Word für Windows, AmiPro und WordStar für Windows

ERGO-Base für Windows im Überblick

- Hoher Bedienungskomfort
- Extrem hohe Geschwindigkeit
- dBase-kompatibel
- Multimedia-Unterstützung mit Sound und Grafik
- Über 2 Milliarden Datensätze, 255 Felder pro Datensatz
- Feldtypen-Text, numerisch, Gleitpunkt, Datum, Logik, Memo, Grafik und Sound
- Unbegrenzte Datenmengen bei Memo, Grafik und Sound
- Leistungsfähige Such- und Reportfunktionen
- Speicherplatzersparnis durch Kompression
- Beliebige, variable Eingabemasken
- Serienbriefanbindung und Etikettendruck
- DTP-Funktionen zur Gestaltung von Etiketten, Karten oder Minipostern
- Kompatibilität mit allen gängigen Windows-Textverarbeitungen

ERGO-Base für Windows

ICON-Designer für Windows

49,80*

nur DM
199,95*

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung. Bitte benutzen Sie die Bestellkarte im Heft. Word für Windows, AmiPro, WordStar für Windows, dBase sind Warenzeichen der jeweiligen Hersteller.

Ton	Frequenz Oktave 3
C	65,41
Cis	69,3
D	73,42
Dis	77,78
E	82,41
F	87,31
Fis	92,5
G	98
Gis	103,83
A	110
Ais	116,54
H	123,47

Tabelle 3. Die Töne der unteren Oktave 3 mit den zugehörigen Frequenzen

Wert 0 enthält, führt sie bei einem Schleifenwert von 2808

$$4 + (2808 - 1)$$

$$17 + 5 = 47728$$

Taktzyklen aus.

Bei einer Taktfrequenz von 4,772727 ist ein Taktzyklus

$$1 \text{ s} / 4772727 = 0,2095 \text{ ns}$$

lang. Damit ergibt sich eine Gesamtdauer der Schleife von

$$47728 \times$$

$$0,2095 \text{ ns} = 10 \text{ ms}$$

Wenn Ihr PC nicht mit einer Taktfrequenz von 4,772727 MHz arbeitet, setzen Sie dessen Frequenz in der Einheit MHz in folgende Formel ein, um den korrekten Schleifenwert für eine Zeitdauer von 10 ms zu erhalten:

$$\text{wert} = (n / 10^{-4} - 9) / 17 + 1$$

Setzen Sie in diese Formel beispielsweise für »n« eine Taktfrequenz von 8 MHz ein, so müssen Sie in Listing 2 in der Zeitschleife statt 2808 den Wert 4706 eintragen.

Das Grundprinzip der Grafikprogrammierung

Bei der Grafikprogrammierung gibt es grundsätzlich zwei Wege, Grafiken beziehungsweise Zeichen darzustellen: erstens mit Hilfe des PC-Zeichensatzes und zweitens durch direkte Ansteuerung des Grafikspeichers. Zuerst soll der PC-Zeichensatz erläutert werden.

● **Zeichen via Interrupt 10hex:** Zu diesem Zweck demonstriert Listing 3, wie sich mit den Grafikzeichen ein Kasten zeichnen und ein einzelnes Sonderzeichen animieren, das heißt in Bewegung setzen läßt. Dazu greift das Programm über den Video-Interrupt 10 auf den Bildschirm zu.

Der Video-Interrupt 10hex besteht aus 16 Funktionen, die ausführlich in der Fachliteratur beschrieben werden. Es existieren zwar auch beim MS-DOS-Interrupt 21hex einige Funktionen für die Bildschirmausgabe, doch eignen sich diese nur für die Textausgabe, weil sie ausschließlich zeichenorien-

tiert arbeiten. Die Funktionen des Video-Interrupts 10hex sind weitaus leistungsfähiger und vielfältiger. Sie können Bildpunkte setzen und löschen, Bildschirmbereiche aufwärts und abwärts rollen sowie einzelne Bildschirmpositionen auf Grafikpunkte oder Textzeichen hin untersuchen.

● **Die Interrupt-Funktionen im Detail:** Für unsere Zwecke sind die Funktionen 0hex, 2hex, 10hex und 15hex des Video-Interrupts interessant.

Funktion 15hex: Die Funktion 15hex ermittelt den Videomodus. Sie gibt nach dem Aufruf in Register AL den Videomodus, in AH die Anzahl der Zeichen pro Seite und in BH die Nummer der aktuellen Bildschirmseite zurück. Diese Funktion wird am Anfang von Listing 3 benutzt, um die aktuelle Bildschirmseite zu ermitteln, auf die das Programm den Kasten und das animierte Sonderzeichen ausgibt.

Die Funktion 0hex: Daraufhin definiert die Funktion 0hex den Videomodus. Als Zusatzeffekt wird der gesamte Bildschirm gelöscht. Die Funktion 0hex läßt sich auch nur zum Löschen des Bildschirms benutzen, indem zuerst der aktuelle Videomodus über die Funktion 15hex eingelesen und anschließend über die Funktion 0hex wieder ausgegeben wird.

Die Funktion 2hex: Bevor Listing 3 Zeichen auf den Bildschirm ausgibt, positioniert es mit der Funktion 2hex den Cursor. Dazu wird im Register DH die Zeile (0 ist ganz links), in DL die Spalte (0 ist oben) und in BH die Bildschirmseite für die Cursorposition übergeben.

Die Funktion 10hex: Die wichtigste Aufgabe in Listing 3 fällt der Funktion 10hex zu, denn diese Funktion schreibt auf die vorgegebene Bildschirmseite an die aktuelle Cursorposition ein oder mehrere Zeichen. Die Funktion versetzt den Cursor nicht auf die nächste Bildschirmposition und behält die Farbe der an der Ausgabeposition befindlichen Zeichen bei. Beim Aufruf der Funktion 10hex enthält das Register AL den ASCII-Wert des auszu-

Ton	Frequenz Oktave 4
C	130,81
Cis	138,59
D	146,83
Dis	155,56
E	164,81
F	174,61
Fis	185
G	196
Gis	207,65
A	220
Ais	233,08
H	246,94

Tabelle 4. Die Töne der unteren Oktave 4 mit den zugehörigen Frequenzen

Ton	Frequenz Oktave 5
C	261,63
Cis	277,18
D	293,66
Dis	311,13
E	329,63
F	349,23
Fis	369,99
G	392
Gis	415,3
A	440
Ais	466,16
H	493,88

Tabelle 5. Die Töne der unteren Oktave 5 mit den zugehörigen Frequenzen

gebenden Zeichens und BH die Nummer der Bildschirmseite. Mit CX können Sie zusätzlich bestimmen, wie oft die Funktion das in AL spezifizierte Zeichen hintereinander ausgibt.

Löschen und Vorrücken simulieren Bewegungen

Nach dem Zeichnen des Kastens folgt die Ausgabe des Sonderzeichens. In Listing 3 wurde das ASCII-Zeichen 1 gewählt, das ein fröhliches Gesicht zeigt. Die Bewegung des Gesichts kommt dadurch zustande, daß das Sonderzeichen nach einem gewissen Zeitraum durch Überschreiben mit einem Leerzeichen (ASCII-Wert 32) gelöscht und um eine Position versetzt erneut ausgegeben wird. Für die Wartezeit bei der Ausgabe des animierten Sonderzeichens ist diesmal nicht, wie in Listing 2, eine interne Warteschleife, sondern der Systemuhr-Interrupt 1Ahex zuständig. Über diesen Interrupt läßt sich der Zeitzähler des PCs direkt ansprechen.

● **Der Interrupt 1Ahex:** Während Sie über die Funktion 2Dhex des MS-DOS-Interrupts 21hex die Uhrzeit des PCs in Stunden, Minuten und Sekunden festlegen, bietet der Systemuhr-Interrupt 1Ahex weniger Komfort, denn er setzt und liest lediglich einen 32-Bit-Zähler, aus dem der Programmierer selbst die Zeit berechnen muß. Der Zeitzähler arbeitet mit dem Takt des Ticker-Interrupts im PC, was bedeutet, daß der Wert des Zählers 18,2mal pro Sekunde erhöht wird.

Der Systemuhr-Interrupt ändert den Inhalt der Register BX, SI, DI, BP und der Segmentregister nicht. Er besitzt zwei Funktionen, mit denen Sie den Zeitzähler lesen und setzen können.

Die Funktion 0hex: Diese liest den Zeitzähler und gibt im Register AL die Anzahl der vergangenen 24-Stunden-Intervalle, in CX die obere Hälfte und in DX die untere Hälfte des 32-Bit-Zählers zurück.

Die Funktion 1hex: Diese Funktion setzt den Zeitzähler, indem der Zählerwert über die Register CX und DX übergeben wird. Beide Funktionen benutzt Listing 3, um eine Wartezeit von jeweils zwei Zählerzyklen, das heißt 2/18,2 Sekunden, zu erzielen.

Durch das Löschen des Zeitzählers in Listing 3 wird allerdings auch die Systemuhr des PCs auf 0 Uhr zurückgesetzt. Dieses Problem umgehen Sie, indem Sie nur zu Beginn den Zählerstand auslesen und immer mit dem aktuellen Stand vergleichen, bis der Unterschied 2 (oder einen anderen gewünschten Wert) beträgt.

Sehr interessant wird die Bildschirm- ausgabe übrigens, wenn Sie in Listing 3 die Warteschleife weglassen und den Befehl

```
call warten
```

durch den Leerbefehl

```
nop
```

ersetzen.

Bevor Listing 3 das Sonderzeichen ausgibt, überprüft es, ob sich dessen neue Bildschirmposition innerhalb des Kastens befindet. Ist der rechte Rand des Kastens erreicht, wird die x-Koordinate auf den Wert 1 gesetzt und die y-Koordinate inkrementiert. Erreicht das Gesicht schließlich die letzte Spalte in der letzten Zeile des Kastens, so übernimmt das Programm die Ursprungs- koordinaten als aktuellen Wert.

Die Animation auf der Basis von bildschirmpunktorientierter Grafikausgabe funktioniert auf die gleiche Weise wie das vorher beschriebene Verfahren. Der Unterschied besteht darin, daß die Adressierung der einzelnen Punkte etwas aufwendiger ist. Animierte Figuren erreichen Sie, indem Sie wie oben die Figur an der alten Position löschen und an der neuen wieder ausgeben. Listing 4 zeigt Ihnen am Beispiel einer hellblauen Linie, die sich über den Bildschirm bewegt, wie Sie Grafiken im vierfarbigen 320 x 200-Punkte-Modus der CGA-, EGA- und VGA-Karten animieren.

● **Lösung für saubere Grafikausgaben:** Wenn Sie das Beispielprogramm übersetzt haben, werden Sie feststellen, daß die Linie flackert. Dieses Problem vermeiden Sie mit einem Trick. Die zu animierende Figur wird zuerst an ihrer neuen Position dargestellt und danach von ihrer alten Position gelöscht. Die für dieses Verfahren notwendigen Änderungen sind in Listing 5 in ein Assembler-Programm umgesetzt. Wenn Sie diese Routine starten, bemerken Sie sofort, daß die Animation ruhiger und sauberer wirkt.

Mit der Ausnutzung der Assembler- sprache haben Sie die schwierigste Hürde, die vor schneller und präziser Grafikausgabe steht, über-

wunden. Nun können Sie endlich die Programmideen verwirklichen, die bislang an der mangelnden Geschwindigkeit der Hochsprachen-Interpreter oder -Compiler gescheitert sind.

Mit diesem Beitrag endet unsere dreiteilige Einführung in die Maschinensprache. Wir hoffen, daß Ihnen die Einweisung in die Geheimnisse der Maschinensprache eine Hilfe sein wird. Mit den Grundkenntnissen, die Sie dadurch erworben haben, sind Sie jetzt in der Lage, auch größere Projekte in Assembler zu verwirklichen.

Natürlich haben noch viele Programmierer Vorbehalte, Assembler in umfangreichen Projekten zu benutzen. Doch kann auch heute noch keine Hochsprache mit der hohen Laufgeschwindigkeit und geringen Größe eines fertigen Assembler-Programms mithalten.

● **Assembler in Hochsprachen:** Wem der Aufwand, ein großes Programm in Assembler zu programmieren, zu hoch ist, braucht heutzutage in Hochsprachen nicht auf die Vorteile der Maschinensprache zu verzichten. Nahezu jede wichtige Hochsprache, wie etwa C/C++ (zum Beispiel der Borland C++-Compiler) oder der beliebte Turbo-Pascal-Compiler, kann Assembler verstehen oder assemblierte Programme einbinden. In Turbo Pascal ab Version 6.0 können Sie zum Beispiel mit dem Befehl »asm ... end« Assembler-Routinen in Ihr Pascal-Programm komplett in Assembler schreiben:

```
program PasProg
```

```
...
begin
asm
mov ax,bx
mov bx,ax
end.
end.
```

Dabei können Sie Ihre in Pascal definierten Variablen, Prozeduren und Funktionen auch im Assembler-Code ansprechen und bearbeiten beziehungsweise aufrufen. Natürlich dür-

fen Sie auch Assembler-Prozedur-Sammlungen in das Objektformat assemblieren und linken und anschließend in Ihr Pascal-Programm einbinden.

Sie sehen also, daß Sie nicht unbedingt meterlangen Assembler-Code produzieren müssen. Sie können Ihr Wissen auch in den wichtigen Hochsprachen ausnutzen und erweitern.

● OS/2 und Windows NT

Nicht unerwähnt sollte bleiben, daß auch die neuen Betriebssysteme wie das großartige OS/2 oder das sich noch in der Warteschleife befindende Windows NT nicht ohne Assembler auskommen.

So erklären sich zum Beispiel gerade die Vorteile von OS/2 gegenüber Windows NT in punkto geringerer Arbeitspeicheranforderungen und Ablaufgeschwindigkeit unter anderem durch das Verhältnis von Assembler- zu Hochsprachencode. Während sich Microsoft darauf beschränkt, nur Teile des NT-Kerns in Assembler zu programmieren, hat sich IBM bei OS/2 die Mühe gemacht, größere Teile des gesamten Systems in Maschinensprache zu entwickeln. Dies hat allerdings den Nachteil, daß der Code schwerer auf andere Systeme, wie zum Beispiel RISC-Rechner, portierbar ist. Doch dürfte sich dieses Problem mit einem OS/2 3.0, das sich sicherlich in der Entwicklung befindet, aber noch einige Zeit auf sich warten lassen wird, in Luft auflösen. Sie sehen, selbst große Softwarefirmen kommen auch heute noch nicht ohne Maschinensprache aus. Lange noch wird die Maschinensprache, zumindest in gewissen Modulen, ihren Anteil vor allem an der kommerziellen Programmierung behalten. Darüber hinaus ist der Gedanke, die Hardware völlig zu kontrollieren und zu programmieren, ein Reiz, dem sich ein ernsthafter Programmierer nicht entziehen kann und darf.

(Markus Zietlow/st)

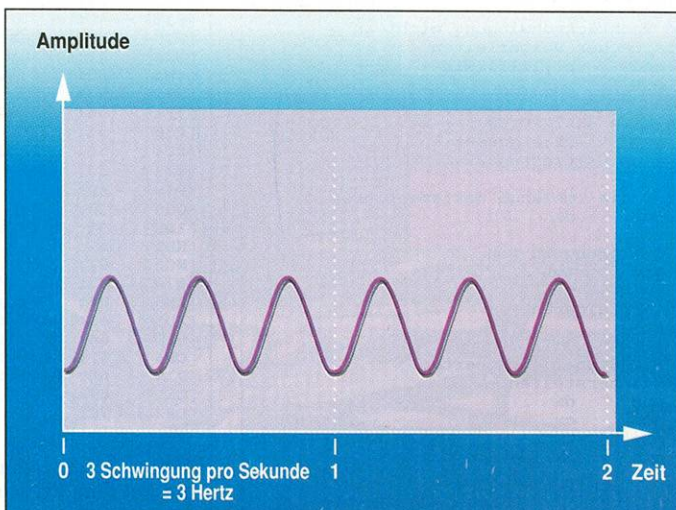


Bild 2. Je mehr Schwingungen pro Sekunde vollzogen werden, um so höher ist die Frequenz.

Ton	Frequenz Oktave 6
C	523,25
Cis	554,37
D	587,33
Dis	622,25
E	659,26
F	698,46
Fis	739,99
G	783,99
Gis	830,61
A	880
Ais	923,33
H	987,77

Tabelle 6. Die Töne der unteren Oktave 6 mit den zugehörigen Frequenzen

```
[9C3] 1: ; Programm: Piepser
[EB2] 2: ; Sprache SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[G66] 6: ;-Assembler-Programm, das einen Pieps
[EY4] 7: ; auf dem Lautsprecher erzeugt
[P30] 8: ;
[P30] 9: ;
[KN2] 10: tonlaenge equ 800
[UK2] 11: frequenz equ 150
[8B2] 12: port equ 61h
[P30] 13: ;
[P30] 14: ;
[NW4] 15: ; Definition des Codesegments
[X54] 16: code segment para 'code'
[P30] 17: ;
[P30] 18: ;
[MC5] 19: ; Definition der Prozedur pieps
[VU2] 20: pieps proc far
[P30] 21: ;
[FA3] 22: ; Segment festlegen
[DK2] 23: assume cs:code
[P30] 24: ;
[9W6] 25: ;-Sichern der wichtigsten Speicherwerte
[XF1] 26: push ax
[ZF1] 27: push bx
[1G1] 28: push cx
[3G1] 29: push dx
[P30] 30: ;
[RL3] 31: ;-Interrupts sperren
[EM0] 32: cli
[P30] 33: ;
[Y15] 34: ;-Länge des Tons in DX ablegen
[153] 35: mov dx,tonlaenge
[P30] 36: ;
[XM6] 37: ;-Status von Ausgabe-Port B holen, keine
[0T5] 38: ; Rechteckschwingung erzeugen und
[D44] 39: ; Lautsprecher aktivieren
[652] 40: in al,port
[4G2] 41: and al,1111110b
[4M3] 42: ton: or al,00000010b
[P30] 43: ;
[YS2] 44: ;-Status setzen
[1E2] 45: out port,al
[P30] 46: ;
[9L4] 47: ;-Warteschleife aktivieren
[B53] 48: mov cx,frequenz
[DY3] 49: hochton: loop hochton
[P30] 50: ;
[2H6] 51: ;-Erzeugte Rechteckschwingung nicht an
[6C5] 52: ; den Lautsprecher weiterleiten
[5G2] 53: and al,11111101b
[1E2] 54: out port,al
[P30] 55: ;
[9L4] 56: ;-Warteschleife aktivieren
[B53] 57: mov cx,frequenz
[104] 58: tiefton: loop tiefton
[P30] 59: ;
[GD7] 60: ;-Zähler für die Tonlänge dekrementieren
[LA1] 61: dec dx
[P30] 62: ;
[K24] 63: ;-Wenn <> 0, Sprung zu ton
[LS1] 64: jnz ton
[P30] 65: ;
[DW3] 66: ;-Interrupts freigeben
[6P0] 67: sti
[P30] 68: ;
[445] 69: ;-Die auf dem Stapel gesicherten
[194] 70: ; Werte wiederherstellen
[XD1] 71: pop dx
[VD1] 72: pop cx
[TD1] 73: pop bx
[RD1] 74: pop ax
```

```
[P30] 75: ;
[2V4] 76: ;-Rückkehr in die MS-DOS-Ebene
[VY1] 77: mov ah,4ch
[3K1] 78: int 21h
[P30] 79: ;
[675] 80: ;-Ende der Prozedur label pieps
[ZZ1] 81: pieps endp
[P30] 82: ;
[P30] 83: ;
[0V3] 84: ;-Ende des Code-Segments
[BW1] 85: code ends
[P30] 86: ;
[P30] 87: ;
[W53] 88: ;-Ende des Programms
[9V1] 89: end pieps
```

Listing 1. Ein schlichter Piepstön wie bei »piepser.asm« lässt sich unproblematisch mit wenigen Anweisungen in Assembler programmieren.

```
[M33] 1: ; Programm: Musika
[EB2] 2: ; Sprache SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[DT6] 6: ;-Assembler-Programm zur Demonstration
[F64] 7: ; der Sound-Eigenschaften
[P30] 8: ;
[P30] 9: ;
[M35] 10: ; Definition des Datensegments
[P44] 11: data segment para 'data'
[P30] 12: ;
[WT4] 13: freq dw 98,110,123,131,147,147
[HU3] 14: dw 147,147,165,131,196,165
[HU3] 15: dw 147,165,131,196,165,147
[3R3] 16: dw 147,131,131,131,131,123
[3P3] 17: dw 123,123,123,110,123,110
[5T3] 18: dw 98,123,147,147,147,131
[KP3] 19: dw 131,131,131,123,123,123
[6L3] 20: dw 123,110,123,110, 98, 0
[P30] 21: ;
[MG5] 22: ;-Für den MASM ersetzen Sie in den
[6V5] 23: ; folgenden 6 Zeilen die Anweisungen
[1X5] 24: ; 'dsw xx,yyy' durch 'dw xx dup (yyy)'
[213] 25: laenge dsw 12,100
[4R1] 26: dsw 1,400
[AR1] 27: dsw 4,100
[4R1] 28: dsw 1,400
[US1] 29: dsw 28,100
[4R1] 30: dsw 1,400
[DU1] 31: data ends
[P30] 32: ;
[P30] 33: ;
[NW4] 34: ; Definition des Codesegments
[X34] 35: code segment para 'code'
[P30] 36: ;
[P30] 37: ;
[VB5] 38: ; Definition der Prozedur musik
[TV2] 39: musik proc far
[P30] 40: ;
[VA3] 41: ; Segmente festlegen
[7Z3] 42: assume cs:code, ds:data
[P30] 43: ;
[9W6] 44: ;-Sichern der wichtigsten Speicherwerte
[XF1] 45: push ax
[ZF1] 46: push bx
[1G1] 47: push cx
[3G1] 48: push dx
[P30] 49: ;
[005] 50: ;-Datensegment initialisieren
[GN6] 51: ;-Für den MASM ersetzen Sie die folgende
[4W4] 52: ; Zeile durch 'mov ax,seg data'
[Z92] 53: mov ax,data
[1V1] 54: mov ds,ax
[P30] 55: ;
[EU4] 56: ;-Tabellenzeiger einrichten
[T72] 57: lea si,freq
[P30] 58: ;
[CU6] 59: ;-Für den MASM ersetzen Sie die folgenden
[1D5] 60: ; 2 Zeilen durch 'lea bp,ds:laenge'
[KB1] 61: seg ds
[NC2] 62: lea bp,laenge
[P30] 63: ;
[KN5] 64: ;-Folgenden Ton aus Tabelle holen
[XU3] 65: spielen: mov di,[si]
[P30] 66: ;
[CP6] 67: ;-Wenn die Melodie beendet ist, Sprung zum
[916] 68: ; Ende, ansonsten Tondauer ermitteln
[GK1] 69: cmp di,0
[D82] 70: je schluss
```

Listing 2. »musika.sam« zeigt, wie wenig Aufwand nötig ist, um mit dem PC Musik zu machen.

Aufgewacht!

**Programmieren ist
Zeit und Wissen**

**Sie haben die Idee.
Wir die Informationen.**

**Der Schlüssel zum Compiler:
Das vierteljährliche Programmier-
Magazin der DOS International
bietet Ihnen qualifizierte und
umfassende Informationen zu
allen Programmierthemen.**

**Und weil wir wissen, daß
Programmierer wenig Zeit haben,
liefern wir sämtliche Programme
und Quelltexte auf Diskette gleich
mit. – Also anfordern.**




```
[P30] 71: ;
[CU6] 72: ; -Für den MASM ersetzen Sie die folgenden
[K45] 73: ; 2 Zeilen durch 'mov bx,ds:[bp]'
[KB1] 74: ;      seg      ds
[YD2] 75: ;      mov      bx,[bp]
[P30] 76: ;
[3U4] 77: ; -Ausgaberegister für Timer-
[9X3] 78: ; Baustein 8253 schreiben
[CW1] 79: ;      mov      al,182
[002] 80: ;      out      43h,al
[P30] 81: ;
[7C5] 82: ; -Für den zweiten Zeitgeber die
[SD5] 83: ; benötigte Frequenz bestimmen
[CZ1] 84: ;      mov      dx,12h
[M92] 85: ;      mov      ax,34dch
[YB1] 86: ;      div      di
[P30] 87: ;
[L37] 88: ; -Das niederwertige und das höherwertige
[LM4] 89: ; Byte des errechneten Wertes
[844] 90: ; an den 8253 übermitteln
[XZ1] 91: ;      out      42h,al
[6T1] 92: ;      mov      al,ah
[XZ1] 93: ;      out      42h,al
[P30] 94: ;
[006] 95: ; -Aktuellen Inhalt des Ausgaberegisters
[3W3] 96: ; des PPI sichern und den
[BF4] 97: ; Lautsprecher einschalten
[2R1] 98: ;      in        al,61h
[2T1] 99: ;      mov      ah,al
[XD1] 100: ;      or        al,3
[YZ1] 101: ;      out      61h,al
[P30] 102: ;
[PX4] 103: ; -Einen Zeitintervall warten
[3B3] 104: ; milli: mov      cx,2808
[ZT3] 105: ; schleif: loop   schleif
[P30] 106: ;
[GE5] 107: ; -Zähler dekrementieren, bis er
[K12] 108: ; gleich 0 ist
[FA1] 109: ;      dec      bx
[9V1] 110: ;      jnz      milli
[P30] 111: ;
[XH7] 112: ; -Ursprüngliche PPI-Werte wiederherstellen
[6T1] 113: ;      mov      al,ah
[YZ1] 114: ;      out      61h,al
[P30] 115: ;
[U17] 116: ; -Zeiger der Ton- und Lautstärken-Tabelle
[7Y5] 117: ; um zwei erhöhen, bis Tabellenende
[D82] 118: ; erreicht ist
[1H1] 119: ;      add      si,2
[MG1] 120: ;      add      bp,2
[4F2] 121: ;      jnz      spielen
[P30] 122: ;
[445] 123: ; -Die auf dem Stapel gesicherten
[194] 124: ; Werte wiederherstellen
[C33] 125: ; schluss: pop      dx
[VD1] 126: ;      pop      cx
[TD1] 127: ;      pop      bx
[RD1] 128: ;      pop      ax
[P30] 129: ;
[2V4] 130: ; -Rückkehr in die MS-DOS-Ebene
[VY1] 131: ;      mov      ah,4ch
[3K1] 132: ;      int      21h
[P30] 133: ;
[R63] 134: ; -Ende der Prozedur
[FZ1] 135: ; musik      endp
[P30] 136: ;
[P30] 137: ;
[0V3] 138: ; -Ende des Code-Segments
[TT1] 139: ; code      ends
[P30] 140: ;
[P30] 141: ;
[W53] 142: ; -Ende des Programms
[8X1] 143: ; end      musik
```

Listing 2. »musika.sam« zeigt, wie wenig Aufwand nötig ist, um mit dem PC Musik zu machen.

```
[FA3] 17: ; Segment festlegen
[XE2] 18: ;      assume cs: code
[P30] 19: ;
[9W6] 20: ; -Sichern der wichtigsten Speicherwerte
[XF1] 21: ;      push      ax
[ZF1] 22: ;      push      bx
[1G1] 23: ;      push      cx
[3G1] 24: ;      push      dx
[P30] 25: ;
[MF5] 26: ; -Definieren des Bildschirmmodus
[TL1] 27: ;      mov      ah,0
[YH1] 28: ;      int      10h
[P30] 29: ;
[TL7] 30: ; -Ermitteln der aktiven Bildschirmseite
[4R1] 31: ;      mov      ah,15
[YH1] 32: ;      int      10h
[P30] 33: ;
[A87] 34: ; -Interruptfunktion 'Positioniere Cursor'
[282] 35: ; ausführen
[FN1] 36: ;      mov      dx,0
[VLI] 37: ;      mov      ah,2
[YH1] 38: ;      int      10h
[P30] 39: ;
[3Y5] 40: ; -19 Zeichen ausgeben auf Position 0,0
[0T1] 41: ;      mov      cx,19
[P30] 42: ;
[ZX5] 43: ; -Das ASCII-Zeichen 205 an der Cursor-
[046] 44: ; position auf dem Bildschirm ausgeben
[UP1] 45: ;      mov      ah,10
[6W1] 46: ;      mov      al,205
[YH1] 47: ;      int      10h
[P30] 48: ;
[FL4] 49: ; -Einmal das ASCII-Zeichen 201
[YM4] 50: ; auf dem Bildschirm ausgeben
[EN1] 51: ;      mov      cx,1
[UV1] 52: ;      mov      al,201
[UP1] 53: ;      mov      ah,10
[YH1] 54: ;      int      10h
[P30] 55: ;
[ZV6] 56: ; -Spalte auf die rechte, obere Ecke setzen
[AR6] 57: ; und entsprechendes Zeichen darstellen
[XLI] 58: ;      add      dl,19
[VLI] 59: ;      mov      ah,2
[YH1] 60: ;      int      10h
[EN1] 61: ;      mov      cx,1
[VW1] 62: ;      mov      al,187
[UP1] 63: ;      mov      ah,10
[YH1] 64: ;      int      10h
[P30] 65: ;
[YS6] 66: ; -Cursor auf untere, rechte Ecke setzen
[7G1] 67: ;      add      dh,9
[VLI] 68: ;      mov      ah,2
[YH1] 69: ;      int      10h
[P30] 70: ;
[3V3] 71: ; -ASCII-Zeichen 188 als
[424] 72: ; Kennzeichen markieren
[YW1] 73: ;      mov      al,188
[UP1] 74: ;      mov      ah,10
[YH1] 75: ;      int      10h
[P30] 76: ;
[ED5] 77: ; -Die untere, linke Ecke zeichnen
[CP1] 78: ;      sub      dl,19
[VLI] 79: ;      mov      ah,2
[YH1] 80: ;      int      10h
[P30] 81: ;
[Z24] 82: ; -Wiederum 19mal das Zeichen 205
[WY3] 83: ; auf den Monitor bringen
[0T1] 84: ;      mov      cx,19
[6W1] 85: ;      mov      al,205
[UP1] 86: ;      mov      ah,10
[YH1] 87: ;      int      10h
[P30] 88: ;
[K66] 89: ; -Einmal das Zeichen 200 positionieren
[EN1] 90: ;      mov      cx,1
[RV1] 91: ;      mov      al,200
[UP1] 92: ;      mov      ah,10
[YH1] 93: ;      int      10h
[P30] 94: ;
[TL5] 95: ; -Jedes Zeichen einmal darstellen
[EN1] 96: ;      mov      cx,1
[P30] 97: ;
[Y15] 98: ; -Zeile um eins vermindern und
[LV3] 99: ; Cursor positionieren
[ST2] 100: ; seiten: dec      dh
[VLI] 101: ;      mov      ah,2
[YH1] 102: ;      int      10h
[P30] 103: ;
[HX3] 104: ; -Zeichen für eine Seite
[SW1] 105: ;      mov      al,186
[P30] 106: ;
[W24] 107: ; -Danach Zeichen ausgeben
[UP1] 108: ;      mov      ah,10
[YH1] 109: ;      int      10h
[P30] 110: ;
[L36] 111: ; -Spalte auf die rechte Seite setzen
[XLI] 112: ;      add      dl,19
[VLI] 113: ;      mov      ah,2
```

```
[KV2] 1: ; Programm: Pieps
[EB2] 2: ; Sprache SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[DT6] 6: ; -Assembler-Programm zur Demonstration
[V65] 7: ; der Blockgrafik-Eigenschaften
[P30] 8: ;
[P30] 9: ;
[NW4] 10: ; Definition des Codesegments
[X34] 11: ; code      segment para 'code'
[P30] 12: ;
[P30] 13: ;
[3G5] 14: ; Defintion der Prozedur kasten
[2Y2] 15: ; kasten  proc  far
[P30] 16: ;
```



```
[YH1] 114:      int    10h
[P30] 115: ;
[HX3] 116: ; -Zeichen für eine Seite
[SW1] 117:      mov    al,186
[P30] 118: ;
[CL4] 119: ; -Seitenzeichen darstellen
[UP1] 120:      mov    ah,10
[YH1] 121:      int    10h
[P30] 122: ;
[006] 123: ; -Spalte auf ursprünglichen Wert
[CP1] 124:      sub     dl,19
[P30] 125: ;
[1X3] 126: ; -Erste Zeile erreicht ?
[EK1] 127:      cmp     dh,1
[P30] 128: ;
[U25] 129: ; -Wenn nicht, Sprung zu seiten
[L22] 130:      jne     seiten
[P30] 131: ;
[7N2] 132: ; -Koordinaten 1,1
[362] 133:      mov     dx,101h
[P30] 134: ;
[NK5] 135: ; -Neue Cursorposition einrichten
[F03] 136: animat: mov     ah,2
[YH1] 137:      int    10h
[P30] 138: ;
[FX5] 139: ; -Gesicht an Cursorposition zeigen
[6M1] 140:      mov     al,1
[UP1] 141:      mov     ah,10
[YH1] 142:      int    10h
[P30] 143: ;
[155] 144: ; -Aufruf der Zeitverzögerung
[642] 145:      call    warten
[P30] 146: ;
[KP4] 147: ; -Wenn Taste gedrückt, dann
[AW4] 148: ; Sprung zum Ende der Routine
[WP1] 149:      mov     ah,11
[3K1] 150:      int    21h
[LK1] 151:      cmp     al,0
[GG2] 152:      jnz     schluss
[P30] 153: ;
[B43] 154: ; -Ausgabe anwählen
[UP1] 155:      mov     ah,10
[P30] 156: ;
[R94] 157: ; -Gesicht wieder löschen
[CR1] 158:      mov     al,32
[YH1] 159:      int    10h
[P30] 160: ;
[ZW2] 161: ; -Spalte erhöhen
[MA1] 162:      inc     dl
[P30] 163: ;
[377] 164: ; -Wenn rechter Rand nicht erreicht, Routine
[435] 165: ; wiederholen, sonst Spalte = 1
[CP1] 166:      cmp     dl,19
[D22] 167:      jne     animat
[CM1] 168:      mov     dl,1
[P30] 169: ;
[4T2] 170: ; -Zeile erhöhen
[8A1] 171:      inc     dh
[P30] 172: ;
[7W5] 173: ; -Wenn untere Zeile nicht erreicht,
[XP4] 174: ; Routine wiederholen, sonst
[TK4] 175: ; Zeile = 1 und weitermachen
[PK1] 176:      cmp     dh,9
[D22] 177:      jne     animat
[0M1] 178:      mov     dh,1
[P30] 179: ;
[W66] 180: ; -Für den MASM müssen Sie die folgende
[ER5] 181: ; Zeile durch 'jmp animat' ersetzen
[952] 182:      jmps    animat
[P30] 183: ;
[P30] 184: ;
[TL6] 185: ; -Anfang der Zeitverzögerungsroutine
[H43] 186:      proc    near
[P30] 187: ;
[NN4] 188: ; -Die anderweitig benutzten
[D03] 189: ; Register sichern
[1G1] 190:      push    cx
[3G1] 191:      push    dx
[P30] 192: ;
[9L3] 193: ; -CX-Register löschen
[PV1] 194:      xor     cx,cx
[P30] 195: ;
[DL3] 196: ; -DX-Register löschen
[TV1] 197:      xor     dx,dx
[P30] 198: ;
[8X4] 199: ; -Systemzeit auf Null setzen
[UL1] 200:      mov     ah,1
[HL1] 201:      int    1ah
[P30] 202: ;
[B26] 203: ; -Funktion Null liest die Systemuhr
[0T1] 204:      xor     ah,ah
[P30] 205: ;
[VA2] 206: ; -Uhr auslesen
[SY2] 207:      zehntel: int    1ah
[P30] 208: ;
[WR3] 209: ; -2/18,2 Sekunden warten
[VK1] 210:      cmp     dl,2
[AC2] 211:      jne     zehntel
```

```
[P30] 212: ;
[757] 213: ; -Gesicherte Register wieder zurückholen
[XD1] 214:      pop     dx
[VD1] 215:      pop     cx
[P30] 216: ;
[TV4] 217: ; -Rückkehr ins Hauptprogramm
[3P0] 218:      ret
[P30] 219: ;
[R63] 220: ; -Ende der Prozedur
[C22] 221:      wait    endp
[P30] 222: ;
[P30] 223: ;
[5T6] 224: ; -Cursor unter dem Kasten positionieren
[PA3] 225:      schluss: mov     dh,15
[VL1] 226:      mov     ah,2
[YH1] 227:      int    10h
[P30] 228: ;
[445] 229: ; -Die auf dem Stapel gesicherten
[194] 230: ; Werte wiederherstellen
[XD1] 231:      pop     dx
[VD1] 232:      pop     cx
[TD1] 233:      pop     bx
[RD1] 234:      pop     ax
[P30] 235: ;
[V24] 236: ; -Rückkehr in die MS-DOS-Ebene
[VY1] 237:      mov     ah,4ch
[3K1] 238:      int    21h
[P30] 239: ;
[R63] 240: ; -Ende der Prozedur
[S12] 241:      kasten  endp
[P30] 242: ;
[P30] 243: ;
[0V3] 244: ; -Ende des Code-Segments
[TT1] 245:      code    ends
[P30] 246: ;
[P30] 247: ;
[W53] 248: ; -Ende des Programms
[H22] 249:      end     kasten
```

Listing 3. Für einfache Animation und schnellen Bildaufbau eignen sich auch die Sonderzeichen des PC-Zeichensatzes.

```
[1G3] 1: ; Programm: Linemove
[EB2] 2: ; Sprache SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[GR6] 6: ; -Beispielprogramm, das eine Linie über
[BW3] 7: ; den Bildschirm bewegt
[P30] 8: ;
[P30] 9: ;
[K85] 10: ; -Byte-Offset der Koordinate (0;0)
[XC2] 11: voffset equ 0
[P30] 12: ;
[UD5] 13: ; -Anfang des Bildschirmspeichers
[063] 14: videosg equ 0b800h
[P30] 15: ;
[D75] 16: ; -Adresse des Video-Interrupts
[6A2] 17: video equ 10h
[P30] 18: ;
[P30] 19: ;
[CT2] 20: ; -Programmstart
[FR4] 21: ; Definition des Codesegment
[X34] 22: code segment para 'code'
[P30] 23: ;
[BN5] 24: ; -Startadresse für eine COM-Datei
[CE1] 25:      org 0100h
[P30] 26: ;
[VA3] 27: ; Segmente festlegen
[D95] 28:      assume cs:code, ds:code, es:code
[P30] 29: ;
[P30] 30: ;
[MS5] 31: ; Definition der Prozedur beispiel
[723] 32: beispiel proc near
[P30] 33: ;
[N24] 34: ; -Bildschirmmodus lesen
[Y43] 35:      call lsmode
[P30] 36: ;
[0S2] 37: ; -Modus sichern
[PC1] 38:      push    ax
[P30] 39: ;
[LF5] 40: ; -Aktivieren des Grafikmodus mit
[XL3] 41: ; dem Video-Interrupt
[EP1] 42:      mov     ax,4h
[H02] 43:      call    srmode
[P30] 44: ;
[7H4] 45: ; -Hellblaue Linie über den
[CA3] 46: ; Bildschirm bewegen
[R36] 47: ; -Alle angegebenen Register löschen
[YS1] 48:      xor     ax,ax
```

Listing 4. »linemove.sam« zeigt, daß die Grafikanimation wesentlich aufwendiger ist als die Animation über Sonderzeichen.


```
[3T1] 49:      xor  bx,bx
[8T1] 50:      xor  cx,cx
[P30] 51:      ;
[F24] 52:      ;-X-Koordinate erhöhen
[HE2] 53:      start1: inc  bx
[P30] 54:      ;
[T04] 55:      ;-Farbattribut hellblau
[7H1] 56:      mov  ch, 1
[P30] 57:      ;
[0C2] 58:      ;-Linie ziehen
[WN1] 59:      call linie
[P30] 60:      ;
[2U3] 61:      ;-Farbattribut schwarz
[4H1] 62:      mov  ch, 0
[P30] 63:      ;
[0C2] 64:      ;-Linie ziehen
[WN1] 65:      call linie
[P30] 66:      ;
[5X4] 67:      ;-Wenn rechter Rand noch nicht
[K95] 68:      ; erreicht ist, Sprung zu 'start1'
[WT1] 69:      cmp  bx,319
[M12] 70:      jnz  start1
[P30] 71:      ;
[144] 72:      ;-Auf Tastendruck warten
[FP1] 73:      call taste
[P30] 74:      ;
[XV4] 75:      ;-Original-Modus wieder lesen
[GA1] 76:      pop  ax
[P30] 77:      ;
[W64] 78:      ;-Bildschirmmodus setzen
[H02] 79:      call srmode
[P30] 80:      ;
[L66] 81:      ;-Programm über Interrupt 21h beenden
[ZW1] 82:      mov  ah,4ch
[E61] 83:      int  21h
[P30] 84:      ;
[TU4] 85:      ;-X- und Y-Koordinate sichern
[FF2] 86:      push ax
[TC1] 87:      push bx
[HG1] 88:      schleife:
[P30] 89:      ;
[1P4] 90:      ;-AX- und BX-Register sichern
[PC1] 91:      push ax
[TC1] 92:      push bx
[P30] 93:      ;
[AV5] 94:      ;-Bitmaske und Byteadresse berechnen
[702] 95:      call pixadr
[P30] 96:      ;
[8S3] 97:      ;-Farbnummer festlegen
[7W1] 98:      mov  al, ch
[P30] 99:      ;
[B34] 100:     ;-Bitmaske aktualisieren
[PW1] 101:     shl  ax, cl
[P30] 102:     ;
[3P3] 103:     ;-Bitmaske invertieren
[CD2] 104:     Pixel: not  ah
[P30] 105:     ;
[XN3] 106:     ;-Bildpunkt löschen
[WU5] 107:     ;-Für den MASM müssen Sie die beiden
[SA6] 108:     ; folgenden Zeilen durch die Anweisung
[494] 109:     ; 'and es:[bx],ah' ersetzen
[C81] 110:     seg  es
[RY1] 111:     and  [bx],ah
[P30] 112:     ;
[XK3] 113:     ;-Bildpunkt ersetzen
[WU5] 114:     ;-Für den MASM müssen Sie die beiden
[SA6] 115:     ; folgenden Zeilen durch die Anweisung
[624] 116:     ; 'or es:[bx],al' ersetzen
[C81] 117:     seg  es
[LV1] 118:     or   [bx],al
[P30] 119:     ;
[6H6] 120:     ;-X- und Y-Koordinate wiederherstellen
[TH2] 121:     weiter: pop  bx
[GA1] 122:     pop  ax
[P30] 123:     ;
[NL4] 124:     ;-Y-Koordinate um 1 erhöhen
[B81] 125:     inc  ax
[P30] 126:     ;
[KB6] 127:     ;-Vergleich mit 199; wenn unterer Rand
[4D6] 128:     ; nicht erreicht, Sprung zu 'schleife'
[DU1] 129:     cmp  ax,199
[E72] 130:     jnz  schleife
[P30] 131:     ;
[L15] 132:     ;-Ursprüngliche Koordinaten
[S43] 133:     ; wiederherstellen
[LA1] 134:     pop  bx
[GA1] 135:     pop  ax
[P30] 136:     ;
[TV4] 137:     ;-Rückkehr ins Hauptprogramm
[3P0] 138:     ret
[P30] 139:     ;
[N24] 140:     ;-Bildschirmmodus lesen
[RB6] 141:     ;-Den aktuellen Video-Modus ermitteln
[653] 142:     lsmode: mov  ah,0fh
[0Y1] 143:     int  video
[3P0] 144:     ret
[P30] 145:     ;
[W64] 146:     ;-Bildschirmmodus setzen
[P34] 147:     ;-Den in AL gespeicherten
[Y34] 148:     ; Video-Modus aktivieren
```

```
[T23] 149: srmode: mov  ah,00h
[0Y1] 150:      int  video
[3P0] 151:      ret
[P30] 152:      ;
[5K6] 153:      ;-Bytenummer und Bitmaske eines Pixels
[TK6] 154:      ; berechnen; das bedeutet, Pufferadresse
[YR5] 155:      ; eines Bildpunkts im 320x200-Modus
[TU3] 156:      ; mit 4 Farben bestimmen
[553] 157:      pixadr: mov  cl, bl
[P30] 158:      ;
[S45] 159:      ;-AX=256*y und durch zwei geteilt
[RS1] 160:      xchg  ah,al
[DE1] 161:      shr  ax,1
[TN1] 162:      add  bh,al
[P30] 163:      ;
[872] 164:      ;-AL löschen
[0U1] 165:      xor  al,al
[P30] 166:      ;
[NE1] 167:      ;-BX=AX+x
[TR1] 168:      add  bx,ax
[P30] 169:      ;
[Y54] 170:      ;-AX erhält den Wert 64*y
[DE1] 171:      shr  ax,1
[DE1] 172:      shr  ax,1
[P30] 173:      ;
[BP1] 174:      ;-BX=320*y+x
[TR1] 175:      add  bx,ax
[P30] 176:      ;
[UA6] 177:      ;-BX erhält den Wert X geteilt durch 8
[HE1] 178:      shr  bx,1
[HE1] 179:      shr  bx,1
[P30] 180:      ;
[S16] 181:      ;-BX=Anfangsadresse im Video-Puffer
[7K2] 182:      add  bx,voffset
[P30] 183:      ;
[RM5] 184:      ;-ES enthält Segmentadresse des
[0S2] 185:      ; Video-Puffers
[7P2] 186:      mov  ax,videosg
[1W1] 187:      mov  es,ax
[FL1] 188:      mov  ah, 3
[FP1] 189:      and  cl,ah
[ST1] 190:      xor  cl,ah
[2K1] 191:      shl  cl, 1
[3P0] 192:      ret
[P30] 193:      ;
[Y35] 194:      ;-Warten auf einen Tastendruck
[Y95] 195:      ;-Ein Zeichen von Tastatur lesen
[GL2] 196:      taste: mov  ah,0
[Z61] 197:      int  16h
[3P0] 198:      ret
[P30] 199:      ;
[R63] 200:      ;-Ende der Prozedur
[G62] 201:      beispiel endp
[P30] 202:      ;
[P30] 203:      ;
[W53] 204:      ;-Ende des Programms
[TT1] 205:      code   ends
[P30] 206:      ;
[P30] 207:      ;
[HL3] 208:      ;-Ende des Quelltextes
[4K1] 209:      end  start
```

Listing 4. »linemove.sam« zeigt, daß die Grafikanimation wesentlich aufwendiger ist als die Animation über Sonderzeichen.

```
[5L3] 1: ; Programm: Linemove2
[EB2] 2: ; Sprache SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[GR6] 6: ;-Beispielprogramm, das eine Linie über
[8R5] 7: ; den Bildschirm bewegt (Version 2)
[P30] 8: ;
[P30] 9: ;
[K85] 10: ;-Byte-Offset der Koordinate (0;0)
[XC2] 11: voffset equ 0
[P30] 12: ;
[UD5] 13: ;-Anfang des Bildschirmspeichers
[063] 14: videosg equ 0b800h
[P30] 15: ;
[D75] 16: ;-Adresse des Video-Interrupts
[6A2] 17: video equ 10h
[P30] 18: ;
[P30] 19: ;
[CT2] 20: ;-Programmstart
[FR4] 21: ; Definition des Codesegment
[X34] 22: code segment para 'code'
[P30] 23: ;
[BN5] 24: ;-Startadresse für eine COM-Datei
[CE1] 25: org 0100h
[P30] 26: ;
[VA3] 27: ; Segmente festlegen
```



```
[D95] 28: assume cs:code, ds:code, es:code
[P30] 29: ;
[P30] 30: ;
[MS5] 31: ; Definition der Prozedur beispiel
[723] 32: beispiel proc near
[P30] 33: ;
[N24] 34: ;-Bildschirmmodus lesen
[Y43] 35: start: call lsmode
[P30] 36: ;
[052] 37: ;-Modus sichern
[PC1] 38: push ax
[P30] 39: ;
[LF5] 40: ;-Aktivieren des Grafikmodus mit
[XL3] 41: ; dem Video-Interrupt
[EP1] 42: mov ax,4h
[H02] 43: call srmode
[P30] 44: ;
[7H4] 45: ;-Hellblaue Linie über den
[CA3] 46: ; Bildschirm bewegen
[R36] 47: ;-Alle angegebenen Register löschen
[YS1] 48: xor ax,ax
[3T1] 49: xor bx,bx
[8T1] 50: xor cx,cx
[P30] 51: ;
[P30] 52: ;
[F24] 53: ;-X-Koordinate erhöhen
[HE2] 54: start1: inc bx
[P30] 55: ;
[T04] 56: ;-Farbattribut hellblau
[7H1] 57: mov ch, 1
[P30] 58: ;
[0C2] 59: ;-Linie ziehen
[WN1] 60: call linie
[P30] 61: ;
[B94] 62: ;-X-Koordinate vermindern
[971] 63: dec bx
[P30] 64: ;
[2U3] 65: ;-Farbattribut schwarz
[4H1] 66: mov ch, 0
[P30] 67: ;
[0C2] 68: ;-Linie ziehen
[WN1] 69: call linie
[P30] 70: ;
[A15] 71: ;-Ursprüngliche X-Koordinate
[S43] 72: ; wiederherstellen
[E81] 73: inc bx
[P30] 74: ;
[VM6] 75: ;-Wenn rechter Rand noch nicht erreicht,
[943] 76: ; Sprung zu start1
[WT1] 77: cmp bx,319
[M12] 78: jnz start1
[P30] 79: ;
[144] 80: ;-Auf Tastendruck warten
[FP1] 81: call taste
[P30] 82: ;
[XY4] 83: ;-Original-Modus wieder lesen
[GA1] 84: pop ax
[P30] 85: ;
[W64] 86: ;-Bildschirmmodus setzen
[H02] 87: call srmode
[P30] 88: ;
[L66] 89: ;-Programm über Interrupt 21h beenden
[ZW1] 90: mov ah,4ch
[E61] 91: int 21h
[P30] 92: ;
[TU4] 93: ;-X- und Y-Koordinate sichern
[FF2] 94: linie: push ax
[TC1] 95: push bx
[HG1] 96: schleife:
[P30] 97: ;
[1P4] 98: ;-AX- und BX-Register sichern
[PC1] 99: push ax
[TC1] 100: push bx
[P30] 101: ;
[AV5] 102: ;-Bitmaske und Byteadresse berechnen
[702] 103: call pixadr
[P30] 104: ;
[8S3] 105: ;-Farbnummer festlegen
[7W1] 106: mov al, ch
[P30] 107: ;
[B34] 108: ;-Bitmaske aktualisieren
[PW1] 109: shl ax, cl
[P30] 110: ;
[3P3] 111: ;-Bitmaske invertieren
[CD2] 112: Pixel: not ah
[P30] 113: ;
[XN3] 114: ;-Bildpunkt löschen
[WU5] 115: ;-Für den MASM müssen Sie die beiden
[SA6] 116: ; folgenden Zeilen durch die Anweisung
[494] 117: ; 'and es:[bx],ah' ersetzen
[C81] 118: seg es
[RY1] 119: and [bx],ah
[P30] 120: ;
[XK3] 121: ;-Bildpunkt ersetzen
[WU5] 122: ;-Für den MASM müssen Sie die beiden
[SA6] 123: ; folgenden Zeilen durch die Anweisung
[624] 124: ; 'or es:[bx],al' ersetzen
```

```
[C81] 125: seg es
[LV1] 126: or [bx],al
[P30] 127: ;
[6H6] 128: ;-X- und Y-Koordinate wiederherstellen
[TH2] 129: weiter: pop bx
[GA1] 130: pop ax
[P30] 131: ;
[NL4] 132: ;-Y-Koordinate um 1 erhöhen
[B81] 133: inc ax
[P30] 134: ;
[KB6] 135: ;-Vergleich mit 199; wenn unterer Rand
[4D6] 136: ; nicht erreicht, Sprung zu 'schleife'
[DU1] 137: cmp ax,199
[E72] 138: jnz schleife
[P30] 139: ;
[LI5] 140: ;-Ursprüngliche Koordinaten
[S43] 141: ; wiederherstellen
[LA1] 142: pop bx
[GA1] 143: pop ax
[P30] 144: ;
[TV4] 145: ;-Rückkehr ins Hauptprogramm
[3P0] 146: ret
[P30] 147: ;
[N24] 148: ;-Bildschirmmodus lesen
[RB6] 149: ;-Den aktuellen Video-Modus ermitteln
[653] 150: lsmode: mov ah,0fh
[0V1] 151: int video
[3P0] 152: ret
[P30] 153: ;
[W64] 154: ;-Bildschirmmodus setzen
[P34] 155: ;-Den in AL gespeicherten
[Y34] 156: ; Video-Modus aktivieren
[T23] 157: srmode: mov ah,0fh
[0V1] 158: int video
[3P0] 159: ret
[P30] 160: ;
[5K6] 161: ;-Bytenummer und Bitmaske eines Pixels
[TK6] 162: ; berechnen; das bedeutet, Pufferadresse
[YR5] 163: ; eines Bildpunkts im 320x200-Modus
[TU3] 164: ; mit 4 Farben bestimmen
[553] 165: pixadr: mov cl, bl
[P30] 166: ;
[S45] 167: ;-AX=256*y und durch zwei geteilt
[RS1] 168: xchg ah,al
[DE1] 169: shr ax,1
[TN1] 170: add bh,al
[P30] 171: ;
[872] 172: ;-AL löschen
[0U1] 173: xor al,al
[P30] 174: ;
[NE1] 175: ;-BX=AX+x
[TR1] 176: add bx,ax
[P30] 177: ;
[Y54] 178: ;-AX erhält den Wert 64*y
[DE1] 179: shr ax,1
[DE1] 180: shr ax,1
[P30] 181: ;
[BP1] 182: ;-BX=320*y+x
[TR1] 183: add bx,ax
[P30] 184: ;
[UA6] 185: ;-BX erhält den Wert X geteilt durch 8
[HE1] 186: shr bx,1
[HE1] 187: shr bx,1
[P30] 188: ;
[S16] 189: ;-BX=Anfangsadresse im Video-Puffer
[7K2] 190: add bx,voffset
[P30] 191: ;
[RMS] 192: ;-ES enthält Segmentadresse des
[0S2] 193: ; Video-Puffers
[7P2] 194: mov ax,videosg
[1W1] 195: mov es,ax
[FL1] 196: mov ah, 3
[FP1] 197: and cl,ah
[ST1] 198: xor cl,ah
[2K1] 199: shl cl, 1
[3P0] 200: ret
[P30] 201: ;
[Y35] 202: ;-Warten auf einen Tastendruck
[Y95] 203: ;-Ein Zeichen von Tastatur lesen
[GL2] 204: taste: mov ah,0
[Z61] 205: int 16h
[3P0] 206: ret
[P30] 207: ;
[R63] 208: ;-Ende der Prozedur
[G62] 209: beispiel endp
[P30] 210: ;
[P30] 211: ;
[W53] 212: ;-Ende des Programms
[TT1] 213: code ends
[P30] 214: ;
[P30] 215: ;
[HL3] 216: ;-Ende des Quelltextes
[4K1] 217: end start
```

Listing 5. Wenn Sie die vorangegangene Linie erst löschen, wenn die neue schon auf dem Bildschirm erscheint, wirkt die animierte Grafik wesentlich ruhiger.

Assembler zum Abtippen

Prozessor-Tuning mit SAM 4.0

Programmieren in Maschinensprache macht nur Spaß, wenn man mit den entsprechenden Werkzeugen arbeiten kann. Makroprogrammierung, eine leicht zu bedienende Benutzeroberfläche, ansprechende Übersetzungszeiten und komfortable Zusatzbefehle gehören zu den meistgenannten Wünschen der Assembler-Programmierer.

Der DOS-Assembler SAM 4.0 erfüllt diese Anforderungen. Mit ihm steht Ihnen eine preiswerte Alternative zum TASM und MASM zur Verfügung, die natürlich auch alle im vorangegangenen Assembler-Kurs abgedruckten Beispiele übersetzt.

In diesem Beitrag erwartet Sie der Makro-Assembler SAM 4.0, der eine neue Version des beliebten Assemblers SAM aus den bisherigen Ausgaben der DOS Extra und DOS International ist. SAM 4.0 ist als Basic-Listing im Quellcode abgedruckt und läßt sich mit QBasic, Quick-Basic oder Basic-PDS kompilieren.

Die Sprache Basic wurde auch dieses Mal gewählt, weil dadurch jeder Leser, der über MS-DOS 5.0 verfügt – dies darf man sicherlich als Grundvoraussetzung annehmen – diesen Assembler der DOS International starten und anwenden kann; denn bei MS-DOS wird ein Interpreter mitgeliefert, der den SAM 4.0 verarbeiten kann: QBasic.

Mit dieser Interpreter-Version von Quick-Basic im Verbund kann SAM 4.0 aus einem Assembler-Listing ein lauffähiges Programm vom Typ Com oder Exe anfertigen.

Für PC-Besitzer ist die Auswahl an Assemblern immer noch sehr spärlich. Am bekanntesten sind der Turbo Assembler von Borland und der Makro-Assembler MASM von Microsoft. Eine preiswerte Alternative stellt der Makro-Assembler SAM 4.0 (Listing 1) dar, der sowohl Com- als auch Exe-Dateien erzeugt und zusätzlich Include-Dateien und Makros verarbeitet.

Um einen Nachteil von SAM 3.0 auszugleichen, wurden die vorher in eine eigene Datei ausgelagerten Daten zum Übersetzen der Assembler-Programme bei SAM 4.0 in den Quelltext integriert.

Wer sich im übrigen für die geschichtliche Entwicklung des Assemblers SAM interessiert, wird in der Tabelle 1 ausführlich informiert.

● **Freie Bahn für SAM 4.0:** SAM 4.0 läßt sich wahlweise mit Benutzeroberfläche (Bild 1 und 2) oder, wenn er mit Quick-Basic oder Basic-PDS eingesetzt wird, kommandozeilenorientiert aufrufen. Im ersten Fall geben Sie in der MS-DOS-Ebene nur

qbasic sam4

ein. Nach dem Start erscheinen Titeldialog und Menü von SAM 4.0 (Bild 3). Tabelle 2 stellt die Bedienungselemente von SAM 4.0 in der Übersicht dar.

Wenn Sie jedoch mit Quick Basic oder Basic-PDS arbeiten, die Variable »command\$« durch »command\$« ersetzen und den Assembler kompilieren, erfolgt der Aufruf durch

sam4

● **Kommandozeilenversion von SAM 4.0:** Möchten Sie dagegen kommandozeilenorientiert arbeiten, so rufen Sie die kompilierte Version von SAM 4.0 auf mit

sam4 <name>

Die Entwicklung des DOS-Assemblers SAM

Version	Bemerkung
SAM 1.0	Die erste GW-Basic-Version von SAM überrascht durch ihre Geschwindigkeit und eingeschränkte Kompatibilität zu MASM.
SAM 2.0	SAM 2.0 weist eine neue Benutzeroberfläche auf. Darüber hinaus verarbeitet er Makros und Include-Dateien. Neben Com- lassen sich auch Exe-Dateien erzeugen.
SAM 3.0	Bei SAM 3.0 sind kleinere Fehler beseitigt. Außerdem arbeitet er mit einer höheren Geschwindigkeit als seine Vorläufer.
SAM 3.01	Die Adressenumrechnung von nachfolgenden Sprungsymbolen bei indexadressierten Befehlen (beispielsweise »mov -12[bp],ax«) funktioniert auch, wenn der Index-Offset zwischen -128 und -1 oder +128 und +255 liegt. Die Multiplikations- beziehungsweise Divisionsbefehle »mul«, »div«, »imul« und »idiv« werden fehlerfrei assembliert, wenn 8-Bit-Operationen (zum Beispiel »mul ax,dx«) durchgeführt werden sollen. Der Pseudobefehl »even« paßt bei ungerader Adresse den Codezeiger richtig an und fügt den erforderlichen Nop-Befehl ein. SAM 3.01 akzeptiert die Pseudobefehle »segment para 'code'« und »segment para 'stack'«, wenn in der gleichen Zeile ein Kommentar folgt. Quelltexte werden auch dann lauffähig assembliert, wenn das Stapelsegment vor dem Codesegment steht. Der Pseudobefehl »ds« reserviert den angeforderten Speicherbereich an der richtigen Stelle und verkürzt den zu generierenden Code nicht mehr um die Anzahl der reservierten Bytes. SAM 3.01 akzeptiert die Mnemonics »ja«, »jae«, »jb«, »jbe«, »jc«, »jnc«, »jg«, »jge«, »jl«, »jle«, »je«, »jne«, »jp«, »jnp«, »shl«, »shlb« und »shlw«.
SAM 4.0	SAM 4.0 fällt durch eine neu konzipierte Bedienungsführung auf und ist wesentlich schneller als seine Vorläufer. Bei Include- und Makrobefehlen bleiben die Assembler-Läufe auf 2 beschränkt. Die Kommandos »mul«, »imul«, »div« und »idiv« funktionieren einwandfrei. Makros können Leerzeichen enthalten. Verschachtelte Include-Anweisungen sind erlaubt. Die Operandenauswertung ist verbessert und beschleunigt. SAM 4.0 erlaubt den Einsatz von Oktalzahlen (beispielsweise »170«). Die die Geschwindigkeit verringernden Sprungmarken sind wesentlich reduziert. Einfacher zu handhabende Do-Loop-Schleifen haben die While-Wend-Schleifen ersetzt. Der Pseudobefehl »ds« wurde um die Varianten »dsb« und »dsw« erweitert. Die bisher ignorierten Einträge in der Relozierungstabelle sind eingefügt. Druckparameter lassen sich einstellen. SAM 4.0 ergänzt Quelltext- und Zieldateinamen automatisch.

Tabelle 1. Dieser Tabelle entnehmen Sie die geschichtliche Entwicklung des Assemblers der DOS International.

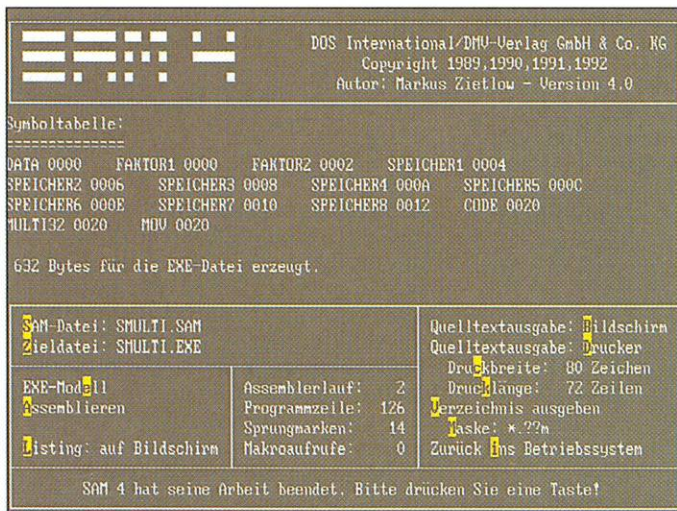


Bild 1. SAM 4.0 erzeugt Exe-Dateien, ohne daß Sie weitere Hilfsprogramme, wie etwa TLink, benötigen

In diesem Fall assembliert SAM 4.0 ohne weitere Eingabe selbstständig die Datei »name.sam« zu »name.com«.

Die Assembler-Programme für SAM 4.0 lassen sich mit jeder Textverarbeitung schreiben, die ASCII-Dateien erzeugt (zum Beispiel MS-Word, PC-Write oder mit der Benutzeroberfläche von Turbo Pascal). Im Zweifelsfall tut es auch der MS-DOS-Zeileneditor »edlin.com« beziehungsweise »edit.com«. Besitzen Sie schon die neue Benutzeroberfläche von Borland Pascal 7.0, dann erzeugen Sie damit das Programm und starten dann SAM 4.0 von dort aus über das Menü »Tools«.

Marken und Adressen mit dem SAM-Parser

Weil der Parser (der Programmteil, der das Assembler-Programm in Marken, Befehle, Operanden und Kommentare zerlegt) einfach und 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 Leerzeichen stehen, weil der Parser sonst die Marke aufgrund des Leerzeichens nicht erkennt. Beim Zerlegen der Programmzeile würde der Rest einfach abgeschnitten.

Ein Beispiel:

```
mov ax,offset(text)
```

● **Displacement unter SAM 4:** Aus dem gleichen Grund muß die Verschiebung (»Displacement«) bei der indirekten Adressierung ohne Pluszeichen direkt vor die eckige Klammer, die die Quell- oder Zieladresse enthält, geschrieben werden. Einige 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.

● **Kompatibilität zu anderen Assemblern:** 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. Haben Sie keine Angabe für die erste Spalte, so leiten Sie den Befehl in dieser Zeile mit mindestens einem Leerzeichen oder einem Tabulatorzeichen ein.

● **Variablen, Marken und Kommentare:** Variablen und Marken werden bei SAM 4.0 ohne Doppelpunkt angegeben. Für Marken ist jedoch der Doppelpunkt zulässig. Kommentartexte

müssen Sie – wie allgemein 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.

● **Sprungbefehle »near« und »far«:** Den Sprungbefehlen »jmp« und »call« muß bei SAM direkt ein »n« oder »f« folgen, wenn die Zieladresse zu den Befehlen in einem Register angegeben ist. Das »n« (near) steht für einen kurzen Sprung innerhalb des Segments (Intrasegment), das »f« (far) gibt entsprechend einen weiten Sprung über die Segmentgrenzen hinweg (Intersegment) an. Ein Beispiel:

```
calln [BX]
```

Möchten Sie Daten über die Programmsegmentgrenzen hinaus schreiben oder lesen (beispielsweise in den Bildschirmspeicher), so laden Sie das Segmentregister DS mit der gewünschten Adresse. Wollen Sie beispielsweise den Wert des ersten Bytes des Bildspeichers einer CGA-Farbgrafikkarte in das AL-Register einlesen, so schreiben Sie

```
mov ax,B800h ;Segmentadresse
mov ds,ax ; in DS schreiben
mov si,0 ; und erste Speicher-
mov al,[si] ; zelle auslesen
```

Auch ein Sprung über die Segmentgrenzen hinweg, beispielsweise in das BIOS, ist erlaubt:

```
basis equ F000h
adresse equ 1000h
jmp adresse,basis
```

Taste	Befehl	Funktion
A	Assemblieren	Starten des Übersetzungsvorgangs
B	Bildschirm	SAM 4.0 gibt den Quelltext seitenweise auf dem Bildschirm aus.
C	Druckbreite	Eingabe der Druckbreite in Zeichen. Die Druckbreite muß zwischen 54 und 999 Zeichen pro Zeile liegen.
D	Drucker	SAM 4.0 listet den Quelltext unter Berücksichtigung der unter »Druckbreite« und »Drucklänge« gespeicherten Werte auf einem angeschlossenen Drucker auf.
E	Modell	Änderung des Speichermodells. Sie können zwischen Exe- und Com-Struktur wählen.
I	Betriebssystem	SAM 4.0 wird beendet, der Assembler kehrt in das Betriebssystem zurück.
K	Drucklänge	Eingabe der Seitenlänge in Zeilen. Die Seitenlänge muß zwischen 15 und 999 Zeilen pro Seite liegen.
L	Listing	Form des Assembler-Listings. SAM 4.0 bietet sowohl »keine Anzeige« als auch die Ausgabe auf Drucker, auf dem Bildschirm und in eine Datei an.
M	Maske	Eingabe der Maske für das Verzeichniskommando. SAM 4.0 erlaubt die Eingabe von Wildcards.
S	SAM-Datei	Eingabe des Quelltextnamens. SAM 4.0 ergänzt die Erweiterung automatisch. Wenn noch keine Zielfile angegeben wurde, bestimmt der Assembler eine passende.
V	Verzeichnis	SAM 4.0 gibt das Inhaltsverzeichnis der auf externem Datenträger befindlichen Dateien unter Berücksichtigung der mit »Maske« festgelegten Dateimaske aus.
Z	Zielfile	Eingabe des Zielfilenamens. SAM 4.0 ergänzt die Erweiterung selbstständig.

Tabelle 2. Die direkte Auswahl eines Befehls erleichtert die Bedienungsführung von SAM 4.0.

Ein entsprechender Unterprogrammaufruf sieht folgendermaßen aus:

```
bios equ F000h
adresse equ 2000h
mov [di],adresse
mov 2[di],bios
callf [di]
```

Die hier angegebenen BIOS-Adressen sind natürlich fiktiv und von der BIOS-Version Ihres Computers abhängig.

● **Die Präfixbefehle:** Eine weitere Besonderheit von SAM 4: Die Präfixbefehle »lock«, »rep« und »seg«, die normalerweise direkt vor den betreffenden Befehl gehören, müssen bei SAM 4.0 in eine eigene Zeile geschrieben werden, damit sie korrekt verarbeitet werden.

Im Gegensatz zu seinen Vorgängern verarbeitet SAM 4.0 Segmentierungen korrekt.

● **Prozedurdefinition:** Mit SAM 4.0 lassen sich wie bei MASM Prozeduren mit den beiden Befehlen »proc« und »enp« definieren. Sie dürfen dabei die Prozeduren auch verschachteln. Listing 2 zeigt ein Beispielprogramm. Das Beispielprogramm gibt die Meldung

(J)a oder (N)ein?

auf dem Bildschirm aus und erwartet eine Tastatureingabe. Es akzeptiert nur die Tasten [J] und [N]. Wenn Sie [J] gedrückt haben, erhält die Betriebssystem-Variable »errorlevel« den Wert 1, wenn Sie [N] drücken, den Wert 0. Das kleine Utility läßt sich gut in Stapeldateien verwenden, die oft auf Ja/Nein-Abfragen angewiesen sind. Und MS-DOS bietet diese Tastaturabfrage nicht.

Arbeiten mit der SAM-Oberfläche

Um den Umgang mit SAM 4.0 zu demonstrieren, erläutern wir Ihnen die Übersetzung von Listing 2. Angenommen, Sie haben Listing 2 unter dem Namen »janein.sam« im Verzeichnis »c:\sam4« gespeichert. Starten Sie jetzt SAM 4.0. Dann drücken Sie »E«, um eine Exe-Datei erzeugen zu lassen. Nach der Anwahl von »S« geben Sie den Dateinamen

```
c:\sam4\janein
```

ein und drücken [Enter]. SAM 4.0 ergänzt automatisch die fehlende Endung ».sam« und legt ebenfalls automatisch den Zieldateinamen fest. Nun geben Sie noch »A« ein, um den Übersetzungsvorgang zu starten. Nach erfolgreicher Übersetzung verlassen Sie SAM 4.0 und starten das Beispiel mit

```
janein
```

So einfach funktioniert die Übersetzung mit SAM 4.0. Sie müssen im Gegensatz zu MASM keinen Linker aufrufen, und bei Com-Dateien erübrigt sich auch der Einsatz von »Exe2bin«.

● **Der Befehl »include«:** Eine interessante Erweiterung bei SAM 4.0 bildet der Befehl »include«. Er fügt den Quelltext mit dem angegebenen Dateinamen in das aktuelle Listing ein. Listing 3 mit dem Namen »mbibdemo.sam« zeigt ein Beispiel für das Einbinden einer Include-Datei mit der Anweisung

```
include macrobib.sam
```

und Listing 4 die Include-Datei selbst. Listing 4, das Sie unter dem Namen »macrobib.sam« speichern, dient als Bibliothek für andere Programme. Es enthält Routinen zur Zeichen- und Textausgabe, Unterprogramme zum Löschen des Bildschirms und Positionieren des Cursors, Prozeduren zum

Sichern und Wiederherstellen aller Register sowie Anweisungen zur Ausgabe eines Bytes beziehungsweise Wortes als Hexadezimalzahl. Sie sehen, daß es keine Probleme bereitet, immer wiederkehrende Kommandos in Bibliotheken zusammenzufassen.

Eine weitreichende Leistungssteigerung erfährt SAM 4.0 mit der Makroprogrammierung. Sie können bei Makroaufrufen und -definitionen bis zu acht Parameter übergeben beziehungsweise vereinbaren, die der Assembler problemlos verarbeitet. Die Makrodefinition beginnt mit der Angabe des Makronamens, der vom Befehl »macro« und den Parametern gefolgt ist. Die Zeile

```
drucken macro %1,%2,%3
```

Die Pseudobefehle des SAM 4.0

Befehl	Bedeutung	Beispiel
● Symboldefinitionen:		
equ	weist einem symbolischen Namen einen konstanten Wert zu	anzahl equ 256
=	entspricht dem Pseudobefehl equ	wert = 256
● Datendefinitionen:		
db	definiert Byte-Speicherbereich für Variable oder Konstanten. Es lassen sich mehrere Werte zuweisen	text db 'Dies ist ein Text',10,13,'\$'
ds	reserviert einen Speicherbereich. Der erste Parameter gibt die Länge, der zweite den Inhalt an.	puffer ds 100h,0
dw	definiert Wortspeicherbereich für Variablen oder Konstanten. Es lassen sich mehrere Werte zuweisen.	zeiger dw 1024, 40h, a000h
● Makrodefinitionen:		
do	Aufruf eines zuvor definierten Makros. Der folgende Parameter gibt den Makronamen an, die weiteren werden in das Makro übernommen.	do ausgabe,al,20h,dx
endm	Ende eine Makrodefinition. Ein Makroname wird nicht benötigt.	endm
macro	die vorausgehende Marke (»Label«) definiert ein Makro mit den nachfolgenden Parameter (Prozentangaben).	ausgabe macro %1,%2,%3
● Segmentbefehle:		
assume	weist die Segmentregister zu	assume cs:code,ds:data
ends segment	markiert das Ende eines Segments kennzeichnet den Beginn eines Segments	programm ends programm segment para 'code'
● Weitere Befehle:		
end	gibt das Ende des Assembler-Programms an	end start
endp	markiert das Ende einer Prozedur	ausgabe endp
even	setzt den Befehlszähler auf eine gerade Adresse	even
include	fügt den angegebenen Quelltext in das Listing ein. Achten Sie darauf, daß das eingefügte Listing keine Fehler enthält.	include standard.sam
org	definiert die Startadresse des Programms innerhalb des Segments	org 100h
proc	kennzeichnet den Beginn einer Prozedur (Unterprogramm)	ausgabe proc far
● Ausgabesteuerung:		
title	definiert den Titel für ein Listing	
subttl	definiert den Untertitel für ein Listing	

Tabelle 3. Die Pseudobefehle von SAM 4.0 im Überblick. Sie sind zum Großteil kompatibel zu MASM.

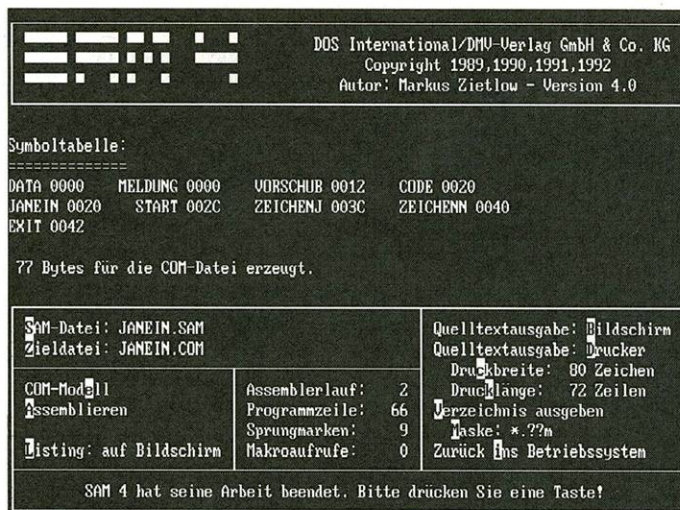


Bild 2. Während einer Übersetzung zeigt SAM 4.0 die Anzahl der Makroaufrufe, Programmzeilen und Sprungmarken an.

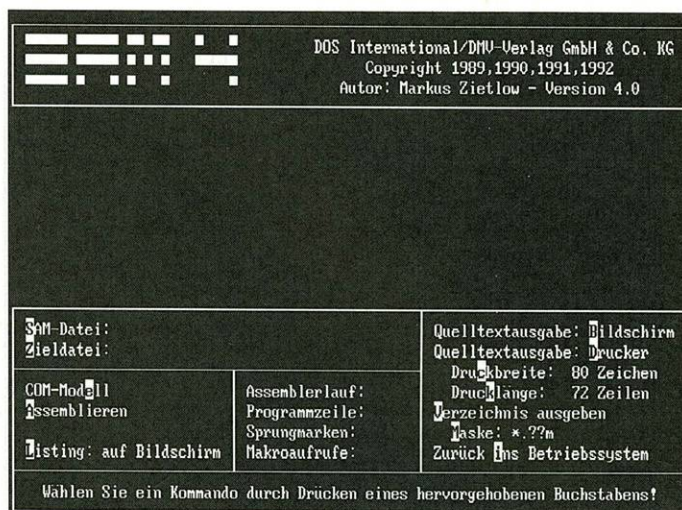


Bild 3. SAM 4.0 überzeugt durch eine neue, benutzerorientierte Oberfläche.

definiert beispielsweise das Makro »drucken«, für das drei Parameter von Bedeutung sind. Darauf müssen die Programmzeilen folgen, die parameterabhängig ausgeführt werden sollen, wie zum Beispiel

```
mov ah,%1
mov al,%2
mov cx,%3
```

Beendet wird eine Makrodefinition durch den Befehl »endm«. Um ein Makro in den Programmtext einzufügen, rufen Sie es mit dem Do-Befehl auf. Wenn beispielsweise das Makro »drucken« mit

```
do drucken,AL,20h,DX
```

angesprochen wird, so fügt SAM 4.0 die Befehle des Makros automatisch in den Programmtext ein und ersetzt dabei »%1« durch »AL«, »%2« durch »20h« und »%3« durch »DX«.

Wird das Makro hingegen mit

```
do drucken,,32
```

aktiviert, so ersetzt SAM 4.0 ausschließlich »%3« durch »32«. Die Programmzeilen, in denen »%1« und »%2« enthalten sind, werden gelöscht.

Listing 3 demonstriert die praktische Anwendung der Makrobefehle. Weil das Makrobeispiel mit nur einem Seg-

ment, dem Codesegment, arbeitet, wählen Sie vor der Assemblierung das Com-Speichermode aus. Listing 3 gibt die Inhalte der Segmentregister auf dem Monitor aus.

Auch die Pseudobefehle von SAM 4.0 wurden gegenüber seinen Vorgängern erweitert. Tabelle 3 zeigt eine Aufstellung der Pseudobefehle von SAM 4.

Zum Kompilieren von SAM 4.0 mit Quick Basic setzen Sie die Optionen »On Error« und »Resume Next«. Am schnellsten erfolgt die Übersetzung aus der MS-DOS-Kommandoebene mit

```
qb sam4 /e /o /q /s /x ;
```

und anschließend dem Linken durch

```
link sam4 ;
```

Mit diesem Assembler erübrigt sich der Einsatz von kommerziellen Assemblern.

Außerdem bekommen Sie einen Einblick über den prinzipiellen Aufbau eines Assemblers. So können Sie SAM 4.0 selbst erweitern, denn Sie sind im Besitz des Quellcodes.

● **Undokumentierte Befehle:** Sie können zum Beispiel undokumentierte Assembler-Befehle wie »loadall« beim 80286 oder »loadalld« beim 80386 in

den Quellcode einbinden und diese Highlights in Ihren Assembler-Programmen verwenden.

● **Befehle des 80586:** Sollte zum Beispiel der 80586 Prozessor auf dem Markt kommen, können Sie den SAM 4.0 um dessen Befehle erweitern und sind so früher als andere in der Lage, einen derartigen Code zu erzeugen.

● **Gedanken über SAM 5:** Ein weiteres Thema wäre zum Beispiel die Umsetzung von SAM 4.0 in eine TurboPascal-Version, um so SAM mit der objektorientierten Bibliothek Turbo Vision eine Oberfläche zu geben, in der Sie editieren und assemblieren können.

Von Interesse wäre sicherlich auch eine SAM-Version mit der Alternative der objektorientierten Assembler-Programmierung.

Sollten Sie, liebe Leser, eine solche Erweiterung für den SAM programmieren, würden wir uns freuen, wenn Sie uns Ihre SAM-Version einsenden. Die beste Version wird dann sicherlich große Chancen haben, veröffentlicht zu werden. Und daß Sie dabei nicht leer ausgehen werden, versteht sich von selbst.

(Markus Zietlow/st)

```
[GL4] 1: DECLARE SUB PseudoASSUME ()
[H45] 2: DECLARE SUB AusMarkenanzahl ()
[Z4] 3: DECLARE SUB AusMakroanzahl ()
[Z4] 4: DECLARE SUB AusLaufnummer ()
[P45] 5: DECLARE SUB AusgebenSymbole ()
[M85] 6: DECLARE SUB EndeAssemblieren ()
[HP4] 7: DECLARE SUB ErzeugenDatei ()
[YW3] 8: DECLARE SUB InitLauf2 ()
[KG4] 9: DECLARE SUB AusgebenCode ()
[4R4] 10: DECLARE SUB SymbolTabelle ()
[5D5] 11: DECLARE SUB AusgebenParameter ()
[PF5] 12: DECLARE SUB CodeDateiSchreiben ()
[7U4] 13: DECLARE SUB AnalyseSymbole ()
[LH4] 14: DECLARE SUB DefinitionDO ()
[S16] 15: DECLARE SUB Makrodefinition (Kanal%)
[AP4] 16: DECLARE SUB InitAssembler ()
[NE5] 17: DECLARE SUB BefehlAssemblieren ()
[4E5] 18: DECLARE SUB PseudoOperationen ()
```

```
[4Z3] 19: DECLARE SUB PseudoORG ()
[RY3] 20: DECLARE SUB PseudoEQU ()
[DU3] 21: DECLARE SUB PseudoDB ()
[YU3] 22: DECLARE SUB PseudoDS ()
[L34] 23: DECLARE SUB PseudoENDP ()
[K44] 24: DECLARE SUB PseudoPROC ()
[KX3] 25: DECLARE SUB PseudoEND ()
[R34] 26: DECLARE SUB PseudoEVEN ()
[3E4] 27: DECLARE SUB PseudoTITLE ()
[EC5] 28: DECLARE SUB Adressierungsart ()
[593] 29: DECLARE SUB Disp16 ()
[KK4] 30: DECLARE SUB Speichertest ()
[844] 31: DECLARE SUB OpisOffset ()
[YF4] 32: DECLARE SUB Phasenfehler ()
[LH4] 33: DECLARE SUB AnalyseZeile ()
[RU4] 34: DECLARE SUB ZeileAufteilen ()
```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic geschrieben.


```
[E65] 35: DECLARE SUB Grossbuchstaben ()
[5N4] 36: DECLARE SUB Fehlermeldung ()
[RAA] 37: DECLARE FUNCTION Input$ (Zkette$, Laenge%,
    y%, x%, Sonder$, neu%)
[GU5] 38: DECLARE FUNCTION Invers$ (Zkette$)
[UM4] 39: DECLARE SUB Status (Zkette$)
[CD4] 40: DECLARE SUB Verzeichnis ()
[CV4] 41: DECLARE SUB Ausgabefenster ()
[345] 42: DECLARE SUB Standardfenster ()
[344] 43: DECLARE SUB WarteTaste ()
[H63] 44: DECLARE SUB Menue ()
[P44] 45: DECLARE SUB InitGesamt ()
[3U5] 46: DECLARE FUNCTION Umkehr$ (Zkette$)
[W34] 47: DECLARE SUB PseudoENDS ()
[SP4] 48: DECLARE SUB PseudoSEGMENT ()
[K57] 49: DECLARE SUB ByteSchluss (AktByte AS LONG)
[864] 50: DECLARE SUB ByteDirekt ()
[W75] 51: DECLARE SUB ByteErweiterung ()
[8P4] 52: DECLARE SUB ByteSpeichern ()
[2V3] 53: DECLARE SUB PseudoDW ()
[6R4] 54: DECLARE SUB Hauptprogramm ()
[1L4] 55: DECLARE SUB AendereModell ()
[3Z4] 56: DECLARE SUB EingebenDlaenge ()
[HN4] 57: DECLARE SUB AendereAssLis ()
[5L4] 58: DECLARE SUB EingebenMaske ()
[R65] 59: DECLARE SUB EingebenQuelldat ()
[L15] 60: DECLARE SUB EingebenZielDat ()
[UP4] 61: DECLARE SUB Adrl6BitBilden ()
[D54] 62: DECLARE SUB AusDrucker ()
[G64] 63: DECLARE SUB AusMonitor ()
[9G4] 64: DECLARE SUB BefehlSuchen ()
[624] 65: DECLARE SUB CodeAufbau ()
[995] 66: DECLARE SUB DifferenzRelativ ()
[615] 67: DECLARE SUB EingebenDbreite ()
[L55] 68: DECLARE SUB LesenZeile (Kanal%)
[VB5] 69: DECLARE SUB OperandAuswerten ()
[265] 70: DECLARE SUB OperandByteWort ()
[7N4] 71: DECLARE SUB OperandOffset ()
[LU4] 72: DECLARE SUB OperandPruefen ()
[TS4] 73: DECLARE SUB OperandZeichen ()
[9W4] 74: DECLARE SUB StatusLoeschen ()
[F65] 75: DECLARE SUB SymbolEintragen ()
[EM4] 76: DECLARE SUB SymbolSuchen ()
[HL3] 77: DECLARE SUB SAMende ()
[R84] 78: DECLARE SUB Teilen16Bit ()
[UK4] 79: DECLARE SUB CodeErzeugen ()
[V64] 80: DECLARE SUB WortDirekt ()
[VP4] 81: DECLARE SUB WortSpeichern ()
[CD4] 82: DECLARE SUB ZeigeAssLis ()
[GC4] 83: DECLARE SUB ZeigeModell ()
[HCS] 84: DECLARE SUB ZeigeDruckstatus ()
[P97] 85: DECLARE SUB ZeigeDatei (Datei$, y%, x%, Lae
    nge%)
[YH3] 86: 'Makro-Assembler SAM 4
[445] 87: '(C) 1992 Daten- und Medienverlag
[1R6] 88: 'Redaktion>>DOS International<<
[M16] 89: 'Geschrieben von Markus Zietlow
[1W5] 90: 'Veröffentlicht in DOS extra
[TL1] 91: DEFINT A-Z
[E76] 92: COMMON SHARED Akku8, Akku16, AktSymbol
[BY6] 93: COMMON SHARED AssLis, Befehl$, BefehlLaenge
[ØX5] 94: COMMON SHARED BefehlTab(), Befehlzahl
[617] 95: COMMON SHARED Byte$, Checksumme AS LONG, CL
[KT5] 96: COMMON SHARED Code, CodeDa, CodeZeig
[3E7] 97: COMMON SHARED CountMakro, CountMarke, Cross
[ZU6] 98: COMMON SHARED CS, D.Z., Dateimaske$, Direkt8
[746] 99: COMMON SHARED Direkt8Bedarf, Direkt16
[AN6] 100: COMMON SHARED Direkt16Bedarf, DirektWert
[L55] 101: COMMON SHARED Disp$, Disp8Bedarf
[AM6] 102: COMMON SHARED Displ6Bedarf, Druckbreite
[E65] 103: COMMON SHARED Dzeile$, EndZeiger
[UT6] 104: COMMON SHARED ErweiBedarf, falsch, Fanzahl
[PA6] 105: COMMON SHARED FCode, Fehler, feld$, Flag
[UZ6] 106: COMMON SHARED FlagTab(), gefunden, HighByte
[XZ6] 107: COMMON SHARED KeinEnde, Kommando$, Konst1
[GK6] 108: COMMON SHARED Konst2, Lauf, Lsb, Lwidth
[DL5] 109: COMMON SHARED M.COM, M.EXE, Makro$()
[VA5] 110: COMMON SHARED MakroIndex, Marke$
[CN6] 111: COMMON SHARED MaxBefehle, ModByteBedarf
[SØ6] 112: COMMON SHARED Modell, Msb, Nah, Nichts
[AV5] 113: COMMON SHARED oben, Objekt$, ObjTyp
[BN6] 114: COMMON SHARED ObjWert1, ObjWert2, Offset
[U86] 115: COMMON SHARED OffsetBedarf, OffsetTyp
[Ø86] 116: COMMON SHARED OffsetWert, OffZusatz
[8D5] 117: COMMON SHARED Operand, OperWert()
[2Y6] 118: COMMON SHARED OperZeig, OpndZeig, param$()
[DK5] 119: COMMON SHARED Plaenge, Prozedur$()
[PX5] 120: COMMON SHARED Prozedurtyp(), Quell$
[AD6] 121: COMMON SHARED QuellDat$, QuellRegTyp()
[LU5] 122: COMMON SHARED QuellTyp, QuellWert1
[3C6] 123: COMMON SHARED QuellWert2, R, Reg8, Reg16
[G66] 124: COMMON SHARED Regis$, RegSum, RegWert
[3V5] 125: COMMON SHARED ReloAnzahl, ReloSeg()
[C16] 126: COMMON SHARED ReloOffset(), Richtung
[EL6] 127: COMMON SHARED Segmente, SegName$, SegReg
[T16] 128: COMMON SHARED Seitenlaenge, Speicher
[G66] 129: COMMON SHARED Speicheradr, SpeicherBedarf
[N47] 130: COMMON SHARED SpeicherReg, Stapel, StapelDa
[U47] 131: COMMON SHARED Stapelgroesse, StapelZeiger
```

```
[Z86] 132: COMMON SHARED SubTtl$, SymAus, Symbol$()
[276] 133: COMMON SHARED Symboltyp(), Symbolzahl
[YG6] 134: COMMON SHARED TabZeiger, Title$, Trenn$
[5Ø7] 135: COMMON SHARED unten, VorDef, wahr, Warnung
[EM6] 136: COMMON SHARED Weit, Wert1(), Wert2(), Wort
[EL6] 137: COMMON SHARED Zahltyp, ZahlWert, ZByte$
[2M6] 138: COMMON SHARED Zeichen, ZeichWert, Zeiger
[LL6] 139: COMMON SHARED Zeile$, ZeilNum, ZeilZeig1
[VL6] 140: COMMON SHARED ZeilZeig2, Ziel$, ZielDat$
[Y36] 141: COMMON SHARED ZielRegTyp(), ZielTyp
[ZU5] 142: COMMON SHARED ZielWert1, ZielWert2
[135] 143: ON ERROR GOTO Fehlerbehandlung
[PNØ] 144: CLS
[GA6] 145: 'Überprüfen, ob in der Kommandozeile
[DS4] 146: 'Parameter übergeben wurden
[933] 147: QuellDat$ = COMMAN$
[493] 148: i = INSTR(COMMAN$, " ")
[LY1] 149: ausgabe$ = " "
[MS1] 150: IF i > Ø THEN
[TM4] 151: QuellDat$ = LEFT$(COMMAN$, i - 1)
[334] 152: ZielDat$ = MID$(COMMAN$, i)
[8Ø1] 153: END IF
[8B4] 154: ZielDat$ = LTRIM$(ZielDat$)
[6L5] 155: 'Bearbeiten des Quelltextnamens
[DD3] 156: IF QuellDat$ = "-" THEN
[DZ1] 157: ausgabe$ = "-"
[312] 158: QuellDat$ = ""
[BW1] 159: ZielDat$ = ""
[ZV3] 160: ELSEIF ZielDat$ = "-" THEN
[DZ1] 161: ausgabe$ = "-"
[BW1] 162: ZielDat$ = ""
[8Ø1] 163: END IF
[HW2] 164: Pruef$ = QuellDat$
[H57] 165: 'Feststellen, ob Erweiterung angegeben ist
[FM4] 166: DO WHILE INSTR(Pruef$, "\") <> Ø
[UL6] 167: Pruef$ = MID$(Pruef$, INSTR(Pruef$, "\") +
    1)
[HSØ] 168: LOOP
[C47] 169: IF (INSTR(Pruef$, ".") = Ø) AND (Pruef$ <>
    "") THEN
[5G4] 170: QuellDat$ = QuellDat$ + ".SAM"
[8Ø1] 171: END IF
[YG5] 172: 'Bearbeiten des Zieldateinamens
[563] 173: IF ZielDat$ <> "" THEN
[Ø51] 174: dos = -1
[ZM2] 175: Pruef$ = ZielDat$
[VW4] 176: 'Feststellen, ob Erweiterung
[492] 177: 'angegeben ist
[FM4] 178: DO WHILE INSTR(Pruef$, "\") <> Ø
[UL6] 179: Pruef$ = MID$(Pruef$, INSTR(Pruef$, "\") +
    1)
[HSØ] 180: LOOP
[C47] 181: IF (INSTR(Pruef$, ".") = Ø) AND (Pruef$ <>
    "") THEN
[P54] 182: ZielDat$ = ZielDat$ + ".EXE"
[8Ø1] 183: END IF
[8Ø1] 184: END IF
[MV1] 185: InitGesamt
[E15] 186: 'Einlesen der Assemblerbefehle
[HM2] 187: READ Befehlzahl
[F44] 188: DIM Kommando$(Befehlzahl)
[L64] 189: DIM OperWert(Befehlzahl)
[AE4] 190: DIM ZielRegTyp(Befehlzahl)
[1X4] 191: DIM QuellRegTyp(Befehlzahl)
[KZ3] 192: DIM FlagTab(Befehlzahl)
[UP4] 193: FOR Zaehler = 1 TO Befehlzahl
[3T3] 194: READ Kommando$(Zaehler)
[VT3] 195: READ OperWert(Zaehler)
[194] 196: READ ZielRegTyp(Zaehler)
[1D4] 197: READ QuellRegTyp(Zaehler)
[1F3] 198: READ FlagTab(Zaehler)
[G42] 199: NEXT Zaehler
[N44] 200: READ VorDef, Symbolzahl
[244] 201: DIM Symbol$(Symbolzahl)
[CL3] 202: DIM Wert1(Symbolzahl)
[EL3] 203: DIM Wert2(Symbolzahl)
[3F4] 204: DIM Symboltyp(Symbolzahl)
[114] 205: FOR Zaehler = 1 TO VorDef
[XH3] 206: READ Symbol$(Zaehler)
[W33] 207: READ Wert1(Zaehler)
[Z33] 208: READ Wert2(Zaehler)
[RØ4] 209: READ Symboltyp(Zaehler)
[G42] 210: NEXT Zaehler
[343] 211: AktSymbol = VorDef
[UX4] 212: IF dos THEN Hauptprogramm: END
[MG7] 213: 'Titelbild, Ausgabe der Benutzeroberfläche
[Ø94] 214: ' und Copyright-Meldungen
[PNØ] 215: CLS
[ØWØ] 216: Menue
[T42] 217: ZeigeAssLis
[DØ3] 218: ZeigeDruckstatus
[616] 219: CALL ZeigeDatei(Dateimaske$, 21, 6Ø, 19)
[YH5] 220: CALL ZeigeDatei(QuellDat$, 16, 14, 34)
[WF5] 221: CALL ZeigeDatei(ZielDat$, 17, 14, 34)
[X32] 222: ZeigeModell
[3G6] 223: 'Eingabe des gewünschten Menüpunktes
[UB1] 224: Eingabe:
[P86] 225: mel$ = "Wählen Sie ein Kommando durch "
[7M5] 226: mel$ = mel$ + "Drücken eines hervor"
```



```
[RS5] 227: mel$ = mel$ + "gehobenen Buchstabens!"
[M22] 228: Status (mel$)
[RS5] 229: a$ = INKEY$: IF a$ = "" THEN GOTO Eingabe
[7L2] 230: StatusLoeschen
[P21] 231: a$ = UCASE$(a$)
[C62] 232: SELECT CASE a$
[651] 233: CASE "A"
[NC2] 234: Hauptprogramm
[A51] 235: CASE "B"
[EX1] 236: AusMonitor
[E51] 237: CASE "C"
[LS2] 238: EingebenDbreite
[K51] 239: CASE "D"
[BW1] 240: AusDrucker
[P51] 241: CASE "E"
[G82] 242: AendereModell
[661] 243: CASE "I"
[YA1] 244: SAMEnde
[E61] 245: CASE "K"
[GP2] 246: EingebenDlaenge
[K61] 247: CASE "L"
[ØB2] 248: AendereAssLis
[P61] 249: CASE "M"
[M82] 250: EingebenMaske
[E71] 251: CASE "S"
[LU2] 252: EingebenQuelldat
[U71] 253: CASE "V"
[T42] 254: Verzeichnis
[A81] 255: CASE "Z"
[ØT2] 256: EingebenZieldat
[PF1] 257: CASE ELSE
[SG6] 258: mel$ = "Unbekanntes Kommando - Drücken"
[LF5] 259: mel$ = mel$ + " Sie bitte eine Taste!"
[M22] 260: Status (mel$)
[1V1] 261: WarteTaste
[GS1] 262: END SELECT
[T12] 263: GOTO Eingabe
[E2Ø] 264: '
[DZ2] 265: Fehlerbehandlung:
[MA2] 266: Fehlermeldung
[T12] 267: Fehler = wahr
[D12] 268: RESUME NEXT
[E2Ø] 269: '
[8W4] 270: 'Anzahl der Assemblerbefehle
[M51] 271: DATA 236
[LW2] 272: 'Assemblerbefehle
[RK2] 273: DATA =,1,896,1Ø24,Ø
[P43] 274: DATA AAA,55,1Ø24,1Ø24,1
[A53] 275: DATA AAD,213,1Ø24,1Ø24,1
[K53] 276: DATA AAM,212,1Ø24,1Ø24,1
[353] 277: DATA AAS,63,1Ø24,1Ø24,1
[GZ2] 278: DATA ADC,2Ø,3,768,41Ø1
[V83] 279: DATA ADC,128,12,768,4245
[E63] 280: DATA ADC,16,144,12,1Ø29
[D53] 281: DATA ADC,18,12,144,9221
[CF3] 282: DATA ADCB,131,144,256,2193
[RH3] 283: DATA ADCW,129,144,768,4241
[1V2] 284: DATA ADD,4,3,768,41Ø1
[S83] 285: DATA ADD,128,12,768,4229
[CY2] 286: DATA ADD,Ø,144,12,1Ø29
[VY2] 287: DATA ADD,2,12,144,9221
[NF3] 288: DATA ADDB,131,144,256,2177
[ØK3] 289: DATA ADDW,129,144,768,4225
[M13] 290: DATA AND,36,3,768,41Ø1
[8A3] 291: DATA AND,128,12,768,4261
[D73] 292: DATA AND,32,144,12,1Ø29
[C63] 293: DATA AND,34,12,144,9221
[PG3] 294: DATA ANDB,131,144,256,22Ø9
[ML3] 295: DATA ANDW,129,144,768,4257
[ML4] 296: DATA ASSUME,13,&HFFFF,&HFFFF,Ø
[TK3] 297: DATA CALL,154,768,768,4Ø97
[4R3] 298: DATA CALL,232,4Ø96,1Ø24,513
[FS3] 299: DATA CALLF,255,144,1Ø24,153
[CT3] 3ØØ: DATA CALLN,255,144,1Ø24,145
[L63] 3Ø1: DATA CBW,152,1Ø24,1Ø24,1
[C73] 3Ø2: DATA CLC,248,1Ø24,1Ø24,1
[B73] 3Ø3: DATA CLD,252,1Ø24,1Ø24,1
[E73] 3Ø4: DATA CLI,25Ø,1Ø24,1Ø24,1
[D73] 3Ø5: DATA CMC,245,1Ø24,1Ø24,1
[L13] 3Ø6: DATA CMP,6Ø,3,768,41Ø1
[2B3] 3Ø7: DATA CMP,128,12,768,4285
[H83] 3Ø8: DATA CMP,56,144,12,1Ø29
[G73] 3Ø9: DATA CMP,58,12,144,9221
[SG3] 31Ø: DATA CMPB,131,144,256,2233
[GL3] 311: DATA CMPSB,166,1Ø24,1Ø24,1
[KN3] 312: DATA CMPSW,167,1Ø24,1Ø24,1
[PL3] 313: DATA CMPW,129,144,768,4281
[N83] 314: DATA CWD,153,1Ø24,1Ø24,1
[953] 315: DATA DAA,39,1Ø24,1Ø24,1
[P53] 316: DATA DAS,47,1Ø24,1Ø24,1
[3B3] 317: DATA DB,3,23Ø4,&HFFFF,Ø
[L63] 318: DATA DEC,254,4,1Ø24,137
[8V2] 319: DATA DEC,72,8,1Ø24,65
[MF3] 32Ø: DATA DECB,254,144,1Ø24,137
[ØH3] 321: DATA DECW,255,144,1Ø24,137
[8H3] 322: DATA DIV,246,144,1Ø24,8373
[SS2] 323: DATA DS,4,768,768,Ø
[LT2] 324: DATA DSB,4,768,256,Ø
```

```
[NU2] 325: DATA DSW,4,768,768,Ø
[9E3] 326: DATA DW,12,768,&HFFFF,Ø
[MW3] 327: DATA END,9,&HFFFF,&HFFFF,Ø
[N64] 328: DATA ENDP,6,&HFFFF,&HFFFF,Ø
[S84] 329: DATA ENDS,11,&HFFFF,&HFFFF,Ø
[YW2] 330: DATA EQU,1,896,1Ø24,Ø
[U74] 331: DATA EVEN,8,&HFFFF,&HFFFF,Ø
[883] 332: DATA HLT,244,1Ø24,1Ø24,1
[YR3] 333: DATA IDIV,246,144,1Ø24,8381
[4T3] 334: DATA IMUL,246,144,1Ø24,8365
[GD2] 335: DATA IN,236,1,8,1
[PD2] 336: DATA IN,237,2,8,1
[L13] 337: DATA IN,228,3,256,2Ø53
[B83] 338: DATA INC,254,4,1Ø24,129
[ØX2] 339: DATA INC,64,8,1Ø24,65
[BH3] 340: DATA INCB,254,144,1Ø24,129
[RK3] 341: DATA INCW,255,144,1Ø24,129
[383] 342: DATA INT,2Ø4,1Ø24,1Ø24,1
[KH3] 343: DATA INT,2Ø5,256,1Ø24,2Ø49
[PD3] 344: DATA INTO,2Ø6,1Ø24,1Ø24,1
[3E3] 345: DATA IRET,2Ø7,1Ø24,1Ø24,1
[NB3] 346: DATA JA,119,4Ø96,1Ø24,257
[9F3] 347: DATA JAE,115,4Ø96,1Ø24,257
[5B3] 348: DATA JB,114,4Ø96,1Ø24,257
[GF3] 349: DATA JBE,118,4Ø96,1Ø24,257
[9B3] 350: DATA JC,114,4Ø96,1Ø24,257
[8U3] 351: DATA JCXZ,227,4Ø96,1Ø24,257
[TB3] 352: DATA JE,116,4Ø96,1Ø24,257
[8C3] 353: DATA JG,127,4Ø96,1Ø24,257
[5G3] 354: DATA JGE,125,4Ø96,1Ø24,257
[GC3] 355: DATA JL,124,4Ø96,1Ø24,257
[UG3] 356: DATA JLE,126,4Ø96,1Ø24,257
[1F3] 357: DATA JMP,234,768,768,4Ø97
[WG3] 358: DATA JMP,233,4Ø96,1Ø24,513
[9K3] 359: DATA JMPF,255,144,1Ø24,169
[1K3] 360: DATA JMPN,255,144,1Ø24,161
[RU3] 361: DATA JMPS,235,4Ø96,1Ø24,257
[WG3] 362: DATA JNA,118,4Ø96,1Ø24,257
[8T3] 363: DATA JNAE,114,4Ø96,1Ø24,257
[UG3] 364: DATA JNB,115,4Ø96,1Ø24,257
[LT3] 365: DATA JNBE,119,4Ø96,1Ø24,257
[VG3] 366: DATA JNC,115,4Ø96,1Ø24,257
[ZG3] 367: DATA JNE,117,4Ø96,1Ø24,257
[4H3] 368: DATA JNG,126,4Ø96,1Ø24,257
[FT3] 369: DATA JNGE,124,4Ø96,1Ø24,257
[8H3] 370: DATA JNL,125,4Ø96,1Ø24,257
[UT3] 371: DATA JNLE,127,4Ø96,1Ø24,257
[5H3] 372: DATA JNO,113,4Ø96,1Ø24,257
[AH3] 373: DATA JNP,123,4Ø96,1Ø24,257
[BH3] 374: DATA JNS,121,4Ø96,1Ø24,257
[MH3] 375: DATA JNZ,117,4Ø96,1Ø24,257
[HC3] 376: DATA JO,112,4Ø96,1Ø24,257
[SC3] 377: DATA JP,122,4Ø96,1Ø24,257
[6H3] 378: DATA JPE,122,4Ø96,1Ø24,257
[HH3] 379: DATA JPO,123,4Ø96,1Ø24,257
[WC3] 380: DATA JS,12Ø,4Ø96,1Ø24,257
[DE3] 381: DATA JZ,116,4Ø96,1Ø24,257
[MB3] 382: DATA LAHF,159,1Ø24,1Ø24,1
[A83] 383: DATA LDS,197,8,144,9217
[273] 384: DATA LEA,141,8,144,9217
[D83] 385: DATA LES,196,8,144,9217
[2D3] 386: DATA LOCK,24Ø,1Ø24,1Ø24,1
[XL3] 387: DATA LODSB,172,1Ø24,1Ø24,1
[ZN3] 388: DATA LODSW,173,1Ø24,1Ø24,1
[ZU3] 389: DATA LOOP,226,4Ø96,1Ø24,257
[TX3] 390: DATA LOOPE,225,4Ø96,1Ø24,257
[H44] 391: DATA LOOPNE,224,4Ø96,1Ø24,257
[574] 392: DATA LOOPNZ,224,4Ø96,1Ø24,257
[SZ3] 393: DATA LOOPZ,225,4Ø96,1Ø24,257
[5B3] 394: DATA MOV,16Ø,3,128,16389
[5B3] 395: DATA MOV,162,128,3,16389
[YA3] 396: DATA MOV,138,12,144,9221
[3A3] 397: DATA MOV,136,144,12,1Ø29
[V93] 398: DATA MOV,184,8,768,4161
[HF3] 399: DATA MOV,183,3,4Ø96,16389
[Ø93] 400: DATA MOV,176,4,256,4161
[NB3] 4Ø1: DATA MOV,142,64,144,9217
[5B3] 4Ø2: DATA MOV,14Ø,144,96,1Ø25
[4L3] 4Ø3: DATA MOVB,198,144,256,4225
[PM3] 4Ø4: DATA MOVSB,164,1Ø24,1Ø24,1
[SP3] 4Ø5: DATA MOVSW,165,1Ø24,1Ø24,1
[9N3] 4Ø6: DATA MOVW,199,144,768,4225
[HL3] 4Ø7: DATA MUL,246,144,1Ø24,8357
[S83] 4Ø8: DATA NEG,246,12,1Ø24,157
[HG3] 4Ø9: DATA NEGB,246,144,1Ø24,153
[XH3] 410: DATA NEGW,247,144,1Ø24,153
[Z83] 411: DATA NOP,144,1Ø24,1Ø24,1
[BA3] 412: DATA NOT,246,12,1Ø24,149
[AK3] 413: DATA NOTB,246,144,1Ø24,145
[PL3] 414: DATA NOTW,247,144,1Ø24,145
[FX2] 415: DATA OR,12,3,768,41Ø1
[H93] 416: DATA OR,128,12,768,4237
[UW2] 417: DATA OR,8,144,12,1Ø29
[913] 418: DATA OR,1Ø,12,144,9221
[UD3] 419: DATA ORB,128,144,256,4233
[MX2] 420: DATA ORG,2,768,1Ø24,Ø
[5F3] 421: DATA ORW,129,144,768,4233
```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic geschrieben.


```
[FL2] 422: DATA OUT,238,8,1,1
[HL2] 423: DATA OUT,239,8,2,1
[H93] 424: DATA OUT,239,256,3,2053
[WY2] 425: DATA POP,88,8,1024,65
[6D3] 426: DATA POP,143,144,1024,129
[BX2] 427: DATA POP,7,64,1024,65
[WD3] 428: DATA POPF,157,1024,1024,1
[DB3] 429: DATA PROC,5,12288,1024,0
[N53] 430: DATA PUSH,80,8,1024,65
[YL3] 431: DATA PUSH,255,144,1024,177
[X53] 432: DATA PUSH,6,96,1024,65
[EN3] 433: DATA PUSHF,156,1024,1024,1
[3T3] 434: DATA RCL,208,12,17408,&H8095
[U34] 435: DATA RCLB,208,144,17408,&H8091
[654] 436: DATA RCLW,209,144,17408,&H8091
[ST3] 437: DATA RCR,208,12,17408,&H809D
[S44] 438: DATA RCRB,208,144,17408,&H8099
[464] 439: DATA RCRW,209,144,17408,&H8099
[583] 440: DATA REP,243,1024,1024,1
[LC3] 441: DATA REPE,243,1024,1024,1
[DL3] 442: DATA REPNE,242,1024,1024,1
[CN3] 443: DATA REPNZ,242,1024,1024,1
[XD3] 444: DATA REPZ,243,1024,1024,1
[Y83] 445: DATA RET,203,1024,8192,1
[W93] 446: DATA RET,195,1024,4096,1
[3L3] 447: DATA RET,202,768,8192,4097
[YL3] 448: DATA RET,194,768,4096,4097
[FU3] 449: DATA ROL,208,12,17408,&H8085
[854] 450: DATA ROLB,208,144,17408,&H8081
[M64] 451: DATA ROLW,209,144,17408,&H8081
[4V3] 452: DATA ROR,208,12,17408,&H808D
[664] 453: DATA RORB,208,144,17408,&H8089
[K74] 454: DATA RORW,209,144,17408,&H8089
[7C3] 455: DATA SAHF,158,1024,1024,1
[YT3] 456: DATA SAL,208,12,17408,&H80A5
[544] 457: DATA SALB,208,144,17408,&H80A1
[H54] 458: DATA SALW,209,144,17408,&H80A1
[RU3] 459: DATA SAR,208,12,17408,&H80BD
[554] 460: DATA SARB,208,144,17408,&H80B9
[H64] 461: DATA SARW,209,144,17408,&H80B9
[X13] 462: DATA SBB,28,3,768,4101
[AA3] 463: DATA SBB,128,12,768,4253
[P73] 464: DATA SBB,24,144,12,1029
[N63] 465: DATA SBB,26,12,144,9221
[RH3] 466: DATA SBBB,128,144,256,4249
[W13] 467: DATA SBBW,129,144,768,4249
[5L3] 468: DATA SCASB,174,1024,1024,1
[7N3] 469: DATA SCASW,175,1024,1024,1
[Z23] 470: DATA SEG,38,96,1024,65
[2W4] 471: DATA SEGMENT,10,&HFFFF,&HFFFF,0
[LU3] 472: DATA SHL,208,12,16640,&H80A5
[154] 473: DATA SHLB,208,144,17408,&H80A1
[D64] 474: DATA SHLW,209,144,17408,&H80A1
[8V3] 475: DATA SHR,208,12,16640,&H80AD
[Z54] 476: DATA SHRB,208,144,17408,&H80A9
[B74] 477: DATA SHRW,209,144,17408,&H80A9
[X93] 478: DATA STC,249,1024,1024,1
[W93] 479: DATA STD,253,1024,1024,1
[Z93] 480: DATA STI,251,1024,1024,1
[BN3] 481: DATA STOSB,170,1024,1024,1
[DR3] 482: DATA STOSW,171,1024,1024,1
[Z33] 483: DATA SUB,44,3,768,4101
[D03] 484: DATA SUB,128,12,768,4269
[S93] 485: DATA SUB,40,144,12,1029
[R83] 486: DATA SUB,42,12,144,9221
[VL3] 487: DATA SUBB,128,144,256,4265
[BL4] 488: DATA SUBTTL,7,&HFFFF,&HFFFF,0
[P03] 489: DATA SUBW,129,144,768,4265
[TC3] 490: DATA TEST,168,3,768,4101
[UF3] 491: DATA TEST,246,12,768,4229
[LD3] 492: DATA TEST,132,144,12,1029
[EE3] 493: DATA TEST,132,12,144,9221
[YU3] 494: DATA TESTB,246,144,256,4225
[DX3] 495: DATA TESTW,247,144,768,4225
[GB4] 496: DATA TITLE,7,&HFFFF,&HFFFF,0
[AD3] 497: DATA WAIT,155,1024,1024,1
[P93] 498: DATA XCHG,144,8,1024,65
[R33] 499: DATA XCHG,144,2,8,8257
[PC3] 500: DATA XCHG,134,144,12,1029
[HD3] 501: DATA XCHG,134,12,144,9221
[HE3] 502: DATA XLAT,215,1024,1024,1
[143] 503: DATA XOR,52,3,768,4101
[ZA3] 504: DATA XOR,48,144,12,1029
[T83] 505: DATA XOR,50,12,144,9221
[9D3] 506: DATA XOR,128,12,768,4277
[YL3] 507: DATA XORB,128,144,256,4273
[3P3] 508: DATA XORW,129,144,768,4273
[A26] 509: 'Anzahl der vordefinierten Symbole,
[VN4] 510: 'maximale Anzahl der Symbole
[W11] 511: DATA 29,500
[MS3] 512: 'Vordefinierte Symbole
[EA2] 513: DATA AX,192,0,26
[6B2] 514: DATA CX,193,8,24
[BE2] 515: DATA DX,194,16,24
[5E2] 516: DATA BX,195,24,24
[SE2] 517: DATA SP,196,32,24
[5D2] 518: DATA BP,197,40,24
[SE2] 519: DATA SI,198,48,24
[BD2] 520: DATA DI,199,56,24
```

```
[T82] 521: DATA AL,192,0,21
[WR2] 522: DATA CL,193,8,16404
[LC2] 523: DATA DL,194,16,20
[DC2] 524: DATA BL,195,24,20
[UB2] 525: DATA AH,196,32,20
[0C2] 526: DATA CH,197,40,20
[ZC2] 527: DATA DH,198,48,20
[TC2] 528: DATA BH,199,56,20
[8Z2] 529: DATA [BX+SI],0,0,16
[YX2] 530: DATA [BX+DI],1,0,16
[6Z2] 531: DATA [BP+SI],2,0,16
[WX2] 532: DATA [BP+DI],3,0,16
[UE2] 533: DATA [SI],4,0,16
[2D2] 534: DATA [DI],5,0,16
[HD2] 535: DATA [BX],7,0,16
[V12] 536: DATA ES,0,0,64
[922] 537: DATA CS,0,8,32
[N92] 538: DATA SS,0,16,64
[A82] 539: DATA DS,0,24,64
[7L2] 540: DATA NEAR,0,0,4096
[DB2] 541: DATA FAR,0,0,8192
[E20] 542: '
[000] 543:
[FL4] 544: 'Bildet eine 16-Bit-Adresse
[543] 545: SUB Adr16BitBilden
[G84] 546: IF ZielTyp = Speicher THEN
[RN2] 547: Lsb = ZielWert1
[ZR0] 548: ELSE
[2T2] 549: Lsb = QuellWert1
[801] 550: END IF
[402] 551: Teilen16Bit
[3Y4] 552: Befehllaenge = Befehllaenge + 2
[405] 553: BefehlTab(Befehllaenge - 1) = Lsb
[8M4] 554: BefehlTab(Befehllaenge) = Msb
[HA1] 555: END SUB
[000] 556:
[G47] 557: 'Unterprogramm, das die Adressierungsart
[5P1] 558: 'behandelt
[RP3] 559: SUB Adressierungsart
[ZXA] 560: Befehllaenge = Befehllaenge + 1
[532] 561: OffZusatz = 0
[7V8] 562: IF ((ZielTyp OR QuellTyp) AND Speicher) <
> falsch THEN
[G84] 563: IF ZielTyp = Speicher THEN
[GA2] 564: M = QuellWert2
[ZR0] 565: ELSE
[U32] 566: M = ZielWert2
[801] 567: END IF
[ZU4] 568: BefehlTab(Befehllaenge) = 6 + M
[SD2] 569: Adr16BitBilden
[TE1] 570: EXIT SUB
[801] 571: END IF
[L44] 572: IF Flag AND Richtung THEN
[G74] 573: M = QuellWert1 + ZielWert2
[ZR0] 574: ELSE
[674] 575: M = ZielWert1 + QuellWert2
[801] 576: END IF
[HD4] 577: BefehlTab(Befehllaenge) = M
[E66] 578: IF OffsetBedarf = Nichts THEN EXIT SUB
[326] 579: IF (Offset <= 127) AND (Offset >= -128) T
HEN
[WG8] 580: BefehlTab(Befehllaenge) = BefehlTab(Bef
ehllaenge) + 64
[6S2] 581: IF Offset < 0 THEN
[1S3] 582: Offset = Offset AND 255
[K72] 583: OffZusatz = -1
[801] 584: END IF
[ZXA] 585: Befehllaenge = Befehllaenge + 1
[C75] 586: BefehlTab(Befehllaenge) = Offset
[TE1] 587: EXIT SUB
[801] 588: END IF
[CR8] 589: BefehlTab(Befehllaenge) = BefehlTab(Befeh
llaenge) + 128
[3Y4] 590: Befehllaenge = Befehllaenge + 2
[202] 591: Lsb = Offset
[402] 592: Teilen16Bit
[405] 593: BefehlTab(Befehllaenge - 1) = Lsb
[8M4] 594: BefehlTab(Befehllaenge) = Msb
[W84] 595: IF Msb = 0 THEN OffZusatz = 1
[HA1] 596: END SUB
[000] 597:
[ZS4] 598: 'Form des Assembler-Listings
[D13] 599: SUB AendereAssLis
[P23] 600: AssLis = AssLis + 1
[2C4] 601: IF AssLis = 5 THEN AssLis = 1
[T42] 602: ZeigeAssLis
[HA1] 603: END SUB
[000] 604:
[XY2] 605: SUB AendereModell
[MZ2] 606: Modell = Modell + 1
[TP2] 607: IF Modell = 3 THEN
[W52] 608: Modell = M.COM
[801] 609: END IF
[X32] 610: ZeigeModell
[MM3] 611: IF LEN(ZielDat$) > 3 THEN
[N78] 612: IF (RIGHT$(ZielDat$, 4) = ".EXE") AND M
odell = M.COM THEN
[7X5] 613: MID$(ZielDat$, LEN(ZielDat$) - 3) = ".COM"
[801] 614: END IF
```



```
[S88] 615: IF (RIGHT$(ZielDat$, 4) = ".COM") AND M
        odell = M.EXE THEN
[8X5] 616: MID$(ZielDat$, LEN(ZielDat$) - 3) = ".EXE"
[801] 617: END IF
[WF5] 618: CALL ZeigeDatei(ZielDat$, 17, 14, 34)
[801] 619: END IF
[HA1] 620: END SUB
[000] 621:
[V06] 622: 'Eintrag des Symbols in die Tabelle
[K73] 623: SUB AnalyseSymbole
[HG2] 624: Befehllaenge = 0
[956] 625: IF Befehl$ = "ENDS" OR Befehl$ = "ENDP" T
        HEN
[DW2] 626: GOTO schluss.seg
[801] 627: END IF
[F24] 628: IF Befehl$ <> "SEGMENT" THEN
[7P5] 629: IF Marke$ <> "" THEN SymbolEintragen
[801] 630: END IF
[362] 631: schluss.seg:
[W46] 632: IF Befehl$ <> "" THEN BefehlAssemblieren
[614] 633: IF Befehl$ = "SEGMENT" THEN
[7P5] 634: IF Marke$ <> "" THEN SymbolEintragen
[801] 635: END IF
[FY4] 636: IF Marke$ <> "" OR Befehl$ <> "" THEN
[R73] 637: IF NOT KeinEnde THEN
[691] 638: PRINT #2,
[8P5] 639: PRINT #2, " Der END-Befehl wurde zu";
[TX5] 640: PRINT #2, " früh oder gar nicht ben";
[NG3] 641: PRINT #2, "utzt. SAM 4 ";
[V34] 642: PRINT #2, "unterbricht die"
[B95] 643: PRINT #2, " Übersetzung der ";
[7B5] 644: PRINT #2, "SAM-Datei. Drücken Sie ";
[445] 645: PRINT #2, "bitte eine beliebige ";
[UG2] 646: PRINT #2, "Taste!"
[1V1] 647: WarteTaste
[GG2] 648: KeinEnde = wahr
[801] 649: END IF
[801] 650: END IF
[206] 651: IF Befehl$ = "END" THEN KeinEnde = falsch
[HA1] 652: END SUB
[000] 653:
[L86] 654: 'Unterprogramm, in dem die eingelesene
[DN3] 655: 'Zeile übersetzt wird
[WV2] 656: SUB AnalyseZeile
[T02] 657: ZeilZeigl = 1
[V02] 658: ZeilZeig2 = 1
[HE1] 659: Marke$ = ""
[SK1] 660: Befehl$ = ""
[GF1] 661: Quell$ = ""
[NA1] 662: Ziel$ = ""
[GS3] 663: EndZeiger = LEN(Zeile$)
[CM4] 664: IF EndZeiger = 0 THEN EXIT SUB
[VX2] 665: Grossbuchstaben
[796] 666: IF INSTR(Trenn$, LEFT$(Zeile$, 1)) = 0 THEN
[NH2] 667: ZeileAufteilen
[902] 668: Marke$ = feld$
[6C4] 669: IF RIGHT$(Marke$, 1) = ":" THEN
[ME5] 670: Marke$ = LEFT$(Marke$, LEN(Marke$) - 1)
[801] 671: END IF
[801] 672: END IF
[NH2] 673: ZeileAufteilen
[754] 674: IF NOT gefunden THEN EXIT SUB
[Z72] 675: Befehl$ = feld$
[AB3] 676: OpndZeig = ZeilZeigl
[NH2] 677: ZeileAufteilen
[754] 678: IF NOT gefunden THEN EXIT SUB
[2Y1] 679: Ziel$ = feld$
[NH2] 680: ZeileAufteilen
[754] 681: IF NOT gefunden THEN EXIT SUB
[812] 682: Quell$ = feld$
[HA1] 683: END SUB
[000] 684:
[EE4] 685: 'Ausgabe eines Quelltextes
[EM2] 686: 'auf dem Drucker
[SK2] 687: SUB AusDrucker
[K93] 688: IF QuellDat$ = "" THEN
[E16] 689: mel$ = "Sie haben noch keine SAM-Datei"
[RU5] 690: mel$ = mel$ + " angeben. Drücken Sie"
[YT4] 691: mel$ = mel$ + " bitte eine Taste!"
[M22] 692: Status(mel$)
[1V1] 693: WarteTaste
[7L2] 694: StatusLoeschen
[ZR0] 695: ELSE
[TC5] 696: mel$ = "Die angegebene Datei wird "
[574] 697: mel$ = mel$ + "ausgedruckt. "
[M22] 698: Status(mel$)
[AK2] 699: Ausgabefenster
[FM1] 700: LOCATE 10, 1
[V76] 701: PRINT "Anzahl der ausgegebenen Zeilen:"
[WT4] 702: OPEN QuellDat$ FOR INPUT AS #1
[E44] 703: IF Fehler THEN GOTO Druckende
[923] 704: Zeilen = 0: Seite = 1
[533] 705: Azeilen = 0: Flag = 0
[E20] 706:
[Y22] 707: DO WHILE NOT EOF(1)
[VB3] 708: LINE INPUT #1, PZeile$
[6K3] 709: IF Fehler THEN EXIT DO
[691] 710: Druck1:
[4W2] 711: IF Azeilen = 0 THEN
```

```
[6D4] 712: A1 = INT((Druckbreite - 54) / 2)
[6L3] 713: A2 = Druckbreite - 54 - A1
[LN5] 714: LPRINT "Listing einer SAM-Datei ";
[924] 715: LPRINT STRING$(A1, 32); "Datum: ";
[Y04] 716: LPRINT MID$(DATE$, 4, 2); ". ";
[304] 717: LPRINT LEFT$(DATE$, 2); ". ";
[S44] 718: LPRINT RIGHT$(DATE$, 4); ". ";
[305] 719: LPRINT STRING$(A2, 32); "Seite: ";
[AS3] 720: LPRINT USING "###"; Seite
[6K3] 721: IF Fehler THEN EXIT DO
[H51] 722: LPRINT
[6K3] 723: IF Fehler THEN EXIT DO
[NA3] 724: LPRINT "SAM-Datei: ";
[6K3] 725: IF Fehler THEN EXIT DO
[526] 726: IF LEN(QuellDat$) > (Druckbreite -
        1) - 11 THEN
[DW7] 727: LPRINT LEFT$(QuellDat$, (Druckbre
        ite - 1) - 7); " ..."
[ZR0] 728: ELSE
[PT2] 729: LPRINT QuellDat$
[801] 730: END IF
[6K3] 731: IF Fehler THEN EXIT DO
[6C5] 732: LPRINT STRING$(Druckbreite - 1, "-")
[6K3] 733: IF Fehler THEN EXIT DO
[H51] 734: LPRINT
[6K3] 735: IF Fehler THEN EXIT DO
[UU1] 736: Azeilen = 5
[801] 737: END IF
[002] 738: IF Flag <> 0 THEN
[FL2] 739: LPRINT SPACES(6);
[N36] 740: LPRINT LEFT$(PZeile$, Druckbreite - 7)
[6K3] 741: IF Fehler THEN EXIT DO
[K21] 742: GOTO Druck2
[801] 743: END IF
[803] 744: Zeilen = Zeilen + 1
[R11] 745: LOCATE 10, 33
[AZ3] 746: PRINT USING "####"; Zeilen
[K64] 747: LPRINT USING "####"; Zeilen;
[6K3] 748: IF Fehler THEN EXIT DO
[AR1] 749: LPRINT ": ";
[N36] 750: LPRINT LEFT$(PZeile$, Druckbreite - 7)
[6K3] 751: IF Fehler THEN EXIT DO
[991] 752: Druck2:
[3R5] 753: IF LEN(PZeile$) > Druckbreite - 7 THEN
[161] 754: Flag = -1
[H46] 755: PZeile$ = MID$(PZeile$, Druckbreite - 6)
[ZR0] 756: ELSE
[231] 757: Flag = 0
[801] 758: END IF
[873] 759: Azeilen = Azeilen + 1
[U95] 760: IF Azeilen = Seitenlaenge - 3 THEN
[H51] 761: LPRINT
[6K3] 762: IF Fehler THEN EXIT DO
[H51] 763: LPRINT
[6K3] 764: IF Fehler THEN EXIT DO
[H51] 765: LPRINT
[6K3] 766: IF Fehler THEN EXIT DO
[6U1] 767: Azeilen = 0
[GF2] 768: Seite = Seite + 1
[801] 769: END IF
[SV3] 770: IF Flag THEN GOTO Druck1
[HS0] 771: LOOP
[E44] 772: IF Fehler THEN GOTO Druckende
[622] 773: IF Azeilen <> 0 THEN
[355] 774: FOR q = 1 TO Seitenlaenge - Azeilen
[H51] 775: LPRINT
[KW3] 776: IF Fehler THEN EXIT FOR
[K21] 777: NEXT q
[801] 778: END IF
[E44] 779: IF Fehler THEN GOTO Druckende
[801] 780: END IF
[GV2] 781: Standardfenster
[656] 782: mel$ = "Die Datei wurde ausgedruckt. "
[AV5] 783: mel$ = mel$ + "Drücken Sie bitte eine "
[922] 784: mel$ = mel$ + "Taste!"
[M22] 785: Status(mel$)
[1V1] 786: WarteTaste
[7L2] 787: StatusLoeschen
[ET1] 788: Druckende:
[NF2] 789: Fehler = falsch
[3W0] 790: CLOSE
[HA1] 791: END SUB
[000] 792:
[R83] 793: SUB Ausgabefenster
[914] 794: VIEW PRINT oben TO unten
[UT0] 795: CLS 2
[HA1] 796: END SUB
[000] 797:
[217] 798: 'Unterprogramm, das den Programmcode und
[NX4] 799: 'das Assemblerlisting ausgibt
[VU2] 800: SUB AusgebenCode
[5C5] 801: hv$ = HEX$(CodeZeig - Befehllaenge)
[WY4] 802: hv$ = STRING$(4 - LEN(hv$), "0") + hv$
[SX2] 803: IF AssLis <> 4 THEN
[523] 804: PRINT #2, TAB(1); hv$;
[801] 805: END IF
```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic geschrieben.


```
[810] 806: i = 1
[2B5] 807: IF AssLis <> 4 THEN PRINT #2, TAB(6);
[N52] 808: DO WHILE i <= 6
[005] 809: IF i > Befehllaenge THEN EXIT DO
[55A] 810: LSET Byte$ = CHR$(BefehlTab(i))
[LE3] 811: D.Z = D.Z + 1: PUT #3, D.Z
[063] 812: Plaenge = Plaenge + 1
[9G3] 813: hv$ = HEX$(BefehlTab(i))
[UT4] 814: IF LEN(hv$) = 1 THEN hv$ = "0" + hv$
[DY4] 815: IF AssLis <> 4 THEN PRINT #2, hv$;
[611] 816: i = i + 1
[HS0] 817: LOOP
[818] 818: IF AssLis <> 4 THEN PRINT #2, TAB(19);
[TU5] 819: IF AssLis <> 4 THEN PRINT #2, ZeilNum;
[9T2] 820: IF AssLis = 3 THEN
[HE4] 821: PRINT #2, SPACE$(2); Dzeile$
[RN3] 822: ELSEIF AssLis <> 4 THEN
[AV2] 823: PRINT #2, SPACE$(2);
[8F5] 824: PRINT #2, LEFT$(Dzeile$, Lwidth - 26)
[801] 825: END IF
[BO4] 826: DO WHILE i <= Befehllaenge
[HN2] 827: IF i MOD 6 = 1 THEN
[2B5] 828: IF AssLis <> 4 THEN PRINT #2, TAB(6);
[801] 829: END IF
[55A] 830: LSET Byte$ = CHR$(BefehlTab(i))
[LE3] 831: D.Z = D.Z + 1: PUT #3, D.Z
[063] 832: Plaenge = Plaenge + 1
[9G3] 833: hv$ = HEX$(BefehlTab(i))
[UT4] 834: IF LEN(hv$) = 1 THEN hv$ = "0" + hv$
[DY4] 835: IF AssLis <> 4 THEN PRINT #2, hv$;
[611] 836: i = i + 1
[HS0] 837: LOOP
[YR3] 838: IF Befehllaenge > 6 THEN
[Y74] 839: IF AssLis <> 4 THEN PRINT #2,
[801] 840: END IF
[HAL] 841: END SUB
[000] 842:
[KE5] 843: 'Angabe der aktuellen Parameter
[1T3] 844: SUB AusgebenParameter
[0G1] 845: x = POS(0)
[NN1] 846: y = CSRLIN
[RW2] 847: VIEW PRINT 1 TO 25
[VR1] 848: LOCATE 20, 44
[0C4] 849: PRINT USING "####"; ZeilNum;
[914] 850: VIEW PRINT oben TO unten
[UT1] 851: LOCATE y, x
[HAL] 852: END SUB
[000] 853:
[PD4] 854: 'Ausgabe der Symboltabelle
[6K3] 855: SUB AusgebenSymbole
[691] 856: PRINT #2,
[8Z3] 857: PRINT #2, "Symboltabelle:"
[HK3] 858: PRINT #2, "=====
[502] 859: i = VorDef + 1
[MC3] 860: perzeil = Lwidth \ 19
[4A3] 861: IF i > AktSymbol THEN
[691] 862: PRINT #2,
[X64] 863: PRINT #2, "Es wurden keine ";
[9G4] 864: PRINT #2, "Symbole definiert."
[912] 865: GOTO sym.w1
[801] 866: END IF
[ZS3] 867: DO WHILE i <= AktSymbol
[DW2] 868: hv$ = HEX$(Wert1(i))
[WY4] 869: hv$ = STRING$(4 - LEN(hv$), "0") + hv$
[N25] 870: PRINT #2, Symbol$(i); " "; hv$; " ";
[611] 871: i = i + 1
[MH5] 872: IF (i - VorDef) MOD perzeil = 1 THEN
[691] 873: PRINT #2,
[801] 874: END IF
[HS0] 875: LOOP
[W91] 876: sym.w1:
[691] 877: PRINT #2,
[HAL] 878: END SUB
[000] 879:
[V33] 880: SUB AusLaufnummer
[0G1] 881: x = POS(0)
[NN1] 882: y = CSRLIN
[RW2] 883: VIEW PRINT 1 TO 25
[CS1] 884: LOCATE 19, 44
[ER3] 885: PRINT USING "####"; Lauf;
[914] 886: VIEW PRINT oben TO unten
[UT1] 887: LOCATE y, x
[HAL] 888: END SUB
[000] 889:
[AB3] 890: SUB AusMakroanzahl
[0G1] 891: x = POS(0)
[NN1] 892: y = CSRLIN
[RW2] 893: VIEW PRINT 1 TO 25
[ZR1] 894: LOCATE 22, 44
[LZ4] 895: PRINT USING "####"; CountMakro;
[914] 896: VIEW PRINT oben TO unten
[UT1] 897: LOCATE y, x
[HAL] 898: END SUB
[000] 899:
[BK3] 900: SUB AusMarkenanzahl
[0G1] 901: x = POS(0)
[NN1] 902: y = CSRLIN
[RW2] 903: VIEW PRINT 1 TO 25
```

```
[XR1] 904: LOCATE 21, 44
[EY4] 905: PRINT USING "####"; CountMarke;
[914] 906: VIEW PRINT oben TO unten
[UT1] 907: LOCATE y, x
[HAL] 908: END SUB
[000] 909:
[EE4] 910: 'Ausgabe eines Quelltextes
[Z83] 911: 'auf dem Bildschirm
[VL2] 912: SUB AusMonitor
[K93] 913: IF QuellDat$ = "" THEN
[345] 914: mel$ = "Sie haben noch keine SAM-"
[3P4] 915: mel$ = mel$ + "Datei angegeben. "
[AV5] 916: mel$ = mel$ + "Drücken Sie bitte eine "
[9Z2] 917: mel$ = mel$ + "Taste!"
[M22] 918: Status (mel$)
[1V1] 919: WarteTaste
[7L2] 920: StatusLoeschen
[ZR0] 921: ELSE
[AK2] 922: Ausgabefenster
[WT4] 923: OPEN QuellDat$ FOR INPUT AS #1
[AL4] 924: IF Fehler THEN GOTO Bildende
[MF1] 925: Zeilen = 0
[YZ2] 926: DO WHILE NOT EOF(1)
[VB3] 927: LINE INPUT #1, Pzeile$
[6K3] 928: IF Fehler THEN EXIT DO
[803] 929: Zeilen = Zeilen + 1
[P64] 930: PRINT USING "####"; Zeilen;
[6W2] 931: PRINT ": "; Pzeile$
[NX2] 932: IF CSRLIN = 14 THEN
[GV2] 933: Standardfenster
[8A5] 934: mel$ = "Drücken Sie bitte eine "
[N54] 935: mel$ = mel$ + "Taste, um die "
[DB4] 936: mel$ = mel$ + "nächste Seite "
[Z65] 937: mel$ = mel$ + "ausgeben zu lassen!"
[M22] 938: Status (mel$)
[1V1] 939: WarteTaste
[7L2] 940: StatusLoeschen
[AK2] 941: Ausgabefenster
[UT0] 942: CLS 2
[801] 943: END IF
[HS0] 944: LOOP
[AL4] 945: IF Fehler THEN GOTO Bildende
[GV2] 946: Standardfenster
[N54] 947: mel$ = "Die Ausgabe der Datei ist "
[MT5] 948: mel$ = mel$ + "beendet. Drücken Sie "
[TU4] 949: mel$ = mel$ + "bitte eine Taste!"
[M22] 950: Status (mel$)
[1V1] 951: WarteTaste
[7L2] 952: StatusLoeschen
[GG1] 953: Bildende:
[3W0] 954: CLOSE
[NF2] 955: Fehler = falsch
[801] 956: END IF
[HAL] 957: END SUB
[000] 958:
[8V5] 959: 'Unterprogramm, das die angegebene
[KM3] 960: 'Operation übersetzt
[0W3] 961: SUB BefehlAssemblieren
[59U] 962: IF Befehl$ <> "ASSUME" AND Befehl$ <> "SE
GMENT" AND Befehl$ <> "TITLE" AND Befehl$ <
> "SUBTTL" AND Befehl$ <> "EVEN" AND Befehl
$ <> "ENDS" AND Befehl$ <> "END" AND Befehl
$ <> "ENDP" THEN
[WC2] 963: Objekt$ = Ziel$
[RZ2] 964: OperandAuswerten
[YW2] 965: ZielTyp = ObjTyp
[2E3] 966: ZielWert1 = ObjWert1
[5E3] 967: ZielWert2 = ObjWert2
[BA3] 968: IF Befehl$ = "RET" THEN
[7E6] 969: QuellTyp = Prozedurtyp(StackZeiger)
[ZR0] 970: ELSE
[8K2] 971: Objekt$ = Quell$
[RZ2] 972: OperandAuswerten
[UX2] 973: QuellTyp = ObjTyp
[PF3] 974: QuellWert1 = ObjWert1
[VF3] 975: QuellWert2 = ObjWert2
[801] 976: END IF
[801] 977: END IF
[9K2] 978: Objekt$ = Befehl$
[542] 979: BefehlSuchen
[UN2] 980: IF gefunden THEN
[NT3] 981: Flag = FlagTab(OperZeig)
[FY3] 982: IF Flag AND Operand THEN
[P62] 983: CodeErzeugen
[ZR0] 984: ELSE
[L33] 985: PseudoOperationen
[801] 986: END IF
[ZR0] 987: ELSE
[EV3] 988: IF Lauf = 1 THEN EXIT SUB
[U63] 989: FAnzahl = FAnzahl + 1
[LV5] 990: PRINT #2, "=> Fehler: Syntax-Fehler: ";
[2F2] 991: PRINT #2, Befehl$
[U5G] 992: IF ((Akku8 OR Akku16 OR Reg8 OR Reg16 O
R SegReg OR CS) AND (ZielTyp OR QuellTyp))
THEN GOTO BASchluss
[USC] 993: IF (QuellTyp AND (Nichts OR Direkt8 OR
Direkt16)) = falsch THEN GOTO BASchluss
[WS8] 994: IF INSTR("BW", RIGHT$(Befehl$, 1)) <> 0
THEN GOTO BASchluss
```



```
[UC3] 995: Warnung = Warnung + 1
[LA5] 996: PRINT #2, "=> Warnung: Achten Sie ";
[T46] 997: PRINT #2, "auf Wort- beziehungsweise ";
[9F4] 998: PRINT #2, "Byte-Operationen"
[801] 999: END IF
[5U1] 1000: BASchluss:
[HA1] 1001: END SUB
[000] 1002:
[556] 1003: 'Unterprogramm, das den angegebenen
[VN6] 1004: ' Assemblerbefehl in der Kommandotabelle
[X71] 1005: ' sucht
[KU2] 1006: SUB BefehlSuchen
[P13] 1007: Index = Befehlzahl
[NH2] 1008: Start = Index / 2
[NX2] 1009: DO WHILE Index >= 2
[PG2] 1010: Index = Index / 2
[575] 1011: IF Befehl$ > Kommando$(Start) THEN
[LB3] 1012: Start = Start + Index
[ZR0] 1013: ELSE
[9B3] 1014: Start = Start - Index
[801] 1015: END IF
[654] 1016: IF Start < 1 THEN Start = 1
[854] 1017: IF Start > Befehlzahl THEN
[N23] 1018: Start = Befehlzahl
[801] 1019: END IF
[HS0] 1020: LOOP
[PP2] 1021: gefunden = falsch
[DV5] 1022: FOR OperZeig = Start TO Befehlzahl
[WY5] 1023: IF Kommando$(OperZeig) > Befehl$ THEN
[AD1] 1024: EXIT FOR
[801] 1025: END IF
[VY5] 1026: IF Kommando$(OperZeig) = Befehl$ THEN
[TX8] 1027: IF (QuellRegTyp(OperZeig) AND QuellTyp
p) <> falsch THEN
[VL8] 1028: IF (ZielRegTyp(OperZeig) AND ZielTyp
p) <> falsch THEN
[904] 1029: gefunden = wahr: EXIT FOR
[801] 1030: END IF
[801] 1031: END IF
[801] 1032: END IF
[EB2] 1033: NEXT OperZeig
[HA1] 1034: END SUB
[000] 1035:
[2M5] 1036: 'Unterprogramm, das den direkten
[KP3] 1037: 'Bytezugriff beachtet
[LL2] 1038: SUB ByteDirekt
[MA4] 1039: IF ZielTyp AND Direkt8 THEN
[NV3] 1040: DirektWert = ZielWert1
[RB2] 1041: ByteSpeichern
[801] 1042: END IF
[XT4] 1043: IF QuellTyp AND Direkt8 THEN
[D64] 1044: DirektWert = QuellWert1
[RB2] 1045: ByteSpeichern
[801] 1046: END IF
[HA1] 1047: END SUB
[000] 1048:
[277] 1049: 'Unterprogramm, das das Erweiterungsbyte
[B91] 1050: 'bildet
[PN3] 1051: SUB ByteErweiterung
[YF1] 1052: Maske = 56
[T33] 1053: Ext = Flag AND Maske
[L44] 1054: IF Flag AND Richtung THEN
[2P2] 1055: ZielWert2 = Ext
[ZR0] 1056: ELSE
[2P2] 1057: QuellWert2 = Ext
[801] 1058: END IF
[A03] 1059: Adressierungsart
[HA1] 1060: END SUB
[000] 1061:
[RU5] 1062: SUB ByteSchluss (AktByte AS LONG)
[W74] 1063: LSET ZByte$ = CHR$(AktByte)
[MR1] 1064: D.Z = D.Z + 1
[NT1] 1065: PUT #2, D.Z
[TN2] 1066: IF HighByte THEN
[1V9] 1067: Checksumme = VAL("&H" + RIGHT$(HEX$(Che
cksumme + AktByte * 256), 4))
HighByte = falsch
[NT2] 1068: ELSE
[ZR0] 1069: Checksumme = VAL("&H" + RIGHT$(HEX$(Che
cksumme + AktByte), 4))
HighByte = wahr
[3L2] 1071: END IF
[801] 1072: END IF
[HA1] 1073: END SUB
[000] 1074:
[S54] 1075: 'Unterprogramm, das 8 Bit
[973] 1076: '(direkt) speichert
[423] 1077: SUB ByteSpeichern
[7W7] 1078: IF NOT (DirektWert <= 255 AND DirektWert
>= 0) THEN
[GA2] 1079: DirektWert = 0
[DC2] 1080: IF Lauf = 2 THEN
[U63] 1081: Fanzahl = Fanzahl + 1
[784] 1082: PRINT #2, "=> Fehler: Daten";
[HK4] 1083: PRINT #2, "anweisung zu lang"
[801] 1084: END IF
[801] 1085: END IF
[ZX4] 1086: Befehllaenge = Befehllaenge + 1
[Z25] 1087: BefehlTab(Befehllaenge) = DirektWert
[HA1] 1088: END SUB
```

```
[000] 1089:
[Y34] 1090: 'Unterprogramm, das den
[3H3] 1091: 'Programmcode aufbaut
[HF2] 1092: SUB CodeAufbau
[ZX4] 1093: Befehllaenge = Befehllaenge + 1
[MB7] 1094: BefehlTab(Befehllaenge) = OperWert(OperZe
ig)
[0M5] 1095: IF (Flag AND RegSum) <> falsch THEN
[865] 1096: IF ZielTyp AND (SegReg OR CS) THEN
[442] 1097: R = ZielWert2
[ZR0] 1098: ELSE
[5C4] 1099: IF (Flag AND Richtung) THEN
[HR2] 1100: R = QuellWert2 / 8
[ZR0] 1101: ELSE
[GH2] 1102: R = ZielWert2 / 8
[801] 1103: END IF
[801] 1104: END IF
[SD8] 1105: BefehlTab(Befehllaenge) = BefehlTab(Bef
ehllaenge) + R
[801] 1106: END IF
[BH5] 1107: IF (Flag AND Konst2) <> falsch THEN
[112] 1108: IF Wort THEN
[P88] 1109: BefehlTab(Befehllaenge) = BefehlTab(B
efehllaenge) + 1
[801] 1110: END IF
[801] 1111: END IF
[7H5] 1112: IF (Flag AND Konst1) <> falsch THEN
[EX3] 1113: IF QuellTyp AND CL THEN
[S88] 1114: BefehlTab(Befehllaenge) = BefehlTab(B
efehllaenge) + 2
[801] 1115: END IF
[801] 1116: END IF
[HA1] 1117: END SUB
[000] 1118:
[0M5] 1119: 'Unterprogramm, das die COM- oder
[S57] 1120: 'EXE-Informationen in einer Datei schreibt
[1X3] 1121: SUB CodeDateiSchreiben
[V62] 1122: Checksumme = 0
[NT2] 1123: HighByte = falsch
[Z25] 1124: OPEN "work.wrk" FOR RANDOM AS #3 LEN = 1
[DT2] 1125: FIELD #3, 1 AS Byte$
[B55] 1126: OPEN ZielDat$ FOR RANDOM AS #2 LEN = 1
[151] 1127: CLOSE #2
[852] 1128: KILL ZielDat$
[B55] 1129: OPEN ZielDat$ FOR RANDOM AS #2 LEN = 1
[AY2] 1130: FIELD #2, 1 AS ZByte$
[M11] 1131: D.Z = 0
[DC3] 1132: IF Modell = M.EXE THEN
[3R3] 1133: 'Exe-Kennung (Byte 00-01)
[852] 1134: ByteSchluss (77)
[KR2] 1135: ByteSchluss (90)
[Z44] 1136: 'Last Page Size (Byte 02-03)
[406] 1137: ByteSchluss ((PLaenge MOD 512) MOD 256)
[YB6] 1138: ByteSchluss (INT((PLaenge MOD 512) / 256))
[RK3] 1139: 'File Pages (Byte 04-05)
[801] 1140: ByteSchluss (INT(PLaenge / 512) + 1 + S
GN(PLaenge MOD 512) * 1)
[KM2] 1141: ByteSchluss (0)
[2P4] 1142: 'Relocation Items (Byte 06-07)
[CG5] 1143: ByteSchluss (ReloAnzahl MOD 256)
[EU5] 1144: ByteSchluss (INT(ReloAnzahl / 256))
[ZV4] 1145: 'Header Paragraphs (Byte 08-09)
[8R2] 1146: ByteSchluss (32)
[KM2] 1147: ByteSchluss (0)
[W93] 1148: 'MINALLOC (Byte 0A-0B)
[KM2] 1149: ByteSchluss (0)
[KM2] 1150: ByteSchluss (0)
[G93] 1151: 'MAXALLOC (Byte 0C-0D)
[FU2] 1152: ByteSchluss (255)
[FU2] 1153: ByteSchluss (255)
[BG5] 1154: 'Anfänglicher SS-Wert (Byte 0E-0F)
[KM2] 1155: ByteSchluss (0)
[KM2] 1156: ByteSchluss (0)
[3S3] 1157: 'SP-Register (Byte 10-11)
[2V4] 1158: ByteSchluss (Stapel MOD 256)
[455] 1159: ByteSchluss (INT(Stapel / 256))
[345] 1160: 'Jetzt die Check-Summe, die noch
[AZ4] 1161: 'nicht definiert ist (Byte 12-13)
[KM2] 1162: ByteSchluss (0)
[KM2] 1163: ByteSchluss (0)
[HR3] 1164: 'IP-Register (Byte 14-15)
[G75] 1165: ByteSchluss ((Code MOD 16) MOD 256)
[AL5] 1166: ByteSchluss (INT((Code MOD 16) / 256))
[HR3] 1167: 'CS-Register (Byte 16-17)
[7N5] 1168: ByteSchluss (INT(Code / 16) MOD 256)
[925] 1169: ByteSchluss (INT(INT(Code / 16) / 256))
[TP5] 1170: 'Relocation Table Offset (Byte 18-19)
[0R2] 1171: ByteSchluss (30)
[KM2] 1172: ByteSchluss (0)
[EK4] 1173: 'Overlay-Nummer (Byte 1A-1B)
[KM2] 1174: ByteSchluss (0)
[KM2] 1175: ByteSchluss (0)
[U15] 1176: 'Spezifische MASM-Erweiterung
[2N2] 1177: '(nicht ändern)
[NM2] 1178: ByteSchluss (1)
[KM2] 1179: ByteSchluss (0)
```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic ge-
schrieben.


```
[1B4] 1180: 'Relocation Pointer Table
[LF3] 1181: IF ReloAnzahl > 0 THEN
[1T4] 1182:   FOR Zaehler = 1 TO ReloAnzahl
[807] 1183:   ByteSchluss (ReloOffset(Zaehler) MOD 256)
[4A7] 1184:   ByteSchluss (INT(ReloOffset(Zaehler) / 25
6))
[896] 1185:   ByteSchluss (ReloSeg(Zaehler) MOD 256)
[2P6] 1186:   ByteSchluss (INT(ReloSeg(Zaehler) / 256))
[42] 1187:   NEXT Zaehler
[801] 1188:   END IF
[MZ5] 1189:   FOR Zaehler = 1 TO 482 - ReloAnzahl * 4
[8M2] 1190:   ByteSchluss (0)
[42] 1191:   NEXT Zaehler
[801] 1192:   END IF
[5S2] 1193:   'Programm-Modul
[064] 1194:   FOR Zaehler = 1 TO Plaenge
[YF2] 1195:   GET #3, Zaehler
[124] 1196:   ByteSchluss (ASC(Byte$))
[42] 1197:   NEXT Zaehler
[DC3] 1198:   IF Modell = M.EXE THEN
[8L3] 1199:   'Checksumme schreiben
[CD7] 1200:   Check$ = RIGHT$(HEX$(Checksumme XOR &HF
FFF), 4)
[S76] 1201:   Check$ = LEFT$("0000", 4 - LEN(Check$))
+ Check$
[EX6] 1202:   LSET ZByte$ = CHR$(VAL("&H" + RIGHT$(Ch
eck$, 2)))
[8E1] 1203:   PUT #2, 19
[NM6] 1204:   LSET ZByte$ = CHR$(VAL("&H" + LEFT$(Che
ck$, 2)))
[FD1] 1205:   PUT #2, 20
[801] 1206:   END IF
[3W0] 1207:   CLOSE
[HA1] 1208:   END SUB
[000] 1209:
[ZN6] 1210: 'Unterprogramm, das im zweiten Lauf den
[3N3] 1211: 'Programmcode erzeugt
[3X2] 1212: SUB CodeErzeugen
[FX2] 1213:   OperandByteWort
[3L3] 1214:   CodeZeig = CodeZeig + 1
[SC4] 1215:   IF Lauf = 2 THEN CodeAufbau
[LX9] 1216:   IF (OperWert(OperZeig) = 213) OR (OperWer
t(OperZeig) = 212) THEN
[3L3] 1217:   CodeZeig = CodeZeig + 1
[DC2] 1218:   IF Lauf = 2 THEN
[ZX4] 1219:   Befehllaenge = Befehllaenge + 1
[HB4] 1220:   BefehlTab(Befehllaenge) = 10
[801] 1221:   END IF
[801] 1222:   END IF
[TV4] 1223:   IF OffsetBedarf <> Nichts THEN
[5K5] 1224:   IF (OffsetBedarf AND Direkt8) THEN
[3L3] 1225:   CodeZeig = CodeZeig + 1
[ZR0] 1226:   ELSE
[7L3] 1227:   CodeZeig = CodeZeig + 2
[801] 1228:   END IF
[801] 1229:   END IF
[CE9] 1230:   IF (Flag AND (ModByteBedarf OR ErweiBedar
f)) <> falsch THEN
[ERA] 1231:   IF (ZielTyp OR QuellTyp) AND Speicher T
HEN CodeZeig = CodeZeig + 2
[801] 1232:   END IF
[KA6] 1233:   IF (Flag AND ErweiBedarf) <> falsch THEN
[3L3] 1234:   CodeZeig = CodeZeig + 1
[ZA5] 1235:   IF Lauf = 2 THEN ByteErweiterung
[801] 1236:   END IF
[SP6] 1237:   IF (Flag AND ModByteBedarf) <> falsch THEN
[3L3] 1238:   CodeZeig = CodeZeig + 1
[A03] 1239:   Adressierungsart
[8A5] 1240:   CodeZeig = CodeZeig + OffZusatz
[801] 1241:   END IF
[A86] 1242:   IF (Flag AND Disp8Bedarf) <> falsch THEN
[3L3] 1243:   CodeZeig = CodeZeig + 1
[2E5] 1244:   IF Lauf = 2 THEN DifferenzRelativ
[801] 1245:   END IF
[W66] 1246:   IF (Flag AND Disp16Bedarf) <> falsch THEN
[7L3] 1247:   CodeZeig = CodeZeig + 2
[VG3] 1248:   IF Lauf = 2 THEN Displ6
[801] 1249:   END IF
[ZM6] 1250:   IF (Flag AND Direkt8Bedarf) <> falsch THEN
[3L3] 1251:   CodeZeig = CodeZeig + 1
[KE4] 1252:   IF Lauf = 2 THEN ByteDirekt
[801] 1253:   END IF
[MT8] 1254:   IF NOT (Wort OR ((Flag AND Direkt16Bedarf
) = falsch)) THEN
[3L3] 1255:   CodeZeig = CodeZeig + 1
[KE4] 1256:   IF Lauf = 2 THEN ByteDirekt
[801] 1257:   END IF
[WU9] 1258:   IF NOT (NOT (Wort) OR ((Flag AND Direkt16
Bedarf) = falsch)) THEN
[864] 1259:   IF ZielTyp = Direkt16 THEN
[FL3] 1260:   CodeZeig = CodeZeig + 4
[ZR0] 1261:   ELSE
[7L3] 1262:   CodeZeig = CodeZeig + 2
[801] 1263:   END IF
[5G4] 1264:   IF Lauf = 2 THEN WortDirekt
[801] 1265:   END IF
[6Z6] 1266:   IF (Flag AND SpeicherBedarf) <> falsch THEN
[7L3] 1267:   CodeZeig = CodeZeig + 2
[DC2] 1268:   IF Lauf = 2 THEN
```

```
[SD2] 1269:   Adrl6BitBilden
[T55] 1270:   IF OperWert(OperZeig) = 183 THEN
[5F4] 1271:   ReloAnzahl = ReloAnzahl + 1
[ZV6] 1272:   ReloSeg(ReloAnzahl) = INT((CodeZeig
- 2) / 16)
[P37] 1273:   ReloOffset(ReloAnzahl) = (CodeZeig
- 2) MOD 16
[801] 1274:   END IF
[801] 1275:   END IF
[801] 1276:   END IF
[HA1] 1277:   END SUB
[000] 1278:
[WV2] 1279: SUB DefinitionDO
[TZ7] 1280:   Parameter$ = MID$(Zeile$, INSTR(Zeile$, B
efehl$) + 3)
[C25] 1281:   Parameter$ = LTRIM$(Parameter$)
[Y25] 1282:   Parameter$ = RTRIM$(Parameter$)
[E43] 1283:   FOR Zaehler = 1 TO 8
[HV2] 1284:   param$(Zaehler) = ""
[G42] 1285:   NEXT Zaehler
[PD1] 1286:   MNAME$ = ""
[H05] 1287:   IF INSTR(Parameter$, ",") = 0 THEN
[FZ2] 1288:   MNAME$ = Parameter$
[572] 1289:   Parameter$ = ""
[YR2] 1290:   GOTO m.parameter
[801] 1291:   END IF
[9S7] 1292:   MNAME$ = LEFT$(Parameter$, INSTR(Paramete
r$, ",") - 1)
[DM8] 1293:   Parameter$ = MID$(Parameter$, INSTR(Param
eter$, ",") + 1)
[Y32] 1294:   m.parameter:
[UP2] 1295:   IF MNAME$ = "" THEN
[GX0] 1296:   PRINT
[426] 1297:   PRINT "Beim Aufruf des Makros fehlt ";
[8G5] 1298:   PRINT "die Angabe des Makronamens."
[L23] 1299:   'GOTO macro.fehler
[801] 1300:   END IF
[UD3] 1301:   IF Parameter$ = "" THEN
[WT1] 1302:   Zaehler = 0
[EW2] 1303:   GOTO parameter.ok
[ZR0] 1304:   ELSE
[0U1] 1305:   Zaehler = 1
[801] 1306:   END IF
[892] 1307:   m.auswertung:
[H05] 1308:   IF INSTR(Parameter$, ",") = 0 THEN
[BG4] 1309:   param$(Zaehler) = Parameter$
[EW2] 1310:   GOTO parameter.ok
[801] 1311:   END IF
[HG9] 1312:   param$(Zaehler) = LEFT$(Parameter$, INSTR
(Parameter$, ",") - 1)
[DM8] 1313:   Parameter$ = MID$(Parameter$, INSTR(Param
eter$, ",") + 1)
[AW2] 1314:   IF Zaehler < 8 THEN
[E93] 1315:   Zaehler = Zaehler + 1
[E23] 1316:   GOTO m.auswertung
[801] 1317:   END IF
[GX0] 1318:   PRINT
[ZU5] 1319:   PRINT "Die Anzahl der uebergebenen ";
[RB6] 1320:   PRINT "Parameter ist zu hoch oder die ";
[ZE6] 1321:   PRINT "Form der Parameter ist falsch ";
[R76] 1322:   PRINT "beim Aufruf des Makros: "; MNAME$
[RE2] 1323:   GOTO do.fehler
[YW4] 1324:   'Form des Makroaufrufs stimmt
[L82] 1325:   parameter.ok:
[0T2] 1326:   IF Marke$ <> "" THEN
[WX2] 1327:   SymbolEintragen
[UD2] 1328:   PRINT #4, Marke$
[801] 1329:   END IF
[EF3] 1330:   IF MakroIndex = 1 THEN
[RE2] 1331:   GOTO do.fehler
[FM4] 1332:   'Fehler bei einem Do-Aufruf
[801] 1333:   END IF
[K22] 1334:   TempIndex = 1
[VA2] 1335:   oeffne.mdatei:
[KU4] 1336:   Werte = VAL(Makro$(TempIndex))
[UZ3] 1337:   TempIndex = TempIndex + 1
[XH4] 1338:   SPNAME$ = Makro$(TempIndex)
[UZ3] 1339:   TempIndex = TempIndex + 1
[TX5] 1340:   IF SPNAME$ = MNAME$ THEN GOTO makro.da
[BG2] 1341:   suche.schluss:
[W24] 1342:   Zeile$ = Makro$(TempIndex)
[UZ3] 1343:   TempIndex = TempIndex + 1
[HF4] 1344:   IF Zeile$ <> "MakroEnde" THEN
[P63] 1345:   GOTO suche.schluss
[801] 1346:   END IF
[Z65] 1347:   IF TempIndex <> MakroIndex THEN
[W33] 1348:   GOTO oeffne.mdatei
[801] 1349:   END IF
[U63] 1350:   FAnzahl = FAnzahl + 1
[ME1] 1351:   Zeile$ = ""
[EY5] 1352:   PRINT #2, "==" Fehler: Das aufgerufene ";
[CE5] 1353:   PRINT #2, "Makro "; MNAME$; " wurde ";
[N25] 1354:   PRINT #2, "noch nicht definiert."
[P54] 1355:   EXIT SUB 'GOTO m.schluss
[8E1] 1356:   makro.da:
[AV3] 1357:   IF Zaehler <> Werte THEN
[UC3] 1358:   Warnung = Warnung + 1
[TD5] 1359:   PRINT #2, "==" Warnung: Anzahl der ";
[WW5] 1360:   PRINT #2, "Parameter bei Definition ";
```



```
[2R4] 1361: PRINT #2, "und Aufruf stimmen ";
[EK4] 1362: PRINT #2, "nicht überein. ";
[KY4] 1363: PRINT #2, "Überzählige werden ";
[8E5] 1364: PRINT #2, "ignoriert, Zeilen mit ";
[AP4] 1365: PRINT #2, "nicht angegebenen ";
[VP6] 1366: PRINT #2, "Parametern gelöscht. Makro: ";
[801] 1367: END IF
[521] 1368: lies.makro:
[W24] 1369: Zeile$ = Makro$(TempIndex)
[U23] 1370: TempIndex = TempIndex + 1
[294] 1371: IF Zeile$ = "MakroEnde" THEN
[EM1] 1372: DZeile$ = ""
[TE1] 1373: EXIT SUB
[GE5] 1374: 'Ende und Zeile nicht abspeichern
[801] 1375: END IF
[XM4] 1376: 'Höchstens acht Parameter
[5A3] 1377: FOR ZAEHLER2 = 1 TO 8
[F76] 1378: 'Wenn Parameter nicht vorhanden ist,
[E06] 1379: 'dann weitermachen; Parameter durch
[HC6] 1380: 'angegebenen Wert ersetzen; falls kein
[286] 1381: 'Parameter angegeben, dann Zeile löschen
[LN8] 1382: DO WHILE INSTR(Zeile$, "%") + RIGHT$(STR
$(ZAEHLER2), 1)) <> 0
[RLP] 1383: Zeile$ = LEFT$(Zeile$, INSTR(Zeile$,
"%") + RIGHT$(STR$(ZAEHLER2), 1)) - 1) + par
am$(ZAEHLER2) + MID$(Zeile$, INSTR(Zeile$,
"%") + RIGHT$(STR$(ZAEHLER2), 1)) + 2)
[W74] 1384: IF param$(ZAEHLER2) = "" THEN
[ME1] 1385: Zeile$ = ""
[801] 1386: END IF
[HS0] 1387: LOOP
[M72] 1388: NEXT ZAEHLER2
[YS2] 1389: IF Zeile$ <> "" THEN
[3F2] 1390: PRINT #4, Zeile$
[RV1] 1391: hv$ = Zeile$
[HC4] 1392: IF INSTR(Zeile$, ";") <> 0 THEN
[DE6] 1393: Zeile$ = LEFT$(Zeile$, INSTR(Zeile$,
";") - 1)
[801] 1394: END IF
[F52] 1395: AnalyseZeile
[5H2] 1396: AnalyseSymbole
[UT1] 1397: Zeile$ = hv$
[801] 1398: END IF
[MS2] 1399: GOTO lies.makro
[TE1] 1400: EXIT SUB
[E20] 1401: '
[R25] 1402: 'Fehlerabfangroutine für 'DO'
[WM1] 1403: do.fehler:
[344] 1404: IF NOT dos THEN GOTO do.w1
[GX0] 1405: PRINT
[V26] 1406: PRINT "SAM 4 wurde durch einen Fehler ";
[4W5] 1407: PRINT "beim DO-Befehl unterbrochen."
[GX0] 1408: PRINT
[TE1] 1409: EXIT SUB
[E20] 1410: '
[MW0] 1411: do.w1:
[GX0] 1412: PRINT
[9G5] 1413: PRINT "Der DO-Befehl wurde falsch ";
[VG6] 1414: PRINT "benutzt. Sie müssen den Befehl ";
[683] 1415: PRINT "berichtigen."
[GX0] 1416: PRINT
[035] 1417: PRINT "Drücken Sie bitte eine ";
[0V3] 1418: PRINT "beliebige Taste !"
[MV3] 1419: DO WHILE INKEY$ = "": LOOP
[TE1] 1420: EXIT SUB
[HAL] 1421: END SUB
[000] 1422: '
[PY6] 1423: 'Unterprogramm, das die Differenz für die
[4W4] 1424: 'relative Adressierung bildet
[KL3] 1425: SUB DifferenzRelativ
[GU3] 1426: D = ZielWert1 - CodeZeig
[XV2] 1427: IF ABS(D) >= 128 THEN
[WK0] 1428: D = 0
[DC2] 1429: IF Lauf = 2 THEN
[Z45] 1430: PRINT #2, "=> Fehler: Distanz zu ";
[466] 1431: PRINT #2, "groß für einen 'short' ";
[MC2] 1432: PRINT #2, "jump'"
[U63] 1433: Fanzahl = Fanzahl + 1
[801] 1434: END IF
[801] 1435: END IF
[PS3] 1436: IF D < 0 THEN D = D AND 255
[ZK4] 1437: Befehllaenge = Befehllaenge + 1
[DC4] 1438: BefehlTab(Befehllaenge) = D
[HAL] 1439: END SUB
[000] 1440: '
[WW5] 1441: 'Routine, die das 16-Bit-Displacement
[YR1] 1442: 'errechnet
[GP1] 1443: SUB Displ16
[GU3] 1444: D = ZielWert1 - CodeZeig
[FW4] 1445: IF Befehl$ = "JMP" AND D <= 128 THEN
[M16] 1446: PRINT #2, "=> Warnung: Sie könnten ";
[Z34] 1447: PRINT #2, "'JMPs' benutzen"
[UC3] 1448: Warnung = Warnung + 1
[801] 1449: END IF
[5G5] 1450: IF NOT (D >= 0 OR Befehl$ = "CALL") THEN
[WK0] 1451: D = 0
[DC2] 1452: IF Lauf = 2 THEN
[XC4] 1453: PRINT #2, "=> Fehler: Nicht ";
```

```
[4M5] 1454: PRINT #2, "erlaubter Rückwärts";
[CR4] 1455: PRINT #2, "sprung ('long jump')";
[U63] 1456: Fanzahl = Fanzahl + 1
[801] 1457: END IF
[801] 1458: END IF
[351] 1459: Lsb = D
[402] 1460: Teilen16Bit
[3Y4] 1461: Befehllaenge = Befehllaenge + 2
[405] 1462: BefehlTab(Befehllaenge - 1) = Lsb
[8M4] 1463: BefehlTab(Befehllaenge) = Msb
[HAL] 1464: END SUB
[000] 1465: '
[0F3] 1466: SUB EingebenDbreite
[K75] 1467: mel$ = "Breite einer Druckseite "
[KM4] 1468: mel$ = mel$ + "eingeben! <ESC>: "
[B66] 1469: mel$ = mel$ + "Abbruch - <Return>: Fertig"
[M22] 1470: Status(mel$)
[AK2] 1471: Ausgabefenster
[XL1] 1472: LOCATE 8, 1
[EH5] 1473: PRINT "Breite in Zeichen (>54):"
[8U4] 1474: hv$ = MID$(STR$(Druckbreite), 2)
[521] 1475: D$ = hv$
[EX5] 1476: D$ = NInput$(D$, 3, 10, 1, "0123456789",
wahr)
[GV2] 1477: Standardfenster
[MN2] 1478: IF VAL(D$) < 54 THEN
[9W5] 1479: mel$ = "Die Breite einer Druckseite "
[446] 1480: mel$ = mel$ + "muß >= 54 Zeichen sein. "
[ZH6] 1481: mel$ = mel$ + "Bitte eine Taste drücken!"
[M22] 1482: Status(mel$)
[V11] 1483: WarteTaste
[521] 1484: D$ = hv$
[801] 1485: END IF
[9E3] 1486: Druckbreite = VAL(D$)
[D03] 1487: ZeigeDruckstatus
[7L2] 1488: StatusLoeschen
[HAL] 1489: END SUB
[000] 1490: '
[XC3] 1491: SUB EingebenDlaenge
[BA5] 1492: mel$ = "Länge einer Druckseite "
[7Z5] 1493: mel$ = mel$ + "eingeben! <ESC>: Abbruch "
[5Y4] 1494: mel$ = mel$ + " - <Return>: Fertig"
[M22] 1495: Status(mel$)
[AK2] 1496: Ausgabefenster
[XL1] 1497: LOCATE 8, 1
[V05] 1498: PRINT "Länge in Zeilen (>15):"
[RU4] 1499: hv$ = MID$(STR$(Seitenlaenge), 2)
[521] 1500: D$ = hv$
[EX5] 1501: D$ = NInput$(D$, 3, 10, 1, "0123456789",
wahr)
[GV2] 1502: Standardfenster
[CN2] 1503: IF VAL(D$) < 15 THEN
[Z25] 1504: mel$ = "Die Länge einer Druckseite "
[VU5] 1505: mel$ = mel$ + "muß >= 15 Zeilen sein. "
[ZH6] 1506: mel$ = mel$ + "Bitte eine Taste drücken!"
[M22] 1507: Status(mel$)
[V11] 1508: WarteTaste
[521] 1509: D$ = hv$
[801] 1510: END IF
[1C3] 1511: Seitenlaenge = VAL(D$)
[D03] 1512: ZeigeDruckstatus
[7L2] 1513: StatusLoeschen
[HAL] 1514: END SUB
[000] 1515: '
[1Z2] 1516: SUB EingebenMaske
[GV2] 1517: Standardfenster
[4E6] 1518: mel$ = "Neue Dateimaske inklusive Pfad "
[7Z5] 1519: mel$ = mel$ + "eingeben! <ESC>: Abbruch "
[5Y4] 1520: mel$ = mel$ + " - <Return>: Fertig"
[M22] 1521: Status(mel$)
[AK2] 1522: Ausgabefenster
[XL1] 1523: LOCATE 8, 1
[U76] 1524: PRINT "Dateimaske (maximal 79 Zeichen):"
[258] 1525: Dateimaske$ = NInput$(Dateimaske$, 79, 10
, 1, "", falsch)
[GV2] 1526: Standardfenster
[616] 1527: CALL ZeigeDatei(Dateimaske$, 21, 60, 19)
[7L2] 1528: StatusLoeschen
[HAL] 1529: END SUB
[000] 1530: '
[0H3] 1531: SUB EingebenQuelldat
[UK5] 1532: mel$ = "Quelldatei inklusive Pfad "
[7Z5] 1533: mel$ = mel$ + "eingeben! <ESC>: Abbruch "
[5Y4] 1534: mel$ = mel$ + " - <Return>: Fertig"
[M22] 1535: Status(mel$)
[AK2] 1536: Ausgabefenster
[XL1] 1537: LOCATE 8, 1
[M26] 1538: PRINT "SAM-Datei (maximal 79 Zeichen):"
[5D2] 1539: hv$ = QuellDat$
[SP8] 1540: QuellDat$ = UCASE$(NInput$(QuellDat$, 79,
10, 1, "", falsch))
[K94] 1541: IF (LEN(QuellDat$) < 76) THEN
[HW2] 1542: Pruef$ = QuellDat$
[B85] 1543: DO WHILE INSTR(Pruef$, CHR$(92)) <> 0
[YD7] 1544: Pruef$ = MID$(Pruef$, INSTR(Pruef$, C
HR$(92)) + 1)
```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic geschrieben.


```
[HS0] 1545: LOOP
[BF4] 1546: IF (INSTR(Pruef$, ".") = 0) THEN
[5G4] 1547: QuellDat$ = QuellDat$ + ".SAM"
[801] 1548: END IF
[801] 1549: END IF
[WT4] 1550: OPEN QuellDat$ FOR INPUT AS #1
[H82] 1551: IF Fehler THEN
[VC2] 1552: QuellDat$ = hv$
[VP2] 1553: GOTO Quell.Ende
[801] 1554: END IF
[H23] 1555: IF (QuellDat$ <> hv$) THEN
[203] 1556: IF ZielDat$ = "" THEN
[A93] 1557: ZielDat$ = QuellDat$
[5D2] 1558: hv$ = QuellDat$
[801] 1559: END IF
[184] 1560: p1 = INSTR(Umkehr$(hv$), ".")
[555] 1561: p2 = INSTR(Umkehr$(ZielDat$), ".")
[7P7] 1562: IF (LEN(hv$) > p1 - 1) AND (LEN(ZielDat$) > p2 - 1) THEN
[H7A] 1563: IF LEFT$(hv$, LEN(hv$) - p1) = LEFT$(ZielDat$, LEN(ZielDat$) - p2) THEN
[F36] 1564: Punkt = INSTR(Umkehr$(QuellDat$), ".")
[0T2] 1565: IF Punkt <> 0 THEN
[A68] 1566: ZielDat$ = LEFT$(QuellDat$, LEN(QuellDat$) - Punkt)
[DT3] 1567: IF LEN(ZielDat$) < 76 THEN
[DC3] 1568: IF Modell = M.EXE THEN
[P54] 1569: ZielDat$ = ZielDat$ + ".EXE"
[801] 1570: END IF
[CC3] 1571: IF Modell = M.COM THEN
[R54] 1572: ZielDat$ = ZielDat$ + ".COM"
[801] 1573: END IF
[801] 1574: END IF
[ZR0] 1575: ELSE
[BW1] 1576: ZielDat$ = ""
[801] 1577: END IF
[801] 1578: END IF
[801] 1579: END IF
[801] 1580: END IF
[GV2] 1581: Standardfenster
[4Y1] 1582: Quell.Ende:
[YH5] 1583: CALL ZeigeDatei(QuellDat$, 16, 14, 34)
[WF5] 1584: CALL ZeigeDatei(ZielDat$, 17, 14, 34)
[7L2] 1585: StatusLoeschen
[3W0] 1586: CLOSE
[NF2] 1587: Fehler = falsch
[HA1] 1588: END SUB
[000] 1589:
[DF3] 1590: SUB EingebenZielDat
[KE5] 1591: mel$ = "Zieldatei inklusive Pfad "
[7Z5] 1592: mel$ = mel$ + "eingegeben! <ESC>: Abbruch "
[5Y4] 1593: mel$ = mel$ + "- <Return>: Fertig"
[M22] 1594: Status(mel$)
[AK2] 1595: AusgabeFenster
[XL1] 1596: LOCATE 8, 1
[UK5] 1597: PRINT "Übersetzte Datei (maximal "
[F13] 1598: PRINT "79 Zeichen):"
[052] 1599: hv$ = ZielDat$
[2K8] 1600: ZielDat$ = UCASE$(NInput$(ZielDat$, 79, 1, 0, 1, "", falsch))
[T74] 1601: IF (LEN(ZielDat$) < 76) THEN
[ZM2] 1602: Pruef$ = ZielDat$
[BB5] 1603: DO WHILE INSTR(Pruef$, CHR$(92)) <> 0
[YD7] 1604: Pruef$ = MID$(Pruef$, INSTR(Pruef$, CHR$(92)) + 1)
[HS0] 1605: LOOP
[BF4] 1606: IF (INSTR(Pruef$, ".") = 0) THEN
[XG7] 1607: IF Modell = M.EXE THEN ZielDat$ = ZielDat$ + ".EXE"
[0H7] 1608: IF Modell = M.COM THEN ZielDat$ = ZielDat$ + ".COM"
[801] 1609: END IF
[801] 1610: END IF
[GV2] 1611: Standardfenster
[WF5] 1612: CALL ZeigeDatei(ZielDat$, 17, 14, 34)
[7L2] 1613: StatusLoeschen
[HA1] 1614: END SUB
[000] 1615:
[8V5] 1616: 'Beenden der Übersetzungsroutine
[XK3] 1617: SUB EndeAssemblieren
[DW3] 1618: IF StapelZeiger > 0 THEN
[U63] 1619: Fanzahl = Fanzahl + 1
[FV4] 1620: PRINT #2, "=> Fehler: Es fehlt ";
[Y22] 1621: PRINT #2, "ein 'ENDP'"
[801] 1622: END IF
[UL3] 1623: IF CodeDa = falsch THEN
[UC3] 1624: Warnung = Warnung + 1
[885] 1625: PRINT #2, "=> Warnung: Kein Code-";
[E65] 1626: PRINT #2, "Segment spezifiziert"
[801] 1627: END IF
[404] 1628: IF StapelDa = falsch THEN
[UC3] 1629: Warnung = Warnung + 1
[PT5] 1630: PRINT #2, "=> Warnung: Kein Stapel-";
[E65] 1631: PRINT #2, "Segment spezifiziert"
[801] 1632: END IF
[V33] 1633: IF Segmente = 1 THEN
[U63] 1634: Fanzahl = Fanzahl + 1
[B74] 1635: PRINT #2, "=> Fehler: Kein ";
```

```
[D05] 1636: PRINT #2, "Segmentende angegeben"
[801] 1637: END IF
[691] 1638: PRINT #2,
[CR6] 1639: PRINT #2, Fanzahl; "Fehler festgestellt,";
[CV6] 1640: PRINT Warnung; "Warnung(en); ausgegeben."
[3W2] 1641: AusgebenSymbole
[691] 1642: PRINT #2,
[CC3] 1643: IF Modell = M.COM THEN
[XS2] 1644: PRINT #2, PLaenge;
[C44] 1645: ELSEIF Modell = M.EXE THEN
[SC3] 1646: PRINT #2, PLaenge + 512;
[801] 1647: END IF
[S94] 1648: PRINT #2, "Bytes für die ";
[CC3] 1649: IF Modell = M.COM THEN
[T42] 1650: PRINT #2, "COM";
[C44] 1651: ELSEIF Modell = M.EXE THEN
[N42] 1652: PRINT #2, "EXE";
[801] 1653: END IF
[B24] 1654: PRINT #2, "-Datei erzeugt."
[KX4] 1655: IF AusgabeListe$ = "lpt1:" THEN
[6V3] 1656: PRINT #2, CHR$(27); CHR$(64)
[801] 1657: END IF
[3W0] 1658: CLOSE
[Z46] 1659: IF Fanzahl = 0 THEN CodeDateiSchreiben
[VU1] 1660: KILL "*.wrk"
[RN0] 1661: BEEP
[GV2] 1662: Standardfenster
[H86] 1663: mel$ = "SAM 4 hat seine Arbeit beendet. "
[125] 1664: mel$ = mel$ + "Bitte drücken Sie "
[KU3] 1665: mel$ = mel$ + "eine Taste!"
[M22] 1666: Status(mel$)
[1V1] 1667: WarteTaste
[7L2] 1668: StatusLoeschen
[EE2] 1669: IF dos THEN END
[HA1] 1670: END SUB
[000] 1671:
[H27] 1672: 'Zweiter Lauf (Erzeugen der ausführbaren
[U95] 1673: 'Datei und des Assemblerlistings)
[D23] 1674: SUB ErzeugenDatei
[DK1] 1675: InitLauf2
[Y22] 1676: DO WHILE NOT EOF(1)
[B72] 1677: LesenZeile (1)
[F52] 1678: AnalyseZeile
[F24] 1679: IF Befehl$ <> "SEGMENT" THEN
[Z05] 1680: IF Marke$ <> "" THEN Phasenfehler
[801] 1681: END IF
[W46] 1682: IF Befehl$ <> "" THEN BefehlAssemblieren
[614] 1683: IF Befehl$ = "SEGMENT" THEN
[Z05] 1684: IF Marke$ <> "" THEN Phasenfehler
[801] 1685: END IF
[F26] 1686: 'Ausgabe des Assemblercodes und -listings
[E42] 1687: AusgebenCode
[M23] 1688: AusgebenParameter
[1C2] 1689: Taste$ = INKEY$
[MX4] 1690: IF Taste$ = "" THEN GOTO weiter
[C34] 1691: DO WHILE INKEY$ <> "": LOOP
[MV3] 1692: DO WHILE INKEY$ = "": LOOP
[XA1] 1693: weiter:
[HS0] 1694: LOOP
[HA1] 1695: END SUB
[000] 1696:
[113] 1697: SUB Fehlermeldung
[GV2] 1698: Standardfenster
[7L2] 1699: StatusLoeschen
[VW1] 1700: FCode = ERR
[6N2] 1701: SELECT CASE ERR
[041] 1702: CASE 24
[A74] 1703: f$ = "Drucker nicht bereit"
[441] 1704: CASE 25
[E33] 1705: f$ = "Gerätefehler"
[C41] 1706: CASE 27
[YB3] 1707: f$ = "Kein Papier mehr"
[141] 1708: CASE 52
[MP3] 1709: f$ = "Falscher Dateiname"
[541] 1710: CASE 53
[FP5] 1711: f$ = "Datei oder Pfad nicht gefunden"
[N41] 1712: CASE 57
[PB5] 1713: f$ = "Gerätefehler (Ein-/Ausgabe)"
[041] 1714: CASE 61
[DS3] 1715: f$ = "Datenträger voll"
[C41] 1716: CASE 64
[626] 1717: f$ = "Fehlerhafter Datei- oder Pfadname"
[341] 1718: CASE 71
[EA4] 1719: f$ = "Laufwerk nicht bereit"
[741] 1720: CASE 72
[WMA] 1721: f$ = "Datenträger fehlerhaft"
[R41] 1722: CASE 76
[6T3] 1723: f$ = "Pfad nicht gefunden"
[PF1] 1724: CASE ELSE
[RU3] 1725: f$ = "Unbekannter Fehler"
[GS1] 1726: END SELECT
[3W0] 1727: CLOSE
[757] 1728: Status (f$ + " - Drücken Sie bitte eine Taste!")
[1V1] 1729: WarteTaste
```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic geschrieben.

DMV PROGRAMMIERER-TOOLS

Neo Graph

Die Turbo-Treiber-Toolbox für EGA/VGA-Grafikkarten

Turbo Pascal macht bei 640 x 480 Schluß.
Hier fängt NeoGraph an!

NeoGraph versetzt Sie in die Lage, die Möglichkeiten Ihrer Grafikkarte voll auszunutzen: höchste Auflösungen bei optimaler Farbwiedergabe. Die Programmsammlung NeoGraph stellt ein unverzichtbares Komplettpaket für die Programmierung von Grafikkartentreibern in (Turbo) Pascal dar, insbesondere, wenn es um höhere Auflösung als 640 x 480 geht.



- NeoGraph enthält einen Satz vorgefertigter Grafikkartentreiber (bis 1024 x 768 bei 256 Farben), zum Beispiel für Tseng, Genoa, Video7, u.a.
- Einige Prototypen von Treibern dienen dazu, auch kommende Formate auszunutzen, und um eigene Sonderformate und Programme anpassen zu können.
- Alle Routinen liegen im Quellcode bei.
- Die Programme sind ausgelegt, um auf DOS-Rechnern in Turbo Pascal ab der Version 4.0 eingebunden zu werden.
- Damit die erweiterten Grafikmodi genutzt werden können, gibt es zwei neue Units: NGRAPH und NCRT!
- Beide haben wesentliche Erweiterungen und Verbesserungen gegenüber den Originalen.
- Ausführliche Dokumentierung versetzt auch "Normalmenschen" in die Lage, die Programme professionell zu nutzen.
- Drei Testprogramme dienen der Erforschung der Video-Hardware.
- Ein sehr detailliertes Handbuch führt auch Einsteiger behutsam an die Programmierung heran.

Der Autor Arne Schäpers ist bekannt für seine hintergründige, aber plausible Art zu schreiben.

NeoGraph

DM 99,—*

Prof Ed

Der Editor für Programmierer:

- Bis zu fünf Dateien lassen sich gleichzeitig bearbeiten.
- Mehrere MByte lange Dateien sind kein Problem mehr.
- Jede Taste der Tastatur kann mit einem oder mehreren Kommandos versehen werden.
- Makros gestatten, beliebig viele Editorkommandos abzuabellen.
- Mehrere Undo-Funktionen lassen den letzten Arbeitsschritt rückgängig machen.
- Teile einer oder verschiedener Dateien lassen sich gleichzeitig am Bildschirm zur Bearbeitung bereitstellen.
- Suchen und Ersetzen von Zeichenfolgen ist problemlos.
- Zeilen- und spaltenweises Löschen, Kopieren, Verschieben und Füllen von markierten Textbereichen gestatten ein schnelles Bearbeiten Ihrer Dateien.
- Diese Funktion arbeitet dateiübergreifend und ermöglicht jede nur denkbare Textmanipulation.

ProfEd

DM 99,—*

MaskEdit PLUS 5.0

Professioneller Maskengenerator liefert Quellcodes Ihrer Wahl

- Statt Listings tippen, Menüpunkte auswählen.
- Schnelle Bildschirmmasken für Ihre Programme:
- hierarchische Menüs
- verschiedene Ausgabefenster
- 8 Rahmentypen, 16 Farben, Schattenwurf
- ASCII-, dBase-, Binär- und ANSI-Ausgabeformate als Standard
- optimaler Treiber für die Ausgabesprache Ihrer Wahl
- alle Treiber im Quelltext
- einfaches, aber effizientes Arbeiten mit über 50 Funktionen
- Komfort durch kontextbezogene Hilfen

Zusatztreiber-Auswahl:

Turbo Pascal 3/4/5/6, Turbo C und Microsoft C, Topspeed-Modula-2, Turbo Basic, Quick Basic

MaskEdit Plus, ein Zusatztreiber

DM 99,—*

MaskEdit Plus, zwei Zusatztreiber

DM 149,—*

MaskEdit Plus, drei Zusatztreiber

DM 199,—*

GraphBas

Die BGI-Anpassung und -Erweiterung für Basic

GraphBas PC erschließt Ihrem Basic-Compiler bisher ungeahnte Grafikwelten. Was Turbo Pascal und Turbo C berühmt gemacht hat, gibt es jetzt auch für Basic:

- Volle Nutzung der BGI-Vektorfonts
- Editor für eigene Schriften, Logos und Pictogramme
- 3D-Balken- und Tortendiagramme, kinderleicht aufgerufen
- Farben, Füllmuster und Figuren mit Assembler-routinen
- einfachste Handhabung durch Quellcodes-Einbindung
- Unterstützung aller Grafikkarten auf dem PC

GraphBas PC für Quick Basic

DM 99,—*

GraphBas PC für Turbo Basic

DM 99,—*

GraphBas PC für Power Basic

DM 99,—*

Jetzt neu für Power Basic!

DMV-PowerTools für Basic

Professionelle Tools für Basic-Programmierer zur Erstellung von Programmen und Oberflächen gemäß dem SAA-Standard.

Reizen Sie Basic voll aus! Steigen Sie ein in die Interrupt-Programmierung! Schreiben Sie sich Ihre eigene Arbeitsumgebung mit allen Raffinessen! Die DMV-PowerTools für Basic stellen die richtigen Module zur Verfügung, damit Sie unabhängig Ihre eigenen Routinen entwickeln können. Wie wäre es zum Beispiel mit einem Kopierprogramm?

Die PowerTools beinhalten über 200 Befehle und Funktionen, und zwar mit folgenden Schwerpunkten:

- MAUSTEXT/MAUSGRAF: Routinen zur Maussteuerung im Textmodus/Grafikmodus
- CONVERT: Konvertier-routinen, Funktionen zur Bitmanipulation, Funktionen zur Umrechnung von Zahlensystemen
- BIOS/2: BIOS-Funktionen (Grundfunktionen + Erweiterungen)
- DOS: DOS-Funktionen
- WINDOW: Fensterverwaltung
- STDIO1/2/3: Ein-/Ausgabe (u.a. Listenauswahl, Menüs, Zeilen-/Maskeneditor, Dialogroutinen)
- LASERJET: Anweisungen zur Steuerung eines HP-kompatiblen Laserdruckers
- MATH: mathematische Funktionen (Trigonometrie, Runden, Formatierung, Umrechnungsfunktionen)
- MAUS.MCD: vordefinierte Mauscursor für den Grafikmodus
- u.a.m.

Sämtliche Module liegen im jeweiligen Basic-Quelltext vor.

Sofort bestellen!

Denn jedes PowerToolspaket enthält
jetzt eine komplette Anwendung
(Musikbox) als kostenlose Zugabe
inklusive 30-seitigem Handbuch

(Oder fordern Sie für 10,— DM ein Software-Probchen mit fünf Demoprogrammen an)

DMV-PowerTools I für PowerBasic 2.10

DM 149,—*

DMV-PowerTools II für Quick Basic 4.5

DM 149,—*

DMV-PowerTools III für MS-Basic PDS 7.1

DM 149,—*

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung.
Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · W-3440 Eschwege

DMV
Daten- und
Medienverlag


```
[7L2] 1730: StatusLoeschen
[HA1] 1731: END SUB
[000] 1732:
[3C7] 1733: 'Unterprogramm, das die in der eingelesenen
[3P6] 1734: ' Zeile vorhandenen Kleinbuchstaben in
[1W4] 1735: ' Großbuchstaben umwandelt
[8M3] 1736: SUB Grossbuchstaben
[S94] 1737: IF INSTR(Zeile$, "'") = 0 THEN
[CF3] 1738: Zeile$ = UCASE$(Zeile$)
[ZR0] 1739: ELSE
[XL4] 1740: FOR Zaehler = 1 TO EndZeiger
[U54] 1741: b$ = MID$(Zeile$, Zaehler, 1)
[512] 1742: IF b$ = "'" THEN
[P96] 1743: ZeichEnde = INSTR(Zaehler + 1, Zeile$, b$)
[753] 1744: IF ZeichEnde > 0 THEN
[5B3] 1745: Zaehler = ZeichEnde
[HH2] 1746: GOTO kommentar
[801] 1747: END IF
[801] 1748: END IF
[1C5] 1749: MID$(Zeile$, Zaehler, 1) = UCASE$(b$)
[VU1] 1750: kommentar:
[G42] 1751: NEXT Zaehler
[801] 1752: END IF
[HA1] 1753: END SUB
[000] 1754:
[LY5] 1755: 'Hauptprogramm, das die benötigten
[LA4] 1756: 'Unterprogramme aufruft
[233] 1757: SUB Hauptprogramm
[AK2] 1758: Ausgabefenster
[U55] 1759: 'Initialisierung der Variablen
[TB2] 1760: InitAssembler
[ZG5] 1761: 'Lauf 1: Aufbau der Symboltabelle
[LC2] 1762: SymbolTabelle
[FM2] 1763: IF KeinEnde THEN
[3W0] 1764: CLOSE
[AT1] 1765: IF dos THEN
[4M0] 1766: END
[ZR0] 1767: ELSE
[TE1] 1768: EXIT SUB
[801] 1769: END IF
[801] 1770: END IF
[XN6] 1771: 'Lauf 2: Erzeugen der ausführbaren Datei
[RH4] 1772: 'und des Assemblerlistings
[0C2] 1773: ErzeugenDatei
[V94] 1774: 'Beenden der Übersetzung
[GW2] 1775: EndeAssemblieren
[GV2] 1776: Standardfenster
[HA1] 1777: END SUB
[000] 1778:
[346] 1779: 'Initialisierung der flexiblen Daten
[623] 1780: SUB InitAssembler
[H82] 1781: CountMakro = 0
[C72] 1782: CountMarke = 0
[882] 1783: ReloAnzahl = 0
[K72] 1784: MakroIndex = 1
[0G1] 1785: Title$ = ""
[1P1] 1786: SubTtl$ = ""
[M11] 1787: D.Z = 0
[ET1] 1788: Operand = 1
[5F1] 1789: Konst2 = 4
[2R1] 1790: RegSum = 64
[1P2] 1791: ErweiBedarf = 128
[0M2] 1792: Disp8Bedarf = 256
[DP2] 1793: Disp16Bedarf = 512
[R33] 1794: ModByteBedarf = 1024
[C23] 1795: Direkt8Bedarf = 2048
[M43] 1796: Direkt16Bedarf = 4096
[XB2] 1797: Richtung = 8192
[SB3] 1798: SpeicherBedarf = 16384
[062] 1799: Konst1 = &H8000
[A91] 1800: Akku8 = 1
[VB1] 1801: Akku16 = 2
[K31] 1802: Reg8 = 4
[G81] 1803: Reg16 = 8
[UH2] 1804: SpeicherReg = 16
[XY0] 1805: CS = 32
[HN1] 1806: SegReg = 64
[B32] 1807: Speicher = 128
[5V1] 1808: Direkt8 = 256
[012] 1809: Direkt16 = 512
[PUI] 1810: Nichts = 1024
[342] 1811: Zeichen = 2048
[DC1] 1812: Nah = 4096
[TL1] 1813: Weit = 8192
[H81] 1814: CL = 16384
[5M2] 1815: StapelZeiger = 0
[F31] 1816: Code = 0
[HF1] 1817: Stapel = 0
[EW1] 1818: Segmente = 0
[1E2] 1819: CodeDa = falsch
[RR2] 1820: StapelDa = falsch
[5T1] 1821: FAnzahl = 0
[HV1] 1822: Warnung = 0
[CS1] 1823: PLänge = 0
[Z96] 1824: 'Öffnen der Ein- und Ausgabedateien
[WT4] 1825: OPEN QuellDat$ FOR INPUT AS #1
[9T2] 1826: IF AssLis = 3 THEN
[XT8] 1827: AusgabeListe$ = LEFT$(ZielDat$, LEN(Zie
lDat$) - 3) + ".lst"
```

```
[LY2] 1828: GOTO flex.oeffnen
[ML3] 1829: 'Ausgabe in eine Datei
[801] 1830: END IF
[7T2] 1831: IF AssLis = 2 THEN
[CR3] 1832: AusgabeListe$ = "lpt1:"
[LY2] 1833: GOTO flex.oeffnen
[MR2] 1834: 'Druckerausgabe
[801] 1835: END IF
[5S3] 1836: AusgabeListe$ = "scrn:"
[B62] 1837: flex.oeffnen:
[NL5] 1838: OPEN AusgabeListe$ FOR OUTPUT AS #2
[0F3] 1839: PRINT : PRINT : PRINT
[ZZ5] 1840: OPEN "work.wrk" FOR RANDOM AS #3 LEN = 1
[DT2] 1841: FIELD #3, 1 AS Byte$
[BS1] 1842: LWidth = 79
[BS8] 1843: c$ = MID$(DATE$, 4, 2) + "." + LEFT$(DATE
$, 2) + "." + RIGHT$(DATE$, 4)
[5X2] 1844: IF AssLis <> 4 THEN
[A23] 1845: PRINT #2, QuellDat$;
[NX4] 1846: PRINT #2, TAB(LWidth - LEN(c$)); c$
[691] 1847: PRINT #2,
[691] 1848: PRINT #2,
[G25] 1849: PRINT #2, "Adr."; TAB(6); "Hex-Code";
[3W3] 1850: PRINT #2, TAB(18); "Zeile";
[4E4] 1851: PRINT #2, TAB(25); "Quelltext"
[691] 1852: PRINT #2,
[801] 1853: END IF
[HA1] 1854: END SUB
[000] 1855:
[E20] 1856: '
[LW2] 1857: 'Initialisierung
[E20] 1858: '
[1K2] 1859: SUB InitGesamt
[GV2] 1860: Standardfenster
[PNO] 1861: CLS
[UV2] 1862: 'Speichermodelle
[F71] 1863: M.COM = 1
[K71] 1864: M.EXE = 2
[W52] 1865: Modell = M.COM
[CC2] 1866: 'Ausgabemedien
[7S2] 1867: 'Laufzeitfehler
[ME1] 1868: Fehler = 0
[RZ2] 1869: Dateimaske$ = "*.??m"
[TC2] 1870: 'Standardfarbe
[7L1] 1871: COLOR 15, 0
[T82] 1872: 'View-Fenster
[L31] 1873: oben = 6
[RH1] 1874: unten = 14
[A26] 1875: 'Assemblerlisting auf den Bildschirm
[EHL] 1876: AssLis = 1
[2P4] 1877: IF ausgabe$ = "-" THEN AssLis = 4
[RB3] 1878: 'Keine Symboltabelle
[PK1] 1879: SymAus = 0
[FU4] 1880: 'Keine Cross-Referenztable
[5C1] 1881: Cross = 0
[ZT3] 1882: 'Datenzeilen pro Seite
[TN2] 1883: Seitenlaenge = 72
[K23] 1884: 'Zeichen pro Zeile
[AN2] 1885: Druckbreite = 80
[YT6] 1886: 'Für den HP-Laserjet sind die folgenden
[XE3] 1887: 'Werte von Interesse:
[NV2] 1888: 'Seitenlaenge = 64
[6M2] 1889: 'Druckbreite = 54
[PX2] 1890: wahr = -1: falsch = 0
[GZ2] 1891: Trenn$ = " " + CHR$(9)
[E20] 1892: '
[BU4] 1893: 'Definition der Datenfelder
[LY1] 1894: DIM param$(8)
[RM5] 1895: MaxBefehle = 1200: Stapelgroesse = 10
[P64] 1896: DIM BefehlTab(MaxBefehle)
[7H5] 1897: DIM Prozedurtyp(10), Prozedur$(10)
[E20] 1898: '
[MS4] 1899: 'Definitions des Makrofeldes
[382] 1900: DIM Makro$(300)
[E20] 1901: '
[ZZ5] 1902: 'Definition der Relocation-Entries
[AY4] 1903: DIM ReloOffset(20), ReloSeg(20)
[HA1] 1904: END SUB
[000] 1905:
[7Z5] 1906: 'Initialisierung des zweiten Laufs
[U82] 1907: SUB InitLauf2
[051] 1908: CLOSE #1
[V05] 1909: OPEN "work2.wrk" FOR INPUT AS #1
[Y41] 1910: Lauf = 2
[XU1] 1911: CodeZeig = 0
[WV1] 1912: ZeilNum = 0
[ED2] 1913: AusLaufnummer
[HA1] 1914: END SUB
[000] 1915:
[NF4] 1916: FUNCTION Invers$(Zkette$)
[9M1] 1917: COLOR 0, 15
[RE2] 1918: PRINT Zkette$;
[7L1] 1919: COLOR 15, 0
[R52] 1920: END FUNCTION
[000] 1921:
[E20] 1922: '
[LV6] 1923: 'Unterprogramm zum Einlesen einer Zeile
[E20] 1924: '
[0P3] 1925: SUB LesenZeile (Kanal)
```



```
[R34] 1926: LINE INPUT #Kanal, DZeile$
[DG2] 1927: Zeile$ = DZeile$
[HG4] 1928: IF INSTR(Zeile$, ";") <> 0 THEN
[DE6] 1929: Zeile$ = LEFT$(Zeile$, INSTR(Zeile$, ";") - 1)
[801] 1930: END IF
[MC3] 1931: ZeilNum = ZeilNum + 1
[1E3] 1932: OffsetBedarf = Nichts
[HG2] 1933: Befehllaenge = 0
[HA1] 1934: END SUB
[000] 1935:
[4P4] 1936: SUB Makrodefinition (Kanal)
[BR2] 1937: IF Marke$ = "" THEN
[GX0] 1938: PRINT
[0K5] 1939: PRINT "Das Makro ist nicht durch ";
[UP5] 1940: PRINT "einen Namen gekennzeichnet."
[KW2] 1941: GOTO macro.fehler
[801] 1942: END IF
[WZ7] 1943: Parameter$ = MID$(Zeile$, INSTR(Zeile$, B
efehl$) + 6)
[F65] 1944: DO WHILE INSTR(Parameter$, " ") <> 0
[5BF] 1945: Parameter$ = LEFT$(Parameter$, INSTR(Pa
rameter$, " ") - 1) + MID$(Parameter$, INST
R(Parameter$, " ") + 1)
[HS0] 1946: LOOP
[R33] 1947: Werte = 0: Werte$ = ""
[262] 1948: param.falsch:
[XH2] 1949: IF Werte = 9 THEN
[GX0] 1950: PRINT
[RY4] 1951: PRINT "Zu viele oder falsche ";
[S46] 1952: PRINT "Parameter bei der Definition ";
[494] 1953: PRINT "des Makros: "; Marke$
[KW2] 1954: GOTO macro.fehler
[801] 1955: END IF
[9E6] 1956: IF Parameter$ = Werte$ THEN GOTO mdef.anf
[MH2] 1957: IF Werte = 0 THEN
[TB1] 1958: Werte = 1
[VL1] 1959: Werte$ = "%1"
[CW2] 1960: GOTO param.falsch
[801] 1961: END IF
[8K2] 1962: Werte = Werte + 1
[FP6] 1963: Werte$ = Werte$ + "," + RIGHT$(STR$(Wert
e), 1)
[CW2] 1964: GOTO param.falsch
[HE1] 1965: mdef.anf:
[Y45] 1966: Makro$(MakroIndex) = STR$(Werte)
[5M4] 1967: Makro$(MakroIndex + 1) = Marke$
[DF4] 1968: MakroIndex = MakroIndex + 2
[LR1] 1969: enm.fehl:
[YR2] 1970: IF EOF(Kanal) THEN
[GX0] 1971: PRINT
[LD5] 1972: PRINT "Der Befehl 'ENDM' fehlt bei ";
[X04] 1973: PRINT "dem Makro: "; Marke$
[KW2] 1974: GOTO macro.fehler
[801] 1975: END IF
[LZ2] 1976: LesenZeile (Kanal)
[F52] 1977: AnalyseZeile
[0D3] 1978: IF Befehl$ = "ENDM" THEN
[B25] 1979: Makro$(MakroIndex) = "MakroEnde"
[9F4] 1980: MakroIndex = MakroIndex + 1
[BC2] 1981: GOTO def.param
[801] 1982: END IF
[CF4] 1983: Makro$(MakroIndex) = DZeile$
[9F4] 1984: MakroIndex = MakroIndex + 1
[CE2] 1985: GOTO enm.fehl
[HP1] 1986: def.param:
[EM1] 1987: DZeile$ = ""
[TE1] 1988: EXIT SUB
[E20] 1989: '
[YR5] 1990: 'Fehlerabfangeroutine für 'MACRO'
[E42] 1991: macro.fehler:
[ZA4] 1992: IF NOT dos THEN GOTO mac.w1
[GX0] 1993: PRINT
[V26] 1994: PRINT "SAM 4 wurde durch einen Fehler ";
[6D6] 1995: PRINT "beim MACRO-Befehl unterbrochen."
[GX0] 1996: PRINT
[TE1] 1997: EXIT SUB
[E20] 1998: '
[061] 1999: mac.w1:
[GX0] 2000: PRINT
[1N6] 2001: PRINT "Sie haben den MACRO-Befehl falsch ";
[KK6] 2002: PRINT "benutzt und müssen Ihre Eingabe ";
[683] 2003: PRINT "berichtigen."
[GX0] 2004: PRINT
[035] 2005: PRINT "Drücken Sie bitte eine ";
[0V3] 2006: PRINT "beliebige Taste !"
[MV3] 2007: DO WHILE INKEY$ = "": LOOP
[WK1] 2008: 'GOTO men2
[HA1] 2009: END SUB
[000] 2010:
[DK1] 2011: SUB Menue
[RX4] 2012: PRINT CHR$(218); STRING$(78, 196);
[992] 2013: PRINT CHR$(191);
[D45] 2014: PRINT CHR$(179); " "; STRING$(5, 223);
[534] 2015: PRINT " "; STRING$(5, 223); " ";
[AV3] 2016: PRINT STRING$(2, 223); " ";
[285] 2017: PRINT STRING$(2, 223); " "; CHR$(223);
[CF4] 2018: PRINT " "; CHR$(223); SPACE$(9);
```

```
[C86] 2019: PRINT "DOS international/DMV-Verlag ";
[CT4] 2020: PRINT "GmbH & Co. KG "; CHR$(179);
[D45] 2021: PRINT CHR$(179); " "; STRING$(5, 223);
[534] 2022: PRINT " "; STRING$(5, 223); " ";
[GG4] 2023: PRINT CHR$(223); " "; CHR$(223); " ";
[K75] 2024: PRINT CHR$(223); " "; STRING$(5, 223);
[VZ5] 2025: PRINT SPACE$(15); "Copyright 1989,1990,";
[1A5] 2026: PRINT "1991,1992"; SPACE$(8); CHR$(179);
[D45] 2027: PRINT CHR$(179); " "; STRING$(5, 223);
[0W4] 2028: PRINT " "; CHR$(223); " "; CHR$(223);
[0W4] 2029: PRINT " "; CHR$(223); " "; CHR$(223);
[CZ4] 2030: PRINT " "; CHR$(223); SPACE$(12);
[AN6] 2031: PRINT "Autor: Markus Zietlow - Version ";
[SR4] 2032: PRINT "4.0"; SPACE$(5); CHR$(179);
[MX4] 2033: PRINT CHR$(192); STRING$(78, 196);
[H72] 2034: PRINT CHR$(217)
[F43] 2035: FOR Zaehler = 1 TO 9
[GX0] 2036: PRINT
[G42] 2037: NEXT Zaehler
[AX4] 2038: PRINT CHR$(218); STRING$(47, 196);
[EW4] 2039: PRINT CHR$(194); STRING$(30, 196);
[992] 2040: PRINT CHR$(191);
[9T4] 2041: PRINT CHR$(179); " "; Invers$("S");
[1B4] 2042: PRINT "AM-Datei: "; SPACE$(36);
[G06] 2043: PRINT CHR$(179); " Quelltextausgabe: ";
[VZ4] 2044: PRINT Invers$("B"); "ildschirm ";
[X92] 2045: PRINT CHR$(179);
[GT4] 2046: PRINT CHR$(179); " "; Invers$("Z");
[845] 2047: PRINT "ieldatei: "; STRING$(36, 32);
[G06] 2048: PRINT CHR$(179); " Quelltextausgabe: ";
[5U4] 2049: PRINT Invers$("D"); "rucker ";
[X92] 2050: PRINT CHR$(179);
[2X4] 2051: PRINT CHR$(195); STRING$(25, 196);
[FW4] 2052: PRINT CHR$(194); STRING$(21, 196);
[MB5] 2053: PRINT CHR$(180); " Dru"; Invers$("c");
[ZZ5] 2054: PRINT "kbreite: "; SPACE$(5); "Zeichen";
[604] 2055: PRINT SPACE$(3); CHR$(179);
[V05] 2056: PRINT CHR$(179); SPACE$(4); "-Mod";
[0R3] 2057: PRINT Invers$("e"); "ll";
[E04] 2058: PRINT SPACE$(14); CHR$(179);
[HG5] 2059: PRINT "Assemblerlauf: "; SPACE$(6);
[453] 2060: PRINT CHR$(179); " Druc";
[H66] 2061: PRINT Invers$("k"); "länge: "; SPACE$(6);
[F85] 2062: PRINT "Zeilen"; SPACE$(4); CHR$(179);
[R54] 2063: PRINT CHR$(179); " "; Invers$("A");
[4W4] 2064: PRINT "ssemblieren"; SPACE$(12);
[9A5] 2065: PRINT CHR$(179); " Programmzeile: ";
[PC4] 2066: PRINT SPACE$(6); CHR$(179); " ";
[RX5] 2067: PRINT Invers$("V"); "erzeichnis aus";
[XV4] 2068: PRINT "geben"; SPACE$(9); CHR$(179);
[NF5] 2069: PRINT CHR$(179); SPACE$(25); CHR$(179);
[BA5] 2070: PRINT "Sprungmarken: "; SPACE$(7);
[HX5] 2071: PRINT CHR$(179); SPACE$(3); Invers$("M");
[H35] 2072: PRINT "aske: "; SPACE$(21); CHR$(179);
[2T4] 2073: PRINT CHR$(179); " "; Invers$("L");
[DN5] 2074: PRINT "isting: "; SPACE$(16); CHR$(179);
[195] 2075: PRINT "Makroaufrufe: "; SPACE$(7);
[164] 2076: PRINT CHR$(179); " Zurück ";
[4E6] 2077: PRINT Invers$("i"); "ns Betriebssystem";
[904] 2078: PRINT SPACE$(4); CHR$(179);
[2X4] 2079: PRINT CHR$(195); STRING$(25, 196);
[CW4] 2080: PRINT CHR$(193); STRING$(21, 196);
[BW4] 2081: PRINT CHR$(193); STRING$(30, 196);
[492] 2082: PRINT CHR$(180);
[8G5] 2083: PRINT CHR$(179); SPACE$(78); CHR$(179);
[MX4] 2084: PRINT CHR$(192); STRING$(78, 196);
[C92] 2085: PRINT CHR$(217);
[HA1] 2086: END SUB
[000] 2087:
[VN8] 2088: FUNCTION NInput$(Zkette$, Laenge, y, x, So
nder$, neu)
[286] 2089: IF (neu <> wahr) AND (neu <> falsch) THEN
[T55] 2090: PRINT "Falscher Parameter 'neu'"
[ET0] 2091: STOP
[801] 2092: END IF
[6T1] 2093: IF neu THEN
[6Y0] 2094: hv$ = ""
[ZR0] 2095: ELSE
[K12] 2096: hv$ = Zkette$
[801] 2097: END IF
[NG0] 2098: m1:
[UT1] 2099: LOCATE y, x
[U22] 2100: PRINT hv$; CHR$(219);
[SC0] 2101: m2:
[WM2] 2102: Taste$ = INPUT$(1)
[3R3] 2103: IF Taste$ = CHR$(13) THEN
[UT1] 2104: LOCATE y, x
[VU2] 2105: PRINT hv$; CHR$(32);
[DZ1] 2106: NInput$ = hv$
[NF4] 2107: ELSEIF Taste$ = CHR$(27) THEN
[UT1] 2108: LOCATE y, x
[RE2] 2109: PRINT Zkette$;
[246] 2110: PRINT SPACE$(Laenge - LEN(Zkette$) + 1);
[5U2] 2111: NInput$ = Zkette$
[BB4] 2112: ELSEIF Taste$ = CHR$(8) THEN
[652] 2113: IF LEN(hv$) > 0 THEN
```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic ge-
schrieben.


```
[C54] 2114: hv$ = LEFT$(hv$, LEN(hv$) - 1)
[801] 2115: END IF
[UT1] 2116: LOCATE y, x
[RF3] 2117: PRINT hv$; CHR$(219); " ";
[981] 2118: GOTO m2
[0F4] 2119: ELSEIF Taste$ < CHR$(32) THEN
[981] 2120: GOTO m2
[ZR0] 2121: ELSE
[3X2] 2122: IF Sonder$ <> "" THEN
[FD5] 2123: IF INSTR(Sonder$, Taste$) = 0 THEN
[CF1] 2124: Taste$ = ""
[801] 2125: END IF
[801] 2126: END IF
[3P3] 2127: IF LEN(hv$) < Laenge THEN
[7G2] 2128: hv$ = hv$ + Taste$
[801] 2129: END IF
[581] 2130: GOTO m1
[801] 2131: END IF
[R52] 2132: END FUNCTION
[000] 2133:
[FR5] 2134: 'Unterprogramm, das den Operanden
[RU1] 2135: 'auswertet
[4P3] 2136: SUB OperandAuswerten
[8E3] 2137: IF LEN(Objekt$) = 0 THEN
[8N2] 2138: Objekt$ = Nichts
[TE1] 2139: EXIT SUB
[801] 2140: END IF
[A82] 2141: SymbolSuchen
[UN2] 2142: IF gefunden THEN
[8Y4] 2143: Objekt$ = Symboltyp(TabZeiger)
[8K4] 2144: ObjWert1 = Wert1(TabZeiger)
[EF4] 2145: IF TabZeiger <= VorDef THEN
[TK4] 2146: ObjWert2 = Wert2(TabZeiger)
[801] 2147: END IF
[TE1] 2148: EXIT SUB
[801] 2149: END IF
[HH2] 2150: OperandPruefen
[UN2] 2151: IF gefunden THEN
[1T2] 2152: Objekt$ = Zahltyp
[6B3] 2153: ObjWert1 = ZahlWert
[TE1] 2154: EXIT SUB
[801] 2155: END IF
[E62] 2156: Speichertest
[UN2] 2157: IF gefunden THEN
[1V2] 2158: Objekt$ = Speicher
[AR3] 2159: ObjWert1 = Speicheradr
[TE1] 2160: EXIT SUB
[801] 2161: END IF
[PA2] 2162: OperandOffset
[UN2] 2163: IF gefunden THEN
[KD3] 2164: Objekt$ = SpeicherReg
[D23] 2165: ObjWert1 = RegWert
[TE1] 2166: EXIT SUB
[801] 2167: END IF
[6V1] 2168: OpisOffset
[UN2] 2169: IF gefunden THEN
[543] 2170: Objekt$ = OffsetTyp
[MK3] 2171: ObjWert1 = OffsetWert
[TE1] 2172: EXIT SUB
[801] 2173: END IF
[RF2] 2174: OperandZeichen
[UN2] 2175: IF gefunden THEN
[5P4] 2176: Objekt$ = Direkt8 OR Direkt16
[7C3] 2177: ObjWert1 = ZeichWert
[TE1] 2178: EXIT SUB
[801] 2179: END IF
[BE4] 2180: IF LEFT$(Objekt$, 1) = "'" THEN
[AN2] 2181: Objekt$ = Zeichen
[TE1] 2182: EXIT SUB
[801] 2183: END IF
[DC2] 2184: IF Lauf = 2 THEN
[CF4] 2185: PRINT #2, "=> Fehler: Symbol ";
[GR5] 2186: PRINT #2, Objekt$; " nicht definiert."
[U63] 2187: FAnzahl = FAnzahl + 1
[801] 2188: END IF
[104] 2189: Objekt$ = Nah OR Speicher
[HA1] 2190: END SUB
[000] 2191:
[HW6] 2192: 'Unterprogramme, das zwischen Wort- und
[635] 2193: 'Byte-Operanden unterscheidet
[WL3] 2194: SUB OperandByteWort
[7X1] 2195: Wort = wahr
[V55] 2196: IF OperWert(OperZeig) = 246 THEN
[X64] 2197: IF QuellTyp AND Reg8 THEN
[T32] 2198: Wort = falsch
[TE1] 2199: EXIT SUB
[801] 2200: END IF
[801] 2201: END IF
[B9C] 2202: IF (ZielTyp OR QuellTyp) AND (Reg16 OR Ak
kul6 OR SegReg OR CS) THEN EXIT SUB
[VV7] 2203: IF (ZielTyp OR QuellTyp) AND (Reg8 OR Akk
u8) THEN
[T32] 2204: Wort = falsch
[TE1] 2205: EXIT SUB
[801] 2206: END IF
[WD4] 2207: IF RIGHT$(Befehl$, 1) = "B" THEN
[T32] 2208: Wort = falsch
[TE1] 2209: EXIT SUB
```

```
[801] 2210: END IF
[HA1] 2211: END SUB
[000] 2212:
[5A6] 2213: 'Unterprogramm, das den Operanden als
[236] 2214: 'Offset zu einem Register behandelt
[313] 2215: SUB OperandOffset
[AD2] 2216: Temp$ = Objekt$
[UK3] 2217: IF Objekt$ = "[BP]" THEN
[8W1] 2218: RegWert = 6
[TL3] 2219: OffsetBedarf = Direkt8
[EF1] 2220: Offset = 0
[2U2] 2221: GOTO ein.symbol
[801] 2222: END IF
[984] 2223: Zeiger = INSTR(Objekt$, "[")
[6T5] 2224: IF Zeiger <= 1 THEN GOTO null.symbol
[KV6] 2225: Regis$ = RIGHT$(Temp$, LEN(Temp$) - Zeig
er + 1)
[0G3] 2226: IF Regis$ = "[BP]" THEN
[8W1] 2227: RegWert = 6
[ZR0] 2228: ELSE
[LK2] 2229: Objekt$ = Regis$
[A82] 2230: SymbolSuchen
[MRC] 2231: IF NOT gefunden OR Symboltyp(TabZeiger)
<> SpeicherReg THEN GOTO null.symbol
RegWert = Wert1(TabZeiger)
[FC4] 2232:
[801] 2233: END IF
[0U4] 2234: Disp$ = LEFT$(Temp$, Zeiger - 1)
[CB2] 2235: Objekt$ = Disp$
[A82] 2236: SymbolSuchen
[526] 2237: IF NOT gefunden THEN GOTO kein.symbol
[D8D] 2238: IF (Symboltyp(TabZeiger) AND (Direkt16 OR
Direkt8)) = falsch THEN GOTO kein.symbol
[D16] 2239: OffsetBedarf = Symboltyp(TabZeiger)
[C04] 2240: Offset = Wert1(TabZeiger)
[2U2] 2241: GOTO ein.symbol
[S52] 2242: kein.symbol:
[HH2] 2243: OperandPruefen
[UN2] 2244: IF gefunden THEN
[SP3] 2245: OffsetBedarf = Zahltyp
[3V2] 2246: Offset = ZahlWert
[2U2] 2247: GOTO ein.symbol
[801] 2248: END IF
[6V1] 2249: OpisOffset
[136] 2250: IF NOT gefunden THEN GOTO null.symbol
[M44] 2251: OffsetBedarf = OffsetTyp
[NC3] 2252: Offset = OffsetWert
[W22] 2253: ein.symbol:
[WC2] 2254: Objekt$ = Temp$
[4G2] 2255: gefunden = wahr
[TE1] 2256: EXIT SUB
[S72] 2257: null.symbol:
[WC2] 2258: Objekt$ = Temp$
[PP2] 2259: gefunden = falsch
[HA1] 2260: END SUB
[000] 2261:
[BL6] 2262: 'Unterprogramm, das überprüft, ob der
[PM3] 2263: 'Operand eine Zahl ist
[Y73] 2264: SUB OperandPruefen
[PP2] 2265: gefunden = falsch
[5Y1] 2266: tn$ = Objekt$
[SU3] 2267: IF RIGHT$(tn$, 1) = "H" THEN
[G54] 2268: tn$ = LEFT$(tn$, LEN(tn$) - 1)
[2K3] 2269: hv$ = HEX$(VAL("&H" + tn$))
[CN3] 2270: DO WHILE LEFT$(tn$, 1) = "0"
[S92] 2271: tn$ = MID$(tn$, 2)
[HS0] 2272: LOOP
[TW3] 2273: IF LEN(hv$) > LEN(tn$) THEN
[H34] 2274: hv$ = RIGHT$(hv$, LEN(tn$))
[801] 2275: END IF
[KR8] 2276: IF tn$ <> (STRING$(LEN(tn$) - LEN(hv$),
"0") + hv$) THEN EXIT SUB
[MP3] 2277: ZahlWert = VAL("&H" + tn$)
[X22] 2278: GOTO zahl.ok
[801] 2279: END IF
[DV3] 2280: IF RIGHT$(tn$, 1) = "O" THEN
[G54] 2281: tn$ = LEFT$(tn$, LEN(tn$) - 1)
[9L3] 2282: hv$ = OCT$(VAL("&O" + tn$))
[KR8] 2283: IF tn$ <> (STRING$(LEN(tn$) - LEN(hv$),
"0") + hv$) THEN EXIT SUB
[9R3] 2284: ZahlWert = VAL("&O" + tn$)
[X22] 2285: GOTO zahl.ok
[801] 2286: END IF
[6U3] 2287: IF RIGHT$(tn$, 1) = "B" THEN
[G54] 2288: tn$ = LEFT$(tn$, LEN(tn$) - 1)
[863] 2289: FOR i = 1 TO LEN(tn$)
[ME2] 2290: b$ = MID$(tn$, i, 1)
[925] 2291: IF INSTR("01", b$) = 0 THEN EXIT SUB
[U11] 2292: NEXT i
[3X1] 2293: ZahlWert = 0
[WB4] 2294: FOR i = LEN(tn$) TO 1 STEP -1
[UT8] 2295: ZahlWert = ZahlWert + VAL(MID$(tn$, i
, 1)) * 2 ^ (LEN(tn$) - i)
[U11] 2296: NEXT i
[X22] 2297: GOTO zahl.ok
[801] 2298: END IF
[ZD5] 2299: IF INSTR("+-", LEFT$(tn$, 1)) <> 0 THEN
[S92] 2300: tn$ = MID$(tn$, 2)
[801] 2301: END IF
```



```

[2Y3] 2302: hv$ = MID$(STR$(VAL(tn$)), 2)
[KR8] 2303: IF tn$ <> (STRING$(LEN(tn$) - LEN(hv$), "
0") + hv$) THEN EXIT SUB
[RM3] 2304: ZahlWert = VAL(Objekt$)
[KC1] 2305: zahl.ok:
[4G2] 2306: gefunden = wahr
[ZM4] 2307: IF LEN(HEX$(ZahlWert)) < 3 THEN
[FN4] 2308: Zahltyp = Direkt16 OR Direkt8
[ZR0] 2309: ELSE
[013] 2310: Zahltyp = Direkt16
[801] 2311: END IF
[HA1] 2312: END SUB
[000] 2313:
[5A6] 2314: 'Unterprogramm, das den Operanden als
[433] 2315: 'Zeichen definiert
[463] 2316: SUB OperandZeichen
[PP2] 2317: gefunden = falsch
[BZ4] 2318: IF LEN(Objekt$) <> 3 THEN EXIT SUB
[Z26] 2319: IF LEFT$(Objekt$, 1) <> "'" THEN EXIT SUB
[DY5] 2320: IF RIGHT$(Objekt$, 1) <> "'" THEN EXIT SUB
[4G2] 2321: gefunden = wahr
[LG5] 2322: ZeichWert = ASC(MID$(Objekt$, 2, 1))
[HA1] 2323: END SUB
[000] 2324:
[5A6] 2325: 'Unterprogramm, das den Operanden als
[E63] 2326: 'Offset deklariert
[LH2] 2327: SUB OpisOffset
[AD2] 2328: Temp$ = Objekt$
[U35] 2329: IF LEFT$(Temp$, 7) <> "OFFSET(" THEN
[PP2] 2330: gefunden = falsch
[TE1] 2331: EXIT SUB
[801] 2332: END IF
[194] 2333: IF Lauf = 1 THEN GOTO arg.ok
[H36] 2334: Objekt$ = MID$(Objekt$, 8, LEN(Objekt$) -
8)
[A82] 2335: Symbolsuchen
[AHH] 2336: IF gefunden AND (Symboltyp(TabZeiger) AND
(Speicher OR Nah)) THEN OffsetWert = Wert1
(TabZeiger): GOTO arg.ok
[U63] 2337: Fanzahl = Fanzahl + 1
[UD5] 2338: PRINT #2, "=> Fehler: Unerlaubtes ";
[396] 2339: PRINT #2, "oder undefiniertes Offset-";
[8H3] 2340: PRINT #2, "Argument "; ""
[F92] 2341: OffsetWert = 0
[991] 2342: arg.ok:
[4G2] 2343: gefunden = wahr
[UC3] 2344: OffsetTyp = Direkt16
[WC2] 2345: Objekt$ = Temp$
[HA1] 2346: END SUB
[000] 2347:
[Y27] 2348: 'Unterprogramm, das die Symboltabelle auf
[E64] 2349: ' Phasenfehler untersucht
[7U2] 2350: SUB Phasenfehler
[BGC] 2351: IF Befehl$ = "EQU" OR Befehl$ = "ENDS" OR
Befehl$ = "ENDP" THEN GOTO phasen.schluss
[ZG2] 2352: Objekt$ = Marke$
[A82] 2353: Symbolsuchen
[K9D] 2354: IF (Symboltyp(TabZeiger) AND (Nah OR Spei
cher)) = falsch THEN GOTO phasen.schluss
[BS5] 2355: IF Wert1(TabZeiger) = CodeZeig THEN
[KG3] 2356: GOTO phasen.schluss
[801] 2357: END IF
[U63] 2358: Fanzahl = Fanzahl + 1
[XD5] 2359: PRINT #2, "=> Fehler: Phasenfehler"
[SN2] 2360: phasen.schluss:
[HA1] 2361: END SUB
[000] 2362:
[8F3] 2363: 'Pseudobefehl ASSUME
[TX2] 2364: SUB PseudoASSUME
[HA1] 2365: END SUB
[000] 2366:
[CU2] 2367: 'Pseudobefehl 'DB'
[P42] 2368: SUB PseudoDB
[AG2] 2369: IF Lauf <> 2 THEN
[0T2] 2370: IF Marke$ <> "" THEN
[WC5] 2371: Symboltyp(AktSymbol) = Speicher
[801] 2372: END IF
[801] 2373: END IF
[FD3] 2374: ZeilZeig1 = OpndZeig
[HD3] 2375: ZeilZeig2 = OpndZeig
[GX4] 2376: DO WHILE ZeilZeig1 < EndZeiger
[NH2] 2377: ZeileAufteilen
[1K4] 2378: IF NOT gefunden THEN EXIT DO
[Y92] 2379: Objekt$ = feld$
[HH2] 2380: OperandPruefen
[DH8] 2381: IF gefunden AND (Zahltyp AND Direkt8) <
> falsch THEN
[ZX4] 2382: Befehllaenge = Befehllaenge + 1
[VP5] 2383: BefehlTab(Befehllaenge) = ZahlWert
[8H3] 2384: GOTO Operand.bekannt
[801] 2385: END IF
[1W3] 2386: IF LEFT$(feld$, 1) = "'" THEN
[F45] 2387: feld$ = MID$(feld$, 2, LEN(feld$) - 2)
[8F3] 2388: FOR i = 1 TO LEN(feld$)
[ZX4] 2389: Befehllaenge = Befehllaenge + 1
[L37] 2390: BefehlTab(Befehllaenge) = ASC(MID$(
feld$, i, 1))
[U11] 2391: NEXT i
[8H3] 2392: GOTO Operand.bekannt
[801] 2393: END IF
[DC2] 2394: IF Lauf = 2 THEN
[MC5] 2395: PRINT #2, "=> Fehler: Unbekannter ";
[6W3] 2396: PRINT #2, "Operand "; feld$
[U63] 2397: Fanzahl = Fanzahl + 1
[801] 2398: END IF
[EV2] 2399: Operand.bekannt:
[HS0] 2400: LOOP
[DE5] 2401: CodeZeig = CodeZeig + Befehllaenge
[HA1] 2402: END SUB
[000] 2403:
[EV2] 2404: 'Pseudobefehl 'DS'
[752] 2405: SUB PseudoDS
[AG2] 2406: IF Lauf <> 2 THEN
[0T2] 2407: IF Marke$ <> "" THEN
[WC5] 2408: Symboltyp(AktSymbol) = Speicher
[801] 2409: END IF
[801] 2410: END IF
[WC2] 2411: Objekt$ = Ziel$
[HH2] 2412: OperandPruefen
[PCC] 2413: IF (gefunden AND (Zahltyp AND (Direkt8 OR
Direkt16))) = falsch THEN GOTO DSEnde
[8K2] 2414: Objekt$ = Quell$
[HH2] 2415: OperandPruefen
[HC2] 2416: IF (gefunden AND (Zahltyp AND (Direkt8 OR
Direkt16))) = falsch THEN GOTO DSEnde
[E83] 2417: IF ZielWert1 < 0 THEN
[XH3] 2418: FOR i = 1 TO ZielWert1
[SG7] 2419: IF Zahltyp AND Direkt8 AND (Befehl$ <
> "DSW") THEN
[ZX4] 2420: Befehllaenge = Befehllaenge + 1
[C06] 2421: BefehlTab(Befehllaenge) = QuellWert1
[UK8] 2422: ELSEIF Zahltyp AND Direkt16 AND (Befe
hl$ <> "DSB") THEN
[TB3] 2423: hv$ = HEX$(QuellWert1)
[T35] 2424: hv$ = LEFT$( "0000", 4 - LEN(hv$)) + hv$
[ZX4] 2425: Befehllaenge = Befehllaenge + 1
[BK7] 2426: BefehlTab(Befehllaenge) = VAL("&H"
+ RIGHT$(hv$, 2))
[ZX4] 2427: Befehllaenge = Befehllaenge + 1
[UA7] 2428: BefehlTab(Befehllaenge) = VAL("&H"
+ LEFT$(hv$, 2))
[801] 2429: END IF
[U11] 2430: NEXT i
[801] 2431: END IF
[291] 2432: DSEnde:
[DE5] 2433: CodeZeig = CodeZeig + Befehllaenge
[HA1] 2434: END SUB
[000] 2435:
[B52] 2436: SUB PseudoDW
[AG2] 2437: IF Lauf <> 2 THEN
[0T2] 2438: IF Marke$ <> "" THEN
[WC5] 2439: Symboltyp(AktSymbol) = Speicher
[801] 2440: END IF
[801] 2441: END IF
[FD3] 2442: ZeilZeig1 = OpndZeig
[HD3] 2443: ZeilZeig2 = OpndZeig
[GX4] 2444: DO WHILE ZeilZeig1 < EndZeiger
[NH2] 2445: ZeileAufteilen
[1K4] 2446: IF NOT gefunden THEN EXIT DO
[Y92] 2447: Objekt$ = feld$
[HH2] 2448: OperandPruefen
[UN2] 2449: IF gefunden THEN
[9E6] 2450: IF Zahltyp AND (Direkt8 OR Direkt16) THEN
[H33] 2451: hv$ = HEX$(ZahlWert)
[T35] 2452: hv$ = LEFT$( "0000", 4 - LEN(hv$)) + hv$
[ZX4] 2453: Befehllaenge = Befehllaenge + 1
[BK7] 2454: BefehlTab(Befehllaenge) = VAL("&H"
+ RIGHT$(hv$, 2))
[ZX4] 2455: Befehllaenge = Befehllaenge + 1
[UA7] 2456: BefehlTab(Befehllaenge) = VAL("&H"
+ LEFT$(hv$, 2))
[PW2] 2457: GOTO OpDwbekannt
[801] 2458: END IF
[801] 2459: END IF
[DC2] 2460: IF Lauf = 2 THEN
[MC5] 2461: PRINT #2, "=> Fehler: Unbekannter ";
[Y33] 2462: PRINT #2, "Operand "; feld$;
[U63] 2463: Fanzahl = Fanzahl + 1
[801] 2464: END IF
[562] 2465: OpDwbekannt:
[HS0] 2466: LOOP
[DE5] 2467: CodeZeig = CodeZeig + Befehllaenge
[HA1] 2468: END SUB
[000] 2469:
[ZK5] 2470: 'Pseudobefehl 'END' (für spätere
[8G4] 2471: 'Erweiterungen vorgesehen)
[E92] 2472: SUB PseudoEND
[HA1] 2473: END SUB
[000] 2474:
[393] 2475: 'Pseudobefehl 'ENDP'
[YG2] 2476: SUB PseudoENDP
[B04] 2477: IF StapelZeiger <= 0 THEN
[9G2] 2478: IF Lauf <> 1 THEN

```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic ge-
schrieben.


```
[U63] 2479: FAnzahl = FAnzahl + 1
[1N5] 2480: PRINT #2, "=> Fehler: ENDP ohne PROC"
[801] 2481: END IF
[TE1] 2482: EXIT SUB
[801] 2483: END IF
[0N6] 2484: IF Marke$ <> Prozedur$(StapelZeiger) THEN
[0C3] 2485: PRINT #2, "Prozedur ";
[NK5] 2486: PRINT #2, Prozedur$(StapelZeiger);
[YU5] 2487: PRINT #2, " wurde nicht geschlossen"
[TE1] 2488: EXIT SUB
[801] 2489: END IF
[Y65] 2490: StapelZeiger = StapelZeiger - 1
[HA1] 2491: END SUB
[000] 2492:
[P93] 2493: 'Pseudobefehl 'ENDS'
[7H2] 2494: SUB PseudoENDS
[9G2] 2495: IF Lauf <> 1 THEN
[G04] 2496: IF Marke$ <> SegName$ THEN
[8C3] 2497: FAnzahl = FAnzahl + 1:
[1Y5] 2498: PRINT #2, "=> Fehler: Das Segmentende ";
[875] 2499: PRINT #2, "hat nicht den gleichen ";
[E46] 2500: PRINT #2, "Namen wie der Segmentanfang"
[801] 2501: END IF
[R33] 2502: IF Segmente = 0 THEN
[U63] 2503: FAnzahl = FAnzahl + 1
[9Z4] 2504: PRINT #2, "=> Fehler: Falsches ";
[CX3] 2505: PRINT #2, "Segmentende "; ""
[ZR0] 2506: ELSE
[EW1] 2507: Segmente = 0
[801] 2508: END IF
[801] 2509: END IF
[HA1] 2510: END SUB
[000] 2511:
[H43] 2512: 'Pseudobefehl 'EQU'
[LA2] 2513: SUB PseudoEQU
[BR2] 2514: IF Marke$ = "" THEN
[DC2] 2515: IF Lauf = 2 THEN
[U63] 2516: FAnzahl = FAnzahl + 1
[BX3] 2517: PRINT #2, "=> Fehler: Die ";
[RK5] 2518: PRINT #2, "Symbolzuweisung fehlt ";
[UX4] 2519: PRINT #2, "bei der EQU-Anweisung"
[801] 2520: END IF
[TE1] 2521: EXIT SUB
[801] 2522: END IF
[AG2] 2523: IF Lauf <> 2 THEN
[TH5] 2524: IF ZielTyp = (Nah OR Speicher) THEN
[U63] 2525: FAnzahl = FAnzahl + 1
[CZ3] 2526: PRINT #2, "=> Fehler: EQU ";
[M46] 2527: PRINT #2, "betrifft in Zeile"; ZeilNum;
[N84] 2528: PRINT #2, "eine noch nicht ";
[A45] 2529: PRINT #2, "definierte Zuweisung"
[TE1] 2530: EXIT SUB
[801] 2531: END IF
[SS4] 2532: Wert1(AktSymbol) = ZielWert1
[D95] 2533: Symboltyp(AktSymbol) = ZielTyp
[801] 2534: END IF
[HA1] 2535: END SUB
[000] 2536:
[E93] 2537: 'Pseudobefehl 'EVEN'
[2H2] 2538: SUB PseudoEVEN
[116] 2539: IF CodeZeig / 2 <> INT(CodeZeig / 2) THEN
[3L3] 2540: CodeZeig = CodeZeig + 1
[ZX4] 2541: Befehllaenge = Befehllaenge + 1
[XE4] 2542: BefehlTab(Befehllaenge) = 144
[801] 2543: END IF
[HA1] 2544: END SUB
[000] 2545:
[VC6] 2546: 'Unterprogramm, das die Pseudobefehle
[5P1] 2547: 'behandelt
[0U3] 2548: SUB PseudoOperationen
[565] 2549: SELECT CASE OperWert(OperZeig)
[DX0] 2550: CASE 1
[6M1] 2551: PseudoEQU
[GX0] 2552: CASE 2
[LM1] 2553: PseudoORG
[LX0] 2554: CASE 3
[9E1] 2555: PseudoDB
[PX0] 2556: CASE 4
[UE1] 2557: PseudoDS
[TX0] 2558: CASE 5
[GV1] 2559: PseudoPROC
[WX0] 2560: CASE 6
[HU1] 2561: PseudoENDP
[ZX0] 2562: CASE 7
[G52] 2563: PseudoTITLE
[2Y0] 2564: CASE 8
[NU1] 2565: PseudoEVEN
[5Y0] 2566: CASE 9
[1L1] 2567: PseudoEND
[D31] 2568: CASE 10
[7C2] 2569: PseudoSEGMENT
[H31] 2570: CASE 11
[UU1] 2571: PseudoENDS
[N31] 2572: CASE 12
[YE1] 2573: PseudoDW
[T31] 2574: CASE 13
[C72] 2575: PseudoASSUME
[GS1] 2576: END SELECT
[HA1] 2577: END SUB
```

```
[000] 2578:
[L33] 2579: 'Pseudobefehl 'ORG'
[0B2] 2580: SUB PseudoORG
[GB3] 2581: CodeZeig = ZielWert1
[HA1] 2582: END SUB
[000] 2583:
[393] 2584: 'Pseudobefehl 'PROC'
[XH2] 2585: SUB PseudoPROC
[U96] 2586: IF StapelZeiger >= Stapelgroesse THEN
[9G2] 2587: IF Lauf <> 1 THEN
[U63] 2588: FAnzahl = FAnzahl + 1
[FA5] 2589: PRINT #2, "=> Fehler: Prozeduren ";
[Z25] 2590: PRINT #2, "sind zu tief verschachtelt"
[801] 2591: END IF
[TE1] 2592: EXIT SUB
[801] 2593: END IF
[U65] 2594: StapelZeiger = StapelZeiger + 1
[9E6] 2595: Prozedurtyp(StapelZeiger) = ZielTyp
[SC5] 2596: Prozedur$(StapelZeiger) = Marke$
[HA1] 2597: END SUB
[000] 2598:
[EV3] 2599: 'Pseudobefehl 'SEGMENT'
[M23] 2600: SUB PseudoSEGMENT
[N54] 2601: IF CodeZeig MOD 16 <> 0 THEN
[EW4] 2602: 'Segmentanpassung durch NOP
[U26] 2603: FOR Zaehler = 1 TO 16 - (CodeZeig MOD 16)
[ZK4] 2604: Befehllaenge = Befehllaenge + 1
[X94] 2605: BefehlTab(Befehllaenge) = 0
[G42] 2606: NEXT Zaehler
[DE5] 2607: CodeZeig = CodeZeig + Befehllaenge
[801] 2608: END IF
[9G2] 2609: IF Lauf <> 1 THEN
[NL2] 2610: SegName$ = Marke$
[V33] 2611: IF Segmente = 1 THEN
[U63] 2612: FAnzahl = FAnzahl + 1
[V15] 2613: PRINT "=> Fehler: Unzulässiger ";
[RY3] 2614: PRINT "Segmentanfang "; ""
[ZM3] 2615: GOTO segment.schluss
[ZR0] 2616: ELSE
[FW1] 2617: Segmente = 1
[801] 2618: END IF
[MM4] 2619: IF UCASE$(Ziel$) <> "PARAM" THEN
[UC3] 2620: Warnung = Warnung + 1
[N55] 2621: PRINT #2, "=> Warnung: PARAM bei ";
[KB6] 2622: PRINT #2, "Segmentzuweisung vergessen"
[801] 2623: END IF
[UU4] 2624: IF UCASE$(Quell$) = "CODE" THEN
[VE2] 2625: Code = CodeZeig
[592] 2626: CodeDa = wahr
[UW5] 2627: ELSEIF UCASE$(Quell$) = "STACK" THEN
[DT2] 2628: Stapel = CodeZeig
[5H2] 2629: StapelDa = wahr
[3L5] 2630: ELSEIF UCASE$(Quell$) = "DATA" THEN
[CW2] 2631: REM Datensegment
[ZR0] 2632: ELSE
[8A3] 2633: FAnzahl = FAnzahl + 1
[955] 2634: PRINT #2, "=> Fehler: Unbekannte ";
[D15] 2635: PRINT #2, "Segmentzuweisung "; ""
[801] 2636: END IF
[801] 2637: END IF
[MY2] 2638: segment.schluss:
[HA1] 2639: END SUB
[000] 2640:
[425] 2641: 'Pseudobefehle 'TITEL' und 'SUBTITLE'
[XV2] 2642: SUB PseudoTITLE
[XP3] 2643: IF Befehl$ = "TITLE" THEN
[R87] 2644: Title$ = MID$(Zeile$, INSTR(Zeile$, Befehl$) + 6)
[YU4] 2645: Title$ = LTRIM$(RTRIM$(Title$))
[801] 2646: END IF
[KX3] 2647: IF Befehl$ = "SUBTTL" THEN
[3C7] 2648: SubTtl$ = MID$(Zeile$, INSTR(Zeile$, Befehl$) + 7)
[395] 2649: SubTtl$ = LTRIM$(RTRIM$(SubTtl$))
[801] 2650: END IF
[HA1] 2651: END SUB
[000] 2652:
[B12] 2653: SUB SAMende
[PNO] 2654: CLS
[GX0] 2655: PRINT
[TY5] 2656: PRINT "SAM 4 hat seine Arbeit beendet."
[GX0] 2657: PRINT
[4M0] 2658: END
[HA1] 2659: END SUB
[000] 2660:
[2M5] 2661: 'Unterprogramm, das den direkten
[ZR4] 2662: 'Speicherzugriff auswertet
[VW2] 2663: SUB Speichertest
[PP2] 2664: gefunden = falsch
[AD2] 2665: Temp$ = Objekt$
[6B9] 2666: IF LEFT$(Temp$, 1) <> "[" OR RIGHT$(Temp$, 1) <> "]" THEN EXIT SUB
[1M5] 2667: Objekt$ = MID$(Temp$, 2, LEN(Temp$) - 2)
[HH2] 2668: OperandPruefen
[UN2] 2669: IF gefunden THEN
[5X3] 2670: Speicheradr = ZahlWert
[1H4] 2671: GOTO Speichertest.Schluss
[801] 2672: END IF
[A82] 2673: SymbolSuchen
```



```

[293] 2674: IF NOT gefunden THEN
[1H4] 2675: GOTO Speichertest.Schluss
[801] 2676: END IF
[8K8] 2677: IF (Symboltyp(TabZeiger) AND Direkt16) =
falsch THEN
[PP2] 2678: gefunden = falsch
[1H4] 2679: GOTO Speichertest.Schluss
[801] 2680: END IF
[615] 2681: Speicheradr = Wert1(TabZeiger)
[1T3] 2682: Speichertest.Schluss:
[WC2] 2683: Objekt$ = Temp$
[HA1] 2684: END SUB
[000] 2685:
[XH3] 2686: SUB Standardfenster
[RW2] 2687: VIEW PRINT 1 TO 25
[HA1] 2688: END SUB
[000] 2689:
[Z93] 2690: SUB Status (Zkette$)
[TM1] 2691: LOCATE 24, 2
[866] 2692: Freizeichen = INT((78 - LEN(Zkette$)) / 2)
[3V5] 2693: PRINT SPACE$(Freizeichen); Zkette$;
[G17] 2694: PRINT SPACE$(78 - Freizeichen - LEN(Zkette
$))
[HA1] 2695: END SUB
[000] 2696:
[M93] 2697: SUB StatusLoeschen
[TM1] 2698: LOCATE 24, 2
[3N2] 2699: PRINT SPACE$(78);
[HA1] 2700: END SUB
[000] 2701:
[707] 2702: 'Unterprogramm, das ein bisher noch nicht
[LD7] 2703: 'definiertes Symbol in die Tabelle einträgt
[9M3] 2704: SUB SymbolEintragen
[ZG2] 2705: Objekt$ = Marke$
[A82] 2706: SymbolSuchen
[UN2] 2707: IF gefunden THEN
[U63] 2708: FAnzahl = FAnzahl + 1
[6L5] 2709: PRINT #2, "=> Fehler: Label "; Marke$;
[VX4] 2710: PRINT #2, " in "; ZeilNum; "zum ";
[A76] 2711: PRINT #2, "wiederholten Mal definiert."
[ZR0] 2712: ELSE
[3C5] 2713: IF AktSymbol >= Symbolzahl THEN
[U63] 2714: FAnzahl = FAnzahl + 1
[AW4] 2715: PRINT #2, "=> Fehler: Zu viele ";
[LZ4] 2716: PRINT #2, "benutzerdefinierte ";
[PR5] 2717: PRINT #2, "Symbole in Zeile"; ZeilNum
[ZR0] 2718: ELSE
[Y04] 2719: AktSymbol = AktSymbol + 1
[TC4] 2720: Symbol$(AktSymbol) = Marke$
[HL4] 2721: Wert1(AktSymbol) = CodeZeig
[MF4] 2722: Symboltyp(AktSymbol) = Nah
[PF4] 2723: CountMarke = CountMarke + 1
[YV2] 2724: AusMarkenanzahl
[801] 2725: END IF
[801] 2726: END IF
[HA1] 2727: END SUB
[000] 2728:
[347] 2729: 'Unterprogramm, das die angegebene Adresse
[2W4] 2730: 'in der Symboltabelle sucht
[RY2] 2731: SUB SymbolSuchen
[PP2] 2732: gefunden = falsch
[T05] 2733: FOR TabZeiger = 1 TO AktSymbol
[WM5] 2734: IF Symbol$(TabZeiger) = Objekt$ THEN
[4G2] 2735: gefunden = wahr
[AD1] 2736: EXIT FOR
[801] 2737: END IF
[FG2] 2738: NEXT TabZeiger
[HA1] 2739: END SUB
[000] 2740:
[SK6] 2741: 'Erster Lauf (Aufbau der Symboltabelle,
[WK6] 2742: 'verarbeitet INCLUDE-Dateien und fügt
[9K5] 2743: 'eventuell vorhandene MAKROS ein)
[033] 2744: SUB SymbolTabelle
[SD5] 2745: OPEN "work2.wrk" FOR OUTPUT AS #4
[343] 2746: AktSymbol = VorDef
[XU1] 2747: CodeZeig = 0
[GG2] 2748: KeinEnde = wahr
[X41] 2749: Lauf = 1
[ED2] 2750: AusLaufnummer
[XM2] 2751: AusMakroanzahl
[YV2] 2752: AusMarkenanzahl
[WV1] 2753: ZeilNum = 0
[YZ2] 2754: DO WHILE NOT EOF(1)
[B22] 2755: LesenZeile (1)
[F52] 2756: AnalyseZeile
[B16] 2757: 'Falls INCLUDE-Anweisungen vorhanden
[X56] 2758: 'sind, werden sie, so weit als möglich,
[X42] 2759: 'ausgeführt
[P04] 2760: IF Befehl$ <> "INCLUDE" THEN
[DT2] 2761: GOTO macrobefehl
[801] 2762: END IF
[P22] 2763: Datei$ = Ziel$
[R94] 2764: OPEN Datei$ FOR INPUT AS #5
[E25] 2765: IF EOF(5) THEN GOTO fehl.include
[A03] 2766: DO WHILE NOT EOF(5)
[L72] 2767: LesenZeile (5)
[M23] 2768: AusgebenParameter
[F52] 2769: AnalyseZeile
[1R3] 2770: IF Befehl$ <> "MACRO" THEN
[H83] 2771: GOTO INCLUDEweiter

```

```

[801] 2772: END IF
[X73] 2773: Makrodefinition (5)
[1G2] 2774: INCLUDEweiter:
[GW2] 2775: IF DZeile$ <> "" THEN
[5H2] 2776: AnalyseSymbole
[MG2] 2777: PRINT #4, DZeile$
[801] 2778: END IF
[HS0] 2779: LOOP
[451] 2780: CLOSE #5
[ME1] 2781: Zeile$ = ""
[EE2] 2782: GOTO m.schluss
[U46] 2783: 'Falls DO-Anweisungen vorhanden sind,
[PL5] 2784: 'werden sie so weit als möglich
[X42] 2785: 'ausgeführt
[522] 2786: macrobefehl:
[1R3] 2787: IF Befehl$ <> "MACRO" THEN
[MD2] 2788: GOTO do.befehl
[801] 2789: END IF
[H73] 2790: Makrodefinition (1)
[882] 2791: 'Weitermachen
[EE2] 2792: GOTO m.schluss
[DL1] 2793: do.befehl:
[3N5] 2794: IF Befehl$ <> "DO" THEN GOTO m.schluss
[F52] 2795: DefinitionDO
[AK4] 2796: CountMakro = CountMakro + 1
[XM2] 2797: AusMakroanzahl
[2T1] 2798: m.schluss:
[GW2] 2799: IF DZeile$ <> "" THEN
[5H2] 2800: AnalyseSymbole
[MG2] 2801: PRINT #4, DZeile$
[801] 2802: END IF
[M23] 2803: AusgebenParameter
[HS0] 2804: LOOP
[351] 2805: CLOSE #4
[L56] 2806: IF NOT KeinEnde THEN GOTO sym.schluss
[A92] 2807: kein.schluss:
[691] 2808: PRINT #2,
[5U5] 2809: PRINT #2, " Der END-Befehl wurde zu ";
[P35] 2810: PRINT #2, "früh oder gar nicht ";
[0K6] 2811: PRINT #2, "benutzt. SAM 4 unterbricht die";
[TY5] 2812: PRINT #2, " Übersetzung der SAM-";
[KU5] 2813: PRINT #2, "Datei. Drücken Sie bitte ";
[345] 2814: PRINT #2, "eine beliebige Taste !";
[1V1] 2815: WarteTaste
[GG2] 2816: KeinEnde = wahr
[R82] 2817: sym.schluss:
[TE1] 2818: EXIT SUB
[E20] 2819: '
[X06] 2820: 'Fehlerabfangroutine für 'INCLUDE'
[T62] 2821: fehl.include:
[FS4] 2822: IF NOT dos THEN GOTO fehl.inw1
[GX0] 2823: PRINT
[V26] 2824: PRINT "SAM 4 wurde durch einen Fehler ";
[RS6] 2825: PRINT "beim INCLUDE-Befehl unterbrochen."
[GX0] 2826: PRINT
[TE1] 2827: EXIT SUB
[E20] 2828: '
[XM1] 2829: fehl.inw1:
[MM1] 2830: LOCATE 10, 6
[FY5] 2831: PRINT "Sie haben den INCLUDE-Befehl ";
[8U5] 2832: PRINT "falsch benutzt und müssen ";
[X73] 2833: PRINT "Ihre Eingabe"
[VR1] 2834: LOCATE 12, 14
[FU6] 2835: PRINT "berichtigen. Drücken Sie bitte ";
[DS4] 2836: PRINT "eine beliebige Taste !";
[MV3] 2837: DO WHILE INKEY$ = "": LOOP
[WK1] 2838: 'GOTO men2
[HA1] 2839: END SUB
[000] 2840:
[AB7] 2841: 'Unterprogramm, das eine 16-Bit-Zahl in den
[8M7] 2842: 'niedrigwertigen und höherwertigen Bereich
[M52] 2843: 'unterteilt
[HP2] 2844: SUB Teilen16Bit
[T42] 2845: hv$ = HEX$(Lsb)
[WY4] 2846: hv$ = STRING$(4 - LEN(hv$), "0") + hv$
[1B4] 2847: Lsb = VAL("&H" + RIGHT$(hv$, 2))
[K34] 2848: Msb = VAL("&H" + LEFT$(hv$, 2))
[HA1] 2849: END SUB
[000] 2850:
[8F4] 2851: FUNCTION Umkehr$ (Zkette$)
[KG1] 2852: Pruef$ = ""
[ZF3] 2853: IF LEN(Zkette$) > 0 THEN
[7W3] 2854: DO WHILE LEN(Zkette$) > 0
[EHS] 2855: Pruef$ = Pruef$ + RIGHT$(Zkette$, 1)
[RB6] 2856: Zkette$ = LEFT$(Zkette$, LEN(Zkette$) - 1)
[HS0] 2857: LOOP
[801] 2858: END IF
[TK2] 2859: Umkehr$ = Pruef$
[R52] 2860: END FUNCTION
[000] 2861:
[L56] 2862: 'Anzeigen des Inhaltsverzeichnisses
[6V2] 2863: SUB Verzeichnis
[AK2] 2864: Ausgabefenster
[FU2] 2865: FILES Dateimaske$
[GV2] 2866: Standardfenster
[HA1] 2867: END SUB
[000] 2868:

```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic geschrieben.


```
[EH2] 2869: SUB Wartetaste
[MV3] 2870: DO WHILE INKEY$ = "": LOOP
[HA1] 2871: END SUB
[000] 2872:
[2M5] 2873: 'Unterprogramm, das den direkten
[5S3] 2874: 'Wortzugriff beachtet
[6M2] 2875: SUB WortDirekt
[ED4] 2876: IF ZielTyp AND Direkt16 THEN
[NV3] 2877: DirektWert = ZielWert1
[AC2] 2878: WortSpeichern
[801] 2879: END IF
[KW4] 2880: IF QuellTyp AND Direkt16 THEN
[D64] 2881: DirektWert = QuellWert1
[AC2] 2882: WortSpeichern
[801] 2883: END IF
[HA1] 2884: END SUB
[000] 2885:
[3A4] 2886: 'Unterprogramm, das 16 Bit
[973] 2887: '(direkt) speichert
[R23] 2888: SUB WortSpeichern
[SR2] 2889: Lsb = DirektWert
[402] 2890: Teilen16Bit
[3Y4] 2891: Befehllaenge = Befehllaenge + 2
[405] 2892: BefehlTab(Befehllaenge - 1) = Lsb
[8M4] 2893: BefehlTab(Befehllaenge) = Msb
[HA1] 2894: END SUB
[000] 2895:
[6V2] 2896: SUB ZeigeAssLis
[SR1] 2897: LOCATE 22, 12
[5T2] 2898: IF AssLis = 1 THEN
[4E3] 2899: PRINT "auf Bildschirm"
[6H3] 2900: ELSEIF AssLis = 2 THEN
[T63] 2901: PRINT "auf Drucker "
[8H3] 2902: ELSEIF AssLis = 3 THEN
[A73] 2903: PRINT "in eine Datei "
[ZR0] 2904: ELSE
[BG3] 2905: PRINT "nicht ausgeben"
[801] 2906: END IF
[HA1] 2907: END SUB
[000] 2908:
[2M5] 2909: SUB ZeigeDatei (Datei$, y, x, Laenge)
[UT1] 2910: LOCATE y, x
[Z94] 2911: IF LEN(Datei$) <= Laenge THEN
[P42] 2912: PRINT Datei$;
[XK5] 2913: PRINT SPACE$(Laenge - LEN(Datei$));
[ZR0] 2914: ELSE
[NR4] 2915: PRINT LEFT$(Datei$, Laenge - 4);
[Z22] 2916: PRINT CHR$(26); "...";
[801] 2917: END IF
[HA1] 2918: END SUB
[000] 2919:
[UP3] 2920: SUB ZeigeDruckstatus
[GS1] 2921: LOCATE 18, 66
[SK4] 2922: PRINT USING "####"; Druckbreite;
[KS1] 2923: LOCATE 19, 66
[P05] 2924: PRINT USING "####"; Seitenlaenge;
[HA1] 2925: END SUB
[000] 2926:
[AU2] 2927: SUB ZeigeModell
[3N1] 2928: LOCATE 19, 3
[835] 2929: IF Modell = M.COM THEN PRINT "COM"
[A35] 2930: IF Modell = M.EXE THEN PRINT "EXE"
[HA1] 2931: END SUB
[000] 2932:
[YD6] 2933: 'Unterprogramm, das den Befehlsaufbau
[U02] 2934: ' analysiert
[283] 2935: SUB ZeileAufteilen
[B45] 2936: DO WHILE ZeilZeigl <= EndZeiger
[8A9] 2937: IF INSTR(Trenn$, MID$(Zeile$, ZeilZeigl
, 1)) = 0 THEN EXIT DO
ZeilZeigl = ZeilZeigl + 1
[HS0] 2939: LOOP
[XN4] 2940: IF ZeilZeigl > EndZeiger THEN
[PP2] 2941: gefunden = falsch
[9U2] 2942: GOTO Aufbau.Ende
[801] 2943: END IF
[1M5] 2944: IF MID$(Zeile$, ZeilZeigl, 1) = "" THEN
[U06] 2945: ZeichEnde = INSTR(ZeilZeigl + 1, Zeile$
, "")
[SC3] 2946: IF ZeichEnde <> 0 THEN
[PW3] 2947: ZeilZeig2 = ZeichEnde + 1
[4A3] 2948: GOTO Feldzuweisung
[801] 2949: END IF
[801] 2950: END IF
[WF3] 2951: ZeilZeig2 = ZeilZeigl
[E45] 2952: DO WHILE ZeilZeig2 <= EndZeiger
[FA9] 2953: IF INSTR(Trenn$, MID$(Zeile$, ZeilZeig2
, 1)) > 0 THEN EXIT DO
ZeilZeig2 = ZeilZeig2 + 1
[HS0] 2955: LOOP
[CM2] 2956: Feldzuweisung:
[8F8] 2957: feld$ = MID$(Zeile$, ZeilZeigl, ZeilZeig2
- ZeilZeigl)
[WF3] 2958: ZeilZeigl = ZeilZeig2
[4G2] 2959: gefunden = wahr
[F02] 2960: Aufbau.Ende:
[HA1] 2961: END SUB
```

Listing 1. Der Makro-Assembler SAM 4.0 ist in QBasic geschrieben.

```
[VP2] 1: ; Program: Janein
[NA2] 2: ; Sprache: SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[574] 6: data segment para 'data'
[6W4] 7: Meldung db '(J)a oder (N)ein?$',
[D63] 8: Vorschub db 13,10,'$'
[ZW1] 9: data ends
[P30] 10: ;
[X54] 11: code segment para 'code'
[P30] 12: ;
[ZM3] 13: ;-Segmentzuweisungen
[KK3] 14: assume cs:code,ds:data
[P30] 15: ;
[P30] 16: ;
[7P2] 17: janein proc far
[P30] 18: ;
[005] 19: ;-Datensegment initialisieren
[A32] 20: mov ax,data
[DS1] 21: mov ds,ax
[P30] 22: ;
[3E4] 23: ;-Tastaturabfrage ausgeben
[DY3] 24: mov dx,offset(Meldung)
[TV1] 25: mov ah,09h
[E61] 26: int 21h
[P30] 27: ;
[507] 28: ;-Ein Zeichen aus dem Tastaturpuffer holen
[KP2] 29: start: mov ah,8
[E61] 30: int 21h
[P30] 31: ;
[EB6] 32: ;-Groß- in Kleinbuchstaben umwandeln
[FN1] 33: or al,60h
[P30] 34: ;
[8T4] 35: ;-Ist es der Buchstabe nJ'?
[VV1] 36: cmp al,6ah
[202] 37: je ZeichenJ
[P30] 38: ;
[GT4] 39: ;-Ist es der Buchstabe nN'?
[7W1] 40: cmp al,6eh
[A02] 41: je ZeichenN
[P30] 42: ;
[1V5] 43: ;-Wiederholung der Tastaturabfrage
[SV1] 44: jmps start
[P30] 45: ;
[2L6] 46: ;-Variable 'errorlevel' auf 0 setzen
[FY2] 47: ZeichenJ: mov al,1
[EL1] 48: jmps exit
[P30] 49: ;
[2L6] 50: ;-Variable 'errorlevel' auf 0 setzen
[343] 51: ZeichenN: xor al,al
[P30] 52: ;
[UC4] 53: ;-Zeilenvorschub ausgeben
[R65] 54: exit: mov dx,offset(Vorschub)
[TV1] 55: mov ah,09h
[E61] 56: int 21h
[P30] 57: ;
[R03] 58: ;-Programm beenden
[ZW1] 59: mov ah,4Ch
[E61] 60: int 21h
[P30] 61: ;
[A32] 62: janein endp
[P30] 63: ;
[P30] 64: ;
[BW1] 65: code ends
[ES1] 66: end start
```

Listing 2. Ein Programmbeispiel mit einem Daten- und einem Codesegment, das sich gut für Einsatz in Stapeldateien eignet

```
[S83] 1: ; Program: Macrobsp
[NA2] 2: ; Sprache: SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[226] 6: prog segment para 'code' ;-Programmsegment
[1D5] 7: org 1000h ;-ORG 1000H(COM-Datei)
[P30] 8: ;
[MY6] 9: include macrobib.sam ;-MACROBIB einbinden,
[S43] 10: ; INCLUDE darf nicht
[LG2] 11: ; in der ersten
[KP2] 12: ; Spalte stehen
[P30] 13: ;
[FX5] 14: do clrscr ;-Makros aufrufen mit
[3E6] 15: do setcursor,40,05 ; Übergabeparametern
[2K6] 16: do zeichenaus,65 ;-Zwischen Makronamen
```



```
[596] 17: do zeichenaus,'B' ; und Parametern muß
[ED6] 18: do zeichenaus,01000011B ; ein Komma stehen
[4M2] 19: do setcursor,0,20
[DD4] 20: do textaus,offset(text),cs
[KM2] 21: do setcursor,5,10
[P30] 22: ;
[LH1] 23: mov al,'A'
[PC1] 24: push ax
[FK2] 25: do zeichenaus,al
[6H2] 26: do zeichenaus,'='
[R31] 27: pop ax
[ZX1] 28: do hexaus,al
[PM2] 29: do setcursor,5,12
[N53] 30: do hexwortaus,1234h
[UM2] 31: do setcursor,5,14
[WP1] 32: mov ax, es
[TR2] 33: do hexwortaus,ax
[SP1] 34: mov ax, cs
[TR2] 35: do hexwortaus,ax
[UP1] 36: mov ax, ds
[TR2] 37: do hexwortaus,ax
[SR1] 38: mov ax, ss
[TR2] 39: do hexwortaus,ax
[AP1] 40: do schluss
[P30] 41: ;
[P30] 42: ;
[H52] 43: ;-Ausgabetext
[Z76] 44: text db 'Macrobib enthält: Clrscr, '
[P30] 45: ;
[UH6] 46: db 'Zeichenaus, Setcursor, Textaus,','0ah
[ZD6] 47: db 0dh,'Hexaus, Hexwortaus, Pushregs, '
[UF3] 48: db 'Popregs.','0ah,0dh,'$'
[P30] 49: ;
[P30] 50: ;
[P30] 51: prog ends ; Programmende
[P30] 52: ;
[XD1] 53: END prog
```

Listing 3. Das Assembler-Programm »mbibdemo.asm« nutzt die Makro- und Include-Technik.

```
[263] 1: ; Program: Macrobib
[NA2] 2: ; Sprache: SAM 4
[R86] 3: ; Copyright 1992 DMV Widuch GmbH & Co. KG
[P30] 4: ;
[P30] 5: ;
[7N4] 6: ;-MACROBIB.SAM einbinden mit:
[BR3] 7: ; "INCLUDE MACROBIB.SAM"
[P30] 8: ;
[9P6] 9: ;-INCLUDE darf nicht in der ersten Spalte
[6M1] 10: ; beginnen
[P30] 11: ;
[L23] 12: ;-Makroaufruf mit:
[8E6] 13: ; "DO MACRONAME[,Param1][,Param2][,...]"
[P30] 14: ;
[P30] 15: ;
[GB2] 16: clrscr macro
[R02] 17: ;-Löscht den
[U02] 18: ; Bildschirm
[994] 19: mov ah,0fh ;-Videomodus
[5X2] 20: int 10h ; lesen
[534] 21: mov ah,0 ;-Videomodus
[N83] 22: int 10h ; setzen
[HP0] 23: endm
[P30] 24: ;
[P30] 25: ;
[673] 26: textaus macro %1,%2
[EZ3] 27: mov ax,%2 ;-Gibt auf dem
[2X2] 28: ; Standardgerät
[ZC2] 29: ; Zeichenkette
[NP0] 30: ; aus
[H44] 31: mov ds,ax ;-Segment der
[ZC2] 32: ; Zeichenkette
[614] 33: mov dx,%1 ;-Adresse der
[ZC2] 34: ; Zeichenkette
[5R3] 35: mov ah,9 ; Ende mit '$'
[381] 36: int 21h
[HP0] 37: endm
[P30] 38: ;
[P30] 39: ;
[743] 40: zeichenaus macro %1
[3H1] 41: ;-Gibt ein
[Y12] 42: ; Zeichen aus
[A24] 43: mov dl,%1 ;-Zeichencode
[V91] 44: mov ah,2
[381] 45: int 21h
[HP0] 46: endm
[P30] 47: ;
[P30] 48: ;
[NB3] 49: setcursor macro %1,%2
```

```
[8L2] 50: ;-Setzt Cursor
[994] 51: mov ah,0fh ;-Videomodus
[5X2] 52: int 10h ; lesen
[NL1] 53: mov ah,02h
[TB3] 54: mov dl,%1 ;-Spalte
[Z03] 55: mov dh,%2 ;-Zeile
[GG4] 56: int 10h ;-Cursor setzen
[HP0] 57: endm
[P30] 58: ;
[P30] 59: ;
[TH2] 60: schluss macro
[5W3] 61: mov ax,04c00h ;-Programm
[773] 62: int 21h ; beenden
[HP0] 63: endm
[P30] 64: ;
[P30] 65: ;
[5S2] 66: hexaus macro %1
[B64] 67: mov dl,%1 ;-Gibt ein Byte
[L12] 68: ; als Hexzahl
[EC4] 69: push dx ; auf Standard-
[262] 70: ; gerät aus
[4E4] 71: shr dl,1 ;-Sprungbefehle
[DV1] 72: ; sind als DB
[7F4] 73: shr dl,1 ; codiert, da SAM
[8M2] 74: ; keine lokalen
[873] 75: shr dl,1 ; Marken
[HR3] 76: shr dl,1 ; verwaltet
[5P2] 77: db e8h,06h,0 ;-call
[D44] 78: pop dx ; Hexausnibble
[7H4] 79: db e8h,02h,0 ;-call Hexausnib.
[4G3] 80: db ebh,13h ;-jumps Ende
[GH4] 81: proc near ;-Hexausnibble:
[CK1] 82: and dl,0fh
[081] 83: or dl,30h
[PL1] 84: cmp dl,3ah
[AY3] 85: db 7ch,03h ;-jl Ausgeben
[GG1] 86: add dl,07h
[EM3] 87: mov ah,2 ;-Ausgeben:
[381] 88: int 21h
[3P0] 89: ret
[MP0] 90: endp
[9M2] 91: endm ;-Ende:
[083] 92: hexwortaus macro %1
[2B4] 93: mov dx,%1 ;-Gibt ein Wort
[L12] 94: ; als Hexzahl
[EC4] 95: push dx ; auf Standard-
[262] 96: ; gerät aus
[FF1] 97: mov dl,dh
[HP4] 98: db e8h,06h,0 ;-call Hexwortau.
[X31] 99: pop dx
[4L4] 100: db e8h,02h,0 ;-call Hexwortau.
[LP4] 101: db ebh,24h ;-jumps Hexworten.
[KM4] 102: proc near ;-Hexwortausml:
[ZC1] 103: push dx
[L91] 104: shr dl,1
[L91] 105: shr dl,1
[L91] 106: shr dl,1
[L91] 107: shr dl,1
[AL4] 108: db e8h,05h,0 ;-call hexwortau.
[X31] 109: pop dx
[2L4] 110: db e8h,01h,0 ;-call hexwortau.
[3P0] 111: ret
[MP0] 112: endp
[MM4] 113: proc near ;-Hexwortausm2:
[CK1] 114: and dl,0fh
[081] 115: or dl,30h
[PL1] 116: cmp dl,3ah
[MV3] 117: db 7ch,03h ;-jl Ausgabe
[GG1] 118: add dl,07h
[CD3] 119: mov ah,2 ;-Ausgabe:
[381] 120: int 21h
[3P0] 121: ret
[MP0] 122: endp
[YX3] 123: endm ;-Hexwortende:
[4L2] 124: pushregs macro
[Z74] 125: push si ;-Register auf
[Z14] 126: push di ; den Stapel-
[0P4] 127: push bp ; speicher legen
[ZC1] 128: push dx
[WC1] 129: push cx
[TC1] 130: push bx
[PC1] 131: push ax
[CD1] 132: push sp
[HP0] 133: endm
[TH2] 134: popregs macro
[E64] 135: pop sp ;-Register vom
[BR4] 136: pop ax ; Stapelspeicher
[LX2] 137: pop bx ; lesen
[V31] 138: pop cx
[X31] 139: pop dx
[L31] 140: pop bp
[G21] 141: pop di
[E31] 142: pop si
[HP0] 143: endm
```

Listing 4. Die kleine SAM-4.0.-Bibliothek »macrobib.asm« bietet unter anderem Funktionen zur Cursorsteuerung und Zeichenausgabe.

Das Ende von Babylon

Wer schon des öfteren nach bestimmten Assembler-Befehlen suchen mußte, merkt sehr schnell, daß er mit einem Buch allein nicht auskommt. Bücher besitzen aber häufig den Nachteil, daß sie nicht ohne eine hanebüchenen Anzahl von Seiten im Kilopaket auskommen. Es fehlt also eine kurze und klare Referenz.

Abkürzung	Bedeutung
CF	Carry-Flag
BCD	binär codierte Dezimalzahlen
Var, Varx	Variable, Größe: x Bit
Adr	Adresse
R, Rx	Register, Größe: x Bit
RAM, RAMx	Arbeitsspeicher, Größe: x Bit
Akku	Akkumulator-Register (AL, AX, EAX)
SegR	Segment-Register
CRx	CR0 bis CRx-Register
DRx	DR0 bis DRx-Register
TR	TR-Register

Tabelle 1. Eine kleine Übersicht über die in Tabelle 2 verwendeten Abkürzungen

Befehl	Syntax	Operanden	Taktzyklen bei den Prozessoren			Beschreibung
			8086/88	80286	80386/80486	
aaa	aaa	—	8	3	4	BCD-Korrektur nach Addition
aad	aad	—	60	14	19	BCD-Korrektur nach Division
aam	aam	—	83	16	17	BCD-Korrektur nach Multiplikation
aas	aas	—	8	3	4	BCD-Korrektur nach Subtraktion
adc	adc Ziel, Quelle	R, R R, RAM RAM, R	3 10 18	2 7 7	2 6 7	Addition mit Übertrag (CF einbezogen bei Addition); Ergebnis im Ziel
add	add Ziel, Quelle	R, R R, RAM RAM, R	3 9 16	2 7 7	2 6 7	Addition ohne CF-Benutzung
and	and Ziel, Quelle	R, R R, RAM RAM, R	3 10 17	2 7 7	2 6 7	UND-Verknüpfung von Ziel mit Quelle
arpl	arpl Zielselektor, Quellselektor	R, R RAM, R	— —	10 11	20 21	Anpassung eines RPL-Felds an Privilegstufe eines Selektors (ab 80286)
bound	Index, Feldgrenzen-Adresse	R16, RAM32 R32, RAM64	— —	13 13	10 10	überprüft, ob Feldindex innerhalb bestimmter Grenzen (ab 186)
bsf	bsf Ziel, Quelle	R, R R, RAM	— —	— —	10+3n 10+3n	durchsucht Operanden vorwärts nach erstem gesetztem Bit (ab 80386)
bsr	bsr Quelle, Ziel	R, R R, RAM	— —	— —	10+3n 10+3n	durchsucht Operanden rückwärts nach erstem gesetztem Bit (ab 80386)
bt	bt Quelle, Ziel	R, R RAM, R	— —	— 12	3 12	testet ein Bit im Quelloperanden und überträgt es ins CF (ab 80386)
btc	btc Quelle, Ziel	R, R RAM, R	— —	— 13	6 13	testet ein Bit, überträgt es ins CF und invertiert es (ab 80386)
btr	btr Quelle, Ziel	R, R RAM, R	— —	— 13	6 13	testet ein Bit, überträgt es ins CF und setzt es zurück (ab 80386)
bts	bts Quelle, Ziel	R, R RAM, R	— —	— 13	6 13	testet ein Bit, überträgt es ins CF (ab 80386)
bswap	bswap R32	R	—	—	1	vertauscht die Bytes in einem 32-Bit-Register (ab 80486)
call	call Zieladresse	Prozedurtyp NEAR	19	7	7	Aufruf einer Prozedur
		Prozedurtyp FAR	28	13	17	
		RAM16	24	11	10	
		R16	16	7	7	
cbw	cbw	RAM32	39	16	22	erweitert AL-Register vorzeichengerecht auf 16 Bit
		—	2	2	3	
		—	—	—	2	
		—	2	2	2	
cdq	cdq	—	—	—	2	erweitert ein Wort auf ein Doppelwort (ab 80386)
clc	clc	—	2	2	2	setzt CF auf 0
cld	cld	—	2	2	2	setzt Richtungsflag auf 0
cli	cli	—	2	3	3	löscht Interrupt-Flag
clts	clts	—	—	2	5	löscht Task-Wechselflag (TS-Bit) (ab 80286)
cmc	cmc	—	2	2	2	dreht CF um
cmp	cmp Ziel, Quelle	R, R	3	2	2	vergleicht zwei Operanden und setzt entsprechend das Status-Flag
		R, RAM	10	6	6	
		RAM, R	9	7	6	

Befehl	Syntax	Operanden	8086/88	80286	80386/80486	Beschreibung
cmpsb	cmpsb Zielstring, Quellstring	–	22	8	10	vergleicht zwei Strings (ES:DI und DS:SI)
cmpsw	cmpsw Zielstring, Quellstring	–	22	8	10	vergleicht zwei Strings (ES:DI und DS:SI)
cmpsd	cmpsd Zielstring, Quellstring	–	–	–	10	vergleicht zwei Strings (ES:DI und DS:SI) (ab 80386)
cmpxchg	cmpxchg Ziel, Quelle	RAM, R R, R	– –	– –	10 6	Vergleich und Austausch von Operanden (ab 80486)
cwde	cwde	–	5	2	2	wandelt ein Wort in Doppelwort um
cdwe	cdwe	–	–	–	3	wandelt ein Wort in Doppelwort um (ab 80386)
daa	daa	–	4	3	4	justiert nach dezimaler Addition
das	das	–	4	3	4	justiert nach dezimaler Subtraktion
dec	dec Ziel	R	3	2	2	verringert Operanden um 1
div	div Quelle	R8 R16 R32 RAM8 RAM16 RAM32	80-90 144 bis 162 – 86 bis 100 150 bis 168 –	14 22 – 150 bis 172 25 –	14 22 38 17 25 41	vorzeichenlose Division
enter	enter Operand1, Operand2	Var, Var8	–	12+4(n-1)	15+4(n-1)	baut einen Stackrahmen auf (ab 80186)
esc	esc Var, Quelle	Var, R Var, RAM	2 10	9 bis 20 9 bis 20	9 bis 20 9 bis 20	bewirkt, daß ein Befehl an Coprozessor geschickt wird
hlt	hlt	–	2	2	5	Wartezustand für CPU
idiv	idiv Quelle	R8 R16 R32 RAM8 RAM16 RAM32	102 bis 113 162 bis 185 – 108 bis 122 170 bis 190 –	17 25 – 20 28 –	19 27 43 22 30 46	dividiert AX:DX oder nur AX (ab 80386 auch EAX:EDX für 64 Bit) mit Vorzeichen durch Operanden
imul	imul Quelle	R8 R16 R32 RAM8 RAM16 RAM32	80 bis 99 128 bis 155 – 86 bis 104 135 bis 161 –	13 21 – 16 24 –	9 bis 13 9 bis 22 9 bis 38 12 bis 16 11 bis 26 13 bis 40	multipliziert DX:AX oder nur AX (ab 80386 auch EDX:EAX für 64 Bit) mit Vorzeichen durch Operanden
imul	imul Ergebnis, Operand1, Operand2	R16, Var R16, R16, Var R16, RAM16, Var R32, Var R32, R32, Var R, R, Var	– – – – – –	22 21 23 – – –	9 bis 21 10 bis 22 12 bis 26 10 bis 37 10 bis 36 10 bis 41	imul ab 80186 (Op1:Op2=Ergebnis); Ergebnisoperand ist optional
imul	imul Quelle, Ziel	R16, R16 R32, RAM32	– –	– –	9 bis 23 13 bis 40	imul ab 80386
in	in AL/AH/AX, DX/Port	AL/AH/AX, Var8 AL/AH/AX, DX	11 8	6 5	12 13	liest Port direkt oder über DX ein Byte/Word (AL/AH/AX) ein (ab 80386 auch EAX)
inc	inc Ziel	R	3	2	2	inkrementiert Zeile um 1
insb	insb	–	–	5	15	liest String (Adresse ES:DI) aus Port (DX) (ab 186)
insw	insw	–	–	5	15	wie insb
insd	insd	–	–	5	15	wie insb
int	int Intrnr	1 bis 255	52	23	36	ruft Interrupt mit der Nummer »Intrnr« auf
into	into	–	53	24	35	Interrupt 4 bei Überlauf (Overflow-Flag) aufrufen
iret	iret	–	32	17	22	beendet die Interrupt-Routine
ja	ja Adr	–	16	7	7	Sprung, wenn größer (wie jnbe)
jae	jae Adr	–	16	7	7	Sprung, wenn größer oder gleich (wie jnb)
jb	jb Adr	–	16	7	7	Sprung, wenn kleiner (wie jnae)
jbe	jbe Adr	–	16	7	7	Sprung, wenn kleiner oder gleich (wie jna)
jcxz	jcxz Adr	–	16	7	7	Sprung, wenn CX gleich 0
je	je Adr	–	16	7	7	Sprung, wenn gleich (wie JZ)
jecxz	jecxz Adr	–	–	–	7	Sprung, wenn CX gleich 0 (ab 80386)
jg	jg Adr	–	16	7	7	Sprung, wenn größer (mit Vorzeichen) (wie jnle)
jge	jge Adr	–	16	7	7	Sprung, wenn größer oder gleich (mit Vorzeichen) (wie jnl)
jl	jl Adr	–	16	7	7	Sprung, wenn kleiner (mit Vorzeichen) (wie jnge)
jle	jle Adr	–	16	7	7	Sprung, wenn kleiner oder gleich (mit Vorzeichen) (wie jng)
jmp	jmp Ziel	short near far R RAM16	15 15 15 11 19	7 7 11 7 11	7 7 13 7 10	unbedingter Sprung
jna	jna Adr	–	16	7	7	Sprung, wenn nicht größer (wie jbe)
jnae	jnae Adr	–	16	7	7	Sprung, wenn nicht größer oder gleich (wie jb)
jnb	jnb Adr	–	16	7	7	Sprung, wenn nicht kleiner (wie jae)
jnbe	jnbe Adr	–	16	7	7	Sprung, wenn nicht kleiner oder gleich (wie »ja«)

Befehl	Syntax	Operanden	8086/88	80286	80386/80486	Beschreibung
jne	jne Adr	–	16	7	7	Sprung, wenn ungleich (mit Vorzeichen) (wie jnz)
jng	jng Adr	–	16	7	7	Sprung, wenn nicht größer (mit Vorzeichen) (wie jle)
jnge	jnge Adr	–	16	7	7	Sprung, wenn nicht größer oder gleich (mit Vorzeichen) (wie jl)
jnl	jnl Adr	–	16	7	7	Sprung nicht kleiner (mit Vorzeichen) (wie jge)
jnle	jnle Adr	–	16	7	7	Sprung, wenn größer oder gleich (mit Vorzeichen) (wie jg)
jno	jno Adr	–	16	7	7	Sprung, wenn kein Überlauf
jnp	jnp Adr	–	16	7	7	Sprung, wenn kein Paritätsflag (wie jpo)
jns	jns Adr	–	16	7	7	Sprung, wenn kein Vorzeichen
jnz	jnz Adr	–	16	7	7	Sprung, wenn ungleich (mit Vorzeichen) (wie jnz)
jo	jo Adr	–	16	7	7	Sprung, wenn Überlauf
jp	jp Adr	–	16	7	7	Sprung, wenn Paritätsflag gesetzt (wie jpe)
jpe	jpe Adr	–	16	7	7	Sprung, wenn gerade Parität (wie jp)
jpo	jpo Adr	–	16	7	7	Sprung, wenn ungerade Parität (wie jnp)
js	js Adr	–	16	7	7	Sprung, wenn Vorzeichen (negativ)
jz	jz Adr	–	16	7	7	Sprung, wenn 0
lahf	lahf	–	4	2	2	lädt AH mit unterer Hälfte des Flag-Registers
lar	lar Ziel, Selektor	R, R R, RAM	– –	14 16	15 16	Zugriffsrechte eines Selektors laden (ab 80286)
lds	lds Ziel, Quelle	R16, RAM16	17	7	7	lädt Zeiger in DS und Zielregister
lea	lea Ziel, Quell	R16, RAM16	4	3	2	lädt Quelloffset in Zielregister
leave	leave	–	–	5	4	entfernt Stackrahmen (siehe enter) (ab 186)
les	les Ziel, Quelle	R16, RAM32	18	7	7	lädt 32-Bit-Zeiger in ES und Zielregister
lfs	lfs Ziel, Quelle	R16, RAM32	–	–	7	lädt 32-Bit-Zeiger in FS-Register und Zielregister (ab 80386)
lgdt	lgdt Operand	RAM48	–	11	11	lädt Adresse der GDT (Globale Deskriptor Tabelle) in GDT-Register (ab 80286)
lgs	lgs Ziel, Quelle	R16, RAM32	–	–	7	lädt 32-Bit-Zeiger in CS und Zielregister (ab 80386)
lidt	lidt RAM48	RAM48	–	12	11	lädt IDT-Adresse (Interrupt-Deskriptor-Tabelle) in IDT-Register (ab 80286)
lldt	lldt Operand	R16 RAM16	– –	17 19	20 24	lädt den Deskriptor einer LDT aus der GDT ins Cache-Register (ab 80286)
lmws	lmws Operand	R16 RAM16	– –	3 6	10 13	lädt Wert in das MSW-Register (ab 80286)
lock	lock Befehl	–	2	0	0	Bus sperren, solange bis Befehl abgearbeitet
lodsb	lodsb Quelle	–	12	5	5	lädt String aus DS:SI ins Register AL
lodsw	lodsw Quelle	–	12	5	5	lädt String aus DS:SI ins Register AX
lodsd	lodsd Quelle	–	–	–	5	lädt String aus DS:SI in EAX (ab 80386)
loop	loop Adr	–	17	8	11	Sprung zu Adr solange CX ungleich 0 ist
loope	loope Adr	–	17	8	11	Sprung zu Adr solange CX ungleich 0 und Null-Flag gesetzt ist
loopne	loopne Adr	–	17	8	11	Sprung zu Adr solange CX ungleich 0 und das Null-Flag nicht gesetzt ist
lsl	lsl Register, Selektor	R, R R, RAM	– –	14 16	20 21	lädt Selektor in Zielregister (ab 80286)
lss	lss Register, RAM	R16, RAM32	–	–	7	lädt Zeiger in FS und ins Ziel (ab 80386)
ltr	ltr Selektor	R16 RAM16	– –	17 19	24 26	lädt Selektor ins aktuelle Taskregister (ab 80286)
mov	mov Ziel, Quelle	R, R R, RAM RAM, R	4 8 8	4 5 3	4 4 2	kopiert Quelle ins Ziel
movsb	movsb Ziel, Quelle	–	19	6	7	kopiert Stringelement aus SI in ES:Zielstring
movsw	movsw Ziel, Quelle	–	19	6	7	kopiert Stringelement aus SI in ES:Zielstring
movsd	movsd Ziel, Quelle	–	–	–	7	kopiert Stringelement aus SI in ES:Zielstring (ab 80386)
movsx	movsx Ziel, Quelle	R, R R, RAM	– –	– –	3 6	kann 8- oder 16-Bit-Operanden in 16- oder 32-Bit-Register kopieren (ab 80386)
movzx	movzx Ziel, Quelle	R, R	–	–	3	kann 8- oder 16-Bit-Operanden in 16- oder 32-Bit-Register kopieren (ab 80386);
mul	mul Quelle	AH/AL, R8 AX, R16 EAX, R32 AH/AL, RAM8 AX, RAM16 EAX, RAM32	70 117 – 80 130 –	13 19 – 15 25 –	10 9 9 12 13 13	multipliziert Quelle mit Akkumulator-Register (AH,AL,AX); als Quelle ein Byte dann Ergebnis in AX, als Quelle 1 Word dann Ergebnis in DX:AX, als Quelle 1 Doubleword dann Ziel EAX und Ergebnis in EDX:EAX
neg	neg Quelle	R RAM	3 20	2 7	2 7	Zweierkomplement von Quelle bilden
nop	nop	–	3	3	3	Leerbefehl (no operate)

Befehl	Syntax	Operanden	8086/88	80286	80386/80486	Beschreibung
not	not Quelle	R	3	2	2	invertiert Quellbit
		RAM	20	8	6	
or	or Ziel, Quelle	R, R	3	2	2	verknüpft die jeweiligen Bits untereinander
		R, RAM	15	7	6	
		RAM, R	20	7	7	
out	out Port/DX, AL/AX/EAX	Var8, Akku	10	3	10	gibt den Wert aus dem Akkumulator-Register auf ein Port aus
outsb	outsb dx, SegR:Quelle	–	–	5	14	Ausgabe der Elemente einer Zeichenkette (DS:SI) in Byte-Blöcken auf ein Port (DX)
outsw	outsw dx, SegR:Quelle	–	–	5	14	Ausgabe der Elemente einer Zeichenkette (DS:SI) in Word-Blöcken auf ein Port (DX)
outsd	outsd dx, SegR:Quelle	–	–	–	14	Ausgabe der Elemente einer Zeichenkette (DS:SI) in Doubleword-Blöcken auf ein Port (DX) (ab 80386)
pop	pop Ziel	R	8	5	4	lädt Ziel auf den Stapelspeicher
		RAM	19	6	5	
popa	popa	–	–	19	25	lädt die oberen 16 Byte vom Stapelspeicher in die Register DI, SI, BP, BX, DX, CX und AX. Das SP-Register wird nicht korrigiert (ab 80186)
popad	popad	–	–	–	24	lädt die oberen 32 Byte vom Stapelspeicher in die Register EDI, ESI, EBP, EBX, EDX, ECX und EAX. Das ESP-Register wird nicht korrigiert (ab 80386)
popf	popf	–	8	5	5	lädt das Flag-Register mit den oberen 16 Byte des Stapelspeichers
popfd	popfd	–	–	–	5	legt das Flag-Register mit den oberen 32 Byte des Stapelspeichers (ab 80386)
push	push Quelle	R	11	3	3	lädt den Inhalt der Quelle auf den Stapelspeicher ab
		RAM	19	5	5	
pusha	pusha	–	–	17	18	lädt den Inhalt der Register AX, CX, DX, BX, SP, BP, SI und DI auf den Stapelspeicher (ab 186)
pushad	pushad	–	–	–	18	lädt den Inhalt der Register EAX, ECX, EDX, EBX, ESP, EBP, ESI und EDI auf den Stapelspeicher (ab 80386)
pushf	pushf	–	10	3	4	lädt den Inhalt des Flag-Registers auf den Stapelspeicher
pushfd	pushfd	–	–	–	4	lädt den Inhalt des Flag-Registers auf den Stapelspeicher (ab 80386)
rcl	rcl Ziel, Anzahl	R, 1	2	2	9	Bits des Ziels um Anzahl mit einbezogenem CF nach links rollen
		R, cl	8+4cl	5+cl	9	
		RAM, 1	17	7	11	
		RAM, cl	19+4cl	7+cl	10	
rcr	rcr Ziel, Anzahl	R, 1	2	2	9	Bits des Ziels um die Anzahl mit einbezogenem CF nach links rollen
		R, cl	8+4cl	5+cl	9	
		RAM, 1	17	7	11	
		RAM, cl	19+4cl	7+cl	10	
rep	rep	rep movsb/movsw/movsd	10+17n	5+4n	8+4/n	Befehl wiederholen bis CX gleich 0
		rep stosb/stosw/stosd	9+10n	4+3n	5+5n	
		rep insb/insw/insd	–	5+4n	13+6n	
		rep outsb/outsw/outd	–	5+4n	13+6n	
repe	repe	repe cmpsb/cmptsw/cmptsd	9+21n	5+9n	5+9n	wiederhole Befehl, solange CX ungleich 0 und das Nullflag gesetzt ist (wie repz)
		repe scasb/scasw/scasd	8+15n	5+8n	5+8n	
repne	repne	repne cmpsb/cmptsw/cmptsd	9+21n	5+9n	5+9n	wiederhole Befehl, solange CX ungleich 0 und das Nullflag gelöscht ist (wie repnz)
		repne scasb/scasw/scasd	8+15n	5+8n	5+8n	
repnz	repnz	repnz cmpsb/cmptsw/cmptsd	9+21n	5+9n	5+9n	wiederhole Befehl, solange CX ungleich 0 und das Nullflag gelöscht ist (wie repne)
		repnz scasb/scasw/scasd	8+15n	5+8n	5+8n	

Befehl	Syntax	Operanden	8086/88	80286	80386/80486	Beschreibung
repz	repz	repz cmpsb/ cmpsw/cmpsd	9+21n	5+9n	5+9n	wiederhole Befehl, solange CX ungleich 0 und das Nullflag gesetzt ist (wie repe)
ret	ret sp-Korrektur	retn (near) retf (far)	16 26	12 15	11 17	Rückkehr aus einer mit Call aufgerufenen Prozedur (sp-Korrektur optional wegen etwaiger Parameterübergabe)
rol	rol Ziel, Anzahl	R, 1 R, cl RAM, 1 RAM, cl	2 7+4cl 16 19+4cl	2 5+cl 7 8+cl	3 3 7 7	Bits des Zielwerts um Anzahl nach links rotieren
ror	ror Ziel, Anzahl	R, 1 R, cl RAM, 1 RAM, cl	2 7+4cl 16 19+4cl	2 5+cl 7 8+cl	3 3 7 7	Bits des Zielwerts um Anzahl nach rechts rotieren
sahf	sahf	—	4	3	3	lädt Flagregister mit dem Wert aus AH
sal	sal Ziel, Anzahl	R, 1 R, cl RAM, 1 RAM, cl	2 7+4cl 16 19+4cl	2 5+cl 7 8+cl	3 3 7 7	Ziel arithmetisch nach links um Anzahl schieben (höherwertiges Bit kommt ins CF, Bit 0 wird mit 0 belegt)
sar	sar Ziel, Anzahl	R, 1 R, cl RAM, 1 RAM, cl	2 7+4cl 16 19+4cl	2 5+cl 7 8+cl	3 3 7 7	Ziel arithmetisch nach rechts um Anzahl schieben
sbb	sbb Ziel, Quelle	R, R R, RAM RAM, R	3 9 17	2 7 7	2 7 7	subtrahiert Quelle CF von Ziel
scasb	scasb String	—	15	7	7	durchsucht String (ES:DI) nach einem Byte (AL); Richtungsflag gibt Suchrichtung an
scasw	scasw String	—	15	7	7	durchsucht String (ES:DI) nach einem Wort (AX); Richtungsflag gibt Suchrichtung an
scasd	scasd String	—	—	—	7	durchsucht String (ES:DI) nach einem Doubleword (EAX); Richtungsflag gibt Suchrichtung an (ab 80386)
seta	seta Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn größer, sonst auf 0 setzen (wie setnbe) (ab 80386)
setae	setae Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn größer oder gleich, sonst auf 0 setzen (wie setnb) (ab 80386)
setb	setb Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn kleiner, sonst auf 0 setzen (wie setnae) (ab 80386)
setbe	setbe Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn kleiner oder gleich, sonst auf 0 setzen (wie setna) (ab 80386)
setc	setc Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn CF gesetzt, sonst auf 0 setzen (ab 80386)
sete	sete Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn gleich, sonst auf 0 setzen (wie setz) (ab 80386)
setg	setg Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn größer (mit Vorzeichen), sonst auf 0 setzen (wie setnle) (ab 80386)
setge	setge Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn größer oder gleich (mit Vorzeichen), sonst auf 0 setzen (wie setnl) (ab 80386)
setl	setl Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn kleiner (mit Vorzeichen), sonst auf 0 setzen (wie setnge) (ab 80386)
setle	setle Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn kleiner oder gleich (mit Vorzeichen), sonst auf 0 setzen (wie setng) (ab 80386)
setna	setna Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn nicht größer, sonst auf 0 setzen (wie setbe) (ab 80386)
setnae	setnae Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn nicht größer oder gleich, sonst auf 0 setzen (wie setb) (ab 80386)
setnb	setnb Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn nicht kleiner, sonst auf 0 setzen (wie setae) (ab 80386)
setnbe	setnbe Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn nicht kleiner oder gleich, sonst auf 0 setzen (wie seta) (ab 80386)
setnc	setnc Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn CF nicht gesetzt, sonst auf 0 setzen (ab 80386)
setne	setne Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn nicht gleich, sonst auf 0 setzen (wie setnz) (ab 80386)
setng	setng Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn nicht größer (mit Vorzeichen), sonst auf 0 setzen (wie setle) (ab 80386)
setnge	setnge Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn nicht größer oder gleich (mit Vorzeichen), sonst auf 0 setzen (wie setl) (ab 80386)
setnl	setnl Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn nicht kleiner oder gleich (mit Vorzeichen), sonst auf 0 setzen (wie setge) (ab 80386)
setnle	setg Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn nicht kleiner oder gleich (mit Vorzeichen), sonst auf 0 setzen (wie setg) (ab 80386)
setno	setno Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn kein Überlauf, sonst auf 0 setzen (wie setg) (ab 80386)
setnp	setnp Operand	R RAM	— —	— —	4 5	Operanden auf 1 setzen, wenn ungerade Parität, sonst auf 0 setzen (wie setpo) (ab 80386)

Befehl	Syntax	Operanden	8086/88	80286	80386/80486	Beschreibung
setns	sets Operand	R	—	—	4	Operanden auf 1 setzen, wenn kein Vorzeichen, sonst auf 0 setzen (ab 80386)
setnz	setnz Operand	R	—	—	4	Operanden auf 1 setzen, wenn Nullflag nicht gesetzt, sonst auf 0 setzen (wie setne) (ab 80386)
seto	seto Operand	R	—	—	4	Operanden auf 1 setzen, wenn Überlauf, sonst auf 0 setzen (ab 80386)
setp	setp Operand	R	—	—	4	Operanden auf 1 setzen, wenn gerade Parität, sonst auf 0 setzen (wie setpe) (ab 80386)
setpe	setpe Operand	R	—	—	4	Operanden auf 1 setzen, wenn gerade Parität, sonst auf 0 setzen (wie setp) (ab 80386)
setpo	setnp Operand	R	—	—	4	Operanden auf 1 setzen, wenn Überlauf und Parität, sonst auf 0 setzen (wie setnp) (ab 80386)
sets	sets Operand	R	—	—	4	Operanden auf 1 setzen, wenn Vorzeichen, sonst auf 0 setzen (ab 80386)
setz	setz Operand	R	—	—	4	Operanden auf 1 setzen, wenn Nullflag gesetzt, sonst auf 0 setzen (wie sete) (ab 80386)
sgdt	sgdt Operand	RAM48	—	11	9	speichert GDT-Register in Operand (ab 80286)
shl	shl Ziel, Anzahl	R, 1	2	2	3	Ziel nach links um Anzahl schieben (höherwertiges Bit kommt ins CF, Bit 0 wird mit 0 belegt)
		R, cl	7+4cl	5+cl	3	
		RAM, 1	16	7	7	
		RAM, cl	19+4cl	8+cl	7	
shld	shld Ziel, Quelle, Bitzahl	R16, R16, cl	—	—	3	schiebt Bits von Quelle ins Ziel und vom Ziel in Quelle (Verschiebung von links) (ab 80386)
		R32, R32, cl	—	—	7	
		RAM16, R16, cl	—	—	3	
		RAM32, R32, cl	—	—	7	
shr	shr Ziel, Anzahl	R, 1	2	2	3	schiebt Anzahl von Bits des Ziels nach rechts; Bit 0 fällt ins CF, und das höherwertige Bit wird mit 0 ausgefüllt
		R, cl	7+4cl	5+cl	3	
		RAM, 1	16	7	7	
		RAM, cl	19+4cl	8+cl	7	
shrd	shld Ziel, Quelle, Bitzahl	R16, R16, cl	—	—	3	schiebt Bits vom Ziel in Quelle und von Quelle ins Ziel (Verschiebung von rechts) (ab 80386)
		R32, R32, cl	—	—	7	
		RAM16, R16, cl	—	—	3	
		RAM32, R32, cl	—	—	7	
sidt	sidt Operand	RAM48	—	12	10	speichert IDT-Register in Operanden (ab 80286)
sldt	sldt Operand	R16	—	2	2	speichert Selektor vom LDT-Register in Operanden (ab 80286)
		RAM16	—	2	2	
smsw	smsw Operand	R16	—	2	2	speichert MSW-Register in Operanden (beim 8080386 wird untere Hälfte vom CR0-Register gespeichert) (ab 80286)
		RAM16	—	3	3	
stc	stc	—	2	2	2	setzt das CF
std	std	—	2	2	2	setzt das Richtungsflag
sti	sti	—	2	2	2	setzt das Interrupt-Flag
stosb	stosb String	—	11	3	4	speichert Inhalt aus AL in Zeichenkette (ES:DI)
stosw	stosw String	—	11	3	4	speichert Inhalt aus ax in Zeichenkette (ES:DI)
stosd	stosd String	—	—	—	4	speichert Inhalt aus eax in Zeichenkette (ES:DI) (ab 80386)
str	str Operand	R16	—	2	2	speichert TR-Register in Operanden (ab 80286)
		RAM16	—	3	3	
sub	sub Ziel, Quelle	R, R	3	2	2	subtrahiert Quelle von Ziel mit Ergebnis im Ziel (CF nicht berücksichtigt)
		R, RAM	10	7	7	
		RAM, R	17	7	7	
test	test Ziel, Quelle	R, R	3	2	2	schaft Und-Verknüpfung zwischen Bits vom Ziel und Bits von Quelle; Ergebnis im Flagregister
		R, RAM	10	5	5	
		RAM, R	10	5	5	
verr	verr Operand	R16	—	14	10	verifiziert den angegebenen Selektor und prüft, ob man diesen in der aktuellen Privilegstufe lesen kann (ab 80286)
		RAM16	—	16	11	
verw	verw Operand	R16	—	14	15	verifiziert den angegebenen Selektor und prüft, ob man diesen in der aktuellen Privilegstufe beschreiben kann (ab 80286)
		RAM16	—	16	16	
wait	wait	—	6+5n	3	6	versetzt die CPU in eine Warteschleife; alle fünf Taktzyklen wird geprüft, ob Leitung »test« aktiv, um »wait« zu beenden
xchg	xchg Ziel, Quelle	R, R	4	3	3	vertauscht Quelle mit Ziel
		R, RAM	18	5	5	
xadd	xadd Ziel, Quelle	R, R	—	—	3	addiert Quelle zum Ziel und lagert Ziel in Quelle; damit enthält das alte Ziel das Ergebnis und die alte Quelle das alte Ziel (ab 80486)
		RAM, R	—	—	3	
xlat	xlat	—	10	6	5	lädt Wert einer Tabelle (maximale Größe 256 Byte, Adresse in BX, Index in AL) in AL
xor	xor Ziel, Quelle	R, R	3	2	1	Exklusiv-ODER-Verknüpfung der Bits von Quelle mit den Bits vom Ziel; Ergebnis im Ziel
		R, RAM	10	7	7	
		RAM, R	15	7	6	

(st/uk)

Tabelle 2. Referenz der Assembler-Befehle

Die Befehle der Coprozessoren

Moderne Assembler erlauben fast komfortables Programmieren.

Aber jeder Programmierer kennt die Arbeit, die zum Beispiel hinter der Entwicklung von Fließkomma-Routinen steht. Wenn Sie einen arithmetischen Coprozessor eingebaut haben, ersetzen Sie umfangreiche Berechnungsprozeduren durch kurze Coprozessoranweisungen. Die Erweiterungen der Standardkommandos erleichtern den Umgang mit den mathematischen Fähigkeiten eines PCs erheblich.

Die meisten Programmierer formulieren PC-Anwendungen in höheren Programmiersprachen wie Basic, Pascal oder C. Die Maschinensprache fungiert jedoch unangefochten als schnellste aller Programmiersprachen.

Wenn Sie direkt in Maschinencode programmieren, sparen Sie viel Zeit, weil der Übersetzungsvorgang vor und während des Programmablaufs entfällt. Die Schwierigkeit liegt darin, daß Maschinensprachenprogramme aus schier endlosen Zahlenkolonnen bestehen.

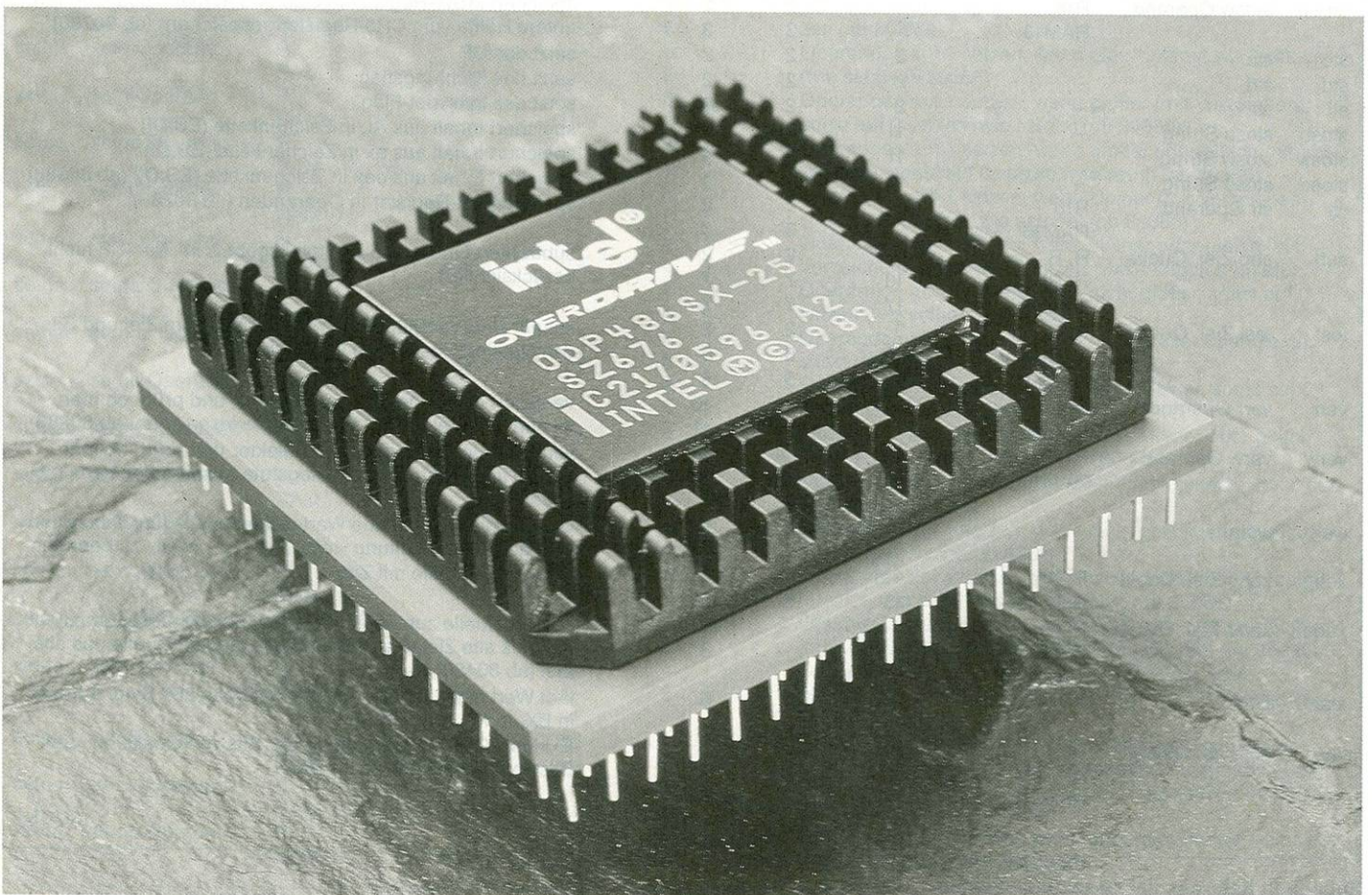
Aus diesem Grund haben sich die Entwickler von Mikroprozessoren zu allen Maschinencodekommandos kurze, aber verständliche Bezeichnungen einfallen lassen. Diese Kurzbezeichnungen – auch Assembler-Befehle genannt – vereinfachen den Umgang mit den Prozessoren erheblich. Um die Assembler-Kommandos wieder in Maschinencode umzuwandeln, ist der Einsatz eines Assemblers erforderlich.

Wenn Sie einen mathematischen Coprozessor Ihr eigen nennen, umgehen Sie viele komplexe Berechnungen, indem Sie den Coprozessor direkt an

sprechen. Allerdings bietet kaum eine Publikation eine übersichtliche Darstellung der Coprozessoranweisungen. Darüber hinaus unterscheiden sich die Befehlssätze der verschiedenen Prozessoren (8087, 80287 und 80387).

Aus diesem Grund listet die Tabelle alle Befehle der arithmetischen Coprozessoren auf. Beachten Sie bitte, daß Sie keine Anweisungen der moderneren Prozessoren benutzen, wenn Ihr Programm auch unter den älteren Modellen lauffähig bleiben soll.

(Markus Zietlow/bm)

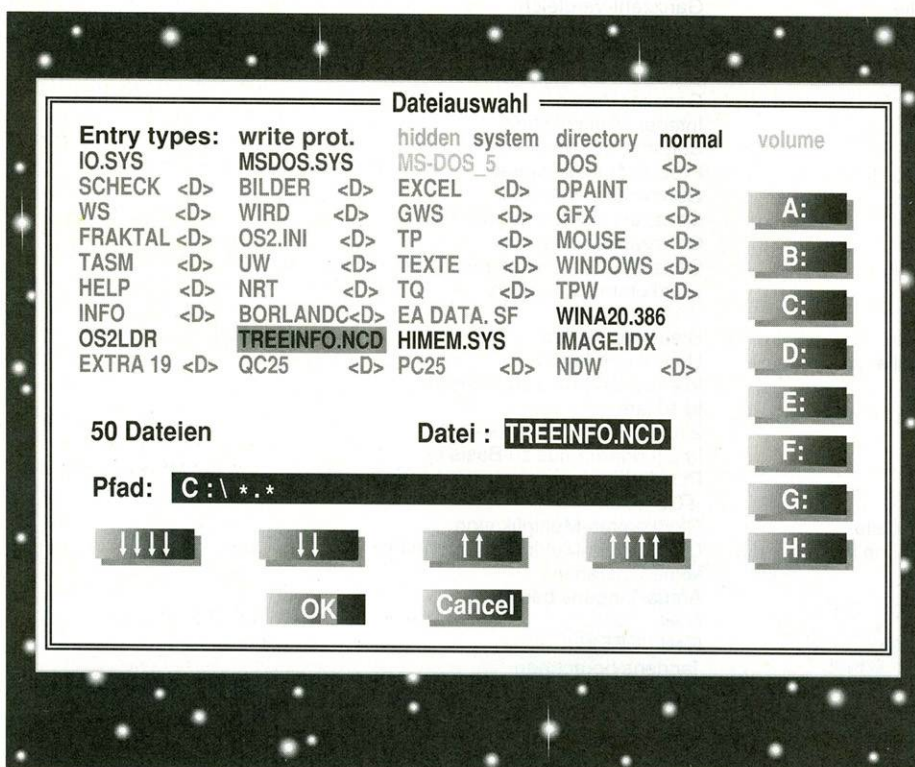


8087	80287	80387	Befehl	Funktion
X	X	X	F2XM1	2+X - 1
X	X	X	FABS	Absolutbetrag
X	X	X	FADD Ziel, Quelle	Gleitkomma-Addition
X	X	X	FADDP Ziel, Quelle	Gleitkomma-Addition und Pop
X	X	X	FBLD Quelle	gepackte Dezimalzahl (BCD) laden
X	X	X	FBSTP	gepackte Dezimalzahl (BCD) speichern und Pop
X	X	X	FCHS	Vorzeichenwechsel
X	X	X	FCLEX/FNCLEX	Exceptions löschen
X	X	X	FCOM Quelle	Gleitkomma-Vergleich
X	X	X	FCOMP Quelle	Gleitkomma-Vergleich und Pop
X	X	X	FCOMPP	Gleitkomma-Vergleich und zweifacher Pop
		X	FCOS	Kosinus
X	X	X	FDECSTP	Stapelzeiger vermindern
X	ignoriert	ignoriert	FDISI/FNDISI	Interrupts sperren
X	X	X	FDIV Ziel, Quelle	Gleitkomma-Division
X	X	X	FDIVP Ziel, Quelle	Gleitkomma-Division und Pop
X	X	X	FDIVR Ziel, Quelle	Gleitkomma-Division und Kehrwert
X	X	X	FDIVRP Ziel, Quelle	Gleitkomma-Division, Kehrwert und Pop
X	ignoriert	ignoriert	FENI/FNENI	Interrupts freigeben
X	X	X	FFREE Ziel	Register freigeben
X	X	X	FIADD Quelle	Ganzzahl-Addition
X	X	X	FICOM Quelle	Ganzzahl-Vergleich
X	X	X	FICOMP Quelle	Ganzzahl-Vergleich und Pop
X	X	X	FIDIV Quelle	Ganzzahl-Division
X	X	X	FIDIVR Quelle	Ganzzahl-Division Kehrwert
X	X	X	FILD Quelle	Ganzzahl laden
X	X	X	FIMUL Quelle	Integer-Multiplikation
X	X	X	FINCSTP	Stapelzeiger erhöhen
X	X	X	FINIT/FNINIT	Prozessor initialisieren
X	X	X	FIST Ziel	Ganzzahl speichern
X	X	X	FISTP Ziel	Ganzzahl speichern und Pop
X	X	X	FISUB Quelle	Ganzzahl-Subtraktion
X	X	X	FISUBR Quelle	Ganzzahl-Subtraktion mit Vorzeichenwechsel
X	X	X	FLD Quelle	Gleitkommazahl laden
X	X	X	FLD1	+1.0 laden
X	X	X	FLDCW Quelle	Steuerwort laden
X	X	X	FLDENV Quelle	Umgebung laden
X	X	X	FLDL2E	ld e (Logarithmus zu Basis 2)
X	X	X	FLDL2T	ld 10 laden
X	X	X	FLDLG2	2 (Logarithmus zu Basis 10)
X	X	X	FLDLN2	ln 2 (Logarithmus zu Basis e)
X	X	X	FLDPI	Pi laden
X	X	X	FLDZ	+0.0 laden
X	X	X	FMUL Ziel, Quelle	Gleitkomma-Multiplikation
X	X	X	FMULP Ziel, Quelle	Gleitkomma-Multiplikation und Pop
X	X	X	FNOP	keine Operation
X	X	X	FPATAN	Arcus-Tangens berechnen
X	X	X	FPREM	Rest
		X	FPREM1	Rest (IEEE)
X	X	X	FPTAN	Tangens berechnen
X	X	X	FRNDINT	auf Ganzzahl runden
X	X	X	FRSTOR Quelle	gesicherten Status zurückholen
X	X	X	FSAVE/FNSAVE Ziel	Status sichern
X	X	X	FSCALE	skalieren
	X	X	FSETPM	in geschützten Modus eintreten («protected mode»)
		X	FSIN	Sinus
		X	FSINCOS	Sinus und Kosinus
X	X	X	FSQRT	Quadratwurzel
X	X	X	FST Ziel	Gleitkomma-Zahl speichern
X	X	X	FSTCW/FNSTCW Ziel	Steuerwort speichern
X	X	X	FSTENV/FNSTENV Ziel	Umgebung speichern
X	X	X	FSTP Ziel	Gleitkomma-Zahl speichern und Pop
X	X	X	FSTSWel/FNSTSW Ziel	Statuswort speichern
X	X	X	FSUB Ziel, Quelle	Gleitkomma-Zahl subtrahieren
X	X	X	FSUBP Ziel, Quelle	Gleitkomma-Zahl subtrahieren und Pop
X	X	X	FSUBR Ziel, Quelle	Gleitkomma-Subtraktion mit Vorzeichenwechsel
X	X	X	FSUBRP Ziel, Quelle	Gleitkomma-Subtraktion mit Vorzeichenwechsel und Pop
X	X	X	FTST	oberstes Stapelelement auf +0.0 prüfen
		X	FUCOM	ungeordneter Vergleich
		X	FUCOMP	ungeordneter Vergleich und Pop
		X	FUCOMPP	ungeordneter Vergleich und zweifacher Pop
X	X	X	FWAIT	warten, solange 8087 beschäftigt
X	X	X	FXAM	oberstes Stapelelement prüfen
X	X	X	FXCH Ziel	Register tauschen
X	X	X	FXTRACT	Exponent und Mantisse extrahieren
X	X	X	FYL2X	y x ld x
X	X	X	FYL2XP1	y x ld 2(x+1)

Die Vielfalt des Befehlssatzes der arithmetischen Coprozessoren gibt Aufschluß über ihre Leistungsfähigkeit.

Die Qual der Wahl

Die Assembler-Programmierung ist ohne Zweifel bis heute eine Herausforderung für jeden Programmierer. Sehr häufig stehen Sie vor der Aufgabe, aus Ihrer Anwendung heraus eine Datei anzuwählen. Also schreiben Sie: »Bitte geben Sie den Namen ein:«. Und er will Ihnen einfach nicht mehr einfallen. Wenn Sie das kennen, wird Ihnen die hier vorgestellte Dateiauswahlbox eine echte Hilfe sein.



Mit Dateiauswahlboxen (oder neu-deutsch: Fileselektorboxen) greifen Sie komfortabel auf Dateien in beliebigen Verzeichnissen auf allen angeschlossenen Laufwerken zu. In der Nach-Windows-Ära ist dieser Komfort geradezu zum Standard geworden. Zum Einsatz in eigenen Programmen ist also ein Befehl wie etwa »call File-Select« wünschenswert, der Dateinamen und Pfad des gewählten Programms zurückliefert. Das Programm »box.asm« (Listing) stellt Ihnen genau diesen Befehl zur Verfügung.

• Daten und Strukturen

Das grundlegende Element zur Erfassung zusammengehöriger Daten ist die Struktur, manchmal auch Datenverbund genannt. Strukturen heißen

in Pascal »records« und in C, genau wie in Assembler, »struct«. Strukturen werden mit dem Befehl »struct« eingeleitet und mit »ends« abgeschlossen. Dazwischen definieren Sie beliebige Variablen. Diese bestehen aus der Angabe des Feldnamens und des Datentyps (Byte oder Word). Aus diesem Typ leiten Sie nun beliebig viele weitere Strukturen ab. Im Listing wird die Struktur »Window« definiert und in Zeile 1887 eine Struktur mit dem Namen »LoadWindow« abgeleitet. Auf die Elemente einer Struktur greifen Sie mit dem ».«-Operator zu. Die Befehlsfolge

```
lea bx, LoadWindow
mov cl, [bx.Win_TopEdge]
```

lädt den Abstand des Fensters vom oberen Bildschirmrand in das CL-Register.

• Das Fenstersystem

Die Dateiauswahlbox entstand im Laufe der Entwicklung eines mehrsegmentigen ASCII-Texteditors. Sie bedient sich eines einfachen Fenstersystems.

Alle Daten, die ein Fenster charakterisieren, definieren Sie in der grundlegenden Struktur »window«, deren Felder in Tabelle 1 erklärt sind.

Das erste Element der Fensterstruktur enthält einen Zeiger auf eine weitere Fensterstruktur. Diese enthält wiederum einen Zeiger auf eine Struktur und so weiter, so daß Sie mit dieser verketteten Liste beliebig viele Strukturen mit einem einzigen Aufruf auswerten können. Die Fensterstruktur enthält neben den Daten für die Abmessungen des Fensters weitere Zeiger auf Strukturen für die Liste der anwählbaren Knöpfe, die Ränder und die Texte. Der Zeiger »Win_Back« zeigt auf den Speicherbereich, in den der Hintergrund gerettet wird, sobald Sie das Fenster öffnen. Hat das Feld »Win_Move« einen Wert ungleich »0«, so wird in der linken oberen Ecke des Fensters ein Schließfeld angezeigt. Sobald Sie das Schließfeld anklicken, wird die Dateiauswahlbox geschlossen.

Die Felder »Win_MouseX«, »Win_MouseY« und »Win_Move« werden nicht ausgewertet.

Das Prinzip der verketteten Listen findet auch in allen weiteren Strukturen Verwendung. Mit der Struktur »Border« (Tabelle 2) definieren Sie Rahmen innerhalb eines Fensters. Die Begrenzungen geben Sie dabei relativ zu den Fensterkoordinaten an. Das Feld »Bor_Typ« bestimmt den Rahmen-Typ: Ist es »0«, wird ein einfacher Rahmen gezeichnet, bei »1« ein doppelter. Mit »Bor_FrontPen« und »Bor_BackPen« legen Sie die Farbre-gister für Vordergrund- und Hintergrundfarbe des Rahmens fest.

Die Ausgabe von Text in Fenstern wird durch die beiden Strukturen »Text« und »IText« unterstützt (Tabelle 3 und 4). »Text« enthält einen Zeiger auf eine Zeichenkette mit Attributen. Jedem auszugebenden Zeichen muß ein Attributbyte folgen. Bit 0 bis 2 des Attributs legen die Farbre-gister für die Vordergrundfarbe fest, Bit 4 bis 6 die für die Hintergrundfarbe. Ein gesetztes Bit 3 läßt die Farben intensiv erscheinen, und ein gesetztes Bit 7 läßt das Zeichen blinken. Die Belegung der Farbre-gister mit den Farbwerten ist in der Tabelle »Farben« definiert. Die Struktur »IText« verwenden Sie zur Darstellung von Text mit konstantem Attribut.

Zu einem Fenster können auch Knöpfe gehören. Diese werden mit der Struktur »Button« (Tabelle 5) definiert. Sie enthält neben dem Zeiger auf die weiteren Knopf-Strukturen In-

formationen über die Position des Knopfes innerhalb des Fensters und seine Abmessungen. Der Zeiger »But_Text« zeigt auf eine Zeichenkette mit Attributen, die die Beschriftung des Knopfes definiert. Die Knöpfe werden pseudo-dreidimensional dargestellt.

• Der Programmablauf

Die Prozedur »Open_Window« wertet alle Strukturen aus. Sie sorgt dafür, daß die Elemente des Fensters (Ränder, Knöpfe und Texte – sofern vorhanden) dargestellt werden. Leiten Sie also eine Fensterstruktur ab, definieren Sie die einzelnen Felder, Zeiger und weitere gewünschte Strukturen und bringen Sie das Fenster mit

```
lea bx, EigeneStruktur
call Open_Window
```

auf den Bildschirm. Sodann warten Sie in einer Schleife das Eintreten einer Mausektion ab. Bei einem Druck auf die linke Maustaste wird die Prozedur »GetMessage« aufgerufen. Sie

holt die Mausposition ab und sieht nach, über welchem Fensterelement Sie die linke Maustaste gedrückt haben. Bei einem gültigen Element (Knopf oder Schließfeld) wird die Kennung des Elements zurückgegeben, so daß Sie entsprechend reagieren können.

Fensterelemente werden mit Zeigern verkettet

Als Kennung wird dabei das Label verwendet, mit dem Sie die Beschriftung des Knopfes definiert haben.

Die Verzeichniseinträge werden in vier Spalten dargestellt. Deshalb überprüfen Sie noch, ob die linke Maustaste in einer dieser Spalten gedrückt wurde. Falls ja, wird der Eintrag unter dem Mauscursor bestimmt, erleuchtet und in dem Feld »Dateiname:« angezeigt. Ist der Eintrag ein Verzeichnis (rote Schrift), wechselt das Programm in dieses Verzeichnis. Der Pfad wird

in dem Feld »Pfad:« angezeigt. Mit den vier Knöpfen unterhalb der Dateinamen bewegen Sie sich durch Ihre Einträge.

Drücken Sie auf den Knopf »OK« oder auf das Schließfeld, wird die Prozedur »Close_Window« aufgerufen und die Dateiauswahlbox geschlossen. Der absolute Dateiname wird daraufhin aus Pfad und relativem Dateinamen zusammengesetzt und ausgegeben.

Die Einträge eines Verzeichnisses werden mit den MS-DOS-Funktionen 4Ehex (FindFirst) und 4Fhex (FindNext) geholt. Die Funktion 4Ehex erwartet in »DS:DX« die Adresse eines Pfadnamens und im CX-Register das Attribut des gewünschten Eintrags. Ist ein gültiger Eintrag gefunden, wird dieser inklusive Attribut in dem Puffer »AllBuff« gespeichert. Mit der Funktion 4Fhex werden dann weitere Einträge des gleichen Typs gesucht und ebenfalls gespeichert. Die Anzahl der Einträge ist auf 500 beschränkt, mehr werden nicht angezeigt. (bm)

Name	Typ	Bedeutung
Win_LeftEdge	Byte	Abstand vom linken Rand
Win_TopEdge	Byte	Abstand vom oberen Rand
Win_Width	Byte	Fensterbreite
Win_Height	Byte	Fensterhöhe
Win_Pen	Byte	Attribut
Win_MouseX	Word	Mausposition innerhalb des Fensters
Win_MouseY	Word	
Win_Button	Word	Zeiger auf die Buttonstruktur
Win_Text	Word	Zeiger auf Textstruktur (verschiedene Attribute)
Win_IText	Word	Zeiger auf Textstruktur (konstantes Attribut)
Win_Border	Word	Zeiger auf Borderstruktur
Win_Close	Byte	0 = kein Close-Button, 1 = mit
Win_Move	Byte	0 = kein Move-Button, 1 = mit
Win_Input	Word	Zeiger auf Input-Struktur (Stringeingabe)
Win_Back	Word	Segmentadresse des Hintergrundpuffers

Tabelle 1. Die Felder der Fensterstruktur

Name	Typ	Bedeutung
Bor_Next	Word	Zeiger auf nächste Struktur
Bor_LeftEdge	Byte	linker Rand relativ zum Fenster
Bor_TopEdge	Byte	oberer Rand relativ zum Fenster
Bor_Width	Byte	Breite
Bor_Height	Byte	Höhe
Bor_FrontPen	Byte	Vordergrundfarbe
Bor_BackPen	Byte	Hintergrundfarbe
Bor_Typ	Byte	Rahmentyp: 0 = einfach, 1 = doppelt

Tabelle 2. Mit dieser Struktur definieren Sie Umrandungen innerhalb von Fenstern.

Name	Typ	Bedeutung
Text_Next	Word	Zeiger auf nächste Textstruktur
Text_LeftEdge	Byte	linker Rand relativ zum Fenster
Text_TopEdge	Byte	oberer Rand relativ zum Fenster
Text_Text	Word	Zeiger auf nullterminierte Zeichenkette mit Attributen

Tabelle 3. Die Struktur zur Definition von Texten in Fenstern mit verschiedenen Attributen.

Name	Typ	Bedeutung
IText_Next	Word	Zeiger auf nächste Textstruktur
IText_LeftEdge	Byte	linker Rand relativ zum Fenster
IText_TopEdge	Byte	oberer Rand relativ zum Fenster
IText_Attribut	Byte	Attribut
IText_Text	Word	Zeiger auf Zeichenkette

Tabelle 4. Mit dieser Struktur definieren Sie Texte mit konstanten Attributen. In der Definition der Zeichenkette muß auf jedes einzelne Zeichen das Attribut-Byte folgen.

Name	Typ	Bedeutung
But_Next	Word	Zeiger auf nächste Buttonstruktur
But_LeftEdge	Byte	linker Rand relativ zum Fenster
But_TopEdge	Byte	oberer Rand relativ zum Fenster
But_Width	Byte	Breite des Knopfes
But_Height	Byte	Höhe des Knopfes
But_Text	Word	Zeiger auf Knopfbeschriftung mit Attributen

Tabelle 5. So sieht die verkettete Struktur für die Definition von Knöpfen aus. Diese werden pseudo-dreidimensional mit einem Schatten dargestellt.

```
[SP3] 1: ;Programm: filebox.asm
[8Y6] 2: ;Funktion: Fileselektorbox im Textmodus
[8G2] 3: ;Sprache: TASM
[ZF4] 4: ;Autor: Burkhard Müller
[7X4] 5: ; (c) 1992 DMV Widuch GmbH & Co. KG
[000] 6:
[XY1] 7: .MODEL SMALL
[FS1] 8: .stack 200h
[FT0] 9: .CODE
[XX0] 10: JUMPS
[BW3] 11: ADR MACRO Zeile, Spalte
[MH1] 12: mov ah, 0
[LD2] 13: mov al, Zeile
[RP1] 14: mov bl, 160
[NH1] 15: mov bh, 0
[371] 16: mul bl
[MG2] 17: mov bl, Spalte
[NH1] 18: mov bh, 0
[8H1] 19: shl bx, 1
```

Mit »box.asm« steht Ihnen eine Dateiauswahlbox zum Einbau in eigene Programme zur Verfügung.


```
[LL1] 20: add ax,bx
[HP0] 21: ENDM
[7V5] 22: ; konstantes Attribut mit Hardware
[7E7] 23: WRITE1 MACRO Attribut,Anz,Zeile,Spalte,Text
[8C1] 24: LOCAL D22
[Y91] 25: push es
[443] 26: ADR Zeile,Spalte
[NP1] 27: mov di,ax
[XD3] 28: add di,ScreenOffset
[SF2] 29: mov ax,Screen
[AR1] 30: mov es,ax
[962] 31: mov si,Text
[4W1] 32: mov cl,Anz
[PH1] 33: mov ch,0
[TV2] 34: D22: mov bl,[si]
[VM2] 35: mov es:[di],bl
[Y41] 36: inc di
[D51] 37: inc si
[GH5] 38: mov BYTE PTR es:[di],Attribut
[Y41] 39: inc di
[FB1] 40: LOOP D22
[S71] 41: pop es
[HP0] 42: ENDM
[GE7] 43: WRITE4 MACRO Attribut,Anz,Zeile,Spalte,Text
[5C1] 44: LOCAL D21
[Y91] 45: push es
[S91] 46: push si
[991] 47: push di
[443] 48: ADR Zeile,Spalte
[NP1] 49: mov di,ax
[XD3] 50: add di,ScreenOffset
[SF2] 51: mov ax,Screen
[AR1] 52: mov es,ax
[7A3] 53: mov si,OFFSET Text
[4W1] 54: mov cl,Anz
[PH1] 55: mov ch,0
[NV2] 56: D21: mov bl,[si]
[VM2] 57: mov es:[di],bl
[Y41] 58: inc di
[D51] 59: inc si
[GH5] 60: mov BYTE PTR es:[di],Attribut
[Y41] 61: inc di
[CB1] 62: LOOP D21
[371] 63: pop di
[K71] 64: pop si
[S71] 65: pop es
[HP0] 66: ENDM
[EE6] 67: ; testen, ob Maus in angegebenen Bereich
[557] 68: MAUSIN MACRO xlinks,xrechts,yoben,yunten
[GF3] 69: LOCAL R1,R2,R3,R4,R5,R6
[CG2] 70: mov ax,MausX
[9G2] 71: cmp ax,xlinks
[Z21] 72: jae R1
[R51] 73: jmp R5
[PD3] 74: R1: cmp ax,xrechts
[431] 75: jbe R2
[R51] 76: jmp R5
[M13] 77: R2: mov ax,MausY
[0D2] 78: cmp ax,yoben
[331] 79: jae R3
[R51] 80: jmp R5
[2C3] 81: R3: cmp ax,yunten
[831] 82: jbe R4
[R51] 83: jmp R5
[5M3] 84: R4: stc ;Carry setzen
[T51] 85: jmp R6
[F81] 86: R5: clc
[FK1] 87: R6: ENDM
[NN2] 88: FileSelect Proc
[UA2] 89: mov ax,@data
[Y91] 90: mov ds,ax
[AR1] 91: mov es,ax
[063] 92: ; DTA-Buffer setzen
[4W3] 93: mov dx,OFFSET DTABuff
[9S1] 94: mov ah,lah
[SB1] 95: int 21h
[ZA2] 96: call ReadMode
[DK2] 97: mov OldMode,ax
[AY2] 98: mov ax,3 ; 80*25 16
[S42] 99: call SetMode
[NW1] 100: call SetDAC
[HX1] 101: call SetCol
[UA2] 102: mov ax,@data
[Y91] 103: mov ds,ax
[AR1] 104: mov es,ax
[AA3] 105: lea bx,Load_Window
[Z22] 106: call Open_Window
[UA2] 107: mov ax,@data
[Y91] 108: mov ds,ax
[AR1] 109: mov es,ax
[5W2] 110: call ReadEntries
[383] 111: call PrintEntries
[3H2] 112: call PrintInfo
[062] 113: call MausEin
[2G2] 114: WaitMausClick:
[262] 115: call MausKey
[RX1] 116: mov ax,Key
[EG1] 117: cmp al,1
```

```
[Z51] 118: jz FB1
[693] 119: jmp WaitMausClick
[754] 120: FB1: lea bx,Load_Window
[8Z2] 121: call GetMessage
[UA2] 122: mov ax,@data
[Y91] 123: mov ds,ax
[AR1] 124: mov es,ax
[PU2] 125: lea ax,OKButton
[FN1] 126: cmp bp,ax
[6C1] 127: jnz FB2
[853] 128: jmp ProgrammEnde
[0T3] 129: FB2: lea ax,UP1Button
[FN1] 130: cmp bp,ax
[9C1] 131: jnz FB3
[6R1] 132: call Hoch2
[693] 133: jmp WaitMausClick
[5T3] 134: FB3: lea ax,UP2Button
[FN1] 135: cmp bp,ax
[CC1] 136: jnz FB4
[5R1] 137: call Hoch1
[693] 138: jmp WaitMausClick
[DR3] 139: FB4: lea ax,DN1Button
[FN1] 140: cmp bp,ax
[FC1] 141: jnz FB5
[R42] 142: call Runter1
[693] 143: jmp WaitMausClick
[KR3] 144: FB5: lea ax,DN2Button
[FN1] 145: cmp bp,ax
[KC1] 146: jnz FB6
[U42] 147: call Runter2
[693] 148: jmp WaitMausClick
[LZ4] 149: FB6: MausIn 3,14,4,12 ; 1. Spalte
[Y91] 150: jnc FB7
[RZ2] 151: mov FileSpalte,0
[UT2] 152: mov FileStart,3
[8X2] 153: call FileMitMaus
[693] 154: jmp WaitMausClick
[Z05] 155: FB7: MausIn 16,27,4,12 ; 2. Spalte
[Y91] 156: jnc FB8
[E13] 157: mov FileSpalte,13
[C03] 158: mov FileStart,16
[8X2] 159: call FileMitMaus
[693] 160: jmp WaitMausClick
[R05] 161: FB8: MausIn 29,40,4,12 ; 3. Spalte
[Z91] 162: jnc FB9
[N13] 163: mov FileSpalte,26
[V03] 164: mov FileStart,29
[8X2] 165: call FileMitMaus
[693] 166: jmp WaitMausClick
[E15] 167: FB9: MausIn 42,53,4,12 ; 4. Spalte
[BF1] 168: jnc FB11
[W13] 169: mov FileSpalte,39
[503] 170: mov FileStart,42
[8X2] 171: call FileMitMaus
[693] 172: jmp WaitMausClick
[PB3] 173: FB11: lea ax,AButton
[FN1] 174: cmp bp,ax
[BK1] 175: jnz FB12
[B62] 176: call ButtonA
[693] 177: jmp WaitMausClick
[UB3] 178: FB12: lea ax,BButton
[FN1] 179: cmp bp,ax
[FK1] 180: jnz FB13
[E62] 181: call ButtonB
[693] 182: jmp WaitMausClick
[YB3] 183: FB13: lea ax,CButton
[FN1] 184: cmp bp,ax
[LK1] 185: jnz FB14
[H62] 186: call ButtonC
[693] 187: jmp WaitMausClick
[2C3] 188: FB14: lea ax,DButton
[FN1] 189: cmp bp,ax
[RK1] 190: jnz FB15
[M62] 191: call ButtonD
[693] 192: jmp WaitMausClick
[6C3] 193: FB15: lea ax,EButton
[FN1] 194: cmp bp,ax
[VK1] 195: jnz FB16
[R62] 196: call ButtonE
[693] 197: jmp WaitMausClick
[AC3] 198: FB16: lea ax,FButton
[FN1] 199: cmp bp,ax
[ZK1] 200: jnz FB17
[U62] 201: call ButtonF
[693] 202: jmp WaitMausClick
[EC3] 203: FB17: lea ax,GButton
[FN1] 204: cmp bp,ax
[3L1] 205: jnz FB18
[X62] 206: call ButtonG
[693] 207: jmp WaitMausClick
[KC3] 208: FB18: lea ax,HButton
[FN1] 209: cmp bp,ax
[7L1] 210: jnz FB19
[072] 211: call ButtonH
[693] 212: jmp WaitMausClick
[UR3] 213: FB19: lea ax,IButton
[FN1] 214: cmp bp,ax
[AL1] 215: jnz FB29
[SR4] 216: Abbruch:lea bx,Load_Window
```



```

[4A3] 217:      call    Close_Window
[NM2] 218:      mov     ax,OldMode
[S42] 219:      call    SetMode
[524] 220:      mov     dx,Offset NichtGewaelht
[SK1] 221:      mov     ah,9
[SB1] 222:      int     21h
[9A3] 223:      mov     ah,4ch ; fertig
[SB1] 224:      int     21h
[GC4] 225: FB29:      lea     ax,Close_Button
[FN1] 226:      cmp     bp,ax
[9K1] 227:      jnz     FB30
[142] 228:      jmp     Abbruch
[BL3] 229: FB30:      lea     ax,OKButton
[FN1] 230:      cmp     bp,ax
[DK2] 231:      jnz     VonVorne
[853] 232:      jmp     ProgrammEnde
[GV4] 233: VonVorne: jmp     WaitMausClick
[Y92] 234: ProgrammEnde:
[AA3] 235:      lea     bx,Load_Window
[4A3] 236:      call    Close_Window
[UA2] 237:      mov     ax,@data
[9R1] 238:      mov     ds,ax
[AR1] 239:      mov     es,ax
[NM2] 240:      mov     ax,OldMode
[S42] 241:      call    SetMode
[AV4] 242:      ; Dateinamen zusammensetzen
[M63] 243:      call    BuildFileName
[LW3] 244:      mov     dx,Offset Meldung
[SK1] 245:      mov     ah,9
[SB1] 246:      int     21h
[R83] 247:      mov     dx,Offset Pfad
[SK1] 248:      mov     ah,9
[SB1] 249:      int     21h
[9A3] 250:      mov     ah,4ch ; fertig
[SB1] 251:      int     21h
[NN2] 252: FileSelect ENDP
[1Y1] 253: SetDAC PROC
[UA2] 254:      mov     ax,@data
[AR1] 255:      mov     es,ax
[2S1] 256:      mov     al,12h
[SR1] 257:      mov     ah,10h
[N92] 258:      mov     bx,1 ; Nr.
[GM1] 259:      mov     cx,16
[TR3] 260:      mov     dx,OFFSET Farben
[NB1] 261:      int     10h
[3P0] 262:      ret
[1Y1] 263: SetDAC ENDP
[YL5] 264:      ; 16 Farbregister aus 64 auswählen
[RZ1] 265: SetCol PROC
[UA2] 266:      mov     ax,@data
[AR1] 267:      mov     es,ax
[MT3] 268:      mov     dx,OFFSET FarbTab
[YR1] 269:      mov     al,02h
[SR1] 270:      mov     ah,10h
[NB1] 271:      int     10h
[3P0] 272:      ret
[RZ1] 273: SetCol ENDP
[472] 274: ReadMode PROC
[LS5] 275:      mov     ah,0fh;Bildschirmmodus lesen
[NB1] 276:      int     10h
[3P0] 277:      ret
[262] 278: ReadMode ENDP
[562] 279: SetMode PROC
[4W5] 280:      mov     ah,00h;Bildschirmmodus setzen
[NB1] 281:      int     10h
[3P0] 282:      ret
[442] 283: SetMode ENDP
[533] 284: BuildFileName PROC
[0C5] 285:      ; Anzahl Zeichen bis zum letzten
[2T4] 286:      ; Backslash im Pfad bestimmen
[TY1] 287:      lea     di,Pfad
[1D4] 288:      add     di,44; letztes Zeichen
[1R1] 289:      xor     dl,dl
[PM1] 290: SearchBS2:
[WV3] 291:      cmp     BYTE PTR [di],'\'
[PC3] 292:      jz      BackSlashFound2
[T31] 293:      dec     di
[567] 294:      ; Anzahl Zeichen nach dem letzten Backslash
[451] 295:      inc     dl
[GH2] 296:      LOOP    SearchBS2
[VS4] 297:      jmp     BSF2Ende;kein Backslash
[WP2] 298: BackSlashFound2:
[Y41] 299:      inc     di
[F85] 300:      ; Filenamen hinter Pfad kopieren
[YN2] 301:      lea     si,FileName
[EL1] 302:      mov     cx,6
[3W1] 303:      rep     movsw
[6B4] 304:      mov     BYTE PTR es:[di],'$'
[T42] 305: BSF2Ende: ret
[623] 306: BuildFileName ENDP
[GZ2] 307: FileMitMaus PROC
[8L2] 308:      mov     ax,AllBuff
[AR1] 309:      mov     es,ax
[LT3] 310:      call    FM_Invert_Entry
[915] 311:      mov     DirFlag,0 ; normales file
[GG2] 312:      mov     ax,MausY
[RH4] 313:      sub     ax,4 ; 4 Zeilen von oben
[MR5] 314:      ; 4 (Einträge/Zeile) * 13 (Z/Eintrag)
[AW1] 315:      mov     bx,4*13

```

```

[V71] 316:      mul     bx
[Y63] 317:      mov     dl,FileSpalte
[RH1] 318:      mov     dh,0
[ZP3] 319:      add     ax,dx ;plus Spalte
[R55] 320:      ;Zeiger auf gewählten Eintrag
[T73] 321:      add     ax,StartEntry
[4R1] 322:      mov     si,ax
[6N2] 323:      call    CheckEntry
[NA2] 324:      cmp     DirFlag,1
[BH2] 325:      jnz     CopyFile
[RL3] 326:      jmp     FileMitMausEnde
[WK1] 327: CopyFile:
[Y55] 328:      ; Eintrag ist normaler Filename
[TX3] 329:      mov     di,OFFSET FileName
[EL1] 330:      mov     cx,6
[BM1] 331: FMM_Loop:
[FP2] 332:      mov     ax,es:[si]
[HN2] 333:      mov     ds:[di],ax
[D51] 334:      inc     si
[D51] 335:      inc     si
[Y41] 336:      inc     di
[Y41] 337:      inc     di
[BM2] 338:      LOOP    FMM_Loop
[2U3] 339:      WRITE4 1,12,14,40,FileName
[PX2] 340: FileMitMausEnde:
[3P0] 341:      ret
[FX2] 342: FileMitMaus ENDP
[UW3] 343: FM_Invert_Entry PROC
[Y91] 344:      push    es
[GG2] 345:      mov     ax,MausY
[HR1] 346:      mov     dx,160
[X71] 347:      mul     dx
[D53] 348:      mov     bl,FileStart
[NH1] 349:      mov     bh,0
[8H1] 350:      shl     bx,1
[LL1] 351:      add     ax,bx
[RS4] 352:      mov     si,ax ; Adresse im Screen
[OC3] 353:      cmp     ax,File_Merken
[FE3] 354:      jnz     FM_New_Entry
[H52] 355:      jmp     FM_Ende
[UH2] 356: FM_New_Entry:
[S91] 357:      push    si
[4A1] 358:      push    ax
[372] 359:      call    MausAus
[8E3] 360:      call    FM_Entleuchten
[Y71] 361:      pop     ax
[K71] 362:      pop     si
[ZA3] 363:      mov     File_Merken,ax
[BK3] 364:      add     si,1 ; Offset Dest
[SF2] 365:      mov     ax,Screen
[N54] 366:      mov     es,ax ; Segment Dest
[CM1] 367:      mov     cx,12
[PZ2] 368: FM_Invert_Loop:
[324] 369:      mov     al,BYTE PTR es:[si]
[FF2] 370:      and     al,10001111b
[VB2] 371:      or      al,00100000b
[MC4] 372:      mov     BYTE PTR es:[si],al
[D51] 373:      inc     si
[D51] 374:      inc     si
[4M3] 375:      LOOP    FM_Invert_Loop
[062] 376:      call    MausEin
[AF1] 377: FM_Ende:
[S71] 378:      pop     es
[3P0] 379:      ret
[TU3] 380: FM_Invert_Entry ENDP
[9G3] 381: FM_Entleuchten PROC
[6D3] 382:      mov     si,File_Merken
[KK1] 383:      or      si,si
[XD3] 384:      jnz     FM_Not_First
[3P0] 385:      ret
[NH2] 386: FM_Not_First:
[D51] 387:      inc     si
[SF2] 388:      mov     ax,Screen
[AR1] 389:      mov     es,ax
[CM1] 390:      mov     cx,12
[TN1] 391: FM_ELoop:
[324] 392:      mov     al,BYTE PTR es:[si]
[FF2] 393:      and     al,10001111b
[XB2] 394:      or      al,01010000b
[MC4] 395:      mov     BYTE PTR es:[si],al
[D51] 396:      inc     si
[D51] 397:      inc     si
[TK2] 398:      LOOP    FM_ELoop
[UA2] 399:      mov     ax,@data
[AR1] 400:      mov     es,ax
[3P0] 401:      ret
[9G3] 402: FM_Entleuchten ENDP
[2P2] 403: CheckEntry PROC
[W56] 404:      ; prüfen, welcher Eintrag ausgewählt
[7U2] 405:      cmp     es:[si], '.'
[ZR2] 406:      jz      ParentFound
[7T2] 407:      ; ' .' nicht nehmen
[R94] 408:      cmp     BYTE PTR es:[si+1],0
[9A2] 409:      jz      DontTake
[5X2] 410:      ; letztes Zeichen

```

Mit »box.asm« steht Ihnen eine Dateiauswahlbox zum Einbau in eigene Programme zur Verfügung.


```
[UL4] 411:      cmp     BYTE PTR es:[si+11], '>'
[DR2] 412:      jz      SubDirFound
[HC3] 413:      jmp     CheckEntryEnde
[HK1] 414:      DontTake:
[7C2] 415:      mov     DirFlag, 1
[HC3] 416:      jmp     CheckEntryEnde
[Y52] 417:      SubDirFound:
[UX5] 418:      ; si = Zeiger auf neues Subdirectory
[BH3] 419:      call    SearchBackSlash
[NB5] 420:      inc     di ; 1. Zeichen nach letzten \
[Y14] 421:      mov     cx, 8 ; max. 8 Zeichen
[E32] 422:      SubDirLoop:
[RN2] 423:      mov     al, es:[si]
[KA4] 424:      cmp     al, 0 ; bis zur ersten 0
[K62] 425:      jz      PfadEnde
[TM2] 426:      mov     ds:[di], al
[D51] 427:      inc     si
[Y41] 428:      inc     di
[H57] 429:      ; neuen Subdirectorynamen in Pfad kopieren
[Q52] 430:      LOOP   SubDirLoop
[RF1] 431:      PfadEnde:
[BZ2] 432:      mov     ds:[di], '*'
[GW2] 433:      mov     ds:[di+2], '*'
[955] 434:      mov     BYTE PTR ds:[di+4], 0 ; ASCIIIZ
[FP5] 435:      mov     DirFlag, 1 ; für Subdirectory
[T43] 436:      call    ClearFileName
[BB1] 437:      call    BW2
[PR2] 438:      CheckEntryEnde:
[3P0] 439:      ret
[2P2] 440:      CheckEntry ENDP
[8Y2] 441:      ParentFound PROC
[BH3] 442:      call    SearchBackSlash
[D55] 443:      ; di zeigt auf letztes Backslash
[2L3] 444:      ; Backslash löschen
[KH3] 445:      mov     BYTE PTR [di], 0
[BH3] 446:      call    SearchBackSlash
[QF2] 447:      mov     [di], '*'
[GC2] 448:      mov     [di+2], '*'
[T55] 449:      mov     BYTE PTR [di+4], 0 ; ASCIIIZ
[FP5] 450:      mov     DirFlag, 1 ; für Subdirectory
[T43] 451:      call    ClearFileName
[BB1] 452:      call    BW2
[3P0] 453:      ret
[7W2] 454:      ParentFound ENDP
[D23] 455:      ClearFileName PROC
[1L3] 456:      ; Filenamen löschen
[FN2] 457:      lea     di, FileName
[MK1] 458:      mov     ax, 0
[EL1] 459:      mov     cx, 6
[YV1] 460:      rep     stosw
[372] 461:      call    MausAus
[2U3] 462:      WRITE4 1, 12, 14, 4, FileName
[Q62] 463:      call    MausEin
[3P0] 464:      ret
[E13] 465:      ClearFileName ENDP
[G93] 466:      CopyExtension PROC
[BH3] 467:      call    SearchBackSlash
[Y41] 468:      inc     di
[YU2] 469:      mov     es:[di], '.'
[Y41] 470:      inc     di
[Y41] 471:      inc     di
[EG5] 472:      ; neue Extension in Pfad kopieren
[MD2] 473:      mov     si, BoxAdr
[2L1] 474:      mov     cx, 3
[EV1] 475:      rep     movsb
[T43] 476:      call    ClearFileName
[BB1] 477:      call    BW2
[3P0] 478:      ret
[H83] 479:      CopyExtension ENDP
[1K3] 480:      SearchBackSlash PROC
[1B4] 481:      ; letztes \ im Pfad suchen
[PP5] 482:      ; Rückgabe: di = zeigt auf letztes \
[TY1] 483:      lea     di, Pfad
[NY3] 484:      mov     cx, 44 ; 44 mal suchen
[5F4] 485:      add     di, cx ; letztes Zeichen
[SF1] 486:      SearchBS:
[UV3] 487:      cmp     BYTE PTR [di], '\'
[Q83] 488:      jz      BackSlashFound
[FG1] 489:      ; löschen
[YN3] 490:      mov     BYTE PTR [di], ' '
[T31] 491:      dec     di
[YF2] 492:      LOOP   SearchBS
[2P2] 493:      BackSlashFound:
[3P0] 494:      ret
[QG3] 495:      SearchBackSlash ENDP
[DC2] 496:      ClearPfad PROC
[9B2] 497:      ; Pfad löschen
[T73] 498:      mov     di, Offset Pfad
[UA2] 499:      mov     ax, @data
[AR1] 500:      mov     es, ax
[MK1] 501:      mov     ax, 0
[EM1] 502:      mov     cx, 20
[YV1] 503:      rep     stosw
[3P0] 504:      ret
[EB2] 505:      ClearPfad ENDP
[6C1] 506:      BW2 PROC
[MZ5] 507:      ; nur ausgeben, wenn LMB losgelassen
[E73] 508:      endlos: mov     ax, 0003h
[XB1] 509:      int     33h
```

```
[HL1] 510:      test    bx, 1
[K92] 511:      jz      Ausgeben
[DY1] 512:      jmp     endlos
[3P0] 513:      ret
[UH1] 514:      Ausgeben:
[M33] 515:      WRITE4 1, 40, 16, 12, Pfad
[H23] 516:      call    ClearAllBuff
[5W2] 517:      call    ReadEntries
[372] 518:      call    MausAus
[SS4] 519:      ; mit erstem Eintrag anfangen
[YF4] 520:      mov     StartEntry, WORD PTR 0
[383] 521:      call    PrintEntries
[3H2] 522:      call    PrintInfo
[Q62] 523:      call    MausEin
[3P0] 524:      ret
[5A1] 525:      BW2 ENDP
[872] 526:      ButtonA PROC
[ZC2] 527:      call    ClearPfad
[TY1] 528:      lea     di, Pfad
[C72] 529:      lea     si, PfadA
[2L1] 530:      mov     cx, 3
[3W1] 531:      rep     movsw
[G34] 532:      mov     BYTE PTR es:[di], 0
[T43] 533:      call    ClearFileName
[A32] 534:      mov     drive, 1
[BB1] 535:      call    BW2
[3P0] 536:      ret
[752] 537:      ButtonA ENDP
[C72] 538:      ButtonB PROC
[ZC2] 539:      call    ClearPfad
[TY1] 540:      lea     di, Pfad
[G72] 541:      lea     si, PfadB
[2L1] 542:      mov     cx, 3
[3W1] 543:      rep     movsw
[G34] 544:      mov     BYTE PTR es:[di], 0
[T43] 545:      call    ClearFileName
[D32] 546:      mov     drive, 2
[BB1] 547:      call    BW2
[3P0] 548:      ret
[B52] 549:      ButtonB ENDP
[G72] 550:      ButtonC PROC
[ZC2] 551:      call    ClearPfad
[TY1] 552:      lea     di, Pfad
[M72] 553:      lea     si, PfadC
[2L1] 554:      mov     cx, 3
[3W1] 555:      rep     movsw
[G34] 556:      mov     BYTE PTR es:[di], 0
[T43] 557:      call    ClearFileName
[G32] 558:      mov     drive, 3
[BB1] 559:      call    BW2
[3P0] 560:      ret
[F52] 561:      ButtonC ENDP
[M72] 562:      ButtonD PROC
[ZC2] 563:      call    ClearPfad
[TY1] 564:      lea     di, Pfad
[S72] 565:      lea     si, PfadD
[2L1] 566:      mov     cx, 3
[LH4] 567:      rep     movsw ; ds:si -> es:di
[G34] 568:      mov     BYTE PTR es:[di], 0
[T43] 569:      call    ClearFileName
[L32] 570:      mov     drive, 4
[BB1] 571:      call    BW2
[3P0] 572:      ret
[L52] 573:      ButtonD ENDP
[S72] 574:      ButtonE PROC
[ZC2] 575:      call    ClearPfad
[TY1] 576:      lea     di, Pfad
[W72] 577:      lea     si, PfadE
[2L1] 578:      mov     cx, 3
[3W1] 579:      rep     movsw
[G34] 580:      mov     BYTE PTR es:[di], 0
[T43] 581:      call    ClearFileName
[P32] 582:      mov     drive, 5
[BB1] 583:      call    BW2
[3P0] 584:      ret
[R52] 585:      ButtonE ENDP
[W72] 586:      ButtonF PROC
[ZC2] 587:      call    ClearPfad
[TY1] 588:      lea     di, Pfad
[Q82] 589:      lea     si, PfadF
[2L1] 590:      mov     cx, 3
[3W1] 591:      rep     movsw
[G34] 592:      mov     BYTE PTR es:[di], 0
[T43] 593:      call    ClearFileName
[T32] 594:      mov     drive, 6
[BB1] 595:      call    BW2
[3P0] 596:      ret
[V52] 597:      ButtonF ENDP
[Q82] 598:      ButtonG PROC
[ZC2] 599:      call    ClearPfad
[TY1] 600:      lea     di, Pfad
[482] 601:      lea     si, PfadG
[2L1] 602:      mov     cx, 3
[3W1] 603:      rep     movsw
[G34] 604:      mov     BYTE PTR es:[di], 0
[T43] 605:      call    ClearFileName
[W32] 606:      mov     drive, 7
[BB1] 607:      call    BW2
[3P0] 608:      ret
```



```
[Z52] 609: ButtonG ENDP
[482] 610: ButtonH PROC
[ZC2] 611: call ClearPfad
[TV1] 612: lea di,Pfad
[882] 613: lea si,PfadH
[2L1] 614: mov cx,3
[3W1] 615: rep movsw
[G34] 616: mov BYTE PTR es:[di],0
[T43] 617: call ClearFileName
[Z32] 618: mov drive,8
[BB1] 619: call BW2
[3P0] 620: ret
[362] 621: ButtonH ENDP
[GY2] 622: ClearAllBuff PROC
[8L2] 623: mov ax,AllBuff
[AR1] 624: mov es,ax
[SP1] 625: xor di,di
[5S1] 626: mov ax,' '
[G22] 627: mov cx,500*6
[YV1] 628: rep stosw
[UA2] 629: mov ax,@data
[AR1] 630: mov es,ax
[3P0] 631: ret
[EX2] 632: ClearAllBuff ENDP
[852] 633: Runter1 PROC
[3M3] 634: ; 4 Einträge zurück
[4G3] 635: ; noch 36 Einträge da ?
[UB3] 636: mov ax,StartEntry
[RU1] 637: cmp ax,4*13
[1P1] 638: jae Ru1OK
[RL3] 639: ; nein, dann von vorne
[543] 640: mov StartEntry,0
[383] 641: call PrintEntries
[3P0] 642: ret
[VC4] 643: ; sonst eine Seite weniger
[PC4] 644: RU1OK: sub StartEntry,4*13
[383] 645: call PrintEntries
[3P0] 646: ret
[732] 647: Runter1 ENDP
[C52] 648: Runter2 PROC
[SW2] 649: ; 36 files zurück
[4G3] 650: ; noch 36 Einträge da ?
[UB3] 651: mov ax,StartEntry
[R02] 652: cmp ax,36*13
[4P1] 653: jae Ru2OK
[7P5] 654: ; nein, dann wieder ganz von vorne
[543] 655: mov StartEntry,0
[383] 656: call PrintEntries
[3P0] 657: ret
[VC4] 658: ; sonst eine Seite weniger
[NK4] 659: RU2OK: sub StartEntry,36*13
[383] 660: call PrintEntries
[3P0] 661: ret
[B32] 662: Runter2 ENDP
[CM1] 663: Hoch1 PROC
[7E3] 664: ; 4 Einträge weiter
[YM6] 665: ; nur weitermachen bei mehr als 36+4 files
[563] 666: cmp AnzahlFiles,40
[TD2] 667: jae HolWeiter
[3P0] 668: ret
[4R1] 669: HolWeiter:
[WB3] 670: mov cx,StartEntry
[AS1] 671: add cx,4*13
[PN2] 672: ; Ende der Tabelle
[BA3] 673: mov ax,Anzahlfiles
[CM1] 674: mov bx,13
[V71] 675: mul bx
[R02] 676: sub ax,36*13
[0P1] 677: cmp cx,ax
[EN1] 678: jbe Ho1OK
[XT5] 679: mov StartEntry,ax ; letzte Seite
[383] 680: call PrintEntries
[3P0] 681: ret
[8F4] 682: Ho1OK: mov StartEntry,cx
[383] 683: call PrintEntries
[3P0] 684: ret
[DL1] 685: Hoch1 ENDP
[EM1] 686: Hoch2 PROC
[NF3] 687: ; 36 Einträge weiter
[YM6] 688: ; nur weitermachen bei mehr als 36+4 files
[563] 689: cmp AnzahlFiles,40
[W22] 690: jae Ho2Weiter
[3P0] 691: ret
[8R1] 692: Ho2Weiter:
[WB3] 693: mov cx,StartEntry
[AY1] 694: add cx,36*13
[PN2] 695: ; Ende der Tabelle
[BA3] 696: mov ax,Anzahlfiles
[CM1] 697: mov bx,13
[V71] 698: mul bx
[UE4] 699: sub ax,36*13 ; letzte Seite
[0P1] 700: cmp cx,ax
[HN1] 701: jbe Ho2OK
[XT5] 702: mov StartEntry,ax ; letzte Seite
[383] 703: call PrintEntries
[3P0] 704: ret
[CF4] 705: Ho2OK: mov StartEntry,cx
[383] 706: call PrintEntries
[3P0] 707: ret
```

```
[FL1] 708: Hoch2 ENDP
[XX2] 709: ReadEntries PROC
[8R6] 710: ; Suche nach Unterverzeichnissen & Files
[RT2] 711: mov StoreFlag,0
[ZR4] 712: mov cx,1+2+4+8+16+32 ; Attribut
[R83] 713: mov dx,OFFSET Pfad
[TS1] 714: mov ah,4eh
[SB1] 715: int 21h
[RV1] 716: jnc DatOK2
[724] 717: mov ax,OFFSET ErrText8
[0R2] 718: mov ErrText,ax
[6X1] 719: call Fehler
[Y71] 720: DatOK2:
[W05] 721: ; Directory gefunden, Namen in
[UY2] 722: ; AllBuff kopieren
[L43] 723: mov AnzahlFiles,0
[8L2] 724: mov ax,AllBuff
[AR1] 725: mov es,ax
[SP1] 726: xor di,di
[KH2] 727: call StoreName
[311] 728: LOOP2:
[YX3] 729: ; DTABuff ab 30 12 löschen
[991] 730: push di
[UA2] 731: mov ax,@data
[AR1] 732: mov es,ax
[KE4] 733: mov di,OFFSET DTABuff[30]
[551] 734: mov ax,' '
[EL1] 735: mov cx,6
[ZA3] 736: rep stosw ; es:di
[8L2] 737: mov ax,AllBuff
[AR1] 738: mov es,ax
[371] 739: pop di
[X15] 740: mov ah,4fh ; DOS weitersuchen
[SB1] 741: int 21h
[673] 742: jnc WeiterSuchen
[MK3] 743: jmp ReadEntriesEnde
[AA2] 744: WeiterSuchen:
[KH2] 745: call StoreName
[8S2] 746: cmp StoreFlag,1
[UW4] 747: ; max. Anzahl überschritten
[VD3] 748: jz ReadEntriesEnde
[4U1] 749: jmp LOOP2
[TV2] 750: ReadEntriesEnde:
[F62] 751: mov ax,@data
[LW1] 752: mov es,ax
[4L5] 753: add AnzahlFiles,4 ; 4 Dummy-Files
[3P0] 754: ret
[KV2] 755: ReadEntries ENDP
[SE2] 756: StoreName PROC
[S45] 757: ; alle Namen in Buffer speichern
[VD4] 758: ; nicht zuviele Einträge
[U73] 759: mov ax,MaxEntries
[T83] 760: cmp ax,AnzahlFiles
[EG2] 761: jnz AnzahlLOK
[924] 762: mov ax,OFFSET ErrText9
[0R2] 763: mov ErrText,ax
[6X1] 764: call Fehler
[UT2] 765: mov StoreFlag,1
[3P0] 766: ret
[4M1] 767: AnzahlLOK:
[WU2] 768: inc AnzahlFiles
[D53] 769: lea si,DTABuff[30]
[UL0] 770: cld
[EL1] 771: mov cx,6
[3W1] 772: rep movsw
[TN3] 773: ; Dateiattribut holen
[3P4] 774: mov al,BYTE PTR DTABuff[21]
[985] 775: test al,1 ; schreibgeschützt ?
[P83] 776: jz Not_Protected
[WT4] 777: ; dann Attribut blau auf grau
[T84] 778: mov BYTE PTR es:[di],83
[Y41] 779: inc di
[H83] 780: jmp StoreNameEnde
[WL2] 781: Not_Protected:
[PV3] 782: test al,2 ; versteckt ?
[XK2] 783: jz Not_Hidden
[AU4] 784: ; dann Attribut gelb auf grau
[Z94] 785: mov BYTE PTR es:[di],86
[Y41] 786: inc di
[H83] 787: jmp StoreNameEnde
[L22] 788: Not_Hidden:
[G64] 789: test al,4 ; Systemdatei ?
[LR2] 790: jz Not_System
[N55] 791: ; dann Attribut grün auf grau
[P84] 792: mov BYTE PTR es:[di],82
[Y41] 793: inc di
[H83] 794: jmp StoreNameEnde
[Z72] 795: Not_System:
[Y83] 796: test al,8 ; Volume ?
[9P2] 797: jz Not_Volume
[4Y4] 798: ; dann Attribut weiss auf grau
[L84] 799: mov BYTE PTR es:[di],81
[Y41] 800: inc di
[H83] 801: jmp StoreNameEnde
[752] 802: Not_Volume:
```

Mit »box.asm« steht Ihnen eine Dateiauswahlbox zum Einbau in eigene Programme zur Verfügung.


```

[NU3] 803: test al,16 ; Directory ?
[Z83] 804: jz Not_Directory
[9A3] 805: ; '<D>' als Extension
[H03] 806: mov es:[di-3], 'D<'
[VD4] 807: mov BYTE PTR es:[di-1], '>'
[952] 808: ; rot auf grau
[W84] 809: mov BYTE PTR es:[di], 84
[Z25] 810: ; nächstes freies Zeichen im Buffer
[Y41] 811: inc di
[H83] 812: jmp StoreNameEnde
[SM2] 813: Not_Directory:
[Z93] 814: ; also normales file
[G84] 815: mov BYTE PTR es:[di], 80
[Y41] 816: inc di
[ZG2] 817: StoreNameEnde:
[3P0] 818: ret
[TD2] 819: StoreName ENDP
[Y13] 820: PrintEntries PROC
[F62] 821: mov ax, @data
[XV1] 822: mov ds, ax
[1W1] 823: mov es, ax
[EB6] 824: ; Directory-Einträge aus AllBuff auf
[HK3] 825: ; Bildschirm kopieren
[3Y4] 826: mov ax, Screen ; Segment Dest
[AR1] 827: mov es, ax
[WG5] 828: mov si, StartEntry ; Offset Dest
[AA3] 829: lea bx, Load_Window
[AR4] 830: mov cl, [bx.Win_LeftEdge]
[PH1] 831: mov ch, 0
[351] 832: inc cl
[351] 833: inc cl
[351] 834: inc cl
[HG1] 835: shl cl, 1
[8E4] 836: mov al, [bx.Win_TopEdge]
[MH1] 837: mov ah, 0
[151] 838: inc al
[151] 839: inc al
[HR1] 840: mov dx, 160
[571] 841: mul dl
[HL1] 842: add cx, ax
[XP1] 843: mov di, cx
[4H5] 844: mov ax, AllBuff ; Segment Source
[9R1] 845: mov ds, ax
[A14] 846: mov cx, 9 ; 9 Zeilen hoch
[Y92] 847: Print_Zeile:
[6A1] 848: push cx
[6A4] 849: mov cx, 4 ; 4 nebeneinander
[SE2] 850: Print_Spalte:
[6A1] 851: push cx
[CM1] 852: mov cx, 12
[MK5] 853: mov dl, BYTE PTR [si+12] ; Attribut
[YM3] 854: ; kein Eintrag mehr da ?
[TN3] 855: cmp BYTE PTR [si], ' '
[W32] 856: jnz SubDir2
[FM1] 857: mov dl, 80
[VB1] 858: SubDir2:
[Z24] 859: mov al, BYTE PTR ds:[si]
[KX5] 860: mov BYTE PTR es:[di], al ; Zeichen
[Y41] 861: inc di
[R26] 862: mov BYTE PTR es:[di], dl ; Attribut
[Y41] 863: inc di
[D51] 864: inc si
[Z42] 865: LOOP SubDir2
[Y41] 866: inc di
[Y41] 867: inc di
[D51] 868: inc si
[081] 869: pop cx
[CB3] 870: LOOP Print_Spalte
[081] 871: pop cx
[ZG1] 872: add di, 56
[W03] 873: LOOP Print_Zeile
[UA2] 874: mov ax, @data
[9R1] 875: mov ds, ax
[3P0] 876: ret
[W03] 877: PrintEntries ENDP
[0G2] 878: PrintInfo PROC
[BH6] 879: ; Anzahl files ausgeben (max. 11 Stellen)
[BA3] 880: mov ax, AnzahlFiles
[3H5] 881: sub ax, 4 ; 4 Dummy files abziehen
[GF2] 882: mov String, ax
[5Y2] 883: mov Stellen, 0005h
[BC3] 884: call ConvertString
[V92] 885: mov Spalte, 5
[0N1] 886: ; loeschen
[6N4] 887: WRITE4 80, 12, 14, Spalte, LeerStr
[DK3] 888: mov bx, StringLaenge
[ZL2] 889: mov sLength, bl
[EF5] 890: WRITE4 80, sLength, 14, Spalte, TRStr
[DK3] 891: mov bx, StringLaenge
[GE1] 892: add bl, 5
[VD2] 893: mov Spalte, bl
[025] 894: WRITE4 80, 8, 14, Spalte, AnzahlText
[3P0] 895: ret
[1F2] 896: PrintInfo ENDP
[MX1] 897: Fehler PROC
[044] 898: ; Fehlermeldung ausgeben
[ZL4] 899: WRITE1 11010111b, 32, 22, 9, ErrText
[KD1] 900: WaitESC:
[RRI] 901: mov ah, 07h

```

```

[N81] 902: int 21h
[7M1] 903: cmp al, 27
[Y52] 904: jnz WaitESC
[SD3] 905: WRITE4 80, 32, 22, 9, Linie
[Z53] 906: stc ; Carry setzen
[3P0] 907: ret
[MX1] 908: Fehler ENDP
[0X4] 909: ; WORD in String konvertieren
[1L3] 910: ; Eingabe: WORD String
[TC6] 911: ; Stellen: max. Stellenanzahl für Text
[RA5] 912: ; Ausgabe: ASCII-Zeichen ab TRStr
[NR2] 913: ; (max. 10 Stellen)
[LE5] 914: ; Anzahl Zeichen in StringLaenge
[V83] 915: ConvertString PROC
[CL3] 916: mov si, OFFSET TRStr
[NN2] 917: mov cx, Stellen
[VW4] 918: Loesch: mov [si], BYTE PTR 32
[D51] 919: inc si
[902] 920: LOOP Loesch
[KH2] 921: mov ax, String
[LR1] 922: mov dx, ax
[3L4] 923: mov bx, 10 ; durch 10 teilen
[0S1] 924: xor cx, cx
[HZ5] 925: NoNull: xor dx, dx ; Rest löschen
[DW1] 926: div bx
[CS4] 927: push dx ; Rest auf Stapel
[V51] 928: inc cx
[PX3] 929: or ax, ax ; word Null ?
[0Z1] 930: jne NoNull
[WL3] 931: mov StringLaenge, cx
[XK3] 932: mov di, OFFSET TRStr
[062] 933: Hole: pop ax
[3G1] 934: add al, 48
[372] 935: mov [di], al
[Y41] 936: inc di
[FS1] 937: LOOP Hole
[3P0] 938: ret
[W73] 939: ConvertString ENDP
[EU3] 940: Open_Window PROC NEAR
[T27] 941: ; Eingabe : bx = Zeiger auf Windowstruktur
[372] 942: call MausAus
[GX4] 943: mov dl, 160 ; Adresse im Screen
[RH1] 944: mov dh, 0
[8E4] 945: mov al, [bx.Win_TopEdge]
[MH1] 946: mov ah, 0
[571] 947: mul dl
[BR4] 948: mov dl, [bx.Win_LeftEdge]
[RH1] 949: mov dh, 0
[AH1] 950: shl dx, 1
[V11] 951: add ax, dx
[AL2] 952: mov SaveAdr, ax
[Z94] 953: mov si, ax ; Offset Source
[Z23] 954: mov di, [bx.Win_Back]
[Z72] 955: mov es, [di]
[6W3] 956: xor di, di ; Offset Dest
[Y54] 957: mov cl, [bx.Win_Height]
[PH1] 958: mov ch, 0
[1A4] 959: inc cl ; + 1 für Schatten
[B34] 960: mov dl, [bx.Win_Width]
[6A4] 961: inc dl ; + 2 für Schatten
[451] 962: inc dl
[H55] 963: mov ax, Screen ; Segment Source
[9R1] 964: mov ds, ax
[992] 965: SaveBackLoop:
[6A1] 966: push cx
[UP1] 967: mov cl, dl
[PH1] 968: mov ch, 0
[3W1] 969: rep movsw
[3B5] 970: mov al, 160 ; Breite einer Zeile
[6N1] 971: sub al, dl
[6N1] 972: sub al, dl
[MH1] 973: mov ah, 0
[3L1] 974: add si, ax
[081] 975: pop cx
[C73] 976: LOOP SaveBackLoop
[UA2] 977: mov ax, @data
[9R1] 978: mov ds, ax
[FW2] 979: ; Window aufbauen
[Y35] 980: mov di, SaveAdr ; Offset Screen
[SF2] 981: mov ax, Screen
[FM4] 982: mov es, ax ; Segment Dest
[SE5] 983: mov ah, [bx.Win_Pen] ; Attribut
[TP1] 984: mov dl, 160
[A34] 985: mov cl, [bx.Win_Width]
[5N1] 986: sub dl, cl
[5N1] 987: sub dl, cl
[RH1] 988: mov dh, 0
[Y54] 989: mov cl, [bx.Win_Height]
[PH1] 990: mov ch, 0
[3W1] 991: WriteRows:
[6A1] 992: push cx
[A34] 993: mov cl, [bx.Win_Width]
[PH1] 994: mov ch, 0
[6M1] 995: mov al, ' '
[H63] 996: rep stosw ; es:di
[081] 997: pop cx
[0L1] 998: add di, dx
[WP2] 999: LOOP WriteRows
[NZ4] 1000: ; Borderstrukturen auswerten

```



```

[N54] 1001:      mov     si,[bx.Win_Border]
[PG1] 1002:      cmp     si,0
[M53] 1003:      jz      OW_No_Border
[AR4] 1004:      mov     cl,[bx.Win_LeftEdge]
[2E4] 1005:      mov     ch,[bx.Win_TopEdge]
[8Y2] 1006:      call    Draw_Border
[9E2] 1007:      OW_No_Border:
[VD6] 1008:      ; Texte aus Windowstruktur schreiben
[PK3] 1009:      cmp     [bx.Win_Text],0
[EH2] 1010:      jz      OW_NoText
[344] 1011:      mov     si,[bx.Win_Text]
[AR4] 1012:      mov     cl,[bx.Win_LeftEdge]
[2E4] 1013:      mov     ch,[bx.Win_TopEdge]
[ES2] 1014:      call    Print_Text
[9Y1] 1015:      OW_NoText:
[K64] 1016:      ; ITEXT-Zeiger auswerten
[DU3] 1017:      cmp     [bx.Win_IText],0
[XY2] 1018:      jz      OW_No_IText
[544] 1019:      mov     si,[bx.Win_IText]
[AR4] 1020:      mov     cl,[bx.Win_LeftEdge]
[2E4] 1021:      mov     ch,[bx.Win_TopEdge]
[U03] 1022:      call    Print_IText
[WD2] 1023:      OW_No_IText:
[ND2] 1024:      ; Close-Button
[PT3] 1025:      cmp     [bx.Win_Close],1
[1F4] 1026:      jnz     OW_No_Close_Button
[DL2] 1027:      mov     di,SaveAdr
[Y41] 1028:      inc     di
[Y41] 1029:      inc     di
[5H3] 1030:      lea     si,Close_Button
[2L1] 1031:      mov     cx,3
[3W1] 1032:      rep     movsw
[1T3] 1033:      OW_No_Close_Button:
[3X4] 1034:      ; Button-Struktur auswerten
[NZ3] 1035:      cmp     [bx.Win_Button],0
[FA3] 1036:      jz      OW_No_Buttons
[L84] 1037:      mov     si,[bx.Win_Button]
[6A3] 1038:      call    Draw_Buttons
[7S2] 1039:      OW_No_Buttons:
[DL2] 1040:      mov     di,SaveAdr
[W54] 1041:      mov     al,[bx.Win_Height]
[MH1] 1042:      mov     ah,0
[SP1] 1043:      mov     cl,160
[PH1] 1044:      mov     ch,0
[W71] 1045:      mul     cx
[MK1] 1046:      add     di,ax
[7R4] 1047:      add     di,5 ; 2 links freilassen
[A34] 1048:      mov     cl,[bx.Win_Width]
[PH1] 1049:      mov     ch,0
[X13] 1050:      OW_Change_Color:
[F55] 1051:      mov     al,BYTE PTR es:[di] ;Attribut
[774] 1052:      and     al,00001111b ; Frontpen
[2P2] 1053:      cmp     al,5 ; grau ?
[G52] 1054:      jnz     NoBlack
[2T4] 1055:      mov     al,00000000b ; dann schwarz
[V32] 1056:      jmp     DoBlack
[EC1] 1057:      NoBlack:
[964] 1058:      mov     al,00000101b ;Attribut
[UB1] 1059:      DoBlack:
[PB4] 1060:      mov     BYTE PTR es:[di],al
[Y41] 1061:      inc     di
[Y41] 1062:      inc     di
[FS3] 1063:      LOOP    OW_Change_Color
[S74] 1064:      ; Schatten am rechten Rand
[DL2] 1065:      mov     di,SaveAdr
[834] 1066:      mov     al,[bx.Win_Width]
[MH1] 1067:      mov     ah,0
[7H1] 1068:      shl     ax,1
[MK1] 1069:      add     di,ax
[NK4] 1070:      inc     di ; Zeiger auf Attribut
[RN4] 1071:      add     di,160 ; ab zweiter Zeile
[Y54] 1072:      mov     cl,[bx.Win_Height]
[PH1] 1073:      mov     ch,0
[Y31] 1074:      dec     cl
[964] 1075:      mov     al,00000101b ;Attribut
[953] 1076:      OW_Change_Color2:
[F55] 1077:      mov     al,BYTE PTR es:[di] ;Attribut
[354] 1078:      and     al,00001111b ; Frontpen
[SS2] 1079:      cmp     al,5 ; grau ?
[SB2] 1080:      jnz     NoBlack2
[2T4] 1081:      mov     al,00000000b ; dann schwarz
[3A2] 1082:      jmp     DoBlack2
[UF1] 1083:      NoBlack2:
[964] 1084:      mov     al,00000101b ;Attribut
[6F1] 1085:      DoBlack2:
[PB4] 1086:      mov     BYTE PTR es:[di],al
[Y41] 1087:      inc     di
[Y41] 1088:      inc     di
[RZ1] 1089:      ; 2. Attribut
[FS5] 1090:      mov     al,BYTE PTR es:[di] ;Attribut
[354] 1091:      and     al,00001111b ; Frontpen
[SS2] 1092:      cmp     al,5 ; grau ?
[WB2] 1093:      jnz     NoBlack3
[2T4] 1094:      mov     al,00000000b ; dann schwarz
[7A2] 1095:      jmp     DoBlack3
[VF1] 1096:      NoBlack3:
[964] 1097:      mov     al,00000101b ;Attribut
[7F1] 1098:      DoBlack3:

```

```

[PB4] 1099:      mov     BYTE PTR es:[di],al
[1L1] 1100:      add     di,158
[RY3] 1101:      LOOP    OW_Change_Color2
[062] 1102:      call    MausEin
[S71] 1103:      End9999:
[3P0] 1104:      ret
[AZ2] 1105:      Open_Window ENDP
[D43] 1106:      Close_Window PROC
[SG5] 1107:      ; Restaurieren des Hintergrundes.
[T27] 1108:      ; Eingabe : bx = Zeiger auf Windowstruktur
[372] 1109:      call    MausAus
[TP1] 1110:      mov     dl,160
[RH1] 1111:      mov     dh,0
[8E4] 1112:      mov     al,[bx.Win_TopEdge]
[MH1] 1113:      mov     ah,0
[571] 1114:      mul     dl
[BR4] 1115:      mov     dl,[bx.Win_LeftEdge]
[RH1] 1116:      mov     dh,0
[AH1] 1117:      shl     dx,1
[VL1] 1118:      add     ax,dx
[NP1] 1119:      mov     di,ax
[SF2] 1120:      mov     ax,Screen
[AR1] 1121:      mov     es,ax
[Y54] 1122:      mov     cl,[bx.Win_Height]
[PH1] 1123:      mov     ch,0
[TA4] 1124:      inc     cx ; + 1 für Schatten
[B34] 1125:      mov     dl,[bx.Win_Width]
[6A4] 1126:      inc     dl ; + 2 für Schatten
[451] 1127:      inc     dl
[KZ3] 1128:      mov     si,[bx.Win_Back]
[445] 1129:      mov     ds,[si] ; Segment Source
[W44] 1130:      xor     si,si ; Offset Source
[T72] 1131:      PutBackLoop:
[6A1] 1132:      push    cx
[UP1] 1133:      mov     cl,dl
[PH1] 1134:      mov     ch,0
[3W1] 1135:      rep     movsw
[081] 1136:      pop     cx
[PP1] 1137:      mov     al,160
[6N1] 1138:      sub     al,dl
[6N1] 1139:      sub     al,dl
[MH1] 1140:      mov     ah,0
[MK1] 1141:      add     di,ax
[4Z2] 1142:      LOOP    PutBackLoop
[UA2] 1143:      mov     ax,edata
[9R1] 1144:      mov     ds,ax
[062] 1145:      call    MausEin
[3P0] 1146:      ret
[B33] 1147:      Close_Window ENDP
[ZX2] 1148:      Draw_Border PROC
[107] 1149:      ; Verkettete Borderstrukturen zeichnen.
[R85] 1150:      ; Bor-Typ = 0 : einfache Umrandung
[EF5] 1151:      ; Bor-Typ = 1 : doppelte Umrandung
[4L6] 1152:      ; Eingabe: si = Zeiger auf BorderStruktur
[0X3] 1153:      ; cl = Spaltenoffset
[LR3] 1154:      ; ch = Zeilenoffset
[372] 1155:      call    MausAus
[E33] 1156:      Draw_Border_Loop:
[G26] 1157:      mov     dl,[si.Bor_LeftEdge] ;Spalte
[PK1] 1158:      add     dl,cl
[RH1] 1159:      mov     dh,0
[DD5] 1160:      mov     al,[si.Bor_TopEdge] ;Zeile
[FK1] 1161:      add     al,ch
[MH1] 1162:      mov     ah,0
[T54] 1163:      push    cx ;Offsets retten
[SP1] 1164:      mov     cl,160
[PH1] 1165:      mov     ch,0
[471] 1166:      mul     cl
[AH1] 1167:      shl     dx,1
[F84] 1168:      add     ax,dx ; Position in ax
[3W3] 1169:      mov     di,ax ; Offset Dest
[3Y4] 1170:      mov     ax,Screen ; Segment Dest
[AR1] 1171:      mov     es,ax
[TC3] 1172:      ; Attribut bestimmen
[FA4] 1173:      mov     ah,[si.Bor_BackPen]
[EK1] 1174:      mov     cl,4
[VM1] 1175:      shl     ah,cl
[H36] 1176:      or      ah,[si.Bor_FrontPen];Attribut
[6D4] 1177:      ; linke obere Ecke zeichnen
[BE3] 1178:      cmp     [si.Bor_Typ],1
[FD2] 1179:      jz      Bor_Typ1A
[8W4] 1180:      mov     BYTE PTR es:[di],', '
[4K2] 1181:      jmp     Bor_Typ0A
[CV1] 1182:      Bor_Typ1A:
[NU4] 1183:      mov     BYTE PTR es:[di],', '
[BV1] 1184:      Bor_Typ0A:
[Y41] 1185:      inc     di
[526] 1186:      mov     BYTE PTR es:[di],ah ;Attribut
[Y41] 1187:      inc     di
[FP3] 1188:      ; obere Zeile zeichnen
[BE3] 1189:      cmp     [si.Bor_Typ],1
[GD2] 1190:      jz      Bor_Typ1B
[AT1] 1191:      mov     al,', '
[5K2] 1192:      jmp     Bor_Typ0B
[EV1] 1193:      Bor_Typ1B:

```

Mit »box.asm« steht Ihnen eine Dateiauswahlbox zum Einbau in eigene Programme zur Verfügung.


```
[LT1] 1194:      mov     al,'='
[DV1] 1195: Bor_Typ0B:
[R04] 1196:      mov     cl,[si.Bor_Width]
[PH1] 1197:      mov     ch,0
[P41] 1198:      dec     cx
[P41] 1199:      dec     cx
[VV1] 1200:      rep     stosw
[SM6] 1201: ; rechtes Eckzeichen in der oberen Zeile
[BE3] 1202:      cmp     [si.Bor_Typ],1
[HD2] 1203:      jz      Bor_Typ1C
[RT4] 1204:      mov     BYTE PTR es:[di],'q'
[6K2] 1205:      jmp     Bor_Typ0C
[GV1] 1206: Bor_Typ1C:
[BT4] 1207:      mov     BYTE PTR es:[di],'q'
[RV1] 1208: Bor_Typ0C:
[Y41] 1209:      inc     di
[526] 1210:      mov     BYTE PTR es:[di],ah ;Attribut
[TV3] 1211: ; rechten Rand zeichnen
[T31] 1212:      dec     di
[KK1] 1213:      add     di,160
[B34] 1214:      mov     cl,[si.Bor_Height]
[PH1] 1215:      mov     ch,0
[P41] 1216:      dec     cx
[P41] 1217:      dec     cx
[BE3] 1218:      cmp     [si.Bor_Typ],1
[KD2] 1219:      jz      Bor_Typ1D
[TS1] 1220:      mov     al,'|'
[WP3] 1221:      jmp     DB_Right_Height
[KV1] 1222: Bor_Typ1D:
[0T1] 1223:      mov     al,'||'
[UY2] 1224: DB_Right_Height:
[PB4] 1225:      mov     BYTE PTR es:[di],al
[Y41] 1226:      inc     di
[526] 1227:      mov     BYTE PTR es:[di],ah ;Attribut
[Y41] 1228:      inc     di
[1L1] 1229:      add     di,158
[KS3] 1230:      LOOP    DB_Right_Height
[2A6] 1231: ; rechtes unteres Eckzeichen zeichnen
[BE3] 1232:      cmp     [si.Bor_Typ],1
[LD2] 1233:      jz      Bor_Typ1E
[5W4] 1234:      mov     BYTE PTR es:[di],'J'
[8K2] 1235:      jmp     Bor_Typ0E
[MV1] 1236: Bor_Typ1E:
[ET4] 1237:      mov     BYTE PTR es:[di],'J'
[LV1] 1238: Bor_Typ0E:
[Y41] 1239:      inc     di
[AB4] 1240:      mov     BYTE PTR es:[di],ah
[Y41] 1241:      inc     di
[ZW3] 1242: ; unteren Rand zeichnen
[R04] 1243:      mov     cl,[si.Bor_Width]
[PH1] 1244:      mov     ch,0
[P41] 1245:      dec     cx
[BN1] 1246:      sub     di,cx
[BN1] 1247:      sub     di,cx
[P41] 1248:      dec     cx
[BE3] 1249:      cmp     [si.Bor_Typ],1
[MD2] 1250:      jz      Bor_Typ1F
[AT1] 1251:      mov     al,'-'
[9K2] 1252:      jmp     Bor_Typ0F
[PV1] 1253: Bor_Typ1F:
[LT1] 1254:      mov     al,'='
[NV1] 1255: Bor_Typ0F:
[VV1] 1256:      rep     stosw
[K56] 1257: ; linkes unteres Eckzeichen zeichnen
[R04] 1258:      mov     cl,[si.Bor_Width]
[PH1] 1259:      mov     ch,0
[P41] 1260:      dec     cx
[BN1] 1261:      sub     di,cx
[BN1] 1262:      sub     di,cx
[P41] 1263:      dec     cx
[BE3] 1264:      cmp     [si.Bor_Typ],1
[ND2] 1265:      jz      Bor_Typ1G
[UT4] 1266:      mov     BYTE PTR es:[di],'L'
[AK2] 1267:      jmp     Bor_Typ0G
[SV1] 1268: Bor_Typ1G:
[KU4] 1269:      mov     BYTE PTR es:[di],'L'
[RV1] 1270: Bor_Typ0G:
[Y41] 1271:      inc     di
[AB4] 1272:      mov     BYTE PTR es:[di],ah
[8M3] 1273: ; linken Rand zeichnen
[T31] 1274:      dec     di
[1N1] 1275:      sub     di,160
[B34] 1276:      mov     cl,[si.Bor_Height]
[PH1] 1277:      mov     ch,0
[MG1] 1278:      sub     cl,2
[BE3] 1279:      cmp     [si.Bor_Typ],1
[PD2] 1280:      jz      Bor_Typ1H
[TS1] 1281:      mov     al,'/'
[KG3] 1282:      jmp     DB_Left_Height
[UV1] 1283: Bor_Typ1H:
[0T1] 1284:      mov     al,'||'
[ZU2] 1285: DB_Left_Height:
[PB4] 1286:      mov     BYTE PTR es:[di],al
[Y41] 1287:      inc     di
[526] 1288:      mov     BYTE PTR es:[di],ah ;Attribut
[Y41] 1289:      inc     di
[5N1] 1290:      sub     di,162
[8K3] 1291:      LOOP    DB_Left_Height
```

```
[VU6] 1292: ; weitere Zeiger auf Borderstrukturen ?
[T03] 1293:      pop     cx ; Offsets
[414] 1294:      mov     ax,[si.Bor_Next]
[2H1] 1295:      cmp     ax,0
[V34] 1296:      jz      DB_No_More_Borders
[4R1] 1297:      mov     si,ax
[H34] 1298:      jmp     Draw_Border_Loop
[NR3] 1299: DB_No_More_Borders:
[P32] 1300:      call    MausEin
[3P0] 1301:      ret
[YV2] 1302: Draw_Border ENDP
[VV2] 1303: Print_Text PROC
[RS5] 1304: ; Verkettete Textstrukturen mit
[9T4] 1305: ; verschiedenen Attributen.
[F44] 1306: ; Null-terminierte Texte.
[P30] 1307: ;
[5R6] 1308: ; Eingabe : si = Zeiger auf Textstruktur
[E14] 1309: ;      cl = Spaltenoffset
[1U3] 1310: ;      ch = Zeilenoffset
[KH2] 1311: PT_Text_Loop:
[A56] 1312:      mov     dl,[si.Text_LeftEdge] ;Spalte
[PK1] 1313:      add     dl,cl
[RH1] 1314:      mov     dh,0
[W55] 1315:      mov     al,[si.Text_TopEdge] ;Zeile
[FK1] 1316:      add     al,ch
[MH1] 1317:      mov     ah,0
[UZ3] 1318:      push    cx ;Offsets retten
[SP1] 1319:      mov     cl,160
[PH1] 1320:      mov     ch,0
[471] 1321:      mul     cl
[AH1] 1322:      shl     dx,1
[Y54] 1323:      add     ax,dx ;Position in ax
[GU3] 1324:      mov     di,ax ;Offset Dest
[XD3] 1325:      add     di,ScreenOffset
[GU4] 1326:      mov     ax,Screen ;Segment Dest
[AR1] 1327:      mov     es,ax
[S91] 1328:      push    si
[L54] 1329:      mov     si,[si.Text_Text]
[CR2] 1330: PT_CopyString:
[CV0] 1331:      lodsb
[AG1] 1332:      cmp     al,0
[Y63] 1333:      jz      PT_StringEnde
[NX0] 1334:      stosb
[MD3] 1335:      jmp     PT_CopyString
[CM2] 1336: PT_StringEnde:
[K71] 1337:      pop     si
[081] 1338:      pop     cx
[GE6] 1339: ; weitere Zeiger auf Textstrukturen ?
[T54] 1340:      mov     ax,[si.Text_Next]
[2H1] 1341:      cmp     ax,0
[TS3] 1342:      jz      PT_No_More_Text
[4R1] 1343:      mov     si,ax
[4F3] 1344:      jmp     PT_Text_Loop
[383] 1345: PT_No_More_Text:
[3P0] 1346:      ret
[VV2] 1347: Print_Text ENDP
[W23] 1348: Print_IText PROC
[RS5] 1349: ; Verkettete Textstrukturen mit
[G64] 1350: ; konstantem Attributen.
[F44] 1351: ; Null-terminierte Texte.
[P30] 1352: ;
[FY6] 1353: ; Eingabe : si = Zeiger auf ITEXT-Struktur
[E14] 1354: ;      cl = Spaltenoffset
[1U3] 1355: ;      ch = Zeilenoffset
[7S2] 1356: PT_IText_Loop:
[RU4] 1357:      mov     dl,[si.IText_LeftEdge]
[PK1] 1358:      add     dl,cl
[RH1] 1359:      mov     dh,0
[016] 1360:      mov     al,[si.IText_TopEdge] ;Zeile
[FK1] 1361:      add     al,ch
[MH1] 1362:      mov     ah,0
[UZ3] 1363:      push    cx ;Offsets retten
[SP1] 1364:      mov     cl,160
[PH1] 1365:      mov     ch,0
[471] 1366:      mul     cl
[AH1] 1367:      shl     dx,1
[F84] 1368:      add     ax,dx ; Position in ax
[W33] 1369:      mov     di,ax ; Offset Dest
[3Y4] 1370:      mov     ax,Screen ; Segment Dest
[AR1] 1371:      mov     es,ax
[ZD4] 1372:      mov     dl,[si.IText_Attribut]
[S91] 1373:      push    si
[184] 1374:      mov     si,[si.IText_Text]
[RX2] 1375: PTI_CopyString:
[KM3] 1376:      mov     al,BYTE PTR [si]
[AG1] 1377:      cmp     al,0
[RB3] 1378:      jz      PTI_StringEnde
[PB4] 1379:      mov     BYTE PTR es:[di],al
[Y41] 1380:      inc     di
[D51] 1381:      inc     si
[WB4] 1382:      mov     BYTE PTR es:[di],dl
[Y41] 1383:      inc     di
[GL3] 1384:      jmp     PTI_CopyString
[4V2] 1385: PTI_StringEnde:
[K71] 1386:      pop     si
[081] 1387:      pop     cx
[GE6] 1388: ; weitere Zeiger auf Textstrukturen ?
[184] 1389:      mov     ax,[si.IText_Next]
```



```
[2H1] 1390:      cmp     ax,0
[234] 1391:      jz      PTI_No_More_Text
[4R1] 1392:      mov     si,ax
[UE3] 1393:      jmp     PTI_Text_Loop
[R83] 1394: PTI_No_More_Text:
[3P0] 1395:      ret
[V03] 1396: Print_IText ENDP
[K33] 1397: Draw Buttons PROC
[5M6] 1398: ; Verkettete Buttons mit verschiedenen
[P42] 1399: ; Attributen.
[N27] 1400: ; Eingabe : si = Zeiger auf Buttonstruktur
[W32] 1401:      call    MausAus
[VG1] 1402: DB_Loop:
[U26] 1403:      mov     dl,[si.But_LeftEdge] ;Spalte
[RH1] 1404:      mov     dh,0
[7L5] 1405:      mov     al,[si.But_TopEdge] ;Zeile
[MH1] 1406:      mov     ah,0
[Z84] 1407:      push    cx ; Offsets retten
[DL2] 1408:      mov     di,SaveAdr
[SP1] 1409:      mov     cl,160
[PH1] 1410:      mov     ch,0
[471] 1411:      mul     cl
[AH1] 1412:      shl     dx,1
[VL1] 1413:      add     ax,dx
[Z53] 1414:      add     di,ax ; Offset Dest
[V13] 1415:      mov     dx,di ; merken
[3Y4] 1416:      mov     ax,Screen ; Segment Dest
[AR1] 1417:      mov     es,ax
[114] 1418:      mov     cl,[si.But_Width]
[PH1] 1419:      mov     ch,0
[S91] 1420:      push    si
[U14] 1421:      mov     si,[si.But_Text]
[3W1] 1422:      rep     movsw
[683] 1423: ; Schatten zeichnen
[K71] 1424:      pop     si
[114] 1425:      mov     cl,[si.But_Width]
[PH1] 1426:      mov     ch,0
[P41] 1427:      dec     cx
[P41] 1428:      dec     cx
[1R1] 1429:      mov     di,dx
[3X1] 1430:      add     di,160+4
[S91] 1431:      push    si
[ZV3] 1432:      mov     si,OFFSET Shadow
[3W1] 1433:      rep     movsw
[K71] 1434:      pop     si
[081] 1435:      pop     cx
[3X6] 1436: ; weitere Zeiger auf Buttonstrukturen ?
[E14] 1437:      mov     ax,[si.But_Next]
[ZH1] 1438:      cmp     ax,0
[T64] 1439:      jz      DB_No_More_Buttons
[4R1] 1440:      mov     si,ax
[D72] 1441:      jmp     DB_Loop
[9U3] 1442: DB_No_More_Buttons:
[062] 1443:      call    MausEin
[3P0] 1444:      ret
[G23] 1445: Draw Buttons ENDP
[MX2] 1446: Get_Message PROC
[T27] 1447: ; Eingabe : bx = Zeiger auf Windowstruktur
[2P1] 1448: ; Ausgabe :
[4D6] 1449: ; bp = Adresse des angeklickten Buttons
[5A1] 1450:      push    bx
[672] 1451:      call    MausPos
[DG2] 1452:      mov     bx,MausX
[AK1] 1453:      mov     cl,3
[MP1] 1454:      shr     bx,cl
[FG2] 1455:      mov     MausX,bx
[HG2] 1456:      mov     bx,MausY
[MP1] 1457:      shr     bx,cl
[GG2] 1458:      mov     MausY,bx
[Z71] 1459:      pop     bx
[L84] 1460:      mov     si,[bx.Win_Button]
[524] 1461: ; Hat Window ein CloseButton ?
[KT3] 1462:      cmp     [bx.Win_Close],0
[NT3] 1463:      jnz     GM_Close_Button
[VN4] 1464:      jmp     GM_Test_Move_Button
[X33] 1465: GM_Close_Button:
[Z27] 1466: ; nur oberste Zeile des Windows zulassen
[AE4] 1467:      mov     cl,[bx.Win_TopEdge]
[PH1] 1468:      mov     ch,0
[1F2] 1469:      cmp     MausY,cx
[NK3] 1470:      jz      GM_Correct_Line
[VN4] 1471:      jmp     GM_Test_Move_Button
[933] 1472: GM_Correct_Line:
[ST3] 1473: ; nur ein Zeichen breit
[AR4] 1474:      mov     cl,[bx.Win_LeftEdge]
[PH1] 1475:      mov     ch,0
[V51] 1476:      inc     cx
[V51] 1477:      inc     cx
[0F2] 1478:      cmp     MausX,cx
[K64] 1479:      jz      GM_Close_Button_OK
[VN4] 1480:      jmp     GM_Test_Move_Button
[9S3] 1481: GM_Close_Button_OK:
[ZH3] 1482:      lea     bp,Close_Button
[3P0] 1483:      ret
[KX4] 1484: ; Hat Window ein Move-Button ?
[EY3] 1485: GM_Test_Move_Button:
[1H3] 1486:      cmp     [bx.Win_Move],0
[1P3] 1487:      jnz     GM_Move_Button
```

```
[645] 1488:      jmp     GM_Check_Button_Struct
[822] 1489: GM_Move_Button:
[Z27] 1490: ; nur oberste Zeile des Windows zulassen
[AE4] 1491:      mov     cl,[bx.Win_TopEdge]
[PH1] 1492:      mov     ch,0
[1F2] 1493:      cmp     MausY,cx
[XS3] 1494:      jz      GM_Correct_Line2
[645] 1495:      jmp     GM_Check_Button_Struct
[N63] 1496: GM_Correct_Line2:
[5V5] 1497:      mov     cl,[bx.Win_LeftEdge] ;links
[PH1] 1498:      mov     ch,0
[5F1] 1499:      add     cx,4
[0F2] 1500:      cmp     MausX,cx
[TR3] 1501:      jae     GM_Move_Left_OK
[645] 1502:      jmp     GM_Check_Button_Struct
[N13] 1503: GM_Move_Left_OK:
[8B5] 1504:      add     cl,[bx.Win_Width] ;rechts
[SH1] 1505:      sub     cx,5
[0F2] 1506:      cmp     MausX,cx
[D74] 1507:      jbe     GM_Move_Button_OK
[645] 1508:      jmp     GM_Check_Button_Struct
[F63] 1509: GM_Move_Button_OK:
[D25] 1510:      mov     bp,1234 ;Code für Move-Button
[3P0] 1511:      ret
[BF4] 1512: GM_Check_Button_Struct:
[1N4] 1513:      mov     cl,[si.But_LeftEdge]
[9L4] 1514:      add     cl,[bx.Win_LeftEdge]
[PH1] 1515:      mov     ch,0
[V51] 1516:      inc     cx
[L34] 1517:      cmp     MausX,cx ;links
[Y92] 1518:      jae     GM_Links_OK
[AR4] 1519:      jmp     GM_Test_Next_Button
[9A2] 1520: GM_Links_OK:
[0X3] 1521:      add     cl,[si.But_Width]
[P41] 1522:      dec     cx
[P41] 1523:      dec     cx
[P41] 1524:      dec     cx
[ZV3] 1525:      cmp     MausX,cx ;rechts
[H83] 1526:      jbe     GM_Rechts_OK
[AR4] 1527:      jmp     GM_Test_Next_Button
[8E2] 1528: GM_Rechts_OK:
[1C4] 1529:      mov     cl,[si.But_TopEdge]
[9A4] 1530:      add     cl,[bx.Win_TopEdge]
[PH1] 1531:      mov     ch,0
[DA3] 1532:      cmp     MausY,cx ;oben
[WR2] 1533:      jae     GM_Oben_OK
[AR4] 1534:      jmp     GM_Test_Next_Button
[H42] 1535: GM_Oben_OK:
[MZ3] 1536:      add     cl,[si.But_Height]
[P41] 1537:      dec     cx
[1F2] 1538:      cmp     MausY,cx
[V24] 1539:      jbe     GM_Unten_OK ;unten
[AR4] 1540:      jmp     GM_Test_Next_Button
[HA2] 1541: GM_Unten_OK:
[DS3] 1542:      jmp     GM_Button_Found
[Y93] 1543: GM_Test_Next_Button:
[UK3] 1544:      cmp     [si.But_Next],0
[K74] 1545:      jz      GM_No_More_Buttons
[Z14] 1546:      mov     si,[si.But_Next]
[645] 1547:      jmp     GM_Check_Button_Struct
[GV3] 1548: GM_No_More_Buttons:
[L35] 1549: ; Maus über keinem Button, Ende
[5K1] 1550:      mov     bp,0
[3P0] 1551:      ret
[P33] 1552: GM_Button_Found:
[S07] 1553: ; Adresse des gewählten Buttons gefunden
[4X3] 1554:      mov     bp,[si.But_Text]
[WU6] 1555: ; Register bp,bx und si nicht verändern
[Z45] 1556: ; jetzt Button "reindrücken"
[372] 1557:      call    MausAus
[SF2] 1558:      mov     ax,Screen
[N54] 1559:      mov     es,ax ; Segment Dest
[ZB4] 1560:      mov     al,[si.But_TopEdge]
[7A4] 1561:      add     al,[bx.Win_TopEdge]
[MH1] 1562:      mov     ah,0
[SP1] 1563:      mov     cl,160
[PH1] 1564:      mov     ch,0
[W71] 1565:      mul     cx
[1N4] 1566:      mov     cl,[si.But_LeftEdge]
[9L4] 1567:      add     cl,[bx.Win_LeftEdge]
[NE3] 1568:      mov     ch,0 ; + Spalte * 2
[9H1] 1569:      shl     cx,1
[RL1] 1570:      add     ax,cx
[NP1] 1571:      mov     di,ax
[EP4] 1572:      mov     dx,ax ; Offset Dest in dx
[KM4] 1573: ; ein Zeichen weiter rechts
[Y41] 1574:      inc     di
[Y41] 1575:      inc     di
[114] 1576:      mov     cl,[si.But_Width]
[PH1] 1577:      mov     ch,0
[7B4] 1578:      mov     ax,cx ;Breite merken
[P41] 1579:      dec     cx
[S91] 1580:      push    si
[0R1] 1581:      mov     si,bp
[3W1] 1582:      rep     movsw
```

Mit »box.asm« steht Ihnen eine Dateiauswahlbox zum Einbau in eigene Programme zur Verfügung.


```
[PP3] 1583: ; Schatten eliminieren
[7A1] 1584:      push    dx
[TP1] 1585:      mov     dl,160
[XM1] 1586:      sub     dl,al
[XM1] 1587:      sub     dl,al
[EE1] 1588:      add     dl,4
[RH1] 1589:      mov     dh,0
[ØL1] 1590:      add     di,dx
[181] 1591:      pop     dx
[HØ3] 1592:      lea     si,ShadowOff
[R13] 1593:      mov     cx,ax ;Breite
[P41] 1594:      dec     cx
[P41] 1595:      dec     cx
[3W1] 1596:      rep     movsw
[K71] 1597:      pop     si
[Ø62] 1598:      call    MausEin
[Z85] 1599:      ; linken Mausknopf losgelassen ?
[PB2] 1600: GM_Check_LMB:
[5A1] 1601:      push    bx
[7A1] 1602:      push    dx
[N91] 1603:      push    bp
[Z62] 1604:      call    MausKey
[F71] 1605:      pop     bp
[181] 1606:      pop     dx
[Z71] 1607:      pop     bx
[RX1] 1608:      mov     ax,Key
[AG1] 1609:      cmp     al,0
[VV2] 1610:      jz      GM_LMB_Left
[TX2] 1611: GM_LMB_Pressed:
[XM5] 1612:      call    GM_Is_Mouse_Over_Button?
[9N4] 1613:      ; Maus ist noch über Button
[MY2] 1614:      jc      GM_Check_LMB
[S83] 1615:      jmp     GM_No_Action
[VZ4] 1616: ; linke Maustaste losgelassen
[BA2] 1617: GM_LMB_Left:
[XM5] 1618:      call    GM_Is_Mouse_Over_Button?
[PB2] 1619:      jc      GM_Action
[R76] 1620: ; Maus nicht über Button losgelassen
[ZE2] 1621: GM_No_Action:
[CU4] 1622:      call    GM_Draw_This_Button
[LM1] 1623:      mov     bp,0
[3PØ] 1624:      ret
[W47] 1625: GM_Action: ;Maus über Button losgelassen
[W15] 1626: ; Button restaurieren und Ende
[1L4] 1627:      call    GM_Draw_This_Button
[3PØ] 1628:      ret
[LV2] 1629: Get_Message ENDP
[W14] 1630: GM_Draw_This_Button PROC
[UX3] 1631: ; Original Button malen
[715] 1632: ; bp = Zeiger auf Buttonadresse
[DA5] 1633: ; si = Zeiger auf Buttonstruktur
[Y54] 1634: ; dx = Position im Fenster
[372] 1635:      call    MausAus
[114] 1636:      mov     cl,[si.But_Width]
[PH1] 1637:      mov     ch,0
[M45] 1638:      mov     ax,cx ;Breite in ax merken
[WU3] 1639:      mov     di,dx ;Offset Dest
[S91] 1640:      push    si
[BC4] 1641:      mov     si,bp ;Offset Source
[3W1] 1642:      rep     movsw
[683] 1643: ; Schatten zeichnen
[R13] 1644:      mov     cx,ax ;Breite
[P41] 1645:      dec     cx
[P41] 1646:      dec     cx
[1R1] 1647:      mov     di,dx
[3X1] 1648:      add     di,160+4
[9C2] 1649:      lea     si,Shadow
[3W1] 1650:      rep     movsw
[K71] 1651:      pop     si
[Ø62] 1652:      call    MausEin
[3PØ] 1653:      ret
[VH4] 1654: GM_Draw_This_Button ENDP
[1H5] 1655: GM_Is_Mouse_Over_Button? PROC
[DA5] 1656: ; si = Zeiger auf Buttonstruktur
[D95] 1657: ; bx = Zeiger auf Windowstruktur
[7A1] 1658:      push    dx
[5A1] 1659:      push    bx
[N91] 1660:      push    bp
[672] 1661:      call    MausPos
[F71] 1662:      pop     bp
[Z71] 1663:      pop     bx
[8R4] 1664:      mov     al,[bx.Win_LeftEdge]
[MH1] 1665:      mov     ah,0
[YG4] 1666:      add     al,[si.But_LeftEdge]
[AK1] 1667:      mov     cl,3
[VN1] 1668:      shl     ax,cl
[NK3] 1669:      cmp     MausX,ax ;links
[TX2] 1670:      jae     GMI_Left_OK
[PLØ] 1671:      clc
[MF2] 1672:      jmp     GMI_End
[W62] 1673: GMI_Left_OK:
[1D5] 1674:      mov     dl,[si.But_Width] ;rechts
[RH1] 1675:      mov     dh,0
[VN1] 1676:      shl     dx,cl
[V11] 1677:      add     ax,dx
[UE2] 1678:      cmp     MausX,ax
[A73] 1679:      jbe     GMI_Right_OK
```

```
[PLØ] 1680:      clc
[MF2] 1681:      jmp     GMI_End
[VC2] 1682: GMI_Right_OK:
[9A5] 1683:      mov     al,[bx.Win_TopEdge] ;oben
[MH1] 1684:      mov     ah,0
[Y74] 1685:      add     al,[si.But_TopEdge]
[VN1] 1686:      shl     ax,cl
[VE2] 1687:      cmp     MausY,ax
[6T2] 1688:      jae     GMI_Top_OK
[PLØ] 1689:      clc
[MF2] 1690:      jmp     GMI_End
[L42] 1691: GMI_Top_OK:
[H85] 1692:      mov     dl,[si.But_Height] ;unten
[RH1] 1693:      mov     dh,0
[VN1] 1694:      shl     dx,cl
[V11] 1695:      add     ax,dx
[M41] 1696:      dec     ax
[VE2] 1697:      cmp     MausY,ax
[C83] 1698:      jbe     GMI_Unten_OK
[PLØ] 1699:      clc
[MF2] 1700:      jmp     GMI_End
[N22] 1701: GMI_Unten_OK:
[486] 1702: ; Maus ist über Button -> Carry setzen
[ENØ] 1703:      stc
[NH1] 1704: GMI_End:
[181] 1705:      pop     dx
[3PØ] 1706:      ret
[ZP5] 1707: GM_Is_Mouse_Over_Button? ENDP
[Z82] 1708: MausKey PROC
[ØL1] 1709:      mov     ax,3
[XB1] 1710:      int     33h
[1X1] 1711:      mov     Key,bx
[3PØ] 1712:      ret
[162] 1713: MausKey ENDP
[P62] 1714: MausEin PROC
[SK1] 1715:      mov     ax,1
[XB1] 1716:      int     33h
[3PØ] 1717:      ret
[N42] 1718: MausEin ENDP
[682] 1719: MausAus PROC
[WK1] 1720:      mov     ax,2
[XB1] 1721:      int     33h
[3PØ] 1722:      ret
[562] 1723: MausAus ENDP
[K82] 1724: MausPos PROC
[ØL1] 1725:      mov     ax,3
[XB1] 1726:      int     33h
[KG2] 1727:      mov     MausX,cx
[PG2] 1728:      mov     MausY,dx
[3PØ] 1729:      ret
[H62] 1730: MausPos ENDP
[8D2] 1731: ClearBuff PROC
[NF4] 1732: ; Tastaturpuffer löschen
[7S1] 1733:      mov     ah,0ch
[ML1] 1734:      mov     al,20
[SB1] 1735:      int     21h
[3PØ] 1736:      ret
[9C2] 1737: ClearBuff ENDP
[ZSØ] 1738: .DATA
[757] 1739: OldMode dw ? ; Auflösung bei Programmstart
[G52] 1740: WINDOW STRUC
[H35] 1741: Win_LeftEdge      db 0 ; linker Rand
[845] 1742: Win_TopEdge       db 0 ; oberer Rand
[4B5] 1743: Win_Width         db 0 ; Fensterbreite
[155] 1744: Win_Height        db 0 ; Fensterhöhe
[9G4] 1745: Win_Pen           db 0 ; Attribut
[LL5] 1746: Win_MouseX        dw 0 ; Mausposition
[NX2] 1747: Win_MouseY        dw 0
[4Y5] 1748: Win_Button        dw 0 ; Buttonstruktur
[5M5] 1749: Win_Text          dw 0 ; Textstruktur 1
[VN5] 1750: Win_IText         dw 0 ; Textstruktur 2
[YT5] 1751: Win_Border        dw 0 ; Borderstruktur
[A95] 1752: Win_Close         db 0 ; Close-Button
[UZ4] 1753: Win_Move          db 0 ; Move-Button
[ZR5] 1754: Win_Input         dw 0 ; Input-Struktur
[Z26] 1755: Win_Back         dw 0 ; Hintergrundbuffers
[112] 1756: WINDOW ENDS
[542] 1757: BORDER STRUC
[5P4] 1758: Bor_Next          dw 0 ; nächste
[TS6] 1759: Bor_LeftEdge     db 0 ; linker Rand relativ
[ES6] 1760: Bor_TopEdge      db 0 ; oberer Rand relativ
[R54] 1761: Bor_Width        db 0 ; Breite
[254] 1762: Bor_Height       db 0 ; Höhe
[MD6] 1763: Bor_FrontPen     db 0 ; Vordergrundfarbe
[576] 1764: Bor_BackPen      db 0 ; Hintergrundfarbe
[Y84] 1765: Bor_Typ          db 0 ; 0 = lfach
[PZ1] 1766: BORDER ENDS
[EF6] 1767: TEXT STRUC ; verschiedene Attributen
[7E4] 1768: Text_Next        dw 0 ; nächste
[KH6] 1769: Text_LeftEdge    db 0 ; linker Rand relativ
[HG6] 1770: Text_TopEdge     db 0 ; oberer Rand relativ
[CB5] 1771: Text_Text        dw 0 ; Zeichenkette
[WK1] 1772: TEXT ENDS
[YU5] 1773: ITEXT STRUC ; konstantes Attribut
[8L4] 1774: IText_Next       dw 0 ; nächste
[4P6] 1775: IText_LeftEdge   db 0 ; linker Rand relativ
[RK6] 1776: IText_TopEdge    db 0 ; oberer Rand relativ
```



```

[385] 1777: IText Attribut db 0 ; Attribut
[LG5] 1778: IText_Text dw 0 ; Zeichenkette
[5R1] 1779: ITEXT ENDS
[R52] 1780: BUTTON STRUC
[TN4] 1781: But_Next dw 0 ; nächster
[345] 1782: But_LeftEdge db 0 ; linker Rand
[U45] 1783: But_TopEdge db 0 ; oberer Rand
[ZZ3] 1784: But_Width db 0 ; Breite
[714] 1785: But_Height db 0 ; Höhe
[M26] 1786: But_Text dw 0 ; Buttondefinition
[812] 1787: BUTTON ENDS
[682] 1788: ErrText dw 0
[BF2] 1789: ; immer 32 Bytes
[987] 1790: ErrText1 db ' Unknown error. Press <ESC>
>'
[R07] 1791: ErrText2 db ' File not found. Press <ESC>
>'
[P27] 1792: ErrText8 db ' Path not found. Press <ESC>
>'
[B37] 1793: ErrText9 db ' Too many files. Press <ESC>
>'
[553] 1794: Linie db 32 DUP (205)
[WA3] 1795: DTABuff db 44 DUP (32)
[S31] 1796: i db 0,0
[U31] 1797: j db 0,0
[KY2] 1798: StringLaenge dw 0
[BV2] 1799: TRStr db 10 DUP (?)
[UR2] 1800: AnzahlFiles dw 0
[SY3] 1801: AnzahlText db ' Dateien'
[5P1] 1802: Spalte db 0
[ER0] 1803: even
[2T1] 1804: BoxAdr dw 0
[DG5] 1805: Shadow db 220,5,220,5,220,5,220,5,220,5,220,5,220,5,220,5,220,5
[VM1] 1806: db 5,220,5,220,5
[PL5] 1807: ShadowOff db 32,80,32,80,32,80,32,80,32,80,32,80,32,80,32,80
[TD2] 1808: db 32,80,32,80,32,80
[KD3] 1809: Schatten db 8 DUP (223)
[ML5] 1810: SchattenWeg db 32,80,32,80,32,80,32,80,32,80,32,80,32,80,32,80
[UU2] 1811: db 80,32,80,32,80,32,80
[LN3] 1812: FileButton db 12 DUP(32)
[ER0] 1813: even
[TE3] 1814: PfadInputLaenge db 0
[ER0] 1815: even
[7A2] 1816: PfadA db 'A:*.**'
[DA2] 1817: PfadB db 'B:*.**'
[LA2] 1818: PfadC db 'C:*.**'
[TA2] 1819: PfadD db 'D:*.**'
[ZA2] 1820: PfadE db 'E:*.**'
[5B2] 1821: PfadF db 'F:*.**'
[BB2] 1822: PfadG db 'G:*.**'
[HB2] 1823: PfadH db 'H:*.**'
[6D6] 1824: Stellen dw 0 ;bei Stringkonvertierung
[YY1] 1825: Zeichen dw 0
[N05] 1826: DirFlag db 0 ;0 = normales file
[ER0] 1827: even
[G36] 1828: ;1.,2.,3. oder 4. Spalte für Directory
[RH2] 1829: FileSpalte db 0 ;
[ER0] 1830: even
[BE5] 1831: FileStart db 0 ;Spalte der Entries
[ER0] 1832: even
[1R2] 1833: File_Merken dw 0
[7M2] 1834: StartEntry dw 0
[ER2] 1835: EntryAnfang dw 0
[586] 1836: MaxEntries dw 500 ;max. Anzahl Einträge
[486] 1837: LeerStr db 12 DUP (32) ;leerer String
[D73] 1838: Leer43 db 44 DUP (32)
[S32] 1839: sLength db 0
[ER0] 1840: even
[W62] 1841: QuitFlag db 0
[966] 1842: ; 1 = max. Anzahl in Dir überschritten
[LA2] 1843: StoreFlag db 0
[BK1] 1844: MausX dw 0
[DK1] 1845: MausY dw 0
[691] 1846: Key dw 0
[PT1] 1847: String dw 0
[TF2] 1848: Screen dw 0b800h
[7Y2] 1849: ScreenOffset dw 0
[HG3] 1850: ; 1 Zeilen a' 8 Zeichen
[595] 1851: UP1Button db 32,80,32,49,32,49,25,49,25
[6B4] 1852: db 49,25,49,25,49,32,49,32,49,223,5
[295] 1853: UP2Button db 32,80,32,49,32,49,32,49,25
[ZA4] 1854: db 49,25,49,32,49,32,49,32,49,223,5
[N75] 1855: DN1Button db 32,80,32,49,32,49,32,49,24
[YA4] 1856: db 49,24,49,32,49,32,49,32,49,223,5
[X75] 1857: DN2Button db 32,80,32,49,32,49,24,49,24
[2B4] 1858: db 49,24,49,24,49,32,49,32,49,223,5
[A65] 1859: AButton db 32,80,32,49,32,49,32,49,65
[1B4] 1860: db 52,58,49,32,49,32,49,32,49,223,5
[D65] 1861: BButton db 32,80,32,49,32,49,32,49,66
[1B4] 1862: db 52,58,49,32,49,32,49,32,49,223,5
[G65] 1863: CButton db 32,80,32,49,32,49,32,49,67
[1B4] 1864: db 52,58,49,32,49,32,49,32,49,223,5
[L65] 1865: DButton db 32,80,32,49,32,49,32,49,68
[1B4] 1866: db 52,58,49,32,49,32,49,32,49,223,5
[P65] 1867: EButton db 32,80,32,49,32,49,32,49,69
[1B4] 1868: db 52,58,49,32,49,32,49,32,49,223,5

```

```
[L65] 1869: FButton db 32,80,32,49,32,49,32,49,70
[1B4] 1870: db 52,58,49,32,49,32,49,32,49,223,5
[P65] 1871: GButton db 32,80,32,49,32,49,32,49,71
[1B4] 1872: db 52,58,49,32,49,32,49,32,49,223,5
[T65] 1873: HButton db 32,80,32,49,32,49,32,49,72
[1B4] 1874: db 52,58,49,32,49,32,49,32,49,223,5
[Z75] 1875: OKButton db 32,80,32,49,32,49,32,49,79
[6B4] 1876: db 52,75,49,32,49,32,49,32,49,223,5
[5E5] 1877: CanButton db 32,80,32,49,32,49,32,49,110
[2H4] 1878: db 52,99,49,101,49,108,49,32,49,223,5
[WR5] 1879: SaveButton db 32,80,32,49,32,49,'S',52,'a'
[5G4] 1880: db 49,'v',49,'e',49,32,49,32,49,223,5
[CM5] 1881: RetryButton db 32,80,32,49,'r',52,'e',49
[525] 1882: db 't',49,'r',49,'y',49,32,49,32,49,223,5
[B45] 1883: CloseButton db '[',80,4,82,']',80
[Z21] 1884: SaveAdd dw 0
[TF1] 1885: SortLaenge db 13
[8L6] 1886: HelpBuffer db 14 DUP (4) ; für Sortieren
[4BC] 1887: Load Window WINDOW <0,2,69,21,80,0,0,L B1,0
,L IText,L Border,1,0,0,BackBuffer>
[C66] 1888: L B1 BUTTON <L B2,5,16,10,1,UP1Button>
[766] 1889: L B2 BUTTON <L B3,18,16,10,1,UP2Button>
[R36] 1890: L B3 BUTTON <L B4,31,16,10,1,DN1Button>
[446] 1891: L B4 BUTTON <L B5,44,16,10,1,DN2Button>
[246] 1892: L B5 BUTTON <L B6,18,18,10,1,OKButton>
[566] 1893: L B6 BUTTON <L B7,31,18,10,1,CanButton>
[CM5] 1894: L B7 BUTTON <L B8,56,2,10,1,AButton>
[VM5] 1895: L B8 BUTTON <L B9,56,4,10,1,BButton>
[ZT5] 1896: L B9 BUTTON <L B10,56,6,10,1,CButton>
[1V5] 1897: L B10 BUTTON <L B11,56,8,10,1,DButton>
[N06] 1898: L B11 BUTTON <L B12,56,10,10,1,EButton>
[016] 1899: L B12 BUTTON <L B13,56,12,10,1,FButton>
[B16] 1900: L B13 BUTTON <L B14,56,14,10,1,GButton>
[WB5] 1901: L B14 BUTTON <0,56,16,10,1,HButton>
[1X6] 1902: L IText ITEXT <L IText2,33,12,80,LIText>
[507] 1903: L IText2 ITEXT <L IText3,26,0,80,LIText2>
[217] 1904: L IText3 ITEXT <L IText4,5,14,80,LIText3>
[BY6] 1905: L IText4 ITEXT <L IText5,3,1,80,LIText4>
[T07] 1906: L IText5 ITEXT <L IText6,16,1,83,LIText5>
[A17] 1907: L IText6 ITEXT <L IText7,28,1,86,LIText6>
[817] 1908: L IText7 ITEXT <L IText8,35,1,82,LIText7>
[K17] 1909: L IText8 ITEXT <L IText9,42,1,84,LIText8>
[Y47] 1910: L IText9 ITEXT <L IText10,52,1,80,LIText9>
[ZL5] 1911: L IText10 ITEXT <0,59,1,81,LIText10>
[TF3] 1912: LIText db 'Datei:',0
[ER4] 1913: LIText2 db 'Dateiauswahl ',0
[963] 1914: LIText3 db 'Pfad:',0
[9G4] 1915: LIText4 db 'Entry types:',0
[8D4] 1916: LIText5 db 'write prot.:',0
[CG3] 1917: LIText6 db 'hidden',0
[WM3] 1918: LIText7 db 'system',0
[124] 1919: LIText8 db 'directory',0
[EK3] 1920: LIText9 db 'normal',0
[8L3] 1921: LIText10 db 'volume',0
[A45] 1922: L Border BORDER <0,0,0,69,21,0,5,1>
[BD4] 1923: farbtab db 1,2,3,4,5,6,7,8,9,10,11
[L02] 1924: db 12,13,14,15,16
[8W2] 1925: db 1 ; Rahmenfarbe
[H93] 1926: Farben db 000h,000h,000h,000h
[GG2] 1927: db 0ffh,0ffh,0ffh,0ffh
[M92] 1928: db 000h,0ffh,000h,000h
[M72] 1929: db 000h,000h,0afh,000h
[882] 1930: db 0ffh,000h,000h,000h
[VB2] 1931: db 0a9h,0a9h,0a9h,0a9h
[2D2] 1932: db 0ffh,0ffh,000h,000h
[Y52] 1933: db 0b0h,000h,000h,000h
[M92] 1934: db 000h,0ffh,000h,000h
[M92] 1935: db 000h,0ffh,000h,000h
[M92] 1936: db 000h,0ffh,000h,000h
[M92] 1937: db 000h,0ffh,000h,000h
[M92] 1938: db 000h,0ffh,000h,000h
[M92] 1939: db 000h,0ffh,000h,000h
[M92] 1940: db 000h,0ffh,000h,000h
[M92] 1941: db 000h,0ffh,000h,000h
[ER0] 1942: even
[Z15] 1943: Meldung db 'Gewählte Datei: $'
[ER0] 1944: even
[5Y5] 1945: NichtGewählt db 'Cancel gedrückt !$'
[ER0] 1946: even
[TM3] 1947: FileName db 12 DUP (' ')
[5U0] 1948: db '$$'
[PV2] 1949: Pfad db 'C:\*.*'
[FK1] 1950: db 43 DUP (0)
[KN0] 1951: db '$'
[ER0] 1952: even
[4G1] 1953: drive db 0
[ER0] 1954: even
[F14] 1955: BackBuffer dw 80*25*2 DUP(?)
[ST3] 1956: AllBuff dw 500*13 DUP(?)
[4M0] 1957: END
[000] 1958:
[000] 1959:
```

Mit »box.asm« steht Ihnen eine Dateiauswahlbox zum Einbau in eigene Programme zur Verfügung.

Grundlagen

Eine Rallye durch das Segment

Für die Systemprogrammierung benötigen Sie unter anderem Grundwissen über den Prozessor und die Adressierung des Speichers. Auch der Program Segment Prefix spielt dabei eine große Rolle. Einen Teil dieses umfangreichen Gebiets lernen Sie im folgenden kennen.

Wenn Sie sich mit der Systemprogrammierung beschäftigen wollen, müssen Sie nicht unbedingt Assembler-Profi sein. Systemprogrammierung ist auch mit C und Pascal realisierbar. Allerdings müssen Sie auf jeden Fall mit dem Prozessor, der Adressierung des Speichers und den Interrupts vertraut sein.

Die Aufgabe eines Prozessors ist die Abarbeitung von Maschinensprachebefehlen. Diese Befehle werden vom jeweils eingesetzten Hochsprachen-Compiler erzeugt. Die Maschinensprache ist dabei nicht für jeden Rechner identisch. Die verschiedenen Prozessoren der 80xxx-Familie arbeiten aber mit der gleichen Maschinensprache. Dadurch entsteht auch die Kompatibilität untereinander.

Den Mittelpunkt jedes Prozessors bilden die Register. Dort lädt der Prozes-

sor die Daten aus dem Speicher, um sie mit Hilfe von Maschinensprachebefehlen zu verarbeiten und danach wieder in den Speicher zurückzuschreiben. Für diese Datenmanipulation benötigt der Prozessor kaum Zeit, da sich die Register unmittelbar im Prozessor befinden. Es sind also keine Bus-Zugriffe notwendig.

Die Register sind auch deshalb interessant, weil sie den Informationsaustausch zwischen dem Programm und den DOS- und BIOS-Funktionen durchführen.

Für die Systemprogrammierer hat sich bei den Registern seit dem 8086 nichts geändert. Das BIOS und DOS unterstützen ausschließlich die 16-Bit-Register des Prozessors. Die 32-Bit-Register eines 80386 und 80486 können unter DOS also nicht genutzt werden. Die Darstellung in Bild 1 zeigt Ihnen den

Aufbau der Register eines 8088. Alle Register haben eine Größe von 2 Byte und können einen Wert zwischen 0 und 65 535 aufnehmen. Sie werden – je nach Aufgabe bei der Programmabarbeitung – in vier Gruppen unterteilt: allgemeine Register, Segmentregister, Programmzähler und Flag-Register. Für den Aufruf von DOS- und BIOS-Funktionen sind beispielsweise die allgemeinen Register zuständig. Die genauen Aufgaben der einzelnen Register entnehmen Sie Tabelle 1.

Diese Register werden mit »H« für »High« und »L« für »Low« bezeichnet. Dadurch unterteilt sich zum Beispiel das 16-Bit-Register AX in die 8-Bit-Register AH und AL. Demnach belegt das L-Register die acht niederwertigen Bit (0 bis 7) und das H-Register die acht höherwertigen Bit (8 bis 15). Durch diese Verknüpfung können Sie die drei Register nicht getrennt voneinander betrachten. Sobald Sie das zweite Bit des AH-Registers verändern, ändert sich auch das zehnte Bit des AX-Registers.

Bildung von Speicheradressen

Der Wert des X-Registers ergibt sich immer aus dem Wert des H- und des L-Registers. Sie multiplizieren den Wert des H-Registers mit 256 und addieren dazu den Wert des L-Registers.

Die Flag-Register dienen zur Kommunikation der aufeinanderfolgenden Maschinensprachebefehle. Bedingten Sprüngen oder Verzweigungen geht beispielsweise immer ein logischer Vergleich zweier Größen voraus. In Abhängigkeit vom Ergebnis dieses Vergleichs wird ein Flag gesetzt (= 1) oder gelöscht (= 0).

Für die Systemprogrammierung ist die Bildung von Speicheradressen der CPU äußerst wichtig. Die Anzahl der Speicherstellen, auf die ein Prozessor zugreifen kann, hängt von der Breite des Adreßregisters ab. Es liest bei jedem Speicherzugriff die Adresse der Speicherstelle, die in der CPU liegt. Weil jede Speicherstelle über eine eigene, fortlaufende Nummer angesprochen wird, entscheidet die Anzahl der darstellbaren Zahlen im Adreßregister, wie viele Speicherstellen angesprochen (adressiert) werden können.

Wie schon festgestellt, ist das Adreßregister 16 Bit groß. Damit Sie aber auch auf 1 MByte Speicher zugreifen können, benötigen Sie ein 20 Bit breites Adreßregister. Ein interessanter Trick erlaubt es, trotzdem eine 20 Bit breite Adresse zu bilden und auf den Adreßbus zu legen. Sie verknüpfen dazu zwei separate 16-Bit-Adressen. Die erste Adresse muß dabei unbedingt aus den Segmentregistern DS, ES, CS oder

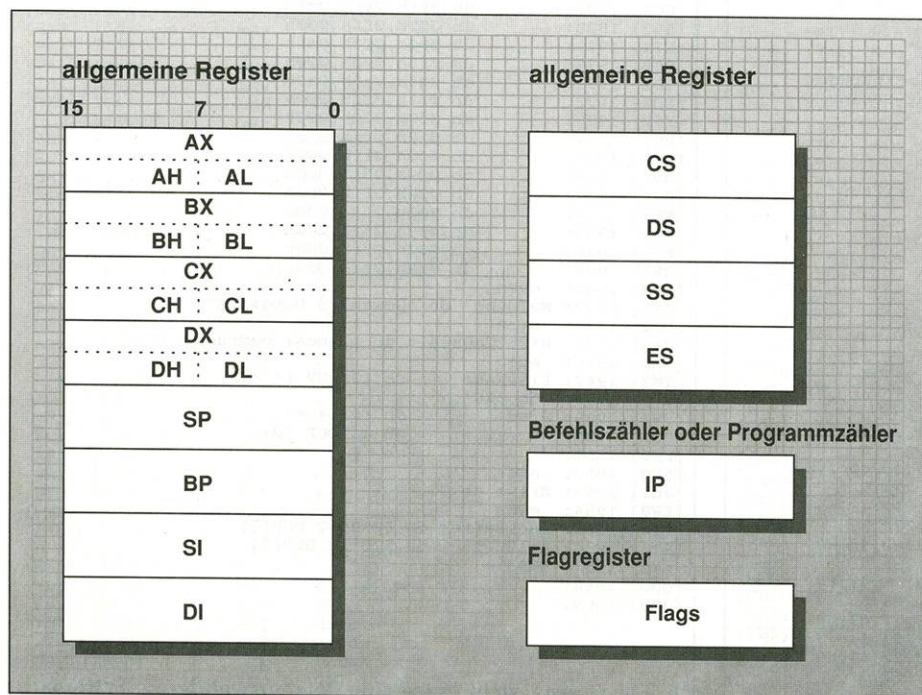


Bild 1. Aufbau der Register eines 8088-Prozessors

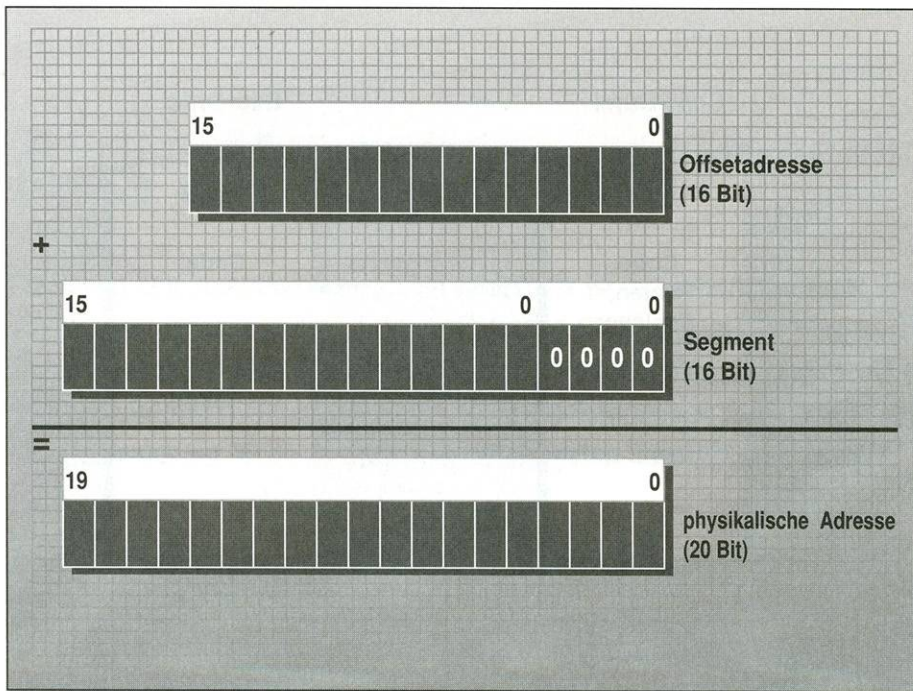


Bild 2. Bildung der physikalischen 20-Bit-Adresse von Registern

SS stammen. Die zweite Adresse entnehmen Sie einer beliebigen anderen Speicherstelle. Zur Bildung der 20-Bit-Adresse verschieben Sie zunächst den Inhalt des Segmentregisters (Segmentadresse) um vier Bit nach links, das heißt, Sie multiplizieren mit 16. Die zweite Adresse (Offsetadresse) addieren Sie anschließend einfach hinzu (Bild 2).

$(\text{Segmentadresse} \times 16) + \text{Offsetadresse} = \text{physikalische Adresse}$

Nun sind Sie mit dem nötigen Grundwissen für die weiteren Ausführungen ausgestattet.

Als MS-DOS-Anwender wissen Sie, daß es zwei verschiedene Arten von ausführbaren Programmen gibt: Com- und Exe-Programme. Die Com-Programme stammen noch aus der CP/M-Zeit, als der Hauptspeicher sehr knapp und die Programme nicht größer als 64 KByte waren. Diese Grenze sollen auch Com-Dateien nicht überschreiten. Für Exe-Dateien gilt diese Beschränkung allerdings nicht – sie dürfen beliebig groß sein. Grenzen setzt nur der Hauptspeicher.

In den maximal 64 KByte großen Com-Dateien müssen der Programmcode, die Daten und der Stack untergebracht sein. Daraus folgt, daß alle Segmentregister beim Programmstart und während der Programmausführung auf den Anfang des 64 KByte großen Speichersegments zeigen, das MS-DOS vor dem Einladen reserviert hat. Diese Werte müssen auch während der Programmausführung – mit Ausnahme des ES-Registers – beibehalten werden, damit der Programmcode in diesem Segment über das CS-Register, die Daten über das DS-Register und

der Stack über das SS-Register angesprochen werden können.

In Tabelle 2 finden Sie noch einmal alle gravierenden Unterschiede zwischen Com- und Exe-Dateien aufgelistet.

Bei Exe-Dateien ist die Anordnung der Segmente nicht so streng vorgegeben. Code, Daten und Speicher werden in verschiedenen Segmenten untergebracht.

Unabhängig davon, ob Sie eine Com- oder eine Exe-Datei starten, erzeugt MS-DOS vor dem Programmstart im Speicher die Datenstruktur PSP (Pro-

gram Segment Prefix). Dabei handelt es sich um einen 256 Byte großen Block, der jedem Programm im Arbeitsspeicher vorangeht. Er enthält wichtige Informationen, die DOS und das Programm für die Ausführung benötigen. Bei Com-Dateien enthalten nach dem Programmstart alle vier Segmentregister die Adresse des PSP. Bei der Exe-Datei zeigen nur das DS- und das ES-Register auf den PSP (Bild 3). Die ersten beiden Bytes 0 und 1 (Tabelle 3) enthalten den Befehl »Int20h«, mit dem ein Com-Programm beendet werden kann. Da das DS-Register stets auf den Beginn des PSP zeigt, können Sie das Programm durch Laden des DS-Registers auf den Stack und einen anschließenden Ret-Befehl beenden.

Der PSP enthält Informationen für das Programm

Die zweite Speicherstelle enthält eine 16-Bit-Zahl, die die oberste Segmentadresse des für das Programm reservierten Arbeitsspeichers festlegt. Benötigt Ihr Programm mehr Speicher, so kann es mit Hilfe dieses Werts feststellen, ob noch freier Speicherplatz vorhanden ist.

Die Speicherplätze 0Ahex bis 15hex enthalten die Adressen der Routinen des Interrupts 22hex zur Beendigung eines Programms, des Interrupts 23hex zur Reaktion auf [Strg-C] und des Interrupts 24hex, der auf kritische Fehler reagiert. Alle Interrupts dienen zur Sicherheit von MS-DOS. Ändert ein Programm diese Vektoren, so werden sie nach dem Programmende

Register Name		Aufgabe
AX	Akkumulator	für die Multiplikation, Division, für I/O und für die schnelle Verschiebung verantwortlich
BX	Basisregister	wird als Zeiger auf die Basisadresse im Datensegment verwendet
CX	Zählregister	enthält Zählwerte für das Wiederholen, Verschieben und Rotieren
DX	Datenregister	unterstützt das AX-Register bei 32-Bit-Operationen (bei Multiplikation und so weiter)
DI	Destination-Index	enthält die Zielzeichenkette und den Indexzeiger
SI	Source-Index	enthält die Quellzeichenkette und den Indexzeiger
SP	Stack-Pointer	ist ein Stackzeiger
BP	Base-Pointer	Zeiger auf die Basisadresse im Stacksegment
DS	Datensegment	enthält Daten und Werte zur Adressierung von Daten
CS	Codesegment	enthält Daten und Befehle
ES	Extrasegment	frei verwendbares Segment
SS	Stacksegment	Segment des Stacks
IP	Befehlszähler	enthält Offset der Befehle
Flags	Flags	Indikatoren für Prozessorstatus und Operationsergebnisse

Tabelle 1. Alle Register im Überblick mit ihren jeweiligen Aufgaben

durch die im PSP gespeicherten Originale ersetzt.

Der Speichbereich 18hex enthält einen Zeiger auf eine Tabelle, die die Handles der zur Zeit geöffneten Dateien beinhaltet. Ab MS-DOS 3.3 können auch mehr als 20 Handles vergeben werden. Deshalb gibt es ebenfalls einen Zeiger auf die neue Handle-Tabelle. Bei 2Chex ist die Segmentadresse des Environment-Blocks enthalten.

Der Environment-Block ist nichts anderes als ein Speicher, der die Strings aufnimmt, die auf der MS-DOS-Ebene durch die Befehle »SET« und »PATH« definiert wurden. Sie werden im Environment-Block als eine Folge von ASCII-Strings mit einer Null als Ende-Kennzeichen gespeichert.

So enthält dieser Datenblock Informationen, die ein Programm zur Ausführung benötigt, zum Beispiel den aktuellen Suchpfad für den Kommandoprozessor auf der Festplatte.

In den PSP-Bereich ab 80hex setzt die Funktion Exec eine als Kommandozeilenparameter übergebene Zeichenkette ein, deren Adresse die Funktion ebenfalls in ihrem Parameterblock erwartet. Da Exec den betreffenden Bereich einfach kopiert und keine eigene Analyse vornimmt, können Sie theoretisch eine beliebige Adresse übergeben – natürlich nur, wenn das aufgerufene Programm darauf eingerichtet ist.

Das PSP-Feld hat eine vorge-schriebene Größe

Da das PSP-Feld 128 Byte groß ist, ist die maximale Zeichenzahl auf 126 festgelegt.

Wenn Sie sich Tabelle 3 anschauen, werden Sie feststellen, daß eine größere Anzahl von Feldern zur Wahrung der Kompatibilität mit CP/M existiert,

Unterschied	Com-Datei	Exe-Datei
Größe	64 KByte	unbegrenzt
Startadresse	PSP:100hex	wird erst durch ein Start-Label festgelegt
Registerinhalte		
CS	PSP	Segment mit dem Start-Label
IP	100hex	Offset des Start-Label
SS	PSP	Segment mit Stack-Attribut
SP	FFFEhex	Größe des Stacksegments

Tabelle 2. Die wichtigsten Unterschiede von Com- und Exe-Dateien

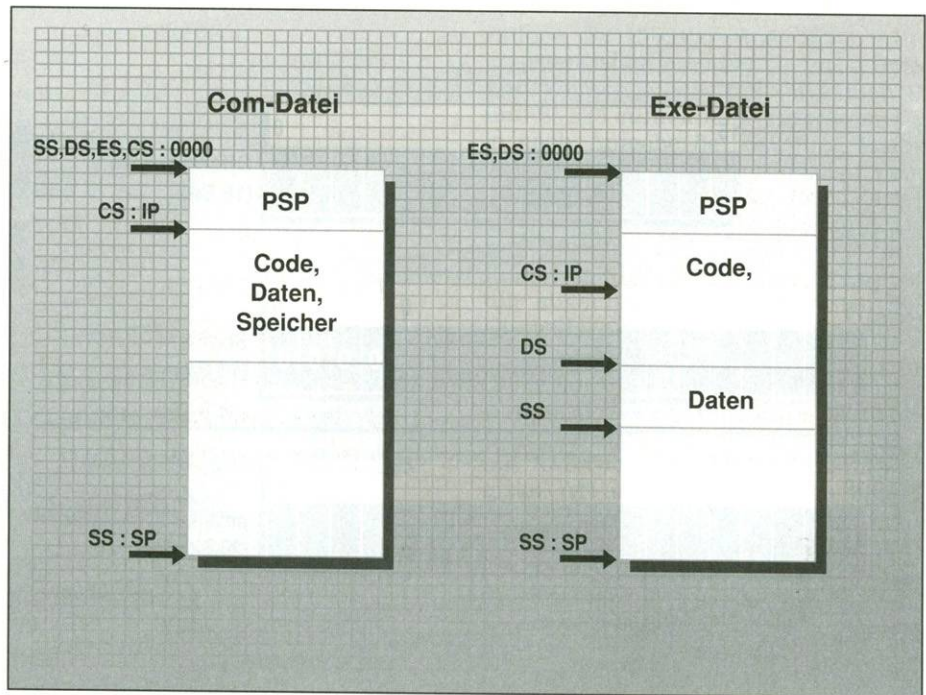


Bild 3. Vergleich einer Com- und einer Exe-Datei nach dem Einladen im Speicher

zum Beispiel die beiden File Control Blocks (FCB). In diesen FCB werden vor dem Programmstart die ersten beiden Parameter eingetragen, die Sie beim Programmaufruf mit dem Programmnamen angegeben haben. Deshalb müssen die Programme nur neu übersetzt werden. Änderungen der Logik beschränken sich auf Kleinigkeiten.

Als Systemprogrammierer haben Sie sich bestimmt schon über die verschiedenen reservierten Bereiche im PSP gewundert. Diese Bereiche werden aber von Microsoft offiziell nicht dokumentiert.

Microsoft will sich damit eine Hintertür für eventuelle Änderungen offenlassen.

Mit dem Interrupt 21hex, Unterfunktion 26hex erarbeiten Sie einen neuen PSP. Das heißt, der PSP eines ausgeführten Programms wird an eine bestimmte Adresse kopiert. Dazu müssen Sie im AH-Register 26hex und im DX-Register die Segmentadresse des neuen PSP angeben. Die Offsetadresse des neuen PSP ist dabei 0.

(uk)

Literatur:

- Arne Schäpers: DOS 5.0 für Programmierer, Addison-Wesley 1991, ISBN 3-89319-350-2
- Hans-Peter Messmer: PC-Hardwarebuch, Addison-Wesley 1992, ISBN 3-89319-357-X

Offset	Datentyp	Aufgabe
00hex	2 Byte	Interrupt 20hex, für Com-Dateien, die mit einem Ret-Befehl enden
02hex	Word	Segmentadresse der oberen Speichergrenze
04hex	Byte	reserviert
05hex	5 Byte	Aufruf des DOS-Funktionskoordinators
0Ahex	DWord	Adresse des Interrupts 22hex (Rücksprung zum aufrufenden Programm)
0Ehex	DWord	Adresse des Interrupts 23hex
12hex	DWord	Adresse des Interrupts 24hex
16hex	Word	PSP-Segmentadresse des aufrufenden Programms
18hex	–	Tabelle für Handles offener Dateien in der System File Table
2Chex	Word	Segmentadresse des eigenen Environment-Blocks
2Ehex	Word	Zwischenspeicher für das SP-Register
30hex	Word	Zwischenspeicher für das SS-Register
32hex	Word	(ab MS-DOS 3.0 definiert) enthält die Größe der Job File Table
34hex	DWord	(ab MS-DOS 3.0 definiert) enthält die Adresse der Job File Table
38hex	DWord	(ab MS-DOS 3.0 definiert) wird von der Funktion »share« genutzt
3Chex	20 Byte	reserviert
50hex	3 Byte	Einsprung in MS-DOS
53hex	2 Byte	reserviert
55hex	7 Byte	Beginn eines erweiterten Steuerblocks für Dateioperationen in FCB-Art (wird von Exe-Dateien nicht gesetzt)
5Chex	16 Byte	Steuerblock für Dateioperationen in CP/M-Art (File Control Block 1)
6Chex	16 Byte	File Control Block 2 (für Dateioperationen)
7Chex	4 Byte	reserviert
80hex	128 Byte	Länge der Kommandozeile

Tabelle 3. Diese Tabelle verschafft Ihnen einen Überblick über die PSP-Struktur

Interrupt 15hex

Aus alt mach neu

Einer von den bemerkenswertesten Interrupts ist der Interrupt 15hex. An Hand der in diesem Interrupt eingegliederten Funktionsgruppe können Sie die geschichtliche Entwicklung des BIOS nachverfolgen. Gleichzeitig zählt der Interrupt 15hex mit den neu hinzugekommenen Funktionen zum Thema Protected Mode zu einem der wichtigsten Interrupts des BIOS.

Die älteren Funktionen von Interrupt 15hex zählen zur Rubrik Hardware-Steuerung. Wenn Sie beispielsweise einen Joystick ansteuern wollen, eine Warteschleife einlegen müssen oder die Speichergröße ermitteln möchten, sind Sie mit den Funktionen des Interrupts 15hex gut bedient.

Eine weitere ursprüngliche und heute überflüssige Aufgabe erfüllt der Systemaufruf durch die Kontrolle eines angeschlossenen Kassettenlaufwerks. Diese antiquiert wirkenden Funktionen hat man jedoch mit moderneren erweitert.

● Rückblick auf den IBM-PC:

Zuerst ein geschichtlicher Rückblick: Die ersten IBM-PCs verfügten über einen Kassettenrekorder-Anschluß, so daß Sie Ihre Daten auf Kassetten speichern konnten. Diese Technik war eine Art Vorläufer zur späteren Disketten-Technologie. Für die Steuerung dieser heute fast vergessenen Hardware-Einheit war der Interrupt 15hex vorgesehen.

Doch nachdem sich die Disketten-Technik durchgesetzt hatte, waren die Eigenschaften des Interrupt 15hex veraltet und so nicht mehr gefragt.

● Protected Mode und der AT:

Doch dann begannen die Hardware-Konstrukteure mit der Entwicklung der AT-Computer, die über wesentlich mehr Eigenschaften verfügten, als der alte XT-Rechner. So kann ein AT in den sogenannten Protected Mode schalten, in dem es unter anderem die Speichermauer 640 KByte nicht mehr gibt und in dem ein Multitasking-System implementiert werden kann. Doch wer einfach in den neuen Modus umschalten will, ohne gewisse Vorbereitungen wie Tabellen-Adressen und andere bürokratisch anmutende Dinge zu erfüllen, der wird seinen Rechner eher in einen neuerlichen Kaltstart treiben, als daß er freudig vom Protected Mode begrüßt wird.

Die dazu nötigen Funktionen implementierten die BIOS-Entwickler in

den längst verstaubten Interrupt 15hex und verhalfen ihm so zu einer neuen Blüte.

Denn so wurde aus einem inzwischen eigentlich unbrauchbaren Interrupt eine sehr interessante Sammlung von wichtigen Funktionen.

Die ersten Funktionen des Interrupts 15hex sind wohl die am seltensten benutzten Aufrufe im PC, denn die wenigsten Computer-Besitzer werden einen PC/XT mit Kassettenrekorder-Anschluß und zugehörigem Kassettenrekorder besitzen.

Sollten Sie dennoch Eigentümer eines dieser betagten Geräte sein und auch tatsächlich einen Kassettenrekorder besitzen, der sich am PC anschließen läßt, dann haben Sie über die vier Funktionen des Interrupt 15hex endlich die Gelegenheit, Ihren Rekorder auch in Assemblerprogrammen zu nutzen.

Im Gegensatz zu den Disketten- und Festplattenroutinen schaltet der Computer bei Ein- und Ausgabeoperationen den Kassettenrekorder nicht automatisch ab.

Wenn Sie Funktion 00 nutzen wollen, dann sollten Sie mit einer Zeitverzögerung arbeiten, um den Start des Motors abzuwarten, bevor der Schreib-/Lesezugriff erfolgt.

Vielleicht möchten Sie den Rekorder aber auch nur abwechselnd ein- und ausschalten, zum Beispiel zur Beruhi-

gung der durch die Geschwindigkeit Ihrer Festplatte gestreßten Nerven oder als Geräuschkunterhaltung für Computerspiele.

Ganz anders sieht die Situation bei AT-Besitzern aus. In diesen Computern wurde der Interrupt 15hex erheblich erweitert.

Denn der Interrupt 15hex hatte beim AT-Computer eigentlich keinen Sinn mehr gehabt, da inzwischen die Entwicklung von Festplatten derart fortgeschritten war, daß man sie schon als Standardausrüstung für den AT-Rechner bezeichnen konnte. Damit verschwanden die im Vergleich zu den Festplatten viel langsameren und von der Speicherkapazität her viele kleineren Kassettenrekorder. Um also dem Interrupt 15hex einen neuen Nutzen zu geben, begann man mit der Erweiterung des Interrupt 15hex.

Der Schneider-PC nutzt die BIOS-Funktionen für die Verwaltung des batteriegepufferten RAM (»NVR«) und die Grafik-Bitplanes für den Grafikmodus 2.

Der AT hat mit dem Interrupt 15hex viele Funktionen erhalten, die das Betriebssystem sinnvoll ergänzen. Einige Routinen sind aber so kompliziert, daß Sie bei Interesse zu ihrer Erklärung auf weiterführende Literatur zurückgreifen sollten.

In den MS-DOS-Versionen 3.x erhält der Programmierer Zugriff auf die undokumentierte Funktion 20hex. Diesem Aufruf kommt nur in Zusammenhang mit dem Betriebssystem-Utility »print« eine Bedeutung zu.

Mit dieser Referenz und dem anderen DOS-Sier über die anderen BIOS-Interrupts besitzt Sie jetzt eine vollständige Sammlung aller wichtigen und nützlichen Funktionen des »Basic Input Output System«, kurz auch BIOS genannt.

Betrachten Sie diese Tabellenwerke, so können Sie nun ermesen, welches komplexes Programmwerk und wie wichtig das BIOS ist.

Durch die Konstruktion des BIOS mit all seinen Funktionen und Interrupts wird die Arbeit mit Ihrem Rechner gerade in der hardwarenahen Programmierung wesentlich erleichtert.

Mit den Unterbrechungen des BIOS können Sie zum Beispiel die Videokarte, den Bildschirm, das Disketten- und das Festplattenlaufwerk sowie deren Controller und andere Elemente Ihres Computers kontrollieren und steuern.

Die dazugehörigen Hardware-Interrupts haben wir natürlich nicht vergessen. Diese finden Sie in der Referenz »Interrupts bis zum Abwinken«. Dort gehen wir ebenfalls in die Interna der Funktionsammlungen und Parameter-tabelle der einzelnen Interrupts des BIOS.

(Markus Zietlow/st)

Der Kassettenstatus

AH Fehlermeldung

00	kein Fehler aufgetreten
01	ungültiger CRC (Prüfsummenfehler beim Lesen oder Verifizieren)
02	keine Datenübergänge gefunden
03	Daten auf Kassette nicht gefunden
04	kein Leitsatz (»Header«) gefunden
80	ungültiger Befehl

Tabelle 1. Das Status-Byte der veralteten Kassettenroutinen zeigt, ob ein Lese- oder Schreibversuch von Erfolg gekrönt ist.

Die Funktionen des Interrupt 15				
AH-Register	Computer	Eingabe	Ausgabe	Bedeutung
00	PC/XT	–	–	schalte Kassettenrekordermotor ein schalte Kassettenrekordermotor aus schalte Kassettenrekordermotor ein, lese Datenblock von Kassette und schalte Kassettenrekordermotor wieder aus
01	PC/XT	–	–	
02	PC/XT	CX=Anzahl der Bytes	AH=Kassettenstatus	
		ES:BX=Lesepuffer	DX=Anzahl der gelesenen Bytes ES:BX=Zeiger nach dem zuletzt gelesenen Byte	
03	PC/XT/AT	CX=Anzahl der Bytes	AH=Kassettenstatus	schalte Kassettenrekordermotor ein, schreibe Datenblock auf Kassette, und schalte Kassettenrekordermotor wieder aus
		ES:BX=Schreibpuffer	ES:BX=Zeiger nach dem zuletzt geschriebenen Byte	
20	AT	AL=00	–	gültig nur für MS-DOS-Version 3.x: das Utility »print« setzt das mit Unterfunktion 01 gesetzte Statuskennzeichen nicht, wenn es eine MS-DOS-Funktion ausführt gültig nur für MS-DOS-Version 3.x: das Utility »print« inkrementiert das angegebene Byte, wenn es eine MS-DOS-Funktion ausführt
		AL=01	–	
		ES:BX=Zeiger auf das zu inkrementierende Status-Byte		
80	AT	BX=Gerätenummer CX=Prozeßart	–	AT-Schnittstelle öffnen
81	AT	BX=Gerätenummer CX=Prozeßart	–	AT-Schnittstelle schließen
82	AT	BX=Gerätenummer	–	Programm beenden auf ein Ereignis warten
83	AT	AL=Auswahl =0: Intervall setzen =1: abbrechen CX-DX=Wartezeit in Mikrosekunden ES:BX=Zeiger auf den Speicher des auftretenden Programms	–	
84	AT	DX=Unterfunktionsnummer =0: momentaner Feuerknopf-Status abfragen =1: Joystick-Stellung ermitteln	C=0: Spiele-Adapter vorhanden AL=Status der Feuerknöpfe Bit 7=1: erster Feuerknopf des ersten Joysticks gedrückt Bit 6=1: zweiter Feuerknopf des ersten Joysticks gedrückt Bit 5=1: erster Feuerknopf des zweiten Joysticks gedrückt Bit 4=1: zweiter Feuerknopf des zweiten Joysticks gedrückt C=1: kein Spiele-Adapter angeschlossen AX=x-Position des ersten Joysticks BX=y-Position des ersten Joysticks CX=x-Position des zweiten Joysticks DX=y-Position des zweiten Joysticks	Joystick-Abfrage
85	AT	AL=Status =0: Taste wurde gedrückt =1: Taste wurde losgelassen	–	Funktion wird aufgerufen, wenn die SysReq-Taste gedrückt oder losgelassen wurde
86	AT	CX-DX=Wartezeit in Mikrosekunden	–	
87	AT	CX-DX=Anzahl der zu verschiebenden Worte ES:SI=Zeiger auf Global Descriptor-Tabelle	C=0: kein Fehler C=1: Fehler aufgetreten, AH enthält Fehlercode =1: RAM-Paritäts-Fehler =2: GDT bei Funktionsaufruf nicht in Ordnung =3: Protected-Mode konnte nicht richtig initialisiert werden	Block verschieben
88	AT	–	AX=Größe des Speichers	
89	AT	–	–	Speichergröße über 1 MByte in KByte ermitteln in den Protected Mode des Prozessors umschalten »Gerät ist beschäftigt«-Schleife aufrufen Flags setzen und Interrupt beenden
90	AT	AL=Typencode	–	
91	AT	AL=Typencode	–	

Tabelle 2. Sinnvolle Systembefehle haben die heutzutage überflüssigen Funktionen zum Ansteuern eines angeschlossenen Kassettenrekorders ergänzt. Zum Beispiel kann man mit Interrupt in den Protected Mode umschalten.

BIOS-Interrupts

Interrupts bis zum Abwinken

Die Interrupt-Struktur des BIOS (Basic Input Output System) ist mindestens ebenso komplex wie die Struktur der MS-DOS Interrupts. Dieses Tabellenwerk macht die Menge der BIOS-Interrupts durchschaubar.

Interrupts, ihre Funktionen und Unterfunktionen geben Ihnen die direkte Kontrolle und Informationen über die an den Prozessor angeschlossene Hardware.

Mit ihren mächtvollen Funktionsbibliotheken bedeutet die Programmierung also für die Hardware eine große Erleichterung.

Die Gesamtstruktur der BIOS-Interrupts mit all Ihren Ausnahmen zwischen speziellen Rechner- und BIOS-Typen würde allerdings mindestens das Dreifache des jetzigen Umfangs beanspruchen. Die Besonderheiten

Wert in BL Beschreibung (in hex)

0	High-Intensitätsattribut ignorieren
1	von Intensität zu »unterstrichen« wechseln
2	Intensität umkehren (reverse video)
3	von Intensität zu ausgewähltem Alternativfont
B0	halbe Intensität (Compaq)
B1	Interpretation des aktiven Intensitätsbits wechseln (Compaq-Portable 386)

Tabelle 2. Intensitätsbits

Code	Beschreibung
0	kein Display
1	monochromer Adapter mit monochromem Display
2	CGA mit Farbdisplay
3	reserviert oder unbenutzt
4	EGA mit Farbdisplay
5	EGA mit Monochromdisplay
6	PGA mit Farbdisplay
7	VGA mit Monochrom-Analogdisplay
8	VGA mit Farb-Analogdisplay
9	reserviert oder unbenutzt
0A	MCGA mit Farb-Digitaldisplay
0B	MCGA mit Monochrom-Analogdisplay
0C	MCGA mit Farb-Analogdisplay
FF	unbekannter Displaytyp

Tabelle 1. Die Codes für den Bildschirmmodus

Wert in BL (in hex) Videomodus

13	320×200
2D	640×350
2E	640×480
2F	640×400
30	800×600

Tabelle 3. Der erweiterte Grafikmodus mit 32 768 Farben für das ET-4000-Chip-BIOS

Modusnummer	Modus	Auflösung	Zeilen und Farbmodus zulässige Grafikkarten Spalten
0	Text	320×200	40×25 einfarbig CGA, VGA
1	Text	320×200	40×25 16 Farben CGA, VGA
2	Text	640×200	80×25 einfarbig monochrom
3	Text	640×200	80×25 16 Farben CGA, VGA
4	Grafik	320×200	40×25 einfarbig CGA, VGA
5	Grafik	320×200	40×25 vier Farben CGA, VGA
6	Grafik	640×200	80×25 einfarbig CGA, VGA
7	Text	720×350	80×25 einfarbig monochrom, Hercules
13	Grafik	320×200	40×25 16 Farben EGA, VGA
14	Grafik	640×200	80×25 16 Farben EGA, VGA
15	Grafik	640×350	80×25 einfarbig EGA, VGA
16	Grafik	640×350	80×25 16 aus 64 Farben EGA, VGA
18	Grafik	640×480	80×25 16 Farben VGA
19	Grafik	320×200	80×25 256 Farben VGA

Tabelle 5. Die verschiedenen Alternativen für den Videomodus

der unzähligen IBM-kompatiblen PCs sowie der unermesslichen Anzahl von Grafikkarten werden hier daher nicht berücksichtigt.

In Tabelle 17 dieser fast 40 Tabellen umfassenden Übersicht bekommen Sie einen Einblick über sämtliche BIOS-Unterbrechungen und deren Bedeutung sowie der Information, ab welcher Tabelle die eigentlichen internen Informationen zu finden sind.

Ob Sie BIOS-Interrupts nun im professionellen Bereich oder privat nutzen wollen, diese Referenz soll Ihnen den Weg durch die BIOS-Interrupts erleichtern. Ciao!

(st)

Bit Bedeutung wenn gesetzt

Shift-Flags Nr. 1

7	SysAbf-Taste gedrückt
6	CapsLock-Taste gedrückt
5	NumLock-Taste gedrückt
4	Rollen-Taste gedrückt
3	AltGr-Taste gedrückt
2	rechte Strg-Taste gedrückt
1	Alt-Taste gedrückt
0	linke Strg-Taste gedrückt

Shift-Flags Nr. 2

7	Einfüge-Status aktiv
6	CapsLock-Taste aktiv
5	NumLock-Taste aktiv
4	Rollen-Taste aktiv
3	Alt-Taste gedrückt
2	Strg-Taste gedrückt
1	linke Umschalt-Taste gedrückt
0	rechte Umschalt-Taste gedrückt

Tabelle 4. Die Shift-Flags von Interrupt 16hex für den AT

Wert in BL Fontzeiger (in hex)

0	Zeiger auf Interrupt 1Fhex
1	Zeiger auf Interrupt 43hex (43hex beinhaltet die Adresse der Grafikdaten der Zeichen 00hex bis 0Fhex des aktuellen Font)
2	8×14 ROM-Zeichenfont Zeiger
3	8×8 double dot ROM-Fontzeiger
4	8×8 double dot ROM-Fontzeiger (obere Hälfte)
5	9×14 ROM-Fontzeiger (monochrom)
6	8×16 ROM-Fontzeiger (VGA, MCGA)
7	9×16 ROM-Fontzeiger (VGA)
11	8×20 Fontzeiger (Ultra Vision v2+, VGA)
11	8×19 Fontzeiger (autosync EGA)
12	8×10 Fontzeiger (Ultra Vision v2+, VGA)
12	8×11 Fontzeiger (autosync EGA)

Tabelle 6. Die verschiedenen Fontzeiger der Grafikkarten und des BIOS

Funktionsnummer (in hex)	Kurzbeschreibung	Aufruf	Ergebnis	Beschreibung
0	Videomodus wählen	AH=00hex, AL=Modus (Tabelle 5)	-	Mit dieser Funktion legen Sie den Videomodus des Bildschirms fest.
1	Cursorgroße wählen	AH=01hex, CH=Startzeile, CL=Endzeile	-	legt die Größe des Cursors fest; die Angabe Zeile bezieht sich auf die Zeilenzahl innerhalb des Bereichs, in der ein Zeichen dargestellt wird; sinnvolle Werte: monochrom/Hercules: 0 bis 13 Zeilen EGA/VGA: 0 bis 21 Zeilen CGA: 0 bis 7 Zeilen
2	Cursorposition wählen	AH=02hex, bh=Bildschirmseite, DH=Zeile, dl=Spalte	-	positioniert den Cursor auf eine bestimmte Bildschirmseite; die Anzahl der Bildschirmseiten wird vom Grafikadapter festgelegt und variiert
3	Cursorgroße und -position bestimmen	AH=03hex, BH=Bildschirmseite	CH=Startzeile der Größe, CL=Endzeile der Größe, DH=Zeile der Position, DL=Spalte der Position	bestimmt die Größe und Position des Cursors, Bildschirmseite siehe Funktion 02hex, Cursorgroße siehe Funktion 01hex
4	Lichtstiftposition bestimmen	AH=04hex	AH=0 wenn kein Lichtstift vorhanden sonst 1, CH=Zeile im Grafikmodus (0 bis 199), BX=Spalte im Grafikmodus (0 bis 639), DH=Zeile im Textmodus (0 bis 24), DL=Spalte im Textmodus (0 bis 79)	bestimmt die Position eines Lichtstifts; im Register AH wird ein Flag gesetzt, wenn ein Stift angeschlossen ist
5	Bildschirmseite wählen	AH=05hex, AL=Bildschirmseite	-	legt die anzuzeigende Bildschirmseite
6	Bildschirm nach oben rollen (schieben)	AH=06hex, AL=Zeilenanzahl, BH=Farbattribute der neuen Zeilen, CH=Zeile links oben, CL=Spalte links oben, DH=Zeile rechts unten, DL=Spalte rechts unten	-	schiebt den Inhalt eines definierten Fensters vom Bildschirm nach oben; AL=0 löscht das Fenster
7	Bildschirm nach unten rollen (schieben)	AH=07hex, AL=Zeilenanzahl, BH=Farbattribute der neuen Zeilen, CH=Zeile links oben, CL=Spalte links oben, DH=Zeile rechts unten, DL=Spalte rechts unten	-	schiebt den Inhalt eines definierten Fensters vom Bildschirm nach unten; AL=0 löscht das Fenster
8	Zeichen an aktueller Position einlesen	AH=08hex, BH=Bildschirmseite	AH=Farbattribut, AL=ASCIIcode	ließt das Zeichen an der aktuellen Position ein
9	Zeichen an aktueller Position ausgeben	AH=09hex, AL=ASCIIcode, BH=Bildschirmseite, BL=Farbattribut des Zeichens, CX=Anzahl der Ausgabe	-	gibt ein Zeichen an der aktuellen Position beliebig oft aus; aktuelle Position wird nicht geändert; im Grafikmodus erfolgt kein Zeilenumbruch
0A	Zeichen an aktueller Position ausgeben	AH=0Ahex, AL=ASCIIcode, BH=Bildschirmseite, CX=Anzahl der Ausgabe	-	gibt ein Zeichen an der aktuellen Position mit dem aktuellen Farbattribut beliebig oft aus; aktuelle Position wird nicht geändert; im Grafikmodus erfolgt kein Zeilenumbruch
0B	Farben festlegen	siehe Tabelle 12	siehe Tabelle 12	legt fest, welche Farben angezeigt werden sollen; besteht aus Unterfunktionen (Tabelle 12)
0C	Pixel ausgeben	AH=0Chex, AL=Farbwert (je nach Palette und Grafikkarte), CX=Spalte im Grafikmodus, DX=Zeile im Grafikmodus	-	gibt ein Pixel an der gewählten Stelle aus
0D	Pixelfarbe bestimmen	AH=0Dhex, CX=Spalte im Grafikmodus, DX=Zeile im Grafikmodus	AL=Pixelfarbe (je nach Palette und Grafikkarte)	gibt die Farbe des angewählten Pixels an
0E	Zeichen und Steuercodes ausgeben	AH=0Ehex, AL=ASCIIcode, BH=Bildschirmseite, BL=Vordergrundfarbe	-	gibt Zeichen an aktueller Position aus; berücksichtigt Steuercodes (Wagenrücklauf, Backspace,...); aktuelle Position wird angepaßt; im Textmodus wird statt Vordergrundfarbe das aktuelle Farbattribut des Zeichens benutzt
0F	Videomodus bestimmen	AH=0Fhex	AH=Zeichenanzahl pro Zeile, AL=Videomodus (Tabelle 5), BH=aktuelle Bildschirmseite	bestimmt Videomodus (Tabelle 5) sowie maximale Zeichenanzahl pro Zeile und aktuelle Bildschirmseite;
10	setzt Register für Farbpalette	siehe Tabelle 13	siehe Tabelle 13	setzt mit den Unterfunktionen die Register für die Farbpalette (ab EGA)
11	Funktionen für Textmodus	-	-	Bibliothek für den Zeichengenerator im Textmodus
12	weitere Videografikfunktionen	siehe Tabelle 11	siehe Tabelle 11	enthält weitere Unterfunktionen für Grafikansteuerung (Tabelle 11)
13	Zeichenkette ausgeben	AH=13hex, AL=Ausgabemodus (Tabelle 26), BH=Bildschirmseite, BL=Farbattribut, CX=Länge der Zeichenkette (eventuelle Attribute werden nicht mitgezählt), DH=Zeile im Textmodus, DL=Spalte im Textmodus, ES:BP=Adresse der Zeichenkette	-	gibt eine Zeichenkette aus; Zeichenkette kann Attribut enthalten (siehe Ausgabemodus und Tabelle 26); Steuercodes wie Wagenrücklauf werden berücksichtigt
14	Zeichenfont ändern	siehe Tabelle 10	siehe Tabelle 10	enthält Unterfunktion, um Zeichenfonts zu ändern
15	Anzeigeparameter ermitteln	AH=15hex	AX=Display-Typ (0:kein, 5140hex: LCD, 5153: CGA, 5151: mono), ES:DI=Adresse der Displayparameter-Tabelle (Tabelle 27)	ermittelt die Parameter der Anzeige (funktioniert nicht immer)
15	nur für Sperry-PC	AH=15hex, AL=Superimpose-Modus (0:Grafikschirm zeigen, 1:Textschirm zeigen, 2:Text superimposed im Grafikdisplay bringen)	-	Diese Funktion gilt nur für Sperry-PCs und erlaubt die gleichzeitige Anzeige von Text- und Grafikmodus. Superimpose könnte man als »Superschwindel« übersetzen. Funktion steht im Konflikt mit der normalen Funktion 15hex.
1A	Display kombinieren	AH=1Ahex, AL= 0 für Kombinationscode lesen und 1 für Code setzen, BH=alternativer Displaycode (Tabelle 1), BL=aktiver Code	AL=1Ahex, wenn Funktion unterstützt wird, BH=alternativer Displaycode, BL=aktiver Code	setzt und liest den Displaykombinationscode (Tabelle 1) aus
1B	Video-Setup ermitteln	AH=1Bhex, BX=0, ES:DI=64 Byte Puffer	AL=1Bhex, wenn kein Fehler, ES:DI=Status-Informationen	ermittelt sämtliche Video- und BIOS-Setup-Informationen
1C	Videostatus sichern/setzen	AH=1Chex, AL=0 für Statuspuffergröße ermitteln oder 1 für Videostatus sichern oder 2 für Videostatus zurücksetzen, CX=abzuzugender Status, ES:BX=Statuspuffer (bei AL=1 oder 2)	AL=1Chex, wenn kein Fehler, BX=Anzahl der benötigten 64-Byte Blöcke	sichert oder setzt den Videostatus

Tabelle 7. Die Funktionen der Video-Interrupt 10hex

Offset-Adresse	Größe	Beschreibung
0	16 Byte	Registerwerte des 6845 für Modi 0 und 1
10	16 Byte	Registerwerte des 6845 für Modi 2 und 3
20	16 Byte	Registerwerte des 6845 für Modi 4 und 5
30	16 Byte	Registerwerte des 6845 für Modi 6 und 7
40	Word	Bytezahl im Videopuffer für Modi 0 und 1
42	Word	Bytezahl im Videopuffer für Modi 2 und 3
44	Word	Bytezahl im Videopuffer für Modi 4 und 5
46	Word	Bytezahl im Videopuffer für Modi 6 und 7
48	8 Byte	Bildschirmspalten für alle Modi (0 bis 7)
50	8 Byte	CRT-Controller-Modus-Byte für alle Modi (0 bis 7)

Tabelle 8. Der Aufbau der Videoparameter-Tabelle

Gesetzte Bits in Register BL	Blockeigenschaften
0 und 1	Block mit Zeichenattribut Bit 3=0
2 und 3	Block mit Zeichenattribut Bit 3=1
0, 1 und 4	Block nur mit Attributbit 3=0
2, 3 und 5	Block nur mit Attributbit 3=1

Tabelle 9. Die verschiedenen Eigenschaften der darzustellenden Blöcke

Unterfunktion (in hex)	Kurzbezeichnung	Aufruf	Ergebnis	Beschreibung
0	LCD-Fonts laden	AX=1400hex, BH=Byte/Zeichen (08hex oder 10hex (Compaq)), BL=0 für Hauptfont laden (Block 1) und 1 für alternatives Font laden (Block 2), CX=zu speichernde Zeichenzahl, DX=Zeichenoffset im Fontbereich vom Arbeitsspeicher, ES:DI=Adresse des Zeichenfont	–	wechselt die Zeichenfonts, ist definiert als »benutzerdefinierte LCD-Zeichenfont laden«
1	Zeichen-Fonts laden	AX=1401hex, BL=0 für Hauptfont laden (Block 1) und 1 für ALternatives Font laden (Block 2)	–	lädt Standardzeichenfonts des System-ROM
2	Mapping der LCD-Attribut setzen	AX=1402hex, BL=Attributwert setzen (Tabelle 2)	–	setzt die Intensitätsattribute bei LCD-Bildschirmen

Tabelle 10. Die Unterfunktionen der Funktion 14hex des Interrupts 10hex

Unterfunktion (in hex)	Kurzbezeichnung	Aufruf	Ergebnis	Beschreibung
Unterfunktionen werden bei Funktion 12hex im Register BL aufgerufen				
10	EGA-Informationen ermitteln	AH=12hex, BL=10hex	BH=0 bei Farbmodus (I/O Port 3Dxhex) oder 1 bei Monomodus (I/O Port 3Bxhex), BL=Videospeicher (0: 64KByte, 1: 128 KByte, 2:192 KByte, 3: 256 KByte), CH=Funktionsbits (EGA), CL=DIP-Schalter-Einstellung	ermittelt Informationen zur Konfiguration der Grafikkarte
20	alternatives Hardcopy aktivieren	AH=12hex, BL=20hex	–	aktiviert die Hardcopy-Routine im Grafikkarten-BIOS
30	Rasterzeilen wählen	AH=12hex, BL= 30hex, AL=0 für 200 Linien (8x8) oder 1 für 350 Linien (8x14) oder 2 für 400 Linien (9x16)	AL=12hex, wenn kein Fehler	wählt die Rasterzeilen für den Textmodus aus
31	Palette laden kontrollieren	AH=12hex, BL=31hex, AL=0 Laden erlaubt oder 1 für Laden nicht erlaubt	–	kontrolliert das Laden von Standardpaletten
32	Speicherzugriffskontrolle	AH=12hex, BL=32hex, AL=0 Zugriff erlaubt oder 1 für Zugriff untersagt	AL=12hex, wenn kein Fehler	kontrolliert CPU-Zugriffe auf den Videospeicher
33	Graustufenkontrolle	AH=12hex, BL=33hex, AL=0 für erlaubt und 1 für nicht erlaubt	AL=12hex, wenn kein Fehler	kontrolliert die Umrechnung von Farbwerten in Graustufen
34	Cursoremulationskontrolle	AH=12hex, BL=34hex, AL=0 für erlaubt und 1 für nicht erlaubt	AL=12hex, wenn kein Fehler	kontrolliert den Zugriff auf die Cursoremulation
35	Monitor wechseln	AH=12hex, BL=35hex, AL= siehe Tabelle 15, ES:DX=Puffer (128 Byte Speicherbereich bei AL = 0 1 oder 3)	AL=12hex, wenn kein Fehler	wechselt zwischen zwei angeschlossenen Bildschirmen. In den Puffer werden die Statusdaten der Karte eingelesen.
36	Bildschirmkontrolle	AH=12hex, BL=36hex, AL=0 für erlaubt und 1 für nicht erlaubt	AL=12hex, wenn kein Fehler	schaltet den aktiven Bildschirm aus oder wieder an
Unterfunktion 55hex ist nur für ATI, Tatung- und Taxan-Grafikarten verfügbar				
55	Erweiterte Funktionen	siehe Tabelle 19	siehe Tabelle 19	hier werden weitere Unterfunktion für bestimmte Grafikkarten zur Verfügung gestellt (Tabelle 19)

Tabelle 11. Die Unterfunktionen der Funktion 12hex von Interrupt 10hex

Funktionsnummer	Kurzbeschreibung	Aufruf	Ergebnis	Beschreibung
0	Rahmen-/Hintergrundfarbe setzen	AH=0Bhex, BH=00hex, BL=Rahmen-/Hintergrundfarbe (Tabelle 30)	–	legt die Rahmenfarbe im Textmodus und die Hintergrundfarbe im Grafikmodus fest. Die Farbwerte werden bitweise interpretiert und haben eine Größe von 4 Bit; Farbwerte siehe Tabelle 30
1	Farbpalette wählen	AH=0Bhex, BH=01hex, BL=Farbpalettennummer	–	setzt die aktive Farbpalette (Tabelle 31) für den Grafikmodus; nicht alle Grafikkarten unterstützen alle Farbpaletten (maximal 4)

Tabelle 12. Weitere Prozeduren für Funktion 0Bhex aus Interrupt 10hex

Unterfunktion (in hex)	Kurzbezeichnung	Aufruf	Ergebnis	Beschreibung
0	setzt ein Palettenregister	AX=1000hex, BH=Farbwert, BL=Palettenregisternummer (00hex bis 0Fhex) oder Attributregisternummer	–	setzt ein Palettenregister und den Farbwert
1	setzt Overscan-Register (Rahmenfarbe)	AX=1001hex, BH=Farbwert	–	diese Funktion ändert die Rahmenfarbe
2	setzt alle Palettenregister	AX=1002hex, ES:DX=Adresse einer Palettenregisterrabelle	–	setzt alle Palettenregister mit den Angaben einer 17 Byte großen Tabelle: Offset 00hex bis 0Fhex (16 Byte Größe) Farben für die Register 00hex bis 0Fhex, Offset 10hex (1 Byte Größe) Rahmenfarbe.
3	Intensitäts-/Blinkbit setzen	AX=1003hex, BL=0 für Hintergrundintensität und 1 für Blinkbit	–	setzt die Hintergrundintensität oder das Blinkbit
7	liest ein Palettenregister	AX=1007hex, BL=Paletten- oder Attributregister (siehe Unterfunktion 00hex)	BH=Registerwert	ermittelt den Wert des Paletten- oder des Attributregisters (siehe Unterfunktion 00hex)
8	Overscan-(Rahmen)-Register lesen	AX=1008hex	BH=Registerwert	liest den Wert aus dem Overscan-Register aus; Ergebnis ist der Farbwert für den Rahmen
9	Werte aller Palettenregister ermitteln	AX=1009hex, ES:DX=Ergebnispuffer (siehe Unterfunktion 02hex)	Ergebnis in ES:DX-Puffer	ermittelt die Werte aller Register und schreibt sie in den Puffer an der Adresse ES:DX; Aufbau des Puffers siehe Unterfunktion 02hex
10	ein DAC-Register setzen	AX=1010hex, BX=Register, ch=Grünwert (0 bis 63), CL=Blauwert (0 bis 63), DH=Rotwert (0 bis 63)	–	Setzt neue Farbmischwerte im Register des Digital-Analog-Konverter (DAC)
12	mehrere DAC-Register setzen	AX=1012hex, BX=erstes Farbreakister, CX=Anzahl der DAC-Register, ES:DX=Tabelle mit den neuen RGB-Werten (3×CX Byte Größe)	–	setzt eine ausgewählte Anzahl der Register des Digital-Analog-Konverter (DAC); die Tabelle in ES:DX hat dabei folgenden Aufbau: Grünwert1 (Byte), Blauwert1 (Byte), Rotwert1, Grünwert2, Blauwert2, ... wählt die Seite beziehungsweise den Pagingmode aus.
13	Video-DAC-Farbseite auswählen	AX=1013hex, bei BL=0 (Pagingmode) gilt: BH= 0 (4 Blöcke aus 64) oder 1 (16 Blöcke aus 16); bei BL=1 (Page) gilt: BH=Seitennummer (00hex bis 03hex oder 00hex bis 0Fhex)	–	
15	ein DAC-Register lesen	AX=1015hex, BL=Registernummer	CH=Grünwert, CL=Blauwert, DH=Rotwert	liest die Farbmischwerte eines einzelnen DAC-Registers aus
17	mehrere DAC-Register lesen	AX=1017hex, BX=erstes Register, CX=Anzahl der Register, ES:DX Lesebuffer	Ergebnis in ES:DX	Liest den Inhalt mehrerer Register in einen Puffer an der Adresse ES:DX (Aufbau siehe Unterfunktion 12hex) ein
18	PAL-Wert setzen	AX=1018hex, BL=neuer PAL-Wert	–	setzt das PAL mit einem neuen Wert und ändert somit die Zeichenfarbe
19	PAL-Wert lesen	AX=1019hex	BL=PAL-Wert	liest den aktuellen Wert des PAL aus
1A	Status der Video-DAC-Farbseite ermitteln	AX=101Ahex	BL=BH=aktuelle Seite, BL=Seitenmodus (siehe Unterfunktion 13hex)	ermittelt den Seitenmodus und die aktuelle Farbseite des DAC
1B	Farbe in Graustufen umwandeln	AX=101Bhex, BX=erstes DAC-Register, CX=Anzahl der Register	–	konvertiert die aktuellen Farben der DAC-Register in äquivalente Graustufen
F0	Hicolor-Grafikmodus setzen	AX=10F0hex, BL=Videomodus (Tabelle 3)	Bei Fehler in AX Fehlercode	Setzt bei Grafikkarten mit dem ET-4000-Chip-BIOS den erweiterten Grafikmodus mit 32 768 Farben (Tabelle 3)
F1	DAC-Typ ermitteln	AX=10F1hex	BL=DAC-Typ (0: normaler DAC-Typ, 1: Sierra-SC1148x Hicolor und andere Hicolor-DAC-Typen)	ermittelt den Typ des Digital-Analog-Konverter mit dem ET-4000-BIOS

Tabelle 13. Die Bibliothek der Funktionen 10hex aus Interrupt 10hex

Funktions- nummer (in hex)	Kurzbezeichnung	Eingabe	Ausgabe	Beschreibung
0	initialisiert Laufwerk	AH=00hex, DH=00hex für Diskettenlaufwerke sonst Code der Festplatte (80hex für erste Festplatte, 81hex für die zweite Platte usw. bis 87hex für die letzte Platte)	Bei Fehler: Carry-Flag gesetzt und AH=Fehlernummer (Tabelle 18), sonst AH=00hex	löst ein Reset bei Diskettenlaufwerk oder Festplatte und deren Controller aus
1	Fehlercode lesen	AH=01hex, DH= siehe Funktion 00hex	siehe Funktion 00hex	Bei Fehler: siehe Funktion 00hex, sonst AH=Fehlercode (Tabelle 18), ermittelt den Fehlercode der letzten Operation des zuletzt benutzten Laufwerks
2	Diskettensektor lesen	AH=02hex, AL= Sektoranzahl, CH=Spur, CL=Sektor, DH=Diskettenseite, DL=Laufwerk (0 für A, 1 für B, usw.), ES:BX=Adresse des Puffers (Größe pro Sektor 512 Byte)	siehe Funktion 00hex und Tabelle 18	liest physikalische Sektoren innerhalb einer Diskettenseite
2	Festplattensektor lesen	wie zuvor, aber: CH=Zylindernummer, DH=Kopfnummer, DL=Festplattencode (siehe Funktion 00hex)	siehe Funktion 00hex und Tabelle 18	liest physikalische Sektoren innerhalb eines Zylinders von Festplatte; die Bits 6 und 7 vom Register CL bilden die Bits 8 und 9 des Register CH, um maximal 1024 Zylinder anzusprechen
3	Diskettensektor schreiben	AH=03hex, sonst wie Funktion 02hex für Disketten	siehe Funktion 00hex und Tabelle 18	schreibt physikalische Sektoren innerhalb einer Diskettenseite
3	Festplattensektor schreiben	AH=03hex, sonst wie Funktion 02hex für Festplatten	siehe Funktion 00hex und Tabelle 18	schreibt physikalische Sektoren innerhalb eines Zylinders von Festplatte; die Bits 6 und 7 vom Register CL bilden die Bits 8 und 9 des Register CH, um maximal 1024 Zylinder anzusprechen
4	Diskettensektor prüfen	AH=04hex, sonst wie Funktion 02hex für Disketten	siehe Funktion 00hex und Tabelle 18	prüft physikalische Sektoren innerhalb einer Diskettenseite
4	Festplattensektor prüfen	AH=04hex, sonst wie Funktion 02hex für Festplatten	siehe Funktion 00hex und Tabelle 18	prüft physikalische Sektoren innerhalb eines Zylinders von Festplatte; die Bits 6 und 7 vom Register CL bilden die Bits 8 und 9 des Register CH, um maximal 1024 Zylinder anzusprechen
5	Diskettenspur formatieren	AH=05hex, sonst wie Funktion 02hex für Disketten	siehe Funktion 00hex und Tabelle 18	formatiert physikalische Sektoren innerhalb einer Diskettenseite; Aufbau des Formatpuffers ES:BX siehe Tabelle 16
8	Drive-Parameter ermitteln	AH=08hex, DL=Laufwerk (0: A, 1: B,...) bit 7 gesetzt bei Festplatten	Bei Fehler Carry-Flag gesetzt und AH=Fehlercode (Tabelle 18); sonst AH=0, BL=Typ (Tabelle 16), CH=niedrigen 8 Bit der maximalen Zylinderzahl, CL=maximale Sektorzahl (Bit 5 bis 0) und höheren 2 Bit der maximalen Zylinderzahl (Bit 7 und 6) DH=Kopffanzahl,	ermittelt die Parameter eines Laufwerkes
9	Controller initialisieren	AH=09hex, DL=Laufwerk (80hex oder 81hex)	bei Fehler Carry-Flag gesetzt und AH=Fehlercode	reinitialisiert den Controller mit den Laufwerksparametern
0A	große Sektoren lesen	AH=0Ahex, AL=Sektoranzahl, CH und CL siehe Funktion 08hex, DH=Kopfnummer, DL=Laufwerk (80hex oder 81hex), ES:BX=Datenpuffer	bei Fehler Carry-Flag gesetzt und AH=Fehlercode (Tabelle 18) und AL=Anzahl gelesener Sektoren	liest Sektoren von einer Festplatte
0B	große Sektoren schreiben	AH=0Bhex, AL, CH, CL, DH, DL und ES:BX= wie Funktion 0Ahex	bei Fehler Carry-Flag gesetzt und AH=Fehlercode (Tabelle 18) und AL=Anzahl geschriebener Sektoren	schreibt Sektoren auf eine Festplatte
0C	Zylinder suchen	AH=0Chex, AL, CH, CL, DH, DL und ES:BX= wie Funktion 0Ahex	bei Fehler Carry-Flag gesetzt und AH=Fehlercode (Tabelle 18)	Position eines bestimmten Zylinders suchen
0D	Festplatten-Reset	AH=0Dhex, DL= wie Funktion 0Ahex	bei Fehler Carry-Flag gesetzt und AH=Fehlercode (Tabelle 18)	Zurücksetzen der Festplatte und des Controllers
10	Drive ready	AH=10hex, DL=wie Funktion 0Ahex	Bei Fehler Carry-Flag gesetzt und AH=Fehlercode (Tabelle 18)	kontrolliert, ob die Festplatte für Operationen verfügbar ist
11	Laufwerk recalibrieren	AH=11hex, DL=wie Funktion 0Ahex	bei Fehler Carry-Flag gesetzt und AH=Fehlercode (Tabelle 18)	veranlaßt den Controller, den Zylinder 0 anzusteuern (recalibrieren)
12	Stop SCSI	AH=12hex, DL=Laufwerks-ID	Bei Fehler Carry-Flag gesetzt und AH=Fehlercode (Tabelle 18)	parkt eine SCSI-Festplatte
14	Controller-Diagnose	AH=14hex	bei Fehler Carry-Flag gesetzt und AH=Fehlercode (Tabelle 18)	ruft das im Controller eingebaute Diagnose-Programm auf
15	Diskettentyp	AH=15hex, DL=Laufwerk	bei Fehler siehe Funktion 08hex, sonst AH=Typencode (0: keine, 1: Floppy ohne Wechsel, 2: Floppy mit Wechsel, 3: Festplatte dabei CX:DX Sektoranzahl)	ermittelt den Laufwerkstyp
16	Diskettenwechsel prüfen	AH=16hex, DL=Laufwerk	wenn Carry-Flag gesetzt, dann AH=06hex für Wechsel ermittelt oder Funktion nicht verfügbar, oder AL=80hex für Laufwerk nicht bereit; sonst AH=0 für kein Diskettenwechsel	ermittelt, ob die Diskette im Laufwerk gewechselt wurde
17	Diskettentyp setzen	AH=17hex, AL=Typ (Tabelle 16), DL=Laufwerk	bei Fehler Carry-Flag gesetzt und AL=Fehlercode	Setzt den Diskettentyp für eine Formatierung
18	Mediatyp setzen	AH=18hex, DL=Laufwerk, CH=niederwertige 8 Bit der Spüranzahl, CL=Sektor pro Spur (Bit 0 bis 5) und oberen 2 Bit der Spüranzahl (Bit 6 und 7)	AH=Status (0:Typ verfügbar, 1:Funktion nicht verfügbar, Chex:Typ nicht verfügbar, 80hex:keine Diskette im Laufwerk), ES:DI 11 Byte Parametertabelle)	Mediatyp für Formatierung setzen
27	Installation prüfen	AH=27hex, BX=0000hex	BX ungleich 0000hex, wenn installiert	prüft, ob QCache installiert wurde
F9	Installation prüfen	AH=F9hex, DL=wie Funktion 0Ahex	bei Fehler nur Carry-Flag gesetzt, sonst DX=Bit 15 gesetzt, wenn installiert	prüft ob SWBIOS – ein TSR-Programm von ONTRACK Systems – installiert ist

Tabelle 14. Der Aufbau des Interrupt 13hex

Offset (in hex)	Adresse	Größe	Beschreibung
0		Byte	Anzahl der Rasterpalte
1		Byte	maximale Rasterzeilen (Anzahl der Zeilen bis 1)
2		Byte	Scanlinie pro Rasterzeile
3		Word	Videopuffergröße in Byte
5		4 Byte	Werte für die Sequenzerregister 1 bis 4
9		Byte	Werte für verschiedene Ausgaberegister
0A		25 Byte	Werte für die CRTC-Register 00hex bis 11hex (Tabelle 29)
23		20 Byte	Standardpalette (Werte für Attributcontroller-Register 00hex bis 13hex)
37		9 Byte	Werte für die Grafikcontroller-Register 00hex bis 08hex

Tabelle 15. Die Daten der Grafikmodustabelle

Interrupt	Vom Prozessor	Bedeutung	ab Tabelle
0	•	Division durch 0	–
1	•	Einzelschrittmodus	–
2	•	Non Maskable Interrupt (NMI)	–
3	•	Breakpoint (Debugger)	–
4	•	INTO erfaßt Überlauf	–
5	•	BOUND-Bereich überschritten	–
6	•	Opcode (entspricht Assembler-Befehl) nicht definiert	–
7	•	Coprozessor nicht vorhanden	–
8		(IRQ0) System-Timer	–
9		(IRQ1) Daten von Tastatur im Puffer	–
0A		(IRQ2) Interrupt durch EGA-Karten	–
0B		(IRQ3) Interrupt durch serielle Schnittstelle Nr. 2	–
0C		(IRQ4) Interrupt durch serielle Schnittstelle Nr. 1	–
0D		(IRQ5) Interrupt durch Festplatten-Controller	–
0E		(IRQ6) Interrupt durch Disketten-Controller	–
0F		(IRQ7) Interrupt durch den Drucker	–
10		Video-Funktionsbibliothek	ab Tabelle 7
11		Systemkonfiguration	ab Tabelle 22
12		Arbeitsspeicher bis 640 KByte ermitteln	ab Tabelle 22
13		Disketten- und Festplatten-Funktionsbibliothek	ab Tabelle 14
14		Funktionsbibliothek für die serielle Schnittstelle	ab Tabelle 24
15		Funktionen für Kassettenunterstützung und Protected Mode	siehe Beitrag »Aus alt mach neu« in dieser Ausgabe der DOS Extra
16		Funktionsbibliothek für die Tastatur	ab Tabelle 25
17		Funktionsbibliothek für den Drucker	ab Tabelle 34
18		ROM-Basic starten	ab Tabelle 20
19		Reboot-Interrupt	ab Tabelle 20
1A		Zeituhr ändern	ab Tabelle 21
1B		Control-Break-Handler	ab Tabelle 20
1C		System-Tick-Handler	ab Tabelle 20
1D		Vektor zur Videoparameter-Tabelle	ab Tabelle 20
1E		Vektor zur Diskettenparameter-Tabelle	ab Tabelle 20
1F		Vektor zum Grafikzeichensfont	ab Tabelle 20

Tabelle 17. Alle BIOS-Interrupts im Überblick

Registerwert (in hex)	Formattyp
1	320/360-KByte-Diskette in 360-KByte-Laufwerk
2	320/360-KByte-Diskette in 1,2-MByte-Laufwerk
3	1,2-MByte-Diskette in 1,2-MByte-Laufwerk
4	720-KByte-Diskette in 720-KByte-Laufwerk

Tabelle 16. Die Registerwerte für die verschiedenen Diskettenformate

Wert in AH (in hex)	Bedeutung
0	kein Fehler
1	Funktion nicht definiert
2	keine Adreßmarkierung
3	schreibgeschützte Diskette
4	Sektor nicht gefunden
5	Reset war erfolglos
6	Diskette wurde gewechselt
7	fehlerhafte Initialisierung
8	DMA-Überlauf
9	DMA-Überlauf bei Segmentgrenze
0A	unbrauchbarer Sektor
0B	unbrauchbare Spur
0C	nicht unterstützte Spur oder unerlaubtes Speichermedium
0D	unerlaubte Sektorzahl im Format
0E	Kontroll-Datenadreßmarkierung gefunden
0F	DMA-Level ungültig
10	Lesefehler
11	falsche Prüfsumme, aber Daten gelesen
20	Controller-Fehler
40	Spur nicht gefunden
80	Laufwerk reagiert nicht
AA	Laufwerk nicht bereit
BB	BIOS-Fehler
CC	Schreibfehler
E0	Statusregisterfehler
FF	unbekannter Fehler

Tabelle 18. Die Fehlermeldungen von Interrupt 13hex

Funktion (in hex)	Kurzbezeichnung	Aufruf	Ergebnis	Beschreibung
0	Funktionen deaktivieren	AH=12hex, BH=55hex, BL=00hex	–	deaktiviert die erweiterten Unterfunktionen
1	Funktionen aktivieren	AH=12hex, BH=55hex, BL=01hex	–	aktiviert die erweiterten Unterfunktionen
2	Status abfragen	AH=12hex, BH=55hex, BL=02hex	AL=Status-Flag (Tabelle 28)	fragt den Bildschirmstatus ab
3	Register-Trapping deaktivieren	AH=12hex, BH=55hex, BL=03hex	–	deaktiviert das Register-Trapping (CGA-Emulation)
4	Register-Trapping aktivieren	AH=12hex, BH=55hex, BL=04hex	–	aktiviert das Register-Trapping (CGA-Emulation)
5	Grafikmodus setzen	AH=12hex, BH=55hex, BL=05hex, ES:BP=Adresse der Modus-Tabelle (Tabelle 15)	–	programmiert die Tabelle des Grafikmodus
6	Grafikmodustabelle lesen	AH=12hex, BH=55hex, BL=03hex, AL=Grafikmodus	Bei Fehler: bp=FFFFhex, sonst ES:BP=Adresse der Grafikmodus-Tabelle (Tabelle 15)	gibt die Adresse der Tabelle für den gewünschten Grafikmodus an

Tabelle 19. Erweiterte Bibliothek für Funktion 1255hex aus Interrupt 10hex

Interrupt (in hex)	Bedeutung	Aufruf	Ergebnis	Beschreibung
18	ROM-Basic starten	-	-	starten bei den IBM-PC den Basic-Interpreter aus dem Arbeitsspeicher; gilt nur für IBM-Original-PCs
19	Für Warmstart: Wert 1234hex an Adresse 0040hex:0072hex ablegen und zu FFFFhex:0000hex springen; für Kaltstart: Wert 0000hex an Adresse 0040hex:0072hex ablegen und zu FFFFhex:0000hex springen	-	-	startet den Rechner, ohne den Speicher oder die Interrupts zu löschen; um Kaltstarts zu produzieren siehe Aufruf
1B	Control-Break-Handler	-	-	wird aufgerufen, wenn Interrupt 09hex ein Control-Break (via Tastatur) bemerkt
1C	System-Tick-Handler	-	-	wird durch Interrupt 08hex aufgerufen (wenn ein System-Tick erfolgt, also 18,2mal in der Sekunde)
1D	Adreßvektor	-	-	kein Interrupt, sondern beinhaltet die Adresse der aktiven Videoparametertabelle
1E	Adreßvektor	-	-	kein Interrupt, sondern beinhaltet die Adresse der aktiven Diskettenparameter-Tabelle
1F	Adreßvektor	-	-	kein Interrupt, sondern beinhaltet die Adresse einer 1 KByte großen Bitmapdaten-Tabelle mit 8 Byte für jedes ASCII-Zeichen (ASCII 128 bis 255) ALs 8x8-Zeichensatz nur für den Grafikmodus.

Tabelle 20. Die Interrupts 18hex, 19hex, und 1Bhex bis 1Fhex

Funktion (in hex)	Bedeutung	Aufruf	Ergebnis	Beschreibung
0	Systemzeit ermitteln	AH=00hex	AL=ungleich 0, wenn Mitternacht seit dem letzten Lesen vergangen ist, CX:DX=Anzahl der seit Mitternacht vergangenen Urticker (18,2 Ticker = 1 Sekunde)	liest die Uhrzeit aus der BIOS-Uhr
1	Systemzeit setzen	AH=01hex, CX:DX=Anzahl der seit Mitternacht vergangenen Urticker (18,2 Ticker = 1 Sekunde)	setzt die Uhrzeit der BIOS-Uhr	
2	Uhrzeit ermitteln	AH=02hex	bei Fehler Carry-Flag gesetzt (Uhr läuft nicht, oder Uhrzeit wird gerade aktualisiert; sonst CH=Stunde, CL=Minute, DH=Sekunden, DL=00hex für Standardzeit)	ermittelt die Uhrzeit in herkömmlichen Angaben gesplittet; die Uhrzeit wird ALs BCD-Zahlen abgelegt
3	Uhrzeit setzen	AH=03hex, CH=Stunde, CL=Minute, DH=Sekunden, DL=00hex für Standardzeit	bei Fehler Carry-Flag gesetzt	setzt die Uhrzeit in herkömmlichen Angaben gesplittet; die Zahlen müssen ALs BCD-Zahlen abgelegt werden
4	Datum ermitteln	AH=04hex	bei Fehler Carry-Flag gesetzt; sonst CX=Jahr, DH=Monat, DL=Tag	ermittelt das aktuelle Datum des Rechners; die Zahlen werden als BCD-Zahlen abgelegt
5	Datum setzen	AH=05hex, CX=Jahr, DH=Monat, DL=Tag	-	setzt das aktuelle Datum des Rechners; die Zahlen werden als BCD-Zahlen abgelegt
6	Alarm setzen	AH=06hex, CH=Stunde, CL=Minute, DH=Sekunde	Bei Fehler Carry-Flag gesetzt	setzt die Alarmzeit im CMOS des Rechners; die Zahlen werden als BCD-Zahlen abgelegt
7	Alarm löschen	AH=07hex	-	stellt die im CMOS befindliche Alarmzeit aus
8	RTC-Aktiv-Mode setzen	AH=08hex, CH=Stunde, CL=Minute, DH=Sekunde	-	setzt die Zeit, bei der der Strom eines Rechners aktiviert wird; nur bei speziellen Rechnern verfügbar; Zahlen werden ALs BCD-Zahlen abgelegt
9	RTC-Alarmstatus ermitteln	AH=09hex	CH=Stunde, CL=Minute, DH=Sekunde, DL=Status	liest die Alarmzeit und den Alarmstatus aus; verfügbar unter anderem für IBM-PS/2-Modell 30; Zahlen werden als BCD-Zahlen abgelegt
0A	Tageszähler auslesen	AH=0Ahex	bei Fehler Carry-Flag gesetzt (Funktion nicht verfügbar); sonst CX=Anzahl der Tage seit dem 1.1.1980	ermittelt die Anzahl der Tage, die seit dem 1.1.1980 vergangen sind; nur auf bestimmten Rechnern, wie den IBM-PS/2-Modellen, verfügbar
0B	Tageszähler setzen	AH=0Bhex, CX=Anzahl der Tage seit dem 1.1.1980	bei Fehler Carry-Flag gesetzt (Funktion nicht verfügbar); sonst	setzt die Anzahl der Tage, die seit dem 1.1.1980 vergangen sind; nur auf bestimmten Rechnern, wie den IBM-PS/2 Modellen, verfügbar

Tabelle 21. Die Funktionen des Interrupts 1Ahex

Interrupt (in hex)	Kurzbezeichnung	Eingabe	Ausgabe	Beschreibung
11	Systemkonfiguration ermitteln	–	AX=Konfiguration bitweise interpretiert	ermittelt die Systemkonfiguration des PCs; der ermittelte Videomodus gibt den Modus nach dem Start des PCs an und nicht den aktiven;
12	Arbeitsspeicher ermitteln	–	AX=Arbeitsspeicher (in KByte)	ermittelt die Größe des Arbeitsspeichers; kann leider nur bis maximal 640 KByte ermitteln

Tabelle 22. Übersicht über die Interrupts 11hex und 12hex

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
AH	Timeout	Transmit-Shift-Register Empty	Transmit-Holding-Register Empty	Break Detected	Framing Error	Parity Error	Overrun Error	Receive Data Ready
AL	Carrier Detect (CD)	Ring Indicator (RI)	Data Set Ready (DSR)	Clear to Send (CTS)	Delta Carrier Detect	Trailing Edge of Ring	Delta Data Set Ready	Delta Clear to Send

Tabelle 23. Aufbau der Statusbits für Modems

Funktion (in hex)	Kurzbeschreibung	Aufruf	Ergebnis	Beschreibung
0	Port initialisieren	AH=00hex, AL=Portparameter (Tabelle 32), DX=Portnummer (0 bis 3)	AH=Status (Tabelle 23), AL=Modemstatus (Tabelle 23)	Einrichtung der seriellen Schnittstelle mit Operationsparameter
1	Zeichen aus Port lesen	AH=01hex, AL=Zeichen, DX=Portnummer (0 bis 3)	bei Fehler Bit 7 in AH gesetzt; sonst AL=Status (Tabelle 23)	gibt ein Zeichen auf ein definiertes Port aus
2	Zeichen zum Port schreiben	AH=02hex, DX=Portnummer (0 bis 3)	AH=Status (Tabelle 23), AL=Zeichen (wenn Bit 7 in AH ungesetzt)	liest ein Zeichen auf einen definierten Port aus
3	Portstatus	AH=03hex, DX=Portnummer (0 bis 3)	AH=Status (Tabelle 23), AL=Modemstatus (Tabelle 23)	liest den Status eines seriellen Ports aus
A4	Block schreiben	AH=A4hex, CX=Länge, DH=00hex, ES:BX=Puffer	CX=Anzahl gesendeter Byte	sendet einen Datenblock aus dem Puffer zur seriellen Schnittstelle
A5	Block lesen	AH=A5hex, CX=Länge, DH=00hex, ES:BX=Puffer	CX=Anzahl gelesener Byte	liest einen Datenblock aus der seriellen Schnittstelle in den Puffer
A6	Kurz-Break senden	AH=A6hex, DH=00hex	–	schickt ein Kurz-Break zum seriellen Port

Tabelle 24. Die Funktionsbibliothek des Interrupts 14hex

Funktion (in hex)	Kurzbeschreibung	Aufruf	Ergebnis	Beschreibung
0	Tastaturabfrage	AH=00hex	AH=Scancode, AL=ASCII-Zeichen	liest die Tastatureingabe ein oder wartet auf eine Tastatureingabe (nicht für erweiterte Tastaturen, siehe Funktion 10hex)
1	Tastaturabfrage	AH=01hex	Null-Flag bei Tastendruck gesetzt, AH= Scancode, AL=ASCII-Zeichen	kontrolliert, ob eine Taste gedrückt wurde (nicht für erweiterte Tastaturen, siehe Funktion 11hex)
2	Shift-Flag lesen	AH=02hex	AL=Shift-Flags	ermittelt den Status der Shift-Flags
3	Wiederholungsrate setzen	AH=03hex, AL=05hex (0 bis 4 für IBM-PS-junior), BH=Zeitraum (0: 250 ms, 1: 500 ms, 2: 750 ms, 3: 1000 ms), BL=Wiederholungsrate (0: 30 Zeichen/Sek bis 13: 10 Zeichen/sec)	–	setzt die Wiederholungsrate der Tastatur
4	Tastaturklick setzen	AH=04hex, AL=Status (0: ein, 1: aus)	–	schaltet das Klicken bei Tastendruck ein und aus
5	Tastendruck speichern	AH=05hex, CH=Scancode, CL=ASCII-Zeichen	AL= Status (0: gespeichert, 1: Puffer voll)	speichert einen Tastaturdruck in den Tastaturpuffer (nur für erweiterte Tastaturen)
10	Tastaturabfrage	AH=10hex	AH=Scancode, AL=ASCII-Zeichen	liest die Tastatureingabe ein bzw. wartet auf eine Tastatureingabe (nur für erweiterte Tastaturen, siehe Funktion 00hex)
11	Tastaturabfrage	AH=11hex	Null-Flag bei Tastendruck gesetzt, AH= Scancode, AL=ASCII-Zeichen	kontrolliert, ob eine Taste gedrückt wurde (nur für erweiterte Tastaturen, siehe Funktion 01hex)
12	erweitertes Shift-Flag lesen	AH=12hex	AH=Shift-Flags Nr. 1, AL=Shift-Flags Nr. 2 (Tabelle 4)	ermittelt den Status der Shift-Flags für erweiterte Tastaturen

Tabelle 25. Die Struktur des Interrupts 16hex

DAS ABO

Das ABO:
Ihr Vorsprung



- Sichern Sie sich Ihre Vorteile:
- Hochkarätige Informationen
- Regelmäßig frei Haus lieferbar
- Über 7 Prozent Preisvorteil

Widerrufsrecht: Jeder Abonnent hat das Recht, seine Bestellung innerhalb einer Woche beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich zu widerrufen. Die rechtzeitige Absendung des Widerrufsschreibens genügt zur Fristwahrung.

DMV Verlag • Postfach 250 • 3440 Eschwege



Modus- numm- er	Arbeitsweise
0	Zeichenattribut in BL; Cursorposition nicht aktualisieren
1	Zeichenattribut in BL; Cursorposition aktualisieren
2	Zeichenattribut für jedes Zeichen in Zeichenkette enthalten (muß nach jedem Zeichen folgen); Cursorposition nicht aktualisieren
3	Zeichenattribut für jedes Zeichen in Zeichenkette enthalten (muß nach jedem Zeichen folgen); Cursorposition aktualisieren

Tabelle 26. Informationen für die Ausgabe einer Zeichenkette

Offset- Adresse (in hex)	Größe	Beschreibung
0	Word	Monitormodelnummer
2	Word	vertikale Pixel pro Meter
4	Word	horizontale Pixel pro Meter
6	Word	vertikale Gesamtzahl der Pixel
8	Word	horizontale Gesamtzahl der Pixel
0A	Word	horizontaler Pixelabstand in Mikrometer
0C	Word	vertikaler Pixelabstand in Mikrometer

Tabelle 27. Diese Tabelle beinhaltet diverse Informationen über den Monitor

Bit	Wert	Bedeutung
3	1	erweiterte Funktionen verfügbar
5 bis 7	000	PS/2 mono
	001	PS/2 color
	010	multisync
	011	Taxan-650, 25 kHz
	100	RGB
	101	mono
	110	EGA
	111	Compaq-interne Funktionen

Tabelle 28. Die Status-Flags für Funktion 1255hex aus Interrupt 10hex beinhalten Informationen über die Funktions- und den Grafikstatus

Position	Registername
0	horizontale Gesamtgröße der Zeichen
1	horizontale Anzeige (Zeichen)
2	horizontale Synchronisationsposition (Zeichen)
3	horizontale Synchronisationsweite (Zeichen)
4	vertikale Gesamtgröße (Zeichen-Rasterzeilen)
5	vertikale Gesamtjustierung (Scanlinien)
6	vertikale Anzeige (Zeichen-Rasterzeilen)
7	vertikale Synchronisationsposition (Zeichen-Rasterzeilen)
8	Interlaced-Modus
9	maximaler Scanwert pro Rasterzeile
0A	Scanlinie für den Cursorstart
0B	Scanlinie für das Cursorende
0C	Bildschirmspeicher-Start (high)
0D	Bildschirmspeicher-Ende (low)
0E	Cursoradresse (high)
0F	Cursoradresse (low)
10	Lichtstift (high)
11	Lichtstift (low)

Tabelle 29. Die CRTC-Register 00hex bis 11hex

Farbwert	Intensitätsbit	Rotbit	Grünbit	Blaubit	Farbe
0	0	0	0	0	Schwarz
1	0	0	0	1	Blau
2	0	0	1	0	Grün
3	0	0	1	1	Kobaltgrün
4	0	1	0	0	Rot
5	0	1	0	1	Violett
6	0	1	1	0	Braun
7	0	1	1	1	Hellgrau
8	1	0	0	0	Dunkelgrau
9	1	0	0	1	Hellblau
10	1	0	1	0	Hellgrün
11	1	0	1	1	Hellkobaltgrün
12	1	1	0	0	Hellrot
13	1	1	0	1	Hellviolett
14	1	1	1	0	Gelb
15	1	1	1	1	Weiß

Tabelle 30. Die Farbwerte und die RGB-Kombinationen

Farbnummer	Palette 0	Palette 1	Palette 2	Palette 3
0	aktive Hintergrundfarbe	aktive Hintergrundfarbe	aktive Hintergrundfarbe	aktive Hintergrundfarbe
1	Grün	Türkis	Hellgrün	Helltürkis
2	Rot	Magenta	Hellrot	Lila
3	Braun	Hellgrau	Gelb	Weiß

Tabelle 31. Die dargestellte Farbe bei den verschiedenen Farbpalnette

Bit (7, 6 und 5)	Baudrate	Bit (4, 3)Paritybits	Bit 2	Stop-Länge	Bit 1, 0 Word
0	110	00	kein	0	1 0 5
1	150	01	ungerade	1	2 1 6
10	300	10	kein		10 7
11	600	11	gerade		
100	1200				
101	2400				
110	4800				
111	9600				

Tabelle 32. Die Parameter der Schnittstellen

Wert in AL	Bedeutung (in hex)
0	externe Karte deaktivieren
1	interne Karte aktivieren
2	aktive Karte ausschalten
3	inaktive Karte einschalten
80	Systemboard Video-Aktiv-Flag setzen

Tabelle 33. Informationen über das Umschalten von Grafikkarten

Funktion (in hex)	Kurzbeschreibung	Aufruf	Ergebnis	Beschreibung
0	Zeichen schreiben	AH=00hex, AL=Zeichen, DX=Druckernummer (0 bis 2)	AH=Druckerstatus	Gibt ein Zeichen auf den Drucker aus
1	Drucker und Port initialisieren	AH=01hex, DX=Drucker (0 bis 2)	AH=Druckerstatus	setzt den Druckerport zurück und initialisiert den Drucker
2	Status ermitteln	AH=02hex, DX=Druckernummer (0 bis 2)	AH=Druckerstatus	ermittelt den Status eines der angeschlossenen Drucker

Tabelle 34. Der Interrupt 17hex

Werkstatt: A86-Assembler

Licht im Speicher

Mit dem Shareware-Assembler A86 steigen Sie zum Spartarif in die Programmierung ein. Gut dokumentierte, kleine Listings ergeben die ersten lauffähigen Programme, die Sie schon als Hilfe für Ihr System verwerten: So schalten Sie die Feststelltasten ein oder die Routine zur Hardcopy aus und geben Zeichenketten am Bildschirm aus.

Beim Programmieren in Assembler ist es nicht nur damit getan, die Befehle zu kennen. Da Assembler-Programme von der verwendeten Hardware abhängig sind, sollten Sie neben der Bedeutung der einzelnen Befehle auch einige Interna des verwendeten Rechners kennen. Dann werden Sie im Lauf der Zeit geradezu ein Gespür dafür entwickeln, wie Sie Probleme in Assembler lösen. Das Gespür erarbeiten Sie sich, indem Sie Programme nachvollziehen und ausprobieren.

Diese Werkstatt soll daher all jenen Lesern dienen, die sich bereits theoretisches Wissen über Assembler-Sprachen angeeignet haben. Dieses Wissen können Sie auch in anderen Rechnerfamilien erworben haben, etwa dem Commodore 64, Schneider CPC oder Apple II. Wer jedoch noch gar keine Assembler-Sprachkenntnisse besitzt, kommt wahrscheinlich nicht um Fachliteratur herum. Die Thematik ist zu komplex, um sie in einer einzigen Ausgabe abzuhandeln.

Doch das sollte Sie nicht vom Thema Assembler abschrecken. An praxisnahen Beispielen läßt es sich besonders

gut lernen. Deshalb zeigt Ihnen diese Werkstatt verschiedene Beispiele anhand kleiner Hilfsprogramme auf. So erhalten Sie – gleichsam als Nebeneffekt – kleine Hilfen, die Ihnen die tägliche Arbeit erleichtern.

Da kein Computer etwas mit der Assembler-Sprache anfangen kann, brauchen Sie einen Übersetzer. Dieser wird, wie die Programmiersprache selbst, als Assembler bezeichnet. Der Werkstatt liegt der A86-Assembler zugrunde, den Sie über den Sharewareweg preisgünstig erhalten. Testen Sie ihn im Rahmen dieses Beitrags in Ruhe und lassen Sie sich, wenn er Ihnen zusagt, registrieren. Ein Manko hat er jedoch: Da es sich um ein amerikanisches Produkt handelt, wird er mit einer englischsprachigen Anleitung auf Diskette geliefert. Eine A86-Version mit deutschsprachigen Texten erhalten Sie bei Computer Solutions. Im Systema-Verlag erschienen ist außerdem das Buch »A86/D86 – Assembler-Programmierung für Einsteiger«, das die im Titel angesprochenen Aspekte umfassend abhandelt. Um diese Werkstatt nachzuvoll-

ziehen, benötigen Sie nur die englische Version, die Sie bei den meisten Sharewareversendern erhalten.

• Der Interrupt enthüllt sein Geheimnis

Das englische Wort »interrupt« bedeutet soviel wie »unterbrechen«. Ein Interrupt veranlaßt den Prozessor aufgrund einer bestimmten Unterbrechung, eine festgelegte Routine abzuarbeiten. Sobald diese Unterbrechungsroutine abgearbeitet ist, fährt der Prozessor dort fort, wo er den normalen Ablauf unterbrechen mußte. Es gibt zwei Arten von Interrupt-Aufrufen. Ein Hardware-Interrupt wird beispielsweise nach einem Tastendruck ausgeführt. Ein Software-Interrupt kann nur durch einen Befehl aufgerufen werden. Hierzu dient der Befehl »INT n«, bei dem »n« die Nummer des Interrupts angibt. Die Interrupts sind fortlaufend nummeriert. Anhand eines Wegweisers, der als Interrupt-Vektor bezeichnet wird, wird eine bestimmte Routine aufgerufen. Also rufen Sie durch den Befehl »INT n« das Unterprogramm »n« auf. Die Adresse des »n« steht im dazugehörigen Interrupt-Vektor. Mit der Änderung der Vektoren variieren Sie also die Adresse eines bestimmten Unterprogramms, selbst eines Programms, das Ihnen das Betriebssystem bereitstellt. Dies ist sinnvoll, da sich in den einzelnen Betriebssystemversionen außer dem Umfang der Unterprogramme auch deren Adressen geändert haben. Sie rufen also Unterprogramme des Systems unabhängig von der Betriebssystemversion auf, da der Wegweiser »Interrupt-Vektor« die richtige Beziehung herstellt. Durch dieses Merkmal sind Rechner des Industriestandards sehr flexibel. Das betrifft auch die Hardware. So sind beispielsweise Mäuse, die Sie an der hierfür eigentlich nicht vorgesehenen seriellen Schnittstelle

```
C:\EXT\21\A86\PROGRAMM>xecho      A-86 Assembler $ _ Werkstatt von Ulrich Borgdorf
$ _DOS EXTRA 21$et[22C/et
A-86 Assembler
Werkstatt von Ulrich Borgdorf
DOS EXTRA 21      /et
C:\EXT\21\A86\PROGRAMM>
```

```
C:\EXT\21\A86\PROGRAMM>led /?
LED [Nn] [Cn] [Sn]

N=Numlock
C=Capslock
S=Scrolllock

n=1 ==> aus
n=0 ==> an

Bsp.:
LED N0 schaltet Num aus
LED N1 C1 S1 schaltet alle
Funktionen ein
C:\EXT\21\A86\PROGRAMM>
```

Bild 1. Xecho arbeitet mit Zeilenumbrüchen und hat mit einer Escape-Sequenz das Kürzel »/et« 22 Spalten nach rechts gerückt.

Bild 2. Led gibt die Syntax beim Aufruf mit dem Parameter »/?« aus – wie ein Befehl von MS-DOS.

anschließen dürfen, Nutznießer des flexiblen Interrupt-Konzepts. Unterprogramme des Betriebssystems rufen Sie nur im Ausnahmefall direkt auf.

• Zeichenausgabe in Assembler

Das Betriebssystem MS-DOS stellt Interrupt-Funktionen zur Zeichenausgabe auf dem Bildschirm und dem Drucker zur Verfügung. Die Zeichen mit einer Codierung von kleiner als 32dez stellen solche Steuercodes dar. So bewirkt der Steuercode 07dez auf dem Bildschirm die Ausgabe eines Pieptons. Der Steuercode 12dez hingegen löst bei den meisten Druckern einen Seitenvorschub aus.

Als erstes schreiben Sie ein kleines Programm, das einen Pieps ausgibt. Sie verschaffen sich das ASCII-Zeichen mit dem Code 7 am Bildschirm anders, als dies die »Tips & Tricks für Einsteiger« der DOS International empfehlen:

```
copy con pieps.bat
echo [Alt-007][F6]
```

Denn diesmal verwenden Sie die Funktion 02hex des Interrupts 21hex. Diese zeigt das Zeichen des im DL-Register enthaltenen ASCII-Codes auf dem Bildschirm an. Geben Sie dazu das Programm »beep.asm« (Listing 1) ein.

Da der A86-Assembler über keinen eigenen Editor verfügt, geben Sie »beep.asm« mit einem fremden Programm ein. Dazu setzen Sie beispielsweise die Editoren Edit von MS-DOS 5.0 oder Editor von DR-DOS ein. Sie dürfen natürlich auch eine beliebige Textverarbeitung verwenden. In diesem Fall sollten Sie jedoch im ASCII-Format speichern.

Die gespeicherte Textdatei können Sie jedoch nicht direkt starten, da Sie sie noch mit dem A86-Assembler in Maschinensprache übersetzen müssen. Geben Sie dazu auf Betriebssystemebene

```
a86 beep.asm
```

ein. Damit legen Sie auf Ihrem Datenträger die Datei »beep.com« an. Wenn »beep.asm« den Formalien nicht entspricht, gibt der A86-Assembler entsprechende Fehlermeldungen aus. Meist finden Sie die Fehlermeldungen in dem zu kompilierenden Programm wieder. In diesem Fall generiert der Assembler eine unveränderte Kopie Ihres Programms. Über diese Vorgänge informiert Sie jedoch der

Ziffer	Stelle	ASCII-Code	Rechnung	Ergebnis
2	100	50dez	$(50\text{dez} - 48\text{dez}) \times 10 \times 10$	=200
3	10	51dez	$(51\text{dez} - 48\text{dez}) \times 10$	=30
4	1	52dez	$(52\text{dez} - 48\text{dez}) \times 1$	= 4

Tabelle 1. Die Übergabe des Werts 234 an die Variable »Errorlevel« gibt das ASCII-Zeichen »Omega« am Bildschirm aus.

A86-Assembler während der Assemblierung.

Nun folgt die Zeichenausgabe auf dem Drucker. Dazu dient die Funktion 05hex des Interrupts 21hex, die das im Register DL enthaltene ASCII-Zeichen an den Drucker ausgibt. Bei den (meisten) Druckern erzeugt das Zeichen mit dem Code 12dez einen Seitenvorschub. Wenn Sie nun »ff.asm« (Listing 2) eingeben, erhalten Sie ein kleines Programm, das einen Seitenvorschub auf Ihrem Drucker auslöst.

Diese ersten beiden Programme geben Steuerzeichen aus. Hier können Sie natürlich mit anderen Zeichencodes, wie etwa dem Code 65dez »A«, experimentieren. Wenn Sie einen hexadezimalen Wert definieren wollen, muß diesem der Buchstabe »h« folgen. Für dezimale Werte ist keine weitere Kennung notwendig.

• Zeichenkettenausgabe und Textendekennung

In der Praxis ist es oft erwünscht, eine Anzahl von unterschiedlichen Zeichen auf dem Bildschirm darzustellen. Das Betriebssystem MS-DOS stellt eine Interrupt-Funktion zur Zeichenkettenausgabe zur Verfügung. Hier-von macht das Programm »text.asm« (Listing 3) Gebrauch. Es nutzt die Funktion 09hex des Interrupts 21hex. Den Text sollten Sie zuvor an einer bestimmten Adresse (Zeilen 7 und 8) speichern, die vor dem Interrupt-Aufruf im Register DX stehen muß. Die Adresse weisen Sie mit der Offset-funktion zu, die in Kombination mit dem Befehl »mov« die Offsetadresse der Textmarke zurückliefert. Nun definieren Sie nur noch mit dem Dollarzeichen »\$« das Textende. Die Anweisung »Doku DB« füllt den Speicher an dieser Stelle im Programm mit dem angegebenen Text.

• Parameter aus der Kommandozeile lesen

Wenn Sie ein externes Betriebssystemprogramm wie etwa »Format« aufrufen, ist es notwendig, die Para-

meter aus der Kommandozeile innerhalb dieses Programms einzulesen. Diese Informationen enthält das »Program Segment Prefix«, das 256 Byte lang ist und unter dem Kürzel »PSP« firmiert. Sie finden es vor jedem Com-Programm. Die von Ihnen mit dem A86 generierten Com-Programme fangen stets bei der Adresse 100hex an, weil in den ersten 256 Bytes (256dez = 100hex) das PSP zu finden ist. Die Kommandozeile enthält alle Eingaben, die dem Dateinamen des aufgerufenen Programms folgen. Bei der Eingabe

```
format a: b:
```

enthält die Kommandozeile also die Zeichen »a: b:«. Die Länge der Kommandozeile ist im Byte 80hex zu finden. Die Zeichen der Kommandozeile folgen in den Bytes 81hex bis FFhex. Die Kommandozeile endet jeweils mit einem Wagenrücklauf »0Dhex« oder »13dez« und enthält immer mindestens ein führendes Leerzeichen. Nun muß das Betriebssystem die Kommandozeile des Beispiels »a: b:« in zwei Blöcke unterteilen und dabei die Buchstaben in Großbuchstaben umwandeln. Dies geschieht nicht in den einzelnen Programmen, wie etwa bei den zum Betriebssystem gehörenden Programmen »Diskcopy« und »Format«, weil dies das System selbständig erledigt. So können auch »selbst-gestrickte« Programme auf diese Umformung zurückgreifen. Dazu werden zwei Dateisteuerblöcke generiert, die es gestatten, maximal zwei Parameter einzulesen. Die Dateisteuerblöcke tragen das Kürzel »FCB« (File Control Block). Wenn zwischen den Zeichen der Kommandozeile ein Leerzeichen steht, wird ein zweiter Dateisteuerblock generiert. Geben Sie überflüssige Leerzeichen ein, filtert das System diese aus. Der erste Dateisteuerblock befindet sich im Bereich 5Chex bis 6Bhex. Der zweite folgt von 6Chex bis 7Fhex.

»xecho.asm« (Listing 4) liest nun die Zeichen der Kommandozeile aus und

Dokumentation	Basic	Assembler
Schleifenkopf	FOR C=10 TO 1 STEP -1	MOV CX,10
Schleifenblock	[Befehlsblock]	Marke:
Schleifenende	NEXT C	[Befehlsblock]
		LOOP Marke

Tabelle 2. Schleifen in Assembler zu programmieren ist auch nicht viel schwerer als in Basic, wie der Vergleich der Befehle beweist.

Operator	Erklärung
AND	und
OR	oder
XOR	exklusives Oder
NOT	nicht

Tabelle 3. Die logischen Operatoren der Algebra begleiten Sie bei Ihrer Arbeit am Rechner.

stellt diese auf dem Bildschirm dar. Ein wesentlicher Unterschied zwischen dem Betriebssystembefehl »echo« und dem Programm Xecho ist, daß Xecho keinen automatischen Zeilenvorschub ausführt, wenn Sie dies nicht ausdrücklich wünschen. Wenn die Kommandozeile die Zeichenkombination »\$__« enthält, wird diese Zeichenkombination gegen einen Zeilenvorschub ausgetauscht. Selbst die Ausgabe des Escape-Codes 27dez gestattet Ihnen Xecho mit der Zeichenkombination »\$E« (Bild 1). Mit Xecho können Sie also auch ANSI-Befehle ausführen, da diese den Escape-Steuercode nutzen. Das Quellenregister SI ist in Xecho bewußt nicht auf 81hex gesetzt, sondern auf 82hex. So ignorieren Sie das erste Leerzeichen, das den Programmnamen vom Parameterblock trennt.

● Hardcopy abschalten

Bekanntlich bewirkt die Tastenkombination [Umschalt-Druck] das Ausdrucken einer Hardcopy. Da es gelegentlich vorkommt, daß Sie diese Tastenkombination bei schneller Fingerakrobatik unbeabsichtigt treffen, ist es wünschenswert, sie abzuschalten und bei Bedarf wieder aktivieren zu können. Ziel ist also ein Programm, daß die gewünschte Einstellung anhand eines Parameters vornimmt. Dazu lesen Sie einen formatierten Parameter ein, dessen überflüssige Leerzeichen entfernt wurden. Sie werten nicht wie in Xecho die im PSP enthaltene Kommandozeile aus, sondern den ersten Dateisteuerblock FCB, da in diesem unbeabsichtigt eingegebene Leerzeichen bereits unterdrückt sind. Je nach Parameter sollten Sie also eine bestimmte Speicherzelle im Rechner setzen können, wie es »hcopy.asm« (Listing 5) vorführt. Rufen Sie »hcopy.com« mit

hcopy -

auf, so ist die Hardcopy-Funktion abgeschaltet. Mit

hcopy +

schalten Sie sie analog dazu wieder an. Sie sollten dazu die Bedeutung der verwendeten Speicherzelle 0050:0000hex genauer betrachten. Wenn Sie die Hardcopy-Tastenkombination betätigen, lösen Sie den Interrupt 5 aus, der die Hardcopy anfertigt. Drücken Sie während des Hardcopy-Vorgangs nochmals diese Tastenkombination, löst dies den Interrupt erneut aus. Da das gleichzeitige Anfertigen zweier Hardcopies aus verständlichen Gründen für Probleme sorgt, wird vor dem Beginn einer Hardcopy immer gefragt, ob nicht bereits eine in Arbeit ist. Die Speicherzelle 0050:0000hex (Zeile 37) gibt darüber Aufschluß. Hcopy gaukelt dem Hardcopy-Programm also nur vor, daß eine

Wert1	Wert2	AND-Operator	OR-Operator	XOR-Operator
X1	X2	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Tabelle 4. Logische Operatoren wandeln Eingabewerte nach einem festgelegten Formalismus um – der NOT-Operator inverteert lediglich.

Hardcopy gerade in Gange ist, indem es den Wert 1 in die Speicherzelle schreibt.

● Zahlen aus dem PSP einlesen

Das Auswerten von Zeichen aus dem PSP gelingt also recht einfach. Problematisch ist dagegen das Einlesen von Zahlen aus dem PSP, da Sie einzelne Ziffern, die als ASCII-Zeichen vorliegen, in eine einzige Zahl umwandeln müssen. »Seterr.asm« (Listing 6) setzt nun die Systemvariable »Errorlevel« mit einem Wert, den Sie als Parameter eingeben. Zusätzlich wird das Zeichen mit dem entsprechenden Code auf dem Bildschirm dargestellt. So erreichen Sie mit

seterr 65

nicht nur, daß die Variable »Errorlevel« diesen Wert annimmt, sondern auch, daß Sie den Buchstaben »A« am Bildschirm sehen, da diesem Buchstaben der ASCII-Wert 65dez zugeordnet ist. Das Auslesen und Konvertieren der Kommandozeile erfolgt hierbei über den ersten Dateisteuerblock. Nachdem Sie dem SI-Register die Adresse des PSP mitgeteilt haben, weisen Sie dem AH-Register den Wert 0 zu. In einer Schleife multiplizieren Sie dann die einzelnen Ziffern mit ihrem Stellenwert und addieren sie zum AL-Register. Dies erledigt der AAD-Befehl (»ASCII Adjust for Division«), der den Inhalt von AH mit 10 multipliziert und das entstandene Produkt zu AL addiert. Da der Lodsb-Befehl ein Ziffernzeichen aus dem ersten Dateisteuerblock in das AL-Register bewegt, kopieren Sie dieses in das AH-Register. Weil dieses Ziffernzeichen im ASCII-Format vorliegt, ziehen Sie hiervon 30hex (= 48dez) ab. Dies geschieht so lange, bis keine Ziffern mehr folgen. Anschließend wird das Zeichen im AH-Register auf dem Bildschirm angezeigt und an »Errorlevel« übertragen. Wie das Beispiel

seterr 234

mit dieser Eingabe verfährt, entnehmen Sie Tabelle 1.

● Zahlen anzeigen

Ein anderes Problem ist das Anzeigen von Zahlen. Wenn beispielsweise im AL-Register die Zahl 234 abgelegt ist, muß diese in drei ASCII-Zeichen umgewandelt werden.

234/10 = 23 Rest 4

23/10 = 2 Rest 3

2/10 = 0 Rest 2

Diese Zeichenfolge lesen Sie also als »234«, denn Sie müssen die ASCII-Werte der Reste, die bei der Division durch 10 entstehen, in umgekehrter Reihenfolge ausgeben. Das Problem lösen Sie, indem Sie während der Berechnung die Divisionsreste auf einen Stapel legen, den Sie nach der Berechnung in umgekehrter Reihenfolge auf dem Bildschirm ausgeben. Dieser Stapelspeicher heißt »Stack«. Mit dem Befehl »Push Register« legen Sie einen Registerinhalt auf dem Stapel ab. Den Wert, der zuletzt auf dem Stapel liegt, übertragen Sie mit dem Befehl »Pop Register« wieder in ein Register. »anzeige.asm« (Listing 7) gibt die im AL-Register gespeicherte Zahl 234dez auf dem Bildschirm aus.

● Schleifen im For-Next-Verfahren

Das CX-Register dient zur Aufnahme des Schleifenzählers »Counter CX«. Nachdem Sie das Schleifenregister mit einem Wert gefüllt haben, definieren Sie eine Sprungmarke, die den Anfang des Schleifenblocks festlegt. Nach dem Schleifenblock springt die Anweisung mit »Loop Sprungmarke« so lange zur »Sprungmarke«, bis das Register CX den Wert 0 angenommen hat. Bei jedem Sprung, den »Loop« auslöst, verringert (dekrementiert) sich das CX-Register um den Wert 1 (Tabelle 2).

Als kleine Demonstration dient »loop.asm« (Listing 8), das im Aufbau sehr dem Programm Beep ähnelt. In letzterem tauschten Sie das dargestellte ASCII-Zeichen aus und fügten eine Schleife hinzu. Nach dem Start von Loop zeichnet der Rechner eine Doppellinie (Zeile 14) in voller Bildschirmbreite. Loop kann nur eine Doppellinie von einer Länge von 80 Zeichen ausgeben. Da diese Werkstatt auch Hilfen für den täglichen Einsatz ergeben soll, sollten Sie das Programm so abändern, daß Sie mit Parametern die Länge des Balkens und des Zeichens, aus dem der Balken bestehen soll, festlegen können. Dazu kopieren Sie die Dateien »loop.asm« und »seterr.asm« zusammen und ändern die Inhalte entsprechend ab. Die Lösung finden Sie in »balken.asm« (Listing 9). Sie rufen das Kompilat mit

balken <Länge> <Zeichen>

auf. So erzeugt beispielsweise »balken 80 A« einen Balken, der aus 80 Zeichen »A« besteht. Die Ausgabe bleibt auf 255 Zeichen beschränkt.

● Logische Operationen

Die in der Algebra immer wieder verwendeten logischen Operatoren entnehmen Sie Tabelle 3. Die kleinste darstellbare Informationseinheit ist

Bit. Ein Bit kann entweder 1 (= gesetzt) oder 0 (= nicht gesetzt) sein. Die einfachste logische Operation ist die NOT-Operation, die den Wert eines Bits invertiert: Aus 0 wird 1, und aus 1 wird 0. Die komplexeren logischen Operatoren AND, OR und XOR verknüpfen zwei Bits und liefern ein logisches Ergebnis. Der Operator AND liefert nur dann als Ergebnis den Wert 1, wenn beide Eingangsbits den Wert 1 haben; in allen anderen Fällen liefert AND den Wert 0. Analog dazu liefert OR als Ergebnis die Zahl 1, wenn eines der Eingangsbits den Wert 1 hat. Folglich ist das Ergebnis dann auch 1, wenn beide Eingangsbits gleich 1 sind. Der Operator XOR liefert nur dann als Ergebnis den Wert 1, wenn ein Eingangsbit den Wert 1 hat. Sind beide Eingangsbits gleich 1, so ist bei der XOR-Operation das Ergebnis gleich 0. Es ergibt sich Tabelle 4 für die Eingänge X1 und X2.

Da der Rechner intern mit binären Zahlenwerten arbeitet, lassen sich diese Operationen auch auf Zahlen im dezimalen oder hexadezimalen System anwenden. Das Beispiel soll die dezimalen Werte 117 und 15 durch eine AND-Operation verknüpfen. Dies erreichen Sie in Basic mit »print 117 and 15«, wobei das Ergebnis den Wert 5 zurückliefert. Dies vollziehen Sie nach, indem Sie beide Dezimalzahlen in das Binärsystem umwandeln und die enthaltenen Bits einzeln verknüpfen (Tabelle 5).

Eine Anwendung der AND-Operation ist das Löschen von Bits in einem Byte. Wollen Sie Bits in einem Byte löschen, verknüpfen Sie den Wert der Bits, die Sie erhalten wollen, durch die AND-Operation mit dem zu bearbeitenden Byte. Wenn das siebte Bit (Wert 128) im Byte 255 gelöscht werden soll, gilt bei Basic:

```
print 255 and (255-128)
```

Als Ergebnis sehen Sie den Wert 127 am Schirm.

Um nun ein bestimmtes Bit in einem Byte zu setzen, bedienen Sie sich der OR-Operation. Das Byte wird einfach per OR-Operation mit dem Bit verknüpft, das gesetzt werden soll. Folgendes Beispiel stellt eine Probe zur OR-Operation dar:

```
print 127 or 128
255
```

Im folgenden wenden Sie dieses Wissen nun sinnvoll in der Praxis an. Die

Dezimalzahl	Binärwert
117	0111 0101
15	0000 1111
5	0000 0101

Tabelle 5. Die Zahlen 117 und 15 verknüpft der AND-Operator 5.

PC-Tastatur hat einige sogenannte Lock- oder Feststell Tasten, wie etwa die Taste [Num]. Wenn eine Lock-Taste einmal gedrückt wird, ist diese so lange aktiv, bis sie ein zweites Mal betätigt wird. Die Stellung der Lock-Tasten wird nicht mechanisch wie etwa bei einer Deckenlampe gespeichert, sondern in der Speicherzelle 0040:0017hex. Jeder der drei Lock-Tasten ist ein Bit zugeordnet, das gesetzt ist, wenn diese aktiv ist: Dann leuchtet also die entsprechende Diode auf der Tastatur. Sie steuern das Programm durch Parameter, die in der Kommandozeile enthalten sind. Da die PC-Tastatur drei Feststell-Tasten hat, können Sie die Kommandozeile nicht über Dateisteuerblöcke auswerten, da diese nur zwei Mal vorhanden sind.

»Led.asm« (Listing 10) löst das Problem, indem es die Kommandozeile direkt auswertet. Wie Sie das Kompi-lat aufrufen, um die Syntax ab Zeile 7 zu lesen, sehen Sie in Bild 2. Der Pro-

Befehl	Logik	Test	Bemerkung, (Sprung wenn)
JA	X > 0	(CF AND ZF) = 0	größer als 0
JAE	X >= 0	CF = 0	größer oder gleich 0
JB/JC	X < 0	CF = 1	kleiner 0, oder Übertrag (Carry)
JBE	X <= 0	(CF OR ZF) = 1	kleiner gleich 0
JE/JZ	X = 0	ZF = 1	gleich 0, A = B
JG	X > Y	((SF XOR OF) OR ZF) = 0	größer
JGE	X >= Y	(SF XOR OF) = 0	größer oder gleich
JL	X < Y	(SF XOR OF) = 1	kleiner
JLE	X <= Y	((SF XOR OF) OR ZF) = 1	kleiner oder gleich
JCXZ	CX = 0	CX-Register gleich 0	

Tabelle 6. Wichtige bedingte Sprungbefehle der CPU 80X86/88

zessor setzt unter bestimmten Bedingungen die Flags. Ein Flag gibt an, ob eine bestimmte Bedingung erfüllt wird oder nicht. Ein Flag besitzt also nur zwei Zustände: »gesetzt« oder »nicht gesetzt«. Das Setzen findet bei Rechenoperationen oder Vergleichen statt. Eine Abfrage tritt nach einer Rechenoperation »JA, JAE, JB, JBE, JE« oder nach einem Vergleich mit »CMP«, »JE, JG, JGE, JL, JLE, JE« auf. Der JE-Befehl wird also sowohl bei Vergleichen mit Null als auch bei Vergleichen mit einem beliebigen Argument verwendet. Ferner können Sie den Zustand des Zählerregisters CX mit »JCXZ« abfragen. Teilweise sind in der Tabelle 6 gleich zwei Befehle aufgeführt. Diese sind gleichbedeutend. In der Regel existiert zu jedem bedingten Sprungbefehl sogar noch ein weiterer. Daher sind nur die wichtigsten Sprungbefehle aufgeführt.

Die Bildschirmmeldungen der Hilfe finden Sie in den Zeilen 7 bis 19. Daß Sie mit der Syntax

led?

diesen Text lesen, entnehmen Sie der Zeile 27, die schon im nächsten Schritt zur Zeile 81, Unterprogramm »Hilfe«, verweist. Wie Sie Texte ausgeben, verfolgen Sie darüber hinaus im Miniprogramm »text.asm«.

Weitere Parameter, die Sie Led übergeben, wandelt das Programm zunächst in Großbuchstaben um. Wie dann die Eingabe der Zeichen »S«, »N« und »C« zu den Marken »Scroll«, »Num« und »Cape« verzweigt, lesen Sie in den Zeilen 31, 33 und 35. Von diesen Sprungzielen aus führen dann Jmp-Befehle zu »anoderaus« (Zeile 47), und von dort geht die Reise abhängig von der übergebenen Zahl »0« oder »1« weiter nach »Aus« oder »An«. Bei dieser Verfolgungsjagd der Sprungmarken durch einzelne Listingzeilen enthüllt das Programm also selbst relativ unkundigen Einsteigern seine Geheimnisse. Nicht viel anders verläuft diese Arbeit jedoch auch bei großen Programmen.

(Ulrich Borgdorf/et)

Info:

- Computer Solutions, 8018 Grafting, Shareware-Disk Nr. 0905 (deutsche Anleitung), Nr. 0114 (A86/D86, Makro-Assembler)
- Günter Born: A86/D86 - Assembler-Programmierung für Einsteiger, Systhema-Verlag München, ISBN 3-89390-111-6
- Kressmann: Mathematik für Datenverarbeitungskaufleute, Winkler Verlag Darmstadt, ISBN 3-8045-5218-8

```
[KA3] 1: ;Programm: beep.asm
[DZ4] 2: ;Funktion: Ausgabe eines Beeps
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[E21] 6: Start:
[BL5] 7: MOV DL,07h ; Gib [Strg-G] (Code 07h)
```

```
[TV4] 8: MOV AH,02h ; auf Bildschirm aus
[C95] 9: INT 21h ; (Funktion 02h/INT 21h).
[XA1] 10: Weiter:
[B85] 11: MOV AH,4Ch ; Beende das Programm
[V55] 12: INT 21h ; (Funktion 4C/ INT 21h)
```

Listing 1. »beep.asm« gibt einen Ton aus.


```
[7R2] 1: ;Programm: ff.asm
[716] 2: ;Funktion: Form-Feed auf dem Drucker
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[E21] 6: Start:
[S16] 7: MOV DL,12 ; CHR(12)=Seitenvorschub
[5S4] 8: MOV AH,05h ; Funktion 05 des
[CK4] 9: INT 21h ; Interrupts 21h
[XT0] 10: Ende:
[RC5] 11: MOV AH,4Ch ; Beende das Programm
[E95] 12: INT 21h ; (Funktion 4Ch/ INT 21h)
```

Listing 2. »ff.asm« spendiert Ihrem Drucker einen Seitenvorschub (Form Feed).

```
[DD3] 1: ;Programm: text.asm
[K76] 2: ;Funktion: Ausgabe einer Zeichenkette
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[BW4] 6: JMP Start ; Gehe zu Start
[VA6] 7: Doku DB "DOS Extra, A-86-Werkstatt",13,10
[8R4] 8: DB "Ulrich Borgdorf/et",13,10,"$"
[Z24] 9: Start: MOV DX,Offset DOKU
[Y24] 10: ; Speichere Adresse von
[AB3] 11: ; Doku im DX-Register
[XR4] 12: MOV AH,09h ; Rufe Funktion 09h
[DM5] 13: INT 21h ; des Interrupts 21h auf
[6B5] 14: Ende: MOV AH,4Ch ; Beende Programm
[3B5] 15: INT 21h ; (Funktion 4C/INT 21hex)
```

Listing 3. »text.asm« gibt Zeichenketten am Bildschirm aus.

```
[CB3] 1: ;Programm: xecho.asm
[6P5] 2: ;Funktion: Kommandozeile ausgeben
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[4R5] 6: Init: JMP Start ; Gehe zu Start
[RK6] 7: Doku DB "Keine Zeichen in Kommandozeile",
[YC1] 8: DB 13,10,"$"
[9Z4] 9: Start: MOV SI,82h ; Zeiger auf
[GP2] 10: ; Kommandozeile
[L73] 11: CMP BYTE PTR [80h],0
[UC3] 12: ; Wenn Kommandozeile
[6C4] 13: JZ Hilfe ; leer, dann Ende
[K93] 14: ; (BYTE PTR[80]) gibt
[VG2] 15: ; die Länge der
[083] 16: ; Kommandozeile an)
[X62] 17: ;----- Schleife
[226] 18: Weiter: LODSB ; Hole Zeichen von
[GAI] 19: ; Zeiger
[ZU4] 20: CMP AL,0Dh ; Zeichen=RETURN ?
[DU3] 21: JZ Ende ; Ja, dann Ende
[HA3] 22: CMP AL,'$' ; Zeichen="$"
[274] 23: JZ Dollar ; Ja dann Dollar
[R36] 24: Zurueck: CALL Schreib; Upro: Schreib
[K35] 25: JMP Weiter ; Gehe nach Weiter
[WL3] 26: ;----- Ende der Schleife
[AB5] 27: Dollar: LODSB ; Hole Zeichen
[F73] 28: CMP AL,0D ; =RETURN
[884] 29: DEC SI ; dann Zeiger auf
[AB3] 30: ; alten Wert setzen
[K05] 31: JZ Zurueck ; und nach Zurueck
[5F4] 32: INC SI ; Zeiger nächstes
[8R2] 33: ; Zeichen setzen
[VR3] 34: CMP AL," " ; Zeichen=" "
[BH4] 35: JZ Return ; Ja, dann Return
[LM3] 36: CMP AL,"E" ; Zeichen="E"
[W94] 37: JZ Escape ; Ja, dann Escape
[LM3] 38: CMP AL,"e" ; Zeichen="e"
[W94] 39: JZ Escape ; Ja, dann Escape
[LH4] 40: MOV AL,"$" ; Wenn nach $ keins
[233] 41: ; der Zeichen e,E,_
[012] 42: ; folgt, dann
[EX3] 43: CALL Schreib ; Schreibe $
[4X4] 44: DEC SI ; Stelle Stringindex
[NX1] 45: ; wieder her
[LW3] 46: JMP Weiter ; und Weiter
[000] 47:
[U75] 48: Escape: MOV AL,27 ; Gib CHR$(27)
[0W2] 49: CALL Schreib ; aus
[LW3] 50: JMP Weiter ; und Weiter
[000] 51:
[0A5] 52: Return: MOV AL,13 ; Gib CHR$(13)
[VF4] 53: CALL Schreib ; (Return) aus
[DK3] 54: MOV AL,10 ; Gib CHR$(10)
[0N4] 55: CALL Schreib ; (Line-Feed) aus
[LW3] 56: JMP Weiter ; und Weiter
[000] 57:
[L35] 58: Schreib: ; Stelle Zeichen
[KX4] 59: MOV AH,02h ; mit dem ASCII-Code,
```

```
[GD3] 60: MOV DL,AL ; der in AL
[CK4] 61: INT 21h ; gespeichert ist,
[FF4] 62: RET ; auf Bildschirm dar
[000] 63:
[2G4] 64: Hilfe: MOV DX,Offset DOKU
[P25] 65: MOV AH,09h ; >>siehe TEXT.ASM<<
[381] 66: INT 21h
[000] 67:
[U75] 68: Ende: MOV AH,4Ch ; Beende Programm
[381] 69: INT 21h
```

Listing 4. »xecho.asm« liest die Zeichen aus der Kommandozeile und gibt sie auf dem Bildschirm wieder.

```
[6D3] 1: ;Programm: hcopy.asm
[5D5] 2: ;Funktion: Hardcopy an oder aus
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[B65] 6: JMP Start ; Gehe zu Start
[CR4] 7: DokuHilfe DB ">>HCOPY [-/+]<<"
[EL4] 8: DB " (de)/aktiviert Hardcopy"
[DA1] 9: DB 13,10,"$"
[986] 10: DokuAn DB "Hardcopy aktiviert",13,10,"$"
[8S5] 11: DokuAus DB "Hardcopy deaktiviert"
[DA1] 12: DB 13,10,"$"
[6B6] 13: Start: MOV SI,82h ; Setze Stringindex
[N83] 14: ; auf Kommandozeile
[XA1] 15: Weiter:
[WW3] 16: LODSB ; Hole Zeichen
[LH3] 17: CMP AL,'-' ; Zeichen="-"
[RR3] 18: JZ Aus ; Ja, dann Aus
[7H3] 19: CMP AL,'+' ; Zeichen="+"
[993] 20: JZ An ; Ja, dann An
[034] 21: MOV DX,Offset DokuHilfe
[H22] 22: ; Sonst Hilfetext
[FX3] 23: JMP Schreibe ; ausgeben
[7S0] 24: Aus:
[SL3] 25: MOV DX,Offset DokuAus
[9D1] 26: MOV BL,01
[E32] 27: JMP Schreibe
[MK0] 28: An:
[BE3] 29: MOV DX,Offset DokuAn
[7D1] 30: MOV BL,00
[E32] 31: JMP Schreibe
[7G1] 32: Schreibe:
[TC1] 33: PUSH BX
[1D5] 34: PUSH DS ; Rette Datensegment
[SU4] 35: PUSH DI ; Rette Datenindex
[133] 36: ; (jeweils auf Stack)
[F73] 37: MOV AX,0050h ; Setze
[8Y5] 38: MOV DS,AX ; Datensegment auf 40hex
[696] 39: MOV DI,00h ; und Datenindex auf 17hex
[6K3] 40: MOV BYTE PTR[DI],BL
[D75] 41: POP DI ; Hole Datenindex und
[9X5] 42: POP DS ; Datensegment vom Stack
[T31] 43: POP BX
[3M1] 44: MOV AH,09h
[381] 45: INT 21h
[XT0] 46: Ende:
[KW4] 47: MOV AH,4Ch ; Beende Programm
[3N5] 48: INT 21h ; (Funktion 4C/INT 21hex)
```

Listing 5. »hcopy.asm« schaltet die Hardcopy-Funktion an und aus.

```
[SF3] 1: ;Programm: seterr.asm
[X66] 2: ;Funktion: Zahlen aus dem PSP auslesen
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[E21] 6: Start:
[9V5] 7: MOV SI,5Dh ; Index auf 1. FCB setzen
[K04] 8: MOV AX,00h ; AH=0 und AL=0
[W91] 9: Wandle:
[LA3] 10: AAD ; AL=AL+AH*10.
[ZW2] 11: MOV AH,AL ; AH=AL
[L66] 12: LODSB ; Hole nächstes Zeichen aus
[SU1] 13: ; FCB nach AL
[5K5] 14: SUB AL,30h ; ASCII nach Binärcode
[CC6] 15: JNC Wandle ; Wenn weitere Ziffer vor-
[S04] 16: ; handen, gehe zu Schleife
[KK5] 17: MOV DL,AH ; Stelle Zeichen durch
[TR5] 18: MOV AH,2 ; die Funktion 02h des
[KV4] 19: INT 21h ; Interrupts 21h dar.
[8V5] 20: MOV AL,DL ; Gib eingelesene Zahl an
[XT0] 21: Ende:
[GA5] 22: MOV AH,4Ch ; ERRORLEVEL und Ende
[K75] 23: INT 21h ; (Funktion 4Ch/INT 21h)
```

Listing 6. »seterr.asm« übergibt einen Parameter an »Errorlevel«.


```
[5P3] 1: ;Programm: anzeige.asm
[6T4] 2: ;Funktion: Ausgabe einer Zahl
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[E21] 6: Start:
[S93] 7: MOV AL,234 ;AL=234dez
[XK5] 8: CALL Anzeige;Rufe Unterprogramm
[DL4] 9: MOV AH,4Ch ;Beende Programm
[PE5] 10: INT 21h ;Funktion 4Chex/INT 21)
[EE1] 11: Anzeige:
[864] 12: MOV BL,10 ;BL=10 (Basis des
[EG4] 13: ; dezimalen Zahlensystems)
[NA1] 14: Ziffer:
[7G2] 15: MOV AH,00h ;AH=0
[YX4] 16: DIV BL ;AL=AL/10 Rest nach AH
[X75] 17: PUSH AX ;AH auf Stack (Stapel)
[NP2] 18: CMP AL,00 ;AL=0 ?
[T94] 19: JZ Disp ;Ja, dann Anzeige
[006] 20: CALL Ziffer ;Nein, dann rufe Ziffer
[LV0] 21: Disp:
[WAS] 22: POP AX ;Hole Ziffer von Stack
[9D5] 23: ADD AH,48 ;Rechne diese ins ASCII-
[LD1] 24: ; Format um
[A15] 25: MOV DL,AH ; und stelle Ziffer
[H96] 26: MOV AH,02h ; mit Hilfe der Funktion 02h
[WX5] 27: INT 21h ; des Interrupts 21hex dar.
[3P0] 28: RET
```

Listing 7. »anzeige.asm« gibt die im AL-Register gespeicherte Zahl 234dez auf dem Bildschirm aus.

```
[EC3] 1: ;Programm: loop.asm
[FR4] 2: ;Funktion: Schleifenzähler
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[E21] 6: Start:
[NZ5] 7: MOV CX,80 ; Die Schleife soll 80 mal
[LB3] 8: ; durchlaufen werden
[HG1] 9: Schleife:
[PL5] 10: MOV DL,205 ; Stelle Zeichen 205 (=)
[NY5] 11: MOV AH,02h ; mit Hilfe der Funktion
[VC4] 12: INT 21h ; 02h/ INT 21h dar.
[E06] 13: LOOP Schleife ; CX um eins verringern
[EW3] 14: ; und wenn nicht Null, zu
[LB3] 15: ; Schleife springen
[XT0] 16: Ende:
[YC5] 17: MOV AH,4Ch ; Beende das Programm
[3K5] 18: INT 21h ; (Funktion 4Ch/INT 21h)
```

Listing 8. »loop.asm« zeichnet eine Doppellinie.

```
[AE3] 1: ;Programm: balken.asm
[8E6] 2: ;Funktion: Balken mit vorg.Wert+Länge
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[E21] 6: Start:
[4Z5] 7: MOV SI,5Dh ; Index auf 1. FCB setzen
[574] 8: MOV AX,00h ; AH=0 und AL=0
[W91] 9: Wandle:
[LA3] 10: AAD ; AL=AL+AH*10.
[XW2] 11: MOV AH,AL ; AH=AL
[L66] 12: LODSB ; Hole nächstes Zeichen aus
[5U1] 13: ; FCB nach AL
[HR5] 14: SUB AL,30h ; ASCII nach Binärcode
[BMS] 15: JNC Wandle ; Wenn weitere Ziffer
[VS4] 16: ; vorhanden gehe zu Schleife
[WY3] 17: MOV AL,AH ; Übergib das
[UD4] 18: MOV AH,00h ; AH-Register an
[YR4] 19: MOV CX,AX ; das CX-Register
[816] 20: CMP CX,00h ; Ist der Balken 0 Zeichen
[R31] 21: ; lang ?
[9V3] 22: JZ Ende ; Ja, dann Ende
[HG1] 23: Schleife:
[DW5] 24: MOV DL,[6Dh] ; Stelle Zeichen durch
[4H5] 25: MOV AH,02h ; die Funktion 02h des
[BV4] 26: INT 21h ; Interrupts 21h dar.
[FV5] 27: MOV AL,DL ; Gib eingelesene Zahl an
[T72] 28: LOOP Schleife
[XT0] 29: Ende:
[N96] 30: MOV AH,4Ch ; AL an ERRORLEVEL und Ende
[LB5] 31: INT 21h ; (Funktion 4Ch/INT 21h)
```

Listing 9. »balken.asm« stellt Balken mit zwei Parametern dar.

```
[2Y2] 1: ;Programm: led.asm
[N75] 2: ;Funktion: logische Operatoren
[D12] 3: ;Sprache: A86
[W24] 4: ;Autor: Ulrich Borgdorf
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[PM4] 6: JMP Start ; Gehe zu Start
[386] 7: Doku DB "LED [Nn] [Cn] [Sn] ",13,10,
[4C3] 8: DB " ",13,10,
[7A4] 9: DB " N=Numlock ",13,10,
[1E4] 10: DB " C=Capslock ",13,10,
[TM4] 11: DB " S=Scrolllock ",13,10,
[4C3] 12: DB " ",13,10,
[674] 13: DB " n=1 ==> aus ",13,10,
[P44] 14: DB " n=0 ==> an ",13,10,
[4C3] 15: DB " ",13,10,
[3V3] 16: DB "Bsp.: ",13,10,
[8E5] 17: DB "LED N0 schaltet Num aus ",13,10,
[LB5] 18: DB "LED N1 C1 S1 schaltet alle",13,10,
[MR4] 19: DB " Funktionen ein",13,10,
[YM0] 20: DB "$"
[E21] 21: Start:
[T05] 22: MOV SI,82h ; Setze Stingindex
[XA1] 23: Weiter:
[C45] 24: LODSB ; Hole Zeichen aus PSP
[5C5] 25: CMP AL,0Dh ; Wenn Zeichen gleich
[KS4] 26: JZ Ende ; Return dann Ende
[8P5] 27: CMP AL,"?" ; Ist Zeichen gleich "?"
[LE4] 28: JZ Hilfe ; Ja, dann Hilfe
[9U6] 29: AND AL,5Fh ; Wandle in Großbuchstaben
[6Y5] 30: CMP AL,"s" ; Wenn Buchstabe="s", dann
[EU4] 31: JZ Scroll ; gehe zu Rollen
[3X5] 32: CMP AL,"N" ; Wenn Buchstabe="N", dann
[7U3] 33: JZ Num ; gehe zu Num
[PUS] 34: CMP AL,"C" ; Wenn Buchstabe="C", dann
[YY3] 35: JZ Caps ; gehe zu Caps
[7C4] 36: JMP Weiter ; und Weiter
[AB1] 37: Scroll:
[YM5] 38: MOV BH,16 ; Rollen ist das 4. Bit
[W34] 39: ; zugeordnet: Wert=2^4=16
[KW5] 40: JMP AnOderAus; gehe zu An oder Aus
[9S0] 41: Num:
[4A5] 42: MOV BH,32 ; 5. Bit also Wert=2^5
[5N2] 43: JMP AnOderAus;
[GU0] 44: Caps:
[UA5] 45: MOV BH,64 ; 6. Bit also Wert=2^6
[AL2] 46: JMP AnOderAus
[TS1] 47: AnOderAus:
[RM5] 48: LODSB ; Hole nächstes Zeichen.
[5L5] 49: CMP AL,"0" ; Wenn Zeichen gleich 0,
[Z83] 50: JZ Aus ; dann AUS.
[CL5] 51: CMP AL,"1" ; Wenn Zeichen gleich 1,
[VV2] 52: JZ An ; dann AN
[FR5] 53: JMP Weiter ; Wenn weder Zeichen
[3D3] 54: ; gleich 0 oder 1, dann
[BB1] 55: ; Weiter
[7S0] 56: Aus:
[LD5] 57: PUSH DS ; Rette Datensegment
[SU4] 58: PUSH DI ; Rette Datenindex
[133] 59: ; (jeweils auf Stack)
[C73] 60: MOV AX,0040h ; Setze
[8Y5] 61: MOV DS,AX ; Datensegment auf 40hex
[N96] 62: MOV DI,17h ; und Datenindex auf 17hex
[GD6] 63: MOV BL,255 ; Lösche das Bit mit dem
[566] 64: SUB BL,BH ; in BH gespeicherten Wert
[3F3] 65: AND BYTE PTR[DI],BL
[D75] 66: POP DI ; Hole Datenindex und
[9X5] 67: POP DS ; Datensegment vom Stack
[7B4] 68: JMP Weiter ; und Weiter
[YR5] 69: An: ; Entspricht Upro >>Aus<<
[BC1] 70: PUSH DS
[3B1] 71: PUSH DI
[DV1] 72: MOV AX,0040h
[1H1] 73: MOV DS,AX
[9M1] 74: MOV DI,17h
[P63] 75: OR BYTE PTR[DI],BH
[WG3] 76: ; Setze jedoch Bit mit
[LA4] 77: ; dem in BH gesetzten Wert
[G21] 78: POP DI
[E31] 79: POP DS
[GUL] 80: JMP Weiter
[T01] 81: Hilfe:
[T33] 82: MOV DX,Offset DOKU
[BU5] 83: MOV AH,09h ; Gib Hilfefunktion aus
[WD5] 84: INT 21h ; (siehe auch TEXT.ASM)
[XT0] 85: Ende:
[KW4] 86: MOV AH,4Ch ; Beende Programm
[3N5] 87: INT 21h ; (Funktion 4C/INT 21hex)
```

Listing 10. »led.asm« schaltet die Feststell-Tasten des Rechners ein und aus.

Tips & Tricks

Tips & Tricks für Assembler

Ob mit MASM, TASM oder Debug – Assembler-Programmierer entschlüsseln die Geheimnisse des Systems: Sie schützen es vor Viren und vor unberechtigtem Zugriff, schalten die Frequenz des Monitors um, sichern das CMOS-RAM und verändern die Befehle des Betriebssystems.

Schutz vor Computerviren und anderen Sabotageprogrammen

Angesichts der nach wie vor stark steigenden Zahl von Computerviren und anderer schadenstiftende Programmen gewinnen die Früherkennung und der wirksame präventive Schutz (Bild 1) gegen derartige Produkte übler Zeitgenossen mehr und mehr an Bedeutung für jeden PC-Anwender.

• Viren, Würmer, Trojanische Pferde und Logische Bomben

Ein Computervirus ist bekanntlich keine mystische Erscheinung und kein Lebewesen wie sein biologischer Namensvetter, sondern eine durch Menschen zielgerichtet geschaffene Programmsequenz. Einmal in ein anderes Programm eingepflanzt, reproduziert sie sich, indem sie das befallene Wirtsprogramm zur weiteren Verbreitung des Virencodes veranlaßt. Computerviren enthalten neben den Routinen zur Reproduktion Programnteile, die eine vorgegebene Wirkung auslösen. Sie kann vom »harmlosen« Schabernack bis zum handfesten Schaden reichen.

Andere schadenstiftende Programmsequenzen oder Programme werden unter den Namen »Würmer«, »Trojanische Pferde« oder »Logische Bomben« geführt. Erstere unterscheiden sich von Viren durch ihre prinzipiell andere Reproduktionsstrategie. Würmer sind selbständige Programme, die sich keines Wirtsprogramms bedienen. Sie werden durch ihre Programmierer meist in Netzen eingepflanzt. Sie werden in der Regel von einem zum anderen Computer in Form von Quelltexten übertragen, die erst im Empfängersystem kompiliert und gestartet werden.

Unter Trojanischen Pferden versteht man eigenständige Programme, die vorgeben, mehr oder weniger nützliche Funktionen zu erfüllen. Hinter dieser Fassade haben die Programmierer jedoch Routinen versteckt, die Schaden stiften oder Daten ausspionieren – zum Beispiel Paßworte in einem Netz. Trojanische Pferde enthalten keine Programnteile zur Reproduktion, sie verbreiten sich als Demo-Disketten oder über DFÜ-Kanäle.

Logische Bomben sind separate Schadensroutinen, die zum Beispiel jemand, der sich wegen einer vermeintlich ungerechten Behandlung rächen will, in Programme der Rechner anderer einpflanzt. Logische Bomben enthalten keine Routinen zur Reproduktion und sind – im Gegensatz zu Trojanischen Pferden – auch nicht zur Verbreitung auf andere Systeme bestimmt.

Es ist schon erschreckend, wieviel menschliche Erfindungskraft auf die Herstellung destruktiver Software verschwendet wird. Die Folgeschäden sind gravierend – wie auch die Aufwendungen zur Vorbeugung und Schadensbegrenzung, die eigentlich sinnvoller Aufgaben gewidmet werden könnten. Über die Motive, die Menschen zur Schaffung derartiger Produkte inspiriert, ist viel philosophiert worden. Ob es sich nun um »computerkundige Terroristen«, gelangweilte Genies oder geistig gestörte Soziopathen« (McAfee) handelt, bleibe dahingestellt. Die starken menschlichen Motive des Hasses, des Neids und der Gewinnsucht spielen gewiß eine erhebliche Rolle.

• Viren im PC

Unter den verschiedenen Formen destruktiver Software fallen in der PC-Welt die Computerviren weitaus am meisten ins Gewicht. Daher erscheint es gerechtfertigt, sich im folgenden weitgehend auf deren Besonderheiten zu beziehen. Ausgangspunkt einer sachlichen Analyse zum Schutz der PC-Systeme vor Viren sind deren Techniken zur Ausbreitung und Tarnung.

Das Ziel eines Virus, das ihm von seinem Programmierer mitgegeben wurde, besteht darin, in kurzer Zeit viele Programme und Datenträger auf vielen Rechnern zu infizieren, ohne vorzeitig entdeckt zu werden. Viren nutzen mehr oder weniger alle Dateien oder Strukturen als Wirtsprogramm, die ausführbaren Code enthalten. Auch der im Boot-Record befindliche Bootstrap-Lader des MS-DOS oder das Ladeprogramm eines Master-Boot-Records werden gezielt als Träger einer Infektion ausgewählt. Computerviren im CMOS-RAM oder in Dateien, die nur Daten, aber keine Programme enthalten, gehören jedoch ins Reich der Fabel; diese Bereiche können jedoch Angriffsziel einer Schadensroutine eines Virus sein. Zu einem gegebenen Zeitpunkt oder in bestimmten vorgegebenen Situationen macht der Virus meist mit Nachdruck auf sich aufmerksam – teils mit Witz, teils mit geistlosen Spielchen, zunehmend jedoch mit gezielter Sabotage der Arbeit am PC und des Datenbestands.

Infizierte Programme müssen durch den Virus in irgendeiner Weise verändert werden: Das Wirtsprogramm wird durch den Virencode kurzerhand überschrieben, oder der Virencode wird angehängt (Link-Viren). Dabei werden Sprungbefehle an den Eintrittspunkten »gepatcht« oder Relokationstabellen umorganisiert, damit der Virencode vor dem eigentlichen Programm abläuft. Der weitaus größte Teil der Computerviren unter MS-DOS gehört in die Gruppe der Link-Viren. Überschreibende Viren haben wenig Chancen, da sie schnell zu entdecken sind. Das durch sie befallene Programm kann ja – für den Anwender sofort offensichtlich – seine eigentliche Aufgabe nicht mehr erfüllen. Neuere Vi-

```
C:\EXT\21\PROGRAMM>chksys

ChkSys 1.1 - (C) 9/92 Christian Raabe
Kontrolliert wichtige Parameter des PC-Systems

Parameter neu in Datei!

C:\EXT\21\PROGRAMM>chksys

ChkSys 1.1 - (C) 9/92 Christian Raabe
Kontrolliert wichtige Parameter des PC-Systems

Interrupt-Vektoren - ok
MCB-Kette          - ok
Partition-Sektor   - ok
Boot-Sektor        - ok

C:\EXT\21\PROGRAMM>
```

Bild 1. »chksys.com« sichert und überwacht Interrupt-Vektoren, MCB-Kette, Partitions- und Boot-Sektor.

ren modifizieren nur kleine Codesequenzen der befallenen Programme und mißbrauchen Bereiche auf den Datenträgern oder innerhalb der DOS-Datenstrukturen, die durch MS-DOS nicht benutzt, oder auf die mit DOS-Funktionen nicht direkt zugegriffen werden kann. Sie legen ihren eigenen Code oder ihre Parameter dort ab und verbergen ihn so vor dem Anwender. Teilweise erzeugen sie derartige Bereiche selbst (zum Beispiel Bad Clusters). Die Veränderungen an den Programmdateien und Parameterstrukturen sind unmittelbar nach einer Infektion vorhanden und mehr oder weniger leicht erkennbar.

Die erste Aktion, die ein Virus ausführt, ist meist die Installation von residentem Code, wobei es BIOS- und DOS-Interrupts auf eigene Routinen umleitet oder neue Interrupts installiert. Ziel dieser Aktion ist es, die nachfolgenden Schritte zur Reproduktion des Virus zu organisieren sowie die beabsichtigten Wirkungen auszulösen. Ein residenter Virus kann mehr oder weniger alle Dateifunktionen des MS-DOS zur Infektion weiterer Programmdateien heranziehen: Jede Programmdatei, die ausgeführt, die geöffnet, ja selbst die, nach der lediglich auf dem Datenträger gesucht wird, kann infiziert werden. Typisch für viele neuere Viren ist bei all diesen Schritten das Umgehen dokumentierter MS-DOS-Funktionen. Einige Viren vermeiden sogar die direkte Auslösung von Interrupts.

Je früher es einem Virusprogramm während der Systemgenerierung gelingt, sich im PC resident zu etablieren, desto größer sind seine Mittel, die Kontrolle über das System zu erlangen. Direkte Auswirkungen des Virus (zum Beispiel die Verlängerung der befallenen Programmdateien) können so – vor dem Auftreten der beabsichtigten Wirkungen – versteckt, Virens Scanner oder andere Kontrollprogramme getäuscht werden. Die Wahrscheinlichkeit, daß der Virus sich unentdeckt über viele Programme und Datenträger verbreiten kann und an andere Systeme weitergegeben wird, wächst damit stark an.

Die ersten Programme, die beim Systemstart – nach den nicht veränderbaren Routinen des ROM-BIOS – die Kontrolle erhalten, sind die Ladeprogramme im Master-Boot- und/oder Boot-Record des Startmediums. Daraus folgt unmittelbar, daß Boot-Viren, die diese Ladeprogramme zur Installation und Reproduktion mißbrauchen, große Chancen haben, sich auftragsgemäß zu verbreiten.

Neben den »indirekt« wirkenden Viren gibt es noch die Klasse der »direkt« wirkenden, bei denen kein residenter Code installiert wird. Nur beim direkten Aufruf eines infizierten Programms wird der Virus aktiv. Er sucht direkt nach nicht infizierten Programmen, um sie zu modifizieren. Gesteuert zum Beispiel durch einen Zufallsgenerator, einen Zähler oder andere Auslösemechanismen, aktiviert er unmittelbar den Aktionsteil.

● Die Suche nach Computerviren

Virenerkennungsprogramme (Scanner) sind für viele Anwender zum Mittel der Wahl geworden, um ihren PC oder ihr Netzwerk vor Viren zu schützen. Viele PC-Anwender veranlassen bereits beim Systemstart die Suche nach eventuell eingeschleppten Computerviren: Sie rufen in der »autoexec.bat« einen Virens Scanner auf. Sie opfern freudig ihre Arbeitszeit für das Durchmusteren des Arbeitsspeichers, ja ganzer Festplattenbestände nach den für Viren charakteristischen Zeichenfolgen, denn die Meldung »No virus found« vermittelt ein beruhigendes Gefühl der Sicherheit. Wird dann doch einmal ein »Israeli« oder ein »Yankee Doodle« gefunden, aktivieren Sie flugs das (integrierte) Reparaturprogramm, den Viren-Cleaner, um den virulenten Code aus dem Programm herauszuoperieren. Alles paletti? Schön wär's.

Virens Scanner können prinzipiell nur bekannte Spezies entdecken. Während die Vielzahl modifizierter Viren noch vergleichsweise schnell einbezogen werden, wächst der zeitliche Abstand zwischen dem Auftreten prinzipiell neuer Viren und deren Berücksichtigung in den Scannern. Die Up-

dates der Scanner müssen also immer schneller aufeinander folgen. Dies zwingt den Anwender zu steigenden Aufwendungen und Kosten, will er nicht Gefahr laufen, das Opfer einer sich lawinenartig ausbreitenden Infektion zu werden. Gleichzeitig verschlechtert sich mit der wachsenden Zahl von Viren und deren Mutanten auch das Verhältnis der Anzahl aufgetretener zur Anzahl der sicher durch Scanner erkannten Viren; zudem wächst die Zahl der ausgelösten Fehlalarme.

Die Entwickler von Virens Scannern stehen vor der schwierigen Aufgabe, neu aufgetretene Viren zu analysieren, Signaturen und Mechanismen zu deren sicheren Erkennung zu finden und diese in neuen Programmversionen zu berücksichtigen. Diese komplizierte und zeitaufwendige Arbeit wird durch die Heimtücke der Viren-Programmierer zunehmend schwieriger. Diese setzen nämlich alles daran, einerseits die Entdeckung des Virencodes in den befallenen Programmdateien oder als residentes Programm im Speicher zu verhindern, andererseits dessen Reassemblierung zu erschweren.

Es leuchtet daher ein, daß sich der Vorsprung der Virenprogrammierer nicht aufholen läßt. Also muß man weitere Mittel und Wege finden, um die PC-Systeme wirkungsvoll vor einer Infektion zu schützen, den verursachten Schaden zu begrenzen und die Ausbreitung von Viren früh zu erkennen und zu unterbinden. Gute Virens Scanner haben bei der Realisierung dieser Aufgabe ihren wichtigen und unumstrittenen Platz, wenn auch keine Monopolstellung.

Es muß eindeutig gesagt werden: Viren-Scanner sind keine Standardprogramme für den täglichen Gebrauch, ist doch die durch sie vermittelte Sicherheit trügerisch. Denn die Zahl der Viren steigt rasant, sie verbreiten sich sehr schnell, und die Virenprogrammierer setzen ständig neue Tarnmechanismen ein. Wenn Sie glauben, der berühmte »XYZ-Scan«, für dessen monatliches Update Sie eine erkleckliche Summe investieren, sei ein wirksamer »Schutzschild gegen alle bekannten und unbekannten Viren«, können Sie eines bösen Erwachens sicher sein. Ein Virens Scanner ist also kein Freibrief dafür, daß Sie ohne Gefahr Programme beliebiger Herkunft kopieren und starten oder jedem Mitarbeiter erlauben können, nach Belieben am PC oder im Netz zu schalten und zu walten.

Welchen praktischen Sinn jedoch Viren-Cleaner haben sollen, bleibt unerfindlich. Offenbar sind sie bestens geeignet, die ökonomischen Interessen ihrer Hersteller und Anbieter zu befriedigen. Es ist doch bereits vom Ansatz her verfehlt, Programme mit dem Anspruch entwickeln zu wollen, aus allen denkbaren Programmstrukturen in allen denkbaren Systemumgebungen den Code einer wachsenden Zahl von Viren entfernen zu können. Solange es nur einige (Link-)Viren gab, die alle nach mehr oder weniger gleichem Schema einfache Com-Programme infizierten, mag dieses Ansinnen noch erfolgversprechend gewesen sein. Bei der aussterbenden Gattung überschreibender Viren kommt ohnehin jede Hilfe zu spät.

Mindestvoraussetzung für das Funktionieren eines Viren-Cleaners ist es, die Viren zu erkennen und sicher von anderen Viren und deren Mutanten zu unterscheiden. Wenn mal wieder ein Viren-Freak einige Bits in einem Virus geändert hat, rennt die ganze Crew der Virenforscher diesen Bits hinterher. Es ist doch für den Anwender völlig unerheblich, welcher Virus seinen PC oder sein Netzwerk lahmlegt. Er braucht Mittel und Wege, um mit großer Sicherheit zu diagnostizieren, ob eine Virusinfektion vorliegt oder nicht. Hat er festgestellt, daß sein Programm von einem Virus befallen wurde, muß er es umgehend löschen und neu von der Original- oder Sicherungsdiskette installieren.

● Erfolgreiche Virenabwehr

Es gibt in der Literatur einige Abwehrstrategien, die – wenn auch nicht ohne erheblichen Aufwand an Hard- und Software, Arbeitszeit und Organisation – mit relativ großer Sicherheit Schaden durch Viren und andere Sabotagepro-

gramme abwenden können. Diese Methoden zielen meist darauf hin, grundsätzlich kein Programm auf einem PC mit sensiblen Daten oder Anwendungen auszuführen, ohne daß die Software zuvor ausgiebig geprüft wurde.

Vorbereitend sollten Sie alle Original-Datenträger mit einem physischen Schreibschutz versehen. (Dies tun leider immer noch die wenigsten Hersteller.) Außerdem sollten Sie Ihre Disketten physisch kopieren und künftig an einem sicheren Ort aufbewahren. Softwarepakete, die keine Sicherheitskopien zulassen oder während der Installation eine Originaldiskette ohne physischen Schreibschutz verlangen, gehören endgültig ins Museum.

Zunächst prüfen Sie die schreibgeschützten Kopien der Original-Disketten auf Verseuchung durch bekannte Viren. Da Software heutzutage meist in gepackter Form auf den Installationsdisketten vorliegt, sollten Sie unmittelbar nach einer vollständigen Installation auf einem mit großer Sicherheit virenfreien PC dessen Festplatte erneut überprüfen. An die dabei eingesetzten Scanner sind hinsichtlich Qualität und Aktualität höchste Anforderungen zu stellen. Anschließend erproben Sie die neuen Programme ausgiebig auf einem PC, der mit einer definierten Softwareumgebung und geeigneten Überwachungsprogrammen ausgestattet ist.

Da in jüngster Vergangenheit mehrfach Virencode auch auf Originaldisketten großer Softwarehäuser gefunden wurde, ist es bei der angedeuteten Prozedur nicht mehr gerechtfertigt, zwischen mehr oder weniger vertrauenswürdigen Quellen von Software zu unterscheiden. Die Überprüfung muß sich notwendigerweise auf jede neu einzusetzende Software ausdehnen. Eine Diskriminierung von Public-Domain- oder Sharewareprogrammen, die in diesem Zusammenhang oft als besonders suspekt angesehen wurden, sollte damit ebenfalls der Vergangenheit angehören. Das heißt natürlich nicht, daß Sie nicht mehr auf die Seriosität Ihrer Lieferanten achten sollten. Kramkisten auf Trödelmärkten und die unter guten Freunden herumgereichten Diskettenboxen sind nach wie vor ein guter Nährboden für allerlei »Getier« und sollten selbst für den privaten PC-Anwender tabu sein.

Ein wichtiges Element bei einer detaillierten Zustandsprüfung eines PC-Systems ist der Kaltstart von einer bekanntenmaßen virenfreien, schreibgeschützten Diskette. Sie sollte außer den notwendigen DOS-Komponenten geeignete Analyse- und Reparaturprogramme sowie Sicherungskopien von Systemdaten enthalten. Das Grundprinzip ist es, mit Sicherheit auszuschließen, daß auch nur ein einziges Programm von der Festplatte des zu prüfenden Systems aufgerufen wird. Erst ein solches Vorgehen sichert einen definierten Ausgangspunkt des Systems für eine Überprüfung der Programme und Systemstrukturen und läßt residente Viren nicht zum Zug kommen.

Wenn Sie neue Software auf Aktivitäten testen, die durch eingeschleppte Viren ausgelöst worden sein können, bieten sich unter anderem folgende Maßnahmen an:

- Prüfen Sie die Integrität der Master-Boot- und der DOS-Boot-Records sowie die der Interrupt-Tabelle.
- Registrieren Sie Veränderungen in der Kette der Memory Control Blocks (MCB).
- Überprüfen Sie die Integrität der File Allocation Table (FAT).
- Stellen Sie fest, ob sich die Zahl der Bad Clusters verändert hat und ob Lost und Linked Clusters aufgetreten sind.
- Halten Sie Veränderungen der Verzeichnisstruktur fest.
- Prüfen Sie, ob Ihre Programmdateien in Ordnung sind (Länge, Datum, Uhrzeit, CRC-Kennungen).
- Registrieren Sie den Aufruf bestimmter »kritischer« Interrupts.
- Stellen Sie fest, ob residenter Programmcode installiert wird.
- Notieren Sie schreibende Zugriffe auf Programmdateien.
- Stellen Sie fest, wie häufig der transiente Teil des Kommandointerpreters »command.com« nachgeladen wird.

Viele der genannten Veränderungen und Aktivitäten sind im normalen Betrieb eines PC-Systems – zum Teil auch nur in bestimmten Situationen – durchaus legal. Es ist daher wenig sinnvoll, jede dieser Aktivitäten im Routinebetrieb ständig durch ein Kontrollprogramm überwachen zu lassen. Eine endlose Folge von Fehlalarmen wäre die Folge. Bedenken Sie auch, daß neuere Viren viele der als kritisch eingestuften Aktionen vermeiden.

Geeignete TSR-Programme, die Systemparameter und PC-Aktivitäten überwachen und verdächtige Aktionen melden, sind in einer Reihe von Anti-Viren-Tools enthalten. Doch auch MS-DOS-Programme wie Chkdsk und Mem sind geeignet, wichtige Systemparameter zu erfassen. Die mächtigen Hilfsprogramme PC-Tools und Norton Utilities sind geradezu Fundgruben für die anstehenden Prüfungen. Nachteilig ist jedoch, daß sie keine residente Überwachung oder keinen unmittelbaren Vergleich zweier Zustände des PC-Systems gestatten. Doch erst, wenn er viele Parameter und Rechneraktivitäten über eine angemessene Testphase hinweg bewertet, kann ein erfahrener PC-Anwender mit großer Sicherheit konstatieren, daß das System frei von Computerviren ist.

● Erfolgreich virenfrei bleiben

Haben Sie auf die Weise sichergestellt, daß das PC-System auch nach Übernahme neuer Programme mit großer Sicherheit virenfrei ist, bleibt Ihnen die Aufgabe diesen Zustand zu erhalten. Das wäre allenfalls erreichbar, wenn eine überschaubare Zahl von Anwendern mit Problembewußtsein und Motivation einen Solo-PC oder ein Netzwerk nutzt. Dabei ist vorausgesetzt, daß sie nicht gezwungen sind, jede Spiel- oder Demodiskette »mal eben schnell auszuprobieren« und Raubkopien für sie tabu sind.

Der private Anwender kann mit seinem PC vielleicht noch mit dem Risiko, das von Computerviren ausgeht, leben: Wenn die Briefe an Tante Sophie und die Datenbank der Briefmarkensammlung sogar noch als Backup existieren, ist keine Gefahr im Verzug. In einem großen Unternehmen mit Hunderten von PCs und einem weitverzweigten Netzwerk kann ohne Maßnahmen, die Zugang und zulässige Anwenderaktivitäten regeln, niemand die erforderliche Kontrolle über die Rechner und Netze ausüben und damit garantieren, daß nicht innerhalb der nächsten Stunden ein Virus zuschlägt. Dieses Risiko kann kein Unternehmen, das auf das Funktionieren der EDV angewiesen ist, auf sich nehmen.

Unumstritten ist die Tatsache, daß Schlendrian und Gedankenlosigkeit beim Umgang mit dem PC, den Datenträgern, Programmen und Daten sowie Bedienungsfehler den Viren täglich neue Chancen eröffnen. Bedienungsfehler scheinen geradezu eine der Grundbedingungen für die schnelle Ausbreitung vieler Viren zu sein.

Die Klasse der Boot-Viren zum Beispiel befällt den Boot- und/oder Master-Boot-Record und reproduziert sich bei Abarbeitung der dort enthaltenen Programmsequenzen. Das MS-DOS-Urladeprogramm – im Boot-Record einer jeden Diskette enthalten – wird einzig und allein beim Systemstart mit eingelegter Diskette ausgeführt. Bei einem PC mit Festplatte, die ja heutzutage selbst im privaten Bereich zum Standard gehört, dienen Disketten in der Regel lediglich dem Datenaustausch und der Installation neuer Software. So gesehen, ist der unbedachte Griff zur Reset-Taste oder die Betätigung der Tastenkombination [Strg-Alt-Entf] bei eingelegter Datendiskette eindeutig ein Bedienungsfehler. Spätestens wenn die Aufforderung »Kein Betriebssystem, Diskette wechseln und Taste drücken!« prompt befolgt wird, kann sich der eventuell enthaltene Virencode unbemerkt im System etablieren.

Es genügt aber nicht, den Zugang und die zugelassenen Aktivitäten eines Anwenders zu regeln und die Datenbestände auf externen Medien regelmäßig zu sichern. Es ist außerdem erforderlich, die Integrität des PC-Systems einschließlich der auf den Speichermedien abgelegten Systemberei-

che und Programmdateien ständig mit geeigneten Tools zu überwachen.

Für die Prüfung der Integrität von Programmdateien bieten sich Prüfsummenprogramme an – vornehmlich solche, die als TSR-Programm bereits vor dem Laden einer Programmdatei in den Arbeitsspeicher aktiv werden und nur unversehrten Programmen die weitere Abarbeitung gestatten. Vorbedingung aller Prüfsummenprogramme ist jedoch, daß die Soll-Werte der CRC-Kennungen unter Abwesenheit residenter Viren ermittelt wurden. Ferner muß sichergestellt sein, daß die zum Zugriff auf die prüfenden Programmdateien eingesetzten Systemfunktionen nicht durch residente Viren modifiziert wurden. Letzteres läuft unmittelbar auf die Sicherung der Integrität des Gesamtsystems hinaus.

Dazu ist es erforderlich, dessen Zustand zu einem definierten Zeitpunkt – zum Beispiel nach dem Systemstart und dem Laden aller standardmäßig benutzten Systemkomponenten – durch geeignete Programme mit einem umfassenden Satz von Parametern zu erfassen und mit entsprechenden Soll-Werten zu vergleichen. Die wichtigsten dieser Parameter wurden bereits genannt.

MS-DOS- und Hilfsprogramme können hier – wie beim Test neuer Software – hilfreich sein, sind für diesen konkreten Anwendungsfall jedoch nicht optimal ausgestattet. Dies zwingt dazu, eigene Tools zu entwickeln. Als Anregung und kleines Beispiel eines derartigen Programms wird im folgenden das Programm »chksys.asm« (Listing 1) vorgestellt. Wie Sie das Kompilat erhalten, entnehmen Sie den Listingzeilen 6 und 7. Sie können sie der Einfachheit halber auch in einer Stapeldatei ablaufen lassen:

```
TASM /z /m /jJUMPS /jLOCALS ChkSys
TLINK /t /x ChkSys
```

ChkSys bietet ein gewisses Maß an Sicherheit gegen das unbemerkte Eindringen von Viren. Darüber hinaus gestattet es – ähnlich wie auch Mirror und Rebuild aus PC-Tools oder MS-DOS 5.0 – die Boot- und Master-Boot-Records der Festplatte wiederherzustellen. Damit ersparen Sie sich im Fall einer Infektion mit Boot-Viren oder einer Zerstörung der verseuchten Bereiche durch einen Virus eine Low-Level-Formatierung der Festplatte und einen völligen Neuaufbau der Datenstrukturen. Absolute Sicherheit kann ChkSys – im Gegensatz zu den Werbeaussagen mancher Hersteller von Antivirenprogrammen – selbstverständlich nicht bieten. Erstens ist dies prinzipiell ausgeschlossen, zweitens ist der Satz der durch ChkSys überwachten Parameter entschieden zu klein.

ChkSys speichert beim ersten Aufruf die aktuellen Interrupt-Vektoren 0 bis 7Fhex, die aktuellen Parameter der Memory Control Blocks (MCB), mit denen MS-DOS den Arbeitsspeicher verwaltet, den Master-Boot- sowie den Boot-Record der aktiven DOS-Partition der ersten Festplatte des PC-Systems in einer Datei im Hauptverzeichnis des aktuellen Laufwerks. Bei jedem folgenden Aufruf prüft ChkSys diese Bereiche auf ihre Unversehrtheit, indem es sie unmittelbar mit den gespeicherten Werten in der Datei vergleicht. Hat ein Virus das System modifiziert, was sich in einer Änderung der Interrupt-Vektoren oder der MCB-Kette bemerkbar macht, gibt ChkSys eine Warnung aus. Da diese Modifikation des Systems auch auf einen residenten Virus zurückzuführen sein könnte, müssen Sie die Frage, ob die neuen Systemdaten gesichert werden sollen, mit Bedacht beantworten. Stimmen die Records auf der Festplatte nicht mit den Sicherungsdaten überein, können Sie entweder die Sicherungsdatei neu erzeugen oder den Inhalt der Datei in die entsprechenden Bereiche auf der Festplatte schreiben.

ChkSys eignet sich somit sowohl zur laufenden Kontrolle des Systemzustands des PC (Aufruf in der »autoexec.bat«) als auch – und das kann besonders nützlich sein – als Komponente der bereits dringend empfohlenen schreibgeschützten startfähigen Notfalldiskette.

(Dr. Christian Raabe/et)

```
[8P3] 1: ;Programm: chksys.asm
[FP6] 2: ;Funktion: Sichert wichtige PC-Parameter
[RP6] 3: ; Erneuter Aufruf vergleicht aktuelle
[XS4] 4: ; mit gespeicherten Werten.
[3R3] 5: ;Sprache: Borland-TASM
[1X5] 6: ;TASM /z /m /jJUMPS /jLOCALS ChkSys
[453] 7: ;TLINK /t /x ChkSys
[AN4] 8: ;Autor: Dr. Christian Raabe
[HS4] 9: ;(c)1992 DMV Widuch GmbH & Co.KG
[ZC3] 10: ;Macro-Definitionen
[W63] 11: ConOut MACRO Msg
[KY1] 12: lea dx,Msg
[OE1] 13: mov ah,9
[E61] 14: int 21h
[HP0] 15: ENDM
[VC3] 16: Request MACRO Msg
[8R1] 17: ConOut Msg
[RW1] 18: mov ah,0ch
[OE1] 19: mov al,1
[E61] 20: int 21h
[7W1] 21: and al,0dfh
[DB3] 22: cmp al,4ah;j oder J ?
[HP0] 23: ENDM
[DM2] 24: ; Code-Segment ---
[SD2] 25: Code SEGMENT
[8K3] 26: ASSUME cs:Code,ds:Code
[391] 27: ORG 100h
[VX2] 28: Entry: jmp Init
[RE1] 29: ; Daten ---
[M77] 30: Buffer1 LABEL BYTE ;Puffer für akt.Werte
[DU4] 31: MCB1 DB 200h DUP (?);MCB-Daten
[K86] 32: PartSec1 DB 200h DUP (?);Partition-Sekt
[8K5] 33: BootSec1 DB 200h DUP (?);Boot-Sektor
[9B7] 34: Buffer2 LABEL BYTE ;Puffer Speicher-Werte
[X06] 35: Intr2 DB 200h DUP (?);Interruptvekt.
[EU4] 36: MCB2 DB 200h DUP (?);MCB-Daten
[FB6] 37: PartSec2 DB 200h DUP (?);Partition-Sekt.
[9K5] 38: BootSec2 DB 200h DUP (?);Boot-Sektor
[D36] 39: PartTab DB 40h DUP (?);Partition-Tab.
[446] 40: FilNam DB '\CHKSYS.FIL',0 ;Dateiname
[WB4] 41: Handle DW ? ;Datei-Handle
[000] 42:
[Z16] 43: SystFlag DB 0 ;System-Daten schreiben
[UE6] 44: PartFlag DB 0 ;Partition-Dat.schreiben
[4N5] 45: BootFlag DB 0 ;Boot-Daten schreiben
[000] 46:
[MV3] 47: HeadHD DW ? ;Kopf und HD
[CK5] 48: CylSec DW ? ;Zylinder und Sektor
[000] 49:
[833] 50: StartMsg DB 0ah,0dh
[ZB2] 51: DB 'ChkSys 1.1 - '
[454] 52: DB '(C) 9/92 Christian Raabe'
[SC1] 53: DB 0ah,0dh
[RH4] 54: DB 'Kontrolliert wichtige '
[RS4] 55: DB 'Parameter des PC-Systems'
[6B2] 56: DB 0ah,0ah,0dh,'$'
[XW5] 57: IntrMsg DB 'Interrupt-Vektoren -$'
[3K4] 58: MCBMsg DB 'MCB-Kette -$'
[TM5] 59: PartMsg DB 'Partition-Sektor -$'
[VY4] 60: BootMsg DB 'Boot-Sektor -$'
[223] 61: NotMsg DB ' nicht-$'
[N82] 62: OkMsg DB ' ok'
[LC3] 63: CRLFMsg DB 0ah,0dh,'$'
[9U3] 64: WrtFilMsg DB ' nicht ok'
[SC1] 65: DB 0ah,0dh
[F92] 66: NewFilMsg DB 7
[3T2] 67: DB ' - neu'
[E94] 68: DB ' in Datei sichern ? : $'
[UD3] 69: WrtSecMsg DB 0ah,0dh,7
[DT2] 70: DB ' - auf'
[NU4] 71: DB ' HD zurückschreiben ? : $'
[LV3] 72: SecurMsg DB 0ah,0ah,0dh,7
[9X2] 73: DB ' ACHT'
[BS3] 74: DB 'UNG ! Falsche Werte'
[SC1] 75: DB 0ah,0dh
[T83] 76: DB ' könn'
[DN3] 77: DB 'en alle Daten auf HD'
[SC1] 78: DB 0ah,0dh
[513] 79: DB ' unzu'
[FK3] 80: DB 'gänglich machen !'
[T52] 81: DB 0ah,0ah,0dh,7
[AZ2] 82: DB ' - Sind'
[9H4] 83: DB ' Sie absolut sicher ? : $'
[AX2] 84: WrtMsg DB 0ah,0dh
[GA4] 85: DB 'Parameter neu in Datei!'
[CT1] 86: DB 0ah,0dh,'$'
[000] 87:
[YV2] 88: ErrMsg DB 0ah,0dh
[BT1] 89: DB 'Fehler !'
[302] 90: DB 0ah,0dh,7,'$'
[P13] 91: BadFilMsg DB 0ah,0dh
[3B5] 92: DB 'Sicherungsdatei zerstört!'
[302] 93: DB 0ah,0dh,7,'$'
[000] 94:
[AH2] 95: ;Datei öffnen ---
[XY5] 96: Main: lea dx,FilNam;ds:dx -> FilNam
[5G5] 97: mov ax,3d02h;Schreib-Lesezugrif
[Z52] 98: int 21h;OPEN
[5T1] 99: jc Create
```



```
[HM4] 100:      mov  Handle,ax ;Datei-Handle
[000] 101:
[KD2] 102: ;Datei lesen ----
[D04] 103:      mov  bx,ax ;Datei-Handle
[B55] 104:      lea  dx,Buffer2;ds:dx -> Buffer2
[SY4] 105:      mov  cx,900h ;2304 Byt.anfordern
[6X1] 106:      mov  ah,3fh
[U42] 107:      int  21h ;READ
[UH1] 108:      jc   Error
[HS4] 109:      cmp  ax,800h ;2048 Byt.gelesen?
[S31] 110:      je   MCB
[0Y1] 111:      jmp  BadFil
[000] 112:
[N05] 113: ;Datei erstellen und öffnen -
[8T5] 114: Create:  xor  cx,cx ;Datei-Attribut
[XW1] 115:      mov  ah,3ch
[ZF2] 116:      int  21h ;CREATE
[UH1] 117:      jc   Error
[HM4] 118:      mov  Handle,ax ;Datei-Handle
[2D5] 119:      mov  SystFlag,1;Sytem-Dat.schr.
[TF5] 120:      mov  PartFlag,1;Partit.-Dat.schr.
[Z24] 121:      mov  BootFlag,1;Boot-Dat.schr.
[000] 122:
[S33] 123: ;MCB-Kette lesen ----
[ZL2] 124: MCB:      mov  ah,52h
[V05] 125:      int  21h ;Get DOS InfoBlock(DIB)
[S75] 126:      mov  ax,es:[bx-2];Segm.des 1.MCB
[GS1] 127:      mov  es,ax
[535] 128:      xor  si,si ;es:si->1.MCB im RAM
[C24] 129:      lea  di,MCB1 ;ds:di -> MCB1
[000] 130:
[S44] 131: Chain:    mov  bx,es:[si+1]
[MV2] 132:      mov  cx,es:[si+3]
[K73] 133:      mov  [di],ax ;Segm.
[0K3] 134:      mov  [di+2],bx ;Owner
[XG4] 135:      mov  [di+4],cx ;Zahl allok.
[P35] 136:      cmp  BYTE PTR es:[si],'Z';Ende?
[3U1] 137:      je   ChkInt
[6P4] 138:      inc  ax;1 Paragraphen weiter
[D34] 139:      add  ax,cx ;nächster MCB
[FH5] 140:      mov  es,ax ;es:si -> nächster MCB
[XU3] 141:      add  di,6;6 Byt. weiter
[1P1] 142:      jmp  Chain
[000] 143:
[TN5] 144: ;Interruptvektoren vergleichen --
[PV6] 145: ChkInt:   cmp  SystFlag,1;Sytem-Dat.schr.?
[402] 146:      je   ReadPart
[HG2] 147:      ConOut IntrMsg
[AK4] 148:      lea  si,Intr2;ds:si -> Intr2
[4P1] 149:      xor  di,di
[XC5] 150:      mov  es,di ;es:di->Int.Vektor-Tab
[LZ4] 151:      mov  cx,100h;512 Byt.(Vekt.0-7fh)
[6Z1] 152:      repe cmpsw
[9H1] 153:      je   IntOk
[X82] 154:      ConOut NotMsg
[2D5] 155:      mov  SystFlag,1;Sytem-Dat.schr.
[000] 156:
[CK3] 157: IntOk:    ConOut OkMsg
[000] 158:
[VN5] 159: ;Daten der MCB-Kette vergleichen ----
[E62] 160:      ConOut MCBMsg
[8C1] 161:      push cs
[BZ3] 162:      pop  es;es auf Code-Segm.
[B54] 163:      lea  si,MCB2 ;ds:si -> MCB2
[G24] 164:      lea  di,MCB1 ;es:di -> MCB1
[V63] 165:      mov  cx,100h ;512 Byt.
[6Z1] 166:      repe cmpsw
[RF1] 167:      je   MCBok
[X82] 168:      ConOut NotMsg
[2D5] 169:      mov  SystFlag,1;Sytem-Dat.schr.
[000] 170:
[BF3] 171: MCBok:    ConOut OkMsg
[PD5] 172:      cmp  SystFlag,1;Sytem-Dat.schr.?
[L52] 173:      jne  ReadPart
[R43] 174:      Request NewFilMsg
[N51] 175:      je   Yes
[CN2] 176:      mov  SystFlag,0
[1A1] 177:      jmp  No
[000] 178:
[MC3] 179: Yes:      ConOut OkMsg
[W62] 180:      jmp  ReadPart
[000] 181:
[0N3] 182: No:       ConOut CRLfMsg
[000] 183:
[H84] 184: ;Partition-Sektor lesen -
[AS2] 185: ReadPart: push cs
[BZ3] 186:      pop  es;es auf Code-Segm.
[U45] 187:      mov  dx,80h;Kopf 0, Festplatte 1
[3V4] 188:      mov  cx,1;Zylinder 0, Sektor 1
[8E5] 189:      lea  bx,PartSec1;es:bx->PartSec1
[894] 190:      mov  ax,201h ;einen Sektor
[Z42] 191:      int  13h ;READ
[UH1] 192:      jc   Error
[000] 193:
[RF5] 194: ;Partition-Sektor vergleichen ---
[065] 195:      cmp  PartFlag,1;Part.-Dat.schr.?
[NZ1] 196:      je   ChkPart
[0G2] 197:      ConOut PartMsg
[PF5] 198:      lea  si,PartSec2 ;ds:si->PartSec2
```

```
[XC5] 199:      lea  di,PartSec1 ;es:di->PartSec1
[V63] 200:      mov  cx,100h ;512 Byt.
[6Z1] 201:      repe cmpsw
[AU1] 202:      je   PartOk
[463] 203:      Request WrtFilMsg
[402] 204:      je   MarkPart
[E53] 205:      Request WrtSecMsg
[G12] 206:      jne  NoPart
[LZ2] 207:      Request SecurMsg
[G12] 208:      jne  NoPart
[000] 209:
[D47] 210: ;Partition-Sektor auf HD zurückschreiben-
[U45] 211:      mov  dx,80h;Kopf 0, Festplatte 1
[3Y4] 212:      mov  cx,1;Zylinder 0, Sektor 1
[A55] 213:      lea  bx,PartSec2;es:bx->PartSec2
[A94] 214:      mov  ax,301h ;einen Sektor
[MG2] 215:      int  13h ;WRITE
[UH1] 216:      jc   Error
[212] 217:      jmp  PartOk
[000] 218:
[KR6] 219: MarkPart: mov  PartFlag,1;Part.-Dat.schr.
[000] 220:
[LP3] 221: PartOk:   ConOut OkMsg
[D62] 222:      jmp  ChkPart
[000] 223:
[L44] 224: NoPart:   ConOut CRLfMsg
[000] 225:
[P44] 226: ;Part.-Sektor auswerten -
[L76] 227: ChkPart:  lea  si,PartSec1+1beh;ds:si->
[805] 228:      lea  di,PartTab;es:di->PartTab
[8X1] 229:      mov  cx,20h
[RZ1] 230:      repe movsw
[000] 231:
[SM6] 232: FindBoot: lea  bx,PartTab;Beginn PartTab
[WC4] 233:      mov  cx,4;4 Part.s möglich
[000] 234:
[LS6] 235: Next:     cmp  BYTE PTR [bx],80h ;Boot-Part.?
[F02] 236:      je   ReadBoot
[ZT1] 237:      add  bx,10h
[FK1] 238:      loop Next
[SS1] 239:      jmp  Error
[000] 240:
[DD3] 241: ;Boot-Sektor lesen --
[CM3] 242: ReadBoot: mov  dx,[bx]
[9L4] 243:      ;80h = Kopf 0 und Festplatte 1
[TV4] 244:      mov  cx,[bx+2] ;Zyl.+Sektor
[Z15] 245:      mov  HeadHD,dx ;Pos.Boot-Sekt.
[LF2] 246:      mov  CylSec,cx
[A55] 247:      lea  bx,BootSec1;es:bx->PartSec1
[894] 248:      mov  ax,201h ;einen Sektor
[Z42] 249:      int  13h ;READ
[UH1] 250:      jc   Error
[000] 251:
[0R4] 252: ;Boot-Sektor vergleichen ----
[C55] 253:      cmp  BootFlag,1;Boot-Dat.schr.?
[VV1] 254:      je   WrtFil
[5G2] 255:      ConOut BootMsg
[V63] 256:      mov  cx,100h ;512 Byt.
[RE5] 257:      lea  si,BootSec1 ;es:di->BootSec1
[PE5] 258:      lea  di,BootSec2 ;ds:si->BootSec2
[6Z1] 259:      repe cmpsw
[NU1] 260:      je   BootOk
[463] 261:      Request WrtFilMsg
[F02] 262:      je   MarkBoot
[E53] 263:      Request WrtSecMsg
[N12] 264:      jne  NoBoot
[LZ2] 265:      Request SecurMsg
[N12] 266:      jne  NoBoot
[000] 267:
[ZF6] 268: ;Boot-Sektor auf HD zurückschreiben -
[K95] 269:      mov  dx,HeadHD ;Posit Boot-Sekt
[VF2] 270:      mov  cx,CylSec
[VG5] 271:      lea  bx,BootSec2 ;es:bx->BootSec2
[A94] 272:      mov  ax,301h ;einen Sektor
[MG2] 273:      int  13h ;WRITE
[UH1] 274:      jc   Error
[D12] 275:      jmp  BootOk
[000] 276:
[LH6] 277: MarkBoot: mov  BootFlag,1;Boot-Dat.schr.
[000] 278:
[NP3] 279: BootOk:   ConOut OkMsg
[L22] 280:      jmp  WrtFil
[000] 281:
[H24] 282: NoBoot:   ConOut CRLfMsg
[000] 283:
[NR3] 284: ;Schreiben der Datei ----
[V26] 285: WrtFil:   mov  bx,Handle ;Datei-Handle
[FD5] 286:      cmp  SystFlag,1;Sytem-Dat.schr.?
[N12] 287:      je   WrtSyst
[065] 288:      cmp  PartFlag,1;Part.-Dat.schr.?
[321] 289:      je   WrtPart
[C55] 290:      cmp  BootFlag,1;Boot-Dat.schr.?
[5Z1] 291:      je   WrtBoot
[ST1] 292:      jmp  NoWrt
[000] 293:
```

Listing 1. »chksys.asm« bietet Systemschutz vor ungebetenen Software-Gästen wie Viren, Trojanischen Pferden und Würmern.


```
[M84] 294: ;Zeiger auf Dateianfang -
[F93] 295: WrtSyst: xor dx,dx
[8T1] 296: xor cx,cx
[C02] 297: mov ax,4200h
[7E2] 298: int 21h ;LSEEK
[UH1] 299: jc Error
[000] 300:
[LD6] 301: ;Interruptvektor-Tabelle schreiben --
[DT1] 302: xor dx,dx
[295] 303: mov ds,dx ;ds:dx->Int.Vekt-Tab.
[X63] 304: mov cx,200h ;512 Byt.
[6V1] 305: mov ah,40h
[FG2] 306: int 21h ;WRITE
[UH1] 307: jc Error
[000] 308:
[2U2] 309: ;MCB-Dat.schr. ----
[8C1] 310: push cs
[H25] 311: pop ds;ds wieder auf Code-Segm.
[LZ3] 312: lea dx,MCB1 ;ds:dx->MCB1
[X63] 313: mov cx,200h ;512 Byt.
[6V1] 314: mov ah,40h
[FG2] 315: int 21h ;WRITE
[UH1] 316: jc Error
[8C1] 317: push cs
[H25] 318: pop ds;ds wieder auf Code-Segm.
[065] 319: cmp PartFlag,1;Part.-Dat.schr.?
[K82] 320: jne WrtBoot?
[000] 321:
[SW5] 322: ;Zeiger auf Anfang der Part.-Daten --
[5D3] 323: WrtPart: mov dx,400h
[8T1] 324: xor cx,cx
[C02] 325: mov ax,4200h
[7E2] 326: int 21h ;LSEEK
[UH1] 327: jc Error
[000] 328:
[YX2] 329: ;Part.-Dat.schr. --
[FE5] 330: lea dx,PartSec1;ds:dx->PartSec1
[X63] 331: mov cx,200h ;512 Byt.
[6V1] 332: mov ah,40h
[FG2] 333: int 21h ;WRITE
[UH1] 334: jc Error
[000] 335:
[UP6] 336: WrtBoot?: cmp BootFlag,1;Boot-Dat.schr.?
[5X1] 337: jne FilEnd
[000] 338:
[UX5] 339: ;Zeiger auf Anfang der Boot-Daten ---
[KD3] 340: WrtBoot: mov dx,600h
[8T1] 341: xor cx,cx
[C02] 342: mov ax,4200h
[7E2] 343: int 21h ;LSEEK
[UH1] 344: jc Error
[000] 345:
[VZ2] 346: ;Boot-Dat.schr. ---
[LE5] 347: lea dx,BootSec1;ds:dx->BootSec1
[X63] 348: mov cx,200h ;512 Byt.
[6V1] 349: mov ah,40h
[FG2] 350: int 21h ;WRITE
[UH1] 351: jc Error
[022] 352: jmp WrtEnd
[000] 353:
[124] 354: ;Zeiger auf Dateiende ---
[V53] 355: FilEnd: mov dx,800h
[8T1] 356: xor cx,cx
[C02] 357: mov ax,4200h
[7E2] 358: int 21h ;LSEEK
[UH1] 359: jc Error
[000] 360:
[534] 361: WrtEnd: ConOut WrtMsg
[000] 362:
[PY4] 363: NoWrt: xor cl,cl ;kein Fehler
[SP1] 364: jmp Close
[000] 365:
[244] 366: ;Datei nicht akzeptiert -
[M74] 367: BadFil: ConOut BadFilMsg
[Y83] 368: mov cl,0ffh ;Fehler
[000] 369:
[UH3] 370: ;Datei schließen ----
[PV2] 371: Close: mov ah,3eh
[2E2] 372: int 21h ;CLOSE
[AW1] 373: jnc Cancel
[000] 374:
[BR2] 375: ;Fehlermeldung --
[DW3] 376: Error: ConOut ErrMsg
[7W1] 377: mov dl,80h
[EN1] 378: xor ah,ah
[5V2] 379: int 13h ;HD-Reset
[000] 380:
[M44] 381: ErrOut: ConOut CRLFMsg
[Y83] 382: mov cl,0ffh ;Fehler
[000] 383:
[WP2] 384: ;Programmende ---
[LK5] 385: Cancel: mov al,cl ;Fehlermeldung
[ZW1] 386: mov ah,4ch
[HE2] 387: int 21h ;CANCEL
[000] 388:
[6T6] 389: ;Initialisierung mit Speicherfreigabe -
[L04] 390: Init: ConOut StartMsg
[LC5] 391: lea bx,Ende;belegten Speichers
```

```
[CE1] 392: mov cl,4
[2R1] 393: shr bx,cl
[2U5] 394: inc bx;Zahl Paragr.für Programm
[TW1] 395: mov ah,4ah
[4X2] 396: int 21h ;SETBLOCK
[UH1] 397: jc Error
[4D5] 398: lea sp,Ende;neuer Stack-Pointer
[BF1] 399: jmp Main
[000] 400:
[7U3] 401: ;256 Byte Stack ab Init -
[162] 402: ORG Init+100h
[000] 403:
[8S4] 404: ;Ende des belegten Speichers -
[7F3] 405: Ende EQU THIS BYTE
[000] 406:
[BW1] 407: Code ENDS
[ER1] 408: END Entry
```

Listing 1. »chksys.asm« bietet Systemschutz vor ungebetenen Software-Gästen wie Viren, Trojanischen Pferden und Würmern.

Paßwort in der »config.sys«

Mit »controll.asm« (Listing 2) können Sie ein Paßwort bereits in der »config.sys« abfragen, zu einem Zeitpunkt also, zu dem Sie die Programmausführung noch nicht mit [Strg-Esc] unterbrechen können. »Controll« bedient sich hierzu eines Tricks: Es täuscht MS-DOS vor, daß es sich bei dem Programm um einen Gerätetreiber handelt. Doch wird von diesem Treiber nur die Initialisierungsfunktion unterstützt, die beim Laden des Treibers aufgerufen wird. Sie sorgt dafür, daß ein Paßwort eingegeben wird, und entfernt den Treiber danach aus dem Speicher. Dies ist deshalb durchführbar, weil MS-DOS als Rückgabeparameter von dieser Funktion unter anderem die Größe des von dem Treiber belegten Speichers erwartet. Hier wird nun einfach der Wert 0 übergeben. Daher wird der Speicher, den der Treiber nun wirklich belegt hat, früher oder später von einem anderen Programm überschrieben.

»controll.sys« unterscheidet zwischen Groß- und Kleinschreibung. Das Paßwort muß acht Zeichen enthalten. Es ist im Quellcode unter der Variablen »CW« abgelegt. Es lautet derzeit »PASSWORD«. Das Programm aktivieren Sie durch folgenden Eintrag in der »config.sys«:

```
device=Laufwerk:\Verzeichnis\controll.sys
```

Laufwerk und Verzeichnis passen Sie bitte Ihren Gegebenheiten an. Sie assemblieren »controll.asm« mit MASM 6.0. Dazu generieren Sie »controll.sys« nach dem Linklauf zuerst als Exe-Programm ohne Stack-Segment. Danach konvertieren Sie es mit »exe2bin.exe« in eine Datei mit der Endung »sys«:

```
exe2bin.exe controll.exe controll.sys
```

Da MASM 6.0 jedoch auch die Assemblierung zu Com-Programmen unterstützt, können Sie, wenn Sie dies Verfahren vorziehen, auch »controll.com« anlegen. Anschließend ändern Sie die Dateinamenserweiterung in »sys«.

(Tobias Zibold/et)

```
[P44] 1: ;Programm: controll.asm
[U76] 2: ;Funktion: Passwort in der config.sys
[U75] 3: ;Sprache: MASM/LINK Version 6.0
[LV5] 4: ;exe2bin controll.exe controll.sys
[0H3] 5: ;Autor: Tobias Zibold
[HS4] 6: ;(c)1992 DMV Widuch GmbH & Co.KG
[H22] 7: code segment
[8K3] 8: assume cs:code,ds:code
[GT0] 9: org 0
[241] 10: sa equ 3
[L71] 11: en equ 14
[501] 12: dw -1,-1
[LE2] 13: dw 10100000000000000000
[H21] 14: dw offset sr
[BY1] 15: dw offset it
[8K2] 16: cw db 'PASSWORD'
```



```
[3L4] 17: m1 db 13,10,13,10,'Passwort : ','$'
[4G7] 18: m2 db 13,10,13,10,'Das Passwort war falsch.
      ',13,10,'$'
[241] 19: cb db 9,?
[5L1] 20: db 18 dup (?)
[L82] 21: dp label dword
[321] 22: do dw ?
[B11] 23: dg dw ?
[ØV1] 24: sr proc far
[C12] 25: mov cs:do,bx
[312] 26: mov cs:dg,es
[3PØ] 27: ret
[981] 28: sr endp
[KX1] 29: it proc far
[PC1] 30: push ax
[TC1] 31: push bx
[WC1] 32: push cx
[ZC1] 33: push dx
[3B1] 34: push di
[GC1] 35: push si
[TB1] 36: push bp
[BC1] 37: push ds
[EC1] 38: push es
[ZXØ] 39: pushf
[8C1] 40: push cs
[E31] 41: pop ds
[FB3] 42: les di,dword ptr dp
[9G1] 43: call init
[SM1] 44: or ax,Ø1ØØh
[1V2] 45: mov es:[di+sa],ax
[LSØ] 46: popf
[G31] 47: pop es
[E31] 48: pop ds
[131] 49: pop bp
[E31] 50: pop si
[G21] 51: pop di
[X31] 52: pop dx
[V31] 53: pop cx
[T31] 54: pop bx
[R31] 55: pop ax
[3PØ] 56: ret
[V71] 57: it endp
[PC2] 58: init proc near
[EC1] 59: push es
[3B1] 60: push di
[AR2] 61: mov dx,offset m1
[3M1] 62: mov ah,Ø9h
[381] 63: int 21h
[LP2] 64: mov dx,offset cb
[LM1] 65: mov ah,Øah
[381] 66: int 21h
[NB1] 67: mov cx,8
[BC1] 68: push ds
[G31] 69: pop es
[TP2] 70: mov si,offset cw
[KW2] 71: mov di,offset cb+2
[HR1] 72: repe cmpsb
[7D1] 73: je code_r
[BR2] 74: mov dx,offset m2
[3M1] 75: mov ah,Ø9h
[381] 76: int 21h
[T73] 77: code_w: jmp code_w
[ME2] 78: code_r: pop di
[G31] 79: pop es
[2W4] 80: mov word ptr es:[di+en],ØØh
[DU4] 81: mov word ptr es:[di+en+2],cs
[GH1] 82: xor ax,ax
[3PØ] 83: ret
[ZG1] 84: init endp
[TF1] 85: code ends
[4MØ] 86: end
```

Listing 2. Mit »controll.asm« erhalten Sie einen Treiber in der »config.sys« zur Paßwortabfrage.

chenden CRTC-Registern (Vertical Total, Overflow, Vertical Retrace Start, Vertical Retrace End, Vertical Display Enable End, Start Vertical Blank und End Vertical Blank). Danach wird im Miscellaneous-Output-Register die Dot-Clock-Rate von 25 auf 28 MHz umgeschaltet. Hierdurch wird die VGA-Spezifikation um etwa 10 Prozent überschritten, so daß zwar nicht alle VGA-Festfrequenz-, jedoch die meisten Multisync-Monitore die erhöhte Bildwiederholfrequenz verarbeiten können. Eventuell sollten Sie nach der Umprogrammierung das Bild am Monitor neu positionieren. Falls sich kein stehendes Bild einstellt, schalten Sie sofort wieder auf normale Betriebsart um, um eine Beschädigung des Monitors zu vermeiden. Sie kompilieren »hz72.asm« mit TASM und binden es mit Tlink/t zur Com-Datei.

(Ingo Spitzcok von Brisinski/et)

```
[Ø73] 1: ;Programm: hz72.asm
[WK5] 2: ;Funktion: Schnelle Bildfrequenz
[RV3] 3: ;Sprache: TASM, TLINK /t
[976] 4: ;Autor: Ingo Spitzcok von Brisinski
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[PC3] 6: code segment public
[8K3] 7: assume cs:code,ds:code
[3A1] 8: org Ø1ØØh
[682] 9: w equ word ptr
[883] 10: dos macro func,param
[PY1] 11: ifnb <param>
[7A3] 12: mov ax,func*256+param
[ZRØ] 13: else
[FZ1] 14: mov ah,func
[TUØ] 15: endif
[381] 16: int 21h
[HPØ] 17: endm
[E21] 18: start:
[RF1] 19: jmp inst
[6SØ] 20: db 8,8
[KS4] 21: rgv db 6h,Øedh,7h,1fh,1Øh,Ødfh,11h
[3M4] 22: db 43h,12h,Ødfh,15h,Ødfh,16h,ØeØh
[722] 23: n1Ø proc near
[781] 24: cmp ah,Ø
[X71] 25: jne sk1
[YH2] 26: mov cs:svax,ax
[5MØ] 27: sk1:
[ZXØ] 28: pushf
[1VØ] 29: db 9ah
[S91] 30: sv1Ø dd Ø
[PC1] 31: push ax
[TC1] 32: push bx
[WC1] 33: push cx
[ZC1] 34: push dx
[WF2] 35: mov ax,cs:svax
[KL1] 36: cmp ax,11h
[R71] 37: je crgv
[ML1] 38: cmp ax,12h
[R71] 39: je crgv
[VM1] 40: cmp ax,2eh
[TB1] 41: jne skp
[2VØ] 42: crgv:
[PX1] 43: mov dx,3d4h
[ØM1] 44: mov al,11h
[6H1] 45: out dx,al
[S11] 46: inc dx
[161] 47: in al,dx
[DK1] 48: and al,7fh
[6H1] 49: out dx,al
[WX2] 50: mov bx,offset rgv
[ØWØ] 51: rrgv:
[LØ1] 52: dec dx
[1H2] 53: mov al,cs:[bx]
[6H1] 54: out dx,al
[M11] 55: inc bx
[S11] 56: inc dx
[1H2] 57: mov al,cs:[bx]
[6H1] 58: out dx,al
[M11] 59: inc bx
[DU1] 60: cmp al,ØeØh
[YD1] 61: jne rrgv
[EX1] 62: mov dx,3c2h
[MW1] 63: mov al,Øe7h
[6H1] 64: out dx,al
[1D2] 65: mov cs:svax,-1
[1SØ] 66: skp:
[X31] 67: pop dx
[V31] 68: pop cx
[T31] 69: pop bx
[R31] 70: pop ax
[GRØ] 71: iret
[TL1] 72: svax dw -1
[371] 73: n1Ø endp
```

Listing 3. Mit »72hz.asm« schalten Sie auch bei 480 Bildpunkten die VGA-Grafikkarten in einem flimmerfreien Modus.

Ergonomische Bildfrequenz auch bei 480 Zeilen

Der VGA-Standard sieht für Videomodi mit einer vertikalen Auflösung von 480 Bildpunkten eine vertikale Bildwiederholfrequenz von 60 Hz vor, so daß Flimmerfreiheit nicht garantiert ist. Oft verwenden auch SVGA-Karten in diesen Modi nur 60 Hz. Das residente Programm »hz72.asm« (Listing 3) gestattet auf Standard-VGA- und registerkompatiblen SVGA-Karten die Darstellung dieser Modi mit ergonomischen 72 Hz, indem es die Grafikkartenregister umprogrammiert.

Zunächst verringert Hz72 die Gesamtzahl der Zeilen und den vertikalen Overscan durch Änderungen in den entspre-


```
[GW0] 74: inst:
[4U1] 75: dos 35h,10h
[322] 76: mov w svl0,bx
[KF2] 77: mov w svl0+2,es
[BG1] 78: lea dx,nl0
[2U1] 79: dos 25h,10h
[XM1] 80: mov ah,0fh
[Y71] 81: int 10h
[9E1] 82: mov ah,0h
[Y71] 83: int 10h
[TZ1] 84: lea dx,inst
[N81] 85: int 27h
[TF1] 86: code ends
[4K1] 87: end start
```

Listing 3. Mit »72hz.asm« schalten Sie auch bei 480 Bildpunkten die VGA-Grafikkarten in einem flimmerfreien Modus.

Sichern des CMOS-RAM

Voraussetzung für »chkmos.asm« (Listing 4) ist eine MS-DOS Version ab 3.0, ein AT mit Echtzeituhr sowie eine Grafikkarte CGA, EGA, VGA auf Grafikmodus 2 oder 3, da der Bildschirm direkt angesteuert wird.

Vor dem Programmstart sollten Sie durch mehrmaliges Drücken der Leertaste den Cursor in die letzte Bildschirmzeile bringen, damit der Cursor während des Programmablaufs nicht störend irgendwo am Bildschirm blinkt.

Sobald Sie das Programm gestartet haben, erscheinen zwei Blöcke am Bildschirm (Bild 2). Im oberen ist der derzeitige Inhalt der CMOS-RAMs dargestellt. Der Inhalt wird ständig aktualisiert, wodurch sich ab und zu der Wert des Status-Register C ändern kann. Dieses Register signalisiert nämlich die Interrupt-Flags der Uhr.

Der Inhalt des Kopie-Blocks ist nach dem Start noch völlig leer, so daß Sie eventuell das CMOS-RAM ohne viel Tippen löschen können.

Mit den Cursortasten steuern Sie, welches Byte aus dem Kopie-Block Sie genauer betrachten wollen. Dieses Byte wird dann in verschiedenen Zahlensystemen angezeigt (hexadezimal, dezimal und binär). Außerdem wird das dazugehörige ASCII-Zeichen dargestellt.

Schalten Sie in den Eingabemodus, so blinkt die einzugebende Stelle in der Hex-Anzeige. Die Anzeige der verschiedenen Zahlensysteme bezieht sich jetzt nur auf den eingegebenen Hex-Wert. Durch [Enter] bestätigen Sie die Eingabe, und an der Stelle, an der sich der Cursor befand, übernehmen Sie sie in den Kopie-Block. Im Eingabemodus bleiben alle Tastenfunktionen erhalten. Die Tastaturbelegung entnehmen Sie Tabelle 1.

Um das CMOS-RAM zu speichern, berücksichtigen Sie noch folgende Punkte:

1. Die Datei »cmos ram« darf sich noch nicht im aktuellen Pfad des Laufwerks befinden.

Taste	Funktion
[Esc]	Programmende
[F1]	schaltet den Eingabemodus ein und aus.
[F3]	ändert die Farbe des Bildschirminhalts und löscht eventuelle Fehlanzeigen.
[F4]	zeigt und unterdrückt den vollen ASCII-Zeichensatz der Grafikkarte – wichtig für den Bildschirmausdruck.
[F5]	kopiert den aktuellen Inhalt des CMOS-RAM in den Kopie-Block hinein.
[F6]	errechnet die Prüfsumme des CMOS-RAMs der Uhr.
[F7]	lädt den Inhalt der Datei »cmos ram« in den Kopie-Block.
[Strg-F8]	schreibt den aktuellen Inhalt des CMOS-RAM auf Diskette in den aktuellen Pfad des aktuellen Laufwerks.
[Strg-F9]	schreibt den Inhalt des Kopie-Block ins CMOS-RAM ohne Berücksichtigung der Zeit.
[Strg-F10]	schreibt den Inhalt des Kopie-Blocks ins CMOS-RAM mit Berücksichtigung der Zeit.

Tabelle 1. Mit diesen Tastenkombinationen bedienen Sie »chkmos.com«.

2. Auf der Diskette müssen mindestens 64 Byte frei sein.

3. Die Diskette darf nicht schreibgeschützt sein.

Wenn Sie trotzdem versuchen, auf eine schreibgeschützte Diskette zu sichern, erhalten Sie eine Fehlermeldung aus der MS-DOS-Ebene heraus. Um die Fehlermeldung wieder

```
CMOS.RAM 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
00 54 22 37 4A 11 17 01 24 09 92 26 02 40 00 00 00 T"7J...$.aa.0G...
10 24 00 F0 00 41 00 02 00 0D 2F 00 D2 03 0E 00 00 $.=..AG.../..#...
20 00 00 00 23 00 00 00 00 00 00 00 00 00 00 00 00 ...#.....?
30 00 0D 19 00 00 00 00 0F 41 00 00 00 00 00 00 00 ...$....A.....

Kopie 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

HEX=[0] DEZ=[0] BIN=[00000000] ASCII=[.]
St.=[00]
```

Bild 2. »chkmos.com« läßt Sie in das CMOS-RAM blicken.

Fehlerstatus	Bedeutung
00	kein Fehler
02	Datei nicht gefunden
05	Diskette schreibgeschützt
50	Datei schon vorhanden

Tabelle 2. Diese Fehlermeldungen in den Diskettenoperationen liefert Ihnen »chkmos.com« in der Variablen »St«.

Wert	Bedeutung
00hex	Sekundenwert der Uhrzeit
01hex	Sekundenwert der Alarmzeit
02hex	Minutenwert der Uhrzeit
03hex	Minutenwert der Alarmzeit
04hex	Stundenwert der Uhrzeit
05hex	Stundenwert der Alarmzeit
06hex	Nummer des Wochentags
07hex	Nummer des Tages innerhalb des Monats
08hex	Nummer des Monats
09hex	Nummer des Jahres
0Ahex	Statusregister A
0Bhex	Statusregister B
0Chex	Statusregister C
0Dhex	Statusregister D
0Ehex	Diagnosestatus
0Fhex	Shutdown-Wert
10hex	Diskettentyp von Laufwerk A und B
11hex	reserviert
12hex	Festplattentyp von Laufwerk C und D
13hex	reserviert
14hex	Ausrüstungscode
15hex	Gesamtspeicher ohne Erweiterungsspeicher Low-Byte
16hex	Gesamtspeicher ohne Erweiterungsspeicher High-Byte
17hex	Erweiterungsspeicher Low-Byte
18hex	Erweiterungsspeicher High-Byte
19hex-2Dhex	reserviert
2Ehex-2Fhex	Checksumme des NVR (Non Volatile RAM)
30hex-31hex	reserviert
32hex	Jahrhundertnummer
33hex	Status-Flags
34hex-3Fhex	reserviert

Tabelle 3. Die Werte des CMOS-RAM haben einfache Bedeutungen.

vom Bildschirm verschwinden zu lassen, ändern Sie mit der F3-Taste die Farbe. Dadurch aktualisieren Sie den gesamten Bildschirminhalt. Den Fehlercode der letzten Diskettenoperation erhalten Sie in der Ausgabe des Status:

St.=[xx]

Die Bedeutungen der anfallenden Fehlermeldungen entnehmen Sie Tabelle 2.

Weil das Programm zum 1024-Mark-Wettbewerb eingesandt wurde und also kurz sein mußte, konnte der Autor verständlicherweise leider nicht alle Fehler berücksichtigen. So werden zwar Fehler beim Anlegen und Öffnen einer Datei, nicht aber beim Schreiben gemeldet. Es wird zum Beispiel kein Fehler registriert, wenn auf der Diskette zu wenig Platz für die Sicherungsdatei »cmos.ram« ist.

Falls es Sie stört, daß Sie nicht das aktuelle CMOS-RAM in die bereits vorhandene Sicherheitskopie schreiben können, müssen Sie nur in der Datei »chkmos.com« an der Speicherstelle »xxxx:0474« den Wert »5Bhex« durch »3Chex« ersetzen. Genauso ändern Sie auch die Farbe, in der das Programm startet, an der Speicherstelle »xxxx:04BE«. Dort vermindern Sie das Zeichenattribut um »1«.

Was die Werte des CMOS-RAM bedeuten, entnehmen Sie Tabelle 3.

Wie Sie das Kompilat aus dem Listing erhalten, weisen schon die Listingzeilen 7 bis 9 an. Sie arbeiten allerdings mit der besser lesbaren langen Version, die neben einer übersichtlicheren Gliederung, sich selbsterklärenden Sprungmarken auch Kommentare enthält.

(Alexander Nußbaumer/et)

```
[RD1] 54:          pop      ax
[000] 55:
[DE5] 56: ;Eingegebenes Zeichen auswerten
[A34] 57:          cmp      al,1bh      ; esc=ende
[BK1] 58:          jz       ende
[X65] 59: ;Eingabemodus ein- & ausschalten
[M74] 60:          cmp      ah,3bh      ; f1=eingabe
[RD2] 61:          jnz      eingabe1
[UK2] 62:          xor      dh,00100000b
[PW1] 63:          jmp      jump2
[000] 64:
[115] 65: ;Programm durch int21h beenden
[213] 66: ende:      mov      ax,4ch
[3K1] 67:          int      21h
[000] 68:
[EA4] 69: eingabe1: test     dh,00100000b
[562] 70:          jz       tasten1
[W46] 71: ;Eingabe quittieren durch [Return]
[434] 72:          cmp      al,0dh      ; return
[SD2] 73:          jnz      eingabe2
[TH4] 74:          mov      cl,byte ptr ds:[ram+bp],cl
[E65] 75:          mov      byte ptr ds:[ram+bp],cl
[RU5] 76: ;Prüfen auf zulässigen 'Hexwert'
[4Y4] 77: eingabe2: cmp      al,30h      ; <30 ?
[042] 78:          jc       tasten1
[7N3] 79:          cmp      al,3ah      ; <3a ?
[G52] 80:          jc       eingabe3
[5G2] 81:          and      al,11011111b
[9L3] 82:          cmp      al,41h      ; <41 ?
[042] 83:          jc       tasten1
[9M3] 84:          cmp      al,47h      ; >47 ?
[D92] 85:          jnc      tasten1
[9K5] 86: ;Eingegebenen 'Hexwert' umrechnen
[T83] 87: eingabe3: mov      ah,al
[NH4] 88:          mov      al,byte ptr ds:[bt]
[AC2] 89:          or       dh,00010000b
[U21] 90:          call     asc11
[YF2] 91:          and      dh,11101111b
[BL2] 92:          test     dh,01000000b
[P72] 93:          jz       eingabe4
[AT1] 94:          mov      cl,ah
[GC2] 95:          jmp      eingabe5
[S83] 96: eingabe4: mov      ch,ah
[FA3] 97: eingabe5: mov      ax,cx
[HX5] 98: ;Umrechnung von ASCII nach Hexwert
[DM1] 99:          mov      cl,4
[HW1] 100:         cmp      al,41h
[372] 101:         js       eingabe6
[TW1] 102:         sub      al,07h
[P93] 103: eingabe6: sub      al,30h
[5W1] 104:         cmp      ah,41h
[472] 105:         js       eingabe7
[DW1] 106:         sub      ah,07h
[F93] 107: eingabe7: sub      ah,30h
[ZR1] 108:         shl      al,cl
[VT1] 109:         shr      ax,cl
[VH4] 110:         mov      byte ptr ds:[bt],al
[TK2] 111:         xor      dh,01000000b
[PW1] 112:         jmp      jump2
[000] 113:
[DE5] 114: ;Eingegebenes Zeichen auswerten
[HA5] 115: tasten1: cmp      ah,3dh      ; farbe f3
[DC2] 116:          jnz      tasten2
[MW1] 117:          jmp      jump1
[YA5] 118: tasten2: cmp      ah,3eh      ; ascii f4
[HC2] 119:          jnz      tasten3
[VK2] 120:          xor      dh,00000001b
[975] 121: tasten3: cmp      ah,4bh      ; links
[NC2] 122:          jnz      tasten4
[R91] 123:          dec      bp
[3C5] 124: tasten4: cmp      ah,4dh      ; rechts
[TC2] 125:          jnz      tasten5
[WAI] 126:          inc      bp
[A25] 127: tasten5: cmp      ah,48h      ; hoch
[XC2] 128:          jnz      tasten6
[EPI] 129:          sub      bp,16
[2E5] 130: tasten6: cmp      ah,50h      ; runter
[LD2] 131:          jnz      tasten7
[ZL1] 132:          add      bp,16
[V05] 133: tasten7: and      bp,000000000111111b
[EE4] 134:          cmp      ah,3fh      ; kopieren f5
[5D2] 135:          jnz      tasten8
[7C2] 136:          or       dh,10000000b
[MP5] 137: tasten8: cmp      ah,40h      ; chksumme f6
[9D2] 138:          jnz      tasten9
[000] 139:
[PB3] 140: ;Checksum errechnen
[XF1] 141:          push     ax
[GS1] 142:          mov      cx,30
[SS1] 143:          mov      bx,16
[5X5] 144: check:      add      al,byte ptr ds:[ram+bx]
[MB1] 145:          inc      bx
[WW1] 146:          loop     check
[TR1] 147:          mov      bl,47
[875] 148:          mov      byte ptr ds:[ram+bx],al
[FA1] 149:          dec      bx
```

Listing 4. »chkmos.asm« gestattet Ihnen, das CMOS-RAM zu prüfen und zu speichern.

```
[4P3] 1: ;Programm: chkcmos.asm
[RN6] 2: ;Funktion: Prüfen+Ändern des CMOS-RAM
[BT2] 3: ;Sprache: MASM 5.1
[0W4] 4: ;Autor: Alexander Nussbaumer
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[000] 6:
[8F2] 7: ; masm chkcmos;
[WE2] 8: ; link chkcmos;
[665] 9: ; exe2bin chkcmos.exe chkcmos.com
[ZA3] 10: ; debug chkcmos.com
[HS0] 11: ; rcx
[UN0] 12: ; 0400
[KH0] 13: ; w
[UG0] 14: ; q
[000] 15:
[6R4] 16: program segment para 'code'
[AG1] 17:          org      100h
[TF5] 18:          assume   cs:program,ds:program
[VG5] 19:          assume   ss:program,es:program
[000] 20:
[G63] 21: start      proc      near
[000] 22:
[MU1] 23:          mov      bp,ax
[HP6] 24: ;Speicher ab [bt] mit 130 Nullen füllen
[M22] 25:          mov      di,offset bt
[0W1] 26:          mov      cl,130
[8Y1] 27:          rep      stosb
[KF7] 28: ;Bildsegmentadresse ins Extrasegment laden
[562] 29:          mov      ah,0b8h
[3V1] 30:          mov      es,ax
[000] 31:
[6R3] 32: ;Bildschirm löschen
[PZ2] 33: jump1:      mov      di,0h
[3Z1] 34:          mov      ax,20h
[B62] 35:          mov      cx,0780h
[KZ1] 36:          rep      stosw
[SH3] 37: ;Farbe ändern (farbe + 1)
[NG4] 38:          inc      byte ptr ds:[farbe]
[3N4] 39:          and      byte ptr ds:[farbe],7
[0G2] 40:          and      dh,1111011b
[722] 41:          call     mult11
[ZE3] 42:          call     textel
[8C2] 43:          or       dh,00000100b
[000] 44:
[7T6] 45: ;Prüfen ob eine Zeicheneingabe erfolgte
[LX1] 46:          mov      ah,01h
[GK1] 47:          int      16h
[XR1] 48:          jz       jump2
[000] 49:
[CC7] 50: ;Eingabepuffer durch Int16h löschen lassen
[XF1] 51:          push     ax
[TL1] 52:          mov      ah,0
[GK1] 53:          int      16h
```



```
[982] 150: xor al,0ffh
[875] 151: mov byte ptr ds:[ram+bx],al
[RD1] 152: pop ax
[000] 153:
[DE5] 154: ;Eingegebenes Zeichen auswerten
[4B5] 155: tasten9: cmp ah,66h ; strg + f9
[7S4] 156: jnz tasten10 ; a.o. zeit
[AC2] 157: or dh,00010000b
[4K2] 158: call cmosschr1
[134] 159: tasten10: and dh,11101111b
[204] 160: cmp ah,67h ; strg + f10
[9R4] 161: jnz tasten11 ; a.m. zeit
[4K2] 162: call cmosschr1
[F03] 163: tasten11: push dx
[VZ3] 164: cmp ah,65h ; strg + f8
[EP4] 165: jnz tasten12 ; schreiben
[S82] 166: call schrbn1
[3D5] 167: tasten12: cmp ah,41h ; laden f7
[WD2] 168: jnz tasten13
[C12] 169: call lesen1
[WU2] 170: tasten13: pop dx
[PW1] 171: jmp jump2
[000] 172:
[XX6] 173: ;Hexwert in ASCII umrechnen und ausgeben
[1V2] 174: ascii1: push ax
[ZF1] 175: push bx
[0T1] 176: xor ah,ah
[1G1] 177: push cx
[DM1] 178: mov cl,4
[PT1] 179: rol ax,cl
[AT1] 180: ror al,cl
[VD1] 181: pop cx
[DX1] 182: cmp al,0ah
[5V1] 183: js ascii2
[AU1] 184: add al,07h
[Y23] 185: ascii2: add al,30h
[1X1] 186: cmp ah,0ah
[8V1] 187: js ascii3
[YT1] 188: add ah,07h
[T23] 189: ascii3: add ah,30h
[167] 190: ;Prüfen ob Zeichen ausgeben o. Wert zurück
[DL2] 191: test dh,00010000b
[PW1] 192: jz ascii4
[EV1] 193: mov cx,ax
[W02] 194: jmp ascii5
[173] 195: ascii4: xchg al,ah
[NX0] 196: stosb
[XB2] 197: call cursor1
[NX0] 198: stosb
[XB2] 199: call cursor1
[3S1] 200: xchg ah,al
[ZV0] 201: stosw
[TN2] 202: ascii5: pop bx
[RD1] 203: pop ax
[3P0] 204: ret
[000] 205:
[0C6] 206: ;blinkenden Cursor steuern bei Eingabe
[B36] 207: cursor1: mov al,byte ptr ds:[farbe]
[CL2] 208: test dh,00100000b
[Y72] 209: jz cursor2
[AL2] 210: test dh,00001000b
[Y72] 211: jz cursor2
[TK2] 212: xor dh,01000000b
[BL2] 213: test dh,01000000b
[Y72] 214: jz cursor2
[UR1] 215: or al,80h
[E52] 216: cursor2: ret
[000] 217:
[BP6] 218: ;Bildposition aus x & y (BH&BL) errechnen
[UR2] 219: posi: push ax
[ZF1] 220: push bx
[1G1] 221: push cx
[3G1] 222: push dx
[BT1] 223: mov cl,bh
[3T1] 224: xor bh,bh
[6T1] 225: xor ch,ch
[WD3] 226: mov ax,80 ;bx*80=ax
[ND1] 227: mul bx
[HW3] 228: add ax,cx ;(ax+cx)*2=di
[TL1] 229: shl ax,1
[3U1] 230: mov di,ax
[XD1] 231: pop dx
[VD1] 232: pop cx
[TD1] 233: pop bx
[RD1] 234: pop ax
[3P0] 235: ret
[000] 236:
[5N2] 237: ;Texte ausgeben
[115] 238: text1: mov si,offset tcmos
[CE2] 239: text2: lodsw
[CV1] 240: mov bx,ax
[7T1] 241: call posi
[706] 242: text3: mov ah,byte ptr ds:[farbe]
[PD1] 243: or ah,8
[CV0] 244: lodsb
[LK1] 245: cmp al,0
[DY1] 246: jz text2
[H62] 247: cmp al,0ffh
```

```
[SX1] 248: jz wert1
[DW1] 249: cmp al,20h
[EW1] 250: jc texte4
[ZV0] 251: stosw
[932] 252: jmp texte3
[663] 253: texte4: xor ah,ah
[2P1] 254: add di,ax
[932] 255: jmp texte3
[000] 256:
[KF4] 257: ;Werte umrechnen & ausgeben
[MT4] 258: wert1: mov si,offset thex
[PW0] 259: lodsw
[CV1] 260: mov bx,ax
[ZF1] 261: add bh,5
[7T1] 262: call posi
[F75] 263: mov al,byte ptr ds:[ram+bp]
[CL2] 264: test dh,00100000b
[VX1] 265: jz wert2
[NH4] 266: mov al,byte ptr ds:[bt]
[3Y2] 267: ;Hexwert ausgeben
[X53] 268: wert2: mov cl,al
[7C2] 269: or dh,00001000b
[U21] 270: call ascii1
[1G2] 271: and dh,11110111b
[KL1] 272: add di,18
[LX4] 273: mov ch,byte ptr ds:[farbe]
[GT1] 274: mov al,cl
[6G5] 275: ;Dezimalwert errechnen & ausgeben
[0T1] 276: xor ah,ah
[WV1] 277: mov dl,100
[7C1] 278: div dl
[LK1] 279: cmp al,0
[242] 280: jnz wert3
[7C2] 281: or dh,00001000b
[162] 282: mov al,0f0h
[253] 283: wert3: add al,30h
[XR1] 284: xchg ah,ch
[ZV0] 285: stosw
[XR1] 286: xchg ah,ch
[6T1] 287: mov al,ah
[0T1] 288: xor ah,ah
[CR1] 289: mov dl,10
[7C1] 290: div dl
[LK1] 291: cmp al,0
[542] 292: jnz wert4
[AL2] 293: test dh,00001000b
[1Y1] 294: jz wert4
[162] 295: mov al,0f0h
[553] 296: wert4: add al,30h
[1G2] 297: and dh,11110111b
[XR1] 298: xchg ah,ch
[ZV0] 299: stosw
[XR1] 300: xchg ah,ch
[6T1] 301: mov al,ah
[0T1] 302: add al,30h
[XR1] 303: xchg ah,ch
[ZV0] 304: stosw
[KL1] 305: add di,18
[KT1] 306: mov bl,cl
[NC5] 307: ;Binärwert errechnen & ausgeben
[6T1] 308: mov bh,cl
[NZ1] 309: mov cx,08h
[363] 310: wert5: xor al,al
[6K1] 311: rcl bl,1
[F12] 312: jnc wert6
[EA1] 313: inc al
[B53] 314: wert6: add al,30h
[ZV0] 315: stosw
[L42] 316: loop wert5
[7L1] 317: add di,22
[0Y3] 318: ;ASCII-Zeichen ausgeben
[7T1] 319: mov al,bh
[MA2] 320: call akontrl
[ZV0] 321: stosw
[000] 322:
[UH7] 323: ;Fehlercode der letzten Diskoperation ausg.
[YM3] 324: fehler: mov bx,1311h
[7T1] 325: call posi
[FX4] 326: mov ah,byte ptr ds:[farbe]
[RX4] 327: mov al,byte ptr ds:[fcode]
[U21] 328: call ascii1
[000] 329:
[7W6] 330: ;Modulsteuerung für CMOSlesen & adrinh
[KS4] 331: multil: mov di,offset cmos
[AL2] 332: test dh,10000000b
[7Y1] 333: jz multi2
[V53] 334: mov di,offset ram
[1G2] 335: and dh,01111111b
[HN3] 336: multi2: mov bx,0e04h
[6D3] 337: mov si,offset cmos
[CL2] 338: test dh,00000001b
[E42] 339: jnz multi3
[1Z1] 340: mov bl,0ah
[T63] 341: mov si,offset ram
[Y14] 342: multi3: xor dh,00000001b
[000] 343:
[MM2] 344: ;CMOS-RAM lesen
[PF1] 345: push es
```



```
[KF1] 346:      push    cs
[GD1] 347:      pop     es
[5M1] 348:      mov     al,0
[5Y1] 349:      mov     cl,40h
[BL3] 350: cmoslesen: out     70h,al
[7S1] 351:      xchg    al,ah
[3R1] 352:      in     al,71h
[NX0] 353:      stosb
[7S1] 354:      xchg    al,ah
[EA1] 355:      inc     al
[1R2] 356:      loop    cmoslesen
[GD1] 357:      pop     es
[000] 358:
[2A4] 359: ;Adresseninhalte ausgeben
[BL2] 360:      test    dh,00000100b
[XA2] 361:      jnz     adrinhl
[000] 362:
[0E6] 363: ;Raster (Zeilen- & Spaltenadr) ausgeben
[ZF1] 364:      push    bx
[PR1] 365:      mov     cl,16
[B91] 366:      dec     bl
[7T1] 367:      call    posi
[5M1] 368:      mov     al,0
[XH4] 369:      or      byte ptr ds:[farbe],8
[XH3] 370: raster1: call    asciil
[BA1] 371:      inc     di
[BA1] 372:      inc     di
[EA1] 373:      inc     al
[WC2] 374:      loop    raster1
[5D7] 375: ;Spaltenadressen über ASCII-Block ausgeben
[PR1] 376:      mov     cl,16
[4L1] 377:      add     di,30
[GR1] 378:      mov     al,15
[1K3] 379: raster2: call    asciil
[PK1] 380:      sub     di,6
[991] 381:      dec     al
[0D2] 382:      loop    raster2
[BA1] 383:      inc     di
[BA1] 384:      inc     di
[ZX1] 385:      mov     al,20h
[NX0] 386:      stosb
[T94] 387: ;Zeilenadressen ausgeben
[DM1] 388:      mov     cl,4
[GAI] 389:      inc     bl
[CK1] 390:      sub     bh,3
[5M1] 391:      mov     al,0
[HE3] 392: raster3: call    posi
[UZ1] 393:      call    asciil
[GAI] 394:      inc     bl
[XT1] 395:      add     al,10h
[4D2] 396:      loop    raster3
[B45] 397:      and     byte ptr ds:[farbe],0f7h
[TD1] 398:      pop     bx
[000] 399:
[T87] 400: ;Speicherinhalt vom CMOS oder RAM ausgeben
[B73] 401: adrinhl: mov     ax,bp
[2T1] 402:      mov     ah,al
[HN1] 403:      mov     cx,4
[BM1] 404:      mov     dl,0
[VX2] 405: adrinhl2: push    cx
[PR1] 406:      mov     cl,16
[7T1] 407:      call    posi
[XD2] 408: adrinhl3: lodsb
[CL2] 409:      test    dh,00000100b
[552] 410:      jz      adrinhl4
[UR1] 411:      cmp     dl,ah
[9B2] 412:      jnz     adrinhl4
[XH4] 413:      or      byte ptr ds:[farbe],8
[NG3] 414: adrinhl4: call    asciil
[B45] 415:      and     byte ptr ds:[farbe],0f7h
[3G1] 416:      add     di,2
[MA1] 417:      inc     dl
[GB2] 418:      loop    adrinhl3
[VD1] 419:      pop     cx
[GAI] 420:      inc     bl
[CB2] 421:      loop    adrinhl2
[8L1] 422:      add     bh,64
[B91] 423:      dec     bl
[4A1] 424:      dec     si
[RF3] 425: ;ASCII-Block ausgeben
[DM1] 426:      mov     cl,4
[822] 427: adrinhl5: std
[1G1] 428:      push    cx
[PR1] 429:      mov     cl,16
[7T1] 430:      call    posi
[ZF1] 431:      push    bx
[000] 432:
[9E2] 433: adrinhl6: lodsb
[MA2] 434:      call    akontr1
[XH4] 435:      mov     bh,byte ptr ds:[farbe]
[F91] 436:      dec     dl
[CL2] 437:      test    dh,00000100b
[H52] 438:      jz      adrinhl7
[UR1] 439:      cmp     dl,ah
[NB2] 440:      jnz     adrinhl7
[SR1] 441:      or      bh,08h
[0A3] 442: adrinhl7: xchg    ah,bh
[ZV0] 443:      stosw
```

```
[WR1] 444:      xchg    ah,bh
[WB2] 445:      loop    adrinhl6
[TD1] 446:      pop     bx
[VD1] 447:      pop     cx
[B91] 448:      dec     bl
[SB2] 449:      loop    adrinhl5
[UL0] 450:      cld
[3P0] 451:      ret
[000] 452:
[C96] 453: ;Prüfen ob 'Steuerzeichen' ausblenden
[L94] 454: akontr1: test    dh,00000001b
[TC2] 455:      jnz     akontr2
[DW1] 456:      cmp     al,20h
[X92] 457:      jnc     akontr2
[124] 458:      mov     al,2eh ; punkt
[542] 459: akontr2: ret
[000] 460:
[TC4] 461: ;CMOS-RAM überschreiben
[TV4] 462: cmosschr1: mov     si,offset ram
[5M1] 463:      mov     al,0
[9Z1] 464:      mov     cx,40h
[465] 465: ;Prüfen auf Änderung mit oder ohne Zeit
[TF4] 466: cmosschr2: test    dh,00010000b
[5E2] 467:      jz      cmosschr3
[SK1] 468:      cmp     al,5
[DH2] 469:      jnc     cmosschr3
[EA1] 470:      inc     al
[9B1] 471:      inc     si
[VG3] 472: cmosschr3: out     70h,al
[7S1] 473:      xchg    al,ah
[CV0] 474:      lodsb
[002] 475:      out     71h,al
[7S1] 476:      xchg    al,ah
[EA1] 477:      inc     al
[KM2] 478:      loop    cmosschr2
[3P0] 479:      ret
[000] 480:
[V26] 481: ;CMOS-RAM in Datei cmos.ram schreiben
[HX2] 482: schrbn1: push    ax
[UV1] 483:      mov     ah,5bh
[964] 484:      mov     dx,offset tcmos + 2
[DN1] 485:      mov     cx,0
[3K1] 486:      int     21h
[182] 487:      jnc     schrbn2
[705] 488:      mov     byte ptr ds:[fcode],al
[P92] 489:      jmp     schrbn3
[G73] 490: schrbn2: mov     bx,ax
[NX1] 491:      mov     ah,40h
[ND3] 492:      mov     dx,offset cmos
[VX1] 493:      call    extra
[4S2] 494: schrbn3: pop     ax
[3P0] 495:      ret
[000] 496:
[M23] 497: extra:  mov     cx,64
[3K1] 498:      int     21h
[YV1] 499:      mov     ah,3eh
[3K1] 500:      int     21h
[1S4] 501:      mov     byte ptr ds:[fcode],0
[3P0] 502:      ret
[000] 503:
[KG3] 504: ;Datei cmos.ram lesen
[LW2] 505: lesen1: push    ax
[WY1] 506:      mov     ah,3dh
[5M1] 507:      mov     al,0
[964] 508:      mov     dx,offset tcmos + 2
[3K1] 509:      int     21h
[M02] 510:      jnc     lesen2
[705] 511:      mov     byte ptr ds:[fcode],al
[822] 512:      jmp     lesen3
[H63] 513: lesen2: mov     bx,ax
[0Z1] 514:      mov     ah,3fh
[873] 515:      mov     dx,offset ram
[VX1] 516:      call    extra
[4R2] 517: lesen3: pop     ax
[3P0] 518:      ret
[000] 519:
[000] 520:
[D52] 521: start  endp
[000] 522:
[3H2] 523: farbe:  db      01h
[PG4] 524: tcmos:  db      3,1,'CMOS.RAM',0
[C34] 525: tkopie: db      9,1,'Kopie',0
[FB5] 526: tstatus: db      17,14,'St.=[',4,','',0
[BW4] 527: thex:   db      15,14,'HEX=[',4,','',6
[M74] 528: tdez:   db      'DEZ=[',6,','',6
[P74] 529: tbin:   db      'BIN=[',16,','',6
[WC5] 530: tascii: db      'ASCII=[',2,','',0ffh
[002] 531: bt:     db      0
[572] 532: fcode:  db      0
[143] 533: ram:    db      64 dup(0)
[W63] 534: cmos:   db      64 dup(0)
[000] 535:
[UG2] 536: programm ends
[4W1] 537:      end     start
```

Listing 4. »chkcmos.asm« gestattet Ihnen, das CMOS-RAM zu prüfen und zu speichern.

System-Beschleunigung

»speedup.asm« (Listing 5) beschleunigt in den meisten Fällen den Computer erheblich – ausgenommen, der Computer arbeitet nicht mit dynamischen Speicherbausteinen (DRAMs), sondern mit statischen Speicherchips.

Jedes System muß in bestimmten Zeitabständen einen DRAM-Refresh durchführen, damit der Speicherinhalt nicht verlorengeht. Dieser Vorgang benötigt einen gewissen Zeitraum, der das System bremst. Der Abstand zwischen den »Auffrischungen« ist jedoch vom Werk aus so eingestellt, daß er öfter als nötig ausgeführt wird. Mit Hilfe von Speedup stellen Sie fest, ob der Refresh-Takt optimal eingestellt ist. Denn ist keine oder nur eine geringe Beschleunigung bemerkbar, so können Sie davon ausgehen, daß der Refresh-Zyklus bereits vom Hersteller optimiert wurde. Speedup verlangt nach dem Start die Eingabe eines neuen Hexadezimalwerts, der den Abstand zwischen den »Auffrischungen« angibt. Falls der eingegebene Hex-Wert zu hoch war, kann es zu einem Paritätsfehler kommen: Der Refresh wurde zu lange nicht ausgeführt, der Speicherinhalt ging verloren. In diesem Fall schalten Sie das System für kurze Zeit aus. Der Paritätsfehler hat keine weiteren Folgen für Ihr System. Eine weitere Wirkung eines zu hoch eingestellten Refresh-Werts betrifft die angeschlossenen Diskettenlaufwerke. Das bedeutet, daß die Lese-/Schreibbefehle der Diskettenlaufwerke nicht mehr fehlerfrei arbeiten können (zum Beispiel Positionierungsfehler) oder daß die Laufwerkszugriffe sehr lange dauern. Dies merken Sie dann bei MS-DOS-Befehlen wie:

a:

Nach der Veränderung des Refresh-Abstands sollten Sie mit einem Testprogramm, etwa mit Landmark, das Ergebnis notieren. Falls das System nicht schneller arbeitet als zuvor, ermitteln Sie den optimalen Wert nur durch weitere Experimente.

In Ausnahmefällen stellen Sie nur eine minimale Beschleunigung des Systems fest. Ab einer Geschwindigkeit von 12 oder mehr MHz erzielen Sie meist die größte Beschleunigung. Bei der Eingabe eines Werts von 15hex oder weniger verlangsamten sich zwar die Rechenoperationen des Prozessors, aber die Zugriffe auf die Diskettenlaufwerke nehmen spürbar ab. Einen Wert von 00hex sollten Sie nicht eingeben, da sonst das System abstürzt.

(Christian Merz/et)

```
[MP3] 1: ;Programm: speedup.asm
[8K5] 2: ;Funktion: Ändern der Refreshrate
[LH4] 3: ;Sprache: MASM ab Version 4.0
[UW3] 4: ;Autor: Christian Merz
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[H22] 6: Code segment
[8U3] 7: assume cs:Code, ds:Code
[A61] 8: org 100h
[000] 9:
[LU2] 10: SpeedUp Proc far
[8C1] 11: push cs
[E31] 12: pop ds
[H62] 13: lea dx,getout
[BD1] 14: mov ah,09
[381] 15: int 21h
[M02] 16: lea si,wert1
[8A1] 17: mov al,3
[U32] 18: mov [si],al
[4H1] 19: mov dx,si
[UC1] 20: mov ah,10
[381] 21: int 21h
[RF1] 22: mov di,si
[B01] 23: inc di
[5D1] 24: mov al,00
[V02] 25: cmp [di],al
[H91] 26: jz ENDEN
[0F1] 27: xor ah,ah
[171] 28: add si,2
[RF1] 29: call zahl
[5F1] 30: mov bh,bl
[911] 31: inc si
[RF1] 32: call zahl
[HD1] 33: mov cl,04
```

```
[ND1] 34: shl bh,cl
[3B1] 35: add bh,al
[GH1] 36: xor ax,ax
[CM1] 37: mov al,74h
[0P1] 38: out 43h,al
[7F1] 39: mov al,bh
[UN1] 40: out 41h,al
[5D1] 41: mov al,00
[UN1] 42: out 41h,al
[N02] 43: ENDEN:int 20h
[5C2] 44: zahl proc near
[532] 45: mov al,[si]
[CK1] 46: and al,6fh
[HK1] 47: cmp al,60h
[5H0] 48: ja n
[AK1] 49: and al,4fh
[FK1] 50: cmp al,40h
[7M0] 51: ja n1
[6K1] 52: and al,0fh
[LA2] 53: back:mov bl,al
[3P0] 54: ret
[DS1] 55: n:cmp al,66h
[671] 56: ja ENDEN
[PV1] 57: gl:and al,0fh
[P91] 58: add al,09
[TB1] 59: jmp back
[RU1] 60: n1:cmp al,46h
[671] 61: ja ENDEN
[D21] 62: jmp gl
[EG1] 63: zahl endp
[0B5] 64: getout db "Neuer Wert in Hex: ",24h
[LS2] 65: wert1 db 6 Dup (?)
[A52] 66: SpeedUp Endp
[TF1] 67: Code Ends
[U22] 68: End SpeedUp
```

Listing 5. »speedup.asm« verpaßt Ihrem Rechner eine individuelle RAM-Refresh-Rate.

Nicht bereit beim Lesen...

Wie oft ist es Ihnen schon beim Wechseln auf das eine oder das andere Laufwerk passiert, daß Sie die falsche Laufwerkskennung eingeben haben? Nach

a:[Enter]

erschreckt Sie die Meldung

Nicht bereit beim Lesen von Laufwerk A

(A)bbrechen, (W)iederholen, (U)ebergehen?

Die Tasten [A] und [W] helfen wenig, wenn Sie keine Diskette parat haben. Mit der Taste [U] wiederholt sich meist die Meldung, und nur etwa beim zweiten oder dritten Mal erlaubt Ihnen MS-DOS den Wechsel auf das richtige Laufwerk mit der Meldung

Aktuelles Laufwerk nicht mehr gültig>

Das Betriebssystem erwartet nun eine Laufwerkskennung wie A, B oder C, die Sie mit [Enter] bestätigen. Das heißt, mit etwas Glück haben Sie nach mindestens sieben Tastendrücken plus Wartezeiten den Wechsel auf das richtige Laufwerk vollzogen.

Wenn Sie anstelle von »a:« den Befehl »cd a:« eingeben, geschieht folgendes: Falls das Laufwerk bereit ist, leuchtet die entsprechende Lampe, es wird aber nicht auf A umgeschaltet. Wenn das Laufwerk nicht bereit ist, erhalten Sie die schon zuvor erwähnte Meldung »Nicht bereit...«, die Sie aber mit »(A)bbrechen« verlassen dürfen.

Diese Schwierigkeit löst das kleine Programm »a.asm« (Listing 6). Nach dem Kompilieren mit MASM ab Version 4.0 und dem Link-Lauf konvertieren Sie die Exe-Datei:

exe2bin a.exe a.com

Das Programm schaltet auf das Laufwerk A um. Sie können es auch mit kleinen Änderungen für andere Laufwerke anpassen. »a.com« reduziert die Tastendrücke bei falscher Wahl des Laufwerks auf drei. Von C aus geben Sie ein:

a:[Enter]

Nach diesen beiden Tastendrücken beantworten Sie die sattsam bekannte Meldung

Nicht bereit beim Lesen von Laufwerk A

(A)bbrechen, (W)iederholen, (U)ebergehen?

mit einem dritten Tastendruck:

a

Damit haben Sie wieder auf Laufwerk C zurückgeschaltet. Das Programm benutzt die Funktionen 3Bhex (Change Directory), 0Ehex (Select Disk) und 4Chex (Terminate a Process) des Interrupts 21hex von MS-DOS.

(Henrique Sousa/et)

```
[LM2] 1: ;Programm: a.asm
[Y57] 2: ;Funktion: vereinfachter Laufwerkswechsel
[LH4] 3: ;Sprache: MASM ab Version 4.0
[RH5] 4: ;Autor: Dipl.-Ing. Henrique Sousa
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[B91] 6: TITLE a
[G73] 7: _TEXT SEGMENT
[DB4] 8: ASSUME cs: _TEXT, ds: _TEXT
[AG1] 9: ORG 1000h
[000] 10:
[XC3] 11: start: jmp begin
[NM3] 12: laufwerk db "a:",13,10
[UN5] 13: begin: mov dx,offset laufwerk
[8W1] 14: mov ax,3b00h
[381] 15: int 21h
[TC1] 16: and ax,3h
[AE1] 17: cmp ax,3h
[VB1] 18: jne ende
[ZF1] 19: mov dx,0h
[BW1] 20: mov ax,0e00h
[381] 21: int 21h
[8K3] 22: ende: mov ax,4c00h
[381] 23: int 21h
[5H2] 24: _TEXT ENDS
[4K1] 25: END start
```

Listing 6. Mit »a.asm« sparen Sie sich beim Laufwerkwechsel mit fehlender Diskette Zeit und Arbeit.

Keine Chance den Boot-Sektor-Viren auf Floppies

»delbvir.asm« (Listing 7) assemblieren Sie mit MASM ab Version 5.0. Delbvir sollte allerdings auch mit anderen Assemblern übersetzbar sein. Nach dem Link-Lauf erhalten Sie nach bekanntem Muster mit Exe2bin die ausführbare Com-Datei.

Delbvir ist aus dem Bedürfnis entstanden, Boot-Sektor-Viren effizient zu entfernen. Sie rufen es mit

delbvir

auf und manipulieren damit den Boot-Sektor des Laufwerks A. Wollen Sie den Boot-Sektor des Laufwerks B ersetzen, passen Sie den Quellcode an den zwei markierten Stellen an.

Die Funktionsweise von Delbvir lässt sich in wenigen Sätzen erklären. Delbvir liest den Boot-Sektor, ersetzt den Boot-Code und schreibt den Sektor wieder zurück. Der neue Boot-Code ist sehr einfach und löst einen Reboot aus, wenn Sie das System von einer mit Delbvir bearbeiteten Diskette starten wollen. Das heißt, daß mit Delbvir behandelte Disketten nicht mehr bootfähig sind. Dies ist auch sinnvoll, da ja der virulente Boot-Anteil gelöscht werden sollte.

Delbvir können Sie nicht auf die Festplatte anwenden. Dort löschen Sie Boot-Sektor-Viren mit folgendem Befehl:

fdisk /mbr

Diese Option des Fdisk in MS-DOS 5.0 ist leider undokumentiert. Für Schäden, die durch den Gebrauch von »fdisk /mbr« oder Delbvir entstehen, übernimmt der Autor keine Haftung.

Wie Sie den Listingzeilen 18 und 19 entnehmen, fängt delbvir auch die Fehlermeldungen ab, wenn das Programm entweder keine oder eine schreibgeschützte Diskette vorfindet. Ein erfolgreiches Bearbeiten der Diskette bemerken Sie allerdings nur am Zugriff auf das Laufwerk, weil Delbvir dies nicht explizit meldet. Die Änderungen für das Laufwerk B führen Sie in den Zeilen 22 und 41 laut Anweisung

durch. Für Anwender, die DFÜ nutzen, dürfte Listingzeile 67 eine wichtige Nachricht enthalten: Hierin finden Sie einen Zugang zum Autor über das weltweit verbreitete Fidonet.

Sollten Sie eine mit Delbvir behandelte Diskette im bootfähigen Laufwerk belassen, wirkt auf der Diskette der Programmteil ab der Sprungmarke »Newboot« (ab Zeile 65). Sollten Sie sich derweil jedoch von Ihrem PC-Arbeitsplatz entfernen müssen, befindet sich die Maschine in einer eigenartigen Endlosschleife: Bootvorgang mit Zugriff auf das Laufwerk und so weiter von vorn. Mit »delbvir.asm« hoffen wir, daß Ihr Rechner ein bißchen besser gegen solche Viren geschützt ist. Ein professionelles Antivirus-Paket kann delbvir allerdings nicht ersetzen.

(Markus Maeder/et)

```
[SN3] 1: ;Programm: delbvir.asm
[M77] 2: ;Funktion: löscht Boot Virus der Disketten
[RH4] 3: ;Sprache: MASM ab Version 5.0
[7H3] 4: ;Autor: Markus Maeder
[RR4] 5: ;(c) 1992 DMV Widuch GmbH & Co.KG
[1Z4] 6: seg_dbv segment byte public
[CX4] 7: assume cs:seg_dbv, ds:seg_dbv
[000] 8:
[A61] 9: org 1000h
[000] 10:
[3L0] 11: Go:
[V11] 12: jmp Start
[000] 13:
[VN0] 14: SEC:
[3R2] 15: JUMP DB 3 dup (?)
[XM2] 16: OEM DB 3Bh dup (?)
[6V2] 17: DATA DB 1C2h dup (?)
[9U5] 18: Lesefehler DB "Lesefehler!",0Dh,0Ah,"$"
[0U6] 19: Schreibfehler DB "Schreibfehler!",0Dh,0Ah,"$"
[000] 20:
[E21] 21: Start:
[0Y2] 22: mov al,0 ; fuer B: 1
[Z22] 23: mov bx, offset SEC
[EB1] 24: mov cx,1
[FB1] 25: mov dx,0
[F81] 26: int 25h
[YY1] 27: jc ErrLesen
[000] 28:
[VT2] 29: mov [JUMP],0EBh
[VN2] 30: mov [JUMP+1],3Ch
[V13] 31: mov ax,offset DATA
[PC1] 32: push ax
[G21] 33: pop di
[3N3] 34: mov ax,offset Newboot
[PC1] 35: push ax
[E31] 36: pop si
[AX1] 37: mov cx,1C2h
[9M1] 38: rep movsb
[000] 39:
[000] 40:
[0Y2] 41: mov al,0 ; fuer b: 1
[Z22] 42: mov bx, offset SEC
[EB1] 43: mov cx,1
[FB1] 44: mov dx,0
[K81] 45: int 26h
[TN2] 46: jc ErrSchreiben
[000] 47:
[4D1] 48: jmp Ende
[000] 49:
[DL1] 50: ErrLesen:
[474] 51: mov dx, offset Lesefehler
[PM1] 52: jmp Error
[000] 53:
[M92] 54: ErrSchreiben:
[DS4] 55: mov dx, offset Schreibfehler
[000] 56:
[931] 57: Error:
[3M1] 58: mov ah,09h
[381] 59: int 21h
[000] 60:
[XT0] 61: Ende:
[081] 62: int 20h
[000] 63:
[YG1] 64: Newboot:
[LE3] 65: ; JMP FAR PTR FFFF:0 ->
[3U3] 66: DB 0EAh,00h,00h,0FFh,0FFh
[CH8] 67: DB "DELETE BOOT VIRUS <BY> M. MAEDER, FIDON
ET 2:301/801.2"
[UR1] 68: DB 512 DUP (?)
[000] 69:
[D42] 70: seg_dbv ends
[Z01] 71: end Go
```

Listing 7. »delbvir .asm« entfernt Boot-Sektoren von Disketten

Paßwortabfrage in COM-Dateien

»password.deb« (Listing 10) schützt Ihre Com-Dateien vor unbefugtem Zugriff. Nachdem Sie es mit

```
debug <password.deb
```

kompiliert haben, können Sie jede beliebige Com-Datei mit einem Paßwort versehen. Aus Platzgründen – es handelt sich um eine Einsendung zum 1024-Wettbewerb der DOS International – verzichtet »Password« auf eine Fehlerbehandlung. Nach dem Aufruf mit

```
password [Dateiname]
```

geben Sie das künftige Paßwort ein. Es darf höchstens 16 Buchstaben umfassen. Sie versehen daraufhin die Com-Datei mit einer Paßwortabfrage. Die Datei vergrößert sich dabei um 127 Byte. Im Test ergaben sich aber auch verschiedene Verlängerungen, beispielsweise von 143 Byte. Auch sprechen dabei Virenwarnsysteme wie PC-Cillin an (DOS International 6'92, Seite 158).

Da die maximale Größe einer Com-Datei 65 535 Byte beträgt, darf diese also vor der Änderung nicht größer als 65 408 Byte sein, was die meisten Com-Dateien erfüllen dürften.

Beachten Sie bitte, daß Sie die Änderung in der Datei nicht mehr rückgängig machen können, da nicht einmal eine Sicherheitskopie angefertigt wird.

Nach dem Start der Com-Datei werden Sie zuerst nach dem Paßwort gefragt. Dies geben Sie durch Sterne »***« verdeckt ein, womit Sie verhindern, daß andere Personen Ihr Paßwort lassen.

Ist das eingegebene Wort richtig, so startet das Programm wie gewohnt, andernfalls endet es sofort. Sie finden sich auf der Betriebssystemebene wieder.

Da fast jedes beliebige Zeichen außer Steuerzeichen im Paßwort vorkommen darf und auch die Groß- und Kleinschreibung beachtet wird, gibt es etwa 6,3 Trillionen Kombinationen.

»password.com« sollten Sie aus der MS-DOS-Umgebung starten, da es auf dessen Parameterübergabe abgestimmt ist. So wird zum Beispiel erwartet, daß die Parameterzeile mit einem Leerzeichen beginnt und mit einem Return-Code (13dez) endet. Es wird außerdem vorausgesetzt, daß Sie einen Dateinamen angeben. Ist das nicht der Fall, versucht »Password« trotzdem, eine Datei zu öffnen; eine Fehlermeldung wird nicht ausgegeben. Dieses Problem konnte wegen Platzmangels nicht abgefangen werden.

Die mit »Paßword« versehene Com-Datei betreffen diese Einschränkungen natürlich nicht, denn sie starten Sie auch von anderen Programmen aus, die die Parameter etwas anders übergeben.

Noch ein Tip: Wenn Sie »command.com« mit einem Paßwort schützen, verhindern Sie, daß Unbefugte den Compter starten können. Denn »command.com« startet während des Hochfahrens noch vor »autoexec.com«. Bei Eingabe eines falschen Paßworts stürzt der Computer ab, da er keinen Befehlsinterpreter laden kann.

Allerdings kann es vorkommen, daß andere Programme wie Norton Commander »command.com« ebenfalls aufrufen; dann müssen Sie das Paßwort erneut eingeben. Manche Programme wie der Editor von MS-DOS 5.0 stürzen aber auch einfach ab, wenn Sie »command.com« ändern.

(Thomas Amthor/et)

```
[T14] 1: ;Programm: password.deb
[S97] 2: ;Funktion: Paßwort-Schutz von Com-Programmen.
[WU4] 3: ;Sprache: debug<password.deb
[BL3] 4: ;Autor: Thomas Amthor
[HS4] 5: ;(c)1992 DMV Widuch GmbH & Co.KG
[RD2] 6: n password.com
[240] 7: a
[2A1] 8: mov ah,9
[KM1] 9: mov dx,1c4
```

```
[3Z0] 10: int 21
[AA1] 11: mov ah,a
[8P1] 12: mov dx,1cf
[3Z0] 13: int 21
[CE1] 14: mov si,82
[CV0] 15: lodsb
[791] 16: cmp al,d
[Z51] 17: jnz 111
[MF1] 18: xor al,a
[182] 19: mov [si-1],al
[4U1] 20: mov ax,3d00
[VE1] 21: mov dx,82
[3Z0] 22: int 21
[CH1] 23: mov bx,ax
[8E1] 24: mov ah,3f
[TZ1] 25: mov cx,ffff
[NM1] 26: mov dx,1f3
[3Z0] 27: int 21
[MG1] 28: mov bp,ax
[6E1] 29: mov ah,3e
[3Z0] 30: int 21
[VG1] 31: mov ax,bp
[9H1] 32: add ax,17f
[Y02] 33: mov [1c0],ax
[2E1] 34: mov ah,3c
[PH1] 35: xor cx,cx
[VE1] 36: mov dx,82
[3Z0] 37: int 21
[CH1] 38: mov bx,ax
[XC1] 39: mov ah,40
[UL1] 40: mov dx,174
[ZG1] 41: mov cx,bp
[ZB1] 42: add cx,7f
[3Z0] 43: int 21
[SL1] 44: mov dx,164
[XC1] 45: mov ah,40
[EE1] 46: mov cx,10
[3Z0] 47: int 21
[6E1] 48: mov ah,3e
[3Z0] 49: int 21
[0Z0] 50: int 20
[DS0] 51: repz
[PX0] 52: movsb
[GH1] 53: xor ax,ax
[LH1] 54: xor bx,bx
[TH1] 55: xor dx,dx
[EF1] 56: xor di,di
[VG1] 57: xor si,si
[PC1] 58: push ax
[LS0] 59: popf
[6P0] 60: sti
[R10] 61: db c3
[2A1] 62: mov ah,9
[AL1] 63: mov dx,150
[3Z0] 64: int 21
[YL1] 65: mov di,16e
[KB1] 66: mov bx,7
[U91] 67: mov ah,1
[GZ0] 68: int 16
[PU0] 69: jz 181
[0F1] 70: xor ah,ah
[GZ0] 71: int 16
[NX0] 72: stosb
[PC1] 73: push ax
[5N1] 74: mov ax,e2a
[YY0] 75: int 10
[R31] 76: pop ax
[EK1] 77: cmp di,17e
[HW0] 78: jz 19d
[791] 79: cmp al,d
[M61] 80: jnz 181
[RM1] 81: mov si,15d
[YL1] 82: mov di,16e
[EE1] 83: mov cx,10
[DS0] 84: repz
[4W0] 85: cmpsb
[VW0] 86: jz 1ac
[0Z0] 87: int 20
[UZ1] 88: mov [14e],cs
[ZZ1] 89: mov cx,[14c]
[WL1] 90: sub cx,17f
[1N1] 91: mov si,17f
[LH1] 92: mov di,100
[3B1] 93: push di
[951] 94: jmp 0:0
[ZC2] 95: db "Password: $"
[172] 96: db 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[YZ1] 97: db 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[YZ1] 98: db 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[YZ1] 99: db 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[000] 100:
[DP0] 101: rcx
[590] 102: F3
[E50] 103: w
[250] 104: q
```

Listing 8. Ein Paßwort für Com-Dateien legen Sie mit »password.deb« fest.

Interrupt-Funktionen

Ein Interrupt für alle Fälle

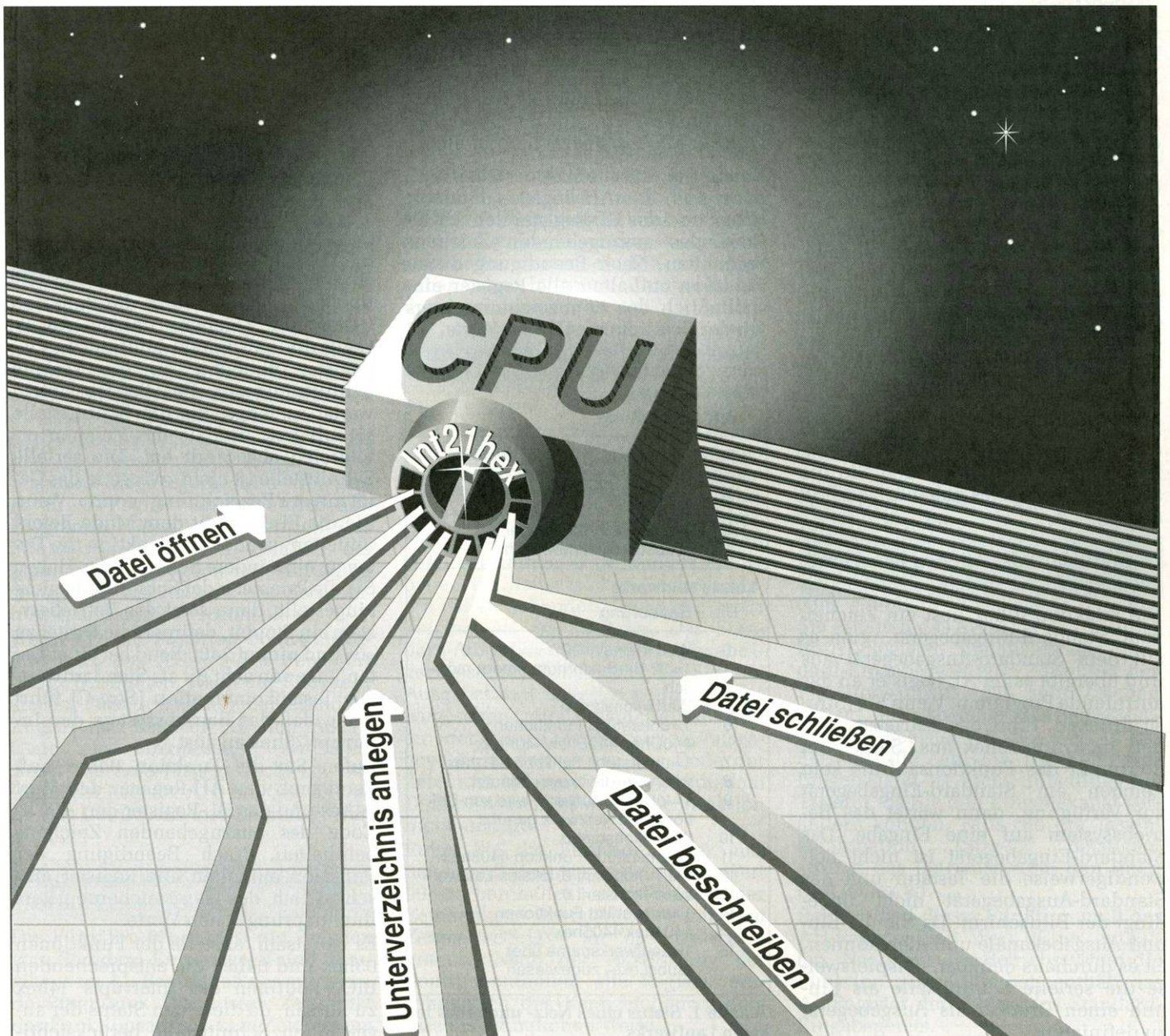
Über den Interrupt 21hex lassen sich alle Standardaufgaben erfüllen, die ein Programmierer zu erledigen hat. Neben den von Microsoft dokumentierten Funktionen gibt es jedoch zahlreiche undokumentierte Routinen, die komplexe Systeminterna behandeln und deren Aufbau Ihnen die mühselige Detailforschung erspart. Sie lernen den Interrupt 21hex kennen und beschäftigen sich mit den CP/M-kompatiblen Funktionen.

Der Interrupt 21hex ist einer der wichtigsten Interrupts, weil Sie über ihn fast alle MS-DOS-Funktionen aufrufen können. Er ist damit für versierte Programmierer ein unerläßliches Hilfsmittel.

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

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 die CP/M-kompatiblen Funktionen über einen Dateisteuerungsblock (FCB = File Control Block) auf die Datei zugreifen, während die Xenix-kompatiblen Funktionen den Zugriff auf eine Datei über ein Handle steuern (einen Zugriffskanal für Dateien und Peripheriegeräte). Beachten Sie bitte, daß sowohl dieser als auch der folgende Beitrag undokumentierte und von Microsoft dokumentierte Funktionen aufführt. Die Tabelle 1, aus der Sie den Status eines Netz- und eines lokalen Laufwerks entnehmen können, ist vor allem für die Funktion 44hex interessant.

Funktion 00hex: beendet das aktuelle Programm

Die Funktion 00hex des Interrupts 21hex (»Terminate Program«) dient dazu, den Ablauf des aktuellen Programms zu unterbrechen und in das aufrufende Programm zurückzukehren. Das ist in vielen Fällen die Kommandoebene des Betriebssystems MS-DOS, die alle zur Verfügung stehenden Dateien verwaltet. Bevor die Prozedur die Kontrolle an das aufrufende Programm übergibt, restauriert sie die drei Interrupt-Vektoren, deren Inhalt vor dem Start des Programms im PSP (Program Segment Prefix) gespeichert wurde. Der PC stellt den Arbeitsspeicher, den das beendete Programm belegt, nach dem Funktionsaufruf anderen Programmen zur Verfügung und schreibt den Inhalt aller Dateipuffer auf den angesprochenen Datenträger. Soweit es vermeidbar ist, sollten Sie anstelle dieser Routine die Funktion 4Chex des Interrupts 21hex verwenden.

Beim Aufruf der Funktion 00hex muß das AH-Register den Wert 00hex enthalten. Im CS-Register legen Sie die Segmentadresse des zugehörigen PSP ab.

Funktion 01hex: Zeichen lesen und ausgeben

Die Funktion 01hex liest ein Zeichen vom Standard-Eingabegerät, gibt es auf dem Standard-Ausgabegerät aus und übergibt es im AL-Register an das aufrufende Programm. Wenn Sie [Strg-C] drücken, führt das Betriebssystem den Interrupt 23hex aus. Steht zum Zeitpunkt des Funktionsaufrufs kein Zeichen am Standard-Eingabegerät zur Verfügung, dann wartet das Betriebssystem auf eine Eingabe. Das Standard-Eingabegerät ist nicht notwendigerweise die Tastatur und das Standard-Ausgabegerät nicht unbedingt der Bildschirm. Da Sie die Ein- und Ausgabekanäle umleiten können, ist es durchaus denkbar, beispielsweise die serielle Schnittstelle als Ein- und einen Drucker als Ausgabegerät zu definieren.

Wenn Sie die Funktion 01hex aufrufen, muß das AH-Register den Wert 01hex beinhalten.

Nach der Rückkehr aus der Funktion enthält das Register AL den ASCII-Code des eingelesenen Zeichens. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 02hex: Zeichen auf dem Standard-Ausgabegerät ausgeben

Die Funktion 02hex gibt ein Zeichen, das das aufrufende Programm im DL-Register übergibt, auf dem Standard-Ausgabegerät aus. Wenn Sie [Strg-C] drücken, führt das Betriebssystem den Interrupt 23hex aus.

Ist das Standard-Ausgabegerät der Bildschirm, dann behandelt das Betriebssystem die ASCII-Zeichen, die einen Code kleiner als 32 haben, als Steuerzeichen. In den anderen Fällen gibt die Prozedur die Zeichen als ASCII-Codes aus. Das Ausgabegerät muß nicht unbedingt der Bildschirm sein, Sie können zum Beispiel einen Drucker als Ausgabegerät definieren.

Rufen Sie die Funktion 02hex auf, dann muß das AH-Register den Wert 02hex und das DL-Register den ASCII-Code des auszugebenden Zeichens beinhalten. Nach Beendigung dieser Funktion enthalten alle Register einschließlich des Kennzeichenregisters wieder ihre ursprünglichen Werte.

Netz-Laufwerke	
Bit	Bedeutung
09	1=Laufwerk wird von mehreren Stationen im Netzwerk genutzt
12	1=Laufwerk ist aus einem Netzwerk
15	1=Laufwerksname über »subst.exe« zugewiesen
Lokale Laufwerke	
Bit	Bedeutung
0	Wert=konstant 0
1	1=32-Bit-Sektornummern möglich
2 bis 5	Wert=konstant 0
6	1=unterstützt Funktionen 440Dhex/440Ehex/440Fhex
7	1=unterstützt Funktion 4411hex
8	wird von »driver.sys« benutzt
9	1=lokales Laufwerk, wird von Stationen im Netzwerk genutzt
10	Wert=konstant 0
11	1=unterstützt Funktion 4408hex
12	Wert=konstant 0, lokales Laufwerk
13	Wert=konstant 0
14	1=unterstützt Funktionen 4404hex/4405hex
15	1=Laufwerksname über »subst.exe« zugewiesen

Tabelle 1. Status eines Netz- und eines lokalen Laufwerks

Funktion 03hex: Zeichen von der seriellen Schnittstelle lesen

Mit der Funktion 03hex lesen Sie ein Zeichen von der seriellen Schnittstelle ein. Das Unterprogramm übergibt es im AL-Register an das aufrufende Programm. Wenn Sie [Strg-C] drücken, führt das Betriebssystem den Interrupt 23hex aus. Steht zum Zeitpunkt des Funktionsaufrufs kein Zeichen an der seriellen Schnittstelle zur Verfügung, wartet das Betriebssystem, bis ein Wert empfangen wird. Die serielle Schnittstelle müssen Sie nicht unbedingt mit »com1« bezeichnen. Wenn Sie die Einheit mit dem Mode-Kommando umleiten, empfängt die Funktion die Daten über eine andere Schnittstelle. Haben Sie die Parameter der Schnittstelle nicht eingestellt, setzt das Betriebssystem ein Stopbit, definiert das Paritätsbit zu »0« und nimmt 2400 Baud bei einer Datenlänge von acht Bit als Standardwert.

Beim Aufruf dieser Prozedur muß das AH-Register den Wert 03hex beinhalten.

Nach der Rückkehr aus der Funktion enthält das Register AL das empfangene Zeichen. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 04hex: Zeichen auf serieller Schnittstelle ausgeben

Im Gegensatz dazu gibt die Funktion 04hex ein Zeichen, dessen ASCII-Code die aufrufende Routine im DL-Register übergibt, an die serielle Schnittstelle aus. Das Betriebssystem wartet so lange, bis die Schnittstelle signalisiert, daß sie das Zeichen erfolgreich abgesandt hat. Die serielle Schnittstelle ist nicht zwingend das Gerät mit der Bezeichnung »com1«. Wenn Sie die Einheit mit dem Mode-Befehl umleiten, sendet die Funktion die Daten an eine andere Schnittstelle. Haben Sie die Parameter der Schnittstelle nicht eingestellt, dann setzt das Betriebssystem ein Stopbit, definiert die Parität zu »0« und nimmt 2400 Baud bei einer Datenlänge von acht Bit als Standardwert. Die Tastenkombination [Strg-C] führt dazu, daß das Betriebssystem den Interrupt 23hex auslöst.

Rufen Sie die Funktion 04hex auf, dann muß das AH-Register den Wert 04hex und das DL-Register den ASCII-Code des auszugebenden Zeichens beinhalten. Nach Beendigung der Funktion enthalten alle Register einschließlich des Kennzeichenregisters ihre ursprünglichen Werte.

Es ist ratsam, anstelle der Funktionen 03hex und 04hex die entsprechenden BIOS-Routinen des Interrupts 14hex zu nutzen, da diese den Status der angewählten Schnittstelle berücksichti-

gen und bei Bedarf an das Hauptprogramm übermitteln.

Funktion 05hex: Zeichen an die parallele Schnittstelle weiterleiten

Analog zur Funktion 04hex leitet die Funktion 05hex ein Zeichen, dessen ASCII-Code das aufrufende Programm im DL-Register übergibt, an die parallele Schnittstelle weiter. Das Betriebssystem wartet so lange, bis die parallele Schnittstelle signalisiert, daß sie das Zeichen weitergeleitet hat. Die Schnittstelle ist nicht unbedingt das Gerät mit der Bezeichnung »prn« beziehungsweise »lpt1«. Leiten Sie die Ausgabe mit dem Mode-Kommando um, dann sendet die Funktion die Daten an die eingestellte Ausgabereinheit. Falls Sie die Tastenkombination [Strg-C] betätigen, springt das Betriebssystem den Interrupt 23hex an.

Sie wählen die Funktion 05hex richtig an, wenn das AH-Register den Wert 05hex und das DL-Register den ASCII-Code des auszugebenden Zeichens enthalten. Nach der Rückkehr aus der Funktion sind alle Register einschließlich des Kennzeichenregisters unverändert.

Beachten Sie bitte, daß die Funktionen 03hex, 04hex und 05hex keinen Fehlercode und keinen Statuswert an das aufrufende Programm übergeben.

Funktion 06hex: ist für die Zeichenein- und -ausgabe verantwortlich

Die Funktion 06hex stellt zwei Unterfunktionen zur Verfügung, die das Betriebssystem dem Inhalt des DL-Registers entsprechend aufruft. Enthält DL einen Wert ungleich FFhex, dann gibt die Prozedur den Inhalt auf dem Standard-Ausgabegerät aus. Andernfalls liest die Prozedur ein Zeichen vom Standard-Eingabegerät und übergibt es im AL-Register an das aufrufende Programm. Wenn zum Zeitpunkt des Funktionsaufrufs kein Zeichen am Standard-Eingabegerät zur Verfügung steht, setzt die Routine das Zero-Statusbit, anderenfalls löscht sie es.

Wenn Sie die Funktion 06hex aufrufen, muß das AH-Register den Wert 06hex beinhalten. Nach der Rückkehr aus der Funktion enthält bei lesendem Zugriff das Register AL den ASCII-Code des eingelesenen Zeichens, wenn das Zero-Flag gelöscht ist. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktionen 07hex und 08hex: Warten auf Zeicheneingabe

Die Funktion 07hex liest ein Zeichen vom Standard-Eingabegerät und übergibt es im AL-Register an das aufrufende Programm. Wenn zum Zeitpunkt des Funktionsaufrufs kein Zeichen am

Standard-Eingabegerät zur Verfügung steht, wartet das Betriebssystem auf eine Eingabe. Auch in diesem Fall ist das Standard-Eingabegerät nicht unbedingt die Tastatur, da Sie den Eingabekanal umleiten können.

Wenn Sie die Funktion 07hex aufrufen, muß das AH-Register den Wert 07hex beinhalten. Nach Beendigung der Funktion enthält das Register AL den ASCII-Code des eingelesenen Zeichens.

Die Funktion 08hex liest ebenfalls ein Zeichen vom Standard-Eingabegerät und übergibt dieses im AL-Register an die aufrufende Routine. Steht zum Zeitpunkt des Funktionsaufrufs kein Zeichen am Standard-Eingabegerät zur Verfügung, dann wartet das Betriebssystem auf eine Eingabe. Der einzige Unterschied zur Funktion 07hex besteht darin, daß das Betriebssystem den Interrupt 23hex ausführt, wenn Sie [Strg-C] drücken.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 08hex enthalten. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den ASCII-Code des eingelesenen Zeichens.

Alle bisher vorgestellten Routinen arbeiten mit einzelnen Zeichen, deren ASCII-Codes die Funktion jeweils in einem Register übernimmt. Das ist in vielen Fällen sinnvoll und oft auch notwendig. Die folgenden Prozeduren sind dafür vorgesehen, Ein- und Ausgaben von längeren Zeichenketten durchzuführen.

Funktion 09hex: gibt Zeichenkette auf Standardausgabegerät aus

Die Funktion 09hex beispielsweise gibt eine Zeichenkette, deren Position durch die Register DS und DX angegeben ist, auf dem Standard-Ausgabegerät aus. Wenn das Standard-Ausgabegerät der Bildschirm ist, dann bearbeitet das Betriebssystem die ASCII-Zeichen, die einen Code kleiner als 32 haben, als Steuerzeichen. In den anderen Fällen behandelt die Prozedur die Zeichen als ASCII-Codes. Das Standard-Ausgabegerät ist nicht notwendigerweise der Bildschirm. Da Sie den Ausgabekanal umleiten können, ist es vorgesehen, zum Beispiel einen Plotter oder einen Laserdrucker als Ausgabegerät zu definieren. Das Ende der Zeichenkette markieren Sie mit dem Dollar-Zeichen (>\$<), das die Funktion nicht anzeigt.

Rufen Sie die Funktion 09hex auf, dann muß das AH-Register den Wert 09hex beinhalten. Die Position der Zeichenkette übergeben Sie mit dem Zeiger »DS:DX« an das Unterprogramm. Nach der Ausführung der Routine enthalten alle Register einschließlich des Kennzeichenregisters ihre ursprünglichen Werte.

Funktion 0Ahex: Zeichenkette einlesen und Speicherung im Puffer

Die Funktion 0Ahex liest eine Zeichenkette vom Standard-Eingabegerät und speichert sie in einem vorher definierten Eingabepuffer. Das Betriebssystem verläßt die Funktion ausschließlich dann, wenn der Benutzer die Eingabe durch Drücken von [Enter] bestätigt.

Empfängt die Funktion Daten, obwohl der Puffer voll ist, dann sendet das Betriebssystem das Klingelzeichen (ASCII-Code 7) an das Standard-Ausgabegerät.

Wenn die Tastatur das Eingabemedium ist, sind Sie in der Lage, die eingegebene Zeichenkette mit den normalen Textkommandos zu editieren. Das Standard-Eingabegerät ist aber nicht unbedingt die Tastatur, da Sie den Eingabekanal umleiten können. Wenn Sie [Strg-C] drücken, führt das Betriebssystem den Interrupt 23hex aus. Die Position des Tastaturpuffers übergeben Sie mit dem Zeiger »DS:DX« an das Unterprogramm.

Um die Funktion 0Ahex aufzurufen, muß das AH-Register den Wert 0Ahex beinhalten. Nach Bearbeitung der Funktion enthält der Eingabepuffer ab dem dritten Byte die eingegebene Zeichenkette. Das zweite Byte gibt die Länge der Zeichenkette an, ohne das abschließende Zeichen zu berücksichtigen.

Vor dem Aufruf der Prozedur geben Sie im ersten Byte des Tastaturpuffers an, wie viele Bytes die Funktion einlesen und ab der dritten Position des Tastaturpuffers speichern soll. Die Längenangabe berücksichtigt das abschließende Zeichen. Die Funktion verändert keinen Registerwert.

Funktion 0Bhex: Prüfen, ob Zeichen bereitsteht

Mit der Funktion 0Bhex stellen Sie fest, ob ein Zeichen bereitsteht und vom Standard-Eingabegerät kommt; Sie erhalten das Ergebnis der Prüfung im AL-Register. Signalisiert das Standard-Eingabegerät zum Zeitpunkt des Funktionsaufrufs, daß kein Zeichen anliegt, dann speichert das Betriebssystem im Register AL den Wert 0, anderenfalls die Zahl 255. Das Standard-Eingabegerät ist nicht unbedingt die Tastatur, da Sie den Eingabekanal verändern können. Falls Sie die Tastenkombination [Strg-C] betätigen, springt das Betriebssystem den Interrupt 23hex an.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 0Bhex enthalten. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das AL-Register den Status des Standard-Eingabegeräts.

Funktion 0Chex: Eingabepuffer löschen, Aufruf zur Zeicheneingabe

Die Funktion 0Chex löscht den Standard-Eingabepuffer und ruft eine der Funktionen zur Zeichen-Eingabe auf (01hex, 06hex, 07hex, 08hex und 0Ahex). Diese Funktionen lesen daraufhin ein Zeichen vom Standard-Eingabegerät und übergeben es im AL-Register an das aufrufende Programm. Wenn Sie einen falschen Parameter übergeben, kehrt die Prozedur ins Hauptprogramm zurück, ohne eine Aktion durchzuführen. Das Löschen des Eingabepuffers lohnt sich natürlich nur dann, wenn die Tastatur das Standard-Eingabegerät ist.

Wenn Sie die Funktion 0Chex aufrufen, muß das AH-Register den Wert 0Chex und AL die Nummer der aufzurufenden Funktion beinhalten. Nach der Rückkehr aus der Funktion enthält das Register AL den ASCII-Code des eingelesenen Zeichens. Die Ein- und Aussprungsangaben entnehmen Sie den entsprechenden Funktionen (01hex, 06hex, 07hex, 08hex, 0Ahex).

Funktion 0Dhex: Dateipuffer leerschreiben auf Ausgabegerät

Die Funktion 0Dhex schreibt alle in einem MS-DOS-internen Dateipuffer gespeicherten Daten auf das gewünschte Ausgabegerät. In jedem Fall gibt die Prozedur die Zeichen als ASCII-Codes aus. Achten Sie bitte darauf, daß die Funktion keine Datei schließt und keinen Datenkanal freigibt.

Rufen Sie die Funktion 0Dhex auf, dann muß das AH-Register den Wert 0Dhex beinhalten. Die Funktion löst die gestellte Aufgabe ohne die Angabe weiterer Parameter. Nach der Rückkehr aus der Routine enthalten alle Register einschließlich des Kennzeichenregisters ihre ursprünglichen Werte.

Funktion 0Ehex: das aktuelle Laufwerk auswählen

Mit der Funktion 0Ehex des MS-DOS-Interrupts 21hex wählen Sie das aktuelle Laufwerk aus. Alle weiteren Dateizugriffe, bei denen keine Laufwerkangabe spezifiziert ist, erfolgen über das in dieser Funktion angegebene Standardlaufwerk. Sie übergeben die Laufwerknummer im AL-Register, das die gewünschte logische Einheit präsentiert. Dabei steht »0« für »A:«, 1 für »B:«, »2« für »C:« und so weiter. Nach dem Aufruf schreibt die Funktion alle geöffneten Dateien auf den Datenträger.

Sie rufen die Funktion 0Ehex korrekt auf, wenn das AH-Register den Wert 0Ehex enthält. Nachdem das Betriebssystem die Routine fehlerfrei ausgeführt hat, sind mit einer Ausnahme alle Register einschließlich des Kennzeichenregisters unverändert. Das Re-

gister AL enthält die Anzahl der verfügbaren logischen Laufwerke. Aber berücksichtigen Sie bei weiteren Programmbefehlen bitte nicht diese Betriebssystemangabe, da sie in einigen Fällen zu ungenau ist. Der Wert in AL bezieht nämlich auch RAM-Disks und die Partition der Festplatte in seine Berechnungen mit ein.

Bei den folgenden Dateioperationen können Sie Wildcards benutzen (»?*«), um den Dateinamen anzugeben.

Funktion 0Fhex: versucht eine Datei zu öffnen

Mit der Funktion 0Fhex versuchen Sie, eine Datei zu öffnen. Als Ergebnis erhalten Sie im AL-Register einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Findet es zum Zeitpunkt des Funktionsaufrufs einen Eintrag im Inhaltsverzeichnis, speichert das Betriebssystem im Register AL den Wert 0, und die Datei ist geöffnet. Die Funktion setzt dann die Datensatzlänge auf 128 Byte und trägt im FCB die Systemzeiten der ursprünglichen Datei ein. Bei sequentiellen Dateien setzen Sie die sequentielle Datensatzlänge (Offset: 20hex), bei relativen Dateien die relative Datensatzlänge (Offset: 21hex). Wenn kein Eintrag vorhanden ist oder die Datei eines der Attribute »versteckt« oder »System« trägt, entspricht der Inhalt von AL der Zahl 255.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 0Fhex enthalten und die Registerkombination »DS:DX« auf einen unbenutzten FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, informiert das AL-Register darüber, ob die Datei im Inhaltsverzeichnis eingetragen ist oder nicht.

Funktion 10hex: versucht eine Datei zu schließen

Mit der Funktion 10hex schließen Sie eine Datei. Als Ergebnis erhalten Sie im AL-Register einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Aktion erfolgreich durchgeführt hat. Findet es zum Zeitpunkt des Funktionsaufrufs den Namen der Datei im Inhaltsverzeichnis, dann legt das Betriebssystem im Register AL den Wert 0 ab und schließt die Datei. Die Funktion vergleicht die Systemzeiten im spezifizierten FCB mit der ursprünglichen Datei und paßt sie bei Bedarf an. Wenn kein Eintrag vorhanden ist, dann erhält AL den Wert 255.

Um eine Datei zu schließen, muß das AH-Register den Wert 10hex beinhalten und die Registerkombination »DS:DX« auf den gewünschten FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, infor-

miert das AL-Register darüber, ob die Datei im Inhaltsverzeichnis eingetragen ist oder nicht.

Funktion 11hex: Ersten Eintrag im Inhaltsverzeichnis suchen

Die Funktion 11hex sucht den ersten Eintrag im Inhaltsverzeichnis des aktuellen Verzeichnisses. Als Ergebnis erhalten Sie im AL-Register einen Wert, der Sie informiert, ob das Betriebssystem die Aktion erfolgreich beendet hat. Findet es zum Zeitpunkt des Funktionsaufrufs einen Eintrag im Inhaltsverzeichnis, dann speichert das Betriebssystem im Register AL den Wert 0 und legt im FCB der ungeöffneten Datei den Dateinamen ab. Handelt es sich um einen nicht erweiterten FCB, dann enthält das erste Byte die Laufwerkangabe, die folgenden 32 Byte enthalten den Eintrag des Inhaltsverzeichnisses. Ist der FCB hingegen erweitert, dann speichert das Betriebssystem zusätzlich vor dem eigentlichen Eintrag eine Zeichenfolge, die aus dem Wert FFhex und fünf Nullen besteht und das Attributbyte des Dateieintrags abschließt. Wenn kein Eintrag vorhanden ist, enthält AL die Zahl 255.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 11hex enthalten und die Registerkombination »DS:DX« auf den ungeöffneten FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, informiert das AL-Register darüber, ob die Datei im Inhaltsverzeichnis eingetragen ist oder nicht. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 12hex: Nächsten Eintrag im Inhaltsverzeichnis suchen

Die Funktion 12hex sucht die nächste Datei im Inhaltsverzeichnis des aktuellen Verzeichnisses, nachdem Sie mit Funktion 11hex den ersten Eintrag gefunden haben. Als Ergebnis erhalten Sie im AL-Register einen Wert. Dieser informiert Sie, ob das Betriebssystem die Aktion erfolgreich beendet hat. Findet es zum Zeitpunkt des Funktionsaufrufs einen Eintrag im Inhaltsverzeichnis, dann speichert das Betriebssystem im Register AL den Wert 0 und legt im FCB der ungeöffneten Datei den Dateinamen ab. Handelt es sich um einen nicht erweiterten FCB, dann enthält das erste Byte die Laufwerkangabe, die folgenden 32 Byte beinhalten den Eintrag im Inhaltsverzeichnis. Ist der FCB hingegen ein erweiterter FCB, dann speichert das Betriebssystem zusätzlich vor dem eigentlichen Eintrag eine Zeichenfolge, die aus dem Wert FFhex und fünf Nullen besteht und das Attributbyte des Dateieintrags abschließt. Wenn die Datei nicht gefun-

den wurde oder kein weiterer Eintrag vorhanden ist, dann enthält AL die Zahl 255.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 12hex enthalten und die Registerkombination »DS:DX« auf den ungeöffneten FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, informiert das AL-Register darüber, ob die Datei im Inhaltsverzeichnis eingetragen ist oder nicht.

Funktion 13hex: eine oder mehrere Dateien löschen

Mit der Funktion 13hex löschen Sie eine oder mehrere Dateien. Als Ergebnis erhalten Sie im AL-Register einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Dateien gelöscht hat. Findet es zum Zeitpunkt des Funktionsaufrufs den Namen der Datei im Inhaltsverzeichnis, dann löscht das Betriebssystem die Datei und legt im Register AL den Wert 0 ab. Sie können mit der Funktion 13hex keine Verzeichnisse löschen. Wenn die Datei, die das Unterprogramm löschen soll, nicht im aktuellen Verzeichnis vorhanden ist, dann enthält AL den Wert FFhex.

Um eine Datei zu löschen, muß das AH-Register die Zahl 13hex beinhalten und die Registerkombination »DS:DX« auf den benötigten FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, informiert Sie das AL-Register darüber, ob die Datei im Inhaltsverzeichnis eingetragen ist oder nicht.

Die Funktion 13hex ist zwar grundsätzlich dokumentiert, weist aber eine undokumentierte Besonderheit auf: Sie löscht das gesamte Verzeichnis und alle zugehörigen Unterverzeichnisse, wenn der Dateiname nur Fragezeichen enthält und die Attribut-Bits 0 bis 4 gesetzt sind. Im Gegensatz zu den anderen Löschoperationen erhält das erste Byte der Namen der gelöschten Dateien den Wert 00hex (»Eintrag nie benutzt«) anstelle von E5hex.

Funktion 14hex: Datensatz in Dateipuffer einlesen

Die Funktion 14hex dient dazu, den folgenden Datensatz in den Dateipuffer einzulesen. Nach dem Aufruf der Funktion gibt das AL-Register an, ob das Betriebssystem die Aktion mit Erfolg durchgeführt hat. Wurde nach dem Funktionsaufruf ein Datensatz ohne Komplikationen eingelesen, enthält das Register AL den Wert 0. Wenn die Funktion das Dateiende erreicht hat, enthält AL die Zahl 1. Ist AL gleich 2, dann liegt der Datenpuffer (DTA = Disk Transfer Area) am Ende eines Segments, und der eingelesene Datensatz geht über das Segmentende hinaus. Beinhaltet AL eine 3, dann be-

findet sich am Dateiende nur der Teil eines Datensatzes, nicht aber ein kompletter Datensatz. In diesem Fall wird der Datenpuffer mit Null-Bytes auf die vorgegebene Länge aufgefüllt. Die Funktion setzt den Dateizeiger nach dem Einlesen der Informationen automatisch auf den nächsten Datensatz.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 14hex enthalten und die Registerkombination »DS:DX« auf den gewünschten FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode.

Funktion 15hex: Nächsten Datensatz schreiben

Die Funktion 15hex schreibt den nächsten im Dateipuffer enthaltenen Datensatz in eine Datei. Nach dem Aufruf der Funktion gibt das AL-Register an, ob das Betriebssystem die Aktion mit Erfolg durchgeführt hat. Hat es einen Datensatz ohne Schwierigkeiten in die Datei geschrieben, dann enthält AL den Wert 0. Ist der Datenträger vollgeschrieben, wird dem Register AL die Zahl 1 zugewiesen. Ist AL gleich 2, dann liegt der Datenpuffer am Ende eines Segments, und der geschriebene Datensatz geht über das Segmentende hinaus.

Wenn Sie diese Funktion anwählen, muß das AH-Register den Wert 15hex enthalten und die Registerkombination »DS:DX« auf den FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 16hex: Entwickeln einer Datei

Mit der Funktion 16hex erzeugen Sie eine Datei. Als Ergebnis erhalten Sie im AL-Register einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Findet es zum Zeitpunkt des Funktionsaufrufs einen Ein-

trag im Inhaltsverzeichnis, wird die Länge der bestehenden Datei auf 0 gesetzt. Existiert die Datei noch nicht, dann erzeugt die Funktion eine neue Datei. Auch in diesem Fall speichert das Betriebssystem im Register AL den Wert 0 und öffnet die Datei. Könnte die Routine die Datei nicht erzeugen, weil beispielsweise das Inhaltsverzeichnis vollgeschrieben ist, dann enthält AL die Zahl 255.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 16hex enthalten und die Registerkombination »DS:DX« auf einen FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, informiert das AL-Register darüber, ob die Datei erzeugt wurde oder nicht.

Funktion 17hex: eine oder mehrere Dateien umbenennen

Mit der Funktion 17hex benennen Sie eine oder mehrere Dateien um. Als Ergebnis erhalten Sie im AL-Register einen Wert, der angibt, ob der Vorgang erfolgreich abgeschlossen wurde. Findet das Betriebssystem zum Zeitpunkt des Funktionsaufrufs den Namen der zu verändernden Datei im Inhaltsverzeichnis, dann legt es im Register AL den Wert 0 ab und übernimmt den neuen Namen für die Datei. Die Funktion vergleicht die Systemzeiten im spezifizierten FCB mit der ursprünglichen Datei und paßt sie bei Bedarf an. Der FCB enthält neben dem ursprünglichen Dateinamen ab der 16. Stelle die Gerätebezeichnung und den neuen Namen der Datei. Benutzen Sie Wildcards, dann ändert die Prozedur den Namen einer Dateigruppe. Wenn die Funktion die Datei nicht gefunden hat oder der neue Name schon existiert, dann speichert sie die Zahl 255 in AL. Um eine Datei umzubenennen, muß das AH-Register den Wert 17hex beinhalten und die Registerkombination »DS:DX« auf den FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, informiert das AL-Register darüber, ob die Datei den neuen Namen erhalten hat oder nicht. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 18hex: Rückkehr ins aufrufende Programm

Die Funktion 18hex kehrt sofort wieder in das aufrufende Programm zurück. Sie ist unbenutzt und wurde wahrscheinlich aus Kompatibilitätsgründen zu CP/M implementiert.

Funktion 19hex: Laufwerkbezeichnung ermitteln

Mit der Funktion 19hex des MS-DOS-Interrupts 21hex ermitteln Sie die

Mediadeskriptorbyte	Diskettengröße (Zoll)	Kapazität
FFhex	5,25	320 KByte
FEhex	5,25	160 KByte
FDhex	5,25	360 KByte
FChex	5,25	180 KByte
FBhex	3,5	640 KByte
FAhex	3,5	320 KByte
F9hex	5,25	600 KByte
F8hex	3,5	720 KByte
F7hex	HD	unbekannt
F6hex	5,25	1,2 MByte
F5hex	3,5	1,44 MByte
F4hex	3,5	2,88 MByte

Tabelle 2. Das Mediadeskriptorbyte (MCB)

Gerätebezeichnung des aktuellen Laufwerks. Die Routine übergibt die Laufwerksnummer im Register AL, das die gewünschte logische Einheit präsentiert. Dabei steht »0« für »A:«, »1« für »B:«, »2« für »C:« und so weiter. Sie wählen die Funktion 19hex richtig an, wenn das AH-Register den Wert 19hex enthält. Nach der Rückkehr aus der Funktion sind mit einer Ausnahme alle Register einschließlich des Kennzeichenregisters unverändert. Das Register AL enthält die Geräte- nummer des aktuellen Laufwerks.

Funktion 1Ahex: Neue Adresse des Datenpuffers bestimmen

Die Funktion 1Ahex legt die neue Adresse des Datenpuffers fest. Das Betriebssystem bearbeitet die Funktion ausschließlich dann korrekt, wenn die Adresseingabe im erlaubten Bereich liegt. Die neue Adresse des Eingabepuffers übergeben Sie mit dem Zeiger »DS:DX« an das Unterprogramm. Unter normalen Umständen beginnt der Datenpuffer ab Adresse 80hex des PSP. Eine Verlegung ist sinnvoll, wenn der Puffer eine Länge erhalten soll, die 128 Bytes übertrifft. Mit der Funktion 2Fhex ermitteln Sie die aktuelle Adresse des Datenpuffers.

Um die Funktion 1Ahex aufzurufen, muß das AH-Register den Wert 1Ahex beinhalten. Die Funktion verändert keinen Registerwert.

Funktion 1Bhex: Daten des aktuellen Laufwerks ermitteln

Mit der Funktion 1Bhex des MS-DOS-Interrupts 21hex ermitteln Sie Daten über das aktuelle Laufwerk. Die Routine übergibt die Anzahl der Sektoren pro Cluster im Register AL, die Gesamtanzahl der Cluster ist im Register DX abgelegt. Die Registerkombination DS:BX zeigt auf den Mediadeskriptor (Tabelle 2), der den physikalischen Aufbau des Datenträgers bestimmt. Dabei steht der Wert

- F8hex für eine Festplatte,
- F9hex für eine zweiseitige Diskette mit 15 Sektoren pro Spur,
- FChex für eine einseitige Diskette mit 9 Sektoren pro Spur,
- FDhex für eine zweiseitige Diskette mit 9 Sektoren pro Spur,
- FEhex für eine einseitige Diskette mit 8 Sektoren pro Spur und
- FFhex für eine zweiseitige Diskette mit 8 Sektoren pro Spur.

Sie wählen die Funktion 1Bhex richtig an, wenn das AH-Register den Wert 1Bhex enthält.

Funktion 1Chex: Daten eines bestimmten Laufwerks ermitteln

Die Funktion 1Chex des MS-DOS-Interrupts 21hex ermittelt Daten über ein bestimmtes Laufwerk. Vor dem Aufruf der Prozedur enthält DL die Laufwerk-

nummer. Dabei steht »0« für »Standard«, »1« für »A:«, »2« für »B:«, »3« für »C:« und so weiter. Die Routine übergibt die Anzahl der Sektoren pro Cluster im Register AL, die Gesamtanzahl der Cluster ist im Register DX und die Anzahl der Bytes pro Cluster in CX abgelegt. Die Registerkombination »DS:BX« zeigt auf den Media-Deskriptor, der den physikalischen Aufbau des Datenträgers bestimmt. Dabei steht der Wert

- F8hex für eine Festplatte,
- F9hex für eine zweiseitige Diskette mit 15 Sektoren pro Spur,
- FChex für eine einseitige Diskette mit 9 Sektoren pro Spur,
- FDhex für eine zweiseitige Diskette mit 9 Sektoren pro Spur,
- FEhex für eine einseitige Diskette mit 8 Sektoren pro Spur und
- FFhex für eine zweiseitige Diskette mit 8 Sektoren pro Spur.

Wenn die angegebene Laufwerksnummer ungültig ist, dann erhält das Register AL den Wert FFhex.

Sie wählen die Funktion 1Bhex korrekt an, wenn das AH-Register den Wert 1Bhex enthält.

Funktion 1Dhex: Rückkehr ins aufrufende Programm

Die Funktion 1Dhex kehrt nach dem Aufruf in das aufrufende Programm zurück. Sie ist unbenutzt und wurde offenbar aus Kompatibilitätsgründen zu CP/M aufgenommen.

Funktion 1Ehex: Kontrolle an das Hauptprogramm übergeben

Der Aufruf der Funktion 1Ehex führt einzig und allein dazu, daß das Hauptprogramm unmittelbar wieder die Kontrolle erhält. Auch diese Funktion haben die Programmierer vermutlich aus Kompatibilitätsgründen zu CP/M integriert.

Funktion 1Fhex: Parametertabelle des aktuellen Laufwerks

Die Funktion 1Fhex übergibt als Ergebnis die Adresse des DPB von dem aktuellen Laufwerk. Die Tabelle 3 zeigt den Aufbau der Parametertabelle. Der Index »DS:BX« zeigt nach dem Aufruf auf den Parameterblock. Wenn das Betriebssystem die Operation fehlerfrei ausgeführt hat, weist das Register AL den Wert 0 auf, ansonsten FFhex. Sie wählen die Funktion 1Fhex richtig an, wenn das AH-Register den Wert 1Fhex enthält.

Die Funktion entspricht in den Grundlagen der undokumentierten Funktion 32hex.

Funktion 20hex: Rückkehr ins aufrufende Programm

Die Funktion 20hex kehrt sofort wieder in das aufrufende Programm zu-

rück. Sie ist unbenutzt und wurde aus Kompatibilitätsgründen zu CP/M implementiert.

Funktion 21hex: Wahlfreien Datensatz einlesen

Die Funktion 21hex dient dazu, einen Datensatz in den Dateipuffer einzulesen. Nach dem Aufruf der Funktion gibt das AL-Register an, ob das Betriebssystem die Aktion mit Erfolg durchgeführt hat. Hat es nach dem Funktionsaufruf einen Datensatz ohne Komplikationen eingelesen, enthält das Register AL den Wert 0. Wenn die Funktion das Dateende erreicht hat, beinhaltet AL die Zahl 1. Ist AL gleich 2, dann liegt der Datenpuffer am Ende eines Segments, und der eingelesene Datensatz geht über das Segmentende hinaus.

Wenn in AL der Wert 3 abgelegt ist, befindet sich am Dateende nur ein Teil des Datensatzes. In diesem Fall wird der Datenpuffer mit Null-Bytes auf die vorgegebene Länge aufgefüllt. Vor dem Aufruf der Funktion speichern Sie im FCB die Position des einzulesenden Datenblocks.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 21hex enthalten und die Registerkombination »DS:DX« auf den gewünschten FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 22hex: Datensatz wahlfrei in Ausgabepuffer schreiben

Die Funktion 22hex schreibt den Datensatz einer Datei in den Ausgabepuffer.

Nach dem Aufruf der Funktion gibt das AL-Register an, ob das Betriebssystem die Aktion mit Erfolg durchgeführt hat. Hat es einen Datensatz ohne Schwierigkeiten in die Datei geschrieben, enthält AL den Wert 0. Ist der Datenträger vollgeschrieben, ist im Register AL die Zahl 1 gespeichert. Ist AL gleich 2, dann liegt der Datenpuffer am Ende eines Segments, und der geschriebene Datensatz geht über das Segmentende hinaus. Vor dem Aufruf der Funktion legen Sie im FCB die Position des zu schreibenden Datenblocks ab.

Diese Funktion können Sie nur anwählen, wenn das AH-Register den Wert 22hex enthält und die Registerkombination »DS:DX« auf den FCB zeigt. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 23hex: Dateigröße ermitteln

Mit der Funktion 23hex ermitteln Sie die Größe einer Datei. Als Ergebnis erhalten Sie im AL-Register einen Wert, der angibt, ob der Vorgang erfolgreich abgeschlossen wurde. Findet das Betriebssystem zum Zeitpunkt des Funktionsaufrufs die Datei im aktuellen Verzeichnis, dann legt das Betriebssystem im Register AL den Wert 0 ab und speichert die Anzahl der Datensätze ab der Adresse 21hex im FCB. Vor dem Aufruf der Routine tragen Sie die Datensatzgröße in den FCB ein. Wenn die Funktion die Datei nicht gefunden hat oder der neue Name schon existiert, dann enthält AL die Zahl 255.

Damit Sie die Dateigröße feststellen können, muß das AH-Register den Wert 23hex beinhalten und die Registerkombination DS:DX auf den FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, informiert das AL-Register darüber, ob die Datei gefunden wurde oder nicht. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 24hex: Datensatznummer auf den aktuellen Wert setzen

Die Funktion 24hex dient dazu, die Datensatznummer im FCB für den Dateizugriff auf den aktuellen Dateizeiger zu setzen.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 24hex enthalten und die Registerkombination »DS:DX« auf den gewünschten FCB zeigen. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 25hex: Interrupt-Vektor neu setzen

Die Funktion 25hex setzt einen Interrupt-Vektor auf eine neue Adresse. Um Probleme zu vermeiden, sollten Sie niemals Änderungen direkt in den Speicher schreiben. Bevor Sie die Adresse eines Vektors ändern, ist es sinnvoll, die ursprüngliche Interrupt-Adresse mit der Funktion 35hex auszulernen und für eine spätere Verwendung zu speichern.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 25hex enthalten und die Registerkombination »DS:DX« auf die neue Interrupt-Adresse zeigen. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Adresse (Offset)	Datentyp	Funktion
00hex	Byte	Laufwerks (0=A, 1=B, ...)
01hex	Byte	Gerätenummer für Treiber, die mehr als ein Laufwerk verwalten
02hex	Word	Bytes/Sektor
04hex	Byte	Sektoren/Cluster, um 1 verringert
05hex	Byte	Sektoren/Cluster in der Form 2^n ; n=0: 1 Sektor/Cluster, n=1: 2 Sektoren/Cluster, n=2: 4 Sektoren/Cluster, ...
06hex	Word	Anzahl reservierter Sektoren
08hex	Byte	Anzahl FATs
09hex	Word	maximal mögliche Einträge im Hauptverzeichnis
0Bhex	Word	erster Datensektor
0Fhex	Word	Anzahl Sektoren/FAT
11hex	Word	erster Sektor im Hauptverzeichnis
13hex	DWord	Gerätetreiberadresse
7hex	Byte	Mediadeskriptorbyte
18hex	Byte	Flag: veranlaßt DOS, Datenträger zu überprüfen, wenn=00hex nächste Drive-Parameterblock-Adresse; FFFFhex=Ende
19hex	DWord	Nummer des zuletzt beschriebenen Clusters
1Dhex	Word	Nummer des zuletzt beschriebenen Clusters
1Fhex	Word	Anzahl freier Cluster

Tabelle 3. Aufbau des Drive-Parameterblocks (DPB)

Funktion 26hex: PSP erzeugen

Die Funktion 26hex versetzt Sie in die Lage, einen neuen PSP zu erzeugen. Das Betriebssystem erreicht das Ergebnis dadurch, daß es den PSP des ausgeführten Programms an die vorgegebene Adresse kopiert. Sie übergeben die Segmentadresse des neuen PSP im Register DX. Als Offset setzt die Routine automatisch den Wert 0 ein.

Nach der Rückkehr aus der Funktion enthalten alle Register einschließlich des Kennzeichenregisters ihre ursprünglichen Werte.

Funktion 27hex: Sequentielle Datensätze einlesen

Die Funktion 27hex dient dazu, mehrere Datensätze einer sequentiellen Datei in den Dateipuffer einzulesen. Nach dem Aufruf der Funktion gibt das AL-Register an, ob das Betriebssystem die Aktion mit Erfolg durchgeführt hat. Wurden nach dem Funktionsaufruf die Datensätze ohne Komplikationen eingelesen, dann enthält das Register AL den Wert 0. Wenn die Funktion das Dateiende erreicht hat, beinhaltet AL die Zahl 1. Ist AL gleich 2, dann liegt der Datenpuffer (DTA) am Ende eines Segments, und die eingelesenen Datensätze gehen über das Segmentende hinaus. Wenn in AL der

Wert 3 abgelegt ist, befindet sich am Dateiende nur ein Teil eines Datensatzes. In diesem Fall wird der Datenpuffer mit Null-Bytes auf die vorgegebene Länge aufgefüllt. Vor dem Aufruf der Funktion speichern Sie im FCB die Position des ersten einzulesenden Datenblocks.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 27hex enthalten und die Registerkombination »DS:DX« auf den gewünschten FCB zeigen. Das Register CX gibt die Anzahl der einzulesenden Datensätze an. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode und CX die Anzahl der gelesenen Datensätze. Der Inhalt der Register AH, BX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 28hex: Mehrere wahlfreie Datensätze schreiben

Die Funktion 28hex schreibt mehrere Datensätze einer sequentiellen Datei in den Ausgabepuffer. Nach Beendigung der Funktion gibt das AL-Register an, ob das Betriebssystem die Aktion mit Erfolg durchgeführt hat. Wurden die Datensätze ohne Schwierigkeiten in die Datei geschrieben, dann enthält AL den Wert 0. Wenn der Datenträger vollgeschrieben ist, dann ist im Register AL die Zahl 1 gespeichert. Ist AL gleich 2, dann liegt der Datenpuffer am Ende eines Segments, und die geschriebenen Datensätze gehen über das Segmentende hinaus. Vor dem Aufruf der Funktion legen Sie im FCB die Position des zu schreibenden Datenblocks ab.

Wählen Sie diese Funktion an, muß das AH-Register den Wert 28hex und CX die Anzahl der zu schreibenden Datensätze enthalten sowie die Registerkombination »DS:DX« auf den FCB zeigen. Hat das Betriebssystem die Funktion ausgeführt, enthält das Register AL den Fehlercode und CX die Anzahl der geschriebenen Datensätze. Der Inhalt der Register AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 29hex: Dateiname in FCB schreiben

Die Funktion 29hex dient dazu, einen Dateinamen in den FCB zu schreiben. Der Dateiname besteht aus Name und Erweiterung und kann jedes ASCII-Zeichen darstellen. Er muß mit dem Wert 0 abgeschlossen sein (dieses Format nennt man auch ASCII-Format). Beim Aufruf dieser Funktion enthält das AH-Register den Wert 29hex, und die Registerkombination »DS:SI« zeigt auf den Dateinamen. Die Position »ES:DI« ist der Beginn des FCB einer ungeöffneten Datei. AL kontrolliert die

Übertragung des Dateinamens. Ist Bit 0 gleich 1, dann ignoriert die Funktion die im Dateinamen enthaltenen Trennzeichen, andererseits kehrt sie beim Auftreten dieser Zeichen ins Hauptprogramm zurück. Bit 1 bestimmt, ob das aktuelle (Bit gesetzt) oder das Standardlaufwerk (Bit gelöscht) angesteuert wird. Bit 2 ersetzt den Dateinamen durch acht Leerzeichen, wenn er unzulässige Zeichen enthält und das Bit gelöscht ist. Andernfalls ändert das Betriebssystem den Namen nicht. Bit 3 ersetzt die Erweiterung durch drei Leerzeichen, wenn sie unzulässige Zeichen enthält und das Bit gelöscht ist. Ist Bit 3 gesetzt, ändert das Betriebssystem die Erweiterung nicht. Sobald MS-DOS die Funktion ausgeführt hat, enthält das Register AL Informationen über die verwendeten Daten. Ist kein Teil des Dateinamens eine Wildcard, dann erhält das Register AL den Wert 0. Besteht ein Teil des Dateinamens aus Wildcards, dann beinhaltet AL die Zahl 1. Ist AL gleich 255, dann ist die angegebene Gerätebezeichnung ungültig. Die Registerkombination DS:SI zeigt auf das erste Byte nach dem Dateinamen. Der Inhalt der Register AH, BX, CX, DX, BP, CS, SS und der des Kennzeichenregisters ändern sich nicht.

Funktion 2Ahex: ermittelt das aktuelle Datum

Mit der Funktion 2Ahex des MS-DOS-Interrupts 21hex stellen Sie das aktuelle Datum fest. Sie erhalten im Register AL eine Zahl, die den Wochentag anzeigt. Dabei steht »0« für »Sonntag«, »1« für »Montag«, »2« für »Dienstag« und so weiter. CX übergibt das aktuelle Jahr, DH den Monat und DL den Tag. Die Unterroutine benutzt den UHrtreiber, um das aktuelle Datum zu ermitteln.

Sie wählen die Funktion 2Ahex richtig an, wenn das AH-Register den Wert 2Ahex enthält. Der Inhalt der Register AH, BX, SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 2Bhex: Datum neu bestimmen

Sie setzen die Funktion 2Bhex ein, um das aktuelle Datum neu zu bestimmen. Im Register CX übergeben Sie das aktuelle Jahr, in DH den Monat und in DL den Tag. Die Unterroutine benutzt den UHrtreiber, um das aktuelle Datum zu setzen. Sind die übergebenen Parameter sinnvoll, löscht das Betriebssystem das alte Datum, aktualisiert es und legt im Register AL den Wert 0 ab. Wenn die Parameter, die Sie an das Unterprogramm übergeben, nicht gültig sind, dann erhält AL die Zahl FFhex.

Funktion 2Chex: ist für die Zeitan-gabe verantwortlich

Die Funktion 2Chex gibt Ihnen die genaue Zeit an. Rufen Sie sie auf, erhalten Sie im Register CH eine Zahl, die die Stunde anzeigt. CL übergibt die aktuelle Minute, DH die Sekunde und DL die Hundertstelsekunde. Die Unterroutine ermittelt die aktuelle Uhrzeit mit dem UHrtreiber.

Sie wählen die Funktion 2Chex richtig an, wenn das AH-Register den Wert 2Chex enthält.

Funktion 2Dhex: setzt die aktuelle Zeit neu

Im Gegensatz dazu setzen Sie die Funktion 2Dhex ein, um die aktuelle Zeit neu zu setzen. Im Register CH übergeben Sie die aktuelle Stunde, in CL die Minute, in DH die Sekunde und in DL die Hundertstelsekunde. Die Unterroutine benutzt den UHrtreiber, um die aktuelle Uhrzeit zu setzen. Sind die übergebenen Parameter sinnvoll, dann aktualisiert das Betriebssystem die alte Uhrzeit und legt im Register AL den Wert 0 ab. Wenn die Parameter, die Sie an das Unterprogramm übergeben, ungültig sind, dann entlt AL die Zahl FFhex.

Sie wählen die Funktion 2Dhex richtig an, wenn das AH-Register den Wert 2Dhex enthält.

Funktion 2Ehex: dient zum Setzen des Verify-Flag

Üblicherweise überprüft das Betriebssystem alle Daten, die es auf einen Datenträger geschrieben hat. Diese Vorgehensweise ist zwar für die Datensicherheit bedeutsam, aber sie benötigt Zeit. Benutzen Sie Disketten guter Qualität, dann entstehen keine Probleme, wenn Sie die Datenüberprüfung ausschalten. Um das Verify-Flag zu setzen, das heißt, um die geschriebenen Daten zu überprüfen, speichern Sie vor dem Funktionsaufruf den Wert 1 im AL-Register, sonst die Zahl 0.

Wenn Sie die Funktion 2Ehex aufrufen, muß das AH-Register den Wert 2Ehex beinhalten. Nach der Rückkehr aus der Funktion enthalten alle Register einschließlich des Kennzeichenregisters ihre ursprünglichen Werte.

Funktion 2Fhex: Adresse des Datenpuffers ermitteln

Die Funktion 2Fhex stellt die Adresse des Datenpuffers fest. Beachten Sie bitte, daß die angegebenen Werte nichts über die Größe des Datenpuffers aussagen. Die Adresse des Eingabepuffers übergibt das Unterprogramm mit dem Zeiger »ES:BX« an die Haupt-routine. Unter normalen Umständen beginnt der Datenpuffer ab Adresse 80hex des PSP. Eine Abfrage der Posi-

tion ist sinnvoll, wenn der Anwender den Puffer verlegt hat, um eine Länge zu erhalten, die 128 Byte übertrifft. Mit der Funktion 1Ahex verändern Sie die aktuelle Adresse des Datenpuffers. Um die Funktion 2Fhex aufzurufen, muß das AH-Register den Wert 2Fhex beinhalten.

Funktion 30hex: MS-DOS-Versionsnummer holen

Mit der Funktion 30hex des MS-DOS-Interrupts 21hex ermitteln Sie die Versionsnummer des Betriebssystems. Nach dem Aufruf der Prozedur übergibt die Routine im Register AH die untergeordnete und in AL die übergeordnete Versionsnummer. Bei der MS-DOS-Version 3.2 wird beispielsweise die Zahl 3 im AL-Register und die Zahl 20 in AH abgelegt. Die Registerkombination enthält eine 24-Bit-Seriennummer. Das Register BH gibt über die Registriernummer des OEM Aufschluß. Sie wählen die Funktion 30hex richtig an, wenn das AH-Register den Wert 30hex enthält.

Funktion 31hex: Programm unterbrechen

Die Funktion 31hex des Interrupts 21hex (»Keep Process«) dient dazu, den Ablauf des aktuellen Programms zu unterbrechen und in das aufrufende Programm zurückzukehren. Das ist in vielen Fällen die Kommandoebene des Betriebssystems MS-DOS. Im Gegensatz zu anderen Funktionen stellt der PC den Arbeitsspeicher, den das beendete Programm belegt, nach dem Funktionsaufruf nicht zur freien Verfügung. Im Unterschied zum Interrupt 27hex, der grundsätzlich die gleiche Aufgabe erfüllt, erlaubt es die Funktion 31hex auch, Speicherplatz über 64 KByte reservieren.

Beim Aufruf der Funktion 31hex muß das AH-Register den Wert 31hex beinhalten. Im AL-Register legen Sie den Rückkehrcode ab, den Sie durch Aufruf der Funktion 4Dhex erhalten. Im Register DX teilen Sie der Funktion mit, wie viele 16-Byte-Paragrafen das aufgerufene Programm einschließt, des PSP belegt.

Funktion 32hex: Adresse der Laufwerk-Parametertabelle

Die Funktion 32hex entspricht von der Funktionsweise her der undokumentierten Funktion 1Fhex. Der einzige Unterschied besteht darin, daß DL vor dem Aufruf der Prozedur die Laufwerknummer enthält. Dabei steht »0« für »Standard«, »1« für »A:«, »2« für »B:«, »3« für »C:« und so weiter. Der Index »DS:BX« zeigt auch hier nach dem Aufruf auf den Parameterblock. Wenn das Betriebssystem die Operation fehlerfrei ausgeführt hat, weist das Regi-

ster AL den Wert 0 auf, bei fehlerhafter Laufwerkangabe den Wert FFhex.

Funktion 33hex: liest und setzt das Break-Flag

Unter normalen Umständen überprüft das Betriebssystem das Drücken der Tastenkombination [Strg-C] nur in bestimmten Fällen. Das ist beispielsweise bei fast allen Ein- und Ausgaben des Standardgeräts der Fall. Diese Vorgehensweise spart zwar viel Zeit, aber es stehen nicht mehr so viele Unterbrechungsvarianten zur Verfügung. Wenn Sie die Interrupts bei Diskettenoperationen unterbrechen wollen (beispielsweise bei der Fehlersuche in Assembler-Programmen), aktivieren Sie die Abfrage von [Strg-C]. Um das Break-Flag zu setzen, speichern Sie vor dem Funktionsaufruf den Wert 1 im AL-Register. In diesem Fall gibt das Register DL an, ob der Test auf [Strg-C] bei jedem Funktionsaufruf (DL gleich 1) oder nur bei der Zeichenein- und -ausgabe (DL gleich 0) erfolgt. Nach der Rückkehr aus der Funktion enthalten alle Register einschließlich des Kennzeichenregisters ihre ursprünglichen Werte. Befindet sich im Register AL hingegen die Zahl 0, dann liest die Funktion den Status des Break-Flags. Wenn die Tastenkombination [Strg-C] bei jedem Funktionsaufruf überprüft wird, dann gibt die Routine den Wert 1 im DL-Register zurück, andernfalls die Zahl 0. Um die Funktion 33hex aufzurufen, muß das AH-Register den Wert 33hex beinhalten.

Funktion 34hex: ermittelt den Zeiger auf Statusbyte

Die Funktion 34hex ermittelt ab MS-DOS 2.0 einen Zeiger auf ein Statusby-

te. Enthält dieses Byte (InDOS = kritischer Abschnitt) einen Wert ungleich 0, bedeutet das, daß gerade eine MS-DOS-Funktion des Interrupts 21hex ausgeführt wird. Der Inhalt 0 zeigt an, daß im Moment keine MS-DOS-Funktion aktiv ist. Dieses Statusbyte ist vor allem dann von Bedeutung, wenn man vorhat, ein TSR-Programm (Terminate and Stay Resident) zu starten. Um Konflikte mit dem Betriebssystem zu vermeiden, sollte in diesem Fall das InDOS-Byte gelöscht sein.

Um die Funktion korrekt aufzurufen, erhält das AH-Register den Wert 43hex. Bei der Rückkehr gibt der Zeiger in den Registern »ES:BX« die Position des In-DOS-Bytes an.

Funktion 35hex: Interrupt-Vektor einlesen

Die Funktion 35hex liest den Inhalt eines Interrupt-Vektors ein. Um Probleme zu vermeiden, sollten Sie niemals Änderungen direkt aus dem Speicher lesen. Möchten Sie die Adresse eines Vektors ändern, ist es sinnvoll, die ursprüngliche Interrupt-Adresse mit dieser Funktion auszulesen und für die spätere Verwendung zu speichern. Die Routine übergibt in der Registerkombination »ES:BX« den Zeiger des Interrupts.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 35hex und das Register AL die Interrupt-Nummer enthalten.

Funktion 36hex: Laufwerkkapazität ermitteln

Die Funktion 36hex des MS-DOS-Interrupts 21hex ermittelt die Kapazität eines bestimmten Laufwerks. Vor dem Aufruf der Prozedur enthält DL die

Laufwerknummer. Dabei steht »0« für »Standard«, »1« für »A:«, »2« für »B:«, »3« für »C:« und so weiter. Die Routine übergibt die Anzahl der Sektoren pro Cluster im Register AX, die Gesamtanzahl der Cluster ist im Register DX und die Anzahl der Bytes pro Cluster in CX abgelegt. Das Register BX teilt Ihnen mit, wie viele Cluster verfügbar sind. Wenn die angegebene Laufwerknummer ungültig ist, enthält das Register AX den Wert FFFhex. Sie wählen die Funktion 36hex richtig an, wenn das AH-Register den Wert 36hex enthält. Der Inhalt der Register SI, DI, BP, CS, DS, SS, ES und der des Kennzeichenregisters ändern sich nicht.

Funktion 37hex: Befehlsschalter bearbeiten

Die Funktion 37hex mit zwei Unterfunktionen dient ab MS-DOS 2.0 dazu, den Befehlsschalter zu bearbeiten. Damit Sie die Funktion korrekt aufrufen, muß AH den Wert 37hex enthalten. Nach der Rückkehr aus der Routine ist das AL-Register auf FFhex gesetzt, wenn das Betriebssystem die jeweilige Unterfunktion nicht unterstützt. Die Unterfunktion 00hex (AL = 00hex) liest das Zeichen, das aktuell als Befehlsschalter fungiert. Wenn MS-DOS die Unterfunktion unterstützt, steht nach dem Aufruf im DL-Register das Trennzeichen. Als Standard erhalten Sie den Wert 47 (ASCII-Zeichen »/«). Die Unterfunktion 01hex (AL = 01hex) setzt hingegen einen neuen Befehlsschalter, dessen ASCII-Code vor dem Aufruf in DL abgelegt sein muß.

Sie steuern die Funktion 37hex richtig an, wenn AH den Wert 37hex und AL den Wert der Unterfunktion enthält.

(Markus Zietlow/uk)

Anwendungen gesucht

Haben Sie eine schnelle und komfortable Textverarbeitung geschrieben? Oder eine Datenbank mit frei wählbaren Eingabemasken? Eventuell ein Zeichenprogramm? Wenn Ihr Programm mit professionellen Programmen konkurrieren kann, ist es mit Sicherheit zu umfangreich, um als Listing in der DOS International zu erscheinen. Zögern Sie deshalb nicht, schicken Sie uns unverbindlich die aktuelle Version Ihres Programms und einen Ausdruck der Dokumentation. Eventuell wird Ihr Programm der Verkaufsschlager im Rahmen der neuen DOS-Software-Schiene.

Schicken Sie Ihre Unterlagen an folgende Adresse:

**Redaktion
DOS International,
Gruberstraße 46c
8011 Poing bei München**

Interrupt-Funktionen

Mit Xenix kompatibel

Die Xenix-kompatiblen Funktionen sind im Vergleich zu dem CP/M-kompatiblen wesentlich moderner. Handle-orientierter Datenzugriff und Netzwerkroutrinen bilden die Grundlagen für zeitgemäße Unterprogramme. Doch gerade in diesem Bereich gibt es neben den Standardfunktionen viele undokumentierte Prozeduren, deren Ausnutzung Ihnen erst einen Überblick über interne Systemvorgänge verschafft.

Aufgrund der Aufwärtskompatibilität ist es sinnvoll, die veralteten CP/M-kompatiblen Funktionen durch Xenix-kompatible zu ersetzen. Eine Aufstellung der vergleichbaren Funktionen finden Sie in Tabelle 1.

Der Aufruf einer Betriebssystemroutine ist leider nicht immer von Erfolg gekrönt. Aus diesem Grund informiert der Interrupt 21hex den Anwender, ob und was für ein Fehler aufgetreten ist. Die Fehlermeldungen handhaben die Xenix- und CP/M-kompatiblen Funktionen unterschiedlich. Während bei den CP/M-kompatiblen Funktionen das AL-Register durch den Wert FFhex das Auftreten eines Fehlers anzeigt, setzen die Xenix-kompatiblen Funktionen in diesem Fall das Carry-Statusbit des Kennzeichenregisters. Das jeweilige Unterprogramm gibt die Fehlernummer im AX-Register zurück. Aus Tabelle 2 ersehen Sie die verschiedenen Fehlercodes und ihre Bedeutung. Im Zusammenhang mit der Funktion 59hex werden Sie sich mit erweiterten Fehlercodes beschäftigen. Die Tabellen 3 bis 5 listen die Rückgabeparameter der Routine auf. Das Register AX beschreibt den Fehler, BH die Fehlerursache und CH die Fehlerquelle. Mit BL stellen Sie fest, welche Vorgehensweise Ihnen das Betriebssystem als Reaktion auf den Fehler vorschlägt. Der Inhalt des AX-Registers entspricht der Tabelle 2.

Wie Sie sehen, erhalten Sie mit dem MS-DOS Interrupt 21hex eine Betriebssystembibliothek, die alle wichtigen Funktionen beinhaltet, die für den Programmierer von Bedeutung sind. Fast alle modernen Programmiersprachen (Turbo Pascal, Quick Basic und C) verfügen über Kommandos, mit denen Sie den Interrupt 21hex aufrufen können und als Ergebnis den gesamten Registersatz erhalten. Der Interrupt 21hex ist ein leistungsfähiges Hilfsmittel, das die moderne Konzeption des Microsoft-Betriebssystems sinnvoll unterstützt.

Funktion 38hex: Länderspezifische Daten ermitteln und setzen

Ein großer Vorteil des PC-Systems ist, daß es sich problemlos an länderspezifische Anforderungen wie beispielsweise Sonderzeichen anpassen läßt. Mit dieser Funktion lesen beziehungsweise setzen Sie die länderspezifischen Systemparameter des Betriebssystems. Wenn Sie im DX-Register den Wert FFFFhex ablegen, stellen Sie das aktuelle Land ein. Diese Funktion versetzt Sie in die Lage, die länderspezifischen Parameter auch nach dem Bootvorgang zu verändern und einzulesen. Das Register AL enthält den Ländercode, der der internationalen Vorwahlnummer entspricht. Ist AL gleich 0, dann liest das Betriebssystem die Parameter des aktuellen Landes. Es gibt aber auch Ländercodes, die größer als 254 sind. In diesem Fall enthält AL den Wert 255 und BX den Ländercode. Nachdem das Betriebssystem die Rou-

tine ausgeführt hat, gibt das Carry-Statusbit Aufschluß über den Erfolg der Operation. Ist es gesetzt, dann war der landesspezifische Code ungültig. Hat die Funktion das Statusbit hingegen gelöscht, dann wurde das gewünschte Land eingestellt.

Wenn das Register DX einen Wert ungleich FFFFhex enthält, ermittelt das Betriebssystem die länderspezifischen Parameter. In den Puffer werden 34 Bytes geschrieben, die folgende Bedeutung haben: Bytes 0 und 1 enthalten das Datumsformat, Bytes 2 bis 6 die Währungsbezeichnung des jeweiligen Landes, Bytes 7 und 8 den ASCII-Code des Tausenderzeichens, Bytes 9 und 10 das Dezimalpunktzeichen, Bytes 11 und 12 das Datumstrennzeichen, Bytes 13 und 14 das Trennzeichen der Zeitangabe. Im 15. Byte befindet sich ein Bit-Feld, das das Währungsformat definiert. Das erste Bit bestimmt, ob das Währungssymbol vor (Bit 0 gleich 0) oder nach dem Wert (Bit 0 gleich 1) ausgegeben wird, das zweite, ob das Betriebssystem zwischen Währungssymbol und Wert ein Leerzeichen einsetzt (Bit 1 gleich 1). Byte 16 beinhaltet die Anzahl der Stellen, die bei den Währungsangaben hinter dem Komma berücksichtigt werden, Byte 17 das Format der Zeitangabe (0: 12-Stunden-Uhr; 1: 24-Stunden-Uhr), Bytes 18 bis 21 die Far-Adresse einer Routine, die das im AL-Register gespeicherte Zeichen in einen Großbuchstaben umwandelt sowie Bytes 22 bis 23 das Datentrennzeichen. Bytes 24 bis 33 sind reserviert. Der Inhalt der Register AX, BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Sie wählen die Funktion 38hex korrekt an, indem Sie im AH-Register den Wert 38hex ablegen.

Funktion 39hex: Erzeugen eines Verzeichnisses

Mit der Funktion 39hex versuchen Sie, ein Unterverzeichnis zu erzeugen. Als Ergebnis erhalten Sie im Carry-Statusbit einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Ist genug Platz auf dem Datenträger vorhanden und ist das angegebene Oberverzeichnis existent, dann löscht die Funktion das Carry-Statusbit und erzeugt das Unterverzeichnis. Konnte die Routine das Unterverzeichnis nicht erzeugen, weil beispielsweise das Inhaltsverzeichnis vollgeschrieben ist, dann setzt sie das Carry-Statusbit. Das AX-Register enthält in diesem Fall den Fehlercode, dessen Bedeutung Sie Tabelle 2 entnehmen. Beim Aufruf dieser Funktion muß das AH-Register den Wert 39hex enthalten und die Registerkombination DS:DX auf den Namen des Unterverzeichnisses

CP/M	Xenix
Funktion 00hex	Funktion 4Chex
Funktion 0Fhex	Funktion 3Dhex
Funktion 10hex	Funktion 3Ehex
Funktion 11hex	Funktion 4Ehex
Funktion 12hex	Funktion 4Fhex
Funktion 13hex	Funktion 41hex
Funktion 14hex	Funktion 3Fhex
Funktion 15hex	Funktion 3Dhex
Funktion 16hex	Funktionen 3Chex, 5Ahex und 5Bhex
Funktion 17hex	Funktion 56hex
Funktion 21hex	Funktion 3Fhex
Funktion 22hex	Funktion 40hex
Funktion 23hex	Funktion 42hex
Funktion 24hex	Funktion 42hex
Funktion 26hex	Funktion 4Bhex
Funktion 27hex	Funktion 3Fhex
Funktion 28hex	Funktion 40hex
Interrupt 20hex	Funktion 4Chex
Interrupt 27hex	Funktion 31hex

Tabelle 1. Die Xenix-kompatiblen Funktionen können Sie anstatt der veralteten CP/M-Aufrufe benutzen.

ses zeigen, der mit einem Null-Byte abgeschlossen ist. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und des Kennzeichenregisters ändert sich nicht.

Funktion 3Ahex: Unterverzeichnis löschen

Mit der Funktion 3Ahex löschen Sie ein Unterverzeichnis. Als Ergebnis erhalten Sie im Carry-Statusbit einen Wert, der Ihnen sagt, ob das Betriebssystem den Pfad gelöscht hat. Findet es zum Zeitpunkt des Funktionsaufrufs den Namen des Pfades im aktuellen Inhaltsverzeichnis, dann löscht das Betriebssystem das Verzeichnis und setzt das Carry-Statusbit auf den Wert 0. Wenn der Pfad, den das Unterprogramm löschen soll, nicht im aktuellen Verzeichnis vorhanden ist, dann setzt die Routine das Carry-Statusbit und gibt den Fehlercode in AX zurück.

Um ein Verzeichnis zu löschen, muß das AH-Register den Wert 3Ahex enthalten und die Registerkombination DS:DX auf den Namen des Unterverzeichnisses zeigen, der mit einem Null-Byte abgeschlossen ist. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und des Kennzeichenregisters bleibt unverändert.

Funktion 3Bhex: Wechseln des aktuellen Verzeichnisses

Mit der Funktion 3Bhex wechseln Sie das aktuelle Inhaltsverzeichnis. Als Ergebnis erhalten Sie im Carry-Statusbit einen Wert, der Sie darüber informiert, ob das Betriebssystem das Unterverzeichnis gewechselt hat. Ist das angegebene Oberverzeichnis existent, dann löscht die Funktion das Carry-Statusbit und wechselt das Unterverzeichnis. Konnte die Routine das Unterverzeichnis nicht wechseln, weil es beispielsweise das Inhaltsverzeichnis nicht gibt, dann setzt sie das Carry-Statusbit. Das AX-Register enthält in diesem Fall den Fehlercode.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 3Bhex enthalten und die Registerkombination DS:DX auf den Namen des neuen Unterverzeichnisses zeigen, der mit einem Null-Byte abgeschlossen ist. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und des Kennzeichenregisters ändert sich nicht.

Funktion 3Chex: Erzeugen einer Datei über Handle

Die Xenix-kompatiblen Funktionen wickeln den Datentransfer über sogenannte »Handles« ab. Ein Handle ist ein numerischer Wert, den das Betriebssystem einem Ein- beziehungsweise Ausgabegerät zuordnet. Der Zu-

Fehlercodes			
AX	Bedeutung		
01hex	ungültiger Funktionsaufruf	32hex	Netzwerkdienst wird nicht unterstützt
02hex	Datei nicht gefunden	33hex	Gegenstelle empfängt nicht
03hex	Pfad nicht gefunden	34hex	Name tritt zweimal im Netzwerk auf
04hex	zu viele Zugriffskanäle geöffnet	35hex	Name im Netzwerk nicht gefunden
05hex	Zugriff nicht erlaubt	36hex	Netzwerk aktiv
06hex	ungültiger Zugriffskanal	37hex	Netzwerkgerät existiert nicht mehr
07hex	Memory-Control-Block zerstört	38hex	Begrenzung eines BIOS-Kommandos im Netzwerk überschritten
08hex	zu wenig Speicherplatz	39hex	Hardwarefehler im Netzwerkadapter
09hex	ungültige Speicheradresse	3Ahex	falsche Anforderung im Netzwerk
0Ahex	ungültige Betriebssystemumgebung	3Bhex	unerwarteter Netzwerkfehler
0Bhex	ungültiges Format	3Chex	Adapter der Gegenstelle inkompatibel
0Chex	ungültige Zugriffsart	3Dhex	Drucker-Warteschlange voll
0Dhex	ungültige Daten	3Ehex	nicht genug Platz vorhanden, um Datei auszudrucken
0Ehex	reserviert	3Fhex	Druckdatei wurde entfernt
0Fhex	ungültige Laufwerkangabe	40hex	Name im Netzwerk wurde gelöscht
10hex	Verzeichnis kann nicht gelöscht werden	41hex	Zugriff nicht erlaubt
11hex	ungültiger Laufwerkwechsel	42hex	Typ des Netzwerktreibers falsch
12hex	keine weiteren Dateien vorhanden	43hex	Name im Netzwerk nicht gefunden
13hex	Diskette ist schreibgeschützt	44hex	Begrenzung eines Netzwerknamens überschritten
14hex	Fehler in der Disketteneinheit	45hex	zu viele Bios-Sitzungen im Netzwerk
15hex	Laufwerk nicht bereit	46hex	systembedingte Pause
16hex	ungültiges Diskettenkommando	47hex	Netzwerkdienst wird nicht akzeptiert
17hex	CRC-Fehler	48hex	Pause in der Druck-Laufwerks-umleitung
18hex	ungültige Länge bei einer Diskettenoperation	4Fhex	reserviert
19hex	Fehler beim Positionieren des Dateizeigers	50hex	Datei ist schon vorhanden
1Ahex	keine MS-DOS-Diskette	51hex	reserviert
1Bhex	Sektor nicht gefunden	52hex	Aktion nicht durchführbar
1Chex	kein Papier vorhanden	53hex	Fehler im Interrupt 24hex
1Dhex	Schreibfehler	54hex	falsche Anordnung
1Ehex	Lesefehler	55hex	schon zugewiesen
1Fhex	allgemeiner Fehler	56hex	ungültiges Paßwort
20hex	Verletzung des gleichzeitigen Zugriffs auf einen Datensatz	57hex	ungültiger Parameter
21hex	Verletzung des nicht erlaubten Zugriffs auf einen gesperrten Datensatz	58hex	Schreibfehler im Netzwerk
22hex	falsche Diskette		
23hex	FCB nicht verfügbar		
24hex	Lock Records		
25hex	besetzt		
31hex	reserviert		

Tabelle 2. Der Interrupt 21hex legt die Fehlernummer im AX-Register ab.

griff auf diese Geräteeinheiten erfolgt ausschließlich über Handles.

Mit der Funktion 3Chex versuchen Sie, eine Datei zu erzeugen. Als Ergebnis erhalten Sie im AX-Register und im Carry-Statusbit einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Findet es zum Zeitpunkt des Funktionsaufrufs einen Eintrag im Inhaltsverzeichnis, dann wird die Länge der bestehenden Datei auf 0 gesetzt. Existiert die Datei noch nicht, dann erzeugt die Funktion eine neue Datei. In beiden Fällen löscht das Betriebssystem das Carry-Statusbit und öffnet die Datei. Konnte die Routine die Datei nicht erzeugen, weil beispielsweise das Inhaltsverzeichnis vollgeschrieben ist, dann setzt sie das Carry-Statusbit und übergibt die Fehlernummer in AX.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 3Chex enthalten

und die Registerkombination DS:DX auf den Dateinamen zeigen, der mit dem Byte 0 endet. Das Register CX enthält vor dem Aufruf das Dateiattribut. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS, ES ändert sich nicht.

Funktion 3Dhex: Öffnen einer bestehenden Datei über Handle

Mit der Funktion 3Dhex öffnen Sie eine existierende Datei. Als Ergebnis zeigen die Zahlen im AX-Register und Carry-Statusbit an, ob das Betriebssystem die Datei erfolgreich geöffnet hat. Findet es zum Zeitpunkt des Funktionsaufrufs einen Eintrag im Inhaltsverzeichnis, dann öffnet die Funktion die Datei. In diesem Fall löscht das Betriebssystem das Carry-Statusbit und speichert das Handle der Datei im Register AX. Konnte die Routine die Datei nicht öffnen, weil beispielsweise

die Datei nicht existiert, dann wird das Carry-Statusbit gesetzt und die Fehlernummer in AX übergeben.

Um eine existierende Datei zu öffnen, muß das AH-Register den Wert 3Dhex enthalten und die Registerkombination DS:DX auf den Dateinamen zeigen, der mit einem Null-Byte endet. Das Register AL enthält vor dem Aufruf den Zugriffscode. Die Bits 0 bis 3 des Zugriffscode bestimmen die Zugriffsart. Die Datei darf entweder ausschließlich gelesen (Kombination 0000), beschrieben (Kombination 0001) oder gelesen und beschrieben werden (Kombination 0010). Die Bits 4 bis 6 kontrollieren den Zugriff durch andere Programme. Der Zugriff auf die Datei ist dem aktuellen Programm im FCB-Modus (Kombination 000) vorbehalten (ansonsten Kombination 001). Andere Programme dürfen die Datei entweder ausschließlich lesen (Kombination 010binär), beschreiben (Kombination 011) oder lesen und beschreiben (Kombination 100).

Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktion 3Ehex: Datei schließen über Handle

Mit der Funktion 3Ehex schließen Sie eine Datei. Als Ergebnis erhalten Sie im Carry-Statusbit einen Wert, dem Sie entnehmen, ob das Betriebssystem die Aktion erfolgreich durchgeführt hat. Ist zum Zeitpunkt des Funktionsaufrufs das angegebene Handle gültig, den das Unterprogramm im Register CX übernimmt, dann löscht die Prozedur das Carry-Statusbit und schließt die Datei. Wenn ein Fehler auftritt, dann setzt die Routine das Carry-Statusbit und legt den Fehlercode in AX ab.

Um eine Datei zu schließen, muß das AH-Register den Wert 3Ehex enthalten. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktion 3Fhex: Zeichenkette über Handle einlesen

Die Funktion 3Fhex liest Daten über ein in BX angegebenes Handle ein und speichert sie in einem vorher definierten Eingabepuffer. Empfängt die Funktion Daten, obwohl der Puffer voll ist, gibt sie eine Fehlermeldung weiter. Wenn die Tastatur das Eingabemedium ist, dann können Sie die eingegebene Zeichenkette mit den normalen Textkommandos edieren. Die Position des Eingabepuffers übergeben Sie mit dem Zeiger DS:DX an das Unterprogramm.

Um die Funktion 3Fhex aufzurufen, muß das AH-Register den Wert 3Fhex beinhalten. Nach Bearbeitung der Funktion enthält der Eingabepuffer

die eingegebene Zeichenkette. Vor dem Aufruf der Prozedur geben Sie im CX-Register an, wie viele Bytes die Funktion in den Tastaturpuffer einlesen und speichern soll.

Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht. Auch bei dieser Funktion enthält das AX-Register den Fehlercode, wenn die Funktion einen Fehler registriert und das Carry-Statusbit setzt.

Funktion 40hex: Zeichenkette über Handle ausgeben

Die Funktion 40hex dient dazu, Daten über ein in BX angegebenes Handle auszugeben, die in einem vorher definierten Eingabepuffer gespeichert sind. Wenn der Zugriff nicht erlaubt ist oder ein ungültiges Handle angegeben wurde, dann gibt die Routine eine Fehlermeldung weiter. Die Schreiboperation beginnt dabei an der aktuellen Position des Dateizeigers, der nach dem Schreibvorgang aktualisiert wird. Die Position des Tastaturpuffers übergeben Sie mit dem Zeiger DS:DX an das Unterprogramm. Das AX-Register enthält bei dieser Funktion den Fehlercode, wenn die Funktion einen Fehler registriert und das Carry-Statusbit setzt.

Um die Funktion 40hex aufzurufen, muß das AH-Register den Wert 40hex beinhalten. Vor dem Aufruf der Prozedur geben Sie im CX-Register an, wie viele Bytes die Funktion vom Tastaturpuffer über das Handle ausgeben soll. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktion 41hex: Datei über Handle löschen

Mit der Funktion 41hex löschen Sie eine Datei über Handle. Als Ergebnis erhalten Sie im Carry-Register einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Datei gelöscht hat. Findet es zum Zeitpunkt des Funktionsaufrufs den Namen der Datei im Inhaltsverzeichnis, dann löscht das Betriebssystem die Datei und das Carry-Statusbit. Sie können mit der Funktion 41hex keine Verzeichnisse löschen. Wenn die Datei, die das Unterprogramm löschen soll, nicht im aktuellen Verzeichnis vorhanden ist, dann wird das Carry-Statusbit gesetzt, und AX enthält den Fehlercode. Sie dürfen keine Wildcards im Dateinamen benutzen.

Um eine Datei zu löschen, muß das AH-Register den Wert 41hex beinhalten und die Registerkombination DS:DX auf den Dateinamen zeigen, der mit einem Null-Byte abgeschlossen wird. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Obwohl die Funktion 41hex eigentlich dokumentiert ist, gibt es eine Besonderheit. Wenn sie über die undokumentierte Netzwerkfunktion 5Dhex aufgerufen wird, können die Dateispezifikationen Wildcards enthalten, und alle dem Muster entsprechenden Dateien werden gelöscht. Zu diesem Zweck muß der Dateiname allerdings in dem dem Netzwerk angepaßten Format vorliegen, das die undokumentierte Funktion 60hex liefert.

Funktion 42hex: Position des Dateizeigers über Handle bestimmen

Die Funktion 42hex dient dazu, die Position des Dateizeigers für den Lesebeziehungsweise Schreibzugriff über ein Handle zu setzen. Die neue Position des Dateizeigers setzt das Betriebssystem nicht absolut, sondern relativ zu einer bestimmten Stelle. Der Parameter AL bestimmt diese Stelle. Enthält er die Zahl 0, dann bezieht sich die Positionsangabe auf den Anfang der Datei, ist er hingegen gleich 1, dann bezieht er sich auf die aktuelle Position des Zeigers. Wenn er den Wert 2 annimmt, dann berechnet das Betriebssystem die neue Position relativ zum Dateende. Unabhängig von AL übergibt die Funktion im Langwort DX:AX die Position des Dateizeigers relativ zum Dateianfang.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 42hex und das 32-Bit-Wort CX:DX den gewünschten Offset enthalten. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode, wenn das Carry-Statusbit gesetzt ist. Der Inhalt der Register BX, CX, SI, DI, BP, CS, DS, SS, ES und des Kennzeichenregisters ändert sich nicht.

Funktion 43hex: Attribut einer Datei ermitteln beziehungsweise setzen

Die Funktion 43hex ist dafür zuständig, die Attribute einer Datei zu verwalten. Mit dieser Prozedur kann der Anwender die Attribute einer Datei lesen und modifizieren. Wenn Sie im Register AL den Parameter 0 übergeben und die Routine aufrufen, erhalten Sie im Register CX eine Zahl, die das Attribut angibt. Bit 1 beinhaltet die Information, ob eine Datei nur gelesen (Bit 0 gleich 1) oder auch beschrieben werden darf (Bit 0 gleich 0). Bit 2 bestimmt hingegen, ob die Datei versteckt ist (Bit 1 gleich 1) oder nicht (Bit 1 gleich 0). Bit 5 gibt an, ob eine Datei seit der letzten Archivierung verändert worden ist (Bit 5 gleich 1) oder nicht (Bit 5 gleich 0). Aus den Bits 3 und 4 erkennen Sie schließlich, ob die Datei ein Volume (Bit 3 gleich 1) oder ein Unterverzeichnis ist (Bit 4 gleich 1). Tritt während der Funktionsausführung ein Fehler auf, dann setzt das

Unterprogramm das Carry-Statusbit und speichert die Fehlernummer im AX-Register. Der Inhalt der Register BX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Im Gegensatz dazu setzen Sie die zweite Unterfunktion ein, um die Angaben der Datei auf die vorgegebenen Parameter zu setzen. Zu diesem Zweck belegen Sie das Register AL mit der Zahl 1. Im Register CX übergeben Sie die neuen Attribute. Liegen die übergebenen Parameter im erlaubten Wertebereich, dann löscht das Betriebssystem die alten Angaben, aktualisiert sie und löscht das Carry-Statusbit. Wenn die Parameter, die Sie an das Unterprogramm übergeben, nicht gültig sind, dann wird das Carry-Statusbit gesetzt und AX beinhaltet den Fehlercode. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Sie wählen die Funktion 43hex richtig an, wenn das AH-Register den Wert 43hex enthält. In der Registerkombination DS:DX legen Sie zusätzlich den gewünschten Namen der Datei ab, der mit einem Null-Byte endet.

Funktion 44hex: Bearbeiten eines Gerätetreibers (IOCTL)

Die Funktion 44hex verfügt über mehrere Unterfunktionen, die sich ausschließlich mit dem Zugriff auf die Gerätetreiber beschäftigen (IOCTL für »Input/Output Control«). Mit dieser Prozedur kann der Anwender die Eigenschaften eines Treibers lesen, modifizieren und schreiben. Wenn Sie im Register AL den Parameter 0 übergeben und die Routine aufrufen, erhalten Sie im Register DX Daten über den Gerätetreiber. Bit 0 beinhaltet die Information, ob der Treiber ein Tastaturreiber ist (Bit 0 gleich 1). Bit 1 bestimmt hingegen, ob der Treiber ein Bildschirmtreiber ist (Bit 1 gleich 1) oder nicht (Bit 1 gleich 0). Bit 2 gibt an, ob es sich um einen Null-Treiber handelt (Bit 5 gleich 1) oder um einen Uhr-Treiber (Bit 3 gleich 1). Die Bits 4, 15 und 8 bis 13 sind reserviert. Wenn Bit 5 gelöscht ist, dann reagiert der Treiber auf Control-Steuersequenzen. Bit 14 bestimmt, daß der Treiber über die Unterfunktionen 2 und 3 Steuersequenzen verarbeiten kann (Bit 14 gleich 1). Tritt während der Funktionsausführung ein Fehler auf, dann setzt das Unterprogramm das Carry-Statusbit und speichert die Fehlernummer im AX-Register. Der Inhalt der Register BX, CX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Im Gegensatz dazu benutzen Sie die zweite Unterfunktion mit dem AL-Wert 1, um die Attribute des Gerätetreibers auf die vorgegebenen Parameter zu setzen. Zu diesem Zweck belegen Sie das Register AL mit der Zahl 1. Im

Register DX übergeben Sie das neue Attribut. Liegen die übergebenen Parameter im erlaubten Wertebereich, dann löscht das Betriebssystem die alten Angaben, aktualisiert sie und löscht das Carry-Statusbit. Wenn die Parameter, die Sie an das Unterprogramm übergeben, nicht gültig sind, wird das Carry-Statusbit gesetzt und AX beinhaltet den Fehlercode. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Mit der dritten Unterfunktion 3hex empfangen Sie eine Anzahl von Zeichen eines Treibers. Das Unterprogramm übergibt sie in einem Puffer an das aufrufende Programm, der durch die Registerkombination DS:DX lokalisiert wird. Sendet der Treiber zum Zeitpunkt des Funktionsaufrufs kein Zeichen, dann wartet das Betriebssystem auf einen empfangenen Wert. Das Register CX enthält die Anzahl der zu empfangenen Zeichen. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Die vierte Unterfunktion benutzen Sie hingegen, um eine Anzahl Zeichen an einen Treiber zu senden. Das aufrufende Programm übergibt sie in einem Puffer an das Unterprogramm, der durch die Registerkombination DS:DX lokalisiert wird. Empfängt zum Zeitpunkt des Funktionsaufrufs der Treiber kein Zeichen, dann wartet das Betriebssystem, bis der Treiber empfangsbereit ist. Das Register CX enthält die Anzahl der zu sendenden Zeichen. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Die Unterfunktionen 5 und 6 basieren auf dem gleichen Prinzip wie die Unterfunktionen 3 und 4. Der einzige Unterschied besteht darin, daß die letztgenannten anstelle von Blocktreibern Zeichentreiber bearbeiten. Die Parameterüber- und -ausgabe ist gleich.

Mit den Unterfunktionen 7 und 8 lesen Sie den Ein-beziehungsweise Ausgabestatus eines Treibers. Wenn die Parameter, die Sie an das Unterprogramm übergeben, nicht gültig sind, dann wird das Carry-Statusbit gesetzt, und AX beinhaltet den Fehlercode. Ist der Funktionsaufruf ohne Schwierigkeiten erfolgt, dann wird das Carry-Statusbit gelöscht, und das Register AX enthält den Status. Der ermittelte Wert beträgt 0, wenn der Treiber bereit ist, und 255, wenn der Treiber nicht bereit ist.

Wenn Sie herausfinden wollen, ob ein Laufwerk einen festen oder wechselbaren Datenträger hat, benutzen Sie die neunte Unterfunktion mit dem AL-Wert 8. Wenn während der Ausführung des Unterprogramms ein Fehler auftritt, dann wird das Carry-Statusbit gesetzt und AX beinhaltet den Fehlercode. Ansonsten gibt das AH-Register

an, ob der Datenträger wechselbar (AH gleich 0) oder fest installiert ist (AH gleich 1).

In Netzwerken sind die Unterfunktionen 10 und 11 von Bedeutung. Mit ihnen stellen Sie fest, ob ein angegebenes Laufwerk Teil des Computers ist, auf dem die Abfrage stattfindet, oder ob es zu einem Netzwerk gehört. Wenn der Funktionsaufruf ohne Schwierigkeiten erfolgt, wird das Carry-Statusbit gelöscht, und das Register DX beinhaltet den Status des Laufwerks. Bit 12 des ermittelten Werts ist gesetzt, wenn das Laufwerk Teil eines Netzwerks ist, andernfalls ist es gelöscht.

Die Funktion mit der AL-Nummer 11 bestimmt die Art des Zugriffs auf die Gerätetreiber. Treten beim Funktionsaufruf Schwierigkeiten auf, dann wird das Carry-Statusbit gesetzt, und das Register AX beinhaltet den Fehlercode. Anderenfalls löscht die Routine das Carry-Statusbit und setzt die Anzahl der Wiederholversuche sowie die Länge der Pause zwischen den Zugriffsversuchen. Gerade bei Netzwerken tritt oft der Fall auf, daß der Zugriff auf bestimmte Dateien für kurze Zeit gesperrt ist. In diesem Fall wiederholt das Betriebssystem seinen Zugriff mehrmals nach gewissen Pausen. Sie übergeben die Anzahl der Wiederholversuche im Register BX. Das Register CX bestimmt die Pause zwischen den Zugriffsversuchen. Sie ist computerabhängig, da dieser Wert die Anzahl von Leerschleifen bestimmt, die das Unterprogramm ausführt.

Sie wählen die Funktion 44hex richtig an, wenn das AH-Register den Wert 44hex und das Register AL die Nummer der Unterfunktion enthält. Bei den Unterfunktionen mit den AL-Nummern 0, 1, 2, 3, 6, 7 und 10 speichern Sie vor dem Aufruf das Handle im BX-Register. Die Unterfunktionen 4, 5, 8 und 9 benötigen die Gerätebezeichnung im Register BL. Dabei steht »1« für Gerätebezeichnung »A«, »2« für »B«, »3« für »C« und so weiter.

Funktion 45hex: Datei-Handle verdoppeln

Beim Aufruf dieser Funktion richtet das Betriebssystem für das im Register BX übergebene Handle ein zweites Handle ein. Ist während der Funktionsausführung kein Fehler aufgetreten, übergibt das Betriebssystem das neue Handle in AX und löscht das Carry-Statusbit. Das neue Handle hat dieselben Eigenschaften wie das übergebene Handle. Sogar die Position des Dateizeigers stimmt überein. Wenn die Parameter, die Sie an das Unterprogramm übergeben, nicht gültig sind, dann wird das Carry-Statusbit gesetzt, und AX beinhaltet den Fehlercode.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 45hex enthalten.

Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktion 46hex: Zwei Datei-Handles angleichen

Beim Aufruf dieser Funktion gleicht das Betriebssystem das in CX enthaltene Handle an das im Register BX übergebene Handle an. Ist während der Funktionsausführung kein Fehler aufgetreten, dann löscht das Betriebssystem das Carry-Statusbit. Das in CX übergebene Handle hat nach Ausführen der Routine dieselben Eigenschaften wie das Handle in BX. Wenn die Parameter, die Sie an das Unterprogramm übergeben, nicht gültig sind, dann wird das Carry-Statusbit gesetzt, und AX beinhaltet den Fehlercode. Bearbeitet der zweite Handle eine geöffnete Datei, dann wird diese Datei vor dem Vorgang der Angleichung geschlossen.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 46hex enthalten. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktion 47hex: Lesen des aktuellen Verzeichnisses

Mit der Funktion 47hex lesen Sie das aktuelle Inhaltsverzeichnis. Als Ergebnis erhalten Sie im Carry-Statusbit einen Wert, der Sie darüber informiert, ob das Betriebssystem einen Fehler bemerkt hat. Ist das im DL-Register angegebene Laufwerk existent, dann löscht die Funktion das Carry-Statusbit und liest das aktuelle Unterverzeichnis. Dabei steht »0« für das Standardlaufwerk, »1« für »A:«, »2« für »B:«, »3« für »C:« und so weiter. Konnte die Routine das Unterverzeichnis nicht lesen, weil es beispielsweise das angegebene Laufwerk nicht gibt, dann setzt sie das Carry-Statusbit. Das AX-Register enthält in diesem Fall den Fehlercode.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 47hex enthalten und die Registerkombination DS:SI auf einen 64 Byte großen Puffer zeigen, in dem die Funktion den Verzeichnisnamen ablegt. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktion 48hex: Arbeitsspeicher reservieren

Die Funktion 48hex des Interrupts 21hex (»Allocate Memory«) dient dazu, Speicherbereich für die Verwaltung eines Programms zur Verfügung zu stellen. Als Ergebnis erhalten Sie im Register AX die Segmentadresse des benötigten Arbeitsspeichers. Der reservierte Bereich beginnt an der Adresse AX:00hex. Beim Aufruf der Funktion 48hex muß das AH-Register

Offset	ab MS-DOS 3.1	ab MS-DOS 4.0	Bedeutung
-0Chex	x	x	Wiederholungszähler (Datei-Sharing)
-0Ahex	x	x	Wiederholungszeit (Datei-Sharing)
-08hex	x	x	Zeiger auf aktuellen Laufwerkpuffer
-04hex	x	x	Zeiger auf die bisher ungelesene Eingabe über »con«
-02hex	x	x	Segment des ersten Speicherkontrollblocks
00hex	x	x	Zeiger auf den ersten Laufwerkparameterblock
04hex	x	x	Zeiger auf Dateitabellen
08hex	x	x	Zeiger auf den Uhr-Gerätetreiber
0Chex	x	x	Zeiger auf den aktuellen Null-Treiber
10hex	x	x	maximale Anzahl der Bytes pro Block eines Blocktreibers
12hex	x	x	Zeiger auf den ersten Laufwerkpuffer
16hex	x	x	Zeiger auf Feld der aktuellen Verzeichnisstrukturen
1Ahex	x	x	Zeiger auf FCB-Tabelle
1Ehex	x	x	Anzahl der geschützten FCBs
20hex	x	x	Anzahl der Blocktreiber
21hex	x	x	Wert des Kommandos »lastdrive« in der »config.sys«
22hex	x	x	Vorspann des aktuellen Null-Gerätetreibers
34hex	x	x	Anzahl der mit »join« verbundenen Laufwerke
35hex	—	x	Zeiger auf spezielle Programmnamen
37hex	—	x	Zeiger auf eine Far-Prozedur für residente IFS-Utility-Funktionen
3Bhex	—	x	Zeiger auf die Kette der IFS-Treiber
3Fhex	—	x	die erste Angabe, die dem Buffers-Befehl in der »config.sys« folgt
41hex	—	x	die zweite Angabe, die dem Buffers-Befehl in der »config.sys« folgt
43hex	—	x	Boot-Laufwerk
44hex	—	x	unbekannt
45hex	—	x	Größe des erweiterten Arbeitsspeichers in KByte

Tabelle 6. Die sogenannte »Liste der Listen« bietet aufschlußreiche Informationen über die interne Struktur des Betriebssystems.

den Wert 48hex beinhalten. Im Register BX teilen Sie der Funktion mit, wie viele 16-Byte-Paragraphen das Unterprogramm zur Verfügung stellen soll. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode, wenn das Carry-Statusbit gesetzt ist, und BX die Anzahl der zur Verfügung stehenden Paragraphen. Der Inhalt der Register CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktion 49hex: Arbeitsspeicher freigeben

Die Funktion 49hex gibt Arbeitsspeicher, den das Betriebssystem vorher mit der Funktion 48hex reserviert hat, wieder frei.

Beim Aufruf der Funktion 49hex beinhaltet das AH-Register den Wert 49hex. Im Register ES teilen Sie der Funktion mit, wo sich die Segmentadresse des reservierten Speicherbereichs befindet. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode, wenn das Carry-Statusbit gesetzt ist. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktion 4Ahex: Größe eines reservierten Speicherbereichs ändern

Die Funktion 4Ahex des Interrupts 21hex (»Set Block«) dient dazu, den für die Verwaltung eines Programms zur Verfügung gestellten Speicherbereich zu ändern. Der reservierte Bereich beginnt an der alten Anfangsadresse.

Beim Aufruf der Funktion 4Ahex muß das AH-Register den Wert 4Ahex beinhalten. Im Register BX teilen Sie der Funktion mit, wie viele 16-Byte-Paragraphen das Unterprogramm insgesamt zur Verfügung stellen soll, und in ES die Segmentadresse des Speicherbereichs. Nachdem das Betriebssystem die Funktion ausgeführt hat, enthält das Register AL den Fehlercode, wenn das Carry-Statusbit gesetzt ist, und BX die Anzahl der zur Verfügung stehenden Paragraphen. Der Inhalt der Register CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Obwohl die Funktion dokumentiert ist, gibt es eine Situation unter MS-DOS 3.x, die nicht dokumentiert ist. Wenn die Funktion 4Ahex den angeforderten Speicherbereich nicht reservieren kann, weil der zur Verfügung stehende Speicherplatz für die Reservierung nicht ausreicht, übergibt sie

in BX die Anzahl von Paragraphen, die sie maximal belegen kann.

Funktion 4Bhex: Programm in den Arbeitsspeicher laden oder zusätzlich ausführen

Die Funktion 4Bhex ist dafür zuständig, ein Programm in den Speicher zu laden. Wenn man im Register AL den Parameter 0 übergibt und die Routine aufruft, startet das Betriebssystem das Programm zusätzlich. Der übergebene Parameterblock kontrolliert das Laden und Ausführen des Programms. Die Bytes 0 und 1 beinhalten die Segmentadresse eines Umgebungsblocks, der beispielsweise den Status des Verify-Statusbits oder des Break-Statusbits enthalten kann. Die Registerkombination der Bytes 4-5:2-3 bestimmt die Adresse der Kommandozeile, die nicht länger als 128 Bytes sein darf und mit dem ASCII-Zeichen 13 enden muß. Das erste Byte der Kommandozeile gibt ihre eigene Länge in Bytes an, wobei das Byte zur Längenangabe mitgezählt und das abschließende »Return« ignoriert wird. Die Bytes 6-7:8-9 geben die Adresse des ersten FCB an. Aus den Bytes 10-11:12-13 erkennen Sie die Adresse des zweiten FCB. Tritt während der Funktionsausführung ein Fehler auf, dann setzt das Unterprogramm das Carry-Statusbit und speichert die Fehlernummer im AX-Register ab.

Im Gegensatz dazu setzen Sie die zweite Unterfunktion ein, um ein Programm zu laden und im Speicher zu behalten, ohne es zu starten. Zu diesem Zweck belegen Sie das Register AL mit der Zahl 3. In den Registern übergeben Sie die gleichen Parameter wie bei der ersten Unterfunktion. Liegen die übergebenen Parameter im erlaubten Wertebereich, lädt das Betriebssystem das Programm und löscht das Carry-Statusbit. Wenn die Parameter, die Sie an das Unterprogramm übergeben, nicht gültig sind, dann wird das Carry-Statusbit gesetzt, und AX beinhaltet den Fehlercode. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Sie wählen die Funktion 4Bhex richtig an, wenn das AH-Register den Wert 4Bhex enthält. In der Registerkombination DS:DX speichern Sie zusätzlich den gewünschten Dateinamen, der mit einem Null-Byte endet und keine Wildcards enthalten darf. Die Adresse ES:BX übergibt die Anfangsadresse der Parameterblocks.

Funktion 4Chex: Programm beenden

Die Funktion 4Chex des Interrupts 21hex (»End Process«) dient dazu, den Ablauf des aktuellen Programms zu unterbrechen und in das aufrufende

Programm zurückzukehren. Das ist fast immer die Kommandoebene des Betriebssystems MS-DOS. Bevor die Prozedur die Kontrolle an das aufrufende Programm übergibt, restauriert sie die drei Interrupt-Vektoren, deren Inhalt vor dem Start des Programms im PSP gespeichert wurde. Der PC stellt den Arbeitsspeicher, den das beendete Programm belegt, nach dem Funktionsaufruf anderen Programmen zur Verfügung und schreibt den Inhalt aller Dateipuffer auf den angesprochenen Datenträger.

Beim Aufruf der Funktion 4Chex muß das AH-Register den Wert 4Chex beinhalten. Im AL-Register legen Sie den Rückkehr-Code, den Sie durch Aufruf der Funktion 4Dhex erhalten, und im CS-Register die Segmentadresse des zugehörigen PSP ab.

Funktion 4Dhex: Rückkehr-Code ermitteln

Mit dieser Funktion ermitteln Sie den Rückkehr-Code der mit den Funktionen 31hex und 4Chex beendeten Prozesse. Nachdem Sie diese Routine aufgerufen haben, enthält das AL-Register den Rückkehr-Code. Zusätzlich klärt Sie das Register AH darüber auf, wie ein Programm unterbrochen wurde. Beinhaltet es die Information, daß der Benutzer das Programm auf normalem Weg beendet hat, dann ist AH gleich 0. Der Wert 1 bestimmt hingegen, daß der Anwender die Datei durch die Tastenkombination [Strg-C] unterbrochen hat. Die Zahl 2 gibt an, daß ein Fehler bei dem Zugriff auf ein Gerät aufgetreten ist. Aus dem Wert 3 schließlich ist ersichtlich, daß das Programm durch den Aufruf der Funktion 00hex beendet worden ist. Wenn beim Ausführen des Unterprogramms ein Fehler auftritt, dann ist das Carry-Statusbit gesetzt, und AX enthält den Fehlercode.

Beim Einsatz dieser Funktion muß das AH-Register den Wert 4Dhex enthalten. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Funktion 4Ehex: Ersten Eintrag im Inhaltsverzeichnis suchen

Die Funktion 4Ehex sucht den ersten Eintrag im Inhaltsverzeichnis des aktuellen Verzeichnisses. Als Ergebnis erhalten Sie im Carry-Statusbit einen Wert, der Sie darüber informiert, ob das Betriebssystem die Aktion erfolgreich beendet hat. Findet es zum Zeitpunkt des Funktionsaufrufs einen Eintrag im Inhaltsverzeichnis, löscht das Betriebssystem das Carry-Statusbit und liest die Dateiinformationen ein. Wenn kein Eintrag vorhanden ist, dann wird das Carry-Statusbit gesetzt, und AX enthält den Fehlercode. Die Routine sucht die Datei mit dem im Re-

gister CX übergebenen Attribut. Beim Aufruf dieser Funktion muß das AH-Register den Wert 4Ehex enthalten, und die Registerkombination DS:DX muß auf den Namen des Eintrags zeigen, der mit dem Byte 0 endet und auch ein Verzeichnis oder Volume sein kann. Der Name kann aus Laufwerkangaben, Unterverzeichnissen und Wildcards bestehen. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Undokumentiert ist, daß die Funktion ab MS-DOS 3.0 auch ein zeichenorientiertes Gerät meldet, wenn der Dateiname keine Wildcards beinhaltet. In diesem Fall meldet das Betriebssystem das Attribut 40hex sowie Uhrzeit und Datum.

Darüber hinaus ist die Funktion 4Ehex fehlerhaft. Falls Sie die eben beschriebene Technik nutzen und ein Suchattribut mit gesetztem dritten Bit definieren, arbeitet der undokumentierte Aufruf nur beim ersten Mal korrekt. Damit er immer fehlerfrei abläuft, müssen Sie vorher eine der Funktionen 3Chex, 3Dhex, 41hex und 56hex aktivieren.

Funktion 4Fhex: Nächsten Eintrag im Inhaltsverzeichnis suchen

Die Funktion 4Fhex sucht den nächsten Eintrag im aktuellen Inhaltsverzeichnis, nachdem Sie mit Funktion 4Ehex den ersten Eintrag gefunden haben. Als Ergebnis erhalten Sie im Carry-Statusbit einen Wert, der Ihnen mitteilt, ob das Betriebssystem einen folgenden Eintrag ermittelt hat. Findet es einen Eintrag im Inhaltsverzeichnis, dann löscht die Funktion das Carry-Statusbit und liest die gewünschten Informationen ein. Ist aber kein Eintrag vorhanden, dann setzt das Unterprogramm das Carry-Statusbit und legt in AX den Fehlercode ab.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 4Fhex enthalten und die Registerkombination DS:DX auf den Namen des Eintrags zeigen, der mit einem Null-Byte abgeschlossen wird. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS, ES und des Kennzeichenregisters ändert sich nicht.

Die Funktionen 50hex und 51hex:

Die undokumentierte Funktion 50hex ersetzt den aktuellen PSP durch einen anderen. Um die Funktion korrekt aufzurufen, speichern Sie im AH-Register den Wert 50hex. BX gibt die Segmentadresse des neuen PSP an. Die Funktion gibt keine Werte zurück.

Die Funktion 51hex hingegen liest die Segmentadresse des aktuellen PSP. Wenn Sie diese undokumentierte Funktion benutzen wollen, legen Sie vor dem Aufruf den Wert 51hex im AH-

Register ab. Nach der Rückkehr in das Hauptprogramm gibt BX die Segmentadresse des aktuellen PSP an.

Funktion 52hex: Adresse der »Liste der Listen« ermitteln

Die Funktion 52hex ist eine der interessantesten Funktionen des MS-DOS-Interrupts 21hex überhaupt. Obwohl Microsoft sie nicht dokumentiert hat, greifen viele Anwendungsprogramme auf sie zu. Selbst OS/2 1.1 benutzt sie in der Kompatibilitätsbox. Die Funktion 52hex ermittelt die Adresse einer Tabelle, die als »Liste der Listen« bezeichnet wird. In der Tat enthält die Tabelle zahlreiche Informationen über das Betriebssystem, die auf anderem Weg kaum zu bekommen sind.

Doch zuerst etwas zur technischen Seite des Aufrufes. Nachdem Sie die Int-21hex-Routine mit dem Wert 52hex im AH-Register aufgerufen haben, gibt das Betriebssystem in der Registerkombination ES:BX die Position zurück, an der sich die Adresse der »Liste der Listen« befindet.

In dieser Liste sind wichtige Informationen zur internen Struktur von MS-DOS abgelegt. Darunter befinden sich zum Beispiel Angaben über die aktuelle Verzeichnisstruktur, das Boot-Laufwerk oder den erweiterten Arbeitsspeicher (Tabelle 6).

Ein Anwendungsbeispiel in dem Beitrag zur Gerätetreiber-Programmierung in dieser Ausgabe der DOS Extra demonstriert die Aufgabe einzelner Tabellenwerte. Dort finden Sie auch weitergehende Informationen zum Aufbau der Liste.

Funktion 53hex: BIOS-Parameterblock übersetzen

Die undokumentierte Funktion 53hex übersetzt einen BIOS-Parameterblock in einen Laufwerk-Parameterblock. Zu diesem Zweck übergeben Sie in DS:SI die Adresse des zu übersetzenden BIOS-Parameterblocks und in ES:BP den Zeiger auf einen Puffer, der den Laufwerk-Parameterblock aufnimmt. Nach dem Aufruf enthält der Puffer ab ES:BP den Laufwerk-Parameterblock. Die Funktion setzt voraus, daß Sie in AH den Wert 53hex speichern.

Funktion 54hex: Verify-Statusbit lesen

Unter normalen Umständen überprüft das Betriebssystem alle Daten, die es auf einen Datenträger geschrieben hat, auf ihre Richtigkeit. Diese Vorgehensweise ist zwar für die Datensicherheit bedeutsam, aber sie benötigt viel Zeit. Benutzen Sie qualitativ hochwertige Disketten, dann entstehen keine Probleme, wenn Sie die Datenüberprüfung ausschalten. Die Funktion 54hex liest den Status des Verify-Statusbits.

Wenn es gesetzt ist, das heißt, wenn die geschriebenen Daten überprüft werden, speichert die Prozedur den Wert 1 im AL-Register, andernfalls die Zahl 0.

Um die Funktion 54hex aufzurufen, muß das AH-Register den Wert 54hex beinhalten. Nach der Rückkehr aus der Funktion enthalten alle Register ihre ursprünglichen Werte einschließlich des Kennzeichenregisters, aber mit Ausnahme von AL.

Funktion 55hex: erzeugt ungeordneten PSP

Die undokumentierte Funktion 55hex erzeugt ab MS-DOS 2.0 einen untergeordneten PSP. Sie entspricht im Grunde genommen der Funktion 26hex. Der Unterschied besteht darin, daß die Funktion den Speicherbedarf ausdrücklich festlegt. DX enthält vor dem Aufruf die Segmentnummer, an der das Betriebssystem den neuen PSP generiert. SI gibt ab MS-DOS 3.0 an, welche Zahl im Speicherfeld des PSP (DX:02hex) abgelegt wird.

Um die Funktion aufzurufen, muß AH den Wert 55hex beinhalten.

Funktion 56hex: Umbenennen von Dateien

Mit der Funktion 56hex benennen Sie eine oder mehrere Dateien um. Als Ergebnis erhalten Sie im Carry-Statusbit einen Wert, der angibt, ob der Vorgang erfolgreich abgeschlossen wurde. Findet das Betriebssystem zum Zeitpunkt des Funktionsaufrufs den Namen der zu verändernden Datei im Inhaltsverzeichnis, dann löscht das Betriebssystem das Carry-Statusbit und übernimmt den neuen Namen für die Datei, indem es den Verzeichniseintrag ändert. Sie können bei der Angabe des Dateinamens keine Wildcards benutzen und somit auch nicht den Namen einer Dateigruppe ändern. Wenn die Funktion die Datei nicht gefunden hat oder der neue Name schon existiert, dann enthält das Carry-Statusbit den Wert 1.

Um eine Datei umzubenennen, muß das AH-Register den Wert 56hex beinhalten und die Registerkombination DS:DX auf den alten sowie ES:DI auf den neuen Dateinamen zeigen. Die Namen müssen mit einem Byte mit dem Inhalt 0 abgeschlossen sein. Der Inhalt der Register BX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Obwohl die Funktion 56hex eigentlich dokumentiert ist, gibt es eine Besonderheit. Wenn sie über die undokumentierte Netzwerkfunktion 5Dhex aufgerufen wird, können die Dateispezifikationen für den alten und den neuen Dateinamen Wildcards enthalten. Zu diesem Zweck müssen sie allerdings in dem dem Netzwerk ange-

paßten Format vorliegen, das die undokumentierte Funktion 60hex liefert.

Funktion 57hex: lesen von Zeit und Datums-Attributen

Auch Programmierer sind vergeßlich. Aus diesem Grund interessieren sie sich besonders für die Angabe, wann sie eine bestimmte Datei zuletzt verändert haben. Die Funktion 57hex nimmt ihnen das aufwendige Führen eines Karteikastens ab. Mit dieser Prozedur kann der Anwender die Zeit- und Datumsattribute einer Datei lesen und modifizieren. Wenn Sie im Register AL den Parameter 0 übergeben und die Routine aufrufen, erhalten Sie im Register CX eine Zahl, die die Uhrzeit angibt, und in DX einen Wert, der das Datum anzeigt, an dem eine Datei zuletzt geändert wurde. Bit 0 bis 4 von CX beinhalten die Sekunden, Bits 5 bis 10 die Minuten und Bits 11 bis 15 die Stunden der Zeitangabe. Die Bits 0 bis 4 der Datumsangabe geben den Tag, Bits 5 bis 8 den Monat und Bits 9 bis 15 das Jahr relativ zu 1980 an. Tritt während der Funktionsausführung ein Fehler auf, dann setzt das Unterprogramm das Carry-Statusbit und speichert die Fehlernummer im AX-Register. Der Inhalt der Register BX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Im Gegensatz dazu setzt man die zweite Unterfunktion ein, um die Angaben der Datei auf den vorgegebenen Wert zu setzen. Zu diesem Zweck belegen Sie das Register AL mit der Zahl 1. Im Register CX übergeben Sie die aktuelle Uhrzeit und in DX das Datum. Sind die übergebenen Parameter sinnvoll, löscht das Betriebssystem die alten Angaben, aktualisiert sie und löscht das Carry-Statusbit. Wenn die Parameter nicht gültig sind, dann wird das Carry-Statusbit gesetzt und AX beinhaltet den Fehlercode. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Sie wählen die Funktion 57hex richtig an, wenn das AH-Register den Wert 57hex enthält. Im Register BX legen Sie zusätzlich den gewünschten Handle des Gerätetreibers ab.

Das Betriebssystem des PC teilt den aufgerufenen Programmen den jeweils benötigten Platz im Arbeitsspeicher zu. Es gibt grundsätzlich drei Konzepte, nach denen das Betriebssystem vorgeht. Beim ersten durchsucht das DOS den Arbeitsspeicher von der niedrigsten Adresse bis zur höchsten und beendet die Suche, wenn es genügend freien Speicherplatz gefunden hat. Die zweite Strategie, die in der Regel eingesetzt wird, sucht den Bereich, dessen Größe am wenigsten über die angeforderte Speicherplatzlänge hinausgeht. Die dritte Methode durchsucht den Speicher von oben nach unten.

Mit der ersten Unterfunktion kann der Anwender das Konzept der verwendeten Speicherverwaltung ermitteln. Wenn Sie im Register AL den Parameter 0 übergeben und die Routine aufrufen, erhalten Sie im Register AX die Nummer des Konzepts. Ist AX gleich, dann durchsucht das Betriebssystem den Speicher von unten nach oben, andernfalls von oben nach unten (AX gleich 2) oder nach der effizientesten Methode (AX gleich 1). Tritt während der Funktionsausführung ein Fehler auf, dann setzt das Unterprogramm das Carry-Statusbit und speichert die Fehlernummer im AX-Register. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Im Gegensatz dazu setzen Sie die zweite Unterfunktion ein, um die Methode der Speicherverwaltung zu bestimmen. Zu diesem Zweck belegen Sie das Register AL mit der Zahl 1. Im Register AX übergeben Sie die Nummer des gewünschten Konzepts. Liegt der übergebene Parameter im Wertebereich, löscht das Betriebssystem die alte Angabe, aktualisiert sie und löscht das Carry-Statusbit. Wenn der Wert, den Sie an das Unterprogramm übergeben, nicht gültig ist, dann wird das Carry-Statusbit gesetzt und AX beinhaltet den Fehlercode. Der Inhalt der Register BX, CX, DX, SI, DI, BP, CS, DS, SS und ES ändert sich nicht.

Sie wählen die Funktion 58hex richtig an, wenn das AH-Register den Wert 58hex enthält.

Funktion 59hex: Erweiterte Fehlermeldung holen

Die Funktion 59hex erlaubt es Ihnen, detaillierte Informationen über einen Fehler zu erhalten, der beim Aufruf einer Funktion des Interrupts 21hex oder 24hex aufgetreten ist. Diese Angaben schließen nicht nur eine genaue Angabe der Fehlerart, der Fehlerursache und der Fehlerquelle ein, sondern auch einen Rat, welche Aktionen Sie aufgrund der Fehlermeldung durchführen sollten.

Wenn Sie die Funktion 59hex aufrufen, muß das AH-Register den Wert 59hex beinhalten. Nach der Rückkehr aus der Funktion enthält das Register AX die Fehlerbeschreibung, BH die Fehlerursache, CH die Fehlerquelle und BH die empfohlene Vorgehensweise. Der Inhalt der Register CS, SS und ES ändert sich nicht. Die genaue Bedeutung der Inhalte der jeweiligen Register entnehmen Sie den Tabellen 3 bis 5.

Funktion 5Ahex: Eine temporäre Datei über Handle erzeugen

Mit der Funktion 5Ahex erzeugen Sie eine temporäre Datei. Als Ergebnis erhalten Sie im AX-Register und im Car-

Fehlerursache

BH	Beschreibung
01hex	Es steht kein Platz mehr auf dem Speichermedium zur Verfügung.
02hex	kurzfristiges Zugriffsverbot
03hex	Zugriff aufgrund mangelnder Autorisation nicht erlaubt
04hex	interner Softwarefehler
05hex	Hardwareproblem
06hex	Fehler in der Systemsoftware
07hex	Fehler im Anwendungsprogramm
08hex	Datei oder Einheit nicht gefunden
09hex	Datei oder Einheit falsch aufgebaut oder ungültig
0Ahex	Datei oder Einheit gesperrt
0Bhex	Es ist ein Problem mit dem Speichermedium aufgetreten.
0Chex	unbekannter Fehler

Tabelle 3. Die Parameter für die erweiterten Fehlermeldungen der Funktion 59hex erlauben eine detaillierte Analyse der Ursache.

BL	Bedeutung
01hex	Vorgang wiederholen und bei weiterem Mißerfolg den Anwender befragen
02hex	Vorgang nach einer bestimmten Pause wiederholen
03hex	Anwender um nochmalige Eingabe der Daten bitten
04hex	Programm auf dem üblichen Weg beenden (inklusive Schließen der offenen Dateien)
05hex	Programm sofort beenden
06hex	Fehler ist unwichtig
07hex	Bitte an den Anwender, die Fehlerursache zu beseitigen und die Funktion noch einmal aufzurufen

Tabelle 4. Der im BL-Register übergebene Ratschlag erleichtert die Reaktion auf einen aufgetretenen Fehler.

Fehlerquelle

CH	Bedeutung
01hex	nicht bekannt
02hex	Blocktreiber des Arbeitsspeichers
03hex	Netzwerk
04hex	serielle Schnittstelle
05hex	Fehler im Arbeitsspeicher

Tabelle 5. Die in CH spezifizierte Fehlerstelle vereinfacht die Untersuchung der Fehlerquelle.

ry-Statusbit einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Die Prozedur erzeugt eine Datei, die vom Betriebssystem kontrolliert und zur zeitweisen Datenauslagerung benutzt wird. MS-DOS bestimmt den Namen selbst, so daß Sie diesen Parameter ignorieren können. In diesem Fall löscht das Betriebssystem das Carry-Statusbit und öffnet die Datei. Konnte die Routine die Datei nicht erzeugen, weil beispielsweise das Inhaltsverzeichnis vollgeschrieben ist, dann setzt sie das Carry-Statusbit und übergibt die Fehlernummer in AX.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 5Ahex enthalten und die Registerkombination DS:DX auf den Namen des Verzeichnisses zeigen, der mit dem Byte 0 endet und die temporäre Datei aufnimmt. Das Register CX enthält vor dem Aufruf das Dateiattribut. Der Inhalt der Register BX,

CX, SI, DI, BP, CS, SS, ES ändert sich nicht.

Funktion 5Bhex: versucht, eine neue Datei zu erzeugen

Mit der Funktion 5Bhex versuchen Sie, eine Datei zu erzeugen. Als Ergebnis erhalten Sie im AX-Register und im Carry-Statusbit einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Findet es zum Zeitpunkt des Funktionsaufrufs einen Eintrag im Inhaltsverzeichnis, wird die Länge der bestehenden Datei auf 0 gesetzt. Existiert die Datei noch nicht, erzeugt die Funktion eine neue Datei. In beiden Fällen löscht das Betriebssystem das Carry-Statusbit und öffnet die Datei. Konnte die Routine die Datei nicht erzeugen, weil beispielsweise das Inhaltsverzeichnis vollgeschrieben ist, setzt sie das Carry-Statusbit und übergibt die Fehlernummer in AX.

Der Unterschied zur Funktion 3Chex besteht darin, daß das Betriebssystem einen Fehler meldet, wenn die Datei bereits existiert (AX gleich 50hex). Beim Aufruf dieser Funktion muß das AH-Register den Wert 5Bhex enthalten und die Registerkombination DS:DX auf den Dateinamen zeigen, der mit dem Byte 0 endet. Das Register CX enthält vor dem Aufruf das Dateiattribut.

Funktion 5Chex: Region sperren und freigeben

Mit der Funktion 5Chex sperren Sie einen Dateibereich oder geben ihn wieder frei. Als Ergebnis erhalten Sie im AX-Register und im Carry-Statusbit einen Wert, der Sie darüber informiert, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Konnte die entsprechende Unterfunktion erfolgreich ausgeführt werden, löscht das Betriebssystem das Carry-Statusbit. Konnte die Routine den gewünschten Vorgang nicht durchführen, setzt sie das Carry-Statusbit und übergibt die Fehlernummer in AX.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 5Chex enthalten. Die Registerkombinationen CX:DX und SI:DI geben zwei 4 Byte lange Integer-Zahlen an. CX:DX bestimmt den Offset des zu bearbeitenden Bereichs ab dem Dateianfang. SI:DI gibt die Länge des Bereichs an. Das Register BX enthält vor dem Aufruf das Handle.

Wenn AL den Wert 0 enthält, wird der definierte Abschnitt gesperrt, ist AL gleich 1, wird er wieder freigegeben.

Die Funktion 5Dhex: für den internen Gebrauch

Die Funktion 50hex hat mehrere voneinander unabhängige Unterfunktionen. Die im AL-Register übergebene Unterfunktionsnummer komplettiert im folgenden die Funktionsnummer. Das heißt, Sie laden einfach das AX-Register mit dem angegebenen Funktionswert.

Die undokumentierte Funktion 5D00hex ruft eine Interrupt-21hex-Funktion unter Berücksichtigung der Zugriffsvorschriften für Netzwerke auf. Die Registerkombination DS:DX muß vor dem Aufruf eine Adresse enthalten, die auf den Anfang der in Tabelle 7 aufgeführten MS-DOS-Parameterliste zeigt. Nach dem Aufruf sind die Register belegt, die auch bei einem Standardaufruf belegt wären.

Mit der Funktion 5D01hex leeren Sie alle Laufwerkspuffer und aktualisieren die Verzeichniseinträge der aktiven Dateien. Als Ergebnis erhalten Sie im AX-Register und im Carry-Statusbit einen Wert, der Ihnen sagt, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Konnte die Funktion erfolgreich ausgeführt werden, löscht

Offset	Beschreibung
00hex	AX-Register
02hex	BX-Register
04hex	CX-Register
06hex	DX-Register
08hex	SI-Register
0Ahex	DI-Register
0Chex	DS-Register
0Ehex	ES-Register
10hex	reserviert
12hex	Computeridentifikation (0000hex gleich aktuelles System)
14hex	Prozeßidentifikation (PSP-Segment auf dem angewählten Computer)

Tabelle 7. Die MS-DOS-Parameterliste findet vor allem bei Netzwerkprogrammen Verwendung.

das Betriebssystem das Carry-Statusbit. Konnte die Routine den gewünschten Vorgang nicht durchführen, dann setzt sie das Carry-Statusbit und übergibt die Fehlernummer in AX.

Beim Aufruf dieser Funktion muß die Registerkombinationen DS:DX den Zeiger auf die MS-DOS-Parameterliste enthalten. Von der Liste benutzt die Funktion allerdings nur die Computer- und Prozeßidentifikation.

Die Funktion 5D02hex wird nur verwendet, wenn das Dienstprogramm »share.exe« aktiv ist. Sie schließt eine Datei, die durch ihren direkt zugeordneten Dateinamen spezifiziert ist.

Als Ergebnis erhalten Sie im AX-Register und im Carry-Statusbit einen Wert, der Sie darüber informiert, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Konnte die Datei geschlossen werden, löscht das Betriebssystem das Carry-Statusbit. Wurde sie nicht geschlossen, dann setzt es das Carry-Statusbit und übergibt die Fehlernummer in AX.

Beim Aufruf dieser Funktion muß die Registerkombinationen DS:DX den Zeiger auf die MS-DOS-Parameterliste enthalten. Das Registerpaar DS:DX der Liste zeigt auf den Dateinamen im ASCII-Format.

Die Funktion 5D03hex findet nur Verwendung, wenn das Dienstprogramm »share.exe« aktiv ist. Sie schließt alle Dateien, die unter Berücksichtigung einer bestimmten Computeridentifikation geöffnet wurden.

Als Ergebnis erhalten Sie im AX-Register und im Carry-Statusbit einen Wert, der Sie darüber informiert, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Konnten die Dateien geschlossen werden, löscht das Betriebssystem das Carry-Statusbit. Wurden sie nicht geschlossen, setzt es das Carry-Statusbit und übergibt die Fehlernummer in AX.

Die Funktion 5D04hex findet nur Verwendung, wenn das Dienstprogramm »share.exe« aktiv ist. Sie schließt alle Dateien, die unter Berücksichtigung einer bestimmten Prozeßidentifikation geöffnet wurden.

Beim Aufruf dieser Funktion muß die Registerkombinationen DS:DX den Zeiger auf die MS-DOS-Parameterliste enthalten. Von der Liste benutzt die Funktion die Computer- und Prozeßidentifikation.

Die Funktion 5D05hex wird nur verwendet, wenn das Dienstprogramm »share.exe« aktiv ist. Sie liest den einer geöffneten Datei zugeordneten Eintrag in einer speziellen Share-Liste.

Als Ergebnis erhalten Sie im AX-Register und im Carry-Statusbit einen Wert, dem Sie entnehmen, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Konnte der Eintrag ermittelt werden, löscht das Betriebssystem das Carry-Statusbit. Wurde er nicht gelesen, setzt sie das Carry-Statusbit und übergibt die Fehlernummer in AX.

Beim Aufruf dieser Funktion muß die Registerkombinationen DS:DX den Zeiger auf die MS-DOS-Parameterliste enthalten. Nach dem Aufruf zeigt BX auf die Netzwerknummer und die Registerkombination ES:DI auf den Dateinamen.

Die Funktion 5Ehex: erfüllt verschiedene Netzwerkfunktionen

ab MS-DOS 3.1, die durch die in AL abgelegte Unterfunktionsnummer näher beschrieben sind. Als Ergebnis erhalten Sie immer im AX-Register und im Carry-Statusbit einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. In diesem Fall löscht das Betriebssystem das Carry-Statusbit. Konnte die Routine die angegebene Funktion nicht fehlerfrei erfüllen, setzt es das Carry-Statusbit und übergibt die Fehlernummer in AX.

Mit AL gleich 1 legen Sie Maschinennummern und -nummern im Netzwerk fest. CL gibt vor dem Aufruf die Maschinennummer an, DS:DX zeigt auf den 15 Byte langen Namen im ASCII-Format.

Wenn AL gleich 4 ist, bestimmen Sie, ob der Drucker im Text- (DX gleich 0000hex) oder im Binärmodus (DX gleich 0001hex) arbeiten soll. BX gibt

den Eintrag in der Umleitungstabelle an. Die Unterfunktion 5 (AL gleich 05hex) überprüft hingegen, in welchem Modus der Drucker angesteuert wird. BX gibt auch hier den Eintrag in der Umleitungstabelle an.

Die Funktion 5Fhex: aktiviert das aktuelle Laufwerk

Mit der Funktion 5Fhex erfüllen Sie verschiedene Netzwerkfunktionen ab MS-DOS 4.0, die durch die in AL abgelegte Unterfunktionsnummer näher beschrieben sind.

Als Ergebnis erhalten Sie im AX-Register und im Carry-Statusbit einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem die Aktion erfolgreich abgeschlossen hat. Wurde die Aktion erfolgreich abgeschlossen, löscht das Betriebssystem das Carry-Statusbit. Konnte die Routine die angegebene Funktion nicht fehlerfrei erfüllen, setzt sie das Carry-Statusbit und übergibt die Fehlernummer in AX. Die beiden Unterfunktionsnummern beschäftigen sich mit der Umleitung von Laufwerk- und Druckerzugriffen. Sie verlangen im BL-Register den Gerätetyp (03hex: Drucker, 04hex: Laufwerk). Die erste Unterfunktion (AL gleich 00hex) ermittelt den Umleitungsstatus des spezifizierten Geräts. Nach dem Funktionsaufruf gibt sie an, ob die Umleitung aktiviert (BH gleich 00hex) oder deaktiviert ist (BH gleich 01hex).

Die zweite Unterfunktion (AL gleich 01hex) setzt den Umleitungsstatus des spezifizierten Geräts. Vor dem Funktionsaufruf legen Sie fest, ob die Umleitung aktiviert (BH gleich 00hex) oder deaktiviert werden soll (BH gleich 01hex).

Funktion 60hex: Dateispezifikation anpassen

Mit der Funktion 60hex wandeln Sie einen Datei- oder Pfadnamen unter Berücksichtigung von Join-, Subst-, Assign- und Netzwerkkommandos in einen angepaßten Namen um.

Als Ergebnis erhalten Sie im Carry-Register einen Wert, der Ihnen Aufschluß darüber gibt, ob das Betriebssystem den Dateinamen erfolgreich umgewandelt hat. Hat es ihn angepaßt, löscht das Betriebssystem das Carry-Statusbit. Wenn bei der Umwandlung Fehler auftreten, dann wird das Carry-Statusbit gesetzt, und AX enthält den Fehlercode.

Um eine Dateispezifikation umzuwandeln, muß das AH-Register den Wert 60hex beinhalten und die Registerkombination DS:SI auf den Dateinamen zeigen, der mit einem Null-Byte abgeschlossen wird. ES:DI gibt einen 128 Byte großen Puffer an, in dem das Betriebssystem den angepaßten Namen ablegen kann.

Offset	Beschreibung
00hex	Informationsebene (auf 0 gesetzt)
02hex	binäre Seriennummer des Datenträgers
06hex	Datenträgerbezeichnung oder »NO NAME«
11hex	nur für die Lesefunktion: Art des Dateisystems (»FAT12« oder »FAT16«)

Tabelle 8. Die Laufwerk-Informationstabelle nimmt die Datenträgerbezeichnung

Die Funktion 61hex: beinhalten verschiedene Routinen

Die Funktion 61hex kehrt sofort wieder in das aufrufende Programm zurück und führt keine Operation aus.

Die Funktion 62hex: stellt die Adresse des aktuellen PSP fest. Die Adresse des Eingabepuffers übergibt das Unterprogramm mit dem Zeiger BX:00hex an die Hauptroutine. Das bedeutet, BX enthält die Segmentadresse des PSP.

Die Funktion 63hex: unterstützt fremdsprachliche 2-Byte-Zeichensätze. Da sie ausschließlich bei asiatischen MS-DOS-Versionen Verwendung findet, gehen wir nicht näher auf sie ein.

Die undokumentierte Funktion 64hex: setzt beziehungsweise liest ein unbekanntes Statusbit ab MS-DOS 3.2. Wenn AL 0 enthält, übergibt die Routine in DL den Wert des unbekannten Statusbits. Ist AL hingegen gleich 1, setzt die Prozedur das unbekannte Statusbit auf den in DL spezifizierten Wert. Befindet sich in AL der Wert 2, setzt die Funktion das unbekannte Statusbit auf den in DL übergebenen Wert und speichert wiederum in DL den alten Wert des unbekannten Parameters.

Die Funktion 65hex: Funktionen für die Bearbeitung von Zeichen

wandelt einen Dateinamen oder eine Zeichenkette ab MS-DOS 4.0 in Großbuchstaben um.

Konnte die angegebene Zeichenkette in Großbuchstaben umgewandelt werden, löscht das Betriebssystem das Carry-Statusbit. Schlug die Umwandlung fehl, dann setzt es das Carry-Statusbit und übergibt die Fehlernummer in AX.

Beim Aufruf dieser Funktion muß AL die Unterfunktionsnummer enthalten. Beachten Sie bitte bei den im folgenden beschriebenen Unterfunktionen, daß Sie zusätzlich das Bit 7 des AL-Registers je nach gewünschter Funktion modifizieren. Wenn Sie eine Standard-Zeichenkette in Großbuchstaben umwandeln möchten, löschen Sie Bit 7, wollen Sie hingegen einen Dateinamen verändern, müssen Sie Bit 7 setzen.

Unterfunktion 20hex (AL gleich 20hex) wandelt das in DL übergebene Zeichen in einen Großbuchstaben um. Nach dem Aufruf enthält DL das umgewandelte Zeichen. Die Unterfunk-

tion 21hex (AL gleich 21hex) wandelt eine Zeichenkette um, deren Position in der Registerkombination DS:DX abgelegt ist. CX enthält vor dem Funktionsaufruf die Länge der Zeichenkette.

Die Unterfunktion 22hex wandelt eine Zeichenkette im ASCII-Format in Großbuchstaben um. Auch hier gibt DS:DX die Lage der Zeichenkette an.

Funktion 67hex bis 63hex: verschiedene Handle-Routinen

Die Funktion 67hex setzt die maximale Handle-Anzahl auf einen neuen Wert. Dieser Parameter wird von MS-DOS immer auf den Mindestwert 20 gesetzt, den auch diese Funktion nicht verringern kann.

Als Ergebnis erhalten Sie im Carry-Statusbit einen Wert, der Ihnen mitteilt, ob das Betriebssystem die Handle-Anzahl angenommen hat. Ist das der Fall, löscht die Funktion das Carry-Statusbit. Andernfalls setzt das Unterprogramm das Carry-Statusbit und legt in AX den Fehlercode ab.

Beim Aufruf dieser Funktion muß das AH-Register den Wert 67hex enthalten und BX die neue Handle-Anzahl.

Mit der Funktion 69hex bearbeiten Sie die Seriennummer eines Datenträgers. Konnte die Routine die Funktion nicht fehlerfrei ausführen, setzt sie das Carry-Statusbit. Das AX-Register enthält in diesem Fall den Fehlercode.

Im BL-Register übergeben Sie der Routine das gewünschte Laufwerk. Dabei steht der Inhalt »0« für das Standard-Laufwerk, »1« für »A:«, »2« für »B:«, »3« für »C:« und so weiter. AL bestimmt, ob Sie die Seriennummer lesen (AL gleich 00hex) oder setzen wollen (AL gleich 01hex).

Beim Aufruf dieser Funktion muß das AH-Register den Wert 69hex enthalten und die Registerkombination DS:DX auf eine Laufwerk-Informationstabelle zeigen. Tabelle 8 zeigt Ihnen den geforderten Aufbau der Tabelle.

Die Funktion 6Ahex: kehrt sofort wieder in das aufrufende Programm zurück und führt keine Operation aus.

Die Funktion 6Bhex: kehrt wieder in das aufrufende Programm zurück. Die erfüllte Aufgabe ist nicht bekannt, aber man geht davon aus, daß ihr Aufruf mit installierbaren Dateistrukturen zusammenhängt. Aber den genauen Sinn wird man sicherlich eines Tages entdecken.

(Markus Zietlow/uk)

Gerätetreiber

Individuelle Treiber

Die offene Struktur von MS-DOS, die viel zu dem Erfolg des Betriebssystems beigetragen hat, wird zum Teil von den individuell einsetzbaren Gerätetreibern getragen. Die Realisierung von Gerätetreibern gehört zu den komplexeren Programmieraufgaben. Deshalb beinhaltet ein Grundgerüst alle Standardfunktionen und die Struktur der Systemmodule.

MS-DOS gehört zu den zuverlässigsten und vielfältigsten Betriebssystemen. Einen Großteil seines Erfolgs hat das Betriebssystem dem Umstand zu verdanken, daß es leicht erweiterbar ist. Die Basis für diesen Aufbau bilden die sogenannten Gerätetreiber, die sämtliche Ein- und Ausgaben steuern.

Was sind eigentlich Gerätetreiber und welche Aufgaben übernehmen sie in einem PC? Um diese Fragen zufriedenstellend zu beantworten, bedarf es einiger Vorkenntnisse.

Um mit Computern sinnvoll zu arbeiten, benötigen Sie ein Betriebssystem. Ein Betriebssystem ist eine Sammlung von Routinen und Unterprogrammen, die ein Anwendungsprogramm in die Lage versetzen, alle Fähigkeiten und Bestandteile eines Computers auszunutzen. Damit das Anwendungsprogramm auf vielen Computern lauffähig ist, stellt das Betriebssystem eine standardisierte Schnittstelle zur Verfügung. Das hat mehrere Vorteile. Erstens müssen die Programmierer nicht die Hardware des Computers direkt ansprechen, sondern sie arbeiten mit den standardisierten Prozeduren. Das garantiert, daß die Programme auch auf späteren Versionen des Betriebssy-

stems lauffähig sind, da die Entwickler die Benutzerschnittstelle nicht ändern, sondern höchstens um zusätzliche Funktionen erweitern. Zweitens ist jedes Anwendungsprogramm auf verschiedenen Computern mit kompatiblen Mikroprozessoren lauffähig, obwohl sich die Betriebssysteme voneinander stark unterscheiden. Die installierte Software ist also hardwareunabhängig. Da ein Betriebssystem sehr pauschal gehalten ist, deckt es natürlich auch nur die Ansteuerung der Standard-Ein- und Ausgabegeräte ab. An diesem Punkt beginnt die Aufgabe der Gerätetreiber.

Treiber verbinden das System mit der Außenwelt

Ein Gerätetreiber (device driver) ist ein hardwareabhängiges Programm, das ein Peripheriegerät steuert und die Kontrolle über diese zusätzliche Einheit an die standardisierte Schnittstelle des Betriebssystems anpaßt. Da der Gerätetreiber hardwareabhängig ist, läßt er sich natürlich nicht so leicht auf andere Systeme übertragen.

Ein Gerätetreiber ist also im Grunde nichts anderes als eine standardisierte

Methode der Datenübergabe. Mit Gerätetreibern kann man bei entsprechender Programmierung viele Anwendungen realisieren. So lassen sich beispielsweise RAM-Disks einrichten, CD-ROM-Laufwerke ansprechen, Meßgeräte ansteuern oder komplette Standardeinheiten wie etwa die Tastatur neu belegen.

Das Betriebssystem stellt – wie schon gesagt – mehrere Standardtreiber zur Verfügung, deren Namen und Funktionen Sie aus Tabelle 1 ersehen. Jedes MS-DOS-System enthält eingebaute Treiber für Tastatur, Bildschirmzeige, serielle Schnittstelle, parallelen Ausgang, Echtzeituhr und mindestens eine Speichereinheit, von der das Betriebssystem gestartet wird. Der Computer lädt diese als »residente Treiber« bezeichneten Standardtreiber als Gruppe aus der Datei »io.sys«. MS-DOS ist sinnvollerweise so aufgebaut, daß jeder Benutzer die Standardtreiber mit eigenen Routinen überlagern kann.

Gerätetreiber besitzen viele Anwendungsgebiete

Man unterscheidet grundsätzlich zwei Klassen von Gerätetreibern: zeichenorientierte und blockorientierte. Die zeichenorientierten Treiber steuern Geräte wie beispielsweise den Monitor, die Tastatur oder einen Drucker. Die blockorientierten Treiber kontrollieren hingegen Peripheriegeräte wie Diskettenlaufwerke, Festplatten oder CD-ROMs.

Als Beispiel für einen installierbaren Treiber ist auf der MS-DOS-Diskette der ANSI-Treiber gespeichert, der die Tastatureingabe und Bildschirmausgabe erleichtert. Sie binden ihn wie jeden anderen Treiber mit der Anweisung

```
device=(Name des Treibers)
```

also in diesem Fall

```
device=ansi.sys
```

in die MS-DOS-Umgebung ein, die zuerst in den Arbeitsspeicher geladen wird. Im Gegensatz zu den üblichen Funktionen leitet das Betriebssystem alle Ausgaben über den ANSI-Treiber

DEV-Test/Gerätetreiber-Analyse aus der »DOS international«/Autor: Markus Zietlow

Nr Header Name Attr Strt Intr Treiber-Anzahl: 19

Nr	Header	Name	Attr	Strt	Intr
1	0116:0048	NUL	8004	0DC6	0DCC
2	1752:0000	MSHOUSE	8000	0016	0021
3	1412:0000	SMARTAAR	C800	00A2	00AD
4	130B:0000	CON	C053	00C1	00CC
5	0C41:0000	CON	C053	057F	05B7
6	0C27:0000	SETVERXX	8000	001E	0029
7	0B5E:0000	XMSXXXX	A000	0040	004B
8	0070:0023	CON	8013	06F5	0700
9	0070:0035	AUX	8000	06F5	0721
10	0070:0047	PRN	A0C0	06F5	0705
11	0070:0059	CLOCK\$	8008	06F5	0739
12	0070:006B	<Blk:83>	80C2	06F5	073E
13	0070:007B	COM1	8000	06F5	0721
14	0070:008D	LPT1	A0C0	06F5	070C
15	0070:009F	LPT2	A0C0	06F5	0713
16	0070:00B8	LPT3	A0C0	06F5	071A
17	0070:00CA	COM2	8000	06F5	0727
18	0070:00DC	COM3	8000	06F5	072D
19	0070:00EE	COM4	8000	06F5	0733

Gerätetreiber-Struktur

FEDCBA9876543210

Attribut-Bits: 1000000000000000

Bit	Bedeutung	Wert
0	Standard-Eingabegerät?	[N]
1	Standard-Ausgabegerät?	[N]
2	NUL-Gerät?	[N]
3	Uhr-Gerät?	[N]
4	Unterstützt INT 29h?	[N]
6	Unterstützt IOCTL?	[N]
8	Versteht "Öffnen/Schließen"?	[N]
D	Unterstützt "Until busy"?	[N]
E	Kontroll-Zeichenketten?	[N]
F	Zeichen- oder Blocktreiber?	[Z]

Wählen Sie mit den Pfeiltasten einen Gerätetreiber! - Abbruch: Escape-Taste

Das Programm »Dev-Test« gibt Aufschluß über die geladenen Gerätetreiber und ihren Aufbau.

Reservierte Gerätenamen

Name	Funktion
AUX	serielles Zusatzgerät (COM1:)
CON	Konsole (Tastatur und Monitor) in »config.sys« vom Anwender definierte Treiber
NUL	Null-Device
PRN	paralleles Zusatzgerät (LPT1:)
[Uhr]	Zeit-Device (Systemuhr mit 18,2 Impulsen pro Sekunde)

Tabelle 1. Einige Gerätetreiber sind vom Betriebssystem vorgegeben.

Gerätetreiber-Handles

Adresse	Länge	Name	Funktion
0hex	32 Byte	Nächster Gerätetreiber	Adresse des nächsten Gerätetreibers beziehungsweise 0FFFF:FFFFhex für den letzten Treiber
4hex	16 Byte	Geräte-Attribut-Code	siehe Tabellen 3 und 4
6hex	16 Byte	Geräte-Strategie-Zeiger	Offset-Adresse, die auf Strategie-Routine zeigt
8hex	16 Byte	Geräte-Interrupt-Zeiger	Offset-Adresse, die auf Interrupt-Routine zeigt
Ahex	8 Byte	Gerätename	Gerätename in Form von ASCII-Zeichen; bei Blocktreibern bestimmt das erste Byte die Anzahl der Geräteeinheiten

Tabelle 2. Gerätetreiber-Handles sind für die Initialisierung notwendig.

um. Dieser Treiber teilt die umgeleiteten Zeichen in zwei Gruppen auf; die eine Gruppe besteht aus den Zeichen, die auf dem Bildschirm erscheinen sollen, und die andere Gruppe ist aus Befehlen aufgebaut. Der ANSI-Treiber filtert aus den Bildschirmdaten die entsprechenden Informationen heraus und gibt sie dann an die Ausgabereinheit weiter. So kennzeichnet beispielsweise das vorangestellte ASCII-Zeichen 27 beim ANSI-Treiber einen Befehl. Die nachfolgenden Befehle bestimmen, ob es sich um ein Kommando zum Löschen des Bildschirms, Ändern der Farbe, Positionieren des Cursors oder ähnliches handelt.

Die Verwendung des ANSI-Treibers bringt zwei wesentliche Vorteile mit sich. Der erste Vorteil liegt darin, daß anstelle der BIOS-Routinen die BIOS-ähnlichen Routinen des Treibers benutzt werden können. Des weiteren laufen alle Programme, die den ANSI-Treiber anstelle der BIOS-Routinen benutzen, auf allen MS-DOS-Computern. Der wesentliche Nachteil des ANSI-Treibers und aller anderen zusätzlichen Gerätetreiber ist, daß die Programme zu den Computern, auf

denen der Treiber nicht installiert ist, nicht kompatibel sind.

Beim Hochfahren werden die Gerätetreiber zuerst geladen

Das Programmieren eines Gerätetreibers ist ziemlich aufwendig, da man einen Fehler in der Struktur kaum lokalisieren kann. Das Betriebssystem installiert die Gerätetreiber automatisch, wenn »command.com« geladen wird. Es ist schwierig, diesen Vorgang mit einem Debugger oder einem ähnlichen Hilfsprogramm nachzuvollziehen. Aus diesem Grund müssen Sie beim Aufbau eines Gerätetreibers besonders sorgfältig vorgehen. Es ist sinnvoll, den Standardaufbau eines Treibers zu übernehmen und daraufhin die Details Schritt für Schritt hinzuzufügen.

Jeder Gerätetreiber besteht grundsätzlich aus drei Teilen. Der erste Teil ist der sogenannte Gerätevorspann (»Device Header«), der den Namen des Treibers bestimmt und die Verbindung zu den anderen Teilen des Treibers herstellt. Der zweite Abschnitt ist der Strategieblock (»Device Strategie-

gy«), der einen vom Betriebssystem übergebenen Anforderungsblock (»Request Header«) kontrolliert. Als dritter und letzter Bestandteil fungiert der Interrupt-Block (englisch »Device Interrupt Handler«), der die Programmroutine des Treibers enthält.

Um den Aufbau eines Gerätetreibers zu verstehen, analysieren Sie am besten einen bereits bestehenden. Für diesen Zweck finden Sie in Listing 1 einen einfachen Gerätetreiber, der nichts anderes macht, als bei seiner Initialisierung eine Meldung auf dem Bildschirm auszugeben. Bevor wir auf Details eingehen, folgen einige Hinweise auf die Übersetzung des Assembler-Listings.

Ein Gerätetreiber muß immer als Com-Datei vorliegen

Die Startadresse eines Gerätetreibers ist grundsätzlich auf Null zu beziehen. Das erreichen Sie mit Hilfe des Org-Pseudobefehls (in Listing 1 »org 0h«). Zusätzlich sind die Randbedingungen für Com-Dateien zu erfüllen, das bedeutet, daß Sie kein Stapelsegment definieren dürfen.

Nach der Eingabe des Quelltextes übersetzen Sie das unter dem Namen »device.asm« gespeicherte Listing 1 mit dem Masm-Assembler und dem Linker. Falls keines der Programme einen Fehler beanstandet hat, wandeln Sie das Exe-Programm mit dem Exe2bin-Utility in eine Com-Datei um. Um Gerätetreiber von anderen ausführbaren Programmen zu unterscheiden, passen Sie die Erweiterung an:

ren device.com device.sys

Zuletzt tragen Sie Pfad und Namen des Gerätetreibers in die Datei »config.sys« ein. Wenn die Datei im Haupt-

Geräte-Attribut-Codes für zeichenorientierte Treiber

Code	Name	Bedeutung
0	Eingabe (Konsole)	Treiber ist Standard-Eingabegerät
1	Ausgabe (Konsole)	Treiber ist Standard-Ausgabegerät
2	Null-Device	Treiber ist Null-Device
3	Uhr-Device	Treiber ist Uhr-Device
4	Int 29hex	Treiber unterstützt Ausgabe über Interrupt 29hex
5	reserviert	
6	IOCTL	IOCTL wird nicht unterstützt
7	reserviert	
8	reserviert	
9	reserviert	
10	reserviert	
11	Öffnen/Schließen	Treiber versteht dieses Kommando
12	reserviert	
13	Ausgabe bis »beschäftigt«	Treiber unterstützt »until busy«
14	Kontroll-Zeichenketten	Treiber kann Kontroll-Zeichenketten bearbeiten
15	zeichenorientierter Treiber	

Tabelle 3. Der Attribut-Code übernimmt die Klassifizierung des Geräts.

Geräte-Attribut-Codes für blockorientierte Treiber

Code	Inhalt	Bedeutung
0	reserviert	
1	32-Bit-Sektoren	unterstützt 32-Bit-Sektoren-Adressierung
2	reserviert	
3	reserviert	
4	reserviert	
5	reserviert	
6	IOCTL	IOCTL wird nicht unterstützt
7	reserviert	
8	reserviert	
9	reserviert	
10	reserviert	
11	Öffnen/Schließen	Treiber versteht dieses Kommando
12	reserviert	
13	Begrenzer des Medium-Typs	FAT-ID-Byte wird benutzt
14	Kontroll-Zeichenketten	Kontroll-Zeichenketten können bearbeitet werden
15	Gerätetyp	kein blockorientierter Treiber

Tabelle 4. Der Attribut-Code eines blockorientierten Treibers unterscheidet sich deutlich von einem Zeichentreiber.

Gerätetreiber-Request-Header

Offset	Länge (Byte)	Name	Funktion
0	1	Länge eines Datensatzes	Länge des Request-Headers in Byte
1	1	Einheits-Code	Untereinheit
2	1	Funktions-Code	siehe Tabelle 6
3	2	Status	übernimmt Status der Funktion
5	8	reserviert	

Tabelle 5. Der Request-Header kontrolliert die Ausführung eines Device-Kommandos.

verzeichnis der Festplatte abgelegt ist, lautet die Anweisung

```
device = c:\device.sys
```

Danach starten Sie Ihren Computer neu. Bei fehlerfreiem Betrieb erscheint die Einschaltmeldung des Beispieldrivers nach kurzer Zeit auf dem Bildschirm, das Betriebssystem befindet sich in der Kommandoebene.

Als ersten Teil des Gerätetreibers finden Sie in Listing 1 den Gerätevorspann, dessen Aufbau Tabelle 2 wiedergibt. Der Vorspann ist 18 Byte lang und besteht aus fünf Feldern. Die ersten beiden Felder (zwei 16-Bit-Werte) geben die Position des folgenden Gerätetreibers an. Damit das Betriebssystem die Adressierung automatisch übernimmt, legen Sie in beiden Feldern vor der Initialisierung den Wert »-1« beziehungsweise »FFFFhex« ab. Wenn ein aktiver Gerätetreiber in diesen Systembereichen die gleichen Werte aufweist, handelt es sich bei dem aktuellen Treiber um den zuletzt geladenen.

Feld drei (ein 16-Bit-Wert) enthält das dem jeweiligen Treiber zugeordnete Attribut-Byte, dessen Inhalt für zeichenorientierte Treiber in Tabelle 3 und für blockorientierte Treiber in Tabelle 4 aufgelistet ist. Das Attribut-Byte legt die Klassifizierung des Treibers fest.

Das vierte und fünfte Feld (wiederum zwei 16-Bit-Werte) bestimmen die Adressen der Strategie- und der Interrupt-Routinen, auf deren Bedeutung wir im folgenden noch eingehen.

Das Attribut-Byte klassifiziert den Gerätetreiber

Das sechste und letzte Feld enthält schließlich den Namen des Gerätetreibers. Der Name muß genau acht Zeichen lang sein, fehlende Buchstaben füllen Sie mit Leerzeichen auf. Bei einem blockorientierten Treiber ist nur das erste Byte signifikant. Es enthält die Anzahl der Einheiten, die der Treiber bearbeiten kann. Ein Blocktreiber ist nicht mit einem Namen gekennzeichnet.

Der zweite Teil des Gerätetreibers ist der Strategieblock, der aus fünf Programmzeilen aufgebaut ist. Sie finden

ihn in Listing 1 unter dem Namen »strategie«. Wenn das Betriebssystem einen Treiber lädt, baut es den Anforderungsblock auf. Dieser erfüllt zwei Funktionen: Erstens agiert er als Datenbereich für MS-DOS; des weiteren stellt er die Schnittstelle dar, durch die das aufrufende Programm Informationen mit dem Gerätetreiber austauscht.

Die Strategie-Routine wird zur Initialisierung benötigt

Das Betriebssystem ruft die Strategie-Routine einmal während der Initialisierung des Treibers auf. Es ist die Aufgabe der Prozedur, die Adresse des Anforderungsblocks zu ermitteln und zwischenspeichern, so daß die Unterprogramme des Treibers immer auf diesen wichtigen Datenblock zugreifen können. In unserem Beispiel ist das Doppelwort »doszeiger« als Zwischenspeicher vorgesehen. In Tabelle 5 finden Sie den typischen Aufbau eines Anforderungsblocks.

Der dritte und letzte Bestandteil des Gerätetreibers ist der Interrupt-Block. Er übernimmt drei Aufgaben. Zum einen enthält er eine Befehlstabelle, die die Adressen der einzelnen Treiberfunktionen zur Verfügung stellt (in Listing 1 mit der Adreßangabe »tabelle« eingeleitet). Zum anderen beinhaltet er zu jedem Tabelleneintrag die für einen reibungslosen Programmablauf verantwortliche Routine. In unserem Fall löschen die am Ende des Listings 1 befindlichen Routinen das Fehlerregister und kehren ins Hauptprogramm zurück. Schließlich ist der In-

Befehlscodes

Kommando- docode	Funktion
00hex	Initialisierung
01hex	Prüfung auf Medienwechsel (nur Blocktreiber)
02hex	BIOS-Parameterblock erzeugen (nur Blocktreiber)
03hex	Ein-/Ausgabegesteuertes Lesen
04hex	Lesen von Gerät
05hex	nicht zerstörendes Lesen (nur Zeichentreiber)
06hex	Lesestatus (nur Zeichentreiber)
07hex	Lesepuffer löschen (nur Zeichentreiber)
08hex	Schreiben
09hex	Schreiben mit Überprüfung
0Ahex	Schreibstatus (nur Zeichentreiber)
0Bhex	Schreibpuffer leeren (nur Zeichentreiber)
0Chex	Ein-/Ausgabegesteuertes Schreiben
0Dhex	Gerät öffnen
0Ehex	Gerät schließen
0Fhex	austauschbares Medium (nur Blocktreiber)
10hex	Ausgabe, bis nicht mehr aufnahmebereit (»busy«)
11hex	reserviert
12hex	reserviert
13hex	IOCTL-Aufbau für Blocktreiber
14hex	reserviert
15hex	reserviert
16hex	reserviert
17hex	Laufwerk Aufbau lesen (nur Blocktreiber)
18hex	Laufwerk Aufbau setzen (nur Blocktreiber)

errupt-Handler dafür verantwortlich, den Befehlscode aus dem Anforderungsblock zu lesen, die entsprechende Funktion auszuführen und schließlich bei Auftreten eines Fehlers das Statusfeld zu ändern. In Listing 1 übernehmen diese Aufgabe die ersten Anweisungen in der Prozedur »interrupt«.

Die Kommandos, die der Gerätetreiber bereitstellt, sind in Tabelle 6 aufgeführt. Zu beachten ist, daß viele Funktionen nur für einen Treibertyp (zeichen- oder blockorientiert) gültig sind.

Nach der Ausführung eines Befehls ersehen Sie aus dem Statusfeld, dessen Aufbau Tabelle 7 zeigt, ob der Treiber eine Operation erfolgreich durchgeführt hat oder nicht. Bei einem Fehler läßt sich leicht feststellen, was zum Programmabbruch geführt hat. Tabelle 8 gibt Aufschluß über die Fehlerursache.

Wenn Sie einen Treiber erfolgreich installiert haben, sprechen Sie ihn über die Funktionen des Interrupts 21hex an. Die Wahl der Kommandos hängt davon ab, ob der Treiber einen Standardtreiber ersetzt oder als zusätzlicher Treiber fungiert.

Um Ihnen den Einblick in den Aufbau von Gerätetreibern zu erleichtern, erhalten Sie mit dem in Listing 2 abgedruckten Programm ein Utility zur Analyse der aktiven Gerätetreiber: »Dev-Test« (Bild).

Dev-Test läuft unter Quick-Basic ab Version 4.0 und Basic-PDS ab Version 7.0, läßt sich jedoch leicht in andere Programmiersprachen umwandeln.

Die grundlegenden Prozeduren halten sich an Standardkonventionen. Das

Statusfeld des Request-Headers

Bit-Nr.	Inhalt	Bedeutung
0 bis 7	Fehler-Code	siehe Tabelle 8
8	beendet	Operation beendet
9	beschäftigt	Treiber beschäftigt (»busy«)
10	reserviert	
11	reserviert	
12	reserviert	
13	reserviert	
14	reserviert	
15	Fehlerstatus	Fehler aufgetreten

Tabelle 7. Das Statusfeld gibt an, ob eine Funktion erfolgreich durchgeführt wurde.

Interessanteste an dem Utility sind die Unterprogramme »ReadListOfLists« und »ReadNextDriver«. »ReadListOfLists« benutzt die undokumentierte Funktion 52hex des Interrupts 21hex. Wenn Sie diese Routine aufrufen, erhalten Sie im Register ES:BX die Adresse der sogenannten »Liste der Listen«. In dieser Liste sind wichtige Informationen zur internen Struktur von MS-DOS abgelegt. Darunter befinden sich zum Beispiel Angaben über die aktuelle Verzeichnisstruktur, das Boot-Laufwerk oder den erweiterten Arbeitsspeicher.

Uns interessiert vor allem der 18 Byte große Speicherbereich ab Offset 22hex. In ihm ist der Gerätevorspann des ersten Treibers, des NUL-Trei-

Fehlercodes

Code	Fehlername
0hex	schreibgeschützt
1hex	unbekannte Geräteeinheit
2hex	Laufwerk nicht bereit
3hex	unbekanntes Kommando
4hex	CRC-Fehler
5hex	falsche Länge der Request-Struktur (Laufwerk)
6hex	Fehler beim Suchen
7hex	unbekanntes Medium (Gerätetyp)
8hex	Sektor nicht gefunden
9hex	Drucker ohne Papier
Ahex	Schreibfehler
Bhex	Lesefehler
Chex	allgemeiner Fehler
Dhex	reserviert
Ehex	reserviert
Fhex	ungültiger Laufwerkwechsel

Tabelle 8. Mit den Fehlercodes grenzen Sie den Fehlerbereich ein.

bers, abgelegt. Durch ihn erfahren wir über die Prozedur »ReadNextDriver« jeweils die Adresse des folgenden Treibers. Wie Sie sich erinnern, gibt das erste Doppelwort eines Gerätetreibers die Adresse des nächsten Treibers an. Auf diese Weise erhalten wir eine Liste aller aktiven Gerätetreiber. Die Auswertung des Attribut-Bytes führt zur Entschlüsselung der Treiberstruktur.

Sehr aufschlußreich sind auch die Funktionen »HexAnpassung\$« und

»Binaer\$«. »HexAnpassung« macht aus der übergebenen Langzahl eine Hexadezimalzahl mit der angegebenen Anzahl von Stellen. »Binaer\$« folgt dem gleichen Prinzip, übergibt aber als Resultat eine Binärzahl. Beachten Sie bitte, daß Sie Quick-Basic mit dem Befehl

```
qb /1
```

starten müssen, um die Bibliothek einzubinden.

Wenn Sie das Listing betrachten, fällt Ihnen auf, daß viele Zahlen mit dem Suffix »&« gekennzeichnet sind. Nur in diesem Fall arbeiten die angegebenen Basic-Versionen fehlerfrei mit großen Zahlen.

Wenn Sie beispielsweise im Direktmodus

```
print 255*200
```

eingeben, meldet die Entwicklungsumgebung einen Overflow-Fehler. Benutzen Sie hingegen

```
print 255&*200&
```

erhalten Sie das richtige Ergebnis: 51000.

Gerätetreiber gehören zu den wichtigsten und interessantesten Bestandteilen des Betriebssystems. Wenn Sie sich näher mit dem Themenkomplex beschäftigen, stoßen Sie auf viele interessante Anwendungsgebiete, die den etwas kargen Befehlsumfang von MS-DOS sinnvoll erweitern.

(Markus Zietlow/uk)

```
[L24] 1: ; Programm: devicell.asm
[757] 2: ; Funktion: Gerüst eines Standard-Geräte-
[ZE2] 3: ; treibers
[H44] 4: ; Sprache: TASM oder MASM
[L44] 5: ; Autor: Markus Zietlow
[7X4] 6: ; (c) 1992 DMV Widuch GmbH & Co. KG
[000] 7:
[PP2] 8: ;-Segmentanfang
[035] 9: code segment byte public 'code'
[ZM3] 10: ;-Segmentzuweisungen
[SC5] 11: assume cs:code,ds:code,es:nothing
[WH1] 12: ;-Offset 0
[S31] 13: org 0h
[EK4] 14: ;-Zeiger auf den Anfang des
[SK4] 15: ; nächsten Gerätetreibers
[F81] 16: Header:
[TV0] 17: dd -1
[4F1] 18: dw 0c840h
[RU6] 19: ;-Bit 15=1: Treiber ist zeichenorientiert
[1M5] 20: ;-Bit 14=1: Kontroll-Zeichenketten
[BX2] 21: ; akzeptiert
[D47] 22: ;-Bit 13=0: "Until Busy" nicht unterstützt
[HG3] 23: ;-Bit 12=0: reserviert
[087] 24: ;-Bit 11=1: "Öffnen/Schließen" akzeptiert
[FG3] 25: ;-Bit 10=0: reserviert
[ME3] 26: ;-Bit 9=0: reserviert
[LE3] 27: ;-Bit 8=0: reserviert
[KE3] 28: ;-Bit 7=0: reserviert
[VA4] 29: ;-Bit 6=1: IOCTL akzeptiert
[GE3] 30: ;-Bit 5=0: reserviert
[HM6] 31: ;-Bit 4=0: unterstützt INT 29hex nicht
[7B5] 32: ;-Bit 3=0: Device kein Uhrtreiber
[GL5] 33: ;-Bit 2=0: Treiber kein NUL-Device
[XH4] 34: ;-Bit 1=0: Treiber ist nicht
[YR4] 35: ; Standard-Ausgabegerät
[WH4] 36: ;-Bit 0=0: Treiber ist nicht
[NP4] 37: ; Standard-Eingabegerät
[Y46] 38: ;-Startadresse der Strategieroutine
[6F2] 39: dw strategie
[NL6] 40: ;-Anfangsadresse der Interrupt-Routine
[UH2] 41: dw interrupt
[Z43] 42: ;-Name des Treibers
[XA2] 43: db 'DOSINTER'
[RE5] 44: ;-Zwischenspeicher für Adresse:
[0P3] 45: ; Anforderungstabelle
```

```
[SC2] 46: doszeiger dd ?
[XL6] 47: ; Befehlstabelle für Treiberkommandos
[2C1] 48: tabelle:
[665] 49: ;-Initialisierung des Treibers
[DE1] 50: dw init
[T64] 51: ;-Prüfung Medienwechsel
[RK2] 52: dw media_check
[YF5] 53: ;-Aufbau des BIOS-Parameterblocks
[YX2] 54: dw bpb_erzeugen
[995] 55: ;-Ein-/Ausgabe-gesteuertes Lesen
[EF2] 56: dw ioctl_ein
[Z81] 57: ;-lesen
[BV1] 58: dw eingabe
[SU4] 59: ;-nicht-zerstörendes Lesen
[W62] 60: dw nz_lesen
[C52] 61: ;-Lesestatus
[BN2] 62: dw ein_status
[VP3] 63: ;-Lesebuffer löschen
[UH2] 64: dw ein_leeren
[UZ1] 65: ;-schreiben
[HV1] 66: dw ausgabe
[R05] 67: ;-schreiben mit Überprüfen
[092] 68: dw aus_test
[GV2] 69: ;-Schreibstatus
[EN2] 70: dw aus_status
[1W3] 71: ;-Schreibpuffer leeren
[0K2] 72: dw aus_leeren
[GZ5] 73: ;-Ein-/Ausgabe-gesteuertes Schreiben
[YG2] 74: dw ioctl_aus
[UR2] 75: ;-Gerät öffnen
[WY1] 76: dw devopen
[XN3] 77: ;-Gerät schließen
[B42] 78: dw devclose
[065] 79: ;-Öffnen/Schließen verstehen
[842] 80: dw remmedia
[BD3] 81: ;-Ausgabe "until busy"
[512] 82: dw outbusy
[U42] 83: ;-reserviert
[4S1] 84: dw fehler
[U42] 85: ;-reserviert
[4S1] 86: dw fehler
[5L2] 87: ;-Generic IOCTL
```

Listing 1. Jeder Gerätetreiber orientiert sich grundsätzlich an dieser Standardstruktur.


```
[G52] 88:      dw      genIOCTL
[U42] 89: ; -reserviert
[4S1] 90:      dw      fehler
[U42] 91: ; -reserviert
[4S1] 92:      dw      fehler
[U42] 93: ; -reserviert
[4S1] 94:      dw      fehler
[DC5] 95: ; -logisches Laufwerk ermitteln
[HR3] 96:      dw      hole_loglaufwerk
[6L5] 97: ; -logisches Laufwerk einstellen
[G04] 98:      dw      setze_loglaufwerk
[VE7] 99: ; -Strategie-Routine des Einheitentreibers,
[3R6] 100: ; die die Adresse der Anforderungstabelle
[225] 101: ; temporär zwischenspeichert
[2F3] 102: ; -Anfang der Prozedur
[813] 103: strategie proc far
[BR2] 104: ; -Adresse ablegen
[AL5] 105:      mov     word ptr cs:[doszeiger], bx
[CW4] 106: ; -Segment zwischenspeichern
[FS5] 107:      mov     word ptr cs:[doszeiger+2], es
[TV4] 108: ; -Rückkehr ins Hauptprogramm
[3P0] 109:      ret
[R63] 110: ; -Ende der Prozedur
[BE2] 111: strategie endp
[2F3] 112: ; -Anfang der Prozedur
[E43] 113: interrupt proc far
[EA5] 114: ; -Alle wichtigen Register mit den
[AP5] 115: ; Betriebssystemdaten durch Push-
[7H3] 116: ; Operationen sichern
[F91] 117:      push    ax
[L91] 118:      push    bx
[R91] 119:      push    cx
[V91] 120:      push    dx
[P91] 121:      push    ds
[U91] 122:      push    es
[C91] 123:      push    di
[8B1] 124:      push    si
[B91] 125:      push    bp
[MH3] 126: ; -Den DOS-Zeiger laden
[K91] 127:      push    cs
[G71] 128:      pop     ds
[7D3] 129:      les     di, [doszeiger]
[3R6] 130: ; -Kommando laden und Adresse der als Wort
[CF5] 131: ; definierten Funktion ermitteln
[C22] 132:      mov     bl, es:[di+2]
[LT1] 133:      xor     bh, bh
[217] 134: ; -Falls Befehlscode zulässig, Sprung zu
[WT5] 135: ; 'befehl_ok', ansonsten Fehlercode
[L94] 136: ; setzen und unterbrechen
[BM1] 137:      cmp     bx, 24
[B52] 138:      jle     befehlda
[WW1] 139:      call    fehler
[SK2] 140:      jmp     befehlende
[736] 141: ; -Datensegment neu festlegen und zum
[545] 142: ; gewünschten Befehl springen
[FW2] 143: befehlda: shl     bx, 1
[EE4] 144:      call    word ptr [bx+tabelle]
[7D3] 145:      les     di, [doszeiger]
[SR5] 146: ; -Den Status 'beendet' setzen und im
[5U6] 147: ; Status-Byte des Request-Headers ablegen
[FL3] 148: befehlende: or     ax, 0100h
[XZ2] 149:      mov     es:[di+3], ax
[PY5] 150: ; -Alle gesicherten Register mit den
[RH5] 151: ; Betriebssystemdaten durch Pop-
[M75] 152: ; Operationen wiederherstellen
[571] 153:      pop     bp
[291] 154:      pop     si
[671] 155:      pop     di
[M71] 156:      pop     es
[G71] 157:      pop     ds
[N71] 158:      pop     dx
[H71] 159:      pop     cx
[D71] 160:      pop     bx
[971] 161:      pop     ax
[TV4] 162: ; -Rückkehr ins Hauptprogramm
[3P0] 163:      ret
[CL4] 164: ; -In jeder der vorgestellten
[5S3] 165: ; Standardroutinen das
[LP4] 166: ; Fehler-Flag löschen und ins
[245] 167: ; Hauptprogramm zurückkehren
[CM3] 168: media_check proc near
[WW1] 169:      xor     ax, ax
[3P0] 170:      ret
[8T2] 171: media_check endp
[BW3] 172: bpb_erzeugen proc near
[WW1] 173:      xor     ax, ax
[3P0] 174:      ret
[M03] 175: bpb_erzeugen endp
[9A3] 176: ioctl_ein proc near
[WW1] 177:      xor     ax, ax
[3P0] 178:      ret
[7E2] 179: ioctl_ein endp
[SW2] 180: eingabe proc near
[WW1] 181:      xor     ax, ax
[3P0] 182:      ret
[M12] 183: eingabe endp
[T73] 184: nz_lesen proc near
[WW1] 185:      xor     ax, ax
```

```
[3P0] 186:      ret
[2C2] 187: nz_lesen endp
[7F3] 188: ein_status proc near
[WW1] 189:      xor     ax, ax
[3P0] 190:      ret
[6S2] 191: ein_status endp
[BC3] 192: ein_leeren proc near
[WW1] 193:      xor     ax, ax
[3P0] 194:      ret
[AN2] 195: ein_leeren endp
[8Y2] 196: ausgabe proc near
[WW1] 197:      xor     ax, ax
[3P0] 198:      ret
[432] 199: ausgabe endp
[573] 200: aus_test proc near
[WW1] 201:      xor     ax, ax
[3P0] 202:      ret
[EB2] 203: aus_test endp
[RG3] 204: aus_status proc near
[WW1] 205:      xor     ax, ax
[3P0] 206:      ret
[PT2] 207: aus_status endp
[VD3] 208: aus_leeren proc near
[WW1] 209:      xor     ax, ax
[3P0] 210:      ret
[UP2] 211: aus_leeren endp
[FA3] 212: ioctl_aus proc near
[WW1] 213:      xor     ax, ax
[3P0] 214:      ret
[DE2] 215: ioctl_aus endp
[XZ2] 216: devopen proc near
[WW1] 217:      xor     ax, ax
[3P0] 218:      ret
[T42] 219: devopen endp
[A53] 220: devclose proc near
[WW1] 221:      xor     ax, ax
[3P0] 222:      ret
[L92] 223: devclose endp
[H23] 224: remmedia proc near
[WW1] 225:      xor     ax, ax
[3P0] 226:      ret
[U62] 227: remmedia endp
[E43] 228: outbusy proc near
[WW1] 229:      xor     ax, ax
[3P0] 230:      ret
[A92] 231: outbusy endp
[343] 232: genIOCTL proc near
[WW1] 233:      xor     ax, ax
[3P0] 234:      ret
[C82] 235: genIOCTL endp
[YN4] 236: hole_loglaufwerk proc near
[WW1] 237:      xor     ax, ax
[3P0] 238:      ret
[7U3] 239: hole_loglaufwerk endp
[TU4] 240: setze_loglaufwerk proc near
[WW1] 241:      xor     ax, ax
[3P0] 242:      ret
[RX3] 243: setze_loglaufwerk endp
[NM2] 244: fehler proc near
[NV1] 245:      mov     ax, 8003h
[3P0] 246:      ret
[MX1] 247: fehler endp
[BD7] 248: ; -Anfang des Initialisierungsteils, der bei
[DN5] 249: ; der Installation automatisch vom
[EN5] 250: ; Betriebssystem aufgerufen wird
[TT3] 251: ; -DX=Anfang des Textes
[4U2] 252: init     proc near
[U91] 253:      push    es
[C91] 254:      push    di
[0D4] 255: ; -Mit der DOS-Funktion 9 die
[122] 256: ; Meldung ausgeben
[EF1] 257:      mov     ah, 9
[W04] 258:      mov     dx, offset meldung
[N81] 259:      int     21h
[2T5] 260: ; -Fehler-Flag löschen und Rückkehr
[5C3] 261: ; ins Hauptprogramm
[671] 262:      pop     di
[M71] 263:      pop     es
[D76] 264:      mov     word ptr es:[di+14], offset init
[VC4] 265:      mov     word ptr es:[di+16], cs
[WW1] 266:      xor     ax, ax
[3P0] 267:      ret
[ZG1] 268: init endp
[KD4] 269: ; -Einschaltmeldung für den
[R23] 270: ; Initialisierungsteil
[KX5] 271: meldung db      13, 10, 'Beispiel-Geräte'
[U65] 272:      db      'treiber von Markus Zietlow'
[LE5] 273:      db      ' aus der DOS International'
[R93] 274:      db      ' geladen', 13, 10, '$'
[R63] 275: ; -Ende der Prozedur
[HH2] 276: interrupt endp
[0V3] 277: ; -Ende des Code-Segments
[TT1] 278: code     ends
[LT4] 279: ; -Ende des Assemblerprogramms
[4M0] 280: end
```

Listing 1. Jeder Gerätetreiber orientiert sich grundsätzlich an dieser Standardstruktur.


```

[ØX3] 1: 'Programm: device12.bas
[F76] 2: 'Funktion: erleichtert die Analyse von
[EL3] 3: 'Gerätetreibern
[GM2] 4: 'Sprache: QBasic
[514] 5: 'Autor: Markus Zietlow
[1W4] 6: '(c) 1992 DMV Widuch GmbH & Co. KG
[ØØØ] 7:
[MV4] 8: DECLARE SUB ReadNextDriver ()
[S55] 9: DECLARE SUB ReadListOfLists ()
[GU5] 10: DECLARE FUNCTION Invers$(Zkette$)
[FR8] 11: DECLARE FUNCTION HexAnpassung$(hv AS LONG,
    Stellen!)
[LP6] 12: DECLARE SUB ZeigeListe (Anfang!, Grenze!)
[AR4] 13: DECLARE SUB AktuellInvers ()
[6R4] 14: DECLARE SUB AktuellNormal ()
[S33] 15: DECLARE SUB Init ()
[H63] 16: DECLARE SUB Menue ()
[VZ3] 17: DECLARE SUB Struktur ()
[2T7] 18: DECLARE FUNCTION Binaer$(hv AS LONG, Stell
    en!)
[DB5] 19: DECLARE SUB JaNein (y!, x!, Maske!)
[2N4] 20: DECLARE SUB BelegungBlock ()
[835] 21: DECLARE SUB BelegungZeichen ()
[ZW3] 22: DECLARE SUB PfeilHoch ()
[HE4] 23: DECLARE SUB PfeilRunter ()
[9F1] 24: 'DEV-Test
[Ø37] 25: 'Ein Analyseprogramm für Gerätetreiber
[945] 26: 'aus der »DOS international«
[SA5] 27: 'Programmautor: Markus Zietlow
[2U2] 28: '$INCLUDE: 'qb.bi'
[HY1] 29: 'DEFINT A-Z
[MS6] 30: COMMON SHARED Aktuell, Anzahl, Attribut$( )
[S95] 31: COMMON SHARED BalkenPos, DName$( )
[M27] 32: COMMON SHARED Header$( ), InRegs AS RegTypeX
[2H6] 33: COMMON SHARED IntEntry$( ), Offset AS LONG
[RL5] 34: COMMON SHARED OutRegs AS RegTypeX
[LW6] 35: COMMON SHARED Segment AS LONG, Strategie$( )
[RW2] 36: VIEW PRINT 1 TO 25
[7L1] 37: COLOR 15, Ø
[PG1] 38: Anzahl = Ø
[ØWØ] 39: Menue
[5X2] 40: ReadListOfLists
[K14] 41: DO WHILE Offset <> &HFFFF&
[AT2] 42: DEF SEG = Segment
[ZZ2] 43: Anzahl = Anzahl + 1
[KK2] 44: ReadNextDriver
[HSØ] 45: LOOP
[6Ø6] 46: DIM Attribut$(Anzahl), Header$(Anzahl)
[VX5] 47: DIM DName$(Anzahl), IntEntry$(Anzahl)
[YM3] 48: DIM Strategie$(Anzahl)
[MRØ] 49: Init
[XN1] 50: LOCATE 2, 56
[522] 51: PRINT Anzahl
[5X2] 52: ReadListOfLists
[WT1] 53: Zaehler = Ø
[K14] 54: DO WHILE Offset <> &HFFFF&
[AT2] 55: DEF SEG = Segment
[E93] 56: Zaehler = Zaehler + 1
[8FB] 57: Header$(Zaehler)=HexAnpassung$(Segment, 4
    )+": "+HexAnpassung$(Offset, 4)
[MDB] 58: Attribut$(Zaehler)=HexAnpassung$(PEEK(Off
    set+4)+PEEK(Offset+5)*256&, 4)
[KAB] 59: Strategie$(Zaehler)=HexAnpassung$(PEEK(Of
    fset+6)+PEEK(Offset+7)*256&, 4)
[CØC] 60: IntEntry$(Zaehler) = HexAnpassung$(PEEK(O
    fset + 8) + PEEK(Offset + 9) * 256&, 4)
[X57] 61: IF VAL("&H" + Attribut$(Zaehler)) AND &H8
    ØØØ& THEN
[P52] 62: FOR q = 1Ø TO 17
[BP8] 63: DName$(Zaehler) = DName$(Zaehler) + C
    HR$(PEEK(Offset + q))
[K21] 64: NEXT q
[ZRØ] 65: ELSE
[NS9] 66: DName$(Zaehler) = "<Blk: " + HexAnpassun
    g$(PEEK(Offset + 1Ø), 2) + ">"
[8Ø1] 67: END IF
[KK2] 68: ReadNextDriver
[HSØ] 69: LOOP
[ØZ2] 70: IF Anzahl >= 2Ø THEN
[Wx1] 71: Schluss = 2Ø
[ZRØ] 72: ELSE
[TR2] 73: Schluss = Anzahl
[8Ø1] 74: END IF
[3G4] 75: CALL ZeigeListe(1, Schluss)
[TC2] 76: AktuellInvers
[RL1] 77: Struktur
[UM1] 78: LOCATE 25, 1
[DZ6] 79: m$ = "Wählen Sie mit den Pfeiltasten einen"
[7G5] 80: m$ = m$ + "Gerätetreiber!" + SPACES(3)
[YT4] 81: m$ = m$ + "-" + SPACES(4) + "Abbruch: "
[UA3] 82: m$ = m$ + "Escape-Taste"
[LX2] 83: PRINT Invers$(m$);
[UB1] 84: Eingabe:
[1C2] 85: Taste$ = INKEY$
[KV4] 86: IF Taste$ = "" THEN GOTO Eingabe
[ZY2] 87: SELECT CASE Taste$
[833] 88: CASE CHR$(Ø) + CHR$(72)
[EK1] 89: PfeilHoch
[433] 90: CASE CHR$(Ø) + CHR$(8Ø)

```

```

[Y52] 91: PfeilRunter
[6V1] 92: CASE CHR$(27)
[PNØ] 93: CLS
[Ø91] 94: DEF SEG
[4MØ] 95: END
[PF1] 96: CASE ELSE
[GS1] 97: END SELECT
[T12] 98: GOTO Eingabe
[ØØØ] 99:
[633] 100: SUB AktuellInvers
[9M1] 101: COLOR Ø, 15
[HF3] 102: LOCATE 3 + BalkenPos, 2
[Z14] 103: PRINT USING "##"; Aktuell;
[S75] 104: PRINT CHR$(179); Header$(Aktuell);
[815] 105: PRINT CHR$(179); DName$(Aktuell);
[VH5] 106: PRINT CHR$(179); Attribut$(Aktuell);
[FU5] 107: PRINT CHR$(179); Strategie$(Aktuell);
[4K5] 108: PRINT CHR$(179); IntEntry$(Aktuell)
[7L1] 109: COLOR 15, Ø
[HAL] 110: END SUB
[ØØØ] 111:
[233] 112: SUB AktuellNormal
[HF3] 113: LOCATE 3 + BalkenPos, 2
[Z14] 114: PRINT USING "##"; Aktuell;
[S75] 115: PRINT CHR$(179); Header$(Aktuell);
[815] 116: PRINT CHR$(179); DName$(Aktuell);
[VH5] 117: PRINT CHR$(179); Attribut$(Aktuell);
[FU5] 118: PRINT CHR$(179); Strategie$(Aktuell);
[4K5] 119: PRINT CHR$(179); IntEntry$(Aktuell)
[HAL] 120: END SUB
[ØØØ] 121:
[YØ3] 122: SUB BelegungBlock
[UR1] 123: LOCATE 11, 43
[AK3] 124: PRINT CHR$(179); SPC(23);
[WR1] 125: LOCATE 12, 43
[GZ5] 126: PRINT "1 32-Bit-Sektor-Adressierung?"
[YR1] 127: LOCATE 13, 43
[ZH3] 128: PRINT CHR$(179); SPC(11);
[ØS1] 129: LOCATE 14, 43
[ZH3] 130: PRINT CHR$(179); SPC(11);
[2S1] 131: LOCATE 15, 43
[2K3] 132: PRINT CHR$(179); SPC(21);
[4S1] 133: LOCATE 16, 43
[282] 134: PRINT CHR$(179)
[6S1] 135: LOCATE 17, 43
[9Z4] 136: PRINT "6 Unterstützt IOCTL?"
[8S1] 137: LOCATE 18, 43
[282] 138: PRINT CHR$(179)
[AS1] 139: LOCATE 19, 43
[9C4] 140: PRINT "B Versteht "; CHR$(34);
[H86] 141: PRINT "Öffnen/Schließen"; CHR$(34); "?"
[TR1] 142: LOCATE 20, 43
[282] 143: PRINT CHR$(179)
[YR1] 144: LOCATE 21, 43
[H96] 145: PRINT "D Benutzt FAT-ID-Byte?"; SPC(5);
[XR1] 146: LOCATE 22, 43
[HK5] 147: PRINT "E Kontroll-Zeichenketten?"
[ZR1] 148: LOCATE 23, 43
[Ux5] 149: PRINT "F Zeichen- oder Blocktreiber?"
[3S1] 150: LOCATE 11, 76
[NØ2] 151: PRINT SPC(3);
[DF3] 152: FOR Zaehler = 13 TO 15
[ØU2] 153: LOCATE Zaehler, 76
[NØ2] 154: PRINT SPC(3);
[G42] 155: NEXT Zaehler
[VP3] 156: CALL JaNein(12, 76, 2 ^ 1)
[MR3] 157: CALL JaNein(17, 76, 2 ^ 6)
[CX3] 158: CALL JaNein(19, 76, 2 ^ 11)
[WW3] 159: CALL JaNein(21, 76, 2 ^ 13)
[2X3] 160: CALL JaNein(22, 76, 2 ^ 14)
[8S1] 161: LOCATE 23, 76
[PS1] 162: PRINT "[B]"
[HAL] 163: END SUB
[ØØØ] 164:
[2H3] 165: SUB BelegungZeichen
[UR1] 166: LOCATE 11, 43
[EB5] 167: PRINT "Ø Standard-Eingabegerät?"
[WR1] 168: LOCATE 12, 43
[SK6] 169: PRINT "1 Standard-Ausgabegerät?"; SPC(5);
[YR1] 170: LOCATE 13, 43
[ZA3] 171: PRINT "2 NUL-Gerät?"
[ØS1] 172: LOCATE 14, 43
[7A3] 173: PRINT "3 Uhr-Gerät?"
[2S1] 174: LOCATE 15, 43
[G15] 175: PRINT "4 Unterstützt INT 29h?"
[4S1] 176: LOCATE 16, 43
[282] 177: PRINT CHR$(179)
[6S1] 178: LOCATE 17, 43
[9Z4] 179: PRINT "6 Unterstützt IOCTL?"
[8S1] 180: LOCATE 18, 43
[282] 181: PRINT CHR$(179)
[AS1] 182: LOCATE 19, 43
[9C4] 183: PRINT "B Versteht "; CHR$(34);
[H86] 184: PRINT "Öffnen/Schließen"; CHR$(34); "?"
[TR1] 185: LOCATE 20, 43

```

Listing 2. Das Utility »Dev-Test« erleichtert die Analyse der ak-tiven Gerätetreiber.


```
[282] 186: PRINT CHR$(179)
[VR1] 187: LOCATE 21, 43
[7C5] 188: PRINT "D Unterstützt "; CHR$(34);
[LU4] 189: PRINT "Until busy"; CHR$(34); "?"
[XR1] 190: LOCATE 22, 43
[HK5] 191: PRINT "E Kontroll-Zeichenketten?"
[ZR1] 192: LOCATE 23, 43
[UX5] 193: PRINT "F Zeichen- oder Blocktreiber?"
[PP3] 194: CALL JaNein(11, 76, 2 ^ 0)
[VP3] 195: CALL JaNein(12, 76, 2 ^ 1)
[ØR3] 196: CALL JaNein(13, 76, 2 ^ 2)
[5R3] 197: CALL JaNein(14, 76, 2 ^ 3)
[AR3] 198: CALL JaNein(15, 76, 2 ^ 4)
[MR3] 199: CALL JaNein(17, 76, 2 ^ 6)
[CK3] 200: CALL JaNein(19, 76, 2 ^ 11)
[WW3] 201: CALL JaNein(21, 76, 2 ^ 13)
[2X3] 202: CALL JaNein(22, 76, 2 ^ 14)
[8S1] 203: LOCATE 23, 76
[6U1] 204: PRINT "[Z]"
[HA1] 205: END SUB
[ØØØ] 206:
[V96] 207: FUNCTION Binaer$(hv AS LONG, Stellen)
[7RØ] 208: h$ = ""
[CM4] 209: FOR Zaehler = Ø TO Stellen - 1
[6Ø4] 210: IF hv AND 2 ^ Zaehler THEN
[P81] 211: h$ = "1" + h$
[ZRØ] 212: ELSE
[K81] 213: h$ = "Ø" + h$
[8Ø1] 214: END IF
[G42] 215: NEXT Zaehler
[6T1] 216: Binaer$ = h$
[R52] 217: END FUNCTION
[ØØØ] 218:
[AA7] 219: FUNCTION HexAnpassung$(hv AS LONG, Stellen)
[5P1] 220: h$ = HEX$(hv)
[MR3] 221: IF LEN(h$) < Stellen THEN
[6L9] 222: HexAnpassung$ = LEFT$(STRING$(Stellen, "Ø"),
Stellen-LEN(h$))+h$
[ZRØ] 223: ELSE
[CK5] 224: HexAnpassung$ = RIGHT$(h$, Stellen)
[8Ø1] 225: END IF
[R52] 226: END FUNCTION
[ØØØ] 227:
[1E1] 228: SUB Init
[122] 229: BalkenPos = 1
[4V1] 230: Aktuell = 1
[HA1] 231: END SUB
[ØØØ] 232:
[NF4] 233: FUNCTION Invers$(Zkette$)
[9M1] 234: COLOR Ø, 15
[RE2] 235: PRINT Zkette$;
[7L1] 236: COLOR 15, Ø
[R52] 237: END FUNCTION
[ØØØ] 238:
[4P3] 239: SUB JaNein(y, x, Maske)
[UT1] 240: LOCATE y, x
[BB7] 241: IF VAL("&H" + Attribut$(Aktuell)) AND Maske THEN
[6T1] 242: PRINT "[J]"
[ZRØ] 243: ELSE
[ET1] 244: PRINT "[N]"
[8Ø1] 245: END IF
[HA1] 246: END SUB
[ØØØ] 247:
[DK1] 248: SUB Menue
[PNØ] 249: CLS
[ØY6] 250: m$ = "DEV-Test/Gerätetreiber-Analyse aus"
[E56] 251: m$ = m$ + " der »DOS international«/"
[WY4] 252: m$ = m$ + "Autor: Markus Zietlow"
[2S2] 253: PRINT Invers$(m$)
[DF4] 254: PRINT CHR$(179); "Nr"; CHR$(179);
[4C5] 255: PRINT "Header "; CHR$(179); "Name ";
[BX4] 256: PRINT CHR$(179); "Attr"; CHR$(179);
[RL4] 257: PRINT "Strt"; CHR$(179); "Intr";
[282] 258: PRINT CHR$(179)
[MN1] 259: LOCATE 2, 41
[YR3] 260: PRINT "Treiber-Anzahl:"
[6S4] 261: PRINT CHR$(195); STRING$(2, 196);
[1T4] 262: PRINT CHR$(197); STRING$(9, 196);
[YS4] 263: PRINT CHR$(197); STRING$(8, 196);
[KS4] 264: PRINT CHR$(197); STRING$(4, 196);
[KS4] 265: PRINT CHR$(197); STRING$(4, 196);
[KS4] 266: PRINT CHR$(197); STRING$(4, 196);
[492] 267: PRINT CHR$(180);
[1B5] 268: PRINT " "; CHR$(218); STRING$(38, 196);
[E72] 269: PRINT CHR$(191)
[873] 270: FOR Zaehler = 1 TO 2Ø
[MD5] 271: PRINT CHR$(179); SPACE$(2); CHR$(179);
[6F5] 272: PRINT SPACE$(9); CHR$(179); SPACE$(8);
[SD5] 273: PRINT CHR$(179); SPACE$(4); CHR$(179);
[LE5] 274: PRINT SPACE$(4); CHR$(179); SPACE$(4);
[SV2] 275: PRINT CHR$(179); " ";
[M85] 276: PRINT CHR$(179); SPACE$(38); CHR$(179)
[G42] 277: NEXT Zaehler
[XR4] 278: PRINT CHR$(192); STRING$(2, 196);
[NS4] 279: PRINT CHR$(193); STRING$(9, 196);
[KS4] 280: PRINT CHR$(193); STRING$(8, 196);
[6S4] 281: PRINT CHR$(193); STRING$(4, 196);
```

```
[6S4] 282: PRINT CHR$(193); STRING$(4, 196);
[6S4] 283: PRINT CHR$(193); STRING$(4, 196);
[YX3] 284: PRINT CHR$(217); SPACE$(2);
[8X4] 285: PRINT CHR$(192); STRING$(38, 196);
[H72] 286: PRINT CHR$(217)
[TN1] 287: LOCATE 4, 5Ø
[BF5] 288: PRINT "Gerätetreiber-Struktur"
[ZN1] 289: LOCATE 6, 6Ø
[DD3] 290: PRINT "FEDCBA987654321Ø"
[XN1] 291: LOCATE 7, 45
[AR3] 292: PRINT "Attribut-Bits:"
[XN1] 293: LOCATE 9, 43
[D15] 294: PRINT "Bit"; SPC(1Ø); "Bedeutung";
[Ø53] 295: PRINT SPC(1Ø); "Wert"
[SR1] 296: LOCATE 1Ø, 43
[343] 297: PRINT STRING$(36, 61)
[HA1] 298: END SUB
[ØØØ] 299:
[V82] 300: SUB PfeilHoch
[8Y2] 301: IF Aktuell > 1 THEN
[NC2] 302: AktuellNormal
[Ø83] 303: IF BalkenPos > 1 THEN
[493] 304: Aktuell = Aktuell - 1
[Ø23] 305: BalkenPos = BalkenPos - 1
[ZRØ] 306: ELSE
[493] 307: Aktuell = Aktuell - 1
[D66] 308: CALL ZeigeListe(Aktuell, Aktuell + 19)
[8Ø1] 309: END IF
[TC2] 310: AktuellInvers
[RL1] 311: Struktur
[ZRØ] 312: ELSE
[RNØ] 313: BEEP
[8Ø1] 314: END IF
[HA1] 315: END SUB
[ØØØ] 316:
[BW2] 317: SUB PfeilRunter
[9X3] 318: IF Aktuell < Anzahl THEN
[NC2] 319: AktuellNormal
[AC3] 320: IF BalkenPos < 2Ø THEN
[W83] 321: Aktuell = Aktuell + 1
[SY3] 322: BalkenPos = BalkenPos + 1
[ZRØ] 323: ELSE
[W83] 324: Aktuell = Aktuell + 1
[266] 325: CALL ZeigeListe(Aktuell - 19, Aktuell)
[8Ø1] 326: END IF
[TC2] 327: AktuellInvers
[RL1] 328: Struktur
[ZRØ] 329: ELSE
[RNØ] 330: BEEP
[8Ø1] 331: END IF
[HA1] 332: END SUB
[ØØØ] 333:
[KL3] 334: SUB ReadListOfLists
[7P2] 335: InRegs.ax = &H52ØØ
[T96] 336: CALL INTERRUPTX(&H21, InRegs, OutRegs)
[AU4] 337: OutRegs.bx = OutRegs.bx + &H22
[XD3] 338: Segment = OutRegs.es
[RA3] 339: Offset = OutRegs.bx
[HA1] 340: END SUB
[ØØØ] 341:
[Z83] 342: SUB ReadNextDriver
[XD7] 343: Segment = PEEK(Offset + 2) + PEEK(Offset + 3) * 256&
[7X6] 344: Offset = PEEK(Offset) + PEEK(Offset + 1) * 256&
[HA1] 345: END SUB
[ØØØ] 346:
[4A2] 347: SUB Struktur
[ØP1] 348: LOCATE 7, 6Ø
[E67] 349: PRINT Binaer$(VAL("&H" + Attribut$(Aktuell)), 16)
[L47] 350: IF VAL("&H" + Attribut$(Aktuell)) AND 2 ^ 15 THEN
[NU2] 351: BelegungZeichen
[ZRØ] 352: ELSE
[HA2] 353: BelegungBlock
[8Ø1] 354: END IF
[HA1] 355: END SUB
[ØØØ] 356:
[K65] 357: SUB ZeigeListe(Anfang, Grenze)
[Y35] 358: FOR Zaehler = Anfang TO Grenze
[6T4] 359: LOCATE 3 + Zaehler - Anfang + 1, 2
[N34] 360: PRINT USING "##"; Zaehler;
[FT4] 361: LOCATE 3 + Zaehler - Anfang + 1, 5
[MV3] 362: PRINT Header$(Zaehler);
[RZ4] 363: LOCATE 3 + Zaehler - Anfang + 1, 15
[LM3] 364: PRINT DName$(Zaehler);
[P24] 365: LOCATE 3 + Zaehler - Anfang + 1, 24
[B74] 366: PRINT Attribut$(Zaehler);
[AØ5] 367: LOCATE 3 + Zaehler - Anfang + 1, 29
[HE4] 368: PRINT Strategie$(Zaehler);
[TT4] 369: LOCATE 3 + Zaehler - Anfang + 1, 34
[R44] 370: PRINT IntEntry$(Zaehler)
[G42] 371: NEXT Zaehler
[HA1] 372: END SUB
```

Listing 2. Das Utility »Dev-Test« erleichtert die Analyse der aktiven Gerätetreiber.

Werkstatt: QBasic

Interrupts unter QBasic

Im Gegensatz zu Quick Basic und Basic PDS sind im Befehlssatz von QBasic keine Befehle zum Aufruf von Interrupts enthalten. Mit Hilfe einer kleinen Routine in Assembler und Basic rufen Sie die Interrupts des PCs direkt auf und genießen alle Vorzüge des Betriebssystems.

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 Assembler-Programmen und einigen Hochsprachen direkt aufrufen; dem QBasic-Programmierer bleibt ihr Einsatz verschlossen.

Fast alle BIOS-Routinen sind Interrupt-Routinen oder Funktionen von Interrupt-Routinen – Sie rufen sie über Unterbrechungsanforderungen der Peripherie oder über die sogenannten Software-Interrupts 00hex bis FFhex auf. 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.

Darüber hinaus hat das Konzept der Software-Interrupts wesentliche Vorteile. Wenn die Interrupt-Routinen mit den Routinen des Betriebssystems belegt sind, lassen sich die Betriebssystemfunktionen damit über einfache Interrupt-Befehle aufrufen.

Nur existieren unter QBasic keine Interrupt-Befehle. Fenster-Rollen, die Abfrage der Systemkonfiguration oder Manipulation am Betriebssystem sind so gut wie ausgeschlossen. Mit dem kleinen Maschinenspracheprogramm »interrup« sind Sie jedoch in der Lage, die Interrupt-Routinen und damit die Betriebssystemroutinen des PC zu nutzen. Diese neugewonnene Fähigkeit von QBasic eröffnet Ihnen Anwendungsbereiche, die weit über das übliche Einsatzgebiet des Basic-Interpreters hinausgehen.

Listing 1 zeigt »interrup« in der Assembler-Fassung. Sie müssen jedoch keinen Assembler besitzen, um den Interrupt-Aufruf zu benutzen. In Listing 2 ist »interrup« in ein Basic-Programm eingebunden. In diesem Fall liegt die Hilfsroutine in den Data-Zeilen als Maschinencode vor. QBasic schreibt diesen Maschinencode beim Programmstart lediglich in den Ar-

beitsspeicher des PC und hält die Routine dort für den Aufruf durch den Call-Absolute-Befehl bereit. Sie müssen also die Basic-Zeilen von »interrup« lediglich in das Programm einbinden, aus dem Sie die Interrupt-Routinen des PC aufrufen möchten.

Beim Programmieren der Routine hat sich ein Nachteil von QBasic herausgestellt. Um die Data-Zeilen vom Hauptprogramm fernzuhalten, wäre es wünschenswert, die Initialisierungsroutine in einem Unterprogramm aufzurufen. QBasic erlaubt aber die Verwendung von Data-Zeilen lediglich in der Globalebene eines Moduls.

Das Demonstrationsprogramm in Listing 3 bedient sich der Hilfsroutine, um den Systemstatus zu ermitteln.

Als nächstes belegen Sie die Übergabeparameter mit den relevanten Registern. Die Werte der nicht belegten Register werden von QBasic automatisch eingesetzt. Als Beispiel dient die Funktion »FScroll« aus Listing 3, die einen Bildschirmbereich verschiebt (»rollt«). Der Video-Interrupt 10hex (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 gerollt werden sollen, CH und CL enthalten die Koordinaten der linken oberen Ecke, DH und DL die der rechten unteren Ecke. Schließlich geben Sie in BH noch das Farbattribut für die gelöschten Zeilen an.

Folgende Werte rollen das Fenster innerhalb der Koordinaten 2,2 und 20,20 um fünf Zeilen aufwärts und weisen den dadurch freiwerdenden Zeilen das Attribut 0 zu:

```
AH=6, AL=5, CH=2, CL=2,
DH=20, DL=20, BH=0
```

In der Variablen »Interrupt« legen Sie die Nummer des aufzurufenden Interrupts ab. Die Hilfsroutine bedient sich eines Tricks, um die Interrupt-Nummer an das Maschinenspracheprogramm zu übergeben. Sie läßt die Interrupt-Nummer bei der Initialisierungsroutine offen und speichert den

entsprechenden Wert vor dem Aufruf an der zugehörigen Stelle in »Asm« durch einen Poke-Befehl.

Haben Sie das Hauptprogramm mit den aufgeführten Variablenzuweisungen versorgt, wird die Betriebssystemroutine durch

```
CALL CallInt(Interrupt, AH, AL, BH, BL, CH, CL, DH, DL)
```

aufgerufen. Vergessen Sie danach nicht, Ihr Programm ordnungsgemäß mit

```
END
```

abzuschließen.

Der Programmiersprache QBasic fehlen einige moderne Leistungsmerkmale wie Menüs oder Fenstertechnik. Die Basic-Erweiterung beziehungsweise das Beispielprogramm aus Listing 3 stellt Ihnen Routinen für dieses Aufgabengebiet zur Verfügung. Mit diesen Prozeduren sind Sie in der Lage, eine eigene Benutzeroberfläche zu entwickeln.

Ein Beispiel soll Ihnen die Aufgabe der Prozeduren verdeutlichen. Ein bemerkenswerte Fähigkeit von vielen Anwendungsprogrammen ist die kontextsensitive Hilfefunktion. Sie ist vor allem für Einsteiger interessant, stellt aber auch für Profis ein nützliches Arbeitsmittel dar.

Sie gibt in jeder Situation passende und umfangreiche Informationen zu einem gewünschten Thema. Wenn Sie beispielsweise nicht mehr wissen, wie ein Kommando aufgebaut ist und welche Parameter es benötigt, geben Sie einfach das entsprechende Befehlswort ein und rufen die Hilfefunktion auf. Das Programm pickt die entsprechenden Informationen aus seiner Hilfebibliothek heraus und zeigt sie übersichtlich am Bildschirm an. Brauchen Sie hingegen eine generelle Hilfestellung, ist es kein Problem, ein Menü aufzurufen, das Ihnen einen Überblick über die angebotenen Themen bietet. Aus den Überschriften suchen Sie sich dann die betreffenden Fakten zusammen.

Die Fenster zur QBasic-Welt

Um den kontextsensitiven Informationsbereich auf dem Bildschirm darzustellen, geben die Anwendungsprogramme in den meisten Fällen ein Fenster aus, das die gewünschten Angaben enthält. Nach der Rückkehr in den Editor muß der vorherige Inhalt des Bildschirms wiederhergestellt werden. Ähnlich funktioniert die Arbeit mit einem Menü. Nachdem es auf dem Bildschirm erschienen ist und der Benutzer einen Menüpunkt ausgewählt hat, verschwindet es wieder.

Die Routinen von Listing 3 stellen Ihnen Unterprogramme zur Verfügung, die es Ihnen erlauben, eine ähnliche

Menütechnik in eigenen Programmen zu verwenden. Die Prozeduren sind ausreichend dokumentiert und verwenden aussagefähige Variablen.

Wenn Sie das Programm starten, initialisiert die Routine die benötigten Variablen und richtet einen Zwischenspeicher namens »FFeld\$« ein. In »FFeld\$« legt das Programm den Inhalt der einzelnen Fenster ab, damit diese nach dem Schließen eines anderen Fensters wieder restauriert werden können. Die Konstante »FGrenze« bestimmt die maximale Größe des Fensterstapels, die im Beispiel auf 3000 ASCII-Zeichen begrenzt ist. Reicht dieser Platz nicht aus, dann ändern Sie einfach den Grenzwert im Definitionsteil des Listings.

Die wichtigste Routine ist die Prozedur »FOeffnen«, die ein neues Fenster öffnet und den Inhalt des darunterliegenden Bildschirmbereichs sichert. Als Parameter übergeben Sie die linke, rechte, obere und untere Grenze des Fensters in den entsprechenden Variablen. Das letzte Argument ist die Titelzeile des Fensters in der Variablen »text\$«. Wenn die Länge des Titels die Breite des Fensters übersteigt, dann kürzt die Prozedur den Namen automatisch ab. Andernfalls stellt sie ihn zentriert dar. Bevor ein Fenster geöffnet wird, stellt es durch den Aufruf von »LiesKonfig« in der Variablen »RGrenze« die Anzahl der Zeichen pro Zeile fest.

Das Unterprogramm »FSchliessen«, das ein Fenster schließt, stellt den vorherigen Bildschirminhalt wieder her. Wenn nach dem Schließen des Fen-

sters noch ein weiteres geöffnet ist, dann wird der Cursor auf die Position gesetzt, die zuletzt in dem Fenster festgelegt wurde.

Mit der Routine »FCursor« bestimmen Sie die Position des Cursors innerhalb eines Fensters. Wie bei dem Locate-Befehl in QBasic übergeben Sie die x- und die y-Koordinaten der Eingabemarke in den zugehörigen Variablen. Die Angabe bezieht sich relativ auf die linke obere Ecke des Fensters, die die Koordinaten 1,1 erhält. Sie können die aktuellen Koordinaten des Cursors abfragen, die in den Variablen »F.x« und »F.y« abgelegt sind.

Die Prozedur »FLoeschen«, die den Inhalt des aktuellen Fensters löscht, benötigt keine Argumente. Sie setzt zusätzlich den Cursor auf die linke, obere Ecke des Fensters. Das sind die relativen Koordinaten 1,1.

Die Ausgaberroutine »FText« gibt den in der Variablen »Text\$« übergebenen Text an der Cursorposition innerhalb eines Fensters aus. Sie erhöht die x- und y-Koordinaten um den der Länge des Textes entsprechenden Wert. Erreicht der Cursor die letzte Fensterzeile, dann rollt das Fenster mit dem dafür vorgesehenen Unterprogramm »FScroll« um eine Zeile aufwärts und setzt die Ausgabe fort.

Sie können die Scroll-Routine auch gesondert aufrufen. Zu diesem Zweck legen Sie in der Variablen »Farbe%« das Farbattribut ab, das die Prozedur in die letzte Zeile des verschobenen Bereichs schreibt. Die Programmzeilen beziehen sich immer auf das aktuelle Fenster.

Als letztes Unterprogramm stellt die Prozedursammlung für Menüs die Routine »MenueZeigen« bereit. Sie benötigt keine Parameter und erlaubt Ihnen, das aktuelle Fenster als Menü zu verwenden. Sie wählen mit den Cursortasten den gewünschten Menüpunkt aus und bestätigen ihn durch Drücken der Enter-Taste. Nach dem Aufruf der Routine wird die Nummer der ausgewählten Menüzeile in der Variablen »M.Punkt« abgelegt. Falls Sie [Esc] gedrückt haben, erhält die Variable den Wert 0.

Wenn Sie die Reaktion auf falsche Parameterübergaben verändern wollen, dann müssen Sie das Unterprogramm, das die Fehler behandelt, entsprechend ändern. Es ist beispielsweise denkbar, einen Programmteil zu konzipieren, der ähnlich wie das QBasic-Kommando »on error goto« jede Fehlermeldung abfängt.

Die Bildschirmroutinen des Fensterpakets unterstützen alle wichtigen Grafikkarten, da sie im Textmodus arbeiten. Den verschiedenen Bildschirmbreiten wurde mit der Entwicklung der Fenster- und Menürountinen Rechnung getragen. Wenn Sie beispielsweise mit dem Screen-Kommando 40 Zeichen pro Zeile einstellen, berücksichtigen die Prozeduren die geänderten Werte und passen die Unterprogramme an die neuen Gegebenheiten an.

Mit dem Unterprogrammpaket steht Ihnen ein leistungsfähiges Hilfsmittel zur Verfügung, das die Programmierung bedienungsfreundlicher Anwendungen erleichtert.

(Markus Zietlow/uk)

```
[KY3] 1: ;Programm: qbasic11.asm
[3G6] 2: ;Funktion: Assemblerprogramm für den
[V36] 3: ; Interrupt-Aufruf unter QBasic
[Y04] 4: ;Sprache: MASM oder TASM
[D24] 5: ;Autor: Markus Zietlow
[9X4] 6: ;(c) 1992 DMV Widuch GmbH & Co. KG
[000] 7:
[063] 8: ; Aufruf in QBasic:
[4F6] 9: ; CALL INTERRUPT(Interrupt,AH,AL,BH,BL,
[W34] 10: ; CH,CL,DH,DL,Asm())
[P30] 11: ;
[3E5] 12: cseg segment para public 'code'
[GR2] 13: public intrprt
[ZW2] 14: intrprt proc far
[YT3] 15: ;-Segmentzuweisungen
[954] 16: assume cs:cseg,ds:nothing
[PV4] 17: assume ss:nothing,es:nothing
[Y95] 18: ;-Basiszeiger sichern und auf die
[K35] 19: ; Übergabeparameter richten
[TB1] 20: push bp
[7S1] 21: mov bp,sp
[E25] 22: ;-DL vom Prozedurstapel holen
[ER2] 23: mov bx,[bp+06]
[052] 24: mov dl,[bx]
[A25] 25: ;-DH vom Prozedurstapel holen
[MR2] 26: mov bx,[bp+08]
[G42] 27: mov dh,[bx]
[A25] 28: ;-CL vom Prozedurstapel holen
[YP2] 29: mov bx,[bp+10]
[X42] 30: mov cl,[bx]
[625] 31: ;-CH vom Prozedurstapel holen
[4R2] 32: mov bx,[bp+12]
[D42] 33: mov ch,[bx]
[625] 34: ;-BL vom Prozedurstapel holen
[AR2] 35: mov bx,[bp+14]
[U42] 36: mov bl,[bx]
```

```
[225] 37: ;-BH vom Prozedurstapel holen
[HW6] 38: ;-Damit BL dabei nicht überschrieben wird,
[WY5] 39: ; wird BL in AL zwischengespeichert
[6P1] 40: mov al,bl
[GR2] 41: mov bx,[bp+16]
[A42] 42: mov bh,[bx]
[7P1] 43: mov bl,al
[225] 44: ;-AL vom Prozedurstapel holen
[PR2] 45: mov bx,[bp+18]
[R42] 46: mov al,[bx]
[Y15] 47: ;-AH vom Prozedurstapel holen
[0R2] 48: mov bx,[bp+20]
[742] 49: mov ah,[bx]
[P93] 50: ;-BP und SP sichern
[TB1] 51: push bp
[CD1] 52: push sp
[LK3] 53: ;-Interrupt aufrufen
[NG6] 54: ;-Die Interruptnummer wird später vom
[D66] 55: ; QBasic-Programm manuell eingesetzt
[W61] 56: int 0h
[ZV4] 57: ;-BP und SP wiederherstellen
[6B1] 58: pop sp
[L91] 59: pop bp
[Y74] 60: ;-Rückgabewert BX sichern
[TC1] 61: push bx
[0S4] 62: ;-Rückgabewert DL speichern
[ER2] 63: mov bx,[bp+06]
[A22] 64: mov [bx],dl
[SR4] 65: ;-Rückgabewert DH speichern
[MR2] 66: mov bx,[bp+08]
[622] 67: mov [bx],dh
[ZR4] 68: ;-Rückgabewert CL speichern
[YP2] 69: mov bx,[bp+10]
[622] 70: mov [bx],cl
[RR4] 71: ;-Rückgabewert CH speichern
```



```

[4R2] 72:      mov  bx,[bp+12]
[222] 73:      mov  [bx],ch
[307] 74:      ;--Rückgabewerte BL und BH speichern, indem
[SM6] 75:      ; der alte Wert von BX wiederhergestellt
[286] 76:      ; und in CX zwischengespeichert wird
[LA1] 77:      pop   bx
[ØT1] 78:      mov  cx,bx
[AR2] 79:      mov  bx,[bp+14]
[622] 80:      mov  [bx],cl
[GR2] 81:      mov  bx,[bp+16]
[222] 82:      mov  [bx],ch
[XR4] 83:      ;--Rückgabewert AL speichern
[PR2] 84:      mov  bx,[bp+18]
[Y12] 85:      mov  [bx],al
[NR4] 86:      ;--Rückgabewert AH speichern
[ØR2] 87:      mov  bx,[bp+20]
[U12] 88:      mov  [bx],ah
[T75] 89:      ;--Basiszeiger wiederherstellen
[L91] 90:      pop   bp
[6S6] 91:      ;--Zurück ins Basic-Programm, indem alle
[T67] 92:      ; Übergabeparameter vom Stapel entfernt
[9V6] 93:      ; werden und ein Far-Ret ausgeführt wird
[E41] 94:      ret   16
[P72] 95:      intrprt endp
[HR1] 96:      cseg  ends
[W53] 97:      ;--Ende des Programms
[4MØ] 98:      end

```

Listing 1. Das Programm »qbasic1.asm« zeigt Ihnen den Aufbau der Interrupt-Routinen.

```

[KW3] 1: 'Programm: qbasic12.bas
[427] 2: 'Funktion: Interrupt-Aufrufe unter QBasic
[GM2] 3: 'Sprache: QBasic
[514] 4: 'Autor: Markus Zietlow
[LW4] 5: '(c) 1992 DMV Widuch GmbH & Co. KG
[ØØØ] 6:
[AF6] 7: 'Ein Interrupt wird mit der Anweisung
[WY6] 8: 'CallInt(Interrupt,AH,AL,BH,BL,CH,CL,DH,DL)
[GV6] 9: 'aufgerufen. Dabei enthalten AH bis DL die
[CC7] 10: 'Werte der Register und Interrupt die
[ZG6] 11: 'Nummer des aufzurufenden Interrupts
[E2Ø] 12: '
[LW2] 13: 'Initialisierung
[HH7] 14: 'Das Feld «Asm()» wird als «common shared»
[X46] 15: 'deklariert, damit es bei Aufrufen von
[DX7] 16: 'Prozeduren nicht mit übergeben werden muß
[E2Ø] 17: '
[TL1] 18: DEFINT A-Z
[413] 19: COMMON SHARED Asm()
[LM3] 20: RESTORE IntProgramm
[936] 21: 'Speichern des Assemblerprogramms in
[WP3] 22: 'dem Integer-Feld 'Asm()'
[E2Ø] 23: '
[Y53] 24: DIM Asm(1 TO (102 / 2))
[WT1] 25: Zaehler = Ø
[6X3] 26: AsmStart = VARPTR(Asm(1))
[7S3] 27: DEF SEG = VARSEG(Asm(1))
[ED3] 28: DO WHILE Code$ <> "ENDE"
[UH1] 29: READ Code$
[R96] 30: POKE AsmStart + Zaehler, VAL("&h" + Code$)
[E93] 31: Zaehler = Zaehler + 1
[HSØ] 32: LOOP
[Ø91] 33: DEF SEG
[AH2] 34: 'Hauptprogramm
[4MØ] 35: END
[CS6] 36: 'Daten für das Maschinenspracheprogramm
[K72] 37: IntProgramm:
[PC5] 38: DATA 55,8b,ec,8b,5e,Ø6,8a,17,8b,5e,Ø8,8a
[TD5] 39: DATA 37,8b,5e,Øa,8a,Øf,8b,5e,Øc,8a,2f,8b
[SC5] 40: DATA 5e,Øe,8a,1f,8a,c3,8b,5e,1Ø,8a,3f,8a
[8A5] 41: DATA d8,8b,5e,12,8a,Ø7,8b,5e,14,8a,27,55
[W85] 42: DATA 54,cd,ØØ,5c,5d,53,8b,5e,Ø6,88,17,8b
[5D5] 43: DATA 5e,Ø8,88,37,8b,5e,Øa,88,Øf,8b,5e,Øc
[CF5] 44: DATA 88,2f,8b,5e,Øe,5b,8b,cb,88,Øf,8b,5e
[V85] 45: DATA 1Ø,88,2f,8b,5e,12,88,Ø7,8b,5e,14,88
[UB3] 46: DATA 27,5d,ca,1Ø,ØØ,ENDE
[ØØØ] 47:
[NR6] 48: 'Unterprogramm für den Interruptaufruf
[ØØØ] 49:
[6L7] 50: SUB CallInt(Interrupt,AH,AL,BH,BL,CH,CL,DH,
DL)
[6X3] 51: AsmStart = VARPTR(Asm(1))
[7S3] 52: DEF SEG = VARSEG(Asm(1))
[7T4] 53: POKE AsmStart + 5Ø, Interrupt
[U98] 54: CALL ABSOLUTE(AH,AL,BH,BL,CH,CL,DH,DL,VAR
PTR(Asm(1)))
[Ø91] 55: DEF SEG
[HA1] 56: END SUB

```

Listing 2. Das Hauptprogramm »qbasic12.bas« bildet die Grundlage für weitere Anwendungen.

DOS INTERNATIONAL

Tips & Tricks gesucht!

Die DOS International ist eine Zeitschrift zum Mitmachen. Vor allem die Tips & Tricks-Rubriken leben von Ihren Beiträgen, Ihren Ideen und Programmen. Sie müssen kein Schriftsteller sein, damit ein Beitrag aus Ihrer Feder (beziehungsweise Ihrer Tastatur) veröffentlicht wird. Einschicken können Sie alles, was irgendwie mit der Programmierung und Bedienung des PC zusammenhängt. Das sind zum Beispiel wichtige Hinweise zu Anwendungs-Software, trickreiche Stapeldateien, interessante Basic-, Assembler-, C-, oder Pascal-Routinen, aber auch Tips zur PC-Hardware. Für jeden veröffentlichten Trick gibt es – je nach Länge und Qualität – ein angemessenes Honorar. Bitte beachten Sie beim Vorbereiten einer Einsendung die folgenden Hinweise, damit der Beitrag in der Redaktion schnell und unproblematisch bearbeitet werden kann:

- Legen Sie eine Diskette bei mit allen zum Trick gehörigen Dateien (Quelltext, ausführbares Programm, Beschreibung).
- Das Anschreiben sollte in kurzen Stichpunkten Ihre Idee beschreiben.
- Diskette und Anschreiben versehen Sie bitte mit Ihrer genauen Adresse und mit dem Programmnamen.
- Sie ersparen uns viel Arbeit, wenn Sie Quelltexte auf 43 Zeichen pro Zeile formatieren. Steuerzeichen sollten nicht enthalten sein.
- Die Programmbeschreibung sollte als unformatierter ASCII-Text vorliegen.

Spätestens zwei Wochen, nachdem Ihre Einsendung bei uns eingegangen ist, erhalten Sie schriftlich Bescheid.

Unsere Adresse:



Redaktion DOS International
Stichwort Tips & Tricks
Gruberstraße 46c
8011 Poing (bei München)

Wir freuen uns über jede Zuschrift.


```
[YC3] 1: 'Programm: qbasic.bas
[UD7] 2: 'Funktion: Fenstertechnik unter Verwendung
[754] 3: ' von Interrupt-Routinen
[GM2] 4: 'Sprache: QBasic
[514] 5: 'Autor: Markus Zietlow
[1W4] 6: '(c) 1992 DMV Widuch GmbH & Co. KG
[000] 7:
[RC9] 8: DECLARE SUB CallInt(Interrupt%,AH%,AL%,BH%,
BL%,CH%,CL%,DH%,DL%)
[9C4] 9: DECLARE SUB Fehler(FCode%)
[UDC] 10: DECLARE SUB Floeschen(FField$,FZeiger%,Oben%,
Unten%,Links%,Rechts%,F.X%,F.Y%)
[0AC] 11: DECLARE SUB FCursor(FField$,FZeiger%,Oben%,
Unten%,Links%,Rechts%,F.X%,F.Y%)
[434] 12: DECLARE SUB LiesKonfig()
[3TC] 13: DECLARE SUB FSchliessen(FField$,FZeiger%,Oben%,
Unten%,Links%,Rechts%,F.X%,F.Y%)
[GPA] 14: DECLARE SUB FScroll(FField$,FZeiger%,Oben%,
Unten%,Links%,Rechts%)
[CNC] 15: DECLARE SUB FText(Text$,FField$,FZeiger%,Oben%,
Unten%,Links%,Rechts%,F.X%,F.Y%)
[0AA] 16: DECLARE SUB MenueZeigen(M.Punkt%,Oben%,Unten%,
Links%,Rechts%)
[M9D] 17: DECLARE SUB FOeffnen(Titel$,FField$,FZeiger%,Oben%,
Unten%,Links%,Rechts%,F.X%,F.Y%)
[000] 18:
[S25] 19: 'Fenstertechnik unter QBasic unter
[L77] 20: 'Verwendung der neuen Interrupt-Funktion
[E20] 21:
[TL1] 22: DEFINT A-Z
[AL6] 23: COMMON SHARED LGrenze, RGrenze, OGrenze
[TM6] 24: COMMON SHARED UGrenze, Farbe, False, Asm()
[5G3] 25: COMMON SHARED FGrenze
[LM3] 26: RESTORE IntProgramm
[Y53] 27: DIM Asm(1 TO (102 / 2))
[WT1] 28: Zaehler = 0
[6X3] 29: AsmStart = VARPTR(Asm(1))
[7S3] 30: DEF SEG = VARSEG(Asm(1))
[ED3] 31: DO WHILE Code$ <> "ENDE"
[UH1] 32: READ Code$
[R96] 33: POKE AsmStart + Zaehler, VAL("&h" + Code$)
[E93] 34: Zaehler = Zaehler + 1
[HS0] 35: LOOP
[091] 36: DEF SEG
[UU1] 37: FZeiger = 1
[PV1] 38: LGrenze = 1
[WV1] 39: OGrenze = 1
[LY1] 40: UGrenze = 25
[G91] 41: False = 0
[K42] 42: FGrenze = 3000
[N82] 43: DIM FField$(3000)
[0A2] 44: FOR Q% = 1 TO 3000
[DV1] 45: FField$(Q%) = " "
[671] 46: NEXT Q%
[412] 47: FOR Q% = 1 TO 4
[DX1] 48: FField$(Q%) = "0"
[671] 49: NEXT Q%
[2R1] 50: AH = 0: AL = 0
[6R1] 51: BH = 0: BL = 0
[AR1] 52: CH = 0: CL = 0
[ER1] 53: DH = 0: DL = 0
[E20] 54:
[XG4] 55: 'Anfang des Demo-Programms
[E20] 56:
[PN0] 57: CLS
[BE3] 58: FOR Zaehler% = 1 TO 25
[6G3] 59: PRINT STRING$(80, 177);
[U62] 60: NEXT Zaehler%
[3D3] 61: Links% = 3: Rechts% = 50
[LW2] 62: Oben% = 2: Unten% = 8
[HS4] 63: Titel$ = "Erstes Demo-Fenster"
[L9C] 64: CALL FOeffnen(Titel$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[65] 65: Links% = 20: Rechts% = 60
[AL3] 66: Oben% = 5: Unten% = 15
[G15] 67: Titel$ = "Zweites Demo-Fenster"
[L9C] 68: CALL FOeffnen(Titel$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[69] 69: Links% = 54: Rechts% = 77
[Y03] 70: Oben% = 4: Unten% = 21
[WZ4] 71: Titel$ = "Drittes Demo-Fenster"
[L9C] 72: CALL FOeffnen(Titel$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[SC3] 73: Links% = 2: Rechts% = 30
[R13] 74: Oben% = 7: Unten% = 18
[D15] 75: Titel$ = "Viertes Demo-Fenster"
[L9C] 76: CALL FOeffnen(Titel$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[LD3] 77: Links% = 4: Rechts% = 75
[W83] 78: Oben% = 11: Unten% = 25
[C15] 79: Titel$ = "Fünftes Demo-Fenster"
[L9C] 80: CALL FOeffnen(Titel$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[G93] 81: FOR Zaehler% = 1 TO 4
[X06] 82: Text$ = "***** Fenster und Pull-Down-"
[K96] 83: Text$ = Text$ + "Menüs sind mit der Ba"
[9C6] 84: Text$ = Text$ + "sic-Erweiterung von M"
```

```
[D36] 85: Text$ = Text$ + "arkus Zietlow aus der"
[V46] 86: Text$ = Text$ + " DOS International ke"
[HW5] 87: Text$ = Text$ + "in Problem mehr *****"
[UMB] 88: CALL FText(Text$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[U62] 89: NEXT Zaehler%
[DF2] 90: F.Y% = 10: F.X% = 8
[RSA] 91: CALL FCursor(FField$, FZeiger, Oben, Unten,
Links, Rechts, F.X, F.Y)
[126] 92: Text$ = "Drücken Sie bitte eine Taste für "
[WN6] 93: Text$ = Text$ + "die Menü-Demonstration!"
[UMB] 94: CALL FText(Text$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[5H3] 95: WHILE INKEY$ = "": WEND
[8K3] 96: Links% = 32: Rechts% = 48
[583] 97: Oben% = 10: Unten% = 20
[AF2] 98: Titel$ = "Menu"
[L9C] 99: CALL FOeffnen(Titel$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[LM3] 100: RESTORE MenuePunkte
[P93] 101: FOR Zaehler% = 1 TO 7
[FX1] 102: READ PUNKT$
[3E3] 103: F.Y% = Zaehler%: F.X% = 4
[RSA] 104: CALL FCursor(FField$, FZeiger, Oben, Unten,
Links, Rechts, F.X, F.Y)
[X82] 105: Text$ = PUNKT$
[UMB] 106: CALL FText(Text$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[U62] 107: NEXT Zaehler%
[KN8] 108: CALL MenueZeigen(M.Punkt, Oben, Unten, Links,
Rechts)
[SFB] 109: CALL FSchliessen(FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[52B] 110: CALL Floeschen(FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[9X2] 111: IF M.Punkt = 0 THEN
[KF2] 112: F.Y% = 3: F.X% = 18
[RSA] 113: CALL FCursor(FField$, FZeiger, Oben, Unten,
Links, Rechts, F.X, F.Y)
[LE5] 114: Text$ = "Sie haben die <ESC>-Taste "
[LA4] 115: Text$ = Text$ + "gedrückt!"
[UMB] 116: CALL FText(Text$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[2R0] 117: ELSE
[KF2] 118: F.Y% = 3: F.X% = 18
[RSA] 119: CALL FCursor(FField$, FZeiger, Oben, Unten,
Links, Rechts, F.X, F.Y)
[8F5] 120: Text$ = "Sie haben den Menüpunkt"
[TH4] 121: Text$ = Text$ + STR$(M.Punkt)
[CF4] 122: Text$ = Text$ + " ausgewählt!"
[UMB] 123: CALL FText(Text$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[801] 124: END IF
[692] 125: F.Y% = 8: F.X% = 6
[RSA] 126: CALL FCursor(FField$, FZeiger, Oben, Unten,
Links, Rechts, F.X, F.Y)
[LY6] 127: Text$ = "Drücken Sie eine Taste für das "
[EC6] 128: Text$ = Text$ + "Ende der Demonstration"
[UMB] 129: CALL FText(Text$, FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[5H3] 130: WHILE INKEY$ = "": WEND
[K93] 131: FOR Zaehler% = 1 TO 5
[SFB] 132: CALL FSchliessen(FField$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[U62] 133: NEXT Zaehler%
[4M0] 134: END
[E20] 135:
[CS6] 136: 'Daten für das Maschinenspracheprogramm
[E20] 137:
[K72] 138: IntProgramm:
[PC5] 139: DATA 55,8b,ec,8b,5e,06,8a,17,8b,5e,08,8a
[TD5] 140: DATA 37,8b,5e,0a,8a,0f,8b,5e,0c,8a,2f,8b
[SC5] 141: DATA 5e,0e,8a,1f,8a,c3,8b,5e,10,8a,3f,8a
[8A5] 142: DATA d8,8b,5e,12,8a,07,8b,5e,14,8a,27,55
[W85] 143: DATA 54,cd,00,5c,5d,53,8b,5e,06,88,17,8b
[5D5] 144: DATA 5e,08,88,37,8b,5e,0a,88,0f,8b,5e,0c
[CF5] 145: DATA 88,2f,8b,5e,0e,5b,8b,cb,88,0f,8b,5e
[V85] 146: DATA 10,88,2f,8b,5e,12,88,07,8b,5e,14,88
[UB3] 147: DATA 27,5d,ca,10,00,ENDE
[K72] 148: MenuePunkte:
[T95] 149: DATA " Auswahl","erfolgt"," durch"
[G95] 150: DATA "<RETURN>"," Abbruch"," durch"
[UY1] 151: DATA " <ESC>"
[000] 152:
[NR6] 153: 'Unterprogramm für den Interruptaufruf
[E20] 154:
[AH8] 155: SUB CallInt (Interrupt, AH, AL, BH, BL, CH,
CL, DH, DL)
[6X3] 156: AsmStart = VARPTR(Asm(1))
[7S3] 157: DEF SEG = VARSEG(Asm(1))
[7T4] 158: POKE AsmStart + 50, Interrupt
[CT8] 159: CALL ABSOLUTE(AH, AL, BH, BL, CH, CL, DH,
DL, VARPTR(Asm(1)))
[091] 160: DEF SEG
[HAL] 161: END SUB
[000] 162:
```



```
[CY6] 163: 'Unterprogramm, das den Cursor innerhalb
[405] 164: 'eines Fensters positioniert
[E20] 165: '
[RVA] 166: SUB FCursor (FFeld$, FZeiger, Oben, Unten
, Links, Rechts, F.X, F.Y)
[F12] 167: FCode = False
[C4E] 168: Wert = VAL("&H" + FFeld$(FZeiger) + FFeld
$(FZeiger + 1) + FFeld$(FZeiger + 2) + FFel
d$(FZeiger + 3))
[7E2] 169: IF Wert = 0 THEN
[5E1] 170: FCode = 10
[1W2] 171: GOTO FCursorEnde
[801] 172: END IF
[UR5] 173: IF (F.Y%>Unten-Oben-3) OR (F.Y%<1) THEN
[8E1] 174: FCode = 11
[1W2] 175: GOTO FCursorEnde
[801] 176: END IF
[MB6] 177: IF (F.X%>Rechts-Links-1) OR (F.X%<1) THEN
[BE1] 178: FCode = 12
[1W2] 179: GOTO FCursorEnde
[801] 180: END IF
[M45] 181: LOCATE Oben + F.Y% + 2, Links + F.X%
[S53] 182: F.X = F.X%: F.Y = F.Y%
[H62] 183: FCursorEnde:
[F12] 184: IF FCode THEN
[4Z2] 185: CALL Fehler(FCode)
[4M0] 186: END
[801] 187: END IF
[HA1] 188: END SUB
[000] 189:
[1T2] 190: 'Fehler-Routine
[E20] 191: '
[PW2] 192: SUB Fehler (FCode)
[GX0] 193: PRINT
[M04] 194: PRINT "Fehler"; FCode; ": ";
[XT2] 195: SELECT CASE FCode
[DX0] 196: CASE 1
[3E6] 197: PRINT "Linke Grenze des Fensters zu ";
[232] 198: PRINT "klein "
[GX0] 199: CASE 2
[EB6] 200: PRINT "Rechte Grenze des Fensters zu ";
[PG2] 201: PRINT "groß"
[1X0] 202: CASE 3
[TD6] 203: PRINT "Obere Grenze des Fensters zu ";
[232] 204: PRINT "klein "
[PX0] 205: CASE 4
[8D6] 206: PRINT "Untere Grenze des Fensters zu ";
[PG2] 207: PRINT "groß"
[TX0] 208: CASE 5
[UY5] 209: PRINT "Linke Grenze >= Rechte Grenze"
[WX0] 210: CASE 6
[A06] 211: PRINT "Obere Grenze >= Untere Grenze"
[2X0] 212: CASE 7
[W86] 213: PRINT "Obere Grenze - Untere Grenze ";
[SH1] 214: PRINT "< 4"
[2Y0] 215: CASE 8
[Z06] 216: PRINT "Rechte Grenze - Linke Grenze ";
[KH1] 217: PRINT "< 2"
[5Y0] 218: CASE 9
[KA6] 219: PRINT "Stapel für Fenster ('FGRENZE')";
[LK2] 220: PRINT "zu klein"
[D31] 221: CASE 10
[976] 222: PRINT "Alle Fenster sind geschlossen"
[H31] 223: CASE 11
[3B6] 224: PRINT "Y-Koordinate außerhalb des ";
[T53] 225: PRINT "Fensters "; ""
[N31] 226: CASE 12
[2B6] 227: PRINT "X-Koordinate außerhalb des ";
[T53] 228: PRINT "Fensters "; ""
[T31] 229: CASE 13
[Z36] 230: PRINT "Text enthält ASCII-Zeichen ";
[TN2] 231: PRINT "kleiner 32"
[X31] 232: CASE 14
[Z16] 233: PRINT "Fenster für Menü zu schmal ";
[CR2] 234: PRINT "(Breite<6)"
[GS1] 235: END SELECT
[HA1] 236: END SUB
[000] 237:
[FZ6] 238: 'Unterprogramm, das ein Fenster löscht
[E20] 239: '
[K1B] 240: SUB FLoeschen (FFeld$, FZeiger, Oben, Unt
en, Links, Rechts, F.X, F.Y)
[F12] 241: FCode = False
[C4E] 242: Wert = VAL("&H" + FFeld$(FZeiger) + FFeld
$(FZeiger + 1) + FFeld$(FZeiger + 2) + FFel
d$(FZeiger + 3))
[7E2] 243: IF Wert = 0 THEN
[5E1] 244: FCode = 10
[4Z2] 245: CALL Fehler(FCode)
[4M0] 246: END
[801] 247: END IF
[594] 248: FOR Y% = Oben + 3 TO Unten - 1
[9T2] 249: LOCATE Y%, Links + 1
[VN5] 250: PRINT STRING$(Rechts - Links - 1, 32);
[Y71] 251: NEXT Y%
[M11] 252: F.X = 1
[S11] 253: F.Y = 1
[HA1] 254: END SUB
```

```
[000] 255:
[1V6] 256: 'Unterprogramm, das ein Fenster öffnet
[E20] 257: '
[W8C] 258: SUB FOeffnen (Titel$, FFeld$, FZeiger, Ob
en, Unten, Links, Rechts, F.X, F.Y)
[F12] 259: FCode = False
[6R2] 260: CALL LiesKonfig
[ZW3] 261: IF Links% < LGrenze THEN
[N91] 262: FCode = 1
[TZ2] 263: GOTO FOeffnenEnde
[801] 264: END IF
[KY3] 265: IF Rechts% > RGrenze THEN
[R91] 266: FCode = 2
[TZ2] 267: GOTO FOeffnenEnde
[801] 268: END IF
[UL3] 269: IF Oben% < OGrenze THEN
[T91] 270: FCode = 3
[TZ2] 271: GOTO FOeffnenEnde
[801] 272: END IF
[1Y3] 273: IF Unten% > UGrenze THEN
[V91] 274: FCode = 4
[TZ2] 275: GOTO FOeffnenEnde
[801] 276: END IF
[HY3] 277: IF Links% >= Rechts% THEN
[X91] 278: FCode = 5
[TZ2] 279: GOTO FOeffnenEnde
[801] 280: END IF
[GM3] 281: IF Oben% >= Unten% THEN
[Z91] 282: FCode = 6
[TZ2] 283: GOTO FOeffnenEnde
[801] 284: END IF
[S53] 285: IF Unten% - Oben% < 4 THEN
[1A1] 286: FCode = 7
[TZ2] 287: GOTO FOeffnenEnde
[801] 288: END IF
[U14] 289: IF Rechts% - Links% < 2 THEN
[3A1] 290: FCode = 8
[TZ2] 291: GOTO FOeffnenEnde
[801] 292: END IF
[9U3] 293: DIF% = Rechts% - Links% - 1
[8R9] 294: LAENGE = FZeiger + (Rechts% - Links% + 1)
* (Unten% - Oben% + 1) + 7 + 4
[GX3] 295: IF LAENGE > FGrenze THEN
[5A1] 296: FCode = 9
[TZ2] 297: GOTO FOeffnenEnde
[801] 298: END IF
[VT4] 299: FFeld$(FZeiger + 4) = CHR$(Links%)
[405] 300: FFeld$(FZeiger + 5) = CHR$(Rechts%)
[6M4] 301: FFeld$(FZeiger + 6) = CHR$(Oben%)
[C4U] 302: FFeld$(FZeiger + 7) = CHR$(Unten%)
[K0D] 303: Wert=VAL("&H"+FFeld$(FZeiger)+FFeld$(FZe
iger+1)+FFeld$(FZeiger+2)+FFeld$(FZeiger+3))
[AK2] 304: IF Wert <> 0 THEN
[FZ3] 305: FFeld$(Wert + 8) = CHR$(F.X)
[RZ3] 306: FFeld$(Wert + 9) = CHR$(F.Y)
[801] 307: END IF
[T12] 308: F.X = 1: F.Y = 1
[CM3] 309: FOR Y% = Oben% TO Unten%
[TY3] 310: FOR X% = Links% TO Rechts%
[SUB] 311: FFeld$(FZeiger+10+(Y%-Oben%)*(Rechts%
-Links%+1)+(X%-Links%))=CHR$(SCREEN(Y%,X%))
NEXT X%
[V71] 312: NEXT Y%
[Y71] 313:
[6M8] 314: NZEIGER=FZeiger+(Rechts%-Links%+1)*(Unten
%-Oben%+1)+10
[EY7] 315: ZAHL$=LEFT$("0000",4-LEN(HEX$(FZeiger)))+
HEX$(FZeiger)
[412] 316: FOR Q% = 1 TO 4
[FB5] 317: FFeld$(NZEIGER+Q%-1)=MID$(ZAHL$,Q%,1)
[671] 318: NEXT Q%
[KW2] 319: FZeiger = NZEIGER
[LT3] 320: IF LEN(Titel$) > DIF% THEN
[VC4] 321: NTITEL$ = LEFT$(Titel$, DIF%)
[ZR0] 322: ELSE
[CL2] 323: NTITEL$ = Titel$
[801] 324: END IF
[4R4] 325: LFREI% = (DIF% - LEN(NTITEL$)) / 2
[FB5] 326: RFREI% = DIF% - LEN(NTITEL$) - LFREI%
[773] 327: LOCATE Oben%, Links%
[VV5] 328: PRINT CHR$(201); STRING$(DIF%,CHR$(205));
[T92] 329: PRINT CHR$(187);
[TK3] 330: LOCATE Oben% + 1, Links%
[GE5] 331: PRINT CHR$(186); STRING$(LFREI%, 32);
[XB5] 332: PRINT NTITEL%; STRING$(RFREI%, 32);
[P92] 333: PRINT CHR$(186);
[UK3] 334: LOCATE Oben% + 2, Links%
[SX5] 335: PRINT CHR$(199); STRING$(DIF%,CHR$(196));
[A92] 336: PRINT CHR$(182);
[8C4] 337: FOR Y% = Oben% + 3 TO Unten% - 1
[1G2] 338: LOCATE Y%, Links%
[P92] 339: PRINT CHR$(186);
[9D3] 340: STRING$(DIF%, CHR$(32));
[P92] 341: PRINT CHR$(186);
[Y71] 342: NEXT Y%
[R73] 343: LOCATE Unten%, Links%
```

Listing 3. Die Befehlsweiterung zeigt Ihnen anhand der Fenster Routinen, wie Sie attraktive Effekte erzielen.


```
[SV5] 344: PRINT CHR$(200); STRING$(DIF%,CHR$(205));
[W92] 345: PRINT CHR$(188);
[K65] 346: Links = Links%: Rechts = Rechts%
[HE4] 347: Oben = Oben%: Unten = Unten%
[872] 348: FoeffnenEnde:
[F12] 349: IF FCode THEN
[422] 350: CALL Fehler(FCode)
[4M0] 351: END
[801] 352: END IF
[HA1] 353: END SUB
[000] 354:
[N06] 355: 'Unterprogramm, das das geöffnete
[713] 356: 'Fenster schließt
[E20] 357: '
[1GB] 358: SUB FSchliessen (FFeld$, FZeiger, Oben, U
nten, Links, Rechts, F.X, F.Y)
[F12] 359: FCode = False
[C4E] 360: Wert = VAL("&H" + FFeld$(FZeiger) + FFeld
$(FZeiger + 1) + FFeld$(FZeiger + 2) + FFel
d$(FZeiger + 3))
[7E2] 361: IF Wert = 0 THEN
[5E1] 362: FCode = 10
[2L3] 363: GOTO FSchliessenEnde
[801] 364: END IF
[RE2] 365: FZeiger = Wert
[TS4] 366: Links = ASC(FFeld$(FZeiger + 4))
[L05] 367: Rechts = ASC(FFeld$(FZeiger + 5))
[NF4] 368: Oben = ASC(FFeld$(FZeiger + 6))
[BU4] 369: Unten = ASC(FFeld$(FZeiger + 7))
[LG3] 370: FOR Y% = Oben TO Unten
[RD2] 371: LOCATE Y%, Links
[9R4] 372: FOR Q% = 1 TO Rechts% - Links% + 1
[LM8] 373: PRINT FFeld$(FZeiger+10+(Y%-Oben)*(Re
chts-Links+1)+Q%-1);
[671] 374: NEXT Q%
[Y71] 375: NEXT Y%
[K0D] 376: Wert=VAL("&H"+FFeld$(FZeiger)+FFeld$(Fzei
ger+1)+FFeld$(FZeiger+2)+FFeld$(FZeiger+3))
[AK2] 377: IF Wert <> 0 THEN
[TA4] 378: Links = ASC(FFeld$(Wert + 4))
[ED4] 379: Rechts = ASC(FFeld$(Wert + 5))
[XZ3] 380: Oben = ASC(FFeld$(Wert + 6))
[EC4] 381: Unten = ASC(FFeld$(Wert + 7))
[3Z3] 382: F.X = ASC(FFeld$(Wert + 8))
[9Z3] 383: F.Y = ASC(FFeld$(Wert + 9))
[801] 384: END IF
[K0D] 385: Wert=VAL("&H"+FFeld$(FZeiger)+FFeld$(Fzei
ger+1)+FFeld$(FZeiger+2)+FFeld$(FZeiger+3))
[7E2] 386: IF Wert = 0 THEN
[G11] 387: F.X = 0
[M11] 388: F.Y = 0
[801] 389: END IF
[VU2] 390: FSchliessenEnde:
[F12] 391: IF FCode THEN
[422] 392: CALL Fehler(FCode)
[4M0] 393: END
[801] 394: END IF
[HA1] 395: END SUB
[000] 396:
[327] 397: 'Unterprogramm, das ein Fenster scrollt
[E20] 398: '
[LF9] 399: SUB FScroll (FFeld$, FZeiger, Oben, Unten
, Links, Rechts)
[F12] 400: FCode = False
[K0D] 401: Wert=VAL("&H"+FFeld$(FZeiger)+FFeld$(Fzei
ger+1)+FFeld$(FZeiger+2)+FFeld$(FZeiger+3))
[7E2] 402: IF Wert = 0 THEN
[5E1] 403: FCode = 10
[422] 404: CALL Fehler(FCode)
[4M0] 405: END
[801] 406: END IF
[382] 407: AH = &H6: AL = &H1
[KM2] 408: BH = Farbe%: BL = 0
[UE3] 409: CH = Oben + 2: CL = Links
[RG4] 410: DH = Unten - 2: DL = Rechts - 2
[3M2] 411: Interrupt = &H10
[LM7] 412: CALL CallInt(Interrupt,AH,AL,BH,BL,CH,CL,
DH,DL)
[HA1] 413: END SUB
[000] 414:
[HL5] 415: 'Unterprogramm, das einen Text im
[243] 416: 'Fenster darstellt
[E20] 417: '
[BKB] 418: SUB FText (Text$, FFeld$, FZeiger, Oben,
Unten, Links, Rechts, F.X, F.Y)
[F12] 419: FCode = False
[YU5] 420: IF LEN(Text$) = 0 THEN GOTO FTextEnde
[C74] 421: FOR AGAIN = 1 TO LEN(Text$)
[0F5] 422: IF ASC(MID$(Text$, AGAIN, 1)) < 32 THEN
[9V2] 423: AGAIN = LEN(Text$)
[801] 424: END IF
[VR1] 425: NEXT AGAIN
[G84] 426: IF AGAIN <= LEN(Text$) THEN
[EE1] 427: FCode = 13
[6H2] 428: GOTO FTextEnde
[801] 429: END IF
[C74] 430: FOR AGAIN = 1 TO LEN(Text$)
[RT4] 431: LOCATE Oben + F.Y + 2, Links + F.X
[H74] 432: PRINT MID$(Text$, AGAIN, 1);
```

```
[MR1] 433: F.X = F.X + 1
[GY4] 434: IF (F.X > Rechts - Links - 1) THEN
[LT2] 435: F.X = 1: F.Y = F.Y + 1
[BP4] 436: IF (F.Y > Unten - Oben - 3) THEN
[ZAL] 437: Farbe% = 0
[4F9] 438: CALL FScroll(FFeld$, FZeiger, Obe
n, Unten, Links, Rechts)
[F83] 439: F.Y = Unten - Oben - 3
[801] 440: END IF
[801] 441: END IF
[VR1] 442: NEXT AGAIN
[ZS1] 443: FTextEnde:
[F12] 444: IF FCode THEN
[422] 445: CALL Fehler(FCode)
[4M0] 446: END
[801] 447: END IF
[HA1] 448: END SUB
[000] 449:
[LU7] 450: 'Unterprogramm, das die System-Konfiguratio
n
[UH1] 451: 'ausliest
[FG2] 452: SUB LiesKonfig
[G82] 453: AH = &HF: AL = &H0
[3M2] 454: Interrupt = &H10
[9E8] 455: CALL CallInt(Interrupt, AH, AL, BH, BL, C
H, CL, DH, DL)
[1D3] 456: 'Anzahl der Spalten
[A02] 457: RGrnze = AH
[HA1] 458: END SUB
[000] 459:
[ZP6] 460: 'Unterprogramm, das ein Menü darstellt
[000] 461:
[1F8] 462: SUB MenueZeigen(M.Punkt,Oben,Unten,Links,Re
chts)
[F12] 463: FCode = False
[D44] 464: IF (Rechts - Links < 7) THEN
[HE1] 465: FCode = 14
[NE2] 466: GOTO MenueEnde
[801] 467: END IF
[6M1] 468: Y% = Oben + 3
[B03] 469: BalkenDarstellen:
[FS4] 470: ALT$ = CHR$(SCREEN(Y%, Links + 1))
[8V5] 471: ALT$ = ALT$ + CHR$(SCREEN(Y%, Links + 2))
[CV5] 472: ALT$ = ALT$ + CHR$(SCREEN(Y%, Links + 3))
[225] 473: ALT$ = ALT$ + CHR$(SCREEN(Y%, Rechts - 3))
[125] 474: ALT$ = ALT$ + CHR$(SCREEN(Y%, Rechts - 2))
[025] 475: ALT$ = ALT$ + CHR$(SCREEN(Y%, Rechts - 1))
[G75] 476: LOCATE Y%, Links + 1: PRINT " " ;
[SW4] 477: LOCATE Y%, Rechts - 3: PRINT " " ;
[6W2] 478: Tastaturabfrage:
[1C2] 479: TASTE$ = INKEY$
[DX4] 480: IF TASTE$ = CHR$(0) + CHR$(72) THEN
[9T2] 481: LOCATE Y%, Links + 1
[263] 482: PRINT LEFT$(ALT$, 3);
[9Y2] 483: LOCATE Y%, Rechts - 3
[MC3] 484: PRINT RIGHT$(ALT$, 3);
[VA1] 485: Y% = Y% - 1
[SU2] 486: WHILE Y% = Oben + 2
[XW1] 487: Y% = Unten - 1
[TR0] 488: WEND
[8R3] 489: GOTO BalkenDarstellen
[801] 490: END IF
[AX4] 491: IF TASTE$ = CHR$(0) + CHR$(80) THEN
[9T2] 492: LOCATE Y%, Links + 1
[263] 493: PRINT LEFT$(ALT$, 3);
[9Y2] 494: LOCATE Y%, Rechts - 3
[MC3] 495: PRINT RIGHT$(ALT$, 3);
[RA1] 496: Y% = Y% + 1
[PH2] 497: WHILE Y% = Unten
[6M1] 498: Y% = Oben + 3
[TR0] 499: WEND
[8R3] 500: GOTO BalkenDarstellen
[801] 501: END IF
[3R3] 502: IF TASTE$ = CHR$(13) THEN
[9T2] 503: LOCATE Y%, Links + 1
[263] 504: PRINT LEFT$(ALT$, 3);
[9Y2] 505: LOCATE Y%, Rechts - 3
[MC3] 506: PRINT RIGHT$(ALT$, 3);
[CE3] 507: M.Punkt = Y% - Oben - 2
[NE2] 508: GOTO MenueEnde
[801] 509: END IF
[PR3] 510: IF TASTE$ = CHR$(27) THEN
[9T2] 511: LOCATE Y%, Links + 1
[263] 512: PRINT LEFT$(ALT$, 3);
[9Y2] 513: LOCATE Y%, Rechts - 3
[MC3] 514: PRINT RIGHT$(ALT$, 3);
[0U1] 515: M.Punkt = 0
[NE2] 516: GOTO MenueEnde
[801] 517: END IF
[UH3] 518: GOTO Tastaturabfrage
[3S1] 519: MenueEnde:
[F12] 520: IF FCode THEN
[422] 521: CALL Fehler(FCode)
[4M0] 522: END
[801] 523: END IF
[HA1] 524: END SUB
```

Listing 3. Die Befehlsweiterung zeigt Ihnen anhand der Fenster-routinen, wie Sie attraktive Effekte erzielen.

DOSsier: PCL-Referenz

Hilfe für die Laserprogrammierung

Um Ihren Laserdrucker zu programmieren, müssen Sie wissen, welcher Befehl was kann und wie dieser Befehl aufgerufen wird. Im folgenden Tabellenwerk haben wir die Befehle der Druckerprogrammiersprache HP-PCL bis einschließlich Version 5 zusammengefaßt. Die Befehle sind nach ihrer Funktion gegliedert, um die Übersicht dieser Referenz zu erleichtern.

Name		Code	Hexa-dezimal	Bemerkung
Steuerbefehle				
Klingel (BELL)		CHR\$(7)	07	läßt einen zirka 0,5 Sekunden langen Piepton ertönen
Rückschritt (BS = Back Space)		CHR\$(8)	08	der Cursor wird um 1 Zeichen nach links bewegt
Horizontaltabulator (HT)		CHR\$(9)	09	der Cursor wird zum nächsten Tabulator-Stop bewegt
Zeilenvorschub (LF = Line Feed)		CHR\$(10)	0A	der Cursor wird in gleiche Spalte der nächsten Zeile bewegt
Seitenvorschub (FF = Form Feed)		CHR\$(12)	0C	die aktuelle Seite wird ausgedruckt
Wagenrücklauf (CR = Carriage Return)		CHR\$(13)	0D	bringt den Cursor an die Anfangsposition der gleichen Zeile
Shift out (SO)		CHR\$(14)	0E	wählt Sekundärschrift
Shift in (SI)		CHR\$(15)	0F	wählt Primärschrift
Escape (ESC)		CHR\$(27)	1B	leitet eine Escape-Sequenz ein
Leerzeichen (SPACE)		CHR\$(32)	20	erzeugt ein Leerzeichen
Arbeitsablaufbefehle				
Zurücksetzen auf Standard		ESCE	1B,45	führt bei PCL5 automatisch den HP-GL/2-Befehl »IN« aus
Anzahl der Kopien		ESC&I#X	1B,26,6C, #,58	(# = 1 bis 99)
Simplex/Duplex-Wahl:	Simplex	ESC&I0S	1B,26,6C,30,53	einseitiger Druck
	Duplex, lange Kante	ESC&I1S	1B,26,6C,31,53	doppelseitiger Druck mit Blattheftung in Hochformat
	Duplex, kurze Kante	ESC&I2S	1B,26,6C,32,53	doppelseitiger Druck mit Blattheftung in Querformat
Positionierung der logischen Seite:	in Querrichtung	ESC&I#U	1B,26,6C, #,55	# = Dezipunkte (1/720 Zoll) (# = -32767 bis +32767)
	in Längsrichtung	ESC&I#Z	1B,26,6C, #,5A	# = Dezipunkte (1/720 Zoll) (# = -32767 bis +32767)
Einstellung des Doppeldrucks (Duplex)		ESC&a0G	1B,26,61,30,47	wähle nächste Seite (nur für Duplex-Drucker)
		ESC&a1G	1B,26,61,31,47	wähle Vorderseite (nur für Duplex-Drucker)
		ESC&a2G	1B,26,61,32,47	wähle Rückseite (nur für Duplex-Drucker)
Trennung von einzelnen Jobs		ESC&I1T	1B,26,6C,31,54	Befehl muß am Auftragsende, vor Reset gesendet werden (PCL5)
Papiereinzug:	Seitenauswurf	ESC&I0H	1B,26,6C,30,48	aktuelle Seite ausgeben
	Mehrzweckkassette I	ESC&I1H	1B,26,6C,31,48	Papier aus dem 1. Schacht einziehen
	manuelle Papierzuführung	ESC&I2H	1B,26,6C,32,48	Papier aus dem manuellen Einzugsschlitz einziehen
	manuelle Umschlagzuführung	ESC&I3H	1B,26,6C,33,48	Briefumschlag aus dem manuellen Einzugsschlitz einziehen
	Mehrzweckkassette II	ESC&I4H	1B,26,6C,34,48	Papier aus dem 2. Schacht einziehen
	automatische Umschlagzuführung	ESC&I6H	1B,26,6C,36,48	
Seitenmaß einstellen:	DIN A4	ESC&I26A	1B,26,6C,32,36,41	Papier: 210mm x 297mm (Achtung: Nicht jeder Drucker kann jedes Papierformat verarbeiten)
	DIN A5	ESC&I27A	1B,26,6C,32,37,41	Papier: 148mm x 210mm
	DIN A6	ESC&I28A	1B,26,6C,32,38,41	Papier: 105mm x 148mm
	B5	ESC&I37A	1B,26,6C,33,37,41	Papier: 257mm x 185mm
	Executive	ESC&I1A	1B,26,6C,31,41	Papier: 7,25 Zoll x 10 1/2 Zoll
	Letter	ESC&I2A	1B,26,6C,32,41	Papier: 8,50 Zoll x 11 Zoll
	Legal	ESC&I3A	1B,26,6C,33,41	Papier: 8,50 Zoll x 14 Zoll
	Monarch	ESC&I80A	1B,26,6C,38,30,41	Umschlag: 3,9 Zoll x 7,5 Zoll
	Commercial 10	ESC&I81A	1B,26,6C,38,31,41	Umschlag: 4,1 Zoll x 9,5 Zoll
	International DL	ESC&I90A	1B,26,6C,39,30,41	Umschlag: 110mm x 220 mm
	International C5	ESC&I91A	1B,26,6C,39,31,41	Umschlag: 162mm x 229 mm
Seitenlänge in Zeilen:		ESC&I#P	1B,26,6C, #,50	

Name		Code	Hexa-dezimal	Bemerkung
Sonstige Seitenbeschreibungsbefehle				
Seitenausrichtung:	Hochformat	ESC&I0O	1B,26,6C,30,4F	
	Querformat	ESC&I1O	1B,26,6C,31,4F	
	umgekehrtes Hochformat	ESC&I2O	1B,26,6C,32,4F	
	umgekehrtes Querformat	ESC&I3O	1B,26,6C,33,4F	
Druckrichtung		ESC&a # P	1B,26,61, # ,50	# = Gradzahl im Uhrzeigersinn, nur 90-Grad-Schritte (PCL5)
Oberer Rand		ESC&l # E	1B,26,6C, # ,45	# = Anzahl der Zeilen, die am oberen Rand frei gelassen werden
Textlänge (unterer Rand)		ESC&l # F	1B,26,6C, # ,46	# = Anzahl der Zeilen pro Seite
Linker Rand		ESC&a # L	1B,26,61, # ,4C	# = Anzahl der Spalten
Rechter Rand		ESC&a # M	1B,26,61, # ,4D	# = Anzahl der Spalten
Seitliche Ränder löschen		ESC9	1B,39	
Perforation überspringen	ein	ESC&I1L	1B,26,6C,31,4C	
	aus	ESC&I0L	1B,26,6C,30,4C	
Abstände:	Spaltenabstand, horizontaler (HMI)	ESC&k # H	1B,26,6B, # ,48	stellt Zeichenabstand auf # /120 Zoll ein
	Zeilenabstand, vertikaler (VMI)	ESC&l # C	1B,26,6C, # ,43	stellt Zeilenabstand auf # /48 Zoll ein
	Zeilen pro Zoll	ESC&l # D	1B,26,6C, # ,44	Werkseinstellung: # = 6 (# = 0,1,2,3,4,6,8,12,16,24,48)
Cursorsteuerung (nur für PCL-Cursor)				
Achtung: PCL und HP-GL/2 haben zwei getrennte Cursor				
Cursorposition speichern		ESC&f0S	1B,26,66,30,53	speichert bis zu 20 Positionen
Cursorposition abrufen		ESC&f1S	1B,26,66,31,53	ruft zuletzt gespeicherte Position auf
Horizontale Cursorbewegung	in Spalten	ESC&a # C	1B,26,61, # ,43	in # Spalten
	in Punkten	ESC*p # X	1B,2A,70, # ,58	in # /300 Zoll
	in Zehntelpunkten	ESC&a # H	1B,26,61, # ,48	in # /720 Zoll
	in Zeilen	ESC&a # R	1B,26,61, # ,52	in # Spalten
Vertikale Cursorbewegung	in Punkten	ESC*p # Y	1B,2A,70, # ,59	in # /300 Zoll
	in Zehntelpunkten	ESC&a # V	1B,26,61, # ,56	in # /720 Zoll
		ESC=	1B,3D	führt einen halben Zeilenvorschub aus
Halber Zeilenvorschub				
Zeilenabschluß/ Zeilenende		ESC&k0G	1B,26,6B,30,47	CR=CR, LF=LF, FF=FF
		ESC&k1G	1B,26,6B,31,47	CR=CR+LF, LF=LF, FF=FF
		ESC&k2G	1B,26,6B,32,47	CR=CR, LF=CR+LF, FF=CR+FF
		ESC&k3G	1B,26,6B,33,47	CR=CR+LF, LF=CR+LF, FF=CR+FF
Weitere Befehle zur Cursorpositionierung siehe Steuerbefehle				
Auswahl einer (Primär)-Schrift				
Ausrichtung:	Hochformat	ESC&I0O	1B,26,6C,30,4F	Achtung: Die Wahl der Ausrichtung beeinflusst immer die primäre und sekundäre Schrift
	Querformat	ESC&I1O	1B,26,6C,31,4F	
Zeichensatz:	PC 850	ESC(12U	1B,28,31,32,55	
	PC-8 (IBM-PC USA)	ESC(10U	1B,28,31,30,55	
	ISO 6 ANSI ASCII	ESC(0U	1B,28,30,55	
	Roman-8	ESC(8U	1B,28,38,55	
	HP Roman Extension	ESC(0E	1B,28,30,45	
	ISO 21 Deutschland	ESC(1G	1B,28,31,47	
	HP Deutschland	ESC(0G	1B,28,30,47	nicht zu empfehlen wegen abnehmender Bedeutung
	HP Greek	ESC(8G	1B,28,38,47	
	HP Legal	ESC(1U	1B,28,31,55	
	HP Linedraw	ESC(0B	1B,28,30,42	
	HP Math 7	ESC(0A	1B,28,30,41	
	HP Spanien	ESC(3S	1B,28,33,53	
	ISO 57 China	ESC(2K	1B,28,32,4B	nicht zu empfehlen wegen abnehmender Bedeutung
	PC-8 Dänemark/Norwegen	ESC(11U	1B,28,31,31,55	
	ECMA-94 Latin 1	ESC(0N	1B,28,30,4E	
	ISO 69 Frankreich (neu)	ESC(1F	1B,28,31,46	
	ISO 25 Frankreich	ESC(0F	1B,28,30,46	nicht zu empfehlen wegen abnehmender Bedeutung
	ISO 2 IRV	ESC(2U	1B,28,32,55	nicht zu empfehlen wegen abnehmender Bedeutung
	ISO 15 Italien	ESC(0I	1B,28,30,49	
	ISO 14 JIS ASCII	ESC(0K	1B,28,30,4B	nicht zu empfehlen wegen abnehmender Bedeutung

Name		Code	Hexa-dezimal	Bemerkung
	ISO 60 Norwegen Version 1	ESC(0D	1B,28,30,44	
	ISO 61 Norwegen Version 2	ESC(1D	1B,28,31,44	nicht zu empfehlen wegen abnehmender Bedeutung
	ISO 84 Portugal (IBM)	ESC(5S	1B,28,35,53	nicht zu empfehlen wegen abnehmender Bedeutung
	ISO 16 Portugal	ESC(4S	1B,28,34,53	nicht zu empfehlen wegen abnehmender Bedeutung
	ISO 10 Schweden/ Finnland	ESC(3S	1B,28,33,53	nicht zu empfehlen wegen abnehmender Bedeutung
	ISO 11 Schweden	ESC(0S	1B,28,30,53	
	HP Spanien	ESC(1S	1B,28,31,53	nicht zu empfehlen wegen abnehmender Bedeutung
	ISO 17 Spanien	ESC(2S	1B,28,32,53	
	ISO 85 Spanien (IBM)	ESC(6S	1B,28,36,53	nicht zu empfehlen wegen abnehmender Bedeutung
	ISO 4 UK (United Kingdom)	ESC(1E	1B,28,31,45	
Zeichenabstand:	fest	ESC(s0P	1B,28,73,30,50	
	variabel (proportional)	ESC(s1P	1B,28,73,31,50	
Zeichendichte:	in Zeichen/Zoll	ESC(s#H	1B,28,73, #, 48	# = Anzahl der Zeichen/Zoll
	in zirka 10 Zeichen/Zoll	ESC&k0S	1B,26,6B,30,53	
	in zirka 16,5 Zeichen/Zoll	ESC&k2S	1B,26,6B,32,53	
	in zirka 12 Zeichen/Zoll	ESC&k4S	1B,26,6B,34,53	
Zeichengröße in Punkt		ESC(s#V	1B,28,73, #, 56	# = Anzahl der Punkte (1 Punkt = 1/72 Zoll)
Schriftstil:	Normal	ESC(s0S	1B,28,73,30,53	upright, solid
	Kursiv	ESC(s1S	1B,28,73,31,53	italic
	Kondenz	ESC(s4S	1B,28,73,34,53	condensed (PCL5)
	Kondenz Kursiv	ESC(s5S	1B,28,73,35,53	condensed italic (PCL5)
	Komprimiert	ESC(s8S	1B,28,73,38,53	compressed, or extra condensed (PCL5)
	Exponiert	ESC(s24S	1B,28,73,32,34,53	expanded (PCL5)
	Außenfläche	ESC(s32S	1B,28,73,33,32,53	outline (PCL5)
	Innenfläche	ESC(s64S	1B,28,73,36,34,53	inline (PCL5)
	Schatten	ESC(s128S	1B,28,73,31,32,3 8,53	shadowed (PCL5)
	Schatten mit Innenflächen	ESC(s160S	1B,28,73,31,36,3 0,53	outline shadowed (PCL5)
Schriftstärke (Anschlagstärke)		ESC(s#B	1B,28,73, #, 42	(# = -7 bis +7) (-3 = hell, 0 = normal)
Schriftart:	Lineprinter	ESC(s0T	1B,28,73,30,54	
	Pica	ESC(s1T	1B,28,73,31,54	
	Elite	ESC(s2T	1B,28,73,32,54	
	Courier	ESC(s3T	1B,28,73,33,54	
	Helvetica	ESC(s4T	1B,28,73,34,54	
	Times Roman	ESC(s5T	1B,28,73,35,54	
	Gothic	ESC(s6T	1B,28,73,36,54	
	Script	ESC(s7T	1B,28,73,37,54	
	Prestige	ESC(s8T	1B,28,73,38,54	
	Caston	ESC(s9T	1B,28,73,39,54	
	Orator	ESC(s10T	1B,28,73,31,30,54	
	Presentations	ESC(s11T	1B,28,73,31,31,54	
	Swiss 721	ESC(s14T	1B,28,73,31,34,54	
	Dutch 801	ESC(s15T	1B,28,73,31,35,54	
	CG Times	ESC(s4101T	1B,28,73,34,31,3 0,31,54	
	Univers	ESC(s4148T	1B,28,73,34,31,3 4,38,54	
Auswahl der Primärschrift über Kennung		ESC(# X	1B,28, #, 58	# = Kennnummer (ID) (# = 0 bis 32767)
Auswahl der Sekundärschrift über Kennung		ESC) # X	1B,29, #, 58	# = Kennnummer (ID) (# = 0 bis 32767)
Standardschrift				
Primär:	wähle Standardschrift	ESC(0@	1B,28,30,40	
	wähle Standardschrift	ESC(1@	1B,28,31,40	
	wähle aktuelle Primärschrift	ESC(2@	1B,28,32,40	
	wähle Primärschrift	ESC(3@	1B,28,33,40	
Standardschrift Sekundär:	wähle Standardschrift	ESC)0@	1B,29,30,40	
	wähle primäre Standardschrift	ESC)1@	1B,29,31,40	
	wähle aktuelle Primärschrift	ESC)2@	1B,29,32,40	
	wähle Sekundärschrift	ESC)3@	1B,29,33,40	

Name		Code	Hexa-dezimal	Bemerkung
Unterstreichen:	fest angepaßt nicht unterstreichen	ESC&d0D ESC&d3D ESC&d@	1B,26,64,30,44 1B,26,64,33,44 1B,26,64,40	
Transparentes Drucken (für ASCII < 32)		ESC&p#X[Daten]	1B,26,70, #,58, [Daten]	# = Anzahl der Bytes – Es werden die nächsten # Bytes auch dann gedruckt, wenn der Wert kleiner als 32 ist.
Verwaltung von Schriften				
Festlegung der Schriftartkennung		ESC*c#D	1B,2A,63, #,44	(# = ID-Nummer, Kennung der Schrift, Wert von 0 bis 32767)
Zeichen auswählen		ESC*c#E	1B,2A,63, #,45	(# = Zeichencode)
Steuerung von Schriften und Zeichen:	lösche alle Schriften	ESC*c0F	1B,2A,63,30,46	
	lösche alle temporären Schriften	ESC*c1F	1B,2A,63,31,46	
	lösche angewählte Schrift	ESC*c2F	1B,2A,63,32,46	Schrift mit der zuletzt angegebenen Kennung wird gelöscht
	lösche angewähltes Zeichen	ESC*c3F	1B,2A,63,33,46	lösche zuletzt angegebenes Zeichen
	mache angewählte Schrift temporär	ESC*c4F	1B,2A,63,34,46	
	mache angewählte Schrift permanent	ESC*c5F	1B,2A,63,35,46	
	mache aktuelle Schrift temporär	ESC*c6F	1B,2A,63,36,46	
Generierung einer ladbaren Schrift				
Schriftkopf (Header) übertragen		ESC)s#W [Daten]	1B,29,73, #,57, [Daten]	# = Anzahl der benötigten Bytes
Zeichen-Code	ESC*c#E		1B,2A,63, #,45	# = ASCII-Code
Zeichen übertragen		ESC(s#W [Daten]	1B,28,73, #,57, [Daten]	# = Anzahl der benötigten Bytes
Rastergrafik				
Einstellung der Auflösung		ESC*t#R	1B,2A,74, #,52	# = 75,100,150,300
Darstellung der Grafik:	Bild um 180° gedreht	ESC*r0F	1B,2A,72,30,46	(PCL5)
	Bild im Querformat	ESC*r3F	1B,2A,72,33,46	(PCL5)
Beginn der Rastergrafik:	am linken Rand	ESC*r0A	1B,2A,72,30,41	linker Rand ganz links
	beim Cürsor	ESC*r1A	1B,2A,72,31,41	linker Rand bei aktueller Cursorposition
Y-Versatz des Rasters		ESC*b#Y	1B,2A,62, #,59	(PCL5)
Komprimierung der Rasterdaten:	uncodiert	ESC*b0M	1B,2A,62,30,4D	(PCL5)
	Laufängen-codiert	ESC*b1M	1B,2A,62,31,4D	(PCL5)
	Tagged Image File Format	ESC*b2M	1B,2A,62,32,4D	(PCL5)
	Delta Row	ESC*b3M	1B,2A,62,33,4D	(PCL5)
Übertragung der Grafikdaten		ESC*b#W [Daten]	1B,2A,62, #,57, [Daten]	# = Anzahl der Grafikdatenbytes, die gesendet werden sollen.
Ende der Rastergrafik		ESC*rB	1B,2A,72,42	schaltet den Drucker wieder in den Textmodus
Rasterhöhe		ESC*r#T	1B,2A,72, #,54	# = Anzahl der Rasterzeilen
Rasterbreite		ESC*r#S	1B,2A,72, #,53	# = Anzahl der Punkte einer gegebenen Auflösung
Druckmodell (nur PCL5)				
				zum Überlagern von Mustern mit Grafik und Text
Muster auswählen:	Vollton Schwarz	ESC*v0T	1B,2A,76,30,54	
	Vollton Weiß	ESC*v1T	1B,2A,76,31,54	
	Grautönung	ESC*v2T	1B,2A,76,32,54	
	Kreuzschraffur	ESC*v3T	1B,2A,76,33,54	
Benutzermuster auswählen:				siehe Füllmusterbefehl ESC+*c#G
Quelle auswählen:	transparent	ESC*v0N	1B,2A,76,30,4E	
	deckend	ESC*v1N	1B,2A,76,31,4E	
Muster auswählen:	transparent	ESC*v0O	1B,2A,76,30,4F	
	deckend	ESC*v1O	1B,2A,76,31,4F	
Füllmuster				
Breite des Rechtecks	in Punkten	ESC*c#A	1B,2A,63, #,41	
	in Dezipunkten	ESC*c#H	1B,2A,63, #,48	in Zehntelpunkten
Höhe des Rechtecks	in Punkten	ESC*c#B	1B,2A,63, #,42	
	in Dezipunkten	ESC*c#V	1B,2A,63, #,56	in Zehntelpunkten
Rechteckfläche ausfüllen:	mit Vollton Schwarz	ESC*c0P	1B,2A,63,30,50	
	mit Vollton Weiß	ESC*c1P	1B,2A,63,31,50	zum Beispiel zum Löschen

Name		Code	Hexa-dezimal	Bemerkung
Kennung des Mu- sters:	mit Grautönung	ESC*c2P	1B,2A,63,32,50	die Prozente der Tönung wurden vorher eingestellt mit ESC+*c#G
	mit Benutzerdefinition	ESC*c3P	1B,2A,63,33,50	wurde vorher mit ESC+*c#G definiert
	mit aktuellem Muster	ESC*c5P	1B,2A,63,35,50	
		ESC*c#G	1B,2A,63, #,47	in Grautönung Prozent (2%,10%,15%,30%,45%,70%,90%,100%)
				oder Mustertyp (Benutzerdefinition): # = 1 horizontale Linien # = 2 vertikale Linien # = 3 diagonale Linien von links unten nach rechts oben # = 4 diagonale Linien von links oben nach recht unten # = 5 Gitter # = 6 diagonales Gitter
Makros				Ein Makro darf keine HP-GL/2-Befehle enthalten!
Makro-Kennnummer festlegen, Makro wählen		ESC&f#Y	1B,26,66, #,59	# = Makro-Kennnummer
Makro-Steuerung:	Beginn der Makro-Definition	ESC&f0X	1B,26,66,30,58	
	Ende der Makro-Definition	ESC&f1X	1B,26,66,31,58	
	Makro ausführen	ESC&f2X	1B,26,66,32,58	
	Makro aufrufen	ESC&f3X	1B,26,66,33,58	
	Überlagern aktivieren	ESC&f4X	1B,26,66,34,58	startet automatische Makro-Ausführung
	Überlagern deaktivieren	ESC&f5X	1B,26,66,35,58	beendet automatische Makro-Ausführung
	Löschen aller Makros	ESC&f6X	1B,26,66,36,58	
	Löschen aller temporären Makros	ESC&f7X	1B,26,66,37,58	
	Löschen des zuletzt gewählten Makros	ESC&f8X	1B,26,66,38,58	
	Makro temporär machen	ESC&f9X	1B,26,66,39,58	
	Makro permanent machen	ESC&f10X	1B,26,66,31,30,58	
Sonstiges:				
Anzeigefunktion	ein	ESCY	1B,59	
	aus	ESCZ	1B,5A	
Automatischer Zeilenumbruch	ein	ESC&s0C	1B,26,73,30,43	
	aus	ESC&s1C	1B,26,73,31,43	
HP-GL/2-Plotteremulation				erlaubt Ausführung von HP-GL/2-Befehlen (PCL5)
HP-GL/2-Modus	an letzte HP-GL/2-Position	ESC%0B	1B,25,30,42	aktiviert den HP-GL/2-Modus (Plotter-)
PCL-Modus	an letzte PCL-Position	ESC%1B	1B,25,31,42	aktiviert den HP-GL/2-Modus
	an letzte PCL-Cursorposition	ESC%0A	1B,25,30,41	aktiviert den PCL-Modus
	an letzte HP-GL-Cursorposition	ESC%1A	1B,25,31,41	aktiviert den PCL-Modus
HP-GL/2-Plot-Breite		ESC*c#K	1B,2A,63, #,4B	# = Breite in Zoll
HP-GL/2-Plot-Länge		ESC*c#L	1B,2A,63, #,4C	# = Länge in Zoll
aktuelle Position im PCL-Grafikbereich		ESC*c0T	1B,2A,63,30,54	
Breite des Grafikbereiches		ESC*c#X	1B,2A,63, #,58	# = Breite in Zehntelpunkt
Länge des Grafikbereiches		ESC*c#Y	1B,2A,63, #,59	# = Länge in Zehntelpunkt

Übersicht über die HP-PCL-Befehle bis zur Version 5

Bisher konnte sich nicht jeder aktiv an der Programmierung seines Laserdruckers beteiligen, da ihm in der Regel eine Befehlsübersicht dazu fehlte. Dabei eröffnen sich Ihnen mit der Programmierung Ihres Laserdruckers neue Chancen, Informationen auf Papier zu drucken. Außerdem können

Sie mit der Sprache HP-PCL den Druckvorgang Ihres Laserdruckers kontrollieren und zusätzlich manipulieren. So lassen sich Vorgänge wie Papiereinzug, Seitenmaß, Positionierung der logischen Seite, Druckrichtung und Cursorpositionierung – um nur einiges zu nennen – per Software beeinflussen.

Zusätzlich können Sie die Schriftarten und Zeichenabstände sowie Grafiksymbbole und Plotteremulationen mit einem selbstgefertigten Programm einstellen. Hier nun die Befehlsreferenz der HP-PCL (Hewlett Packard Printer Control Language) bis Version 5.

(Ulrich Borgdorf/st)

Drucker-Programmierung

Vom Laser aufs Papier

Laserdrucker werden grundsätzlich, Tintenstrahldrucker häufig in PCL (Printer Control Language) programmiert. Andere Sprachen, wie Postscript oder TrueType, sind in den Grundversionen der meisten Laserdrucker nicht implementiert, so daß PCL 5 den Schwerpunkt bildet.

Die Druckersprache PCL ist wie die meisten anderen in zwei Befehlskategorien (Steuercodes und Escape-Sequenzen) unterteilt. Ein Steuercode ist immer ein Zeichen (Byte) lang, das einen Wert kleiner als 32 hat.

Eine Escape-Sequenz ist mehrere Byte lang und beginnt mit dem Escape-Steuercode (Wert: 27 = 1Bhex), worauf eine Anzahl von Zeichen folgt. Dies hat den Vorteil, daß in einer Escape-Sequenz ein Parameter integriert werden kann.

Von Escape-Sequenzen und Parametern

Ein Parameter ist eine ergänzende Angabe zu einem Befehl, die den vorgesehenen Ablauf beeinflusst. Bei der Druckersprache PCL ist es üblich, Parameter durch das Zeichen »#« darzustellen. Anstelle dieses Zeichens muß dem Drucker ein entsprechender Wert übermittelt werden. So wird zum Beispiel der Zeilenabstand über einen Parameter definiert, da es unzumutbar ist, für jeden denkbaren Zeilenabstand einen eigenen Befehl zu erzeugen.

● **Parameter-Übergabe:** Ein Parameter wird in der PCL-Sprache grundsätzlich mit der Funktion »STR\$(Wert)« eingebunden und nicht wie bei Nadeldruckern mit »CHR\$(Wert)«. Dadurch sind auch Parameter mit Nachkommastellen erlaubt, die jedoch durch einen Punkt anstatt durch ein Komma zu trennen sind. Diese Eigenart der Laserdrucker bekommt man zu spüren, wenn zum Beispiel ein Druckertreiber, der aus hexadezimalen Werten besteht, erzeugt werden soll – hier sind nämlich die Parameter umständlich umzurechnen: Aus »STR\$(10)« werden dann die beiden Byte 31hex und 30hex.

Eine Escape-Sequenz ist bei Laserdruckern grundsätzlich wie folgt aufgebaut:

<Escape-Steuercode> <Befehlsgruppe>
<Parameter> <Unteranweisung>.

So lautet der Befehl, der den linken Rand auf 12 Zeichen einstellt

<ESC> &a 12L.

Alle Befehle, die das horizontale Seitenformat beschreiben, haben die Befehlsgruppenkennung »&a«. Wenn Sie mehrere Befehle einer Befehlsgruppe an den Drucker übermitteln möchten, so können Sie dies durch Verkettung der Parameter und der Unteranweisungen verkürzen. Zum Beispiel wird aus der Sequenz

<ESC> xx 12 A <ESC> xx 15 B <ESC> xx 30 C
in der verkürzten Darstellung

<ESC> xx 12 a 15 b 30 C.

Dabei ist die Unteranweisung immer in Kleinbuchstaben anzugeben. Nur die letzte Unteranweisung muß aus Großbuchstaben bestehen, weil diese das Ende des Escape-Befehlsmodus signalisiert.

● **PCL4, PCL5 und HP-GL/2:** Ein Problem bei den Escape-Sequenzen ist, daß es verschiedene Versionen von PCL gibt. Derzeit verbreitet sind die beiden Versionen PCL4 (zum Beispiel beim HP-Laserjet II) und PCL5 (beim HP-Laserjet III). Einige Befehle sind nicht versionsabhängig, sondern hängen von der Druckerausstattung ab. So kann das Papier aus der zweiten Mehrzweckkassette nur entnommen werden, wenn diese auch vorhanden ist, oder das Papier kann nur von beiden Seiten bedruckt werden, wenn es sich um einen Duplex-Drucker handelt. Wesentliches Merkmal der Version PCL5 ist die Erweiterung um skalierbare Zeichensätze und um die integrierte Plottersprache HP-GL/2 (Hewlett Packard Graphic Language). Ein skalierbarer Zeichensatz ist dadurch gekennzeichnet, daß seine Größe durch einen Parameter bestimmt wird.

Die Steuerung eines Seitendruckers

Da der Befehlsumfang diesen Beitrag sprengen würde, finden Sie im Beitrag »Hilfe für die Laserprogrammierung« eine entsprechende PCL-Referenz bis Version 5.

Da Laserdrucker zur Kategorie der Seitendrucker gehören, ist es notwendig, daß erst eine ganze Druckseite übermittelt wird. Normalerweise haben Laserdrucker eine Genauigkeit von 1/300stel Zoll (1 Zoll = 2,54 cm). Ein

Byte	Bedeutung (der Header folgt der Sequenz »ESC)(s #W«)
0	hat immer den Wert 0
1	hat immer den Wert 26
2	hat immer den Wert 0
3	Zeichenvorrat: # = 0 Zeichen 33 bis 127, # = 1 Zeichen 33 bis 127 und 160 bis 255 (Roman-8), # = 2 Zeichen 33 bis 255
4/5	haben immer den Wert 0
6/7	Position der Grundlinie (Oberlänge)
8	hat immer den Wert 0
9	Breite der Zeichen in Druckpunkten
10	hat immer den Wert 0
11	Höhe der Zeichen in Druckpunkten
12	Ausrichtung (0=Hochformat, 1=Querformat)
13	Zeichenabstand (0=fest, 1=proportional)
14/15	Zeichensatzkennung in 2 Byte verschlüsselt: (Nummer des Zeichencodes*32)+(ASCII-Wert des Buchstabens-64). Beispiel – Zeichensatzkennung 8U: (8*32=256)+(85-64=21)=277; Byte 14 = INT(277/256) = 1, Byte 15 = 277 - Byte 14 = 21
16/17	Zeichendichte in Zeichen pro Zoll, multipliziert mit 4. Beispiel: 10 cpi = 300 dpi / 10 cpi, also 30 d/c oder 30 Punkte pro Zeichen; 30*4=120, Byte 16 = INT(120/256) = 0, Byte 17 = 120 - Byte 16 = 120
18/19	Schriftgröße in Druckpunkten, multipliziert mit 4. Beispiel: 12-Punkt-Schrift = 72/12 = 6 Druckpunkte; 300 dpi / 6 = 50; 50*4=200; Byte 16 = INT(200/256) = 0, Byte 17 = 200 - Byte 16 = 200
20-22	haben immer den Wert 0
23	Schriftstil (0=normal, aufrecht; 1=kursiv)
24	Schriftstärke (-7 bis +7) wird umgerechnet: -7 bis -1 werden zu 249 bis 255, 0 bleibt 0, +1 bis +7 werden zu 1 bis 7
25	Schriftart gemäß Parameter des Befehls »ESC)(s #T«. Beispiel: # = 0 für Lineprinter, # = 4 für Helvetica, # = 5 für Times Roman

Tabelle 1. Aufbau des Font-Headers

Laserdrucker gibt Bilder aus, die aus Punkten der Größe 1/300stel Zoll bestehen. Wenn ein Text ausgedruckt werden soll, holt der Laserdrucker für jedes benötigte Zeichen ein Musterbild einer Schrift (Font) und kopiert dieses an die derzeitige Druckposition, die Cursor genannt wird. Die Druckposition kann innerhalb der aktuellen Druckseite verändert werden.

● **Der Steuercode:** Bestimmte Bedingungen signalisieren dem Drucker, daß er die in seinem Speicher befindliche Seite ausdrucken soll, wenn die Übertragung der folgenden Seite begonnen hat, 30 Sekunden lang keine Daten zum Drucker übermittelt wurden oder ein entsprechender Befehl übergeben wurde.

Wenn Sie im Betriebssystem mit der Anweisung »dir > prn« das Inhaltsverzeichnis eines Datenträgers zum Laserdrucker schicken, werden Sie dies zu spüren bekommen, da der Drucker mit dem Ausdruck der letzten Seite eine halbe Minute wartet. Wenn Sie jedoch direkt nach der Anweisung »dir > prn« noch die Zeile »echo [alt-12] > prn« einfügen, erfolgt der gesamte Ausdruck sofort. Dies erreicht der Steuercode-Befehl »FormFeed« (Wert: 12/0Chex).

● **Cursorsteuerung:** Wenn Sie mit einer Textverarbeitung arbeiten, zeigt Ihnen der Cursor, an welcher Stelle das nächste Zeichen, das auf der Tastatur eingegeben wird, erscheint. Die Cursorposition ändern Sie mit den Cursorsteuerungstasten. Bei der Cursorsteuerung einer Textverarbeitung wird der Cursor relativ zur letzten Position bewegt. Eine absolute Cursorsteuerung liegt vor, wenn der Cursor zu einer bestimmten Spalte und Zeile direkt »springen« kann.

Ein Laserdrucker kennt beide Arten der Positionierung. Der Cursor kann entweder auf ein Druckzeichen oder auf 1/300stel Zoll genau positioniert werden, letzteres ist genauer.

Aus diesem Grunde finden Sie immer zwei Befehle, wenn es um die Positionierung des Cursors geht. Zusätzlich gibt es Befehle zur Zehntelpunkt-genauen Positionierung.

Ein Zehntelpunkt entspricht 1/720stel Zoll, so daß es den Anschein hat, daß die Positionierung genauer ist. Aber die Werte werden auf 1/300stel Zoll umgerechnet, also bedeutet diese Vorgehensweise keinen Vorteil.

Die Cursorposition läßt sich mit zwei Sequenzen definieren. Die Sequenzen »ESC&a#R« (# = Zeilennummer) und »ESC&p#Y« (# = Anzahl der Punkte) definieren die vertikale Position und »ESC&a#C« und »ESC&p#X« (# = Anzahl der Punkte) die horizontale. Wenn ein absoluter Parameter (ohne Vorzeichen) angegeben wird,

Byte	Bedeutung
0	hat immer den Wert 4
1	hat immer den Wert 0
2	hat immer den Wert 14
3	hat immer den Wert 1
4	Ausrichtung: (0 = Hochformat, 1 = Querformat)
5	hat immer den Wert 0
6/7	linker Abstand (-128 bis + 128) zum Zeichenreferenzpunkt; Byte 06: Vorzeichen (0 = positiv, 1 = negativ), Byte 07: Betrag des Abstandes (0 bis 128)
8/9	oberer Abstand zum Zeichenreferenzpunkt (siehe linker Abstand zum Referenzpunkt)
10/11	Zeichenbreite bzw. Zeichenhöhe (0 bis 128) bei Querformatschrift in Druckpunkten.
12/13	Zeichenhöhe (siehe 10/11 Zeichenbreite)
14/15	Delta X: Nach dem Druck eines Zeichens wird der neue Referenzpunkt um diese Strecke weiterbefördert. Der Wert von Delta X ist dabei gleich der Anzahl der Druckpunkte dieser Strecke, multipliziert mit 4. Anschließend folgt die Zeichendefinition. Dabei wird in der linken oberen Ecke des Zeichenrasters begonnen. Bei Schriften im Querformat ist der Anfang jedoch an der oberen rechten Ecke.

Tabelle 2. Die Eckdaten eines Zeichens bei der Übermittlung mit »LPRINT CHR\$(Byte)«-Anweisungen

erfolgt eine absolute Cursorpositionierung. Hat der Parameter ein negatives Vorzeichen, wird der Cursor nach links bzw. nach oben bewegt. Bei einem positiven Vorzeichen verschiebt sich der Cursor entsprechend nach rechts bzw. nach unten.

Besondere Effekte durch Cursorsteuerung

Manchmal ist es wünschenswert, die Cursorposition zu speichern, um zu einem späteren Zeitpunkt an diese ursprüngliche Position zurückzuspringen. Mit der Sequenz »ESC&f0S« speichern Sie die aktuelle Position. Es können bis zu 20 Positionen fixiert werden, die anschließend in umgekehrter Reihenfolge, also nach dem »first in last out«-Schema, angefahren werden.

Die Position, die zuletzt eingegeben wurde, wird mit der Sequenz »ESC&f1S« als erstes angesteuert. Besondere Druckeffekte erzielen Sie, wenn Sie einen Text mehrmals ausdrucken und dabei den Cursor leicht versetzen (Listing 1).

Zum Thema Cursorpositionierung gehören auch die Zeichen- und Zeilenabstände.

Diese Einstellungen definieren eine automatische, ereignisabhängige Cursorpositionierung, die durch den Druck eines Zeichens oder durch einen Zeilenvorschub ausgelöst wird. Die Anzahl der Zeilen pro Zoll legt der Befehl »ESC&l#D« fest. Der Zeilenabstand kann aber auch auf direkte Weise mit »ESC&#lC« in Schritten von #/48 Zoll definiert werden (Listing 2). Mit »ESC&k#H« legen Sie den Zeichenabstand fest.

● **Füllmuster:** Laserdrucker können rechteckige Flächen mit verschiedenen Inhalten füllen. Häufig werden Füllmuster benutzt, um Überschriften

zu hinterlegen. Um eine rechteckige Fläche auszufüllen, positionieren Sie zunächst den Cursor auf die linke obere Ecke und legen anschließend mit zwei Sequenzen die horizontale und vertikale Größe fest. Mit der Escape-Sequenz »ESC*c#G« können Sie sowohl ein vorgegebenes Muster (Parameter 1 bis 6) auswählen als auch den prozentualen Schwärzungsgrad (Parameter 1 bis 100) einer grauen Füllung bestimmen.

In einem zweiten Schritt entscheiden Sie sich für die Art der Füllung. Soll das Rechteck mit dem ausgewählten Muster ausgefüllt werden, wird dies mit »ESC*c3P« erreicht, die Grautönung mit »ESC*c2P« (Listing 3).

● **Linien:** Unter der Voraussetzung, daß eine Linie nichts anderes ist als ein Rechteck mit einer extrem geringen Zeilenbreite, können Sie horizontale und vertikale Linien erzeugen, indem Sie einfach Rechtecke kleiner Zeilenbreite ausfüllen (Listing 4). Durch die integrierte Plottersprache HP-GL/2 bei der Version PCL5 ist es erlaubt, Linien direkt und mit beliebigen Winkeln zu zeichnen. Listing 5 zeigt die Einstellung der Druckrichtung mit PCL5-Sequenzen.

● **Rastergrafik:** Der Laserdrucker speichert jede Druckseite zunächst als Sammlung von gesetzten und nicht gesetzten Punkten. Mit der Rastergrafik beeinflussen Sie diese Punkte direkt. Wenn Sie sich im Zusammenhang mit Nadeldruckern bereits mit Rastergrafik beschäftigt haben, werden Sie feststellen, daß die Programmierung von Rastergrafik bei Laserdruckern wesentlich einfacher ist als bei Nadeldruckern.

Die Übertragung einer Rastergrafik erfolgt zeilenweise von oben nach unten. Die Punkte jeder einzelnen Zeile werden in Gruppen zu je 8 Punkten unterteilt, die zu einem Byte umge-

rechnet werden. Sollte die Zahl der Punkte einer Zeile nicht durch 8 teilbar sein, so wird die Anzahl der Punktgruppen aufgerundet und die so entstehenden restlichen Punkte als nicht gesetzt behandelt. Zur Übermittlung einer Rastergrafik wählen Sie zunächst die gewünschte Auflösung mit »ESC*t#R«. Erlaubt sind die Auflösungen 75, 100, 150 und 300 Punkte pro Zoll. Eine hohe Auflösung hat zur Folge, daß der Laserdrucker sehr viel Arbeitsspeicher benötigt. Wenn dieser nicht vorhanden ist, wird bei gefülltem Speicher ein Seitenvorschub ausgelöst und auf der nächsten Seite weitergedruckt.

Bevor der Druckvorgang gestartet wird, legen Sie fest, ob die Grafik am linken Druckrand (»ESC*r1A«) der aktuellen Cursorzeile oder an der aktuellen Cursorposition (»ESC*»r0A«) erfolgen soll.

Nun werden die Grafikdaten zeilen- und byteweise übermittelt. Jede Zeile wird mit »ESC*b#W« eingeleitet. Dabei gibt der Parameter an, wie viele Bytes für die Zeile übermittelt werden. Anschließend werden alle Bytes der Zeile übertragen. Dazu ist es notwendig, die numerischen Werte mit Hilfe der Funktion »CHR\$(Byte)« in den Lprint-Befehl einzubinden. Sind alle Grafikzeilen an den Laserdrucker gesandt worden, wird mit »ESC*rB« der Grafikausdruck beendet. Nun arbeiten Sie wie gewohnt im Textmodus weiter (Listing 6).

Eine Schrift oder ein Font ist eine Sammlung von Grafiken, bei der jedem druckbaren Zeichen eine entsprechende Grafik zugeordnet wird. Verschiedene Schriften unterscheiden sich durch die Merkmale:

- Ausrichtung,
- Zeichensatz,
- Zeichenabstandsart,
- Zeichendichte,
- Schriftgröße/Schriftstil,
- Schriftstärke,
- Schriftart.

Die Begriffe Schrift und Schriftart werden häufig verwechselt. Die Schriftart beschreibt optische Charakterzüge wie Häkchen, Serifen und Bogen, die einer Schrift ihr typisches Erscheinungsbild verleihen. Der Begriff Schrift hingegen bezeichnet die Summe der oben aufgeführten Merkmale.

● **Schriften wählen:** Ein Laserdrucker ist grundsätzlich nicht in der Lage, eine Schrift in verschiedenen Variationen zu drucken. Deshalb ist es notwendig, für jede gewünschte Variante eine eigene Schrift bereitzuhalten. Doch zeigen die neuen Modelle der Laserdrucker-Hersteller, daß sich diese Eigenart ändert. Die PCL5-Drucker sind bei allen Schriften – also auch bei den nicht skalierbaren Bitmap-

Fonts – in der Lage, die Ausrichtung selbständig umzurechnen. Das Schriftmerkmal Ausrichtung beschreibt, ob die Schrift im Hoch- oder Querformat gedruckt werden soll.

Unter dem Merkmal Zeichensatz versteht man die Sammlung der Buchstabenbilder und das Schema der Codierung, nach dem die einzelnen Buchstabenbilder geordnet sind. Ein spanischer Zeichensatz ist anders aufgebaut als ein deutscher. Bei »Roman-8« sind die Zeichen nicht so codiert wie beim Zeichensatz »PC-8« (IBM SET 2). Da die meisten Laserdrucker werkseitig auf »Roman-8« eingestellt sind, die Personalcomputer jedoch den »PC-8«-Zeichensatz eingebaut haben, kommt es immer wieder zu dem Problem, daß der Laserdrucker beispielsweise anstelle von deutschen Umlauten scheinbar beliebige Zeichen ausgibt. Abhilfe schafft das Listing 7, das Ihren Drucker auf den »PC-8«-Zeichensatz einstellt – Sie können dies aber auch mit Hilfe des Setup-Menüs.

Von Schriften und Schriftarten

Die Zeichenabstandsart gibt an, ob der Abstand zwischen den Zeichen konstant (fest) oder variabel (proportional) ist. Bei einem festen Zeichenabstand haben alle Zeichen die gleiche Breite. Wenn Sie aber in einem Buch oder einer Zeitschrift die einzelnen Buchstabenbreiten vergleichen, werden Sie feststellen, daß für »i« eine geringere Breite veranschlagt wird als für »m«. In diesem Fall ist der Zeichenabstand variabel.

Mit der Schriftgröße wird die Höhe der einzelnen Zeichen in Punkt festgelegt. Die Einheit Punkt kommt aus der Schriftsetzung konventioneller Druckereien und hat nichts mit den Punkten zu tun, aus denen ein Laserdrucker eine Druckseite zusammensetzt. Ein Punkt entspricht 1/72stel Zoll.

Das Schriftmerkmal »Schriftstil« gibt an, ob entweder ein normaler oder ein kursiver (nach rechts schräggestellter) Zeichensatz genutzt wird. Weitere Stilarten sind in PCL5 implementiert.

Die Schriftstärke oder Druckintensität besagt, welchen Schwärzungsgrad eine Schrift hat. Erlaubt sind Werte von -7 bis +7. Dabei gilt -3 als hell, 0 als normal und +3 als fett.

Anhand dieser acht Merkmale wählen Sie eine Schrift aus. Dazu bestimmen Sie die Eigenschaften in der oben aufgeführten Reihenfolge, in der die Prioritäten festgelegt sind. Zwar ist es auch zulässig, eine Teilbeschreibung vorzunehmen, diese muß sich jedoch der vorherigen Einstellung unterord-

nen. Wenn ein Laserdrucker derzeit die Lineprinter-Schrift aktiviert hat und er nicht in der Lage ist, diese Schriftart »fett« zu drucken, können Sie nicht einfach auf Fettdruck umschalten. Tun Sie das trotzdem, wird dies ignoriert. Es ist wichtig bei der Programmierung, das Prioritätensystem genau einzuhalten. Hier werden die meisten Fehler bei der Wahl einer Schrift gemacht. Besonders häufig ist der Fehler, die Schriftart als erstes zu wählen, was häufig zur Folge hat, daß diese Selektion ignoriert wird, da die Schriftart die niedrigste Priorität hat.

● **Schrift generieren:** Das Erzeugen einer eigenen Schrift ist eine aufwendige Angelegenheit. Aus diesem Grunde ist es ratsam, dies mit einem Zeichengenerator zu tun. Ein guter Zeichengenerator ist das Shareware-Produkt »QFont«.

Um einen eigenen Font zu erzeugen, sind fünf Arbeitsschritte nötig. Zunächst wählen Sie die Kennung (Font-ID) mit »ESC*c#D« (# = 0 bis 32767). Anschließend wird der Font-Header (Tabelle 1), der die wesentlichen Merkmale einer Schrift beinhaltet, mit »ESC)s#W« übertragen. Für den Parameter »#« ist die Anzahl der Bytes des Font-Headers zu wählen – im Normalfall sind 26 Byte erforderlich. Anschließend werden die Daten des Font-Headers übermittelt.

Übertragung der neuen Schriften

Nachdem die Vorbereitungen zum Generieren der neuen Schrift getroffen wurden, werden die einzelnen Zeichen der neuen Schrift übertragen. Dazu wählen Sie mit »ESC*c#E« den Code des zu definierenden Zeichens aus, und anschließend werden seine Daten übertragen. Dabei legen Sie mit »ESC)s#W« die Anzahl der dazu erforderlichen Bytes fest. Nun können die einzelnen Bytes mit »LPRINT CHR\$(Byte)«-Anweisungen übermittelt werden. Dabei enthalten die ersten 16 Byte die Eckdaten des Zeichens (Tabelle 2).

Nun folgt das eigentliche Muster. Muster sind einzelne Rastergrafiken, die dem zu definierenden Zeichen zugeordnet werden. Diese Prozedur wird solange wiederholt, bis alle Zeichen eines Zeichensatzes übermittelt worden sind. Abschließend wird der Status der Schrift festgelegt. Soll diese mit einem permanenten Status versehen werden, wird der Befehl »ESC*c4F« zum Drucker gesendet; mit »ESC*c5F« wird der Schrift ein temporärer Status zugewiesen.

(Ulrich Borgdorf/st)


```
[DG3] 'PROGRAMM: CURSOR.BAS
[TR5] 'FUNKTION: DRUCKERCURSOR ändern
[2E3] 'SPRACHE: QUICK BASIC
[M24] 'AUTOR: ULRICH BORG DORF
[1W4] '(C) 1992 DMV WIDUCH GMBH & CO. KG
[NA3] 100 *** CURSOR.BAS ***
[ZV4] 110 ESC$=CHR$(27):LRESET$=ESC$+"E"
[XR5] 120 MERKE$=ESC$+"&f0$":HOLE$=ESC$+"&f1$"
[3K4] 130 TEXT$="DOS International"
[F36] 140 CLS:PRINT "Cursor-Demo für ";TEXT$
[Q25] 150 OPEN "LPT1:" FOR OUTPUT AS #1
[BN4] 160 PRINT #1, LRESET$;PRINT #1,
[1L0] 190 :
[RN6] 200 PRINT #1, ESC$;"*p300xp300Y";MERKE$;TEXT$
[4B6] 210 PRINT #1, HOLE$;ESC$;"*p+0xp+0Y";TEXT$;
[3L0] 290 :
[3P6] 300 PRINT #1, ESC$;"*p300xp350Y";MERKE$;TEXT$
[MZ6] 310 PRINT #1, HOLE$;ESC$;"*p+1xp+0Y";TEXT$;" x+1"
[5L0] 390 :
[WN6] 400 PRINT #1, ESC$;"*p300xp400Y";MERKE$;TEXT$
[UZ6] 410 PRINT #1, HOLE$;ESC$;"*p+2xp+0Y";TEXT$;" x+2"
[7L0] 490 :
[8P6] 500 PRINT #1, ESC$;"*p300xp450Y";MERKE$;TEXT$
[VZ6] 510 PRINT #1, HOLE$;ESC$;"*p+0xp+1Y";TEXT$;" y+1"
[9L0] 590 :
[1P6] 600 PRINT #1, ESC$;"*p300xp500Y";MERKE$;TEXT$
[107] 610 PRINT #1, HOLE$;ESC$;"*p+0xp+2Y";TEXT$;" y+2"
[BL0] 690 :
[DP6] 700 PRINT #1, ESC$;"*p300xp550Y";MERKE$;TEXT$
[7H7] 710 PRINT #1, HOLE$;ESC$;"*p+1xp+1Y";TEXT$;" x+1
y+1"
[DL0] 790 :
[6P6] 800 PRINT #1, ESC$;"*p300xp600Y";MERKE$;TEXT$
[HH7] 810 PRINT #1, HOLE$;ESC$;"*p+2xp+2Y";TEXT$;" x+2
y+2"
[FL0] 890 :
[KP6] 900 PRINT #1, ESC$;"*p300xp650Y";MERKE$;TEXT$
[VH7] 910 PRINT #1, HOLE$;ESC$;"*p+3xp+3Y";TEXT$;" x+3
y+3"
[BL0] 970 :
[M13] 980 PRINT #1, LRESET$
[0C1] 990 CLOSE
```

Listing 1. »cursor.bas« demonstriert Druckeffekte durch Versetzen des Cursors.

```
[YM3] 'PROGRAMM: ABSTAND.BAS
[4M5] 'FUNKTION: ZEILENABSTAND ÄNDERN
[2E3] 'SPRACHE: QUICK BASIC
[M24] 'AUTOR: ULRICH BORG DORF
[1W4] '(C) 1992 DMV WIDUCH GMBH & CO. KG
[KZ3] 100 REM *** ABSTAND.BAS ***
[3D5] 110 ESC$ = CHR$(27): LRESET$ = ESC$ + "E"
[TZ4] 120 TEXT$ = "DOS International"
[M41] 140 CLS
[6W5] 150 PRINT "Abstand-Demo für "; TEXT$
[C25] 190 OPEN "LPT1:" FOR OUTPUT AS #1
[KB6] 200 PRINT #1, "****Zeilenabstand >>>&l#C<<<";
[WR1] 210 PRINT #1,
[PN2] 220 FOR I = 0 TO 20
[SL6] 230 PRINT #1, ESC$; "&l"; STR$(I); "C"; TEXT$
[UG1] 240 NEXT I
[H63] 260 PRINT #1, LRESET$;
[KE6] 300 PRINT #1, "****Zeichenabstand >>>&k#H<<<";
[YR1] 310 PRINT #1,
[SN2] 320 FOR I = 0 TO 20
[CM6] 330 PRINT #1, ESC$; "&k"; STR$(I); "H"; TEXT$
[WG1] 340 NEXT I
[Z03] 350 PRINT #1, LRESET$
[FP6] 400 PRINT #1, "****Zeilen pro Zoll >>>&l#D<<<";
[0S1] 410 PRINT #1,
[UN2] 420 FOR I = 0 TO 20
[0M6] 430 PRINT #1, ESC$; "&l"; STR$(I); "D"; TEXT$
[YG1] 440 NEXT I
[VK0] 450 :
[N63] 460 PRINT #1, LRESET$;
[PB1] 490 CLOSE
```

Listing 2. Wie der Zeilen- und Zeichenabstand verändert wird, zeigt »abstand.bas«.

```
[693] 'PROGRAMM: FUELL.BAS
[4U6] 'FUNKTION: RECHTECKE MIT MUSTER FUELLEN
[2E3] 'SPRACHE: QUICK BASIC
[M24] 'AUTOR: ULRICH BORG DORF
[1W4] '(C) 1992 DMV WIDUCH GMBH & CO. KG
[G13] 100 *** FUELL.BAS ***
[ZV4] 110 ESC$=CHR$(27):LRESET$=ESC$+"E"
[R15] 120 OPEN "LPT1:" FOR OUTPUT AS #1
```

```
[FK0] 130 :
[226] 140 '==== Setze linken Rand =====
[G92] 210 FOR I=1 TO 6
[XH5] 220 '----- Setze Cursor -----
[A06] 230 PRINT#1,ESC$;"*p400x*p";(I-1)*400;"Y";
[DW6] 240 '----- Horizontale Größe 200 Punkte --
[0U3] 270 PRINT#1,ESC$;"*c200A";
[L47] 280 '----- Vertikale Größe 200 Punkte ---
[8U3] 290 PRINT#1,ESC$;"*c200B";
[1S5] 300 '----- Füllmuster #I -----
[CY4] 310 PRINT#1,ESC$;"*c";STR$(I);"G";
[P17] 320 '--- Zeichne eingestelltes Füllmuster -
[DK3] 330 PRINT#1,ESC$;"*c3P";
[0E5] 340 '----- Beschrifte -----
[PX4] 350 PRINT#1,ESC$;"*p+300x*p+100Y";
[NH5] 360 PRINT #1,"Füllmuster #";STR$(I)
[BH1] 390 NEXT I
[GP6] 400 '==== Mache noch ein schwarzes Muster=
[LG5] 410 PRINT#1,ESC$;"*p400x*p";(7-1)*400;"Y";
[GD5] 420 PRINT#1,ESC$;"*c200A";ESC$;"*c200B";
[UB3] 440 PRINT#1,ESC$;"*c0P";
[FR4] 450 PRINT#1,ESC$;"*p+300x*p+100Y";
[DD5] 460 PRINT#1,"vollschwarz (solid)";
[UK0] 540 :
[Y63] 550 PRINT #1,LRESET$;
[FB1] 560 CLOSE
```

Listing 3. Mit »fuell.bas« werden Rechtecke bunter.

```
[W83] 'PROGRAMM: LINIE.BAS
[FR4] 'FUNKTION: LINIEN AUSDRUCKEN
[2E3] 'SPRACHE: QUICK BASIC
[M24] 'AUTOR: ULRICH BORG DORF
[1W4] '(C) 1992 DMV WIDUCH GMBH & CO. KG
[CM3] 100 rem *** LINIEN.BAS ***
[EA2] 110 ESC$=CHR$(27)
[E41] 120 CLS
[845] 130 PRINT " Start: 100,100 L= 8, B=400"
[PZ4] 140 PRINT " Start: 50,150 L=400, B=4"
[HC1] 150 PRINT
[925] 180 OPEN "LPT1:" FOR OUTPUT AS #1
[826] 200 PRINT "Startpunkt (X,Y) (0,0=Ende) :";
[E12] 210 INPUT X,Y
[0D4] 220 if X=0 AND Y=0 then goto 500
[SZ4] 230 INPUT "Länge der Linie L=";L
[RW4] 240 input "Breite der Linie B=";B
[RK0] 250 :
[D25] 260 PRINT #1, ESC$;"*p";X;"x";Y;"Y";
[896] 270 PRINT #1, ESC$;"*c";L;"A";ESC$;"*c";B;"B";
[0L3] 280 PRINT #1, ESC$;"*c0P";
[PN1] 300 goto 200
[CK0] 400 :
[9E6] 500 PRINT #1, CHR$(12);:REM Seitenvorschub
[5B1] 900 CLOSE
```

Listing 4. Wie Linien ausgedruckt werden, zeigt »linie.bas«.

```
[163] 'PROGRAMM: PCL5.BAS
[4B5] 'FUNKTION: DEMO VON PCL5/HP-GL/2
[2E3] 'SPRACHE: QUICK BASIC
[M24] 'AUTOR: ULRICH BORG DORF
[1W4] '(C) 1992 DMV WIDUCH GMBH & CO. KG
[HN2] 100 ' *** PCL5.BAS ***
[8W4] 140 ESC$=CHR$(27):LRESET$=ESC$+"E"
[325] 150 OPEN "LPT2:" FOR OUTPUT AS #1
[WL2] 170 TEXT$=" DOS "
[YK0] 180 :
[7H6] 190 '*** Einstellung der Druckrichtung ***
[EA3] 200 PRINT #1,LRESET$;
[LK4] 210 PRINT #1,ESC$;"&a0P";TEXT$;
[2K4] 220 PRINT #1,ESC$;"&a90P";TEXT$;
[ER4] 230 PRINT #1,ESC$;"&a180P";TEXT$;
[GR4] 240 PRINT #1,ESC$;"&a270P";TEXT$;
[0L0] 280 :
[515] 290 '*** HP-GL/2-Demonstration ***
[B63] 300 PRINT #1,LRESET$;
[B73] 310 PRINT #1,ESC$;"%0B";
[EE2] 320 PRINT #1,"IN";
[EL3] 330 PRINT #1,"SP1PA50,50";
[SV4] 340 PRINT #1,"PD2500,50,50,1500,50,50";
[8G2] 350 PRINT #1,"%0A";
[X63] 360 PRINT #1,LRESET$;
[5L0] 390 :
[PB1] 490 CLOSE
```

Listing 5. »pcl5.bas« zeigt, wie Sie mit der Plottersprache HP-GL/2 zeichnen und drucken können.


```
[RS2] 'PROGRAMM: GRAFIK
[7P4] 'FUNKTION: GRAFIK AUSDRUCKEN
[2E3] 'SPRACHE: QUICK BASIC
[M24] 'AUTOR: ULRICH BORG DORF
[1W4] '(C) 1992 DMV WIDUCH GMBH & CO. KG
[W73] 100 '*** GRAFIK.BAS ***
[EA2] 110 ESC$=CHR$(27)
[9K3] 130 CLS:WIDTH "LPT2:",255
[RA2] 140 ESC$=CHR$(27)
[025] 150 OPEN "LPT1:" FOR OUTPUT AS #1
[CW1] 200 GRAFIK$=""
[RC2] 210 FOR I=1 TO 16
[8K4] 220 GRAFIK$=GRAFIK$+ESC$+"*b2W"
[383] 230 READ BYTE1,BYTE2
[YF6] 240 BYTE=BYTE1:GOSUB 1000:PRINT MUSTER$;
[2G6] 250 BYTE=BYTE2:GOSUB 1000:PRINT MUSTER$;
[D53] 260 PRINT SPACES(3);
[YA4] 270 PRINT USING "###";BYTE1;
[ES4] 280 PRINT " ";USING "###";BYTE2
[BT4] 300 GRAFIK$=GRAFIK$+CHR$(BYTE1)
[FT4] 310 GRAFIK$=GRAFIK$+CHR$(BYTE2)
[PG1] 320 NEXT I
[5L0] 390 :
[E03] 400 PRINT #1, "75 dpi:"
[0U4] 405 PRINT #1, ESC$;"*t75R";GRAFIK$
[K23] 410 PRINT #1, "100 dpi:"
[905] 415 PRINT #1, ESC$;"*t100R";GRAFIK$
[Z23] 420 PRINT #1, "150 dpi:"
[H05] 425 PRINT #1, ESC$;"*t150R";GRAFIK$
[U23] 430 PRINT #1, "300 dpi:"
[R05] 435 PRINT #1, ESC$;"*t300R";GRAFIK$
[SR6] 440 PRINT #1, CHR$(12);:REM (Seitenvorschub)
[7L0] 490 :
[P54] 500 DATA 96,12,96,28,96,60,96,108
[DF4] 510 DATA 96,204,97,140,99,12,102,12
[2E4] 520 DATA 108,12,120,12,112,30,96,63
```

```
[RK3] 530 DATA 64,30,0,30,0,12,0,12
[BL0] 970 :
[N02] 980 CLOSE:END
[HL0] 990 :
[NW3] 999 '*** Unterprogramm ***
[V72] 1000 MUSTER$=""
[GK3] 1010 FOR J=7 TO 0 STEP -1
[GEB] 1020 IF BYTE=>(2^J) THEN MUSTER$=MUSTER$+CHR$(2
19) ELSE MUSTER$=MUSTER$+" "
[Z46] 1030 IF BYTE=>(2^J) THEN BYTE=BYTE-(2^J)
[WN1] 1050 NEXT J
[WV1] 1090 RETURN
```

Listing 6. Grafiken ausdrucken können Sie mit »grafik.bas«.

```
[5V2] 'PROGRAMM: PC8.BAS
[C86] 'FUNKTION: DRUCKER EINSTELLUNG ÄNDERN
[2E3] 'SPRACHE: QUICK BASIC
[M24] 'AUTOR: ULRICH BORG DORF
[1W4] '(C) 1992 DMV WIDUCH GMBH & CO. KG
[R63] 100 REM *** PC8.BAS ***
[UA2] 150 ESC$=CHR$(27)
[P15] 200 OPEN "LPT2:" FOR OUTPUT AS #1
[1F3] 210 PRINT #1,ESC$;"(10U";
[CP7] 250 PRINT #1,"PC-8 Zeichensatz äöüÄÖÜ ß";
[Z22] 260 PRINT #1,CHR$(12);
[KB1] 290 CLOSE
```

Listing 7. »pc8.bas« stellt Ihren Drucker auf den PC8-Zeichensatz um.

Impressum

Herausgeber: Michael Scharfenberger
Redaktionsdirektor: Volker Everts

Redaktion

Chefredakteur: Ulrich Eike (ue)
Chefin vom Dienst: Gabi Börger (gb)
Leitender Redakteur: Wolfgang Hofner (wh)
Redaktion: Burkhard Müller (bm), Michael Starke (st)
Erhard Thomas (et), Kerrin Uhlmann (uk)
Redaktionsmarketing: Ingo Everts (ie)
Textredaktion: Annette Baumhof (ab), Anne Brückner (br),
Vera Eich (ve), Dr. Hans-Dietrich Pfeil (pf)
Redaktionsassistent: Anke Ludwig, Helga Menne, Lydia Roitzheim
Gestaltung und Layout: Christiana Seiser, Michaela Küppers
Grafik: Leonhard Büttner
Fotografie: Heinz Harcuba
Leitung des Redaktionsbüros: Volker Everts
Anschrift Redaktion: Redaktion DOS International,
Gruberstr. 46c, 8011 Poing bei München,
Telefon: (081 21) 769-0,
Telefax: (081 21) 790 46

Sie erreichen die Redaktion werktags telefonisch
von 15 bis 17 Uhr.

Manuskripte, Programme bitte an die Redaktion DOS International, Gruberstr. 46c, 8011 Poing bei München, senden.
Für unverlangt eingesandte Manuskripte und Datenträger sowie Fotos übernehmen 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. Für veröffentlichte Programme und Basteile übernehmen wir weder Gewähr noch Haftung. Schaltungen und verwendete Bezeichnungen müssen nicht frei von gewerblichen Schutzrechten sein. Namentlich gekennzeichnete Fremdbeiträge geben nicht in jedem Fall die Meinung der Redaktion wieder.

Anzeigenverkauf

Anzeigenverkaufsleitung: Thomas Goldmann
Anzeigenverkauf für PLZ 1, 4, 5:
Karina Ehrlich (0 56 51) 80 93 71, Telefax (0 56 51) 80 94 44
Bernd Heckmann (0 56 51) 80 93 81,
Gerlinde Rachow (0 56 51) 80 93 90,
Sylvia Stephani (0 56 51) 80 93 80,
Anzeigenverkauf für PLZ 2 + 3
DMV-Verlagsbüro Hamburg, Kunastr. 4a, 2000 Hamburg 65,
Telefon: (0 40) 60 03 075-77, Telefax: (0 40) 60 15 475
Leitung: Sylvia Ehrenpfordt
Anzeigenverkauf: Sabine Bindeil
Coordinating Manager Asia: Ralph Streng
Coordinating Manager USA: Tobias J. Hoffmann
Anzeigenverkauf für PLZ 6 - 8
DMV-Verlagsbüro München, Zaunkönigweg 2c,
8000 München 82,
Telefon: (0 89) 4 39 10 87, Telefax: (0 89) 4 39 10 80
Leitung: Britta Fiebig
Anzeigenverkauf: Peter Schätzle, Hannelore David, Ilona Sehm
Repräsentanten im Ausland:
Great Britain: Huson European Media, Gerry Rhoades-Brown,
10-11 The green Business Centre, The Causeway, Staines, Middlesex, TW18 3AL, Telephone No: 0784 46 99 00 Fax No: 0784 46 99 96
Taiwan: PRISCO, Mr. Parson Lee, 2F, No.17, Alley 7, Lane 217, Chung Hsiao E. Rd. Sec. 3, Taipei, Taiwan, R.O.C., Phone: [886]-2-711 48 33-6, Fax: [886]-2-741 51 10,
Hong Kong: ELECTRONIC (HK) PUBLISHING CO: Mr. Herman Lau, 1301 Po Sang Bank Building, 33 Argyle Street, Kin., Hong Kong. Tel.: 391 2977, 391 5088, FAX: 852 789 8027, Cable: ELPUB
Singapore: Major Media Private Ltd., Mr. Hoo Siew-Sai, 6th Fl., 52 Chin Swee Road, Singapore 0316. Phone: (65)-738 0122, Fax: (65)-738 2108
Anzeigenverwaltung: Andrea Giese, Beate Kranz
Druckunterlagen-Disposition
Christina Wabra (0 56 51) 809 - 3 72
Anzeigenpreise
Es gilt die Anzeigenpreisliste Nr. 7 vom 01.12.1991

Verlag

Produktionsleitung: Uwe Siebert
Koordination: Petra Hugo
Fotosatz: Reinhilde Schwarz
Montage/Reprografie: Margarete Schenk (Ltg.),
Helmut Skoupy (Ltg.), Manuela Eska, Monika Martin
Art director: Mohamed Hawa
Werbegratik: Uwe Kalischinsky, Norbert Risch
Anschrift Verlag: DMV Daten- und Medienverlag Widuch GmbH & Co. KG, Fuldaer Str. 6, 3440 Eschwege,
Telefon: (0 56 51) 8 09-0, Telefax: (0 56 51) 8 09-333
Geschäftsführung: Helmuth Schmitz, Michael Scharfenberger
Vertrieb: Verlagsunion Erich Pabel-Arthur Moewig KG (VPM)
Friedrich-Bergius-Straße 20, 6200 Wiesbaden
Druck: Druckerei Jungfer, 3420 Herzberg
Bezugspreise
DOS Extra erscheint dreimonatlich. Einzelpreis DM 18,- / sfr 18,- / ös 145,- / Lit 15.900
Abonnementpreise
Die Preise verstehen sich grundsätzlich einschließlich Porto und Verpackung.
Inland:
Bezugspreis 4 Ausgaben: DM 60,-
Europäisches Ausland:
Bezugspreis 4 Ausgaben: DM 70,-
Außereuropäisches Ausland:
Bezugspreis 4 Ausgaben: DM 80,-
Bankverbindungen:
Postcheck Frankfurt/M: Kto.-Nr.: 23043-608
Raiffeisenbank Eschwege: BLZ: 522 603 85, Kto.-Nr.: 245 7008
Die Abonnementbestellung kann innerhalb einer Woche nach Auftrag beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich widerrufen werden. Zur Wahrung der Frist reicht der Poststempel. Das Abonnement verlängert sich automatisch um 6 bzw. 12 Monate, wenn es nicht mindestens 6 Wochen vor Ablauf beim Verlag schriftlich gekündigt wird.

DOS EXTRA

DOS EXTRA 4

Speicherresidente
Programme
DTP-Programm
EGA-Grafik
MS-DOS intern

DOS EXTRA 5

PC-Grafikkarten
Das BIOS des PC
GW-Basic
Gerätetreiber programmieren

DOS EXTRA 6

Basic im Detail
GW-Basic intern
SBasic-Compiler
Basic-Utilities

DOS EXTRA 7

PC-Komponenten
Sprite-Editor
SBasic Plus
C-Grundlagen

DOS EXTRA 8

Assembler SAM III
Assembler-Werkstatt
Assembler-Grundlagen

DOS EXTRA 10

MS-DOS 4.01
Hardware-Praxis
Snapshot-Programm
Turbo-Pascal-Units

DOS EXTRA 12

Grafik in Basic
Hardcopy-Utility
Fraktalgrafik in Turbo Pascal
Basic für PC-Grafikkarten

DOS EXTRA 13

Die besten Spiele
Abenteuerspiele
Flugsimulator
Sharewarespiele

DOS EXTRA 14

Datensicherungsprogramm
Zeichensatzeditor
Barcode-Darstellung

DOS EXTRA 15

SBasic Version 2.0
Skeybgr Version 4.0
Schiffe versenken
Simulation von Tastatureingaben

DOS EXTRA 16

Batch-Compiler der S-Klasse
Präprozessoren für MASM
und TASM
Assembler-Befehle und
Interrupts
RS-232-Programmierung

DOS EXTRA 17

Transpiler: Basic nach C
Flexibles Kopierprogramm für al-
le Formate
Formatierer der S-Klasse
BGI-Treiber für Super-VGA
Kiloweise Hilfsprogramme für
jeden Zweck

DOS EXTRA 18

universelle Entwickler-
oberfläche
PCX-Bilder in Basic
Pretty Printer für Turbo Pascal
Tips, Tricks und Tools
kiloweise Hilfsprogramme für
jeden Zweck

DOS EXTRA 19

kompletter Raytracer in C
schnelle Sprites mit Pascal
Animation mit Quick Basic
Vektorgrafik in Echtzeit
Superschnelle Fraktale
Tips und Tricks



DOS EXTRA Nr. 20

Aus dem Inhalt:

- Turbo Vision im Grafikmodus
- TP-Unit, um erweiterten Speicher für Turbo-Pascal-Programme zu nutzen
- TP-Unit für Textbearbeitung
- Objekt-Browser
- Gruppen-Manager für Windows

Vorsprung
für
PC-Benutzer
mit
DOS EXTRA
im
Abonnement!

4 Ausgaben bei vierteljährlicher
Erscheinungsweise mit dem
bequemen Zustellerservice per
Post:

BRD DM 60,-
Europa DM 70,-
Außereuropäisches Ausland DM 80,-

Keine
Bestellprobleme
mit dem
Dauerbezug
der
DOS-EXTRA-
DATABOX!

Bequemes Dauerbezugsverfah-
ren mit Bankabbuchung und vier-
teljährlicher Kündigungsmöglich-
keit:

BRD DM 39,-
Europa DM 41,-
Außereuropäisches Ausland DM 42,-

Bitte benutzen Sie nur die Bestellkarte!

DATABOX

Die Software zu den
DOS-EXTRA-Heften
Sagenhaft günstig, extra viel
gute Software fürs Geld!
Bringt alle Programme der
DOS EXTRA auf Datenträger

Den Aufbau der Databox-Diskette finden Sie in der
Datei »read.me« erklärt. Geben Sie deshalb vor dem
Start »type read.me« ein. Sämtliche Pascal- und C-
Programme liegen auch in einer kompilierten Form
für Hercules- und Farbgrafikkarten vor. Es ist aller-
dings nicht gewährleistet, daß diese Versionen auf
Hardware, die nicht vollständig kompatibel zum IBM-
PC ist, problemlos laufen. In diesem Fall müssen Sie
den Quellcode nach der Anpassung mit dem ent-
sprechenden Compiler neu übersetzen.

Die Basic-Programme arbeiten mit dem GW-Basic-
Interpreter von Microsoft einwandfrei. Bei Interpre-
tern und Compilern anderer Hersteller kann es zu
Problemen kommen, sofern diese nicht vollkompati-
bel sind.

jeweils 2 Disketten 5 1/4"
1 Diskette 3 1/2"

35,- DM	
Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:	
Inland:	Ausland:
Einzelpreis 35,- DM	Einzelpreis 35,- DM
zzgl. Versandkosten 4,- DM	zzgl. Versandkosten 6,- DM
Endpreis 39,- DM	Endpreis 41,- DM

Unabhängig von der Anzahl der bestellten Pro-
dukte berechnen wir für das Inland 4,- DM bzw. für
das Ausland 6,- DM Porto und Verpackung.

Bitte benutzen Sie nur die Bestellkarte!

DMV-Verlag • Postfach 250 • 3440 Eschwege



Noch erhältlich:

DOS EXTRA Nr. 4

- Desktop-Publishing-Programm »SDTP«
- Kommandoeditor für MS-DOS
- Editor für Disketten und Festplatten
- Speicherresidente Programme verwalten
- Paßwort-Schutz für MS-DOS
- Tastatureingabe automatisiert
- Clock-Utility für alle Grafikkarten
- Cursor und Zeichensatz für EGA-Karte
- Fenstertechnik für Turbo C

DOS EXTRA Nr. 5

- 16 Farben für die CGA-Farbgrafikkarte
- Druckerzeichensatz selbstgemacht
- Mausgesteuertes Zeichenprogramm
- Editor in GW-Basic
- Setup-Programm für ATs
- Maschinensprache für GW-Basic

DOS EXTRA Nr. 6

- SBasic-Compiler im Quelltext
- Datenbankprogramm »Parat«
- Fensterfunktionen für GW-Basic und SBasic
- Analyse von Grafikkarten
- Maskengenerator für Binärdateien
- dBase-Datenbanken in Basic
- Superhirn für Quick Basic 4.0

DOS EXTRA Nr. 7

- MS-DOS-Benutzeroberfläche Easydos
- Benutzeroberfläche für SBasic
- Sprite-Editor für Farbgrafikkarten
- DOS-Packer
- Virenschutzprogramm
- Weltkarte im Computer
- Einfacher Prozessor-Emulator

DOS EXTRA Nr. 8

- Makroassembler SAM III
- Exe2bin-Ersatz durch »exe2com«
- Speicherverwaltung unter MS-DOS

DOS EXTRA Nr. 10

- Units-Bibliotheken für Turbo Pascal:
- Datum, Sound, Bildschirmausgabe
- Snapshot-Utility, Version 2.7
- TSR-Datei-Manager
- DOS-Assembler SAM III, Version 1.1
- Schnelles Kopierprogramm
- Komfortable Dateiumleitung
- Hercules-Grafik in Basic programmiert
- Patch für BGI-Treiber von Turbo Pascal

DOS EXTRA Nr. 12

- Kreative Grafik in Basic zum Mitmachen
- Grafik-Anpassung von Basic-Programmen
- Außergewöhnliche Fraktalgrafiken
- Das ultimative Hardcopy-Utility
- Testbild für Monitore in Basic programmiert

DOS EXTRA Nr. 14

- Komfortables Datensicherungsprogramm
- Zeichensatzeditor für 24-Nadel-Drucker
- Pascal-Quelltexte optimieren
- Utilities zur Bearbeitung von Dateien und Verzeichnissen

DOS EXTRA Nr. 15

- Der Nachbrenner für GW-Basic:
- SBasic Version 2.0
- Der Tastatortreiber »Skeybgr« in der Version 4.0
- »Schiffe versenken« mit einem Quick-Basic-Programm
- Simulation von Tastatureingaben mit »Prokey«

DOS EXTRA Nr. 16

- Batch-Compiler der S-Klasse
- Config-Editor beim Booten
- Assembler-Preprozessor für MASM und TASM
- GEM-API-Unit für Turbo Pascal
- SCheck-Prüfsummenprogramm in der Version 3.0

DOS EXTRA Nr. 17

- Transpiler: Basic nach C
- Flexibles Kopierprogramm für alle Formate
- Formattierer der S-Klasse
- BGI-Treiber für Super-VGA
- Kiloweise Hilfsprogramme für jeden Zweck

DOS EXTRA Nr. 18

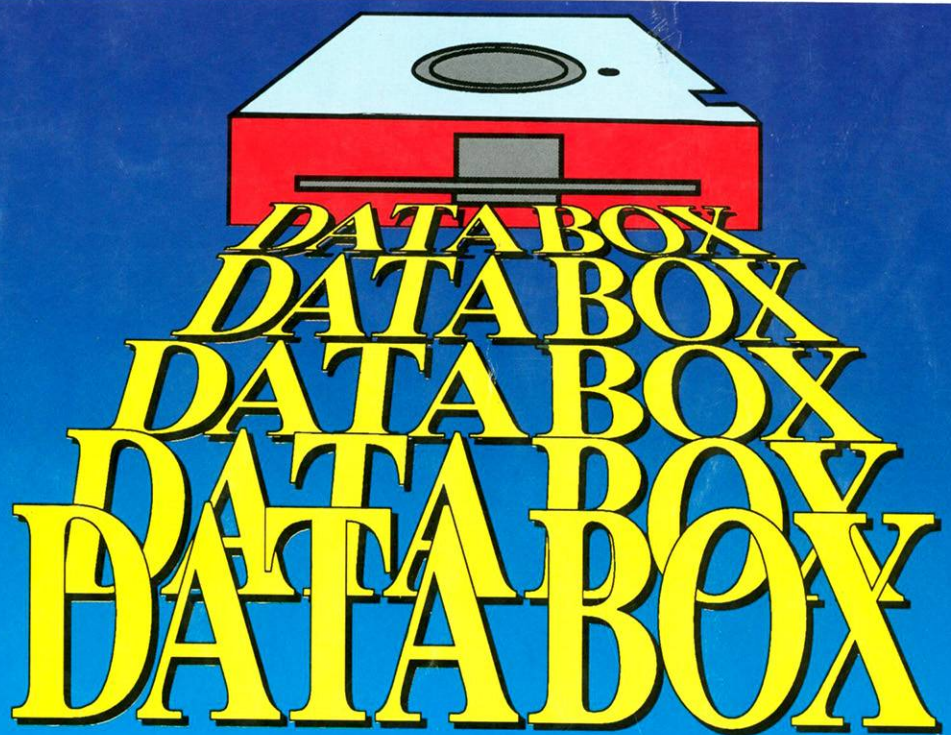
- universelle Entwickleroberfläche
- PCX-Bilder in Basic
- Pretty Printer für Turbo Pascal
- Tips, Tricks und Tools
- kiloweise Hilfsprogramme für jeden Zweck

DOS EXTRA Nr. 19

- komplettes Raytracing-Programm
- Sprite-Programmierung
- Fraktal-Explorer
- Units zur Darstellung von 2D- und 3D-Grafiken
- Tips und Tricks zur Grafik

DOS EXTRA Nr. 20

- Objektorientierte Programmierung: Roboter im Labyrinth
- Programmierung der Soundblasterkarte
- Arbeiten mit dem integrierten Assembler
- BGI-Grafiktreiber intern
- Interrupts in eigenen Programmen nutzen



Die Software zu den DOS-EXTRA-Heften

Sagenhaft günstig, extra viel gute Software fürs Geld!

DATABOX:

- mehr als der übliche Softwareservice
- bringt alle Programme der DOS EXTRA auf Datenträger

Für alle PCs unter MS-DOS

Den Aufbau der Databox-Diskette finden Sie in der Datei »read.me« erklärt. Geben Sie deshalb vor dem Start »type read.me« ein. Sämtliche Pascal- und C-Programme liegen auch in einer kompilierten Form für Hercules- und Farbgrafikkarten vor. Es ist allerdings nicht gewährleistet, daß diese Versionen auf Hardware, die nicht vollständig kompatibel zum IBM PC ist, problemlos laufen. In diesem Fall müssen Sie den Quellcode nach der Anpassung mit dem entsprechenden Compiler neu übersetzen. Die Basic-Programme arbeiten mit dem GW-Basic-Interpreter von Microsoft einwandfrei. Bei Interpretern und Compilern anderer Hersteller kann es zu Problemen bei der Programmausführung kommen, sofern diese nicht voll kompatibel sind.

jeweils 2 Disketten 5 1/4"
1 Diskette 3 1/2"

35,- DM (unverbindliche Preisempfehlung)

Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:

Inland:		Ausland:	
Einzelpreis	35,- DM	Einzelpreis	35,- DM
zzgl. Versandkosten	4,- DM	zzgl. Versandkosten	6,- DM
Endpreis	39,- DM	Endpreis	41,- DM

Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland 4,- DM bzw. für das Ausland 6,- DM Porto und Verpackung.

- Bitte benutzen Sie die Bestellkarte. -



DOS EXTRA Nr. 21

- Programmieren mit der Assemblersprache
- SAM 4.0 - Der DOS-Makroassembler
- Dateiauswahlboxen leicht selbstgemacht
- Geräteprogrammierung intern
- Programmierung von Laserdruckern