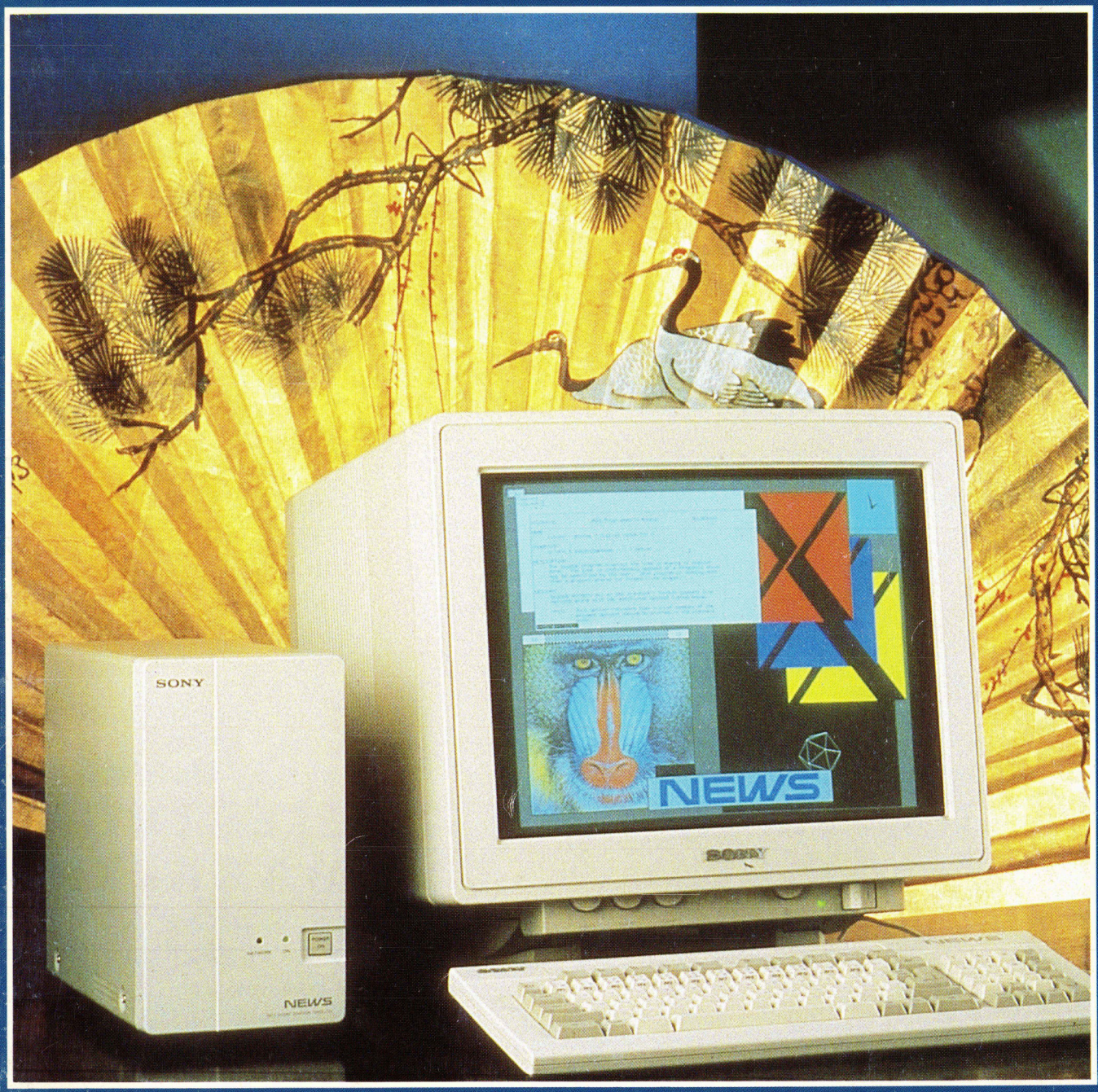
88-5

MIKRO + KLEIN COMPUTER

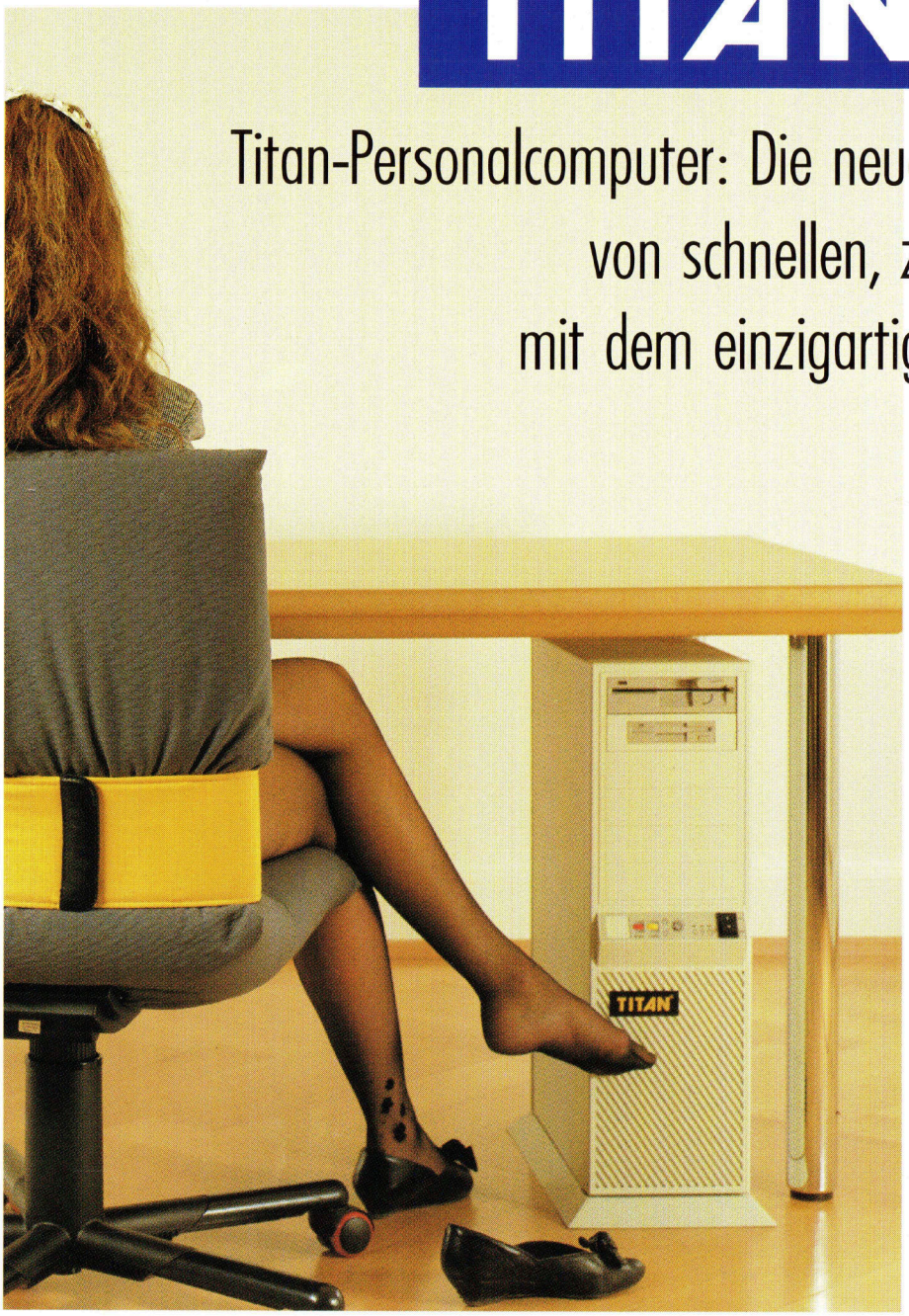
DAS SCHWEIZER FACHMAGAZIN FÜR KLEINE UND MITTLERE COMPUTERSYSTEME Fr. 8.–



UNIX – Konkurrent für OS/2?

Mikroprozessor-Architekturen

TITAN®



Titan-Personalcomputer: Die neue Generation von schnellen, zuverlässigen Systemen mit dem einzigartigen «Swiss Finish».

Titan bringt die neue Personalcomputer-Linie mit 3 Systemen, die für alle Ihre Bedürfnisse eine Lösung bieten: Vom preisgünstigen Einstiegsmodell mit 80286-Mikroprozessor bis zum schnellen 80386-Mehrplatzsystem.

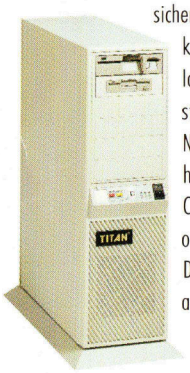
Die Wahl zwischen verschiedenen Bildschirmen für Text und Grafik sowie unterschiedliche interne und externe Speichergrössen erlauben dem Anwender eine optimale Feinabstimmung für seinen Einsatzbereich.

Die Betriebssysteme DOS, Xenix und OS/2, die auf den Titan-Systemen laufen, erschliessen ein immenses Potential an Anwendersoftware. Titan-Personalcomputer sind hundertprozentig kompatibel zum Industriestandard.

Unser einzigartiger «Swiss Finish» garantiert Ihnen die Zuverlässigkeit und Langlebigkeit der Hardware.

Das Kraftpaket:
Titan Tower 386+

Seine technische Leistung und die rasante Arbeitsgeschwindigkeit sind beeindruckend. Mit Ausbaumöglichkeiten auf mehrere Festplatten, Datensicherungs- und Diskettenlaufwerke. Ideal als Hauptstation in einem Netzwerk, für hochstehende CAD/CAM/CAE oder schnelle Datenbankanwendungen.



Der Schnellarbeiter:
Mini Titan 286+

Noch schneller als sein kleiner Bruder. Sein Mikroprozessor arbeitet mit 12 Mhz ohne Wartezyklus. Er eignet sich für anspruchsvolle kommerzielle Einsätze, rechenintensive CAD/CAM-Software sowie technisch-wissenschaftliche Anwendungen.

Der Allrounder:
Mini Titan 286

Kostengünstig, schnell und zuverlässig wie jedes Gerät der Titan-Reihe. Speziell geeignet zur Textverarbeitung, Buchhaltung, als Workstation in einem Netzwerk oder als Terminal an einem Hostsystem.

ELECTRONIC MARKETING

Your Swiss distributor for high technology

EM Electronic Marketing AG
Bahnhofstrasse 60
CH-4132 Muttenz-Basel
Tel. 061-61 53 53
Fax 061-61 48 60

Rufen Sie uns an.
Die detaillierten Unterlagen mit Bezugsquellen-Angabe stehen für Sie bereit.
Tel. 061/61 53 53.

UNIX – Konkurrent für OS/2?

UNIX ist das Standardbetriebssystem bei Mittel- und Grossrechnern. Hat es nun auch für Personal Computer seine Existenzberechtigung, nachdem IBM mit seinem OS/2 multitaskingfähig wurde? Seit vor etwa knapp zwei Jahren die ersten 386-PCs vorgestellt wurden, gibt es eine immer grösser werdende Anzahl von UNIX-Implementationen zu kaufen. Dieser Artikel soll Sie bei der Entscheidungsfindung unterstützen, ob UNIX auch für Ihre Zwecke geeignet ist und sich eine Einarbeitung lohnt. Es wird hier kein Wert auf Vollständigkeit der Befehle gelegt. Stattdessen werden nur sporadisch einige wichtige Kommandos angesprochen, die einen kleinen Einblick in die Leistungsfähigkeit von UNIX gewähren.

Michael Schlingmann

Der Erfinder von UNIX ist der amerikanische Telefonmulti AT&T. Dort wurde um 1970 ein neues Betriebssystem entwickelt, das zunächst nur für den hausinternen Gebrauch vorgesehen war. Bald wies es aber einige herausragende Merkmale auf, so dass kein Grund mehr dafür bestand, es der Oeffentlichkeit vorzuenthalten. UNIX kann seit etwa 1975 von jedermann gekauft werden.

Eine seiner Eigenschaften ist die hervorragende Portabilität: UNIX kann man sowohl auf PCs als auch auf Grossrechnern fahren, da 90% des Quellcodes in C geschrieben ist (ein Programmierkurs in C beginnt in der nächsten Ausgabe von M+K). Programmpakete müssen also nur einmal angeschafft werden.

Wer aber deshalb denkt, dass in UNIX alles standardisiert ist, liegt falsch. Auch hier gibt es eine Menge Implementationen, die nicht immer untereinander kompatibel sind (siehe Bild 1). Dies rührt vor allem daher, dass UNIX zuerst an verschiedenen Universitäten eingesetzt und verbreitet wurde. Die Uni's konnten für relativ wenig Geld eine Lizenz von AT&T erwerben und waren damit berechtigt, das Betriebssystem für ihre Zwecke weiterzuentwickeln und finanziell zu vermarkten. In der Regel sind die Unterschiede zum Original aber geringerer Natur, so dass eventuell anfallende Aenderungen schnell durchgeführt werden können.

Was in Bild 1 als erstes auffällt, sind die verschiedenen Bezeichnungen für quasi dasselbe Betriebssystem. Den Namen UNIX dürfen nur die Versionen führen, die Original-Implementationen der AT&T sind. Schon aus diesem Hause sind drei Alternativen lieferbar, wobei UNIX System V Version 3 die zur Zeit aktuellste ist. Auf diese Version beziehen sich auch der vorliegende Beitrag und die meisten Bücher.

Nachdem sich der Erfolg von UNIX absehen liess, brachte die Universität von Berkeley ein Produkt auf den Markt, das dem technisch-wissenschaftlich interessierten Kunden näher kommt und sich durch einige Verbesserungen wie z.B. der Möglichkeit der virtuellen Adressierung und dem schnelleren Aufruf von Dateien auszeichnet. Durch die Verwendung flexibler Dateigrössen wird der Speicher besser ausgenutzt. Die Berkeley-Portierungen sind von AT&T lizensierte Anpassungen an eine neue Hardware unter Beibehaltung von Systemschnittstellen und Funktionsumfang.

Die dritte Gruppe besteht aus den sogenannten UNIX-Derivaten, deren Kompatibiltät zu AT&T unter Umständen nicht mehr gegeben ist, da sie nicht lizensiert sind. Der bekannteste Vertreter ist XENIX von Microsoft. AT&T und Microsoft unterzeichneten im letzten Jahr ein Abkommen über eine engere Zusammenarbeit, so dass bei XENIX wohl keine Kompatibilitätsprobleme auftreten dürften. Insbesondere ist XENIX 3.0 kompatibel zu UNIX Version III und XENIX 5.0 zu UNIX V.

Als weitere wichtige UNIX-Derivate sind zu nennen: PC/IX (IBM), HP-UX (Hewlett-Packard), SINIX (Siemens), ULTRIX (DEC) und VNX (läuft ebenfalls auf DEC-Rechnern unter dem Betriebssystem VMS).

Noch einige Anmerkungen zu XENIX: Es handelt sich hier um eine UNIX-Anpassung für die IBM-PC/AT-Serie und Kompatible, die einige Erweiterungen für die kommerzielle Nutzung aufweist. Zu erwähnen sind hier Record- und File-Locking, Systemverwaltung, Unterstützung für Grafik und Netzwerke und die Abwärtskompatibilität zu früheren XENIX-Versionen. Insgesamt wurde dieses Betriebssystem schon 200'000 mal installiert und ist damit weiter verbreitet als alle anderen UNIX-Systeme zusammen. Die Anzahl der unter XENIX verfügbaren Programme liegt zur Zeit etwa bei 1'000 und steigt stetig. Obwohl auf die Intel-Prozessoren optimiert, können XENIX-Applikationen auch auf andere Computer übertragen werden. In Kooperation mit AT&T wird zur Zeit eine Version für den Intel 80386 entwickelt. Dieses Produkt wird dann wieder unter dem Namen UNIX vertrieben.

Eigenschaften von UNIX

Wird ein Wechsel des Betriebssystems ins Auge gefasst, so muss das neue System (UNIX) gegenüber dem alten (MS-DOS) natürlich einige Vorteile bringen. Hier kurz die wichtigsten:

- *portierbar*
  Dieselben Programme können auf völlig unterschiedlichen Prozessoren laufen.
- *multiuserfähig*
  Mehrere Benutzer arbeiten gleichzeitig am System, ohne sich gegenseitig zu stören. Die Daten der einzelnen Personen können durch Passworte geschützt werden.

Originale	Lizenzen	Derivate
UNIX III	bsd 4.1	HP-UX
UNIX V		PC/ix
UNIX 7		SINIX
		ULTRIX
		XENIX

Bild 1: Die verschiedenen UNIX-Implementationen

- *multitaskingfähig*
  Damit ist es dem Benutzer möglich, mehrere Programme gleichzeitig zu bearbeiten, z.B. einen Brief zu schreiben und im Hintergrund ein Spreadsheet berechnen zu lassen.
- *timesharing*
  Bei Multitasking können den einzelnen Prozessen unterschiedliche Prioritäten zugeordnet werden. So wird z.B. nur gedruckt, wenn gerade keine Berechnungen laufen.
- *Zugriffsrechte für Dateien*
  Dateien können selektiv gegen Zugriff von aussen geschützt werden.
- mit der *shell* steht eine mächtige Kommandosprache zur Verfügung, die es erlaubt, relativ komplizierte Programme zu schreiben Ausserdem werden 200 bis 300 Dienstprogramme serienmässig mitgeliefert.
- *USENET,* ein weltweites UNIX-Netz, kann benutzt werden. Diesem Netz ist ein Informationsdienst angegliedert, über den man UNIX-Neuheiten über sein jeweiliges Fachgebiet erfahren kann.

Natürlich hat auch UNIX seine Nachteile, die nicht verschwiegen werden sollen:

- Bei der Verarbeitung sehr grosser Datenbestände (in Datenbanken) ist UNIX relativ langsam. Das hat seinen Grund im Zugriff über Indexblöcke. Bei diesem Verfahren sind zum Finden eines Datensatzes in der Regel mehrere Plattenzugriffe notwendig. Es gibt hier bessere Lösungen.
- Echtzeitverarbeitung wie z.B. die Messung von Temperaturverläufen ist nur unter grossen Schwierigkeiten möglich. Auf diesem Gebiet ist sogar MS-DOS überlegen.
- Zu Beginn einer Sitzung muss der Benutzer zwar ein Passwort eingeben, es ist aber nicht möglich, weitergehende Sicherungen einzubauen. Hat sich also jemand unberechtigterweise einen Benutzernamen samt Passwort verschafft (was wohl nicht allzu schwierig sein dürfte), kann er nach Herzenslust in den Dateien wühlen.
- Systemblockierung durch unsachgemässe Handhabung ist relativ leicht zu erreichen, da der Benutzer praktisch keinen Beschränkungen unterliegt. Hat er es durch ein fehlerhaftes Programm geschafft, die Festplatte vollzuschreiben, sind auch die anderen Benutzer blokkiert.
- UNIX ist multiuserfähig. Das heisst aber nur, dass zwei Benutzer auf dieselben Daten zugreifen können.

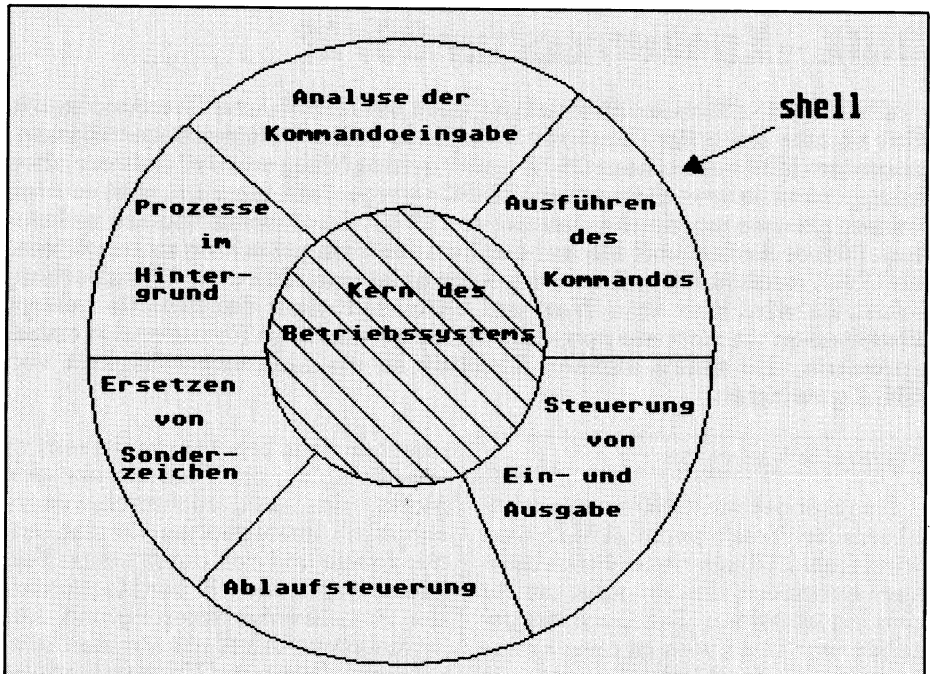


Bild 2: Die shell umschliesst das Betriebssystem wie eine Schale

Der Haken dabei ist, dass jeder nur eine Kopie des Originaldatensatzes bekommt. Was passieren kann, wenn zwei Benutzer gleichzeitig auf denselben Datensatz zugreifen, können Sie sich wohl denken.
- Durch die Unzahl von Hilfsprogrammen und shell-Befehlen ist man zwar äusserst flexibel, doch fällt zumindest dem Anfänger der Ueberblick schwer. Vor allem können an fast jeden Befehl eine Reihe von Optionen angehängt werden, so dass der Durchblick vollends verloren geht. Es existiert zwar ein «elektronisches Handbuch», das Sie mit dem Befehl

man KOMMANDO

auf den Bildschirm bringen können, doch hier eine Antwort auf eine Frage zu finden, ist oft nicht ganz einfach. Und das Sytemhandbuch ist etwas umfangreicher als das von MS-DOS...

Es besteht allerdings der Grund zur Hoffnung, dass in der Ihnen vorliegenden UNIX-Implementation der eine oder andere Nachteil behoben ist. Wie dem auch sei, UNIX bietet wesentlich mehr als MS-DOS. Im folgenden werden einige Stichworte näher beleuchtet.

Aufbau von UNIX

Bild 2 zeigt schematisch die Struktur von UNIX. Der Benutzer kommt nur mit der shell in Berührung, aber nie mit dem Kern des Betriebssystems. Die shell leistet etwa dasselbe wie command.com in MS-DOS und noch einiges mehr.

Da es sich um ein Multiusersystem handelt, ist das Dateisystem hierarchisch aufgebaut (auch das wird Ihnen bekannt vorkommen), siehe Bild 3. Der Benutzer kommt also nur an seine eigenen Daten heran, es sei denn, die anderen Benutzer geben ihre Daten für den Publikumsverkehr frei, was durchaus möglich ist.

Die shell

UNIX bietet statt grösserer Programmpakete eine Menge einzelner Dienstprogramme an, die in der Regel nur für eine spezielle Aufgabe taugen. Für die Kommunikation mit UNIX ist die shell zuständig. Sie umschliesst den Betriebssystemkern wie eine Schale (daher auch der Name) und ermöglicht es, im Dialog Anweisungen an UNIX zu erteilen. Im Wesentlichen hat sie folgende Aufgaben zu erfüllen:

- Interpretation der Kommandoeingabe
- Ausführung des Kommandos, wenn möglich
- I/O-Verkehr
- Pipeverarbeitung (dazu später mehr)
- Ablaufsteuerung
- Erzeugen von Vorder- und Hintergrundprozessen

Der letzte Punkt bedarf einer ausführlicheren Erklärung. Ein Prozess ist

ein Programm, das gerade läuft. Ein Prozess, der im Vordergrund abläuft, kann über das Terminal mit Daten versorgt werden. Bei der Eingabe von shell-Kommandos erscheint das prompt-Zeichen, wenn UNIX bereit ist für weitere Dateneingaben. Dieses Zeichen taucht erst dann auf, wenn das letzte Kommando beendet wurde. Ein Hintergrundprozess unterscheidet sich dadurch, dass UNIX nicht auf die Beendigung des Programms oder Kommandos wartet, sondern sich sofort wieder mit dem prompt-Zeichen meldet. Das heisst aber nicht, dass das Kommando schon ordnungsgemäss beendet ist. Vielmehr wird es abgearbeitet, wenn gerade Prozessorzeit zur Verfügung steht.

Wenn Sie ein Programm im Hintergrund laufen lassen wollen, hängen Sie einfach ans Ende der Kommandozeile ein &-Zeichen. Es wird dann eine Prozessnummer ausgegeben, mit der das Hintergrundprogramm bei Bedarf vorzeitig abgebrochen werden kann. Der Grund für die Prozessnummer besteht in der Tatsache, dass ein im Hintergrund ablaufendes Programm nicht mehr auf dem Bildschirm zu sehen ist. Sie können es damit auch nicht mehr kontrollieren. Durch die Vergabe einer Nummer gibt es aber immerhin die Möglichkeit, das Programm abzubrechen oder zu unterbrechen. Dazu gibt es das Kommando

kill ZAHL PROZESSNUMMER

ZAHL ist eine Option für die Behandlung des Hintergrundprogramms. ZAHL=9 bedeutet z.B., dass sofort abgebrochen wird. Selbstverständlich können Sie nur die Programme beeinflussen, die Sie auch selbst gestartet haben. An die anderer Benutzer können Sie nicht heran.

Es existiert auch ein nützlicher Befehl, der Sie über die gerade vom System in Arbeit stehenden Prozesse informiert.

ps [-ef]

gibt ein Verzeichnis der Prozesse heraus. Aufgelistet werden hier ausser der Prozessnummer der Name des Terminals und des gestarteten Programms sowie die bisher verbrauchte CPU-Zeit. Geben Sie die Option -e an, dann werden alle Prozesse, nicht nur Ihre eigenen, ausgedruckt. -f informiert über das Programm, von dem der Prozess aus gestartet wurde und gibt zusätzlich noch die Uhrzeit des Programmstarts an.

Vielleicht brennt Ihnen die Frage auf den Nägeln, warum denn Hintergrundprozesse überhaupt erwünscht sind, wenn man sie doch nur bedingt kontrollieren kann. Stellen Sie sich dazu vor, Sie hätten ein umfangreiches Tabellenkalkulationsprogramm in Arbeit, bei dem Sie einige Aenderungen vornehmen müssen. Zur gleichen Zeit müssen Sie aber ganz dringend einen Brief schreiben. Bei MS-DOS warten Sie nun zehn Minuten, bis die Tabellen reorganisiert sind. UNIX-Benutzer dagegen schieben die Kalkulation in den Hintergrund und haben dann Zeit für den Brief.

Wie in MS-DOS kann man Batch-Dateien anlegen, die die Programme hintereinander ablaufen lassen. Im Vergleich zum Kommandointerpreter von MS-DOS hat die shell einen grösseren Wortschatz. Wenn Sie sich nicht an C herantrauen, können Sie viele Anwendungen direkt auf der shell programmieren. Der Vorteil besteht in der interaktiven Vorgehensweise, die eine Compilation nach Aenderungen überflüssig macht. Sie haben allerdings nicht den Zugang zur Maschinenebene. Programme, die irgendeine Hardware steuern, müssen also in einer höheren Programmiersprache geschrieben werden.

Einloggen und Passwort

Unter Einloggen wird die Mitteilung an das Betriebssystem verstanden, dass man ab sofort mit ihm arbeiten will. Dazu muss es Rechenzeit zur Verfügung stellen.

Rechenzeit muss in der Regel von irgend jemandem bezahlt werden. Ausserdem besteht die Möglichkeit, dass man Dateien bearbeitet, die nicht von jedermann eingesehen werden dürfen. Zu diesem Zweck gibt es ein Passwort, das nach dem Einloggen eingetippt werden muss. Natürlich ist es auf dem Bildschirm nicht zu sehen. Wer das erste Mal mit UNIX arbeitet, bekommt sein Passwort vom Super-User, dem Systemverwalter, zugeteilt. Sie können es aber jederzeit ohne Rücksprache mit ihm abändern. Der Befehl dazu lautet

passwd

Das alte Passwort erscheint auf dem Bildschirm mitsamt der Aufforderung, ein neues einzugeben. Zur Vermeidung von Schreibfehlern müssen Sie das zweimal tun. Alle Benutzer, auch der Super-User, sind in der Datei /etc/passwd eingetragen. In dieser Datei stehen der Name des Benutzers, sein Passwort, eine Kennummer, eine Gruppennummer (es besteht die Möglichkeit, dass die Mitglieder einer Gruppe Dateien miteinander teilen), ein Kommentarfeld und Name des Verzeichnisses, in dem gearbeitet wird. Die letzte Eintragung schliesslich bezieht sich auf das Programm,

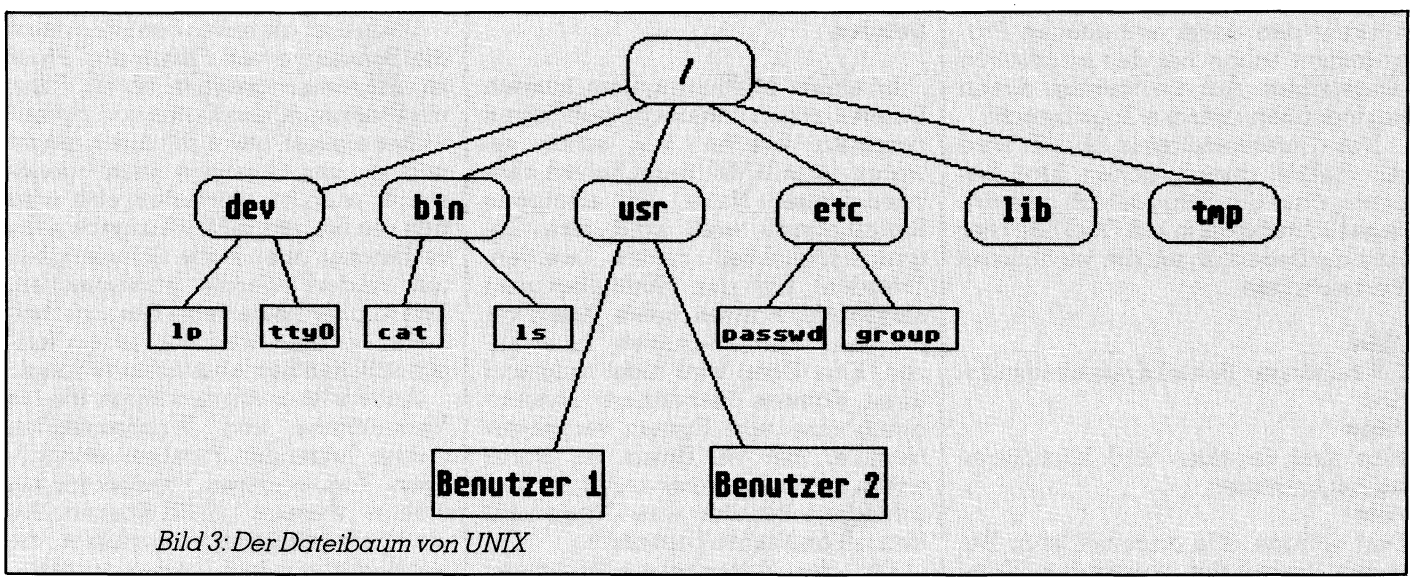


Bild 3: Der Dateibaum von UNIX

das nach dem Einloggen automatisch gestartet werden soll. Siehe dazu autoexec.bat in MS-DOS.

An /etc/passwd kommt natürlich nur der Super-User heran. Die Geschwindigkeit des Rechners hängt wesentlich von der Anzahl der gerade eingeloggten Benutzer ab. Wenn Sie alleine am System arbeiten, ergeben sich natürlich keine Wartezeiten. Ansonsten müssen Sie aber die Prozessorzeit mit Ihren Kollegen teilen. Wollen Sie wissen, wer gerade angeschlossen ist, tippen Sie

who

ein und erhalten danach eine Liste der Benutzer mitsamt dem Datum des letzten Einloggens.

Eine Arbeitserleichterung stellt die Möglichkeit zum Versenden elektronischer Post dar. Mit

write BENUTZER

können Sie eine Botschaft an einen beliebigen Benutzer schicken, die dann auf seinem Bildschirm erscheint. Bei Angabe einer Reihe von Namen werden alle angegebenen Personen mit einbezogen. Nicht verschwiegen werden sollte in diesem Zusammenhang, dass es einen Befehl zum Sperren des Terminals gegen externe Botschaften gibt. Sie haben also immer die Alternative, ungestört arbeiten zu können.

Dateihierarchie und Verzeichnisse

Aehnlich wie MS-DOS sind auch in UNIX Befehle zum Anlegen von Unterverzeichnissen enthalten. Da UNIX aber ein Multiusersystem ist, sollten Redundanzen so weit als möglich vermieden werden. Aus diesem Grund werden einige Verzeichnisse mitsamt den darin enthaltenen Programmen schon bei der Installation eingerichtet. Auf die meisten davon hat jeder Benutzer das Zugriffsrecht.

Das Unterverzeichnis NAME wird mit /NAME angesprochen. Man beachte, dass der Schrägstrich eine andere Richtung als in MS-DOS hat. Hier kurz die Bedeutungen der wichtigsten Verzeichnisse:

/bin
Oft benötigte Betriebsystemkommandos
/dev
Hier sind Drucker- und Bildschirmtreiber enthalten
/etc
Dort werden alle angemeldeten Benutzer aufgelistet. Ausserdem Kommandos zum Starten und Herunterfahren des Systems. Dieses Verzeichnis ist nur dem Super-User zugänglich.
/lib
Bibliotheken, z.B. die des mitgelieferten C-Compilers.
/usr
Unterverzeichnisse für die Benutzer
/usr/bin
Selten benötigte Betriebssystemkommandos
/usr/doc und **usr/man**
Elektronisches Handbuch, praktisch die gesamte Bedienungsanleitung für das System.
/usr/sys
Zugang zum Kernel

Nach dem Einloggen setzt das System Sie in Ihr persönliches Hauptverzeichnis. Wenn Sie wissen wollen, wo Sie sich im grossen «Dateibaum» von UNIX befinden, so erreichen Sie das durch Eingabe von

pwd

Dies ist eine Abkürzung für «**p**rint **w**orking **d**irectory» und zeigt den Pfad bis zu Ihrem Verzeichnis an.

Ein angemeldeter Benutzer kann von seinem Verzeichnis aus selbst Unterverzeichnisse anlegen. Die dazu notwendigen Befehle werden Ihnen zum Teil sicher bekannt vorkommen:

cd NAME
Wählt das Unterverzeichnis NAME an
mkdir NAME
Legt ein neues Verzeichnis an.
rmdir NAME
Löscht eine leeres Verzeichnis
mv NAME1 NAME2
Aehnlich dem rename in MS-DOS, hier wird aber ein Verzeichnis umbenannt.

Dateien

In einem Multiusersystem müssen Dateien gegen unberechtigten Zugriff geschützt werden. Wie schon erwähnt, ist in UNIX dieser Schutz eher zweifelhafter Natur, da erfahrene Eindringlinge wohl bald eine Zugangsmöglichkeit finden werden. Trotzdem hat der Normalbenutzer einige Alternativen, seine Daten vor anderen Normalbenutzern zu schützen. Eine Datei wird nicht aufgrund ihres Namnes identifiziert, sondern durch eine vom System vergebene Nummer, die ein Unikat ist. Damit treten keine Probleme auf, falls verschiedene Benutzer ihre Dateien mit denselben Namen benennen.

Für die Zugangsberechtigungen gibt es in jeder Datei neun «protection bits», die frei gesetzt werden können:

r Datei darf gelesen werden
w Datei darf beschrieben werden
x Datei darf als Programm ausgeführt werden, es sind jedoch keine Aenderungen möglich

Haben wir anstatt einer Datei ein Verzeichnis vor uns, so gilt

r Auflisten des Verzeichnisses erlaubt
w Dateien dürfen angelegt und gelöscht werden
x Auf Einträge in diesem Verzeichnis darf zugegriffen werden

Eigentlich sind für diese Funktionen nur drei Bits notwendig. Es werden aber drei Gruppen von Benutzern unterschieden:

- Der Benutzer selbst (also der, der die Datei angelegt hat)
- Die Arbeitsgruppe, der er angehört
- Alle übrigen Benutzer

Insgesamt werden also neun Bits belegt.

Das Kopieren von Dateien geschieht analog zu MS-DOS:

cp DATEIALT DATEINEU

kopiert DATEIALT in DATEINEU. DATEINEU kann auch ein Verzeichnis sein, in das Sie mehrere Dateien hineinkopieren möchten.

Auch die meisten der für Verzeichnisse gültigen Kommandos können in Verbindung mit Dateien benutzt werden.

Filter und Pipes

UNIX liess als erstes Betriebssystem die Benutzung von Filtern und Pipes zu. Allgemein gesehen ist ein Filter ein Programm, das Daten von irgendwoher einliest, etwas mit ihnen macht und sie anschliessend nach irgendwohin ausgibt. Normalerweise wird das die Standard-Ein/Ausgabe sein, es bereitet aber keine Schwierigkeiten, andere Geräte anzusprechen. Das ist aber heute keine Besonderheit mehr, sondern wird von jedem fortschrittlichen Betriebssystem verlangt.

Anders ist es mit den Pipes, die die Verknüpfung und Synchronisation zweier laufender Prozesse ermöglichen. Zum besseren Verständnis ein kleines Beispiel. UNIX-Kommandos sind klein geschrieben, Variablen, die der Benutzer zuteilt, dagegen gross.

ls > INHALT

nimmt das gerade aktuelle Verzeichnis her und schreibt dessen Inhalt als ASCII-Zeichen in die Datei INHALT. Ist INHALT nicht vorhanden, wird die Datei automatisch angelegt.

wc < INHALT

gibt die Anzahl der Zeichen, Worte und Zeilen aus, die in INHALT vorkommen.

Nun wäre es natürlich toll, wenn die Worte im Verzeichnis direkt gezählt werden könnten, ohne den Umweg über INHALT. Gerade diese Aufgabe erledigt die Pipe

ls | wc

Der Ablauf von ls wird angehalten, wenn wc gerade in Arbeit ist. Ebenso stoppt wc, wenn von ls gerade nichts gelesen wird. Die Synchronisation ist Sache von UNIX.

Interessant wird die Sache vor allem dann, wenn Filter und Pipe verbunden werden.

Die Einfachkeit, mit der Ein- und Ausgaben auf andere Geräte umgeleitet werden können, ist eine der Stärken von UNIX. So hängt z.B. die Sequenz >> eine Ausgabe an eine bestehende Datei an. Dazu ein Beispiel: Mit

cat DATEI1

wird in der Regel der Inhalt von DATEI1 auf den Bildschirm gebracht. Schreiben wir stattdessen

cat DATEI1 >> DATEI2

so wird DATEI1 an DATEI2 angehängt. Die logische Konsequenz besteht darin, dass wir mit

KOMMANDO >> DATEI2

das Ergebnis jedes beliebigen Kommandos an DATEI2 anfügen können. Dies erleichtert die Protokollierung von Arbeitsabläufen ganz wesentlich.

Der Super-User (SU)

Damit das hierarchische Dateisystem funktioniert, muss es jemanden geben, der das Ganze überwacht und auch einige notwendige Verwaltungsarbeiten übernimmt. Diese Person nennt man den «Super-User». Nur er hat Zugang zum gesamten System und kennt alle Passworte. Er teilt den Benutzern ihre Verzeichnisse zu, die man auch home directories oder login directories nennt. Unterverzeichnisse darf der Benutzer selbst ohne Rücksprache mit dem SU einrichten. Zum System haben nur die Personen Zugang, die dem SU gemeldet sind. Von ihnen gestartete Programme kann der SU unterbrechen.

Zum directory /etc hat nur der SU Zugang. Dort befinden sich Programme, die es erlauben, das System hoch- und wieder herunter zu fahren. Ebenso existieren Tools zur Ueberprüfung der Konsistenz von Dateien usw.

Der SU hat die Möglichkeit, eine Meldung an alle Benutzer weiterzugeben. Diese erscheint auf dem Bildschirm der Person, die sich gerade anmeldet.

UNIX auf PCs

Da der Einsatz von UNIX nicht ganz billig ist, stellt sich spätestens jetzt die Frage, ob Sie das System überhaupt brauchen. Dazu einige Gedanken, die vielleicht zur Entscheidungsfindung beitragen können: Gelegenheitsbenutzer von Computern sollten die Idee ganz schnell wieder aufgeben. Im Gegensatz z.B. zum Macintosh existiert keine grafische Benutzeroberfläche. Wenn Sie schon mit den Befehlen in MS-DOS Ihre Schwierigkeiten haben, so werden diese sich mit Sicherheit vergrössern. Sie sind zwar mit UNIX wesentlich flexibler. Die Flexibilität wird aber erkauft mit einem Rattenschwanz von Optionen bei den meisten Befehlen, die zu merken auch dem Profi nicht leicht fallen.

Die formalisierte Ausdrucksweise im Verkehr mit der shell trägt nicht gerade zum Verständnis bei. Meldungen des Betriebssystems fehlen in der Regel. Man ist sich also über den Zustand des Rechners meist im Unklaren. Sie erhalten z.B. keine Warnung, wenn Sie einen Befehl zur Formatierung der Festplatte eingeben. Dafür wird sie aber wesentlich schneller formatiert als in MS-DOS, falls das ein Trost ist.

UNIX auf einem 8086-Prozessor einzusetzen ist zwar nicht ganz unmöglich, aber zumindest sinnlos. Sie brauchen wenigstens einen AT mit grosser Festplatte, da Sie eine ganze Reihe von Disketten bekommen.

Wenn Sie sich mit Computern auskennen und vielleicht sogar professionelle Software entwickeln, werden Sie sich mit UNIX schnell anfreunden. Sie haben ausgedehnte Möglichkeiten, Dateien zu manipulieren und können Programme schreiben, die auch auf Workstations laufen. Der normalerweise im Lieferumfang enthaltene C-Compiler wird durch die Hilfsprogramme yacc und lint zu einem mächtigen Entwicklungswerkzeug.

Da UNIX im Gegensatz zu MS-DOS die volle Datenbreite von 16 Bit ausnutzt, laufen die Programme wesentlich schneller ab.

Haben Sie sich für UNIX entschieden, so gibt es zwei Möglichkeiten zur Anschaffung: entweder Sie nehmen ein Original-UNIX oder ein Derivat. Das Original, das z.B. von IBM vertrieben wird, kommt finanziell etwas teurer, aber dafür ist es eben vom Marktführer. Die Alternative besteht im Erwerb von XENIX. Es ist für zwei- bis dreitausend Franken zu haben und bietet zumindest genauso viel. Vor allem wird sich ein früherer Benutzer von MS-DOS schneller zurecht finden. □

Literaturhinweise

S.R. Bourne
Das UNIX System
Internationale Computer Bibliothek, 1985

Schnupp/Müller
UNIX-Programmieren mit Kommandos
Oldenbourg Verlag München, 1985

X/O/P/E/N
Portability Guide
Elsevier Science Publishers B.B., Amsterdam, 1985

AT&T and Bell Labs
UNIX System User's Manual
Western Electric, Nr 307-101, 1983

C. Wolfinger
Keine Angst vor UNIX
VDI Verlag Düsseldorf, 1987

COMPUTER-SPLITTER

Apple und DEC

(484/eh) Apple und DEC verkündeten eine engere Zusammenarbeit. Ein erstes Produkt dieser Zusammenarbeit soll eine Software sein, die den Einsatz der VAX als File-Server im Appleshare-Netz erlaubt sowie ein Appletalk-DECNet-Uebergang, der jedem Macintosh Zugang zu jeder DECNet-Arbeitsstation verschafft. □

Mit PCTOOLS wird MS-DOS einfach

Haben Sie schon einmal versehentlich ein Datenfile von der Harddisk gelöscht, von dem Sie keine Sicherungskopie besassen oder versucht die Harddisk Ihres PCs auf Diskette zu sichern? Nun dann sind Sie mit Sicherheit der nächste Käufer von PCTOOLS, denn mit diesem Programm ist es ein Leichtes, den Löschvorgang eines Files rückgängig zu machen oder das Backup der Harddisk um ein mehrfaches zu beschleunigen.

Heinz Kastien

PCTOOLS ist ein sogenanntes Utility-Programm, also ein Werkzeug, das gewisse Disk- oder Fileoperationen vereinfacht oder beschleunigt. Neben dieser reinen Effizienzsteigerung werden aber zusätzliche Programmhilfen angeboten, die im MS-DOS nicht enthalten sind, wie z.B. die UNDELETE-Funktion zum Rückgängigmachen eines Löschvorganges.

Einführung und Installation

PCTOOLS Deluxe wird auf einer 5.25 Zoll bzw. 3.5 Zoll Diskette angeboten und eignet sich für alle IBM-kompatiblen PCs oder ATs sowie das IBM PS/2. MS-DOS 2.0 oder eine höhere Version ist erforderlich. Ein RAM-Speicher von 256 KByte ist zwingend vorgeschrieben, soll PCTOOLS speicherresident installiert werden, sind 512 KByte wünschenswert.

Die Installation von PCTOOLS erfolgt mit dem Installationsprogramm PCSETUP. Dieses Programm eröffnet einen Subdirectory eigener Wahl in einem näher zu spezifizierenden Drive und kopiert alle erforderlichen Programmfiles in dieses Directory. Gleichzeitig wird im DOS der Befehl FORMAT.EXE in FORMAT!.EXE umbenannt sowie ein spezielles FORMAT.BAT-File erstellt, mit dem PCFORMAT.EXE aufgerufen wird. Ein weiterer Bestandteil des Installationsprogramms ist das Programm PC CACHE, das einen schnelleren Zugriff auf die Disk zulässt. Da PCTOOLS nicht kopiergeschützt ist, können von der Originaldiskette beliebig viele Sicherheitskopien angefertigt werden.

Das Hauptprogramm wird mit dem Befehl PCTOOLS aufgerufen, wobei eine Reihe von Parametern zur Optimierung definiert werden können.

PCTOOLS [/BW] [/RnnnK] [/Fn] [/]

Der Parameter [BW] erlaubt eine klarere Darstellung auf einem Monochrombildschirm, der an einen Farbgrafikadapter angeschlossen ist.

Der Parameter [RnnnK] installiert das Programm speicherresident, wobei mit nnn der Speicherbereich in KByte definiert wird, der für PCTOOLS reserviert werden soll. Die idealen Speicherbereiche sind im Handbuch ausführlich beschrieben.

Wird PCTOOLS als speicherresidentes Programm benutzt, erzeugt es ein Overlay-File im aktuellen Directory. Der Parameter [/Od] bestimmt einen alternativen Drive für das Overlay-File. Diese Anwendung ist dann von Interesse, wenn ein Single-Drive-Disk zusammen mit einer RAM-Disk benutzt wird.

Der Parameter [Fn] definiert eine Taste, die das speicherresidente PCTOOL aktiviert.

Durch die verschiedensten Installationsarten kann PCTOOLS allen Bedürfnissen und Hardwarekonfigurationen optimal angepasst werden. Das Hauptprogramm enthält alle DOS-Utilities. Sofort nach dem Aufstarten von PCTOOLS springt das Programm in diesen Hauptteil, der die File-Funktionen enthält. Der Bildschirm ist grafisch sehr gut in einen Kopf-, Haupt- und einen Fussteil gegliedert.

Bei Verwendung eines Farbenmonitors ist die Darstellung durch die gezielte Farbkombination sehr übersichtlich. Im Kopf wird der Titel, die PCTOOL-Version und der Volumelabel der zu untersuchenden Disk sowie die Stellung der Scroll Lock-Taste angezeigt. Der Mittelteil des Bildschirms dient der Anzeige und Kommunikation. Die Fusszeilen zeigen die Auswahl der unterstützten DOS-Befehle in Form eines Menüs an. Farbig hervorgehobene Buchstaben der Befehle markieren die Taste, mit denen diese initialisiert werden.

DOS-Utilies

Die DOS-Utilities unterstützen die normalen DOS-Befehle und machen die Handhabung dieser Befehle, die für den MS-DOS-Benutzer oftmals nicht ganz einfach zu erlernen sind, zum Vergnügen. PCTOOLS unterstützt die Befehle COPY, COMPARE, FIND, RENAME, DELETE, VERIFY, ATTRIBUTE und PRINT. Diese Befehle entsprechen dem normalen DOS-Befehlssatz und bedürfen daher keiner besondere Erläuterung.

Die ESC-Taste hat in allen Stufen die gleiche Funktion, nämlich die Beendigung von PCTOOLS bzw. die Rückkehr zum aufrufenden Grundmenü. Beim Aufstarten von PCTOOLS erscheinen immer die ersten 26 Files des aktiven Directories. Angezeigt wird der Filename, dessen Extension, die Filegrösse, das Attribut und das Datum der letzten Aenderung. Mit der Funktionstaste F2 wird zu einer Darstellung umgeschaltet, bei der nur 13 Files angezeigt werden, in dieser Darstellungsform erscheinen jedoch zusätzlich die Nummer des Clusters, die Aenderungszeit und die erweiterten Fileattribute. Auch die Funktionstaste F10 erfüllt in allen Stufen des Programms die gleiche Funktion. Mit ihr wird der aktive Drive geändert, wobei nach einem Wechsel immer zuerst der Drive mit allen Subdirectories und Zugriffspfaden auf dem Bildschirm erscheint. Aus diesem Baum lässt sich dann der gewünschte Directory selectionieren. Im Hauptprogramm kommt noch den Funktionstasten F8 und F9 eine besondere Bedeutung zu. F8 listet nur speziell ausgewählten Files, F9 markiert die definerten Files farbig. Durch dieses Auswahlverfahren werden die übrigen Files vom nachträglichen Listen oder Sortieren ausgeschlossen. Die Funktionstaste F1 macht die Auswahl der Taste F9 wieder rückgängig. Von spezieller Bedeutung sind die Befehle

MOVE
Dieser Befehl kopiert ein File von einem Directory in einen anderen und löscht gleichzeitig das Sourcefile.

VIEW/EDIT
Zeigt den Inhalt eines Filesektors in hexadezimaler Form sowie im ASCII-Code und ermöglicht die Aenderung einzelner Bit.

LIST
Mit dieser Funktion werden alle Files eines Directories auf dem Bildschirm dargestellt, der Befehl entspricht prinzipiell dem Format, wie es beim Aufstarten von PCTOOLS vorliegt. Die Funktionstasten F2, F8 und F9 haben auch hier die bereits oben beschriebenen Aufgaben.

SORT
Sortiert die Files eines Directories nach Namen, Extension, Grösse oder Datum und speichert sie, je nach Wunsch, in der sortierten Form wieder auf dem Datenträger ab.

Ueber die Funktionstaste F3 wird ein weiteres Menü aufgerufen, das Disk und Spezialfunktionen zugänglich macht. Auch in diesem Menü findet man die Befehle COPY, COMPARE, FIND, RENAME, VERIFY, und EDIT. Die Funktion dieser Befehle ist die gleiche, wie bei den Filebefehlen, nur beziehen sich diese nun nicht mehr auf einzelne Files, sondern auf Directories oder die komplette Disk. So entspricht in diesem Programmteil der Befehl COPY dem DISKCOPY des MS-DOS und RENAME dient der Umbenennung von Directories. In dieser Form kennt MS-DOS den RENAME-Befehl nicht. Zusätzlich bietet dieses Menü noch die Befehle:

MAP
zeigt den frei verfügbaren Speicherplatz sowie die Belegung aller Tracks in grafischer Darstellung. Hierbei wird in der Grafik zwischen Boot Records, File Allocated Table, Directories, Allocated Clusters, Hidden Files, Read Only Files und Bad Clusters unterschieden. Dieser Befehl ist besonders wertvoll, um eine schadhafte Disk zu untersuchen.

LOCATE
durchsucht das Laufwerk nach einem definierten Filenamen und gibt alle Subdirectories an, in denen sich der gesuchte Name befindet.

INITIALIZE
entspricht dem FORMAT-Befehl des MS-DOS, es lassen sich Disk aller Formate formatieren.

DIRECTORY MAINT
dient der Verwaltung von Verzeichnissen und ermöglicht das Eröffnen, Umbenennen oder Löschen.

UNDELETE
ist einer der wichtigsten Befehle von PCTOOLS. Beim Löschen eines Files oder eines Subdirectories wird dieser nicht physikalisch von der Platte entfernt, sondern im Directory nur der erste Charakter des Eintrags gelöscht. Erst beim erneuten Abspeichern von Files an dieser Stelle, wird das alte File überschrieben. Solange dies nicht der Fall ist, besteht die Möglichkeit, das «gelöschte» File zu retten. PCTOOLS ruft mit dem UNDELETE-Befehl im definierten Laufwerk oder Directory alle «gelöschten» Files und Directories auf. Mit Cursor- und RETURN-Taste werden die zu restaurierenden Files oder Directories ausgewählt. Es muss nun leglich der erste Charakter des Eintrags wieder hergestellt werden, um die Files wieder zur Verfügung zu haben. Um ganze Unterverzeichnisse zu retten, muss zuerst der Subdirectory neu erstellt werden und dann die einzelnen Files restauriert werden. Die UNDELETE Funktion ist in einer Reihe anderer Utilities in verschiedenen Varianten enthalten und stellt eine der wichtigsten Funktionen des gesamten Programms dar.

SYSTEM INFO
Diese Funktion ist in vielen SETUP-Programmen der Computerhersteller enthalten, SYSTEM INFO gibt Angaben über Computertyp, Operating System, Anzahl der logischen Drive, CPU-Typ Speicherplatz, Grafikadapter usw.

PARK
Auch dieser Befehl ist bekannt, er setzt den Schreib/Lesekopf einer Disk auf eine nicht benutzte Stelle, um die Festplatte während des Transportes vor Beschädigungen zu bewahren.

HELP
liefert schliesslich Kurzinformationen zu jedem PCTOOL-Befehl.

Die beschriebene Bedienungserleichterung wird in den mannigfaltigsten Arten von vielen ähnlichen Programmen angeboten. Die effektiven Vorzüge von PCTOOLS gegenüber diesen Programmen kommen daher erst bei den Programmteilen COMPRESS, PCBACKUP, PCRESTOR, MIRROR und REBUILD voll zur Geltung. Diese Programme sind nicht im Hauptprogramm integriert und können auch von diesem aus nicht direkt aufgerufen werden.

COMPRESS

Das Programm COMPRESS belegt 100 KByte, jedoch sind mindestens 256 KByte freier Speicher erforderlich. Welcher Aufgabenbereich kommt aber nun diesem Programm zu. Beim Abspeichern eines Files wird dieses unter Umständen in verschieden grosse Fragmente zerlegt und dann auf der Disk in den unterschiedlichsten Sektoren abgelegt. Je öfter Files auf der Festplatte gelöscht und wieder überschrieben werden, umso grösser ist der Fragmentierungsgrad. Dies hat zur Folge, dass der Zugriff auf die Disk immer langsamer wird. Hier greift Compress ein, das Programm, dass völlig menügesteuert arbeitet, analysiert die Disk und sortiert anschliessend die zusammengehörigen Blöcke eines Files. Das COMPRESS-Programm verfügt über die Unterroutinen Filesortdefinition, Analyse der Disk, Analyse der Files, Analyse der Diskoberfläche, Organisationsanalyse und Datenverdichtung.

Analyse von Disk, Files und Diskoberfläche

Die Analyse der Disk hat die Aufgabe, diese rein physikalisch auf ihren Zustand zu überprüfen. Das Programm macht dann je nach Fragmentierungsgrad selbständig Vorschläge zu Komprimierung. Die Sortiervorschläge definieren die Art des Filesort, absteigend, aufsteigen, nach Datum, Extension, Grösse usw. Diese Sortierroutine wird aber erst bei der abschliessenden Compression durchgeführt. Die Diskanalyse stellt die Anzahl belegter, freier und fehlerhafter Cluster sowie Anzahl der Files fest. Die Fileanalyse zeigt die Belegung der Subdirectories an und mit der Disk Organisation Analyse wird auf Grund des Filefragmentierungsgrades die Art der Compression vorgeschlagen.

Wichtigster Teil der Analyse ist die Untersuchung der Diskoberfläche, hierbei wird auf dem Bildschirm in der bereits bekannten Grafik die Verteilung der Cluster auf der Disk und die Lage der fehlerhaften Blöcke angezeigt. Für eine 10 MB Festplatte dauert dieser Vorgang ca. 4 Minuten. Diese Analyse sollte vor jeder Komprimierung durchgeführt werden, um Sicherheit über den ordnungsgemässen Zustand der Disk zu haben und die Notwendigkeit einer Komprimierung abzuwägen.

Beim eigentlichen Komprimieren der Disk werden alle zusammengehörigen Fragmente einer Datei wieder zusammengesetzt und hierbei nach vorgängig definierten Kriterien neu geordnet. Gleichzeitig besteht die Möglichkeit frei werdende Cluster zu löschen. Auf dem Bildschirm kann die Arbeit des Programms verfolgt werden, da im Bildschirmkopf das in Arbeit befindliche File angezeigt wird. Der Hauptvorteil dieses Programmteils liegt jedoch nicht im zusätzlich frei werdenden Speicherplatz sondern viel mehr im Zeitgewinn des Diskzugriffs.

Selbstverständlich braucht ein derart komplexer Vorgang Zeit, in unserem Beispiel 7 Minuten und 38 Sekunden für die Analyse und Komprimierung einer 30 MB Festplatte mit insgesamt 16'339 Clusters, von denen 7'859 belegt waren und bei einem Fragmentierungsgrad von 19 %, sowie

weitere 4 Minuten für das Löschen der frei werdenden Cluster.

Das Programm PCBACKUP ist an und für sich nichts grundlegend Neues, es dient der Erstellung eines Backup der gesamten Festplatte auf Disketten oder der Sicherung von Files nach bestimmten Kriterien, also beispielsweise aller neuen Files seit der letzten Sicherung, oder der Sicherung von Files ab einem bestimmten Datum. Das Backup-Programm des MS-DOS macht genau das gleiche, benötigt hierzu aber wesentlich mehr Zeit. Zu Sicherung der oben beschriebenen 30 MB Festplatte mit 15.4 MB benötigt das MS-DOS-Backup 37 Minuten, hinzu kommen 20 Minuten für das Formatieren der erforderlichen vierzehn 1.2 MB Disketten. Mit PCBACKUP dauert der gleiche Vorgang nur 14.5 Min. inkl. des Formatierens der dreizehn 1.2 MB Disketten. Dies entspricht einer Geschwindigkeit von 1.084 MB pro Minute gegenüber 0.42 MB pro Minute beim MS-DOS-Backup.

PCRESTOR ist das Gegenstück zu PCBACKUP und kopiert die Backup-Disketten auf die Festplatte, es entspricht in seiner Funktion dem Restore des MS-DOS. Mit PCBACKUP gesicherte Disketten können nur mit dem PCRESTOR wieder auf die Harddisk zurückkopiert werden.

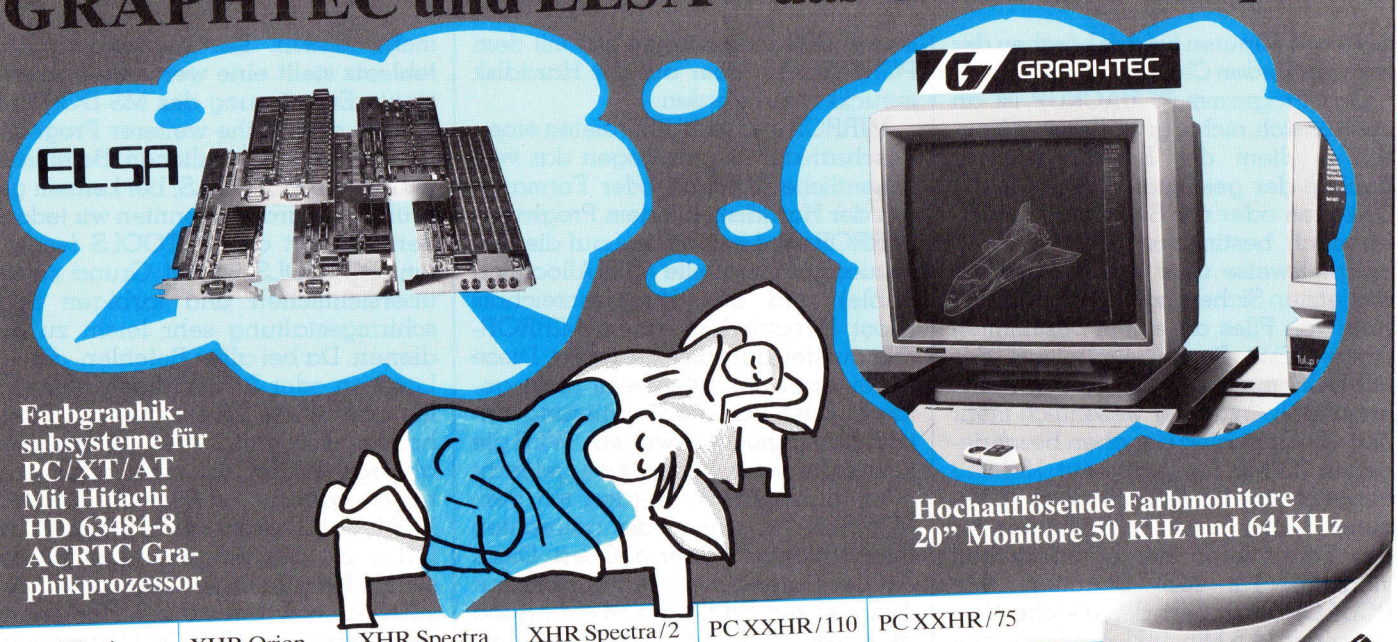
MIRROR und REBUILD bieten einen beschränkten Schutz gegen das versehentliche Löschen oder Formatieren der Harddisk. Mit dem Programm MIRROR wird bei jedem Lauf die Dateizuordnungstabelle (File Allocation Table) und das Hauptverzeichnis (Root Directory) in einem MIRROR-File auf der Harddisk gesichert. Diese Datei kann normalerweise nicht gelöscht oder durch Formatierung unbrauchbar gemacht werden. Wird die Festplatte irrtümlicherweise gelöscht oder formatiert, so werden mit dem REBUILD Programm die gelöschten Informationen wieder aktiviert. Diese Art der Datensicherung hat aber nur Aussicht auf Erfolg, wenn sie häufig genug durchgeführt wird. Ausserdem sind einige DOS-Versionen in der Lage, die MIRROR-Datei zu zerstören.

Zusammenfassung

Wer einmal mit PCTOOLS gearbeitet hat, möchte dieses Programm nicht mehr missen. Der geschilderte Befehlsatz stellt eine wertvolle und effiziente Erweiterung des MS-DOS dar. Es gibt eine Reihe weiterer Programme, die einen ähnlichen Befehlsatz bieten wie PCTOOLS, bei keinem geprüften Programm konnten wir jedoch den Komfort des PCTOOLS feststellen. PCTOOLS ist auf Grund seiner übersichtlichen und farbigen Bildschirmgestaltung sehr leicht zu bedienen. Da bei allen Befehlen, die ein Löschen oder Beschädigen eines Files oder eines Directories zur Folge haben könnten, eine Sicherheitsabfrage erfolgt, ist eine Fehlbedienung nahezu unmöglich. Obwohl PCTOOLS Deluxe nur in englisch lieferbar ist, dies trifft auch auf das 164 seitige Handbuch zu, kann man durch spielerisches Ueben die Bedienung leicht erlernen. PCTOOLS bietet jedem etwas, dem Anfänger die einfache Bedienung des MS-DOS, dem fortgeschrittenen Benutzer den Zeitgewinn bei komplizierten Diskanalysen und Sicherungsvorgängen.

Kurz vor Drucklegung hat uns die Zürcher Firma Intech/Tresordata, Tel. 01/41'57'57, mitgeteilt, dass PCTOOLS ab sofort auch in einer deutschen Version erhältlich ist. □

GRAPHTEC und ELSA – das CAD-Traumpaar



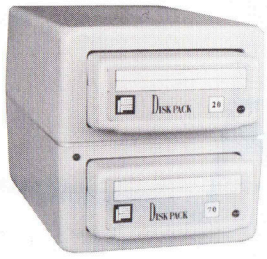
Farbgraphik-subsysteme für PC/XT/AT Mit Hitachi HD 63484-8 ACRTC Graphikprozessor

Hochauflösende Farbmonitore 20" Monitore 50 KHz und 64 KHz

Spezifikation	XHR Orion	XHR Spectra	XHR Spectra/2	PC XXHR/110	PC XXHR/75
Auflösung (noninterlaced)	1024 × 768	1280 × 1024	1280 × 1024	1280 × 1024	1024 × 768
Farben	16 aus 4096	16 aus 4096	16 aus 4096	256 aus 4096	256 aus 4096
Zeilenfrequenz	50 KHz	64 KHz	64 KHz	64 KHz	50 KHz
Bandbreite	64 MHz	110 MHz	110 MHz	110 MHz	75 MHz
Bemerkungen	Für PC + AT	Für PC + AT	Für PS/2 System (Microchannel)	Für PC + AT	Für PC + AT

SEYFFER+CO. AG
8048 Zürich, Hohlstr. 550, Tel. 01/62 82 00

AUSWECHSELBARE SCSI DISK PACKS



- Kapazitäten **20, 45, 70** oder **100 MBytes.**
- Direkt kompatibel zu **Macintosh Plus, SE** und **II** (mit **Zusatzkarte** kompatibel zu **IBM PC XT/AT, PS/2** usw.).
- Als externe Hard Disk Station oder als Streamer verwendbar.
- Bis 2 unterschiedliche Speichermodule pro Basis können eingesetzt werden.
- Transportierbar und auswechselbar nach Belieben.

MULTICAP SA
Av. Général-Guisan 58
1800 Vevey
Tel. 021/922 80 40

Büro in Zürich:
Bucheggstrasse 156
8057 Zürich
Tel. 01/362 33 00

Suchen Sie die Administrations-Software (Auftragsbearbeitung, Buchhaltung, etc.), die sich Ihren Wünschen anpasst? – Dann sollten Sie Miracle kennenlernen!

«Miracle»

Die neue Software-Generation für die Zukunft Ihres Unternehmens

- Zukunftsorientiert
- Modular aufgebaut
- Benutzerfreundlich
- Netzwerkfähig
- Entwickelt mit den neuesten Programmiertechniken
- Formulare, Masken, Listen, Statistiken, Farben können individuell definiert werden.
- Informationen sind über das integrierte Direkt-Informationssystem (DIS) jederzeit abrufbar. Miracle dient als Ressource für fundierte und kompetente Entscheide.

Möchten Sie mehr über diese sensationelle Software erfahren? Rufen Sie uns an! Wir sind für Sie da!

Phönix EDV-Dienstleistungen
Bahnstrasse 14
8105 Regensdorf
Telefon: 01/841 00 21 oder
01/841 01 21

Künstliche Intelligenz in der Praxis (3)

In den letzten Folgen haben wir bereits Programme kennengelernt, welche offensichtlich von einer gewissen Intelligenz geprägt sind. Alle diese Programme dienten dazu, Probleme aus der Welt der Gedankenspiele zu lösen. Damit drängt sich natürlich die Frage auf, ob wir die gesammte Problematik der KI nicht durch die Anwendung auf solche trivialen Probleme allzusehr vereinfachen. In Anbetracht der geballten Kraft, die in wissenschaftlichen und industriellen KI-Anwendungen wirksam wird, ist die Ueberlegung nicht abwegig. Wir sind jedoch der Meinung, dass sich KI-Prinzipien, also die Grundlagen, auf denen letztlich auch die stärksten solcher Anwendungsprogramme beruhen, anhand einfacher Beispiele am eindrücklichsten und leicht fassbar erklären lassen.

Beat und Fred Kipfer

In frühreren Folgen haben wir definiert: Ein Programm verhält sich dann intelligent, wenn es Probleme zu lösen vermag, welche vom Menschen Intelligenz erfordern. Auf der Basis dieser Aussage dürfen wir die bereits vorgestellten Programme ruhig als intelligent bezeichnen. Diejenigen Leser, welche die angebotenen Disketten besitzen, finden im übrigen dort noch eine Auswahl weiterer Programmbeispiele zum Studium und für Experimente.

Wir wollen gleich zu Beginn dieser neuen Folge eine andere Art von Gedankenspielen betrachten, welche ebenso zu den Klassikern der KI-Probleme gezählt werden darf. Es handelt sich dabei um eine sogenannte Logelei.

Die Aufgabenstellung lautet folgendermassen: In einem Haus (ohne bewohntes Erdgeschoss) wohnen in den Stockwerken 1 bis 4 die Herren Müller, Meier, Huber und Sutter. Diese Personen üben die vier Berufe: Maler, Bäcker, Schreiner und Mechaniker aus. Jeder von ihnen spielt ein anderes Instrument: Geige, Klavier, Pauke und Trompete.

Wir wissen von diesen Personen ausserdem folgendes:

- Herr Meier spielt Pauke.
- Herr Sutter wohnt ein Stockwerk über dem Maler.
- Der Trompetenspieler wohnt im 1. Stock.
- Herr Huber ist Bäcker.
- Der Geigenspieler wohnt zwei Stockwerke über dem Schreiner.
- Herr Müller spielt nicht Klavier.
- Der Mechaniker wohnt nicht über Herrn Sutter.

Listing 1 zeigt die Lösung des Problems in PD-Prolog. Besitzer der Disketten haben damit die Möglichkeit, das Programm auf dem eigenen Computer laufen zu lassen.

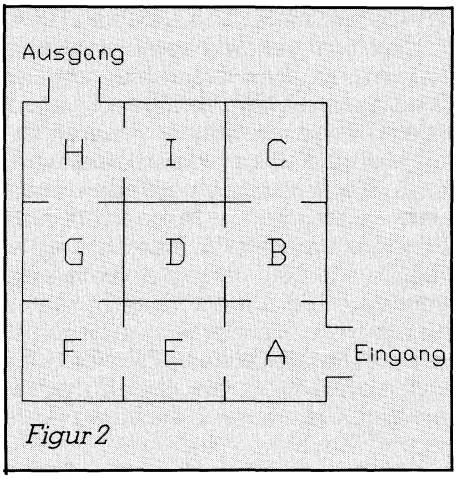
Wie geht nun der Computer bei der Lösung solcher Probleme vor? Genau genommen verhält sich der Rechner wenig intelligent. Eine Möglichkeit, die Lösung eines Problems zu finden besteht darin, am einen Ende des Zustandsdiagramms loszulaufen in der Hoffnung, den Zielzustand irgendeinmal mehr oder weniger zufällig zu finden.

In der obigen Logelei gibt es 13'824 verschiedene Kombinationsmöglichkeiten. Das ist auch der Grund, weshalb der Computer für das Finden der Lösung über PD-Prolog ziemlich viel Zeit braucht.

Betrachten wir dazu wie schon in der letzten Folge einen Irrgarten (Figur 2). Das Zustandsdiagramm kann man wie in der Figur 3 gezeigt darstellen. Es ist ziemlich schwierig, diese Form der Darstellung auf eine Maschine umzusetzen.

Wenn wir das Diagramm näher betrachten, so lässt es sich als eine Wiederholung von Verzweigungen auffassen. Aus dieser Betrachtungsweise kann man leicht eine Baumstruktur ableiten (Figur 4). In der Literatur wird oftmals der Suchbaum auch als «Folge» einer Suche definiert. Aus dieser Terminologie lässt sich nun direkt auf eine Hierarchie in den Entscheidungsknoten schliessen.

Jeder Knoten ausser dem Anfangsknoten hat genau einen einzigen Vorgänger. In der KI spricht man beim Anfangsknoten von der «Wurzel», ein Knoten ohne Nachfolger wird «Blatt» genannt. Die Verbindungen zwischen den Knoten wollen wir im weiteren «Aeste» nennen.



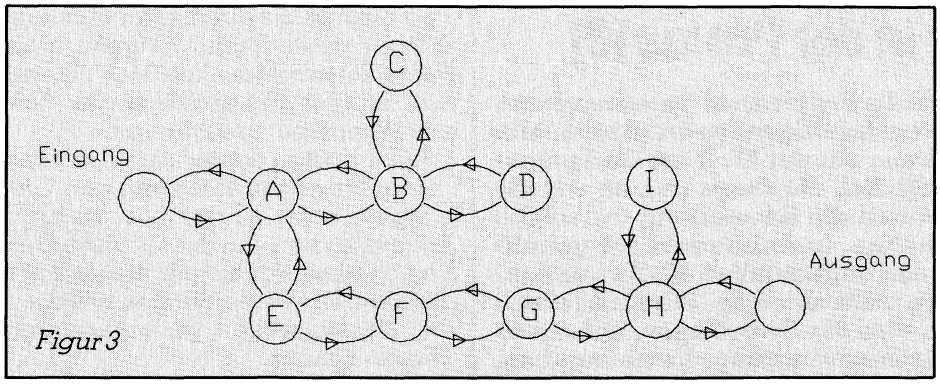
Figur 2

Die Hierarchie entspricht einer Eltern-Kindbeziehung. Knoten der selben Ebene werden «Geschwister» genannt, ihre Vorgänger «Eltern» und Ihre Nachkommen dementsprechend «Kinder».

Die meisten Denksportaufgaben und auch die meisten anderen Probleme lassen sich als Suchbaum auffassen. Mit Hilfe des Prolog-Listings aus M+K 88-4 löst der Rechner das Irrgarten-Problem wie folgt: Er geht in das Zimmer A und kontrolliert, ob er

Listing 1

```
/* Listing zum Spiel "Logelei" in PD-Prolog
   ----------------------------------------

   Aufruf : stockwerk(X).
*/


   permutiere ([],[]).
   permutiere (L,[E|L2]):- element(E,L,L1),
                           permutiere(L1,L2).

   element(E,[E|L],L).
   element(E,[E1|L],[E1|L1]):- element(E,L,L1).


   stockwerk([Meier,Mueller,Huber,Sutter,Trompete,Pauke,
              Klavier,Geige,Maler,Baecker,Schreiner,Mechaniker]):-
         permutiere([1,2,3,4],[Meier,Mueller,Huber,Sutter]),
         permutiere([1,2,3,4],[Trompete,Pauke,Klavier,Geige]),
         permutiere([1,2,3,4],[Maler,Baecker,Schreiner,Mechaniker]),

         Meier     = Pauke,
         Hilfe is Maler+1, Hilfe = Sutter,
         Trompete  = 1,
         Huber     = Baecker,

         Hilfe is Maler+1,       Hilfe = Sutter,
         Hilfe is Schreiner+2,   Hilfe = Geige,

         not(Klavier    = Mueller),
         not(Mechaniker > Sutter),nl.
```



Figur 3

bereits den Ausgang gefunden hat. Ist dies nicht der Fall, begibt er sich in das Zimmer B, und anschliessend in das Zimmer C. Nun befindet er sich in einer Sackgasse. Er geht deshalb solange zurück, bis er wieder einen weiteren Weg findet, den er bisher noch nicht gegangen ist. In unserem Fall führt das zunächst in das Zimmer B, aber auch von dort gibt es keinen weiteren Weg. Die Suche geht weiter zurück nach Zimmer A. Da sich von hier aus wieder eine Möglichkeit anbietet, geht die Suche über E nach F und nach H weiter in Richtung Ausgang (Figur 5).

Verfolgen wir diese Suche im Suchbaum, so wird ersichtlich, dass der Rechner von der Wurzel über das Geäst nach abwärts gesucht hat, also eine Suche in die Tiefe (Depth-First-Search) durchführt.

Definition der Suchtiefe

Der Verlauf dieser Form der Suche kann mit einem Kartenstapel verglichen werden. Zuunterst liegt in diesem Fall der zuerst durchlaufene «Wurzelknoten», obenauf jeweils die Kind-Knoten. Werden gültige Lösungen für die Knoten gefunden, so werden ihre Kinder jeweils oben auf den Stack gelegt (Figur 6). Kann ein Knoten nicht gelöst werden, so wird er vom Stapel entfernt und seine Geschwister werden auf weitere Kindbeziehungen untersucht. Der Stapel wird also von oben nach unten wieder abgebaut, bis sich wieder neue Lösungswege anbieten.

Die Gefahr dieser Suche liegt darin, dass man auf einem ganz falschen Pfad sucht, welcher niemals zu einer Lösung führt, dessen Knoten aber immer wieder jeweils mindestens einen gültigen Kind-Knoten aufweisen. In einem solchen Falle wächst der Stapel immer weiter an, bis er unendlich gross wird, bzw. überläuft - der Speicher des Rechners reicht nicht mehr aus.

Um das zu vermeiden, empfiehlt es sich, für die Suche in die Tiefe eine maximale «Suchtiefe» zu definieren. Die Festlegung eines solchen Grenzwertes bewirkt zugleich eine gewisse Optimierung des Suchweges. Lange, umständliche Wege werden vom System abgebrochen.

Prolog arbeitet grundsätzlich mit dieser Tiefen-Suche. Findet man auf diese Weise eine Lösung, dann ist es kinderleicht, den Lösungsweg über den vorhandenen Stapel zurückzuverfolgen. Der Umstand, dass sich jeweils nur noch die für die Lösung relevanten Knoten auf dem Stapel befinden, hat den willkommenen Nebeneffekt, dass die Lösung mit vernünftigem Speicherplatzbedarf gefunden werden kann.

Weiss man allerdings, dass sich die Lösung eines Problems innerhalb der ersten Ebenen finden lässt, so lässt sich die Suche in die Tiefe bestimmt nicht als optimale Strategie bezeichnen. Erst recht nicht, wenn bekannt ist, dass es innerhalb der bestehenden Suchmöglichkeiten Pfade gibt, welche auf einen unendlich tiefen Weg ohne Lösung führen.

In diesem Fall würde man die Suche in die Breite vorziehen. Bei dieser Art der Suche werden die Knoten Ebene für Ebene abgearbeitet und keine Ebene wird verlassen, bevor nicht alle ihre Knoten untersucht worden sind.

Am besten vergleichen wir wiederum am Beispiel des Labyrinths, wie die Suche mit Hilfe dieser Strategie vor sich gehen würde (Figur 7).

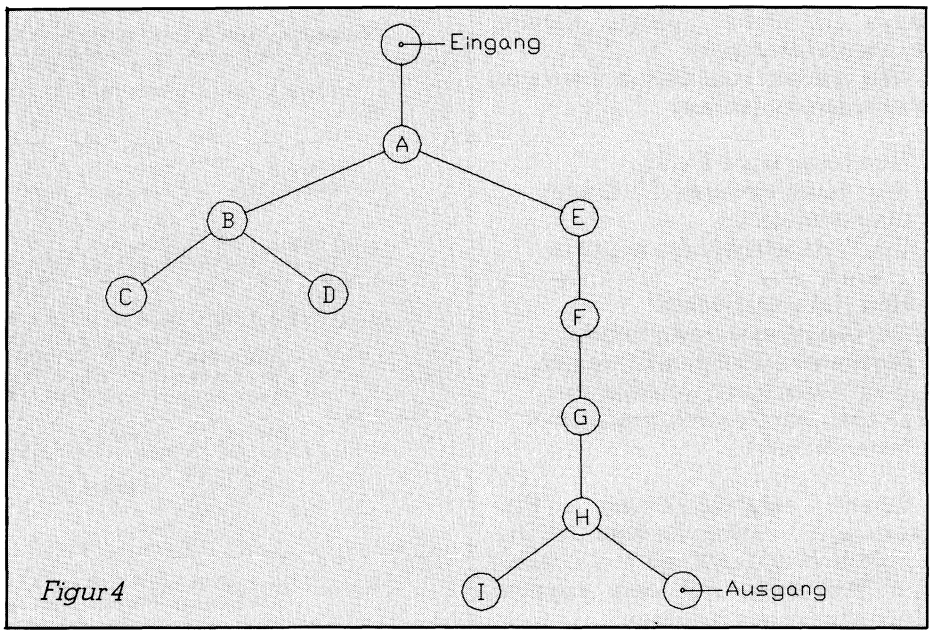
Zuerst begibt sich der Rechner in Zimmer A und testet, ob er sich bereits am Ziel befindet. Da dies nicht der Fall ist, begibt er sich anschliessend in alle Zimmer, welche er direkt von A aus erreichen kann also, in Zimmer B und E. Nun untersucht er die Möglichkeiten von B und E aus in andere Zimmer zu gelangen. Diese Methode der Untersuchung stellt er solange an, bis er die Lösung gefunden hat.

Wie man leicht einsieht, hat der Rechner in diesem Fall wesentlich mehr Knoten durchlaufen und gelöst, als es für das Erreichen des Zieles notwendig gewesen wäre. Die Suche war demzufolge in diesem Fall auch ganz bedeutend speicherintensiver als bei der Suche in die Tiefe.

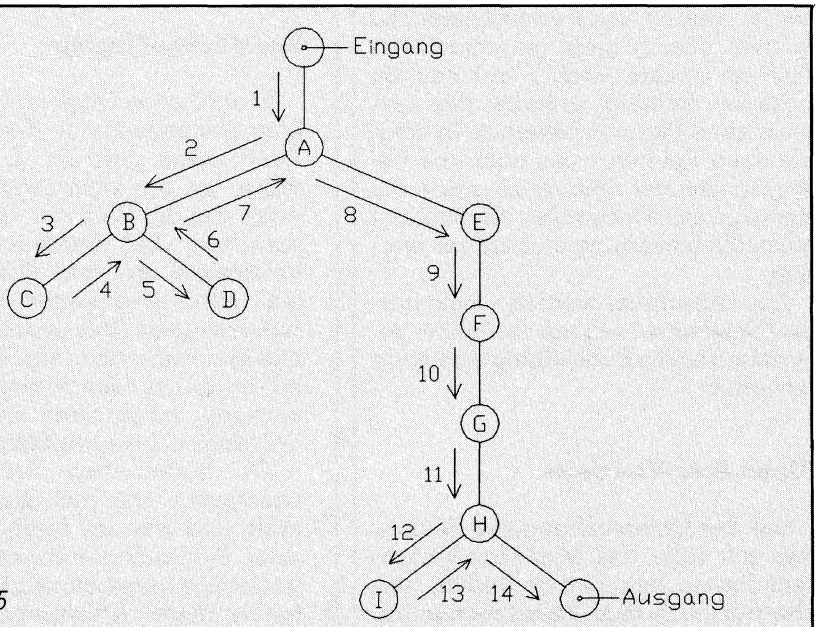
Der entscheidende Vorteil dieser Art der Suche liegt darin, dass der Rechner in jedem Fall den kürzesten Weg zur Lösung findet. Sie nimmt aber auch in den meisten Fällen sehr viel mehr Zeit in Anspruch, als die Depth-First-Suche.

Gesteuerte Suche

Die bisher beschriebenen Suchmethoden führen rein zufällig zur Lösung. Gelänge es uns, die vorhandenen Suchwege zu bewerten, bevor sie beschritten werden, so liesse sich schneller und sicherer ein optimaler Lösungsweg durch den Suchbaum



Figur 4



Figur 5

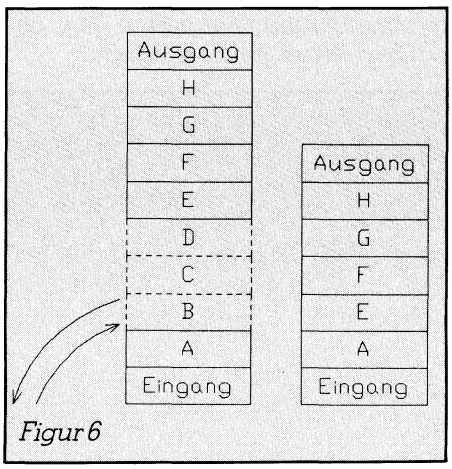
finden. Von Vorteil wären zu diesem Zwecke zwei Parameter:

1) Wie weit haben wir uns vom ursprünglichen Zustand entfernt?
2) Wie weit sind wir noch vom Ziel entfernt?

Der erste Parameter steht während der Suche stets zur Verfügung. Er ist identisch mit der Suchebene.

Der zweite Parameter ist eher schwierig festzustellen. Wir wissen meist nicht, wie weit wir noch vom Ziel entfernt sind und auch nicht, ob wir uns überhaupt auf einem erfolgversprechenden Weg befinden.

Deshalb wird es nützlich sein, sich am Ziel des Problemes zu orientieren, indem mit Hilfe einer Faustregel verbleibende Distanz zwischen dem erreichten Zustand und dem Ziel abgeschätzt wird. Der beste Weg wird der sein, bei welchem sich bei einem möglichst kurzen bereits zurückgelegten Weg (erster Parameter) der geringste Wert für den zweiten Parameter ergibt.



Figur 6

Der Erfolg der auf diese Weise gesteuerten Suche hängt extrem von der «Qualität» der Faustregel ab. Im in den letzten beiden Folgen besprochenen «Zahlenspiel» wird eine gesteuerte Suche sehr oft in eine Sackgasse führen, weil in diesem Spiel notwendigerweise immer wieder durch Unordnung überhaupt die Möglichkeit geschaffen werden muss, einen einzelnen Stein an die für ihn vorgesehene Position zu schieben.

Die Gestaltung einer wirksamen Faustregel für diese Problem scheint infolgedessen äusserst schwierig zu sein.

Eine weitere Möglichkeit die Suche erfolgreicher zu gestalten liegt darin, die Kinder entsprechend zu ordnen. Wir könnten Kinder, welche wiederum gemäss einer Faustregel eher zum Ziel führen, vorne auf den Stapel bringen und so eine wirksamere Suche gewährleisten.

Diese Art der Optimierung lässt sich beispielsweise in lernfähigen Systemen sehr aussichtsreich einsetzen, indem die «Kinder» anhand einer Erfolgsstatistik geordnet werden.

Auf dem Gebiet der KI werden Steuerungsmechanismen, wie sie hier beschrieben worden sind, als «Heuristik» bezeichnet.

Spielbäume

Auch ein Spiel zwischen mehreren Spielern kann als Baum aufgefasst werden. Dessen Wege führen entweder zum Gewinn oder zum Verlust der Partie.

Bei den bisher besprochenen Suchen wurde das Problem jeweils zu Ende gedacht. Wenn es sich um die Lösung von komplexeren Problemen handelt, führt diese Vorgehensweise zu sogenannten kombinatorischen Explosionen.

Ein Spielbaum ist weder ein reiner UND-, noch ein reiner ODER-Baum. Ein Spiel kann nur dann gewonnen werden, wenn man erstens richtig spielt und zweitens der Gegner den entsprechenden, für unsere Strategie günstigen Zug macht.

Für unseren Zug ergibt sich also eine ODER-Erweiterung, denn wir

Wie entstehen kombinatorische Explosionen?

Um nachzuvollziehen, wie eine kombinatorische Explosion zustande kommt, betrachten wir uns nochmals die Zugmöglichkeiten im Schachspiel, denn hier ist es sehr schnell möglich, auch sehr grosse und leistungsfähige Rechner zu überfordern.

Um den ersten Zug auszuführen, kann «Weiss» aus 16 Bauernzügen und 4 Springerzügen wählen, also gesamthaft aus 20 möglichen Zügen. Auf jeden dieser möglichen Züge kann «Schwarz» wiederum mit derselben Auswahl von 20 Zügen antworten. Kombinatorisch sind also nach den ersten beiden Zügen bereits 400 Spielstellungen möglich.

Auf diese 400 möglichen Spielstellungen kann nun «Weiss» wiederum mit einer Auswahl von etwa 20 Zügen reagieren (je nachdem, was «Weiss» als erstes gezogen hat, können dies aber auch bis zu 30 auswählbaren Züge sein). Das Gleiche gilt, wenn nun wieder «Schwarz» am Zuge ist.

Der Einfachheit halber wollen wir bei den weiteren Zügen mit 20 Möglichkeiten weiter rechnen, das Ergebnis ist auch so noch beeindruckend genug. Das ergibt für den vierten Zug von «Weiss» und «Schwarz» 25'600'000'000 Kombinationen.

Was es bedeuten könnten, diese Zahl, sowie alle vorhergehenden und nachfolgenden in notwendigen Arbeitsspeicherbedarf umzurechnen, können Sie abschätzen. Daraus lässt sich aber andererseits auch die Wichtigkeit der Heuristik im Schachcomputer erkennen - ohne sie könnte man solche Geräte derzeit gar nicht bauen.

können so oder so ziehen. Der Zug unseres Gegenübers kommt jedoch als UND-Verknüpfung zur Geltung, da wir auf diesen keinen Einfluss ausüben können.

Die Spielstrategie erfordert nun, dass wir jedesmal einen Zug spielen müssen, welcher uns zum Gewinner macht oder, wenn dies nicht möglich ist, wenigsten zu einem Unentschieden führt. Führen alle Knoten zum Verlust des Spieles, so können wir die Partie aufgeben.

Eine weitverbreitete Strategie zur Lösung von Spielproblemen ist die Mini-Max-Methode. Betrachten wir den Spielbaum aus Figur 8. Die Züge entsprechen den vorhandenen Knoten. Die Knoten sind jeweils mit einer Zahl bewertet. Will der Rechner das Spiel gewinnen, so verfolgt er immer den Weg, welcher für ihn das Maximum und für den Gegner das Minimum aus den jeweiligen Kindknoten ergibt.

Wie wir im Verlauf dieser Theorie festgestellt haben, lässt sich ein so komplexes Spiel wie Schach nicht in seiner Gesamtheit untersuchen. Wir sind deshalb genötigt, nur einzelne relevante Teile des ganzen Baumes zu untersuchen. Konkret bedeutet dies, dass der Rechner jede Stellung als «Ausgangsposition» betrachten muss und sich nicht mehr für Vergangenes interessieren darf.

Vernünftigerweise untersucht man das Spiel bis zu einer Position, bei welcher ein gegnerischer Zug keine spielentscheidende Wirkung mehr hat. Diese Positionen werden im allgemeinen «tote Stellungen» genannt.

In unserem «Mini-Suchbaum» sind diese Stellungen die Blätter. Wie bereits beschrieben, müssen diese bewertet werden, um sie vergleichen zu können. Hierzu dient im einfachsten Fall ein Punkte-Modell, welches die aktuelle Stellung anhand der vorhandenen Figuren bewertet. In komplexeren Spielen muss auch die Beweglichkeit der Spielsteine sowie die strategische Wirksamkeit der Stellung mit in die Bewertung einbezogen werden.

Zum Abschluss werden Computer- und Spielerstellung ins Verhältnis gesetzt und so die Endstellung endgültig bewertet.

Alpha-Beta-Verfahren

Bei der Untersuchung der Stellungen mit Hilfe des Mini-Max-Verfahrens lassen sich Zweige finden, welche nur noch eine Verschlechterung der momentanen Stellung ergeben können.

Führt ein Zug zu Verschlechterung der momentanen Stellung, so muss er nicht weiter berechnet werden, weil der Zug des Gegners auch zur schlechten Seite für den Spieler tendiert. Die Suche wird in diesem Fall abgebrochen. Man nennt dies Alpha-Schnitt. Die gleiche Problematik von der Gegnerseite betrachtet wird Beta-Schnitt genannt.

Beide Verfahren dienen dazu, die Rechenzeit zu verkürzen, Speicherplatz zu sparen und dadurch den Rechner zu entlasten.

Zum Abschluss der heutigen Folge können Sie das Listing 2 in PD-Prolog in Ihren Computer eingeben. Es handelt sich dabei um das bekannte Spiel Tic-Tac-Toe. Versuchen Sie zu erkennen, wie der Rechner vorgeht, um das Spiel zu gewinnen.

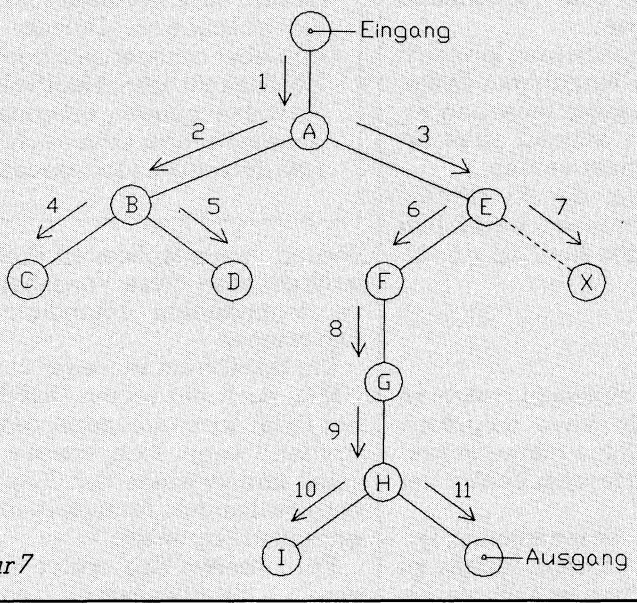
Kannibalen-Problem

In der letzten Folge hatten wir kurz das sogenannte Kannibalen-Problem gestreift. Es geht dabei um ein klassisches Problem aus der Welt der Gedankenspiele, bei welchem drei Missionare und drei Kannibalen unter erschwerten Bedingungen einen Fluss überqueren müssen, ohne dass die Kannibalen dabei Gelegenheit bekommen, einen oder mehrere der Missionare aufzufressen.

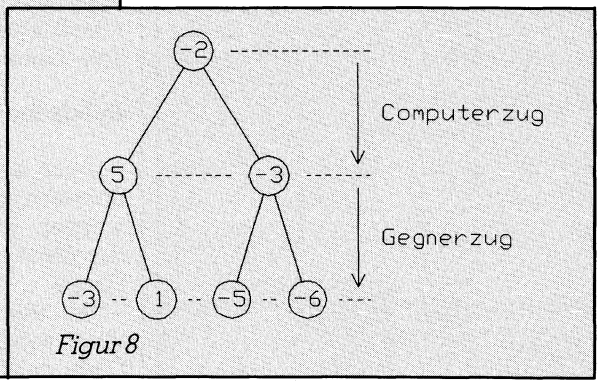
Wir haben dazu Anfragen bekommen, wie das Spiel abläuft, und wie das Listing aussieht. Beides finden Sie auf den im Service angebotenen Disketten zu dieser Artikelseire, welche Sie mit den Karten hinten im Heft bestellen können, und zwar in den Paketen 1 und 2.

Das Spiel läuft grafisch mit musikalischer Begleitung ab, je nach Ihrer Computerausrüstung farbig oder monochrom. Das Listing ist in PD-Prolog vorhanden, so dass Sie auf dem Computer die verschiedenen Lösungsmöglichkeiten austesten können. Ebenfalls in PD-Prolog auf diesen Disketten vorhanden ist das Spiel der Türme von Hanoi - ebenfalls ein Klassiker, der immer wieder bei Prolog-Anwendungen demonstriert wird, obwohl das Programm nicht unbedingt als typisch bezeichnet werden kann.

Wir sind uns bewusst, dass diese Folge ziemlich viel graue Theorie zum Inhalt hatte, und wir haben diese deshalb auch so kurz wie möglich gehalten. Leider sind solche Passagen nicht ganz zu umgehen, wenn es gilt, die Hintergründe für ganz bestimmte, nicht ohne weiteres erklärbare Programmablaufssteuerungen und deren Auswirkungen zu erklären.



Figur 7



Figur 8

Listing 2

```
/* Listing zum Spiel "Tic-Tac-Toe" in PD-Prolog
   -------------------------------------------

   Aufruf : spiel('o').  : der Spieler beginnt
            spiel('x').  : der Computer beginnt
*/

   spiel(Begin):-nl,nl,nl,
                 write('Ziel dieses Spieles ist es, alle drei Felder'),nl,
                 write('einer Reihe, Spalte oder Diagonale zu besetzen.'),nl,
                 write('o --> Spieler  ;  x --> Computer'),nl,nl,
                 write('ACHTUNG: Schliessen Sie die Eingabe eines Zuges'),nl,
                 write('         immer mit einem Punkt ab !!'),nl,nl,
                 write('Spielfeld :'),nl,nl,
                 ausgabe(['1','2','3','4','5','6','7','8','9']),
                 vorgang([' ',' ',' ',' ',' ',' ',' ',' ',' '],Begin,0),nl,nl

   vorgang(Liste,_,Anzahl):-linie(L),
                     test(L,Liste,'x'),
                     write('Ich habe nach ',Anzahl,' Zuegen gewonnen.'),nl.

   vorgang(Liste,_,Anzahl):-linie(L),
                     test(L,Liste,'o'),
                     write('Sie haben nach ',Anzahl,' Zuegen gewonnen.'),nl

   vorgang([A,B,C,D,E,F,G,H,I],_,_):-A\=' ',B\=' ',C\=' ',D\=' ',E\=' ',
                                     F\=' ',G\=' ',H\=' ',I\=' ',
                                     write('Spiel unentschieden').

   vorgang(Liste,'o',Anzahl):-Zahl is Anzahl+1,
                              einlesen(Liste,Neu_Liste,0),
                              ausgabe(Neu_Liste),!,
                              vorgang(Neu_Liste,'x',Zahl).

   vorgang(Liste,'x',Anzahl):-Zahl is Anzahl+1,
                              verteilung(V),
                              linie(L),
                              not(spezial(Anzahl,L)),
                              zug_suchen(V,L,Liste,Neu_Liste),
                              ausgabe(Neu_Liste),!,
                              vorgang(Neu_Liste,'o',Zahl).

   spezial(Zug,[_,_,_,1]):- Zug\=3.

   ausgabe([A,B,C,D,E,F,G,H,I]):-write('     ',A,'|',B,'|',C),nl,
                                 write('     ----+-----+----'),nl,
                                 write('     ',D,'|',E,'|',F),nl,
                                 write('     ----+-----+----'),nl,
                                 write('     ',G,'|',H,'|',I),nl,nl.

   einlesen(Liste,Neu_Liste,Ort):-stelle(Ort,Liste,'o',1,Neu_Liste,' ').
   einlesen(Liste,Neu_Liste,_)  :-write('Zug eingeben : '),
                                  read(X),nl,
                                  einlesen(Liste,Neu_Liste,X).

   zug_suchen(['x',Y,Z],[U,V,W,_],Liste,Neu_Liste):-
                                 stelle(U,Liste,'x',1,Neu_Liste,' '),
                                 stelle(V,Liste,Y,1,_,Y),
                                 stelle(W,Liste,Z,1,_,Z).

   zug_suchen([X,'x',Z],[U,V,W,_],Liste,Neu_Liste):-
                                 stelle(U,Liste,X,1,_,X),
                                 stelle(V,Liste,'x',1,Neu_Liste,' '),
                                 stelle(W,Liste,Z,1,_,Z).

   zug_suchen([X,Y,'x'],[U,V,W,_],Liste,Neu_Liste):-
                                 stelle(U,Liste,X,1,_,X),
                                 stelle(V,Liste,Y,1,_,Y),
                                 stelle(W,Liste,'x',1,Neu_Liste,' ').

   zug_suchen(['?','?','?'],[_,_,_,_],Liste,Neu_Liste):-
                                 stelle(V,Liste,'x',1,Neu_Liste,' ');
                                 stelle(U,Liste,'x',1,Neu_Liste,' ');
                                 stelle(W,Liste,'x',1,Neu_Liste,' ').

   test([X,Y,Z,0],Liste,Zeichen):-stelle(X,Liste,' ',1,_,Zeichen),
                                 stelle(Y,Liste,' ',1,_,Zeichen),
                                 stelle(Z,Liste,' ',1,_,Zeichen).

   stelle(X,[Test|Rest],Stein,X,[Stein|Rest],Test).
   stelle(X,[K|Rest],Stein,Y,[K|N_Liste],Test):-
                                       Ort is Y+1,
                                       stelle(X,Rest,Stein,Ort,N_Liste,Test)

   linie([5,4,6,0]).
   linie([5,2,8,0]).
   linie([1,5,9,0]).
   linie([3,2,1,0]).
   linie([9,8,7,0]).
   linie([3,5,7,0]).
   linie([7,4,1,0]).
   linie([9,6,3,0]).
   linie([1,2,4,1]).
   linie([4,7,8,1]).

   verteilung(['x','x','x']).
   verteilung(['o','o','x']).
   verteilung(['o','x','o']).
   verteilung(['x','o','o']).
   verteilung(['x',' ','x']).
   verteilung(['x',' ',' ']).
   verteilung(['?','?','?']).
```

PD-Prolog / Turbo-Prolog

In unserer KI-Serie wurde und wird immer wieder auch Turbo-Prolog erwähnt. Für denjenigen, der diese KI-Sprache nicht kennt, wirft das natürlich die Frage auf, was denn für den Anwender Turbo-Prolog offensichtlich von PD-Prolog unterscheidet, und weshalb unsere Serie nicht auf Turbo-, statt auf PD-Prolog aufgebaut ist.

Für Letzteres war primär entscheidend, dass PD-Prolog frei auf dem Public-Domain-Markt erhältlich ist und folglich mit einigen von uns mitangebotenen bequemen Oberflächen-Utilities und erklärenden Handbüchern und Beispielen für einen geringen Preis angeboten werden konnte. Die Experimental-Praxis, ein Kerngehalt unserer Serie, und damit das Sammeln eigener Erfahrungen sollte mit geringem Aufwand ermöglicht werden. Das liess sich mit PD-Prolog realisieren.

Wir haben dabei in Kauf genommen, dass PD-Prolog vergleichsweise langsam abläuft und beispielsweise keine Trace-Funktion und daher auch kein bequemes «debuging» anbietet.

Turbo-Prolog ist eine von Borland speziell für PC-Anwendungen geschaffene KI-Sprache mit einer ausserordentlich bequemen Bedieneroberfläche, Compiler, Trace-Funktion und sehr viel schnellerem Ablauf. Für die Lösung der Logelei dieser Folge braucht Trubo-Prolog nur wenige Augenblicke, PD-Prolog vergleichsweise aber doch nahezu eine Minute Rechenzeit. Dazu kann der Suchablauf bei Turbo-Prolog ohne spezielle Programm-Manipulationen verfolgt werden.

Man hat uns darauf aufmerksam gemacht, und wir sind uns auch klar darüber, dass manche Leser dieser Serie den Praxisteil lieber über Turbo-Prolog abwickeln möchten, umsomehr, als ja auch diese Sprache nicht besonders teuer zu erwerben ist. Wir kommen dem Servicewunsch entgegen und bieten ab sofort auch Turbo-Prolog und dessen Toolbox für Interessenten an. Zum Bestellen dient wie immer die Karte hinten im Heft.

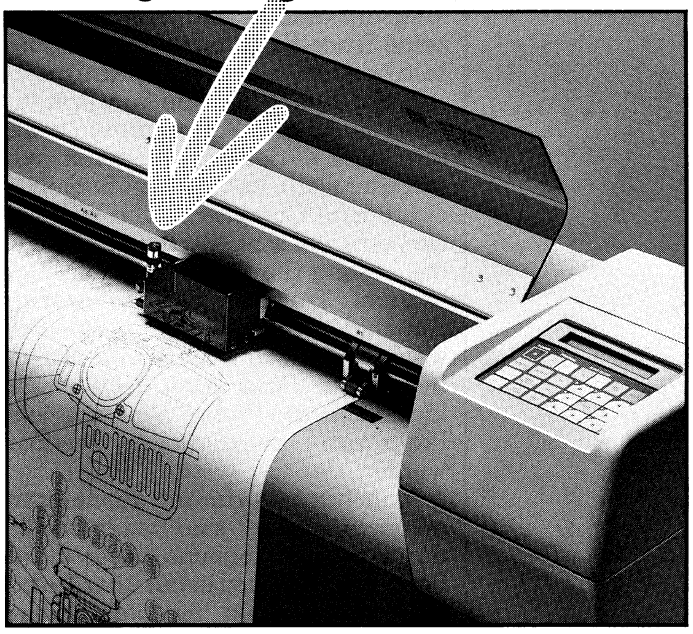
Wir möchten aber an dieser Stelle darauf hinweisen, dass die in PD-Prolog geschriebenen und auf Disketten angebotenen Listings an Turbo-Prolog angepasst werden müssen, bevor sie dort laufen. Das dürfte zwar nach dem Studium des hervorragenden Turbo-Prolog-Handbuches nicht schwierig sein, muss aber zumindest im derzeitigen Stadium der Serie jeweils vom Leser selbst vorgenommen werden. □

MUTOH?
MUTOH?
MUTOH!

Die bewährte Plotterfamilie von Mutoh wurde um ein neues Mitglied erweitert: den ip-500 Personal Plotter für Formate DIN A4 bis DIN A1.

Die hohe Auflösegenauigkeit des ip-500 beim Plotten gewährleistet eine hohe Zeichenqualität. Selbstverständlich bei Mutoh, dass nebst Tusche und Tintenrollern auch Bleistiftminen verwendet werden können.

Die Plottzeiten werden beim ip-500 dank seiner Doppel-Prozessor-Architektur und einer hohen Eigenintelligenz drastisch verkürzt.



Befehlssprache ip-H (kompatibel HP-GL*), Plottgeschwindigkeit 500 mm/s, Beschleunigung 1.5 G, Auflösung 0,025 mm.

* HP-GL eingetragenes Warenzeichen von HP

MUTOH INDUSTRIES LTD.

Racher

COMPUTER GRAFIK SYSTEME

Hard- und Software für CAD und grafische Datenverarbeitung

Racher & Co AG, 8919 Rottenschwil
Telefon 057 34 19 12 oder 057 34 19 13

BOCA

Erweiterungskarten für IBM PS/2™, IBM XT/AT™ und kompatible Systeme

BOCARAM XT 1 MB EMS-Speicherkarte (0 KB)	**sFr. 249.–**
BOCARAM AT 2 MB EMS-Speicherkarte (0 KB)	**sFr. 349.–**
BOCARAM AT 2 MB EMS-Zusatzkarte (0 KB)	**sFr. 189.–**
BOCARAM PS2 25/30 2 MB EMS-Speicherkarte (0 KB)	**sFr. 299.–**
BOCARAM PS2 50/60 4 MB EMS-Speicherkarte (0 KB)	**sFr. 449.–**
BOCA I/O XT Multifunktionskarte (1×P, 1×S, Uhr, Kalender)	**sFr. 199.–**
BOCA I/O XT Gameport-Zusatzkarte	**sFr. 59.–**
BOCA I/O AT Schnittstellenkarte (1×P, 2×S)	**sFr. 179.–**
BOCA I/O PS2 25/30 Schnittstellenkarte (1×P, 2×S)	**sFr. 179.–**
BOCA I/O PS2 50/60 Schnittstellenkarte (1×P)	**sFr. 189.–**
BOCA I/O PS2 50/60 Schnittstellenkarte (1×P, 2×S)	**sFr. 329.–**
BOCA MULTI-EGA 640×480 Autoswitch-Grafikkarte	**sFr. 349.–**

Best **O**ption **C**ards **A**vailable

Generalvertretung

ETP

Rudolf Schmid
Beckenhofstr. 10
CH-8006 Zürich
Tel. 01/362 88 78

Elektronische + Technische Produkte

●● NEUERÖFFNUNG 1. SEPTEMBER 1988 ●●

COMPUTER

NTR Computer für jede Leistungsklasse

XT-225 10 MHZ — Fr. **990.–**
256KB RAM, FDD, Ser., Herc., 102 Tast.

XT-225/30 10 MHZ — Fr. **1 990.–**
640KB RAM, 30MB HDD, 14"-Monitor

AT-250/40 16 MHZ — Fr. **3 490.–**
640KB RAM, 40MB HDD 14"-Monitor

Laptop XT-20 10 MHZ — Fr. **1990.–**
640KB RAM, 2×FDD, LCD, 640×200 + 720×348

Laptop AT-20 10 MHZ — Fr. **3 290.–**
640KB RAM, 2× FDD, LCD 640×200 + 720×348

QMAUS QM-E1 — Fr. **125.–**
Microsoft ser. kompatible

Datentransfer zwischen verschiedenen Computern und Laufwerken wird mit dem mitgelieferten Utility ein Kinderspiel.

Werkstatt für IBM und kompatible

NEUTRONIC AG

Kronenacker 6–8 ●●● 053/33 35 36
● **NTR** ● CH-8200 Schaffhausen ● **NTR** ●

Einführung in Turbo BASIC (3)

In diesem Beitrag beschäftigen wir uns mit dem Schreiben von Prozeduren und dem Einsatz der Rekursion in Turbo BASIC. Damit betreten wir echtes Neuland in BASIC, denn diese Domäne war bis jetzt nur Pascal, Modula und anderen modernen Hochsprachen vorbehalten.

Marcel Sutter

3. Kapitel: Die Sprachelemente von Turbo BASIC (2)

In diesem Kapitel beschäftigen wir uns mit Unterprogramm-Strukturen und rekursiven Aufrufen von Unterprogrammen.

Im klassischen interpretativen BASIC gibt es nur zwei Unterprogramm-Strukturen:

1. Die einzeilige Funktionsprozedur

Beispiele:

```
DEF FNF(X)=50*EXP(X)*COS(-2*X/15)
DEF FNR(Y)=INT(100*Y+0.5)/100
DEF FNB(W)=W*3.14159/180
```

2. Der Aufruf eines (unechten) Unterprogramms mit GOSUB n

Beispiel:

```
.....
500 GOSUB 1000
510 .....

.....
1000 Anweisung(en)
1010 Anweisung(en)
.....
1200 RETURN
```

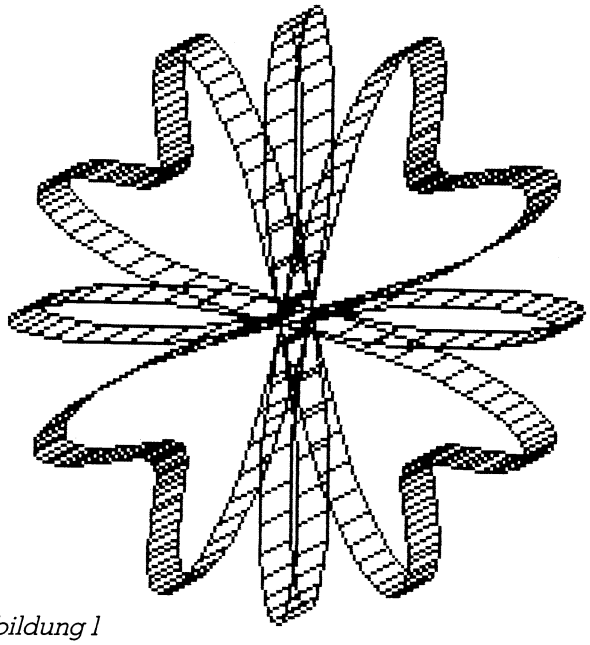


Abbildung 1

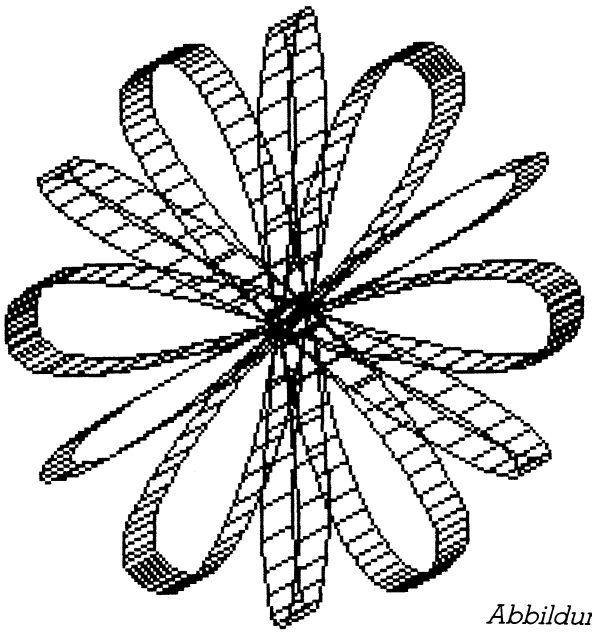


Abbildung 2

Im klassischen BASIC sind alle Variablen global. Eine Veränderung des Wertes der Variablen A im Unterprogramm bewirkt automatisch die gleiche Veränderung aller Werte aller Variablen A im Hauptprogramm, da es zu einem gegebenen Variablennamen A genau 1 Speicherplatz im Computer gibt.

Die Variable X (resp. Y,W) in der einzeiligen Funktionsprozedur ist ein sogenannter formaler Parameter, oft auch Dummy-Variable genannt. Sie repräsentiert keinen Speicherplatz im Computer sondern erhält beim Funktionsaufruf den aktuellen Wert, der zur Funktionsberechnung von FNF(X) benützt wird. Danach kennt das Programm den Wert des formalen Parameters X nicht mehr.

Turbo BASIC kennt ausser diesen beiden Möglichkeiten die fortgeschritteneren Möglichkeiten der mehrzeiligen Funktionsprozedur und der echten Prozedur mit lokalen Variablen.

13. Einzeilige Funktionsprozedur

Sie hat die aus dem interpretativen BASIC bekannte Form:

```
DEF FNFunktionsname(Parameterliste) = Ausdruck
```

Der freigewählte Funktionsname darf 31 Zeichen lang sein und muss den Regeln für Variablennamen gehorchen. Die einzelnen formalen Parameter müssen durch Kommas voneinander getrennt sein. Der Ausdruck ist eine arithmetische oder String-Anweisung mit den formalen Parametern, die beim Aufruf der Funktionsprozedur abgearbeitet wird. Der Wert wird nach aussen, d.h. an die aufrufende Stelle übergeben.

Beispiele:

1.
```
DEF FNKegelvolumen(R,H) = 3.14159*R*R*H/3
```

   Der Aufruf kann beispielsweise so erfolgen:
```
V=FNKegekvolumen(12,18) : PRINT V
```

2.
```
DEF FNVerdoppeln$(TEXT$) = TEXT$ + « » + TEXT$
```

   Hier erfolgt ein Aufruf etwa mit:

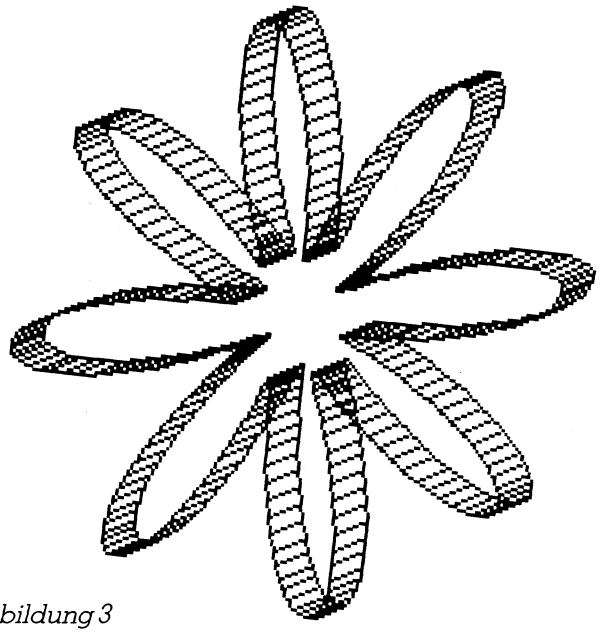


Abbildung 3

PRINT FNVerdoppeln$(«Turbo BASIC»)
Auf dem Schirm steht Turbo BASIC Turbo BASIC

14. Die mehrzeilige Funktionsprozedur

Sie stellt eine Erweiterung des klassischen BASICs dar. Die allgemeine Form lautet:

```
DEF FNFunktionsname(Parameterliste)
        [LOCAL Variablenliste]  optional
        [STATIC Variablenliste] optional
        [SHARED Variablenliste] optional
        Anweisung(en)
        .....
        [EXIT DEF]              optional
        Anweisung(en)
        FNFunktionsname=Ausdruck
END DEF
```

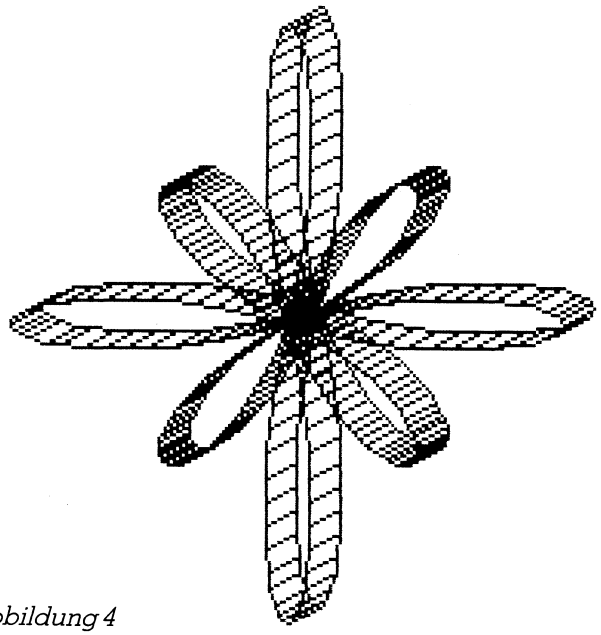


Abbildung 4

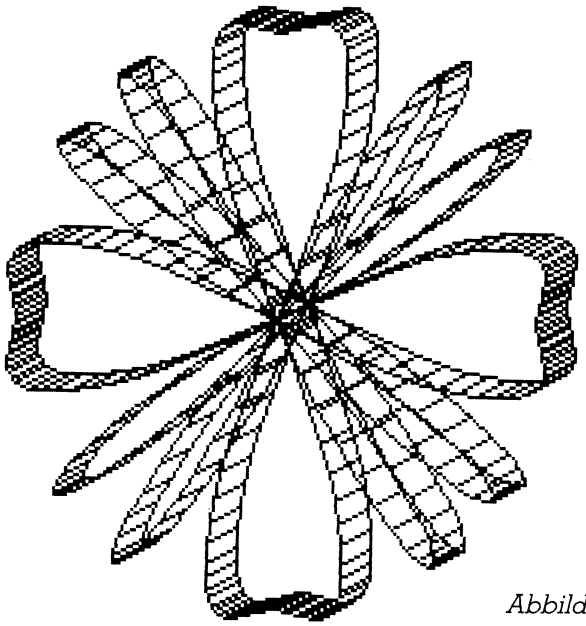


Abbildung 5

Statt langer Theorie geben wir ein Beispiel, bei dem alle obigen Möglichkeiten ausgeschöpft werden.

Die Berechnung von n! = 1*2*3*.....*(n-1)*n gehört nicht zum Sprachumfang von Turbo BASIC. Wir schreiben daher für n! eine mehrzeilige Funktionsprozedur:

```
DEF FNFakultaet#(n%)
        LOCAL J%,FAK#
        IF N%<0 OR N%>170 THEN
           FNFakultaet#=-1
           EXIT DEF
        ELSE
           FAK#=1
           FOR J%=2 TO N%
                FAK#=FAK#*J%
           NEXT J%
           FNFakultaet#=FAK#
        END IF
END DEF
```

Mit dem folgenden kleinen Hauptprogramm können Sie bequem Fakultäten berechnen:

```
CLS
INPUT«Geben Sie n ein (0-170).....»;N%
Y#=FNFakultaet#(N%)
IF Y#=-1 THEN
        PRINT«Die Fakultät ist nicht berechenbar»
ELSE
    PRINT«Die Fakultät von»;N%;«ist»;Y#
END IF
END
```

Geben Sie n=200 ein, dann erhalten Sie die Fehlermeldung

'Die Fakultät ist nicht berechenbar'.

Tippen Sie n=100 ein, dann erhalten Sie das beachtliche Ergebnis

'Die Fakultät von 100 ist 9.33262154439441E+157'.

Die Programmstruktur der mehrzeiligen Funktionsprozedur erklärt sich weitgehend selbst. Neu ist hier nur die Variablen-Deklaration LOCAL J%, FAK#. Mit dieser Anweisung sorgen wir dafür, dass während des Funktionsaufrufes kurzfristig zwei Speicherplätze J% und FAK# zur Aufnahme von Zahlen bereitgestellt werden. Sobald der Funktionsaufruf beendet ist, werden die beiden Speicherplätze wieder gelöscht und freigegeben.

Wenn im Hauptprogramm ebenfalls die Variablen J% und FAK# vorkommen sollten, dann werden deren Werte nicht verändert. Die lokalen Variablen J% und FAK# der Funktionsprozedur sind an anderer Stelle des Speichers untergebracht als die globalen Variablen J% und FAK# des Hauptprogramms. Die Definition von lokalen Variablen in Prozeduren ist das entscheidend Neue in Turbo BASIC, welches nun endlich auch BASIC voll prozedural

Listing 1

```
def fnr(phi,n)
    select case n
           case 1
           fnr=cos(4*sin(2*phi))
           case 2
           fnr=cos(3*cos(3*phi))
           case 3
           fnr=0.4*sin(8*phi)+0.6
           case 4
           fnr=0.5*cos(5*sin(2*phi))+0.5
           case 5
           fnr=sin(5*cos(2*phi))
    end select
end def

'Menü ---------------------------------------------------
menue:
screen 0 : width 80 : cls
print"5 Kurven in Polarkoordinaten"
print"---------------------------"
print : print : print
print"Was wolllen Sie?" : print
print tab(10);"1 ..... 1. Kurve"
print tab(10);"2 ..... 2. Kurve"
print tab(10);"3 ..... 3. Kurve"
print tab(10);"4 ..... 4. Kurve"
print tab(10);"5 ..... 5. Kurve" : print
print tab(10);"0 ..... Programm beenden"
locate 20,1
input"Geben Sie die entspr. Zahl ein (0-5) ........";n
if n=0 then end
if n<0 or n>5 or n<>int(n) then
   print"falsche Eingabe!"
   stop
end if

'Hauptprogramm ------------------------------------------
screen 1 : color 1,1
rad=3.14159/180

for w=0 to 360
    phi=w*rad : r=90*fnr(phi,n)
    x=160+r*cos(phi) : y=100-r*sin(phi)
    x2innen=x      : y2innen=y
    x2aussen=x+10 : y2aussen=y-5
    if phi=0 then
       line(x2innen,y2innen)-(x2aussen,y2aussen),3
    else
       line(x1innen,y1innen)-(x2innen,y2innen),3
       line(x1aussen,y1aussen)-(x2aussen,y2aussen),3
       line(x2innen,y2innen)-(x2aussen,y2aussen),3
    end if
    x1innen=x2innen   : y1innen=y2innen
    x1aussen=x2aussen : y1aussen=y2aussen
next w

beep
locate 1,1 : print"Taste";
taste$=""
while taste$=""
      taste$=inkey$
wend
goto menue
```

wie LOGO, Pascal, Modula und andere moderne Hochsprachen macht.

Sie können den Unterschied zwischen lokalen und globalen Variablen leicht an obigem Fakultätsprogramm überprüfen. Wenn Sie im Hauptprogramm nach der Anweisung CLS die Anweisungen J%=-100 und FAK#= -200 einfügen und vor der letzten Anweisung END die Anweisung PRINT J% ; FAK# schreiben, dann werden Sie sehen, dass die Werte -100 und -200 der globalen Variablen J% und FAK# nicht verändert werden, obwohl innerhalb der Funktionsprozedur die lokalen Variablen J% und FAK# unterschiedliche Werte durchlaufen!

Wir besprechen diesen Sachverhalt deshalb so ausführlich, weil wir beim nachfolgenden Abschnitt über Prozeduren die gleiche Programmiertechnik antreffen werden.

Die Bedeutung der Anweisungen STATIC Variablenliste und SHARED Variablenliste werden wir im Abschnitt Prozeduren in Turbo BASIC ausführlich behandeln.

Bevor wir uns aber mit Prozeduren beschäftigen, wollen wir die Theorie mit einem kleinen Demoprogramm auflokkern. Das Programm in Listing 1 zeichnet fünf verschiedene Kurven mit der allgemeinen Gleichung r=f(phi), wo r und phi Polarkoordinaten sind und phi von 0 bis 360 Grad läuft. Ueber ein Menü können Sie die entsprechende Kurve auswählen. Die Berechnung von r erfolgt in einer mehrzeiligen Funktionsprozedur über die SELECT CASE n-Anweisung. Die Abbildungen 1-5 sind Hardcopies vom Bildschirm.

15. Unechte Unterprogramme mit GOSUB MARKE

Sie haben die bekannte Form:

```
.....
GOSUB MARKE
.....
END

MARKE:
Anweisung(en)
.....
RETURN
```

Statt GOSUB Zeilennummer n schreiben Sie besser GOSUB MARKE. Dabei ist MARKE ein von Ihnen frei gewählter Name für die Kopfstelle des Unterprogramms. Das Label MARKE muss für sich allein in einer Zeile stehen und nach dem letzten Zeichen (Buchstabe oder Ziffer) einen Doppelpunkt haben. **Achtung:** Alle Variablen innerhalb des Unterprogramms sind global!

Wir empfehlen Ihnen, diese veraltete Unterprogrammtechnik für alle Zeiten zu vergessen. Sie ist so gefährlich und unübersichtlich wie das berüchtigte GOTO und birgt wegen des Konzepts der globalen Variablen Gefahrenstellen.

16. Prozeduren in Turbo BASIC

Die folgenden Ausführungen richten sich an jene Leser, die bis anhin ausschliesslich im interpretativen BASIC programmiert haben und die den modernen Prozedurbegriff, wie er in Pascal, LOGO usw. üblich ist, noch nicht genau kennen.

Eine Prozedur ist ein eigenständiges Unterprogramm, das vom Hauptprogramm oder von einem anderen Unterprogramm aus mit seinem Namen aufgerufen wird. Dabei werden normalerweise aktuelle Werte über die sogenannte Parameterliste an die Prozedur übergeben. Die Prozedur kann, aber muss nicht, einen Wert an den aufrufenden Programmteil zurückgeben.

Sie können die Prozedur vollkommen unabhängig vom Hauptprogramm schreiben, da die von Ihnen verwendeten Variablen der Prozedur an einer anderen Stelle des Speichers abgelegt werden als die Variablen des Hauptprogramms. Eine Variable X in Ihrer Prozedur stört also den Wert einer Variablen X im Hauptprogramm und den Wert der Variablen X in einer weiteren Prozedur überhaupt nicht. Deshalb können jetzt auch in BASIC Grossprogramme geschrieben werden, indem verschiedene Programmierer einzelne Prozeduren schreiben und sich nicht mehr um die Benennung der Variablen kümmern müssen. Das Hauptprogramm besteht dann meistens nur noch aus dem zeitlich richtigen Aufruf der einzelnen Prozeduren. Ein Programmkonzept, das in allen modernen Programmiersprachen schon lange üblich ist.

Die allgemeine Form der Prozedur lautet:

```
SUB Prozedurname[(Parameterliste)] [INLINE]
        [LOCAL Variablenliste]
        [STATIC Variablenliste]
        [SHARED Variablenliste]
        Anweisung(en)
        .....
        [EXIT SUB]
        Anweisung(en)
        .....
END SUB
```

Die Anweisungen innerhalb der eckigen Klammern sind optional.

Jede Prozedur wird mit der Anweisung

```
CALL Prozedurname[(Parameterliste)]
```

aufgerufen.

Im Unterschied zu Pascal und LOGO können Sie in Turbo BASIC eine Prozedur nicht allein mit ihrem Namen aufrufen sondern müssen vor den Prozedurnamen das Schlüsselwort CALL setzen (FORTRAN-Programmierer kennen das). In Pascal und Turbo BASIC können Prozeduren nur aus einem anderen Programmteil aufgerufen werden, während in LOGO eine Prozedur unabhängig von einem Hauptprogramm direkt mit ihrem Namen aufgerufen werden kann.

Bevor wir auf die verschiedenen Details der Wertübergabe und der Variablen-Deklarationen näher eintreten, geben wir drei einfache Beispiele für Prozeduren:

1. Beispiel:

```
.....
CALL Spielbeschreibung
.....
END

SUB Spielbeschreibung
PRINT«In diesem Spiel müssen Sie eine
                                vom Computer zufällig»
PRINT«erzeugte ganze Zahl zwischen 1 und 256 erraten.»
PRINT«Sie haben maximal 10 Versuche!»
PRINT«Auf Tastendruck beginnt das Spiel.»

WHILE NOT INSTAT
WEND
CLS
END SUB
```

Dies ist eine Prozedur ohne Wertübernahme aus dem Hauptprogramm und ohne Wertübergabe an die aufrufende Stelle.

2. Beispiel:

```
.....
CALL Dreieckflaeche(10,15,18)
.....
CALL Dreieckflaeche(10,5,4)
.....
END

SUB Dreieckflaeche(A;B,C)
LOCAL S,R
S=(A+B+C)/2:R=S*(S-A)*(S-B)*(S-C)
IF R<=0 THEN
   PRINT«Die Seiten ergeben kein Dreieck»
ELSE
    PRINT USING«Dreiecksfläche = ######.##»;
                                        SQR(R)
END IF
END SUB
```

Die Prozedur wird zweimal mit aktuellen Werten aufgerufen. Auch sie liefert wie üblich keinen Wert an das Hauptprogramm zurück. Mit der Anweisung LOCAL S,R erklären Sie die in der Prozedur auftretenden Variablen zu lokalen Variablen. Die formalen Parameter A,B,C müssen Sie selbstverständlich nicht deklarieren, denn diese sind immer lokal.

3. Beispiel:

```
CLS
SCREEN 1:COLOR 1,1
U=160:V=100:R=90
FOR W=0 to 6.30 STEP 0.1
        CALL Kurvenzeichnen(W)
NEXT W
END

SUB Kurvenzeichnen(W)
LOCAL X,Y
SHARED U,V,R
X=U+R*COS(W):Y=V-R*SIN(W)
IF W=0 THEN PSET(X,Y),0 ELSE LINE-(X,Y),3
END SUB
```

Listing 2

```
menue:
screen 0 : width 80 : cls
print"Graphische Spielereien mit Kreisen"
print"--------------------------------"
print : print : print
print"Was wollen Sie?" : print
print tab(10);"1 ..... 1. Kreisfigur"
print tab(10);"2 ..... 2. Kreisfigur"
print tab(10);"3 ..... 3. Kreisfigur"
print tab(10);"4 ..... 4. Kreisfigur"
print tab(10);"5 ..... 5. Kreisfigur" : print
print tab(10);"0 ..... Programm beenden"
locate 20,1
input"Geben Sie die entspr. Zahl ein (0-5) ........";n
rad=3.14159/180

screen 1 : color 1,1
select case n
  case 1
  for r=48 to 4 step -4
      u=160+r : v=100
      call kreis(u,v,r,0,360)
      u=160 : v=100-r
      call kreis(u,v,r,0,360)
      u=160-r : v=100
      call kreis(u,v,r,0,360)
      u=160 : v=100+r
      call kreis(u,v,r,0,360)
   next r

   case 2
   for r=3 to 24 step 3
       u=160+r
       call kreis(u,100,r,0,360)
   next r
   for r=27 to 48 step 3
       u=160+48-r
       call kreis(u,100,r,0,360)
   next r
   for r=51 to 72 step 3
       u=160-48+r
       call kreis(u,100,r,0,360)
   next r
   for r=75 to 96 step 3
       u=160+96-r
       call kreis(u,100,r,0,360)
   next r
```

```
    case 3
    input"Anzahl Strahlen (4-20) .......";m
    cls
    for j=1 to m
        dw=j*360/m
        u=160+24*cos(dw*rad) : v=100-24*sin(dw*rad)
        call kreis(u,v,24,dw,180+dw)
        u=160+72*cos(dw*rad) : v=100-72*sin(dw*rad)
        call kreis(u,v,24,180+dw,360+dw)
        u=160+48*cos(dw*rad) : v=100-48*sin(dw*rad)
        call kreis(u,v,48,180+dw,360+dw)
    next j

    case 4
    for a=10 to 350 step 10
        r1=30
        u=160+r1+r1*cos(a*rad) : v=100-r1*sin(a*rad)
        r=sqr((160+2*r1-u)^2+(100-v)^2)
        call kreis(u,v,r,0,360)
    next a

    case 5
    for a=0 to 315 step 45
        for r=3 to 99 step 3
            call kreis(160,100,r,a,a+30)
        next r
    next a

    case else
        screen 0 : width 80 : end
end select

beep
locate 1,1 : print"Taste"
taste$=""
while taste$=""
      taste$=inkey$
wend
goto menue

sub kreis(xm,ym,r,w1,w2)
    local x,y,w
    shared rad
    for w=w1 to w2
        x=xm+r*cos(w*rad) : y=ym-r*sin(w*rad)
        if w=w1 then
           pset(x,y),0
        else
           line-(x,y),3
        end if
    next w
end sub
```

Mit der Anweisung SHARED U,V,R erreichen Sie, dass innerhalb der Prozedur auf die globalen Variablen U,V,R des Hauptprogramms zugegriffen werden kann. Mit LOCAL X,Y definieren Sie wie schon besprochen die Variablen X,Y zu lokalen Variablen.

Wir wollen jetzt näher auf die Einzelheiten beim Schreiben von Prozeduren eingehen.

1. Der Prozedurname kann bis zu 31 Zeichen lang sein und muss den gleichen Regeln wie bei der Vergabe von Variablennamen gehorchen.

2. In der Parameterliste werden die formalen Parameter durch Kommas getrennt. Eine implizite Typenvereinbarung ist natürlich erlaubt.

   Beispiel:

   SUB Mittelwert(A%,B%,C%,D%,E%,TEXT$)

3. Die Option INLINE bedeutet, dass die Anweisungen der Prozedur in Maschinensprache geschrieben sind.

   Beispiel:

   SUB Maschinenprogramm INLINE
   $INLINE &HFE 'Maschinenbefehle im
   $INLINE &HB9, &H20, &H4E '16-er System
   usw.
   END SUB

4. Die Variablen-Deklarationen lauten:

LOCAL X,Y	X und Y sind lokale Variable
SHARED U,V,R	U,V,R sind globale Variable
STATIC SUMME	SUMME ist eine statische Variable

   Statische Variable sind ebenfalls lokale Variable. Ihre Speicherplätze werden aber im Gegensatz zu den lo-

kalen Variablen nach der Bearbeitung der Prozedur nicht gelöscht. Aber auch hier gilt, dass eine statische Variable X eine globale Variable X nicht beeinträchtigt, da deren Speicherplätze verschieden sind.

5. EXIT SUB beendet die Prozedur vorzeitig und sollte nur in Ausnahmefällen wie etwa Fehlerbehandlung und Rekursion verwendet werden.

6. Eine Prozedur kann mit CALL Prozedurname(Parameterliste) beliebig oft aufgerufen werden. Die aktuellen Werte müssen natürlich in der gleichen Reihenfolge und typenkonform aufgelistet sein, wie dies in der formalen Parameterliste vereinbart ist.

Merke: Es zeugt von gutem Programmierstil, wenn Sie alle Variablen, die in der Prozedur auftreten, am Kopf der Prozedur vereinbaren (LOCAL, SHARED, STATIC). Die formalen Parameter in der Parameterliste sind immer lokale Variable und müssen nicht deklariert werden. Wenn Sie in einer Prozedur gewisse oder sogar alle Variablen nicht vereinbaren, dann weist ihnen Turbo BASIC automatisch das Attribut STATIC zu.

Uebergabe von Arrays bei Prozeduren

Während Sie bei der einzeiligen und mehrzeiligen Funktionsprozedur nur Werte (integer, langinteger, single und double precision, String) an die formalen Parameter übergeben können, haben Sie bei der Prozedur sogar die Möglichkeit, ein- und mehrdimensionale Felder (Arrays) zu übergeben.

Beispiel:

```
SUB Summieren(N,A(1),P,Q,B(2))
.....
END SUB
```

Der formale Parameter A(1) bedeutet, dass ein *eindimensionales Feld*

A(0),A(1),A(2),A(3),.....,A(N)

übertragen wird. Die 1 ist also kein Feldindex sondern die Anzeige für eindimensional!

Der formale Parameter B(2) bedeutet, dass ein *zweidimensionales Feld*

```
B(1,1),B(1,2),B(1,3),.....,B(1,Q)
B(2,1),B(2,2),B(2,3),.....,B(2,Q)
.....
B(P,1),B(P,2),B(P,3),.....,B(P,Q)
```

übertragen wird. Der Aufruf der Prozedur erfolgt z.B. mit

CALL Summieren(12,A(),5,3,B())

Wir wollen ein Beispiel geben:

```
cls
print «Extremwerte in einem Zahlenfeld suchen»
print « ———————————————— »
print
randomize timer
input«Wieviele Zahlen.....»;n
print
dim a(n)

for j=1 to n
    a(j)=100+int(900*rnd):print a(j);
next j

call extremwerte(n,a())
end

sub extremwerte(anzahl, x(1))
local max,min,j
max=-1000:min=10000
for j=1 to anzahl
    if x(j) < min then
      min=x(j)
    end if
    if x(j) > max then
      max=x(j)
    end if
next j
print:print
print«kleinster Wert.....»;min
print«grösster Wert.....»;max
end sub
```

Gegenüberstellung von Funktionsaufruf und Prozedur

	Funktionsaufruf	Prozedur
Aufruf	X = FNName (Parameterliste)	CALL Name (Parameterliste)
Rückgabe	Funktionswert	Wert(e) oder nichts
Parameter	Werteparameter	Variablen und/oder Werteparameter
Undeklarierte Variable	SHARED	STATIC
Array als Parameter	nicht möglich	möglich

17. Demoprogramme für den Umgang mit Prozeduren

Wiederum möchten wir die trockene Theorie mit zwei Demo-Programmen beleben. Das Programm in Listing 2 erzeugt einen Strauss von fünf hübschen Grafikfiguren, die aus Kreisen oder Kreisbögen aufgebaut sind. Die Prozedur KREIS(XM,YM,R,W1,W2) ist eine Zeichenroutine, die einen Kreisbogen um den Mittelpunkt M(XM,YM) mit dem Radius R zeichnet. Der Startwinkel W1 und der Endwinkel W2 müssen in Grad eingegeben werden, wobei W2>W1 sein muss. Für W1=0 und W2=360 wird ein voller Kreis gezeichnet. Die Abbildungen 6-9 sind Hardcopies vom Bildschirm.

Das Programm in Listing 3 erlaubt wahlweise die Navigation längs eines Grosskreises auf der Erdoberfläche oder die Navigation unter festem Kurswinkel. Für Hobby-Piloten und künftige Weltumsegler ist es ein nützliches Hilfsmittel. Zu diesem Programm bin ich durch einen Leserbrief angeregt worden.

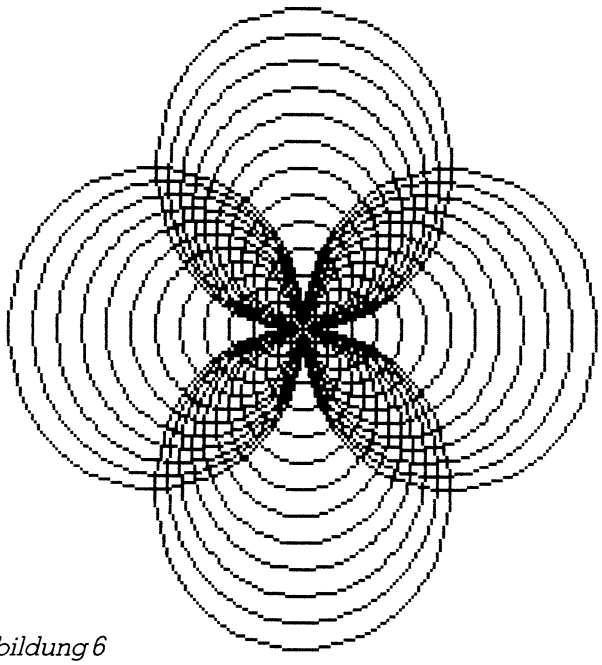


Abbildung 6

Mathematische Theorie

Die drei Punkte A,B,C auf einer Kugeloberfläche spannen ein sogenanntes sphärisches Dreieck auf. Die drei Seiten (AB)=c, (BC)=a und (CA)=b sind dabei Bögen der entsprechenden Grosskreise durch je zwei Punkte. Der Winkel (BAC) zwischen den Bögen (AB)=c und (AC)=b heisst alpha. Entsprechend sind die Winkel beta bei B und gamma bei C definiert.

Das Ziel der sphärischen Trigonometrie besteht darin, bei Kenntnis von drei der sechs Stücke a,b,c,alpha,beta, gamma alle übrigen Stücke zu berechnen.

Für unser Navigationsproblem ist nur der folgende Fall von Interesse: Gegeben zwei Seiten und der Zwischenwinkel, gesucht die dritte Seite und die beiden anderen Winkel.

Die Lösung ergibt sich aus dem sogenannten Seitencosinussatz:

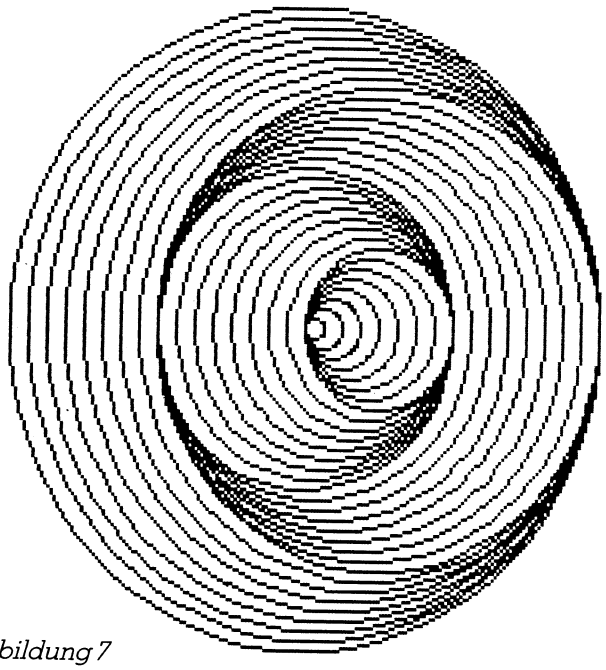


Abbildung 7

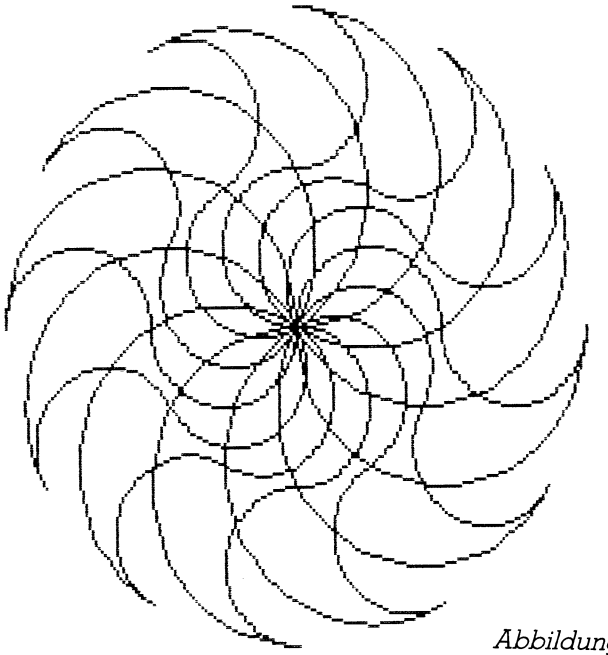


Abbildung 8

$$\cos(a) = \cos(b)*\cos(c) + \sin(b)*\sin(c)*\cos(alpha)$$
$$\cos(b) = \cos(c)*\cos(a) + \sin(c)*\sin(a)*\cos(beta)$$
$$\cos(c) = \cos(a)*\cos(b) + \sin(a)*\sin(b)*\cos(gamma)$$

Für diese Formeln wird eine Kugel mit dem Radius r=1 vorausgesetzt. Dann sind die Bögen a,b,c gerade das Bogenmass der zugehörigen Zentriwinkel im Kugelmittelpunkt, unter denen die Bögen a,b,c gesehen werden. Die wahren Längen der Bögen ergeben sich dann zu a= r*a(Bogenmass), b=r*b(Bogenmass) und c=r*c(Bogenmass).

Angenommen ein Flugzeug startet im Punkt A (z.B. New York) und fliegt nach Osten zum Punkt B (z.B. Hamburg). Dann bilden die drei Punkte A,B und der Nordpol C ein sphärisches Dreieck. Kennt man die geografische Länge und Breite des Start- und Zielortes, dann kann man leicht die Seiten a=(BC), b=(AC) und den Zwischenwinkel gamma bei C angeben. Mit Hilfe des Seitencosinussatzes für

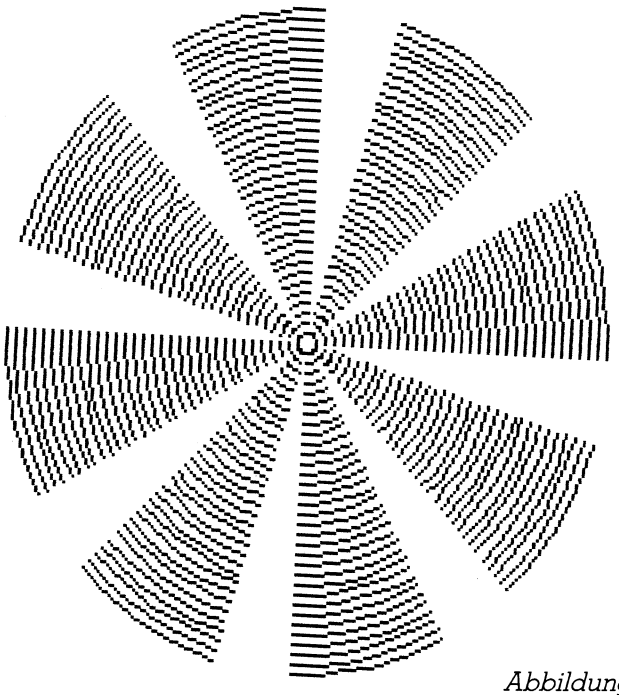


Abbildung 9

cos(c) lässt sich dann die kürzeste Entfernung c=(AB) sowie mit den beiden anderen Sätzen die Kurswinkel alpha bei A und beta bei B berechnen.

Es ist a = 90 Grad - phi(B)
b = 90 Grad - phi(A)
gamma= lambda(B) - lambda(A)

Dabei bedeuten phi die geographische Breite und lambda die geographische Länge eines Punktes der Erdoberfläche in Grad. Wir wollen westliche Längengrade und südliche Breitengrade als negative Winkel eingeben.

Fliegen wir von West nach Ost (kann man immer erreichen, wenn man A mit B vertauscht), dann liegt der Startwinkel alpha zwischen 0 und 180 Grad und der Endwinkel beta zwischen 0 und 180 Grad. Messen wir die Winkel im Uhrzeigersinn von der Nordrichtung aus (sog. Azimut), dann ist alpha gerade das Azimut, während 360-beta das korrekte Azimut bei B ist. Oft spricht man auch von Azimut Ost und Azimut West.

Im Programmsegment für die Navigation längs des Grosskreises haben wir obige Formeln verwendet. Dank den Programmbemerkungen sollte es jetzt verständlich sein.

Formeln für die Navigation längs des Grosskreises:

cos(c) = cos(a)*cos(b) + sin(a)*sin(b)*cos(gamma)

Bogen(AB) = 6371.22*arccos(c), wo c im Bogenmass sein muss

cos(alpha) = (cos(a) - cos(b)*cos(c))/(sin(b)*sin(c))
→alpha = arccos(alpha)

cos(beta) = (cos(b) - cos(a)*cos(c))/(sin(a)*sin(c))
→beta = arccos(beta)

Leider ist es technisch nicht möglich, ein Flugzeug genau längs eines Grosskreises zu steuern. Der Kurswinkel ändert sich nämlich von Punkt zu Punkt stetig. Man behilft sich daher folgendermassen:

Längs des Grosskreises (meist eine offizielle Flugroute) werden eine Reihe von Zwischenpunkten P1, P2, ..., PN eingeflochten. Das sind oft Flughäfen von überflogenen Städten oder Funkfeuer längs der Flugroute. Man fliegt dann mit festem Kurs alpha1 von A nach P1, wechselt den Kurs und fliegt mit neuem festem Kurs alpha2 von P1 nach P2, usw. Auf diese Weise ist die Navigation sehr einfach und kann vom Bordcomputer mittels nachfolgender Formeln berechnet werden. Der Distanzzuwachs ist erstaunlich gering, wie das durchgerechnete Beispiel (Flug von Rio de Janeiro über Lissabon nach München) zeigt.

Formeln für die Navigation mit festem Kurs:

x = tan(phi(B)/2 + pi/4)
y = tan(phi(A)/2 + pi/4)

tan(alpha) = (lambda(B) - lambda(A))/ln(x/y)

Bogen(AB) = (phi(B) - phi(A))/cos(alpha)*60*1.852km

Diese Formeln sind nur gültig, wenn phi(A) <> phi(B) ist! Falls phi(A) = phi(B) ist, benützen Sie die folgenden Formeln:

alpha = 90 Grad

Bogen(AB) = (lambda(B) - lambda(A))/
cos (phi(A))*60*1.852km

Damit Sie mit dem Programm experimentieren und Ihre künftigen Weltreisen vorbereiten können, haben wir Ihnen am Ende des Programms die Koordinaten einiger Grossstädte angegeben. Vermeiden Sie Flüge über den Nord- und Südpol und überqueren Sie nicht die internationale Datumslinie. Happy flying and happy landing.

Listing 3

```
menue:
cls
print"Navigation auf der Erdoberfläche"
print"-------------------------------"
print:print:print
print"Was wollen Sie?":print:print
print tab(10);"1 ..... Navigation längs des Grosskreises"
print
print tab(10);"2 ..... Navigation mit festem Kurs"
print
print tab(10);"3 ..... Koordinaten einiger Städte der Erde
print
print tab(10);"0 ..... Programm abbrechen"
locate 20,1
input"Geben Sie die entsprechende Zahl (0-3) ein.....";zahl

select case zahl
  case 1
  call Grosskreis.Navigation

  case 2
  call fester.Kurs.Navigation

  case 3
  call Koordinaten

  case else
  cls:end
end select
```

```
sub Grosskreis.Navigation
cls
print"Navigation längs des Grosskreises"
print"--------------------------------"
print"Ein Flugzeug fliegt vom Startort A ostwärts auf dem kürzesten Weg nach "
print"dem Zielort B längs des Grosskreises durch A und B. Geben Sie westliche "
print"Längengrade und südliche Breitengrade als negative Winkel ein."
print

input"Name des Startortes ..................... ";a$
input"geographische Breite in Grad, Minuten ..... ";b,mb
input"geographische Länge  in Grad, Minuten ..... ";l,ml
b1=sgn(b)*(abs(b)+mb/60)    'Umrechnung in Dezimalen von Grad
l1=sgn(l)*(abs(l)+ml/60)    '     "             "
print
input"Name des Zielortes ..................... ";b$
input"geographische Breite in Grad, Minuten ..... ";b,mb
input"geographische Länge  in Grad, Minuten ..... ";l,ml
b2=sgn(b)*(abs(b)+mb/60)    'Umrechnung in Dezimalen von Grad
l2=sgn(l)*(abs(l)+ml/60)    '     "             "
print

pi=3.14159 : rad=pi/180 : r=6371.22
a=(90-b2)*rad : b=(90-b1)*rad : gamma=(l2-l1)*rad
'Seitencosinussatz cc=cos(c)
cc=cos(a)*cos(b)+sin(a)*sin(b)*cos(gamma)
c=atn(sqr(1-cc*cc)/cc)        'arccos(c) über arctan berechnen
if c<0 then c=pi+c
distanz=r*c
print using"Entfernung längs des Grosskreises = ######.### km";distanz

'Seitencosinussatz ca=cos(alpha), cb=cos(beta)
ca=(cos(a)-cos(b)*cos(c))/(sin(b)*sin(c))
if ca=0 then
   alpha=90
else
   alpha=atn(sqr(1-ca*ca)/ca)/rad
end if
if alpha<0 then alpha=180+alpha
cb=(cos(b)-cos(a)*cos(c))/(sin(a)*sin(c))
if cb=0 then
   beta=90
else
   beta=atn(sqr(1-cb*cb)/cb)/rad
end if
if beta<0 then beta=180+beta
print using"Kurswinkel beim Start              = ###.## Grad Ost";  alpha
print using"Kurswinkel bei der Ankunft         = ###.## Grad West"; beta
locate 20,1
print"Bitte eine Taste drücken";
while not instat
wend
goto menue
end sub

sub fester.Kurs.Navigation
cls
print"Navigation mit festem Kurswinkel"
print"---------------------------------"
print"Ein Flugzeug fliegt vom Startort A ostwärts nach dem Zielort B."
print"Zwischen A und B werden maximal 9 Zwischenpunkte eingeflochten."
print"Das Flugzeug fliegt von A nach P1 mit dem konstanten Winkel alpha1,"
print"von P1 nach P2 mit dem konstanten Winkel alpha2, usw."
print"Geben sie westliche Längen und südliche Breiten als negative Winkel"
print"ein."
print"Fliegen Sie nur ost-, süd- oder nordwärts, vermeiden Sie Flüge über"
print"die Pole und die internationale Datumslinie."
print
input"Wie viele Zwischenpunkte (0-9).....................";n
if n<0 or n>9 or n<>int(n) then exit sub
input"Name des Startortes ..............................  ";a$
input"geographische Breite in Grad, Minuten ..............";b,mb
input"geographische Länge  in Grad, Minuten ..............";l,ml
b(0)=sgn(b)*(abs(b)+mb/60) : l(0)=sgn(l)*(abs(l)+ml/60)
```

```
for j=1 to n
    print
    print"Zwischenpunkt";j
    input"geographische Breite in Grad, Minuten ..............";b,mb
    input"geographische Länge  in Grad, Minuten ..............";l,ml
    b(j)=sgn(b)*(abs(b)+mb/60) : l(j)=sgn(l)*(abs(l)+ml/60)
next j

print
input"Name des Zielortes ..................................";b$
input"geographische Breite in Grad, Minuten ..............";b,mb
input"geographische Länge  in Grad, Minuten ..............";l,ml
b(n+1)=sgn(b)*(abs(b)+mb/60) : l(n+1)=sgn(l)*(abs(l)+ml/60)
print
pi=3.14159:rad=pi/180:p4=pi/4

for j=0 to n
    l1=l(j)*rad:l2=l(j+1)*rad
    b1=b(j)*rad:b2=b(j+1)*rad
    x=tan(b2/2+p4) : y=tan(b1/2+p4)
    if x=y then
       alpha=pi/2
       distanz=(l2-l1)*cos(b1)*60*1.852/rad
    else
       alpha=atn((l2-l1)/log(x/y))
       if alpha<0 then alpha=pi+alpha
       distanz=(b2-b1)/cos(alpha)*60*1.852/rad
    end if
    print using"Kurswinkel von P# nach P# = ###.## Grad";j;j+1;alpha/rad
    print using"Distanz    von P# nach P# = #####.## km";j;j+1;distanz
    print
next j
print
print"Bitte eine Taste drücken";
while not instat
wend
goto menue
end sub

sub Koordinaten
cls
for j=1 to 21
read stadt$,phi,phimin,lambda,lambdamin
f$="\                    \: phi = ### Grad ## Min /lambda = #### Grad ## Min"
print using f$;stadt$,phi,phimin,lambda,lambdamin
next j

print
print"Bitte eine Taste drücken";
while not instat
wend
goto menue
end sub

data Berlin,52,30,13,24
data Bombay,19,0,73,0
data Frankfurt,50,06,8,42
data Galapagos,0,0,-92,0
data Hamburg,53,30,10,0
data Istanbul,41,0,37,37
data Johannesburg,-26,18,28,12
data Kap gute Hoffnung,-34,30,18,30
data Kap Horn,-55,59,-67,50
data Leningrad,59,54,30,18
data Lissabon,38,42,-9,20
data London,51,30,0,30
data Melbourne,-37,42,145,0
data Muenchen,48,06,11,30
data New York,40,45,-73,57
data Paris,48,48,2,18
data Rio de Janeiro,-22,36,-43,12
data Rom,41,48,12,36
data San Franzisko,37,45,-122,24
data Sydney,-33,52,151,12
data Tokio,35,36,139,54
```

```
Navigation längs des Grosskreises
---------------------------------
Ein Flugzeug fliegt vom Startort A ostwärts auf dem kürzesten Weg nach
dem Zielort B längs des Grosskreises durch A und B. Geben Sie westliche
Längengrade und südliche Breitengrade als negative Winkel ein.

Name des Startortes ..................... ? New York
geographische Breite in Grad, Minuten ..... ? 40,42
geographische Länge  in Grad, Minuten ..... ? -74,0

Name des Zielortes ...................... ? Hamburg
geographische Breite in Grad, Minuten ..... ? 53,36
geographische Länge  in Grad, Minuten ..... ? 10,0

Entfernung längs des Grosskreises =   6128.982 km
Kurswinkel beim Start             =  46.01 Grad Ost
Kurswinkel bei der Ankunft        =  66.80 Grad West


Bitte eine Taste drücken
```

Abbildung 10

```
Navigation mit festem Kurswinkel
--------------------------------
Ein Flugzeug fliegt vom Startort A ostwärts nach dem Zielort B.
Zwischen A und B werden maximal 9 Zwischenpunkte eingeflochten.
Das Flugzeug fliegt von A nach P1 mit dem konstanten Winkel alpha1,
von P1 nach P2 mit dem konstanten Winkel alpha2, usw.
Geben sie westliche Längen und südliche Breiten als negative Winkel
ein.
Fliegen Sie nur ost-, süd- oder nordwärts, vermeiden Sie Flüge über
die Pole und die internationale Datumslinie.

Wie viele Zwischenpunkte (0-9).....? 1
Name des Startortes ..................... ? Rio de Janeiro
geographische Breite in Grad, Minuten .....? -22,36
geographische Länge  in Grad, Minuten .....? -43,12

Zwischenpunkt 1
geographische Breite in Grad, Minuten .....? 38,42.5
geographische Länge  in Grad, Minuten .....? -9,20

Name des Zielortes .....? München
geographische Breite in Grad, Minuten .....? 48,06
geographische Länge  in Grad, Minuten .....? 11,30

Kurswinkel von P0 nach P1 =  27.43 Grad
Distanz    von P0 nach P1 =  7675.53 km

Kurswinkel von P1 nach P2 =  58.10 Grad
Distanz    von P1 nach P2 =  1974.90 km


Bitte eine Taste drücken
```

Abbildung 11

18. Rekursion

Turbo BASIC erlaubt den rekursiven Aufruf einer Prozedur und einer mehrzeiligen Funktionsdefinition. Im interpretativen Microsoft BASIC ist dies bekanntlich nicht möglich.

In den einschlägigen Informatikbüchern wird der rekursive Aufruf einer Prozedur als der Selbstaufruf der Prozedur aus der Prozedur heraus erklärt. Wesentlich für die Rekursion ist dabei, dass bei jedem Selbstaufruf sogenannte schwebende Operationen vorliegen, die vor dem Selbstaufruf noch nicht abgearbeitet werden können. Erst wenn nach mehreren rekursiven Aufrufen eine bestimmte Bedingung erfüllt ist, können diese ausstehenden Operationen durchgeführt werden.

Betrachten wir ein bekanntes Beispiel aus der Mathematik.

Die Fakultät von n, also n! = 1*2*3*4*....*(n-1)*n kann auf folgende Art elegant rekursiv definiert werden:

n! = 1, wenn n=0
sonst n! = n*(n-1)!

n! ist also nur berechenbar, wenn (n-1)! bekannt ist. (n-1)! ist wiederum nur berechenbar, wenn (n-2)! bekannt ist,

usw. Bei jedem Schritt verkleinern wir n um 1. Wenn wir bei n=0 angekommen sind, dann wissen wir, dass 0!=1 ist und können jetzt die schwebenden Operationen (hier lauter Multiplikationen) von hinten nach vorn ausführen.

```
Also n! = n*(n-1)!
        = n*(n-1)*(n-2)!
        = n*(n-1)*(n-2)*(n-3)!
              usw.
        = n*(n-1)*(n-2)*(n-3)* .....*3*2*1*1
```

Dieses Zurücklaufen mit n, bis die Bedingung n=0 erfüllt ist, nennt der Mathematiker eine Rekursion. Wir werden sehen, dass man mit Hilfe der rekursiven Darstellung verwickelte Zusammenhänge elegant darstellen kann.

Merke: Programme, die rekursive Aufrufe von Prozeduren enthalten, sind erstaunlich kurz. Leider sind sie aber oft schwer zu durchschauen. Ihre Wirkungsweise ist längst nicht so durchsichtig wie ein iteratives Programm.

Um uns im Schreiben von rekursiven Programmen zu üben, wollen wir die Berechnung von n! auf zwei verschiedene Arten realisieren. Natürlich ist es sehr einfach, n! iterativ zu berechnen. Im Abschnitt 14 bei der Behandlung der mehrzeiligen Funktionsprozedur haben wir Ihnen ein entsprechendes Programm vorgestellt.

Das erste Programm (Listing 4) benützt die Möglichkeit, eine mehrzeilige Funktionsdefinition rekursiv aufzurufen. Beachten Sie, dass die rekursive Prozedur DEF FNFakultaet# (n%) fast wörtlich mit obiger mathematischen rekursiven Definition übereinstimmt. Abbildung 12 zeigt einen Programmlauf. Sie sehen, Turbo BASIC schafft wie Pascal und andere Hochsprachen mühelos die Rekursion. Wer hätte das von BASIC je erwartet!

Das zweite Programm (Listing 5) benützt die Möglichkeit, eine Prozedur rekursiv aufzurufen. Obwohl es sehr ähnlich wie das erste Programm aufgebaut ist, besteht doch ein fundamentaler Unterschied zwischen den beiden Programmen.

Eine Funktionsprozedur gibt immer einen Wert nach aussen ab. Deshalb ist der Ausgabewert nicht in der Liste der formalen Parameter beim Prozedurkopf enthalten. Eine Prozedur gibt normalerweise keinen Wert nach aussen ab. Wenn sie aber rekursiv aufgerufen wird, muss ja ein Wert (Operator einer noch schwebenden Operation) nach aussen, also an den erneuten rekursiven Aufruf übergeben werden. Das ist der Grund, warum im Prozedurkopf bei der Liste der formalen Parameter die Variable FAK# aufgeführt ist.

Abbildung 13 zeigt, dass auch mit diesem Programm n! korrekt rekursiv berechnet wird.

Bevor wir ein weiteres Beispiel für eine Rekursion geben, müssen wir auf folgende Schwierigkeit aufmerksam machen:

Wenn in einem Turbo BASIC-Programm ein rekursiver Aufruf erfolgt, dann speichert der Computer alle Variablen der Prozedur mit ihren momentanen Werten auf den sogenannten Stack im Speicher. Auch die hängigen Operationen sowie die Aufruftiefe (sog. Level der Rekursion) werden auf dem Stack festgehalten. Erfolgt ein zweiter rekursiver Aufruf, dann werden zusätzlich die neuen lokalen Variablen der Prozedur auf den Stack abgelegt. Im Normalfall umfasst der Stack 768 Byte. Rechnet man pro rekursiven Aufruf im Durchschnitt eine zusätzliche Belegung von 12 Byte, dann ist der Stack nach 768:12=64 rekursiven Aufrufen hintereinander voll. Ein nächster Aufruf erzeugt einen Ueberlauf des Stacks. Im Normalfall merkt das der Computer nicht. Die abgelegten Variablen sind dann nicht mehr im Stack-Speichersegment des Computers sondern in einem nachfolgenden Segment. Wenn jetzt die Bedingung der Rekursion erfüllt ist und der Com-

Listing 4

```
def fnfakultaet#(n%)
    if n%=0 or n%=1 then
       fnfakultaet#=1
    elseif n%>1 and n%<171 then
       fnfakultaet#=n%*fnfakultaet#(n%-1)
    else
       fnfakultaet#=-1
    end if
end def

'Hauptprogramm ------------------------
cls
print"Rekursive Berechnung von n!"
print"---------------------------"
eingabe:
print
input"Geben Sie ein ganzes n ein (0-170) .......";zahl%
fak#=fnfakultaet#(zahl%)
if fak#=-1 then
   print"falsche Eingabe"
   stop
else
   print zahl%;"fakultät =";fak#
end if
input"Noch eine Berechnung (j/n) .....";t$
if t$="j" then
   goto eingabe
else
  print"Programm abgebrochen"
  end
```

```
Rekursive Berechnung von n!
---------------------------

Geben Sie ein ganzes n ein (0-170) .......? 10
 10 fakultät = 3628800
Noch eine Berechnung (j/n) .....? j

Geben Sie ein ganzes n ein (0-170) .......? 100
 100 fakultät = 9.33262154439441E+157
Noch eine Berechnung (j/n) .....? j

Geben Sie ein ganzes n ein (0-170) .......? 170
 170 fakultät = 7.257415615307994E+306
Noch eine Berechnung (j/n) .....? j

Geben Sie ein ganzes n ein (0-170) .......? 5
 5 fakultät = 120
Noch eine Berechnung (j/n) .....? j

Geben Sie ein ganzes n ein (0-170) .......? 200
falsche Eingabe
```

Abbildung 12

puter die schwebenden Operationen rückwärts ausrechnet, dann berücksichtigt er nur die Variablen, die auf dem Stack liegen. Also muss ein falsches Ergebnis herauskommen!

Wollen Sie das vermeiden, dann müssen Sie folgende Schritte unternehmen:

1. Mit der Anweisung $STACK 32767 vergrössern Sie den Stack auf maximal 32767 Byte. Mehr ist nicht möglich. Dieser Compilerbefehl sollte am Anfang Ihres BASIC-Programms stehen. Selbstverständlich könnten Sie auch $STACK 5000 eingeben.
2. Bevor Sie Ihr Programm ausführen, wählen Sie im Menü Option den Punkt Stack Test. Drücken Sie die Returntaste, worauf der Schalter von OFF auf ON gestellt wird. Turbo BASIC prüft jetzt bei jedem rekursiven Aufruf zuerst, ob auf dem Stack noch genügend Platz zum Ablegen der Variablen ist. Falls das nicht der Fall ist, wird das laufende Programm mit einer Fehlermeldung abgebrochen.

Auf diese Art können Sie sich vor unliebsamen Ueberraschungen schützen. Allerdings läuft jetzt das Programm langsamer ab, da der Computer laufend den Stack Test durchführen muss. Bei allen folgenden Programmen müssen Sie den Stack nicht vergrössern.

Ein berühmtes Beispiel für eine elegante Rekursion ist die Lösung des Puzzles «Die Türme von Hanoi». Es fehlt in keinem Lehrbuch der Informatik. Böse Zungen behaupten, dass es nur deshalb erfunden wurde, damit die Informati-

Listing 5

```
sub fakultaet(n%,fak#)
    if n%=0 or n%=1 then
       fak#=1
    elseif n%>1 and n%<171 then
       call fakultaet(n%-1,fak#)
       fak#=n%*fak#
    else
       fak#=-1
    end if
end sub

'Hauptprogramm -------------------------
cls
print"Rekursive Berechnung von n!"
print"---------------------------"
eingabe:
print
input"Geben Sie ein ganzes n ein (0-170) .......";zahl%
call fakultaet(zahl%,fak#)
if fak#=-1 then
   print"falsche Eingabe"
   stop
else
   print zahl%;"fakultät =";fak#
end if
input"Noch eine Berechnung (j/n) .....";t$
if t$="j" then
   goto eingabe
else
  print"Programm abgebrochen"
  end
```

```
Rekursive Berechnung von n!
---------------------------

Geben Sie ein ganzes n ein (0-170) .......? 0
 0 fakultät = 1
Noch eine Berechnung (j/n) .....? j

Geben Sie ein ganzes n ein (0-170) .......? 1
 1 fakultät = 1
Noch eine Berechnung (j/n) .....? j

Geben Sie ein ganzes n ein (0-170) .......? 50
 50 fakultät = 3.0414093201713E+064
Noch eine Berechnung (j/n) .....? j

Geben Sie ein ganzes n ein (0-170) .......? 150
 150 fakultät = 5.7133839564458E+262
Noch eine Berechnung (j/n) .....? j

Geben Sie ein ganzes n ein (0-170) .......? -5
falsche Eingabe
```

Abbildung 13

ker den Wunsch nach rekursiven Prozeduraufrufen in einer Programmiersprache anbringen können. Das ist natürlich Unsinn. Auch für das Hanoi-Puzzle gibt es eine iterative Lösung und in der Informatik gibt es wichtigere Gebiete, bei denen rekursive Verfahren zum Zug kommen. Wir denken etwa an das Zeichnen von rekursiven Mustern nach Benoit Mandelbrot, an Hilbert-, Sierpinski- und Kochkurven, usw.

Obwohl wir annehmen, dass das Puzzle dem Leser bekannt ist, sei es nochmals kurz erläutert.

In der Ausgangsstellung sind n runde durchlochte Scheiben an einem Stift A (Ausgangsstift) in abnehmender Grösse übereinander getürmt, wobei die grösste Scheibe unten und die kleinste Scheibe oben liegt. Jetzt sollen diese n Scheiben vom Stift A auf den Stift B (Zielstift) übertragen werden. Dabei gelten folgende Einschränkungen:

1. Es darf pro Zug nur eine Scheibe bewegt werden.
2. Niemals darf eine grössere über eine kleinere Scheibe gelegt werden.

Um diese Aufgabe zu lösen, brauchen wir einen dritten Stift C (Hilfsstift), der die Rolle eines Zwischenspeichers spielt. Die rekursive Lösung des Puzzles lautet wie folgt:

1. Bringe (n-1) Scheiben vom Stift A auf den Stift C unter Benützung des Hilfsstiftes B.
2. Lege die unterste Scheibe n vom Stift A auf den Stift B.
3. Bringe die (n-1) Scheiben vom Stift C auf den Stift B unter Benützung des Hilfsstiftes A.

Das Puzzle kann also nur gelöst werden, wenn man weiss, wie (n-1) Scheiben von einem Stift auf einen anderen Stift übertragen werden. Das ist aber die gleiche Fragestellung wie das Ausgangsproblem, nur dass n um 1 verkleinert ist. Man erkennt die Rekursion.

Das Programm in Listing 6 folgt wörtlich obigem Lösungsalgorithmus. Es ist erstaunlich kurz. Die eigentliche Arbeit wird in der rekursiven Prozedur HANOI (N%,A$,B$,C$) geleistet. Diese besteht nur aus 4 Zeilen. Beachten Sie den Abbruch der rekursiven Aufrufe. Die Anweisung IF N% = 0 THEN EXIT SUB bewirkt, dass wenn N auf 0 reduziert wurde, die Aufruftiefe wieder um 1 erhöht wird und dann die auf CALL HANOI(N%-1,A$,C$,B$) folgende PRINT-Anweisung ausgeführt wird. Diese schreibt auf dem Bildschirm die entsprechende Umlegung einer Scheibe auf. Abbildung 14 zeigt die Lösung des Puzzles für die Fälle n=4 und n=6.

Auf der mitgelieferten Turbo BASIC-Demodiskette ist ein umfangreiches Programm mit dem Namen HANOI drauf. Wenn Sie es fahren, dann erleben Sie die Umlegungen grafisch. Die ganze Prozedur läuft wie ein Film ab. Listen Sie das Programm heraus, studieren Sie es und Sie können eine Menge über rekursive Prozeduren, über Grafikrouti-

Listing 6

```
cls
print"Die Türme von Hanoi, rekursive Lösung"
print"-----------------------------------"
print : print
input"Wie viele Scheiben (2-10).....";n%
input"Name des Startstiftes.....";a$
input"Name des Zielstiftes.....";b$
input"Name des Hilfstiftes.....";c$
print : print
call hanoi(n%,a$,b$,c$)
end

sub hanoi(n%,a$,b$,c$)
    if n%=0 then exit sub
    call hanoi(n%-1,a$,c$,b$)
    print"Scheibe";n%;"von Stift ";a$;" nach Stift ";b$
    call hanoi(n%-1,c$,b$,a$)
end sub
```

Listing 7

```
cls
defint a-z
print"Die Türme von Hanoi, iterative Lösung des Puzzles"
print"-------------------------------------------------"
print : print
input"Wie viele Scheiben (2-10).................";n
if n<2 or n>10 or n<>int(n) then
   print"falsche Eingabe!"
   stop
end if
dim turm(2,n),h(2),stift$(2)
p=1
stift$(0)="A" : stift$(1)="B" : stift$(2)="C"

for j=1 to n
    turm(0,j)=n+1-j          'Stift 0 hat n Scheiben
    turm(1,j)=0              'Stift 1 ist leer
    turm(2,j)=0              'Stift 2 ist leer
    p=p*2                    '2^n berechnen
next j
anzahl=p-1                   'Anzahl der Umlegungen = 2^n - 1
h(0)=n : h(1)=0 : h(2)=0     'Höhe der Scheiben zu Beginn
klein=0                      'Stiftnummer, wo kleinste Scheibe liegt

'Hauptschleife für die Anzahl Umlegungen
cls
for j=1 to anzahl
    if j/2<>int(j/2) then
       klein=(klein+1) mod 3
       call kleine.Scheibe.umlegen(turm(),h(),klein)
    else
       call Scheibe.umlegen(turm(),h(),klein)
    end if
next j
end
'Ende des Hauptprogramms

'Prozedur, um kleinste Scheibe im Gegenuhrzeigersinn zu bewegen
sub kleine.Scheibe.umlegen(turm(2),h(1),klein)
    local von,nach,k1
    k1=(klein+2) mod 3
    von=k1 : nach=klein
    call zug(von,nach,turm(),h())
end sub

'Prozedur, um eine andere Scheibe geeignet umzulegen
sub Scheibe.umlegen(turm(2),h(1),klein)
    local von,nach,k1,k2
    k1=(klein+2) mod 3 : k2=(klein+1) mod 3
    if h(k1)=0 then
       von=k2 : nach=k1
    elseif h(k2)=0 then
       von=k1 : nach=k2
    elseif turm(k1,h(k1))<turm(k2,h(k2)) then
       von=k1 : nach=k2
    elseif turm(k1,h(k1))>turm(k2,h(k2)) then
       von=k2 : nach=k1
    end if
    call zug(von,nach,turm(),h())
end sub

'Prozedur, um Zug auf den Bildschirm zu schreiben
sub zug(von,nach,turm(2),h(1))
    local scheibe
    shared j,stift$()
    scheibe=turm(von,h(von))
    turm(von,h(von))=0 : h(von)=h(von)-1
    h(nach)=h(nach)+1 : turm(nach,h(nach))=scheibe
    print"Scheibe";scheibe;"von Stift ";stift$(von);" nach Stift ";stift$(nach)
    if j/20=int(j/20) then
       print"Für Fortsetzung eine Taste drücken"
       taste:
       t$=inkey$:if t$="" then taste
    end if
end sub
```

```
Die Türme von Hanoi, rekursive Lösung
-------------------------------------

Wie viele Scheiben (2-10).....? 4
Name des Startstiftes.........? A
Name des Zielstiftes..........? B
Name des Hilfstiftes..........? C

Scheibe 1 von Stift A nach Stift C
Scheibe 2 von Stift A nach Stift B
Scheibe 1 von Stift C nach Stift B
Scheibe 3 von Stift A nach Stift C
Scheibe 1 von Stift B nach Stift A
Scheibe 2 von Stift B nach Stift C
Scheibe 1 von Stift A nach Stift C
Scheibe 4 von Stift A nach Stift B
Scheibe 1 von Stift C nach Stift B
Scheibe 2 von Stift C nach Stift A
Scheibe 1 von Stift B nach Stift A
Scheibe 3 von Stift C nach Stift B
Scheibe 1 von Stift A nach Stift C
Scheibe 2 von Stift A nach Stift B
Scheibe 1 von Stift C nach Stift B
```

Abbildung 14

nen u.a.m. lernen. Das Programm macht von allen Tricks Gebrauch, welche Turbo BASIC anbietet.

Zum Schluss dieses Abschnittes bringen wir noch ein Programm, welches das Hanoi-Puzzle nicht rekursiv sondern iterativ löst. Wenn Sie das Puzzle selber von Hand lösen wollen, dann bleibt Ihnen nur dieser Weg offen. Die iterative Lösung ist aber noch wenig bekannt.

Algorithmus:

1. Bei jedem Zug, der eine ungerade Nummer hat, muss die kleinste Scheibe im Uhrzeigersinn um einen Stift weiter transportiert werden. Die kleine Scheibe durchläuft den Zyklus A → B → C → A → B → C →.....
2. Bei jedem Zug, der eine gerade Nummer hat, muss die kleinere (nicht die kleinste!) der beiden oben liegenden Scheiben entsprechend umgelegt werden. Diese Umlegung erfolgt also bei jenen zwei Stiften, bei denen nicht die kleinste Scheibe liegt.

Es ist erstaunlich, wie leicht schon ein Kleinkind einen Turm mit sieben Scheiben umlegen kann, wenn es sich an diesen Algorithmus hält.

Das Programm in Listing 7 soll nochmals eine Vielzahl der neuen Möglichkeiten von Turbo BASIC zeigen. Es macht regen Gebrauch von der blockstrukturierten IF...ELSEIF...ELSE-Anweisung und dem Aufruf von Prozeduren mit lokalen und globalen Variablen.

Erläuterungen:
1. Die Stifte A,B,C erhalten die Nummern 0,1,2.
2. In der Variablen KLEIN wird die jeweilige Nummer desjenigen Stiftes festgehalten, bei dem sich momentan die kleinste Scheibe befindet.
3. Die drei Arrays TURM(0,N), TURM(1,N) und TURM(2,N) geben die jeweilige Belegung der drei Stifte mit Scheiben wieder. So bedeutet z.B. TURM(1,4)=3, dass beim Stift mit der Nummer 1 die 4. Scheibe die Nummer 3 trägt.
4. Die drei Variablen H(0), H(1) und H(2) halten bei jedem Stift dessen Höhe fest. So bedeutet H(1)=0, dass Stift 1 keine Scheiben hat und H(2)=4, dass Stift 2 genau 4 Scheiben hat.

Mit diesen Erklärungen müsste das Programm von Ihnen gelesen werden können. Sie sehen, es ist erheblich länger als das rekursive Programm.

In Turbo Pascal ab Version 3.x ist die Turtle Grafik eingebaut. Mit ihrer Hilfe und dem rekursiven Aufruf von Prozeduren kann man wunderbare fraktale Kurvenmuster zeichnen. Leider hat man in Turbo BASIC die Turtle Grafik vergessen. Der Autor wird in einem folgenden Beitrag ausführlich auf das rekursive Zeichnen von fraktalen Kurven zu sprechen kommen. Man kann die Prozeduren FORWARD, BACK, RIGHT und LEFT der Turtle Grafik auch in Turbo BASIC simulieren.

Vergleich der neuen BASIC-Versionen Turbo BASIC 1.0, Quick BASIC 2.0 und True BASIC 1.0

Wenn Sie diese Lehrgang-Serie «Einführung in Turbo BASIC» aufmerksam gelesen haben, dann wissen Sie genau, wo die Vorzüge dieser neuen BASIC-Version gegenüber allen interpretativen BASIC-Dialekten liegen. Hat Turbo BASIC wirklich nur Vorzüge?

Die nachfolgenden kritischen Bemerkungen geben die subjektive Meinung des Autors wieder. Andere Programmierer kommen vermutlich zu anderen Schwachstellen von Turbo BASIC oder gewichten sie anders als der Schreibende.

1. Warum wurde bei Turbo BASIC keine Turtle Grafik eingebaut? Beim grossen Bruder Turbo Pascal ist sie bekanntlich vorhanden. Auch in der vorallem an deutschen Schulen beliebten Sprache COMAL, eine Sprache, die etwa in der Mitte zwischen BASIC und Pascal steht, fehlt die Turtle Grafik nicht. Es ist dies umso unverständlicher, da mit dem neuen Prozedurkonzept und der Möglichkeit des rekursiven Aufrufs von Prozeduren und Funktionen die Realisation von Turtle Grafik leicht möglich gewesen wäre. Sicher, man kann seine eigenen Prozeduren für FORWARD, BACK, RIGHT, LEFT, usw. schreiben. Doch das ist unbequem.
2. Warum kann man in Turbo BASIC ein laufendes Programm nicht mit der Break-Taste abbrechen, selbst wenn der Schalter Keyboard Break auf ON gesetzt ist? Dies ist vorallem dann lästig, wenn der Rechner eine langdauernde Grafik auf dem Schirm aufbaut, die den Betrachter nicht befriedigt. Es gibt Compilersprachen, wo man ein compiliertes laufendes Programm jederzeit mit der Break-Taste abstoppen kann.
3. Der Autor vermisst den Befehl LLIST für das unkomplizierte Auslisten eines BASIC-Programms sehr. Man kann die Tastenfolge Ctrl+Pg Up, F7, Ctrl+Pg Dn, F8, Ctrl+KP oder kürzer Crtl+KP im Texteditor von Turbo BASIC nicht gerade benutzerfreundlich bezeichnen. Warum hat man diese Option nicht in das Rolladenmenü von File aufgenommen?
4. Wenn man keinen Coprozessor eingebaut hat, ist die Arbeitsgeschwindigkeit von Turbo BASIC zwar deutlich höher als im interpretativen BASIC, hinkt aber gegenüber anderen compilierten BASIC-Versionen nach. Turbo BASIC simuliert einen nicht vorhandenen Coprozessor, wodurch die Rechengeschwindigkeit gesenkt wird. Profis bezeichnen Turbo BASIC ohne Coprozessor als zu langsam.
5. Da Turbo BASIC keinen eigentlichen Linker hat, sondern Compilation und Linken in die Turbo-Umgebung integriert sind (es genügt ja, die Taste r wie Run anzutippen, um das Programm fahren zu können), können leider keine compilierten Programm-Module eingebunden werden. Wer das will, muss eben auf Modula

umsteigen. Hingegen können ganze Programme oder ein Prozedurpaket direkt ab Diskette mit dem Compilerbefehl $INCLUDE «Dateiname» während der Compilation eines Programmes eingeschoben werden.

6. Ein Leser hat uns mitgeteilt, dass das Löschen von nicht mehr gebrauchten Arrays und die Fehlerbehandlung mit ON ERROR GOTO MARKE nicht einwandfrei läuft. Der Autor, hat solche Fehler bei seinen Programmen bis jetzt noch nicht angetroffen. Er kann sie aber nicht ausschliessen, da er noch längst nicht alle im Handbuch angegebenen Features von Turbo BASIC auf Herz und Nieren geprüft hat.

Hoffen wir, dass Borland mit einer verbesserten Version Turbo BASIC 2.0 einige dieser vom Autor vorgebrachten Mängel behebt.

Vergleichen wir zum Schluss noch Turbo BASIC mit den beiden anderen modernen BASIC-Systemen Quick BASIC Version 2.0 (in den USA ist schon die Version 3.0 erhältlich) und True BASIC.

1. Kompatibilität
Turbo BASIC und Quick BASIC sind aufwärts kompatibel zu BASICA und GWBASIC. Das bedeutet, dass alle Ihre im interpretativen Microsoft BASIC geschriebenen Programme auf den beiden neuen BASIC-Systemen laufen. True BASIC-Programme haben ihre eigene Syntax. Daher laufen Microsoft BASIC-Programme nicht unter True BASIC! Dafür sollen aber True BASIC-Programme ohne Programmänderung auf dem IBM PC und allen dazu Kompatiblen, auf dem Atari, dem Amiga und dem Macintosh problemlos gefahren werden können.

2. Texteditor
True BASIC hat eigene Befehle zum Editieren der Programme. Der Texteditor von Quick BASIC ist umständlicher als der WordStar-ähnliche Editor von Turbo BASIC.

3. Farbgrafik
Alle drei BASIC-Systeme unterstützen die CGA und EGA-Karte. Beim True BASIC können Sie auch noch die Herculeskarte einsetzen. Die Grafikbefehle entsprechen dem bekannten Standard von Microsoft. True BASIC hat allerdings die Palette der Grafikbefehle deutlich erweitert.

4. Coprozessor
Nur Quick BASIC unterstützt den Coprozessor nicht. In der neuen Version 3.0 soll dieser Mangel behoben sein.

5. Zahlendarstellungen
Nur Turbo BASIC kennt den Typ langinteger und die Binärzahlen. True BASIC rechnet konsequent in Double Precision, während die beiden anderen Systeme im Normalfall mit Single Precision, also 6-7 Stellen rechnen.

6. Operationen mit Matrizen
True BASIC hat eigene Befehle für den Umgang mit Matrizen. So gibt es Befehle für die Inversion einer Matrix und die Multiplikation zweier Matrizen, usw.

7. Strings
Bei Quick BASIC und Turbo BASIC beträgt die maximale Stringlänge 32'767 Byte, bei True BASIC sogar 65'528 Byte. Der Stringspeicher ist bei Quick BASIC und Turbo BASIC auf 64 KByte begrenzt, während er bei True BASIC den gesamten Speicher umfassen kann. Die gleichen Grenzen gelten auch für den Speicher für numerische Variablen und Arrays.

8. Prozeduren und Rekursion
Alle drei BASIC-Systeme kennen das Konzept lokaler Variablen. Aber nur Turbo BASIC erlaubt den rekursiven Aufruf von Prozeduren und mehrzeiligen Funktionsaufrufen.

9. Kontrollstrukturen
Turbo BASIC und True BASIC erlauben dank der blockstrukturierten IF-Anweisung, der SELECT CASE n-Anweisung und den DO-Schleifen mit dem EXIT LOOP eine optimale strukturierte Programmierung. In Quick BASIC ist dies nur bedingt möglich.

10. Aufruf von anderen Programmen
Turbo BASIC und True BASIC erlauben den direkten Aufruf von Maschinenprogrammen. Dafür kann man nur in Quick BASIC Assembler-Programme abrufen. Wie wir schon erwähnt haben, können Sie in Turbo BASIC im Gegensatz zu den anderen beiden Systemen keine compilierten Programme einbinden. Dafür ist nur in Turbo BASIC ein sogenanntes Conditional Compiling möglich.

Fazit

Wie Sie sehen, hat jedes der drei BASIC-Systeme seine spezifischen Vor- und Nachteile. Wir müssen es daher dem Leser überlassen, welches der drei Systeme ihm am besten zusagt. Da die Turbo-Versionen von Borland dank dem Erfolgsrenner Turbo Pascal auf dem Markt die vorderen Plätze belegen, ist anzunehmen, dass auch Turbo BASIC die Nr. 1 unter den compilierten BASIC-Versionen wird. □

Wie funktionieren Roboter? (2)

Im ersten Teil dieser Lehrgangserie (M+K 88-4) haben wir die Grundlagen der Steuerung elektromechanischer Geräte erarbeitet und eine Aufstellung der Materialien gemacht, die notwendig sind, um die Beispiele nachzuvollziehen. Wir beginnen nun mit der Besprechung einiger grundlegender Experimente zur Ansteuerung von Lampen und Motoren.

Heinz Kastien

Das in der letzten Ausgabe besprochene Interface, verfügt über vier digitale Ausgänge. Diese Ausgänge können mit Relais oder Schaltern verglichen werden, die bei einer Gleichspannung von 5 V bis zu 1 A belastbar sind. Wird einer dieser Ausgänge initialisiert, so steuert er während ca. 0.5 Sekunden durch und kann einen angeschlossenen Verbraucher einschalten. Die Ansteuerung der vier Ausgänge ist aber nur eine funktionelle Gruppe des gesamten Interface. Wie wir bereits festgestellt haben, verfügt dieses Interface aber über eine Vielzahl von Funktionen, für die verschiedene Bereiche des Interface verantwortlich sind.

Betrachtet man den Schaltplan des Interface einmal etwas genauer, so kann man vier Blöcke erkennen, denen die Funktionen Netzteil (1), Analogeingang (2), Digitaleingang (3) und Treiberstufen (4) zugeordnet werden können.

Netzteil (1)

Das Netzteil bedarf keiner besonderen Erklärung, da es sich hauptsächlich aus dem Spannungsregler MC 7805 und den Siebkondensatoren zusammensetzt. Dieser einfache Aufbau resultiert aus der Tatsache, dass die Speisung bereits durch eine unstabilisierte Gleichspannung von ca. 7 V erfolgt, die zum Betrieb des Interface auf 5 V begrenzt wird, die Ausgänge werden direkt mit der unstabilisierten Gleichspannung betrieben, da ansonsten die Stabilisierung sehr gross dimensioniert werden müsste, ausserdem ist eine Siebung für die Verbraucher nicht notwendig. Die Eingangsdioden schützen vor falscher Polung der Eingangsspannung.

Analogeingang (2)

Hauptbestandteil des Analog-Digital-Wandlers ist der Dual-Timer NE 556. Ein Teil des Timers arbeitet als Monoflop der zweite als Oszillator. Eine Widerstandsänderung an den Eingängen EX und EY beeinflusst die Schaltzeiten des Monoflop, der den Oszillator ein- und nach Beendigung des Monoflopimpulses wieder ausschaltet. Während des Monoflopimpulses erzeugt also der Oszillator eine bestimmte Anzahl Impulse, die dem Widerstandwert am Eingang EX oder EY proportional sind, da die Dauer des Monoflopimpulses bei konstanter Kapazität (0.47 µF) dem Widerstandswert ebenfalls direkt proportional ist. Auf diese etwas komplizierte Art ist eine Umwandlung einer Analogspannung in eine Folge digitaler Signale möglich.

Digitaleingang (3)

Die Digitaleingangsstufe wird aus dem 8-Bit Eingangsschieberegister CD 4014 und dem Ausgangsregister CD 4094 gebildet. Die Uebernahme der Signale und das dazu gehörige Impulsdiagramm wurde bereits in M+K 88-4 veröffentlicht. Ein Teil des Dualzeitschalter NE 556 arbeitet als Monoflop und erzeugt den Freigabeimpuls. Der zweite Teil des NE 556 ist hier nicht beschaltet. Das CD 4094 arbeitet als Speicherregister für die Motortreiber.

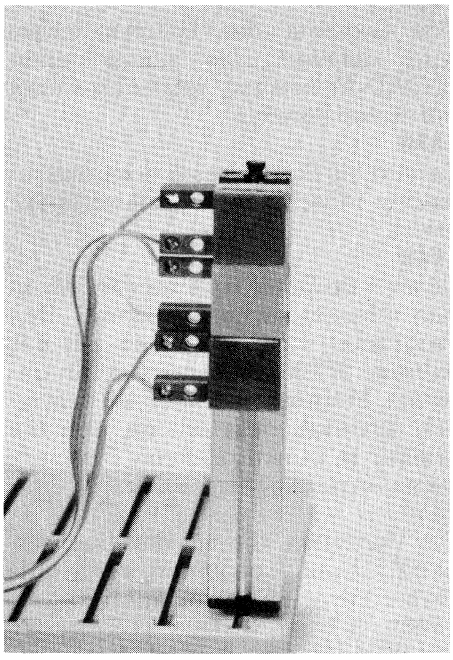
Motortreiber (4)

Die Motortreiber sind vierfach vorhanden, da das Interface gleichzeitig vier Ausgänge steuern kann, hierbei ist es gleichgültig, ob Motoren oder ohmsche Lasten betrieben werden. Liegen die beiden Basisanschlüsse der BC 548 eines Treibers auf Null-Potential, sperren die beiden Transistoren und somit auch die Leistungstransistoren BD 135 und BD 136. Durch den Motor fliesst kein Strom. Wird dagegen einer der beiden BC 548 positiv, leitet der Transistor und steuert die Leistungstransistoren an. Durch den Motor fliesst ein Strom. Die Drehrichtung des Motors wird dadurch definiert, welcher der beiden BC 548 leitend wird. Die vier Dioden schützen die Leistungstransistoren vor den hohen Spannungspitzen, wie sie beim Ausschalten induktiver Lasten auftreten. Softwaremässig erfolgt die Ansteuerung aus einem BASIC-Programm heraus mit dem Befehl
CALL M(Nummer)

Wird dieser Befehl in einer Schleife periodisch aufgerufen, so bleibt das Interface und somit der Verbraucher aktiv, andernfalls schaltet der Verbraucher nach 0.5 Sekunden ab.

Verkehrsampelsteuerung

Die modellmässige Steuerung einer Verkehrsampel kann stellvertretend für alle digitalen Ausgänge angesehen werden. Der Aufbau einer Ampel ist denkbar einfach, da lediglich an die Ausgänge M1-M3 drei Glühlampen angeschlossen werden, die selbstverständlich die Anforderungen nach einer Spannung von 5 V und einer maximalen Leistungsaufnahme von 5 VA erfüllen müssen. Es empfiehlt sich auch hier die Originallampen des Fischer Bausatzes zu verwenden und sie zur besseren Demonstration mit einer roten, einer gelben und einer grünen Abdeckung zu versehen. Der mechanische Aufbau dieses Modells bedarf keiner speziellen Erläuterung, da der Aufbau aus der Abbildung eindeutig hervorgeht.



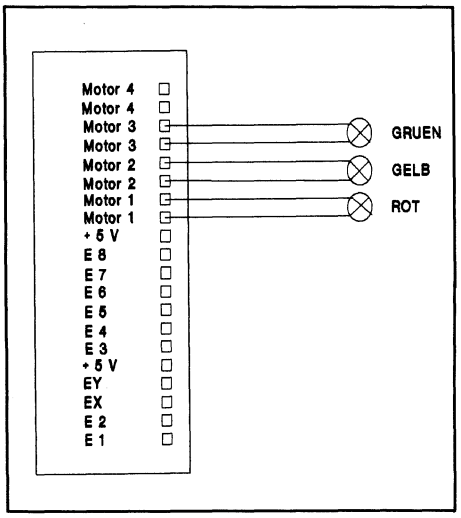
Die Verkehrsampel

Das Anschlussschema, das in dieser Form auch in Zukunft alle Modelle begleiten soll, zeigt den Anschluss der Lampen an das Interface.

Programmbeschreibung

In der letzten Ausgabe haben wir die Initialisierungsroutine des Fischer Interface abgedruckt, deren Funktion im nachfolgenden Programm der Ampelsteuerung beschrieben wird.

Diese Initialisierungsroutine umfasst die Zeilennummern 1500 bis 1600



Schaltplan der Verkehrsampel

und ist auch in allen folgenden Programmen die gleiche.

1510
Es werden alle Variablen gelöscht und der BASIC-Speicherbereich auf den Maximalwert $FF00 (Dez. 65280) gesetzt.

1520
Das Segment des Hauptspeichers wird definiert. Die nachfolgende BLOAD-Anweisung setzt den Offset auf diesen Bereich.

1530
Es wird die Startadresse des aufzurufenden Maschinenprogramms spezifiziert.

1540
Es wird die Datei «Interface.COM» in den Hauptspeicher geladen. Das Programm wird beginnend mit der Adresse $FF00 abgespeichert. In unserem Programmbeispiel ist zusätzlich ein Pfad definiert, da sich die Treiber in einem anderen Directory befinden als die Programme.

1550
Die Variablen, die mit den Buchstaben E,R,L und A beginnen, werden als Ganzzahlvariablen deklariert.

1560
Die Startadressen der Unterprogramme werden definiert. Der Aufruf von INIT versetzt das Interface in den Grundzustand. M1-M4 steuern die digitalen Ausgänge an.

1570-1590
Die PEEK-Werte der Eingangsparameter werden definiert.

1600
Das Interface wird in den Ausgangszustand versetzt.

Das BASIC-Programm der Ampelsteuerung umfasst die Zeilen 2000-3500 und die Subroutine in den Zeilen 42000-42200. In dieser Subroutine wird lediglich das Symbol einer Ver-

Listing 1

```
1000 REM Ampel by H. Kastien 23.12.1987
1010 CLS:KEY OFF:TA$="Ampelsteuerung":TB$="AMP"
1020 GOSUB 41000
1030 GOSUB 42000
1500 REM Treiberroutinen
1510 CLEAR ,&HFF00
1520 DEF SEG
1530 DEF USR=&HFF50
1540 BLOAD "ÖROBOTÖTREIBERÖINTERFAC.COM",&HFF00
1550 DEFINT E,R,L,A
1560 INIT=&HFF00:M1=INIT+6:M2=M1+4:M3=M2+4:M4=M3+4
1570 E1=1:E2=2:E3=4:E4=8:E5=16:E6=32:E7=64:E8=128
1580 EX=160:EY=144
1590 RECHTS=85:LINKS=170:EIN=85:AUS=255
1600 CALL INIT
2000 LOCATE 8,35:COLOR 10:PRINT "Rotphase in Sek.  : "
2010 LOCATE 10,35:COLOR 10:PRINT "Gelbphase in Sek. : "
2020 LOCATE 12,35:COLOR 10:PRINT "Grünphase in Sek. : "
2030 LOCATE 14,35:COLOR 10:PRINT "Gelbphase in Sek. : "
2040 LOCATE 16,35:COLOR 10:PRINT "Start / Stop mit (S)"
2100 LOCATE 8,56:A%=3:GOSUB 40000:ROT=VAL(X$)*89:IF ZZ = 13 AND AN%=0 THEN 2500
2110 IF ROT <= 0 THEN LOCATE 8,56:PRINT SPACE$(3):LOCATE 8,56:GOTO 2100
2120 LOCATE 10,56:A%=3:GOSUB 40000:GE1=VAL(X$)*89
2130 IF GE1 <= 0 THEN LOCATE 10,56:PRINT SPACE$(3):LOCATE 10,56:GOTO 2120
2140 LOCATE 12,56:A%=3:GOSUB 40000:GRN=VAL(X$)*89
2150 IF GRN <= 0 THEN LOCATE 12,56:PRINT SPACE$(3):LOCATE 12,56:GOTO 2140
2160 LOCATE 14,56:A%=3:GOSUB 40000:GE2=VAL(X$)*89
2170 IF GE2 <= 0 THEN LOCATE 14,56:PRINT SPACE$(3):LOCATE 14,56:GOTO 2160
2180 LOCATE 16,56:A%=1:GOSUB 40000:IF X$="s" OR X$="S" THEN 3000 ELSE 2180
2500 LOCATE 18,35:COLOR 10:PRINT "Blinkanlage Sek.  : ";:A%=3:GOSUB 40000:BLINK=
VAL(X$)*89
2510 IF BLINK <= 0 THEN LOACATE 18,56:PRINT SPACE$(3):GOTO 2500
2600 LOCATE 9,11:PRINT CHR$(32)
2610 LOCATE 11,11:COLOR 14:PRINT CHR$(254)
2620 LOCATE 13,11:PRINT CHR$(32)
2630 FOR I=1 TO BLINK
2640 CALL M1(AUS)
2650 CALL M2(EIN)
2660 CALL M3(AUS)
2670 NEXT
2700 LOCATE 9,11:PRINT CHR$(32)
2710 LOCATE 11,11:PRINT CHR$(32)
2720 LOCATE 13,11:PRINT CHR$(32)
2730 FOR I=1 TO BLINK
2740 CALL M1(AUS)
2750 CALL M2(AUS)
2760 CALL M3(AUS)
2770 NEXT
2780 GOTO 2600
3000 REM Lampensteuerung
3010 CALL INIT
3020 LOCATE 9,11:COLOR 4:PRINT CHR$(254)
3030 LOCATE 11,11:PRINT CHR$(32)
3040 LOCATE 13,11:PRINT CHR$(32)
3050 FOR I=1 TO ROT
3060 CALL M1(EIN)
3070 CALL M2(AUS)
3080 CALL M3(AUS)
3090 NEXT
3200 LOCATE 9,11:COLOR 4:PRINT CHR$(254)
3210 LOCATE 11,11:COLOR 14:PRINT CHR$(254)
3220 LOCATE 13,11:PRINT CHR$(32)
3230 FOR I=1 TO GE1
3240 CALL M1(EIN)
3250 CALL M2(EIN)
3260 CALL M3(AUS)
3270 NEXT
3300 LOCATE 9,11:PRINT CHR$(32)
3310 LOCATE 11,11:PRINT CHR$(32)
3320 LOCATE 13,11:COLOR 10:PRINT CHR$(254)
3330 FOR I=1 TO GRN
3340 CALL M1(AUS)
3350 CALL M2(AUS)
3360 CALL M3(EIN)
3370 NEXT
3400 LOCATE 9,11:PRINT CHR$(32)
3410 LOCATE 11,11:COLOR 14:PRINT CHR$(254)
3420 LOCATE 13,11:PRINT CHR$(32)
3430 FOR I=1 TO GE2
3440 CALL M1(AUS)
3450 CALL M2(EIN)
3460 CALL M3(AUS)
3470 NEXT
3500 GOTO 3020
```

```
40000 REM GET-Routine Zeile 40000-40310
41000 REM Rahmen zeichnen
42000 REM Ampel zeichnen
42010 COLOR 10
42020 LOCATE 8,10:PRINT CHR$(201)+CHR$(205)+CHR$(187)
42030 LOCATE 9,10:PRINT CHR$(186)+CHR$(32)+CHR$(186)
42040 LOCATE 10,10:PRINT CHR$(199)+CHR$(196)+CHR$(182)
42050 LOCATE 11,10:PRINT CHR$(186)+CHR$(32)+CHR$(186)
42060 LOCATE 12,10:PRINT CHR$(199)+CHR$(196)+CHR$(182)
42070 LOCATE 13,10:PRINT CHR$(186)+CHR$(32)+CHR$(186)
42080 LOCATE 14,10:PRINT CHR$(200)+CHR$(203)+CHR$(188)
42090 LOCATE 15,11:PRINT CHR$(186)
42100 LOCATE 16,11:PRINT CHR$(186)
42200 RETURN
```

kehrsampel auf dem Bildschirm abgebildet. Der Hauptteil des Programms gliedert sich in die Teile Erfassung der Schaltzeiten 2000-2510, Blinkmodus 2600-2780 und Regelmodus 3000-3500.

Erfassung der Schaltzeiten

2000-2040
Bildschirmdarstellung
2100-2110
Rotphase; Erfassung der Einschaltdauer für die rote Lampe mit der GET Subroutine in Zeile 40000-40220, die Effektivzeit in Sekunden ergibt sich durch Multiplikation mit 300, wird keine Zeit eingegeben und nur mit RETURN quittiert, springt das Programm auf Zeile 2500.
2120-2130
Gelb-Rotphase; Erfassung der Einschaltdauer für die rote und gelbe Lampe mit der GET Subroutine in Zeile 40000-40220, die Effektivzeit in Sekunden ergibt sich durch Multiplikation mit 300.
2140-2150
Grünphase; Erfassung der Einschaltdauer für die grüne Lampe mit der GET Subroutine in Zeile 40000-40220, die Effektivzeit in Sekunden ergibt sich durch Multiplikation mit 300.
2160-2170
Gelbphase; Erfassung der Einschaltdauer für die gelbe Lampe mit der GET Subroutine in Zeile 40000-40220, die Effektivzeit in Sekunden ergibt sich durch Multiplikation mit 300.
2180
Starten des Programmablaufes

Blinkmodus

2500-2510
Blinkmodus-Uebernahme der Blinkzeit mit der GET Subroutine in Zeile 40000-40220, die Effektivzeit in Sekunden ergibt sich durch Multiplikation mit 300.
2600-2620
Auf dem Bildschirm werden alle Lampen bis auf die mittlere, gelbe gelöscht.
2630-2670
Mit der FOR...NEXT-Schleife wird die Schleife durchlaufen, hierbei bleibt durch den ständigen Aufruf von CALL M2(EIN) die gelbe Lampe eingeschaltet und erlischt nicht nach 0.5 Sekunden.
2700-2720
Auf dem Bildschirm werden alle Lampen gelöscht.
2730-2770
Mit der FOR...NEXT-Schleife wird die Schleife durchlaufen, hierbei werden durch den Aufruf von CALL M2(AUS) alle Lampen ausgeschaltet.
2780
Sprung zum Beginn der Blinkroutine.

Regelmodus

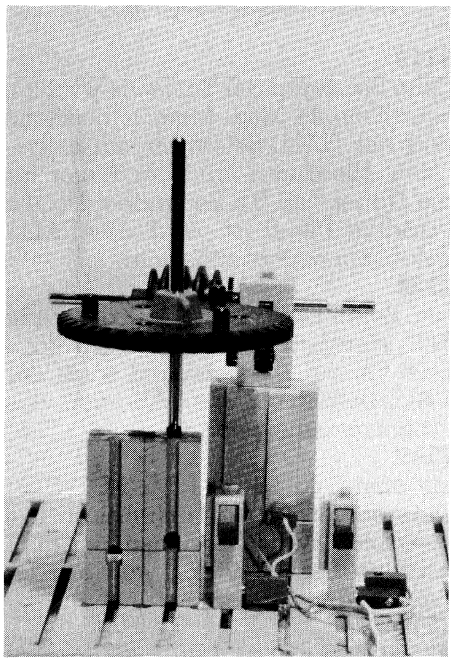
Die Blöcke in den Zeilen 3020-3040, 3200-3220, 3300-3320 und 3400-3420 entsprechen in ihrer Funktion dem Programmblock 2600-2620, es wird die entsprechende Ampelkonfiguration auf dem Bildschirm dargestellt. In den Blöcken 3050-3090, 3230-3270, 3330-3370 und 3430-3470 werden die Lampen geschaltet.

Dieses einfache Beispiel zeigt bereits, dass sich softwaremässig eine Reihe von Steuerungsaufgaben mit dem Computer bewältigen lassen. Für elektromechanische Geräte ist aber sicherlich die Ansteuerung von Motoren eines der wichtigsten Anwendungsgebiete, da der Motor der bekannteste und universellste elektromechanische Wandler ist.

Ansteuerung von Motoren

Neben den beiden Zuständen (EIN-AUS), die bereits bei den Lampen der Ampelsteuerung behandelt worden sind, kennt der Motor weitere Parameter, nämlich die Drehrichtung und Drehzahl.

Die Drehzahl eines Motors ist mit einer einfachen Steuerung, wie sie mit dem Fischer Interface ermöglicht wird, nicht realisierbar, die Drehrichtung der Gleichstrommotoren lässt sich dagegen durch Umpolen der Versorgungsspannung sehr leicht beeinflussen, da beim Aufruf des digitalen Ausgangs, an den der Motor angeschlossen ist, nur die Drehrichtung als zusätzlicher Parameter angehängt werden muss. Bei den drei nachfolgenden Programmbeispielen ist der mechanische Versuchsaufbau immer der gleiche, der Motor treibt über ein Untersetzungsgetriebe ein Zahnrad an, das zur Erkennung der Position und Drehrichtung mit einer farbigen Markierung versehen wird. (Die beiden Schalter im Vordergrund sind im ersten und zweiten Programm noch ohne Bedeutung.)

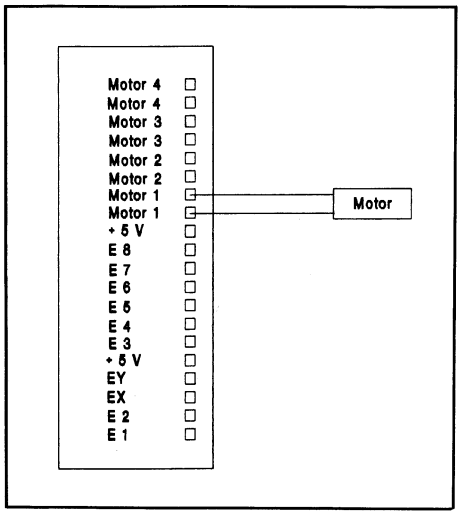


Der Motorantrieb

Im ersten Beispiel (Motorsteuerung 1) ist der Rechtslauf eines Motors im Impuls oder Dauerbetrieb möglich. Das zweite Beispiel (Motorsteuerung 2) ist mit dem ersten Programm in allen Funktionen vergleichbar, jedoch lassen sich vier Motoren im Rechts- und Linkslauf ansteuern. Im dritten Programm (Motorsteuerung 3) kann ein Motor durch zwei Drucktastenschalter ein- oder ausgeschaltet und in der Drehrichtung verändert werden.

Motorsteuerung 1

2000-2030
Bildschirmanzeige der Steuerungsvarianten
3010
Initialisierung des Motors und Zurücksetzen aller Parameter
3100
INKEY$-Befehl



Schaltplan Motorsteuerung 1

3110
Beim Betätigen der Taste «I» wird ein Schaltimpuls des Motors bestimmt und der CALL-Befehl einmal aufgerufen. Hierdurch läuft der Motor für die Dauer von 0.5 Sekunden. Da die Variable F%=0 ist, wird der CALL-Befehl nur einmal initialisiert.
3120
Die Befehlszeilen sind mit der Zeile 3110 identisch, jedoch wird die Varibale F% auf 1 gesetzt und somit der CALL-Befehl bis zum Abbruch des Programms ausgeführt.
3200
Mit dem Befehl (CALL M1(RECHTS)) wird der Motor M1 initialisiert. Die Variable RECHTS bestimmt die Drehrichtung. Diese Variable wurde in der Zeile 1590 spezifiziert.
3210
IF...THEN-Verzweigung bei Variable F%=0 zu Zeile 3100 zur erneuten Betätigung einer Taste bzw. nach Zeile 3200 zum ständigen Initialisieren des CALL-Befehls.

Motorsteuerung 2

Zum Unterschied zur Motorsteuerung 1 wird in diesem Beipiel zusätzlich der anzusteuernde Motor definiert, dies erfolgt in Zeile 2150, die Nummer des Motors geht dann in die Variable MOT ein. Da das Fischer Interface über vier digitale Ausgänge verfügt, lassen sich vier Motoren gleichzeitig individuell ansteuern.
1000-1010
Variablenübergabe
1500-1500
Treiberprogramm
2000-2140
Bildschirmaufbau zur Erfassung der Steuerungsvarianten
2150
Definition des Motors mit der GET-Routine in Zeile 40000-40310

Listing 2

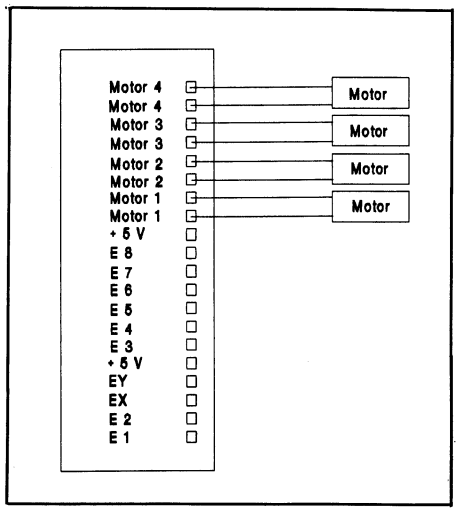
```
1000 REM Motorsteuerung by H. Kastien 26.12.1987
1010 CLS:KEY OFF
1020 TA$="Motorsteuerung mit digitaler Abfrage":TB$=" MOT 3":GOSUB 41000
1500 REM Treiberroutinen Zeile 1500-1600
1650 LOCATE 10,25:COLOR 10:PRINT "Einschaltzustand":LOCATE 10,50:PRINT "Drehrich
tung"
1700 STAT$(1)="RECHTS":IF USR(E1)=1 THEN STAT$(1)="LINKS "
1710 STAT$(2)="AUS":IF USR(E2)=0 THEN STAT$(2)="EIN"
1720 LOCATE 12,30:PRINT STAT$(2):LOCATE 12,52:PRINT STAT$(1)
1730 IF USR(E2)=1 THEN CALL M1(AUS):GOTO 1760
1740 IF USR(E1)=0 THEN CALL M1(RECHTS)
1750 IF USR(E1)=1 THEN CALL M1(LINKS)
1760 GOTO 1700
40000 REM GET-Routine 40000-40310
41000 REM Rahmen zeichnen 41000-41070
```

Listing 3

```
1000 REM Motorsteuerung by H. Kastien 23.12.1987
1500 REM Treiberroutinen  Zeile 1500-1600
2000 CLS:TA$="Motorsteuerung 1":TB$=" MOT 1":GOSUB 41000
2010 LOCATE 8,15:COLOR 10:PRINT "Impuls rechts     : (I)"
2020 LOCATE 10,15:COLOR 10:PRINT "Dauerlauf rechts  : (D)"
2030 LOCATE 15,15:COLOR 14:PRINT "Treffen Sie Ihre Wahl !"
3000 REM Motorsteuerung
3010 CALL INIT
3100 K$=INKEY$
3110 IF K$="I" OR K$="i" THEN F%=0:GOTO 3200
3120 IF K$="D" OR K$="d" THEN F%=1:GOTO 3200
3130 GOTO 3100
3200 CALL M1(RECHTS)
3210 IF F%=0 THEN 3100 ELSE 3200
41000 REM Rahmen zeichnen 41000-41070
```

Listing 4

```
1000 REM Posititonseinstellung by H. Kastien 28.12.1987
1010 CLS:KEY OFF
1500 REM Treiberroutinen  1500-1600
2000 REM Menu
2010 CLS:TA$="Positionseinstellung":TB$=" POS 1":GOSUB 41000
2020 LOCATE 18,25:COLOR 10:PRINT "Drehwinkel":LOCATE 18,45:COLOR 10:PRINT "Anzah
l Segmente"
2030 LOCATE 12,25:COLOR 10:PRINT "Umdrehung in Winkelgrad : ";:A%=4:GOSUB 40000:
UM=VAL(X$)
2040 UM=INT(UM*32/360)
2100 I%=0:CALL M1(EIN)
2110 B%=USR(E1):IF B%=1 THEN 2110
2120 B%=USR(E1):IF B%=0 THEN 2120
2130 IF I%=UM THEN CALL M1(AUS):LOCATE 12,51:PRINT SPACE$(4):GOSUB 40300:GOTO 20
00
2140 LOCATE 20,28:PRINT USING "####";(I%+1)*360/32:LOCATE 20,50:PRINT USING "###
#";I%+1:I%=I%+1:GOTO 2110
40000 REM GET-Routine Zeile 41000-40310
41000 REM Rahmen zeichnen 41000-41070
```



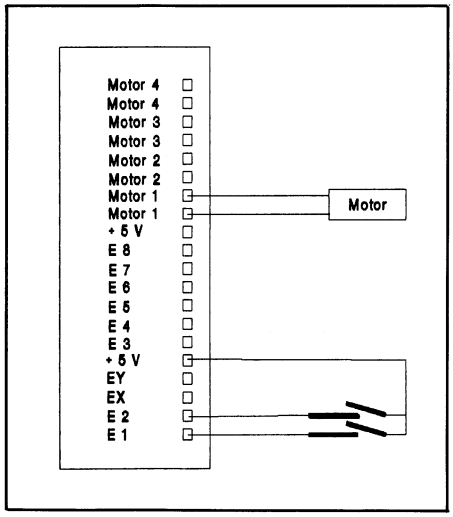
Schaltplan Motorsteuerung 2

2160-2180
Definition von Drehrichtung und Impuls sowie Dauerbetrieb
3000-3240
Steuerung der Motoren nach dem gleichen Schema wie in vorhergehenden Programmbeispiel, neu ist die Definition des Motors 1-4 in den Zeilen 3200-3210

Motorsteuerung 3

Im dritten Beispiel treten die beiden Schalter (im Vordergrund der Abbildung) in Funktion, denn in diesem Programm wird die Drehrichtung und der Einschaltzustand des Motors 1 nicht durch die Parameter des Programms gesteuert, sondern durch

diese beiden Schalter. Wie wir bereits in M+K 88-4 gesagt haben, können an den Eingängen E1 bis E8 durch digitale Signale Schaltvorgänge ausgelöst werden. In diesem Experiment hat der Schalter 1 die Funktion des EIN-AUS-Schalters, der Schalter 2 bestimmt die Drehrichtung.



Schaltplan Motorsteuerung 3

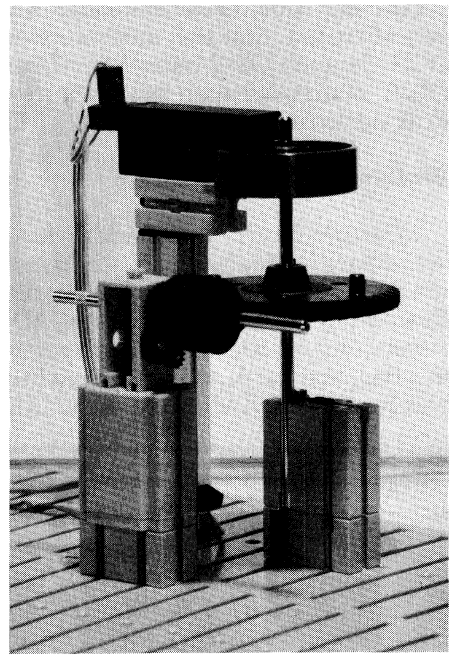
Das Programm ist denkbar einfach, in der Zeile 1700 wird mit dem Befehl USR(E1) der Status des Eingangs E1 abgerufen, liegt dieser Eingang auf logisch «1», wurde also der Schalter 1 betätigt, so dreht sich der Motor nach links, liegt der Eingang auf logisch «0», Schalter 1 wurde nicht betätigt, so ist die Drehrichtung des Motors nach rechts. Auf die gleiche Art und Weise wird am Eingang E2 der Schaltzustand des Schalter 2 ermittelt und in Zeile 1710 vom Computer abgefragt. E2 logisch «0» entspricht «AUS», logisch «1» entspricht «EIN». Der jeweilige Zustand des Motors wird in Zeile 1650 auf dem Bildschirm angezeigt. Dem Status der beiden Schalter wird in den Zeilen 1730-1750 mittels CALL-Befehl in die Drehrichtung und den Schaltzustand des Motors umgesetzt.

Fotozellensteuerung

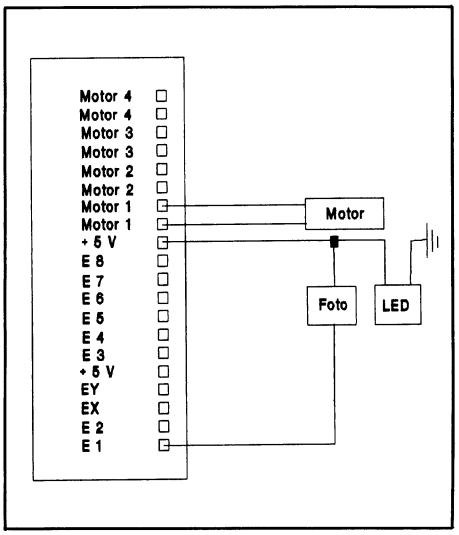
Eine besondere Art der digitalen Schalter ist die Lichtschranke. Bei einer Lichtschranke handelt es sich um eine Lichtquelle, die eine Fotozelle beleuchtet. Wird der Strahlengang unterbrochen, kann der Impuls der Fotozelle eine beliebige Steuerung auslösen. Die Fotozelle würde bei einer sehr langsamen Beleuchtungsänderung nicht exakt schalten, daher wird meist hinter die Fotozelle ein Schwellwertschalter gelegt, der den Impuls in einen solchen mit steilem Flankenanstieg umwandelt. Bei der Fotozelle des Fischer Bausatz handelt es sich um eine Gabellichtschranke, also einer Kombination aus einem Galium-Arsenid Infrarot LED als Sender und einem Fototransistor als Empfänger. Gabellichtschranken haben einen U-förmigen Aufbau; um die Beeinflussung durch Störlicht weitgehend auszuschalten, verwendet man keine Lichtquellen im sichtbaren Bereich, sondern IR-Leuchtdioden und IR-empfindliche Fototransistoren, deren Schaltzustand vom Umgebungslicht praktisch nicht beeinflusst werden.

Beim nachfolgend beschriebenen Experiment wird auf die Achse des untersetzten Antriebs ein Rad gesetzt, das auf der Aussenseite mit 32 schwarzen Streifen versehen ist. Dieses Rad bewegt sich zwischen der Lichtquelle und der Fotozelle der Gabellichtschranke. Eine volle Umdrehung dieses Rades von 360 Grad erzeugt 32 Impulse an der Fotozelle und somit auch am digitalen Eingang des Interface.

Man kann sich nun vorstellen, dass es durch Zählen der Impulse leicht möglich ist, das Antriebsrad um jeden beliebigen Winkel zu drehen. Diese Anwendung, nämlich die Drehung eines Motors um einen vorgewählten Winkel, wird in der Praxis sehr häufig benötigt, sofern man nicht mit Schrittmotoren arbeiten will. Das folgende Programm zeigt eine derartige Anwendung.



Die Positionierung



Schaltplan Positionssteuerung 1

Listing 5

```
1000 REM Motorsteuerung by H. Kastien 23.12.1987
1010 CLS:KEY OFF:TA$="Motorsteuerung 2":TB$="  MOT 2":GOSUB 41000
1500 REM Treiberroutine Zeile 1500-1600
2000 LOCATE 8,15:COLOR 10:PRINT "Welcher Motor     : 1-2-3-4  "
2100 LOCATE 10,15:COLOR 10:PRINT "Impuls rechts     : R"
2110 LOCATE 12,15:COLOR 10:PRINT "Dauerlauf rechts  : A"
2120 LOCATE 14,15:COLOR 10:PRINT "Impuls links      : L"
2130 LOCATE 16,15:COLOR 10:PRINT "Dauerlauf links   : B"
2140 LOCATE 18,15:COLOR 10:PRINT "STOP              : Ctrl-Break"
2150 LOCATE 8,44:A%=1:GOSUB 40000:MOT=VAL(X$):IF MOT < 1 OR MOT > 4 THEN LOCATE
8,44:PRINT " ":GOTO 2000
2160 LOCATE 16,43:A%=1:GOSUB 40000:K$=X$
2170 IF K$ = "1" OR K$ = "L" OR K$ = "r" OR K$ = "R" OR K$ = "a" OR K$ = "A" OR
K$ = "b" OR K$ = "B" THEN 3000
2180 GOTO 2160
3000 REM Motorsteuerung
3010 CALL INIT
3100 F%=0
3110 IF K$="r" OR K$="R" THEN ZU=RECHTS
3120 IF K$="a" OR K$="A" THEN ZU=RECHTS:F%=1:GOTO 3200
3130 IF K$="l" OR K$="L" THEN ZU=LINKS:
3140 IF K$="b" OR K$="B" THEN ZU=LINKS:F%=1:GOTO 3200
3200 M(1)=M1:M(2)=M2:M(3)=M3:M(4)=M4
3210 M=M(MOT)
3220 CALL M(ZU)
3230 IF F%=1 THEN 3220
3240 GOTO 2000
40000 REM GET-Routine 40000-40310
41000 REM Rahmen zeichnen
```

Ueber die Tastatur wird ein beliebiger Drehwinkel vorgegeben (Zeile 2030), der vom Programm in Zeile 2040 in Impulse umgerechnet wird. Beim Auslösen der Drehung des Motors wird die effektive Impulszahl am Eingang E1 mit der Vorgabe verglichen (Zeilen 2110-2130), sofort nach Erreichen der Vorgabe wird der Motor abgeschaltet. Die Anzeige des Drehwinkels und der Impulszahl erfolgt mit den Variablen UM und I% in Zeile 2020 und 2140. Bei kleinen Drehwinkeln ist diese Experimentierschaltung relativ ungenau, da sich der Motor nach dem Abschalten auf Grund der Trägheit noch um einen kleinen Betrag weiter dreht. Eine Möglichkeit diesem Effekt entgegen zu wirken, ist ein grösseres Uebertragungsverhältnis des Getriebes.

Zusammenfassung

In diesen Demonstrationsprogrammen konnte die Ansteuerung von ohmschen und induktiven Lasten gezeigt werden, ebenso die Steuerung dieser Verbraucher mit digitalen Eingangssignalen. In der nächsten Folge stehen dann die analogen Geber und Sensoren zur Diskussion.

COMPUTER-SPLITTER

2 MB RAM auf einem Chip

(486/eh) Texas soll zur Zeit an einem DRAM-Chip mit einer Kapazität von 16 MBit arbeiten. Der Chip wird mit dem 0,8 mikron EPIC-IIB Verfahren von Texas hergestellt werden. Vermutlich wird dieser sensationell dichte Speicherbaustein in CMOS-Technik ausgeführt. □

Multiplan für OS/2

(487/eh) Das neue Multiplan 4.0 von Microsoft lässt sich sowohl unter DOS als auch OS/2 betreiben. Unter OS/2 ist das Arbeiten mit mehreren Kalkulationsblättern möglich. □

Walktrough in 3D: AutoFlix

(594/fp) Nach AutoShade für's schattierte Darstellen in 3D - bekannt als Solid - kommt nun AutoFlix für die Bewegung. AutoFlix des AutoCAD-Herstellers Autodesk kommt ab sofort als Beigabe zu AutoShade, welches seinerseits ein Add-on zu AutoCAD darstellt. Solid-Modelle können mit der «Walktrough»-Funktion betreten und von innen betrachtet werden. Die «Kinetic Animation» von AutoFlix erlaubt die Bewegung von Solid-Modellen oder -Teilmodellen. □

100% mehr Daten auf der Festplatte

(600/fp) Mit DiskDoubler soll es möglich sein, doppelt so viele Daten auf der Festplatte zu plazieren. DiskDoubler (Datran Corporation, 2505 Foothill Blvd., La Crescenta, CA 91214 USA) ist eine kurze Karte, deren Prozessor die Daten (Programm- und Datendateien) so komprimiert, dass höchstens der halbe periphere Speicherbedarf belegt ist. dBASE-Dateien werden sogar auf einen Drittel ihres Platzbedarfs komprimiert. DiskDoubler sprengt auf diese Weise auch die Limite der 32 MB unter DOS adressierbarem Speicherraum. Die Installation soll sehr einfach sein und ohne Neuformatierung der Disk erfolgen können. □

COMPUTERMARKT 6/88 erscheint am 10. Nov.

Das Programm INFO

Informationen über den Hardware-Zustand erhält man nur mangelhaft vom DOS-Betriebssystem. Will man wissen wieviel Speicherplatz noch auf einem Datenträger oder im Grundspeicher zur Verfügung steht, muss der Befehl «CHKDSK» eingegeben werden. Darauf erscheinen am Bildschirm meistens zusätzliche, nicht gebrauchte Informationen. Am Schluss erhält man endlich die gewünschte Auskunft. Schlimmer wird es, wenn man wissen möchte, wieviel Platz noch im RAM-Speicher zur Verfügung steht. CHKDSK vermag diese Information zu liefern, doch zuerst überprüft und meldet dieses Programm den Zustand der gesamten Festplatte, um erst danach die Grösse des verfügbaren RAM-Speichers darzustellen. Dies ist mit Zeitverlust verbunden und entsprechend lästig.

Dr. Herbert Steiner

Viele Computer bringen beim Einschalten eine Uebersicht der benutzten Hardware-Konfiguration. Dies ist aber nicht bei allen der Fall. Oefters erhält man Fragen wegen der Beschaffenheit des eigenen Computers (z.B. welche DOS-Version man verwendet oder mit welcher Video-Karte man arbeitet). Entweder hat der Computer die oben erwähnte Möglichkeit und man müsste, um die Information zu erhalten, wieder starten oder es bleibt einem nichts anderes übrig als die Handbücher zu konsultieren. DOS selbst hat keinen Befehl, der Informationen über die benutze Hardware gibt.

Das Programm INFO schliesst diese Lücken. Mit Hilfe eines Befehles und zweier Parameter zu diesem Befehl erhält man jeweils schnell und eindeutig nur die gewünschte Information. Die Syntax des Befehles lautet:

Syntax: INFO [LAUFWERK] oder [*]
Ohne Parameter: Aktuelles LAUFWERK
LAUFWERK = Info über Laufwerk und Speicher
* = Info über Konfiguration

INFO + LAUFWERK gibt Information über den jeweiligen freien Speicherplatz des angesprochenen Laufwerkes und die freie Kapazität des Grundspeichers. Die lästigen Doppelpunkte nach dem Laufwerk können weggelassen werden. Will man wissen wieviel Speicherplatz in der Diskette, die sich im Laufwerk A befindet, vorhanden ist, wird der Befehl C > info a eingegeben. Der gleiche Befehl ohne Laufwerkangabe gibt dann die Information über das aktuelle Laufwerk, in diesem Fall die Festplatte, aus.

```
C>info
```

```
LAUFWERK C (BYTES) FREI  :    3811328
LAUFWERK C         TOTAL :   33462272

GRUNDSPEICHER      FREI  :     568528
GRUNDSPEICHER      TOTAL :     655360
```

Mit dem Befehl «Info *» erhält man, wie in der Abbildung unten auf dieser Seite dargestellt, eine Information über die benutzte Hardware.

On-Line-Hilfe

Von nun an wird folgendes für alle zu veröffentlichenden Programme eingeführt: Wenn man dem Programm-Namen als Parameter ein Fragezeichen beigibt, zeigt dies, wie die Syntax des Programm-Abrufes lautet. Diese Norm (Programm + ? = Syntax) sollte allgemein angenommen werden. Obwohl man weiss wie ein Programm funktioniert, vergisst man sehr oft, wie dieses abgerufen wird. Wird die hier benutzte Befehlsform eingeführt, könnte man dem Verbraucher lästiges Nachschauen in den Handbüchern ersparen.

Programm-Aufbau

Der Aufbau des Programmes gliedert sich wie folgt:

1. Ermittlung der eingegebenen Parameter. Diese Parameter bestimmen, welche Information das Programm benötigt (Laufwerk und Speicher, Konfiguration oder Syntax des Programmes).
2. Einholen der Informationen, die für den Bildschirm-Ausdruck benötigt werden.
3. Bildschirm-Darstellung der Information.

Eingabe Parameter

In der Routine «Parameter» wird festgestellt, welche Argumente (Laufwerk-Angabe, ? oder *) beim Abruf des Programmes benutzt werden. Da jedes Zeichen des Alphabetes als Laufwerk interpretiert werden kann und da kein Argument ebenfalls eine Bedeutung für das Programm hat, wird in dieser Routine nicht auf Fehler geachtet.

Nach Feststellung des Parameters erhält die Variable «wo» die Adresse der auszuführenden Programm-Routine.

```
                    KONFIGURATION

        MATH. COPROZESSOR .. 0          SERIELL  (RS232)  .. 1
        FESTPLATTE(N)     .. 1          PARALLEL (Drucker).. 2
        DISKETTENLAUFWERK .. 1          ERW. TASTATURTRB. .. 1

        DISK. FORMAT   1 = [5.25"]   0 = [3.50"]   0 = [A.BTRB.]
        VIDEO ADAPTER (KARTE) .. HERCULES
        DOS VERSION ..  3.20

        GRUNDSPEICHER (KB) ..  640      EXPANDED             0
                                        ERW.(. 1MB)          0

 M + K (88-5)                                    (c) HSM , Luzern
```

```
;INFO.TXT
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

cga_t           DB  "CGA ",00
cp_t            DB  "(c) HSM , Luzern",00
disk_par_t      DB  "DISK. FORMAT    "
                DB  "  = [5.25",34,"]      = [3.50",34, "]       = [A.BTRB.] ",00
diskp_st_t      DB  "STANDARD DISKTETTENLAUFWERK(E) ",00
dos_t           DB  "DOS VERSION .. ",00
dos_v           DB  " .  ",00
druck_t         DB  "Moechten Sie einen Ausdruck J/[N] ? ", 00
ega_t           DB  "EGA ",00
erw_tast_t      DB  "ERW. TASTATURTRB. ..",00
erw_sp_t        DB  "ERW.(. 1MB)",00
exp_sp_t        DB  "EXPANDED   ",00
fest_t          DB  "FESTPLATTE(N)       ..",00
floppy_t        DB  "DISKETTENLAUFWERK ..",00
herc_t          DB  "HERCULES ",00
math_t          DB  "MATH. COPROZESSOR ..",00
mono_t          DB  "MONOCHROME ",00
muk_t           DB  "M+K (88-5)",00
null_druck_t    DB  50 DUP (" "), 00H
parallel_t      DB  "PARALLEL (Drucker)..",00
seriell_t       DB  "SERIELL  (RS232)   ..",00
speicher_t      DB  "GRUNDSPEICHER (KB) .. ",00
video_t         DB  "VIDEO ADAPTER (KARTE) ..",00
vor_druck_t     DB  "Drucker ueberpruefen ... eine Taste druecken", 00

laufwerk_t      DB  "║ LAUFWERK   (BYTES) FREI  :",00
lfwk_tot_t      DB  "║ LAUFWERK           TOTAL :",00
spch_fr_t       DB  "║ GRUNDSPEICHER      FREI  :",00
spch_tot_t      DB  "║ GRUNDSPEICHER      TOTAL :",00
neu_zeile       DB  0DH, 0AH ,00
info_t   DB  13,10,"╔══════════════════════════════════════════╗"
         DB  13,10,"║ Syntax: INFO [LAUFWERK] oder [*]         ║"
         DB  13,10,"║ Ohne Parameter = Aktuelles LAUFWERK      ║"
         DB  13,10,"║ LAUFWERK = Info ueber Laufwerk und Speicher ║"
         DB  13,10,"║ * = Info ueber Konfiguration             ║"
         DB  13,10,"╚══════════════════════════════════════════╝"
         DB  13,10,00

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;RAHMEN

kl_rahmen_e     DB  " ",186,00
kl_rahmen_m     DB  199, 39 DUP (196), 182, 00
kl_rahmen_o     DB  201, 39 DUP (205), 187, 00
kl_rahmen_u     DB  200, 39 DUP (205), 188, 00
rahmen_oben     DB  201, 30 DUP (205)
                DB  181, " KONFIGURATION ",198
                DB  29 DUP (205),187, 00
rahmen_unten    DB  200, 76 DUP (205), 188, 00
```

Diese Möglichkeit, einer Variablen die Adresse einer Routine zu übergeben, erspart viel Mühe bei der Programmierung.

Einholen der notwendigen Informationen

Wenn der Anwender Informationen über seine Hardware-Konfiguration wünscht (Option *), ist es vorgesehen, dass er diese Konfiguration auch ausdrucken können sollte. Deswegen wird in diesem Falle vor Beginn der eigentlichen Informations-Einholung der vorhandene Bildschirm-Inhalt gelöscht. Dies geschieht mit Hilfe der Routine «CLS». Sie benutzt Funktion 6 des Interrupts hex 10 (ausführlich kommentiert beim Programm LIES im M+K 88-2).

Bei der Ermittlung des Laufwerkes oder der Syntax des Programmes wird der Bildschirm-Inhalt so gelassen wie er sich vorfindet und auf der nächstmöglichen Zeile die Information geschrieben.

Bei der Informations-Einholung wurde strikte darauf geachtet, nur solche Funktionen oder Interrupts zu benutzen, die dokumentiert sind. Microsoft unter anderen benutzt sehr oft bei ihren Programmen reservierte oder undokumentierte Funktionen des Betriebssystems, um spezifische Informationen zu speichern, bzw. einzuholen. Viele Programmierer (insbesondere sogenannte Hacker) beschäftigen sich dann mit dem «auseinandernehmen» des Programmes und stossen dabei auf die Bedeutung dieser Interrupts oder Funktionen. Es scheint heutzutage in Mode zu sein, solche nicht dokumentierte oder reservierte Interrupts für eigene Programme verwenden zu wollen. Davor sei ganz eindringlich gewarnt. Nicht dokumentierte Funktionen oder Interrupts können jederzeit (schon bei der nächsten DOS-Version) ihre Bedeutung verlieren, bzw. die Handhabung geändert werden. So wird sehr oft bei der Ermittlung der Konfiguration eines Computers auf die Funktion hex 32 zurückgegriffen. In der Tat benutzte DOS intern diese Funktion, um die Parameter des Laufwerkes zu ermitteln. Ein Programm, das diese Funktion benutzt, kann ab einem unbekannten Zeitpunkt nicht mehr tauglich sein. Neugierde ist gut, aber beim Programmieren sollte man sich darauf verlassen, dass die Hersteller von Betriebssystemen ein Mindestmass an Seriösität haben

```
TITLE INFO Version 0
PAGE ,132

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  PROGRAMM       INFO                            ;
;  AUTOR          DR. HERBERT STEINER             ;
;  DATUM          24.05.1988                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; NUMERISCHE KONSTANTEN

ENTER             EQU      0DH
NULL              EQU      30H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; ALPHA KONSTANTEN

SUMME             EQU      ADD
ABRUF             EQU      CALL
VERGLEICH         EQU      CMP
MINUS_1           EQU      DEC
ERHOEHE           EQU      INC
WENN_GROESSER     EQU      JA
WENN_GROSS_GL     EQU      JAE
WENN_FEHLER       EQU      JC       ;DOS = CARRY FLAG
WEITER_MIT        EQU      JMP
WENN_K_FEHLER     EQU      JNC      ;DOS = CARRY FLAG
WENN_UNGLEICH     EQU      JNZ
WENN_GLEICH       EQU      JZ
ZEIGT_AUF         EQU      LEA
VON_SI_NACH_AL    EQU      LODSB
SCHLEIFE          EQU      LOOP
NACH_VON          EQU      MOV
MAL               EQU      MUL
BEHALTE           EQU      PUSH
RUECKRUF          EQU      POP
ZURUECK           EQU      RET
VON_AL_NACH_DI    EQU      STOSB
MINUS             EQU      SUB

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; MACROS

INCLUDE INFO.MAC

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CODE SEGMENT
ASSUME CS:CODE,DS:CODE
ORG 100H

BEGINN:        WEITER_MIT    ANFANG

;;;;;;;;;;;;;; INTEGER      ;;;;;;;;;;;;;;;;;;;;;;

act_seite      DB  00
drei_fl        DB  00
fl_zahl        DB  00
fuenf_fl       DB  00
stoer_fl       DB  ?

;;;;;;;;;;;;;; ZEIGER       ;;;;;;;;;;;;;;;;;;;;;;

int_11         DW  ?
video_port     DW  03BAH
wo             DW  00

;;;;;;;;;;;;;; KONSTANTE    ;;;;;;;;;;;;;;;;;;;;;;

emm_sig        DB  'EMM'

;;;;;;;;;;;;;; REIHEN       ;;;;;;;;;;;;;;;;;;;;;;

lwk_buffer     DB  27 DUP (00)
num_string     DB  11 DUP (" "),00

;;;;;;;;;;;;;; STACK  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;

               DB  32   DUP  ("STACK")
UNS_STACK      DB  "* ENDE *"

;;;;;;;;;;;;;; TEXTE  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;

INCLUDE INFO.TXT
```

und deswegen nur das benützen, was für den Programmierer freigegeben wurde.

Bildschirm-Darstellung

Drei Makrofunktionen werden benutzt, um Informationen auf den Bildschirm zu bringen. Diese sind: «SCHREIB_BIS_0, «SCHREIB_AB» und «SCHREIB.CHAR».

Zuerst wird der Cursor am Bildschirm positioniert. Dies ist notwendig, da die benutzte Funktion ab dort schreibt wo sich der Cursor befindet. Siehe «STELLUNG_CHAR» und «STELLUNG_STRING».

Cursor positionieren

Register	BIOS = Interrupt hex 10
AL	Funktion = 02
BH	Aktuelle Seite auf der geschrieben wird (normalerweise = 00)
DH	Zeile (normalerweise 0 und 24)
DL	Spalte (normalerweise zwischen 0 und 7a)

Es wird solange geschrieben, bis ein 0 Charakter vorgefunden wird. Diese Strings, die mit einem 0 beendet werden, gehören fast zur Norm. So z.B. ist dies der Fall bei C genauso wie bei der Schreibung von DOS-Dateinamen, die intern gebraucht werden.

Ermittlung der freien Kapazität eines Datenträgers

Da das Programm auch die freie Kapazität des Laufwerkes (ohne Angabe) ermitteln soll, muss zuerst das aktuelle Laufwerk festgestellt werden. Im Falle einer vorhandenen Laufwerk-Angabe wird die Ermittlung der freien Kapazität auf diese Angabe bezogen (siehe Struktur der Routine «FREI_SPEICHER»). Die Funktion hex 19 von DOS ermöglicht festzustellen, welches Laufwerk aktiv ist.

Ermittlung des aktuellen Laufwerkes

Register	Inhalt
AH	hex 19
Rückkehr	
AL	aktuelles Laufwerk (0 = A, 1 = B, 2 = C, usw.)

Bei den DOS-Funktionen muss man darauf achten, welcher Zusammenhang zwischen den numerischen Werten und der alphabetischen Bezeichnung besteht. Grundsätzlich sind hier zwei Möglichkeiten vorhanden: Entweder bezeichnet 0 das aktuelle Laufwerk und 1 entspricht A usw., oder es beginnt mit 0 = A, 1 = B usw. Hierauf muss man achten, da ansonsten schwerwiegende Programmierfehler entstehen können. Da DOS in diesem Falle numerische Werte (normalerweise zwischen 0 und 27, bzw. A bis Z) liefert, ist es einfach, diese Werte in druckbare ASCII-Zeichen zu übersetzten. Das ASCII-Zeichen A hat den Wert 65. In dem man dem von DOS ausgegebenen numerischen Wert entweder 64 oder 65 dazuaddiert, erhält

man den entsprechenden ASCII-Charakter. Die Funktion hex 36 von DOS liefert Informationen über die Organisation eines beliebigen Datenträgers.

Ermittlung der Datenträgerfunktion

Register	Inhalt
DL	Nummer des Laufwerkes wo sich der Datenträger befindet (0 = aktuell, 1 = A, 2 = B, usw.)
AH	hex 36
Rückkehr	
DX	Gesamtzahl der Informationseinheiten
BX	freie Informationseinheiten
AX	Sektoren pro Informationseinheiten (Achtung: Wert hex FFFF = Fehler)
CX	Anzahl der Bytes je Sektor

Alle unter DOS verwendbaren Datenträger sind einheitlich strukturiert. Der Datenträger wird zuerst in Informationseinheiten (sehr oft auch mit Cluster bezeichnet) eingeteilt. Beim Suchen auf einem Datenträger werden diese Informationseinheiten benutzt. Jede dieser Informationseinheiten wird wiederum in Sektoren eingeteilt. Eine Informationseinheit enthält einen oder mehrere Sektoren (je nach Art des Datenträgers). Zuletzt enthält jeder Sektor eine bestimmte Anzahl von Bytes.

Mit den Werten aus Funktion hex 36 erhalten wir durch einfaches Multiplizieren die uns interessierende Anzahl von Bytes.

Wenn das Laufwerk nicht ansprechbar ist, folgt eine Fehlermeldung. Im Gegensatz zu anderen DOS-Funktionen wird hier nicht das CARRY FLAG gesetzt, sondern im Register AX der Wert hex FFFF eingesetzt. Was diese Fehlermeldung bedeutet, ist nicht ermittelbar. Es kann sowohl sein, dass das Laufwerk temporär gestört ist oder dass grundsätzliche Laufwerkfehler vorhanden sind. Im Falle eines Fehlers meldet das Programm, dass 0 Bytes zur Verfügung stehen.

In den jeweiligen Registern werden Werte, die wir erhalten, in hex-Schreibweise abgesetzt. Hier sind zwei Tä-

Disketten-Service

Auch das Programm INFO können Sie, wie auch die bisher vorgestellten Programme WO, LIES und START auf Diskette anfordern. Die Diskette enthält eine Datei mit dem Quellcode, der von Ihnen mit jedem Texteditor verändert werden kann, die zugehörige Cross-Referenz-Datei sowie das Object-File. Und selbstverständlich als auf allen IBM-kompatiblen Computern lauffähige Version des Programmes. Dies alles für den geringen Unkostenbeitrag von Fr. 10.-- pro Diskette. Bestellen Sie Ihre Diskette mit der dem Heft am Schluss beigehefteten Karte «Disketten-Service» und vergessen Sie bitte nicht anzugeben, ob Sie eine 5.25- oder 3.5-Zoll-Diskette wünschen. Senden Sie uns kein Geld zum voraus. □

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; PROGRAMM AUFBAU
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ANFANG:
        NACH_VON        SP,OFFSET UNS_STACK
        ABRUF           PARAMETER
        ABRUF           wo
BEENDEN:
        PROGRAMM_ENDE

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; HAUPTROUTINEN
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PARAMETER:
        NACH_VON        SI, 80H
        VON_SI_NACH_AL
        VERGLEICH_00    AL
        WENN_UNGLEICH   PAR_VORHANDEN
        NACH_VON        wo, OFFSET FREI_SPEICHER
        WEITER_MIT      PAR_ENDE
PAR_VORHANDEN:
        VON_SI_NACH_AL
        VERGLEICH       AL, 0DH
        WENN_GLEICH     PAR_ENDE
        VERGLEICH       AL, 20H
        WENN_GLEICH     PAR_VORHANDEN
        VERGLEICH       AL, "*"
        WENN_GLEICH     INFO_KONFIG
        VERGLEICH       AL, "?"
        WENN_UNGLEICH   LAUFWERK
        SCHREIB_AB_     info_t
        NACH_VON        wo, OFFSET BEENDEN
        WEITER_MIT      PAR_ENDE
INFO_KONFIG:
        NACH_VON        wo, OFFSET KONFIGURATION
        WEITER_MIT      PAR_ENDE
LAUFWERK:
        GROSS_SCHREIB   AL
        NACH_VON        wo, OFFSET FREI_SPEICHER
PAR_ENDE:
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

KONFIGURATION:
        CLS
        ABRUF           RAHMEN
        AUSSTATTUNG
        NACH_VON        int_11,AX
        ABRUF           MATH_KP
        ABRUF           DISK
        ABRUF           FLOPPY
        ABRUF           SERIELL
        ABRUF           PARALLEL
        ABRUF           TASTATUR
        ABRUF           FL_FORMAT
        ABRUF           VIDEO
        ABRUF           DOS_VERSION
        ABRUF           SPEICHER
        ABRUF           EXPANDED
        ABRUF           ERWEITERT
        ABRUF           OB_DRUCK
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

FREI_SPEICHER:
        VERGLEICH_00    AL
        WENN_UNGLEICH   LWK_ANGABE
        LAUFW_HOLEN
        SUMME           AL, "A"
        REG_NULL        DL
        WEITER_MIT      INST_LWK
LWK_ANGABE:
        BEHALTE         AX
        MINUS           AL, "A" - 1
        NACH_VON        DL, AL
        RUECKRUF        AX
INST_LWK:
        NACH_VON        DI, OFFSET laufwerk_t[11]
        VON_AL_NACH_DI
        NACH_VON        DI, OFFSET lfwk_tot_t[11]
```

```
        VON_AL_NACH_DI
        LAUFW_DATEN
        VERGLEICH       AX, 0FFFFH
        WENN_UNGLEICH   LWK_VORHANDEN
        REG_NULL        AX
LWK_VORHANDEN:
        BEHALTE         DX
        MAL             CX
        BEHALTE         AX
        MAL             BX
        ABRUF           DEZIMAL_32
        SCHREIB_AB_     neu_zeile
        SCHREIB_AB_     kl_rahmen_o
        SCHREIB_AB_     neu_zeile
        SCHREIB_AB_     laufwerk_t
        SCHREIB_AB_     num_string
        SCHREIB_AB_     kl_rahmen_e
        SCHREIB_AB_     neu_zeile
        RUECKRUF        AX
        RUECKRUF        DX
        MAL             DX
        ABRUF           DEZIMAL_32
        SCHREIB_AB_     lfwk_tot_t
        SCHREIB_AB_     num_string
        SCHREIB_AB_     kl_rahmen_e
        SCHREIB_AB_     neu_zeile
        SCHREIB_AB_     kl_rahmen_m
        SCHREIB_AB_     neu_zeile

        BEHALTE         ES
        NACH_VON        AX, CS
        NACH_VON        ES, AX
        NACH_VON        BX, 150
        SPEICHER_NEU
        NACH_VON        BX, 2CH
        NACH_VON        ES, [BX]
        REG_NULL        BX
        SPEICHER_NEU
        SPEICHER_FREI
        NACH_VON        AX, BX
        SUMME           AX, 151
        NACH_VON        CX, 10H
        MAL             CX
        RUECKRUF        ES
        ABRUF           DEZIMAL_32
        SCHREIB_AB_     spch_fr_t
        SCHREIB_AB_     num_string
        SCHREIB_AB_     kl_rahmen_e
        SCHREIB_AB_     neu_zeile
        SPEICHER_TOTAL
        REG_NULL        DX
        NACH_VON        BX, 1024
        MAL             BX
        ABRUF           DEZIMAL_32
        SCHREIB_AB_     spch_tot_t
        SCHREIB_AB_     num_string
        SCHREIB_AB_     kl_rahmen_e
        SCHREIB_AB_     neu_zeile
        SCHREIB_AB_     kl_rahmen_u
        SCHREIB_AB_     neu_zeile
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; AUSSTATTUNGS ROUTINEN
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MATH_KP:
        SCHREIB_BIS_0   math_t, 030DH
        NACH_VON        DL, NULL
        TEST            int_11, 2
        WENN_GLEICH     MATH_ENDE
        ERHOEHE         DL
MATH_ENDE:
        SCHREIB_CHAR    DL, 0322H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;

DISK:
        SCHREIB_BIS_0   fest_t, 040DH
        ANZAHL_FEST_DL
        SUMME           DL, NULL
        SCHREIB_CHAR    DL, 0422H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;
FLOPPY:
        SCHREIB_BIS_0   floppy_t, 050DH
        ANZAHL_DISK_DL
        NACH_VON        fl_zahl, DL
```

tigkeiten notwendig: Erstens diese hex-Werte in Dezimalnotation zu übersetzen und zweitens diese Dezimalwerte in druckbare ASCII-Zeichen umzuwandeln. Dies erledigt eine sehr nützliche Routine (DEZIMAL_32). Mit Hilfe dieser Routine ist es möglich, einen hex-Wert, der sich in den Registern AX und DX befindet, in einen numerischen String zu übersetzen, der druckbare Zeichen beinhaltet. Für alle, die mit dieser Aufgabe konfrontiert werden, ist die genaue Untersuchung dieser Routine empfehlenswert. Grob wird von der Basis 16 (hex) auf die Basis 10 (dezimal) umgewandelt. Danach wird der jeweils resultierende Wert in ein ASCII-Zeichen übersetzt. Da der Wert 0 dem druckbaren ASCII-Zeichen 48 entspricht, ist dies relativ einfach.

Um die freie Kapazität des Grundspeichers zu ermitteln, muss man zuerst davon ausgehen, dass während das Programm läuft (da es sich um eine COM-Datei handelt) der Gesamtspeicher in Anspruch genommen wird. Daher ist keine freie Kapazität vorhanden. Zuerst muss man den Speicherbedarf des laufenden Programmes auf ein bestimmtes Mass reduzieren. Danach wird die freie Kapazität des Grundspeichers ermittelt. Von dieser nun vorhandenen freien Kapazität muss die dem Programm zugestandene dazuaddiert werden. Das Ergebnis: Freie Kapazität des Grundspeichers, die vorhanden ist beim Verlassen des Programmes (eine genaue Beschreibung der Speicherbedarf-Struktur befindet sich im Programm LIES im M+K 88-2).

Mit Hilfe des Interrupts hex 12 ist es möglich, die Grösse des Grundspeichers zu ermitteln. Diese Grösse wird in 1 KByte Blöcke wiedergegeben. Mit Hilfe der Funktion DEZIMAL_32 wird dieser Wert in Bytes transformiert (1 KByte = 1'024 Bytes) und ausgegeben. An dieser Stelle sei wiederum auf ein Problem hingewiesen. Die Angaben von Kapazitäten in den DOS-Funktionen sind nicht einheitlich. Die zwei hier besprochenen Funktionen z.B. liefern einerseits Werte in Bytes, andererseits Werte in KBytes-Blöcken.

Hardware-Information

Der Interrupt hex 11 liefert eine Anzahl wesentlicher Informationen über die jeweilige Ausrüstung des angesprochenen Computers.

Ermittlung der Ausrüstung

Register BIOS Interrupt hex 11

Rückkehr

AX	Bits	15,14	Anzahl der parallelen Schnittstellen
		13	ohne Bedeutung
		12	Spielport
		11,10,9	Anzahl der seriellen Schnittstellen
		8	ohne Bedeutung
		7,6	Anzahl der Disketten-Laufwerke (wenn Bit 0 = 1, dann 00 = 01, 01 = 02, usw.)
		5,4	Bildschirmmodus
		3,2	Speichergrösse auf Hauptplatine
		1	mathematischer Coprozessor
		0	Disketten-Laufwerk vorhanden (00 = nein, 1 = ja)

Für das vorliegende Programm wurden die Informationen über mathematischen Coprozessor, Anzahl der Schnittstellen und Anzahl der physikalisch vorhandenen Disketten-Laufwerke benutzt.

Bei der Ermittlung der jeweiligen Werte wird dreistufig vorgegangen:

1. Die Bits innerhalb des AX-Registers werden überprüft.
2. Die Information über gesetzte Bits wird in Zahlen umgewandelt.
3. Diese Zahlen werden ausgedruckt.

Bei den Schnittstellen werden sehr oft die Beschaffenheit derselben mit dem angeschlossenen Peripherie-Gerät verwechselt. Wir erhalten Informationen über vorhandene parallele oder serielle Schnittstellen. An einer parallelen Schnittstelle wird meistens der Drucker angeschlossen. Drucker können aber auch über serielle Schnittstellen angesprochen werden. Hier muss betont werden, dass die physikalisch vorhandenen Schnittstellen ausgewiesen werden und nicht ob Peripherie-Geräte angeschlossen sind oder nicht.

Bei der Anzahl der physikalisch vorhandenen Disketten-Laufwerke muss man berücksichtigen, dass die Aussage 00 (auf Bit 6 und 7) bedeutet, dass ein Disketten-Laufwerk vorhanden ist. Die effektiv vorhandene Anzahl ist daher immer um 1 grösser als die ausgewiesene Anzahl. Diese muss davon abhängig gemacht werden, ob überhaupt Disketten-Laufwerke vorhanden sind oder nicht (1 oder 0 bei Bit 0).

Wenn ein mathematischer Coprozessor richtig installiert wurde, erhalten wir ebenfalls über den Interrupt hex 11 die Meldung, ob vorhanden oder nicht. Es kann jedoch der Fall sein, dass ein mathematischer Coprozesor installiert, aber die Computer-Schalter nicht richtig gesetzt wurden. In diesem Falle meldet das Programm keinen mathematischen Coprozessor, da dieser nicht ansprechbar ist.

Im niedrigsten Speicherbereich (ab Segment 00) finden wir eine Anzahl von Informationen über den Zustand der Hardware. Im Laufe der Zeit wurde teilweise die Bedeutung dieser Stellen geändert. Deswegen muss diese Information mit Vorsicht verwendet werden. Das heisst man sollte in den technischen Handbüchern überprüfen, ob und wie ab welcher Version eine Information in diesem Bereich geändert wurde. Es ist sicher, dass wir auf Stelle hex 475 dieses Bereiches die Anzahl der Festplatten, die physikalisch vorhanden sind, vorfinden. Hier muss das physikalische Vorhandensein in den Vordergrund gestellt werden. Eine Festplatte kann mehrere, unabhängig voneinander arbeitende Teile (Partitions) haben. Es kann daher zu Verwechslungen führen, wenn man auf die Festplatte im Grunde genommen zwei oder mehr Laufwerke (z.B. C, D und E) installiert hat.

Obwohl Tastaturen recht mannigfaltig gestaltet werden können, gibt es nur zwei Grundtypen. Eine Standard- und eine erweiterte Tastatur. Die erweiterte Tastatur erkennt man leicht an der Tatsache, dass die Cursor-Tasten nicht nur im numerischen Felde sondern auch getrennt vorhanden sind. Hinzu kommt meistens eine Bildschirm-Abdrucktaste, eine Systemtaste und eine Pausetaste. Diese erweiterte Tastatur verlangt einen besonderen Treiber. Ob der Treiber vorhanden ist oder nicht, kann mit Hilfe der Funktion 12 des Interrupts hex 16 festgestellt werden. Diese Funktion ist für den erweiterten Tastaturcode (101, bzw. 102 Tasten) gedacht. Wenn korrekte Antworten erhalten werden, heisst dies, dass die Möglichkeit besteht, eine erweiterte Tastatur zu verwenden. Dementsprechend wird in der Routine «TASTATUR» vorgegangen. Zuerst wird un-

```
        SUMME           DL, NULL
        SCHREIB_CHAR    DL, 0522H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SERIELL:
        SCHREIB_BIS_0   seriell_t, 032FH
        ANZAHL_SRLL_AH
        SUMME           AH, NULL
        SCHREIB_CHAR    AH, 0344H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PARALLEL:
        SCHREIB_BIS_0   parallel_t, 042FH
        ANZAHL_PARL_AH
        SUMME           AH, NULL
        SCHREIB_CHAR    AH, 0444H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TASTATUR:
        SCHREIB_BIS_0   erw_tast_t, 052FH
        NACH_VON        DL, NULL
        PRUEF_ERW_TST
        WENN_GLEICH     TEST_ERW
        WEITER_MIT      TAST_ENDE
TEST_ERW:
        UMSCHALT_GROSS
        PRUEF_ERW_TST
        WENN_UNGLEICH   TEST_ENDE
        ERHOEHE         DL
TEST_ENDE:
        UMSCHALT_GROSS
TAST_ENDE:
        NACH_VON        AX, CS
        NACH_VON        ES, AX
        SCHREIB_CHAR    DL, 0544H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;

FL_FORMAT:
        REG_NULL        DX
        NACH_VON        BP, DX
INT_13:
        LAUFWERK_PAR
        WENN_K_FEHLER   FL_TEST
        WEITER_MIT      FL_STANDARD
FL_TEST:
        REG_OB_NULL     AX
        WENN_GLEICH     FL_PRUEF
        WEITER_MIT      FL_STANDARD
FL_PRUEF:
        REG_OB_NULL     BL
        WENN_UNGLEICH   FORMAT
        ERHOEHE         stoer_fl
        WEITER_MIT      FL_FORT
FORMAT:
        VERGLEICH       BL, 02
        WENN_GROESSER   FL_DREI
        ERHOEHE         fuenf_fl
        WEITER_MIT      FL_FORT
FL_DREI:
        ERHOEHE         drei_fl
FL_FORT:
        ERHOEHE         BP
        NACH_VON        DX, BP
        VERGLEICH       DL, fl_zahl
        WENN_GROSS_GL   FL_END
        WEITER_MIT      INT_13
FL_END:
        SCHREIB_BIS_0   disk_par_t, 070DH
        NACH_VON        AL, NULL
        SUMME           fuenf_fl, AL
        SUMME           drei_fl, AL
        SUMME           stoer_fl, AL
        SCHREIB_CHAR    fuenf_fl, 071CH
        SUMME           BP, 0EH
        SCHREIB_CHAR    drei_fl, BP
        SUMME           BP, 0EH
        SCHREIB_CHAR    stoer_fl, BP
        ZURUECK         ;;;;;;;;;;;;;;;;;;
FL_STANDARD:
        SCHREIB_BIS_0   diskp_st_t, 070DH
        ZURUECK         ;;;;;;;;;;;;;;;;;;

VIDEO:
        SCHREIB_BIS_0   video_t, 080DH
        VIDEO_AL
        VERGLEICH       AL, 07
```

```
        WENN_GLEICH     OB_HERCULES
        WEITER_MIT      NICHT_MONO
OB_HERCULES:
        NACH_VON        BL, 05
        NACH_VON        DX, video_port
        PORT_MLD_LT_AL
        NACH_VON        AH, AL
        NACH_VON        CX, 8000H
HERC_LOOP:
        PORT_MLD_LT_AL
        VERGLEICH       AL, AH
        WENN_GLEICH     KN_HERC_ZEICHN
        MINUS_1         BL
        WENN_GLEICH     HERCULES
KN_HERC_ZEICHN:
        SCHLEIFE        HERC_LOOP
        SCHREIB_BIS_0   mono_t, 0826H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;
HERCULES:
        SCHREIB_BIS_0   herc_t, 0826H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;
NICHT_MONO:
        EGA_INFO_BX
        VERGLEICH       BL, 10H
        WENN_UNGLEICH   EGA
        SCHREIB_BIS_0   cga_t, 0826H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;
EGA:
        SCHREIB_BIS_0   ega_t, 0826H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;

DOS_VERSION:
        SCHREIB_BIS_0   dos_t, 090DH
        DOS_VERSION_AX
        NACH_VON        BX, AX
        ZEIGT_AUF       DI, dos_v
        SUMME           AL,NULL
        VON_AL_NACH_DI
        ERHOEHE         DI
        NACH_VON        AL, BH
        REG_NULL        AH
        DEZIMAL_AL
        SCHREIB_BIS_0   dos_v, 091DH
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;

SPEICHER:
        SCHREIB_BIS_0   speicher_t, 0B0DH
        SPEICHER_TOTAL
        REG_NULL        DX
        ABRUF           DEZIMAL_32
        SCHREIB_BIS_0   num_string[7], 0B23H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;

EXPANDED:
        SCHREIB_BIS_0   exp_sp_t, 0B2FH
        INT_ADRESSE     67H
        NACH_VON        SI,OFFSET emm_sig
        NACH_VON        DI,0AH
        NACH_VON        CX,3
        VERGL_DI_SI_CX
        BEHALTE         CS
        RUECKRUF        ES
        WENN_UNGLEICH   EXP_NULL
        EXPANDED_AX
        ABRUF           DEZIMAL_32
        SCHREIB_BIS_0   num_string[7], 0B41H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;
EXP_NULL:
        SCHREIB_CHAR    NULL, 0B44H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;

ERWEITERT:
        SCHREIB_BIS_0   erw_sp_t, 0C2FH
        ERWEITERT_AX
        WENN_FEHLER     ERW_NULL
        VERGLEICH       AH,88H
        WENN_GLEICH     ERW_NULL
        REG_OB_NULL     AX
        WENN_GLEICH     ERW_NULL
        REG_NULL        DX
        ABRUF           DEZIMAL_32
        SCHREIB_BIS_0   num_string[7], 0C41H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;
ERW_NULL:
        SCHREIB_CHAR    NULL, 0C44H
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

tersucht, ob diese Funktion die gleiche Meldung gibt über den Status der Tastatur, wie im niedrigen Bereich hex 417. Hier ist nämlich ein Duplikat der Statusmeldung vorhanden. Wenn die neue Funktion und diese Stelle übereinstimmen, könnte es auch Zufall sein. Um sicher zu gehen, wird vom Programm die Grosschreibung umgeschaltet und überprüft, ob dies von der neuen Funktion registriert wird. Im positiven Falle kann man sicher sein, dass der erweiterte Tastaturtreiber vorhanden ist.

Es ist wichtig zu wissen, welche Disketten (5.25 oder 3.5 Zoll) in einem Computer vorhanden sind. Diese Information liefert der Interrupt hex 13.

Ermittlung der Datenträger Parameter

Register	BIOS Interrupt hex 13
AH	08
DL	Disketten-Laufwerknummer (00 = A, 01 = B, usw.)
Rückkehr	
AX	00
BL	Bits 3 - 0 Format 00 = unbekannt 01 = 5.25 Zoll, 360 KByte 02 = 5.25 Zoll, 1,2 MB 03 = 3.5 Zoll, 720 KByte 04 = 3.5 Zoll, 1,4 MB
BH	0
DL	Anzahl der Disketten-Laufwerke

Vorsicht ist geboten bei der Interpretation der Rückkehr-Register. Mehrere Antworten, die auf Fehler hindeuten, sind möglich. Aeltere Modelle antworten, wenn auf Interrupt 13 Funktion 8 angesprochen und das Laufwerk ein Disketten-Laufwerk ist, mit CARRY FLAG gesetzt. Dies ist darauf zurückzuführen, dass diese Funktion ursprünglich nur für die Festplatte gedacht wurde. Wenn die Funktion 08 im Register AH nichts verändert, zeigt dies, dass es sich um Modelle handelt, die 3.5-Laufwerke nicht ansteuern können. Die Möglichkeit ist auch gegeben, dass das Register AX insgesamt nur 0 beinhaltet. Dies bedeutet, dass 3.5-Laufwerke ansprechbar sind. An der angesprochenen Stelle kann sich jedoch kein 3.5-Laufwerk befinden. Entsprechend werden die Antworten, die wir erhalten, bewertet. Das Programm ermöglicht es noch nicht, zwischen Laufwerken mit hoher Auszeichnungsdichte (1,2 bzw. 1,4 MB) und normalen (360 und 720 KByte) zu unterscheiden. Dies ist auch nicht notwendig. Die Laufwerke mit hoher Aufzeichnungsdichte sind auch imstande, normale Disketten zu lesen. Da die Konfigurationsmeldung zeigen soll, welche Datenträger der Computer zu verarbeiten imstande ist, genügt dies vollkommen.

Die aktuelle Videomodalität des Computers kann durch Interrupt hex 10 angezeigt werden. Es ist möglich, klar und deutlich zwischen einer Monochrom- und einer Farbgrafik-Karte (CGA) zu unterscheiden. *Achtung:* Auch wenn eine Farbgrafik-Karte in Betrieb ist, besagt dies noch nicht, dass auch der Bildschirm imstande ist, Farben korrekt zu verarbeiten. Deswegen sieht man bei vielen Programmen, die nur ermitteln, ob ein Farbgrafik-Adapter vorhanden ist, Fehler. Es wird angenommen, dass auch ein Farbbildschirm vorhanden ist, was nicht immer der Fall sein muss. Das Ergebnis: Wenn ein Zweifarb-Bildschirm an einer

Farbgrafik-Karte angeschlossen wird und das Programm versucht, Farben zu benutzen, erhält man als Ergebnis unklare Bildschirme, die kaum lesbar sind. Insbesondere bei amerikanischen Programmen ist dieser Fehler sehr oft bemerkbar. Auch wenn sie dann als Option die Möglichkeit einer Farbumstellung gestatten, ist dies sehr oft nicht möglich, da man schon den Ausgangs-Bildschirm kaum lesen kann. An dieser Stelle eine Empfehlung: Programme sollten grundsätzlich in schwarz-weiss oder höchster Kontraststufe beginnen und erst dann auf Wunsch des Anwenders auf Farbe umschalten. Dies sichert klare Verhältnisse von Anfang an und erspart dem Anwender Aerger. Es genügt nicht zu wissen, ob die Videomodalität einer Monochrom- oder Farbgrafik-Karte enspricht, sondern es muss ein Schritt weitergegangen werden. Monochrom-Videomodalitäten können von einer normalen Monochrom-Karte, aber auch von einer Hercules-Karte her gesteuert werden.

Es sei nur kurz geschildert, wie der Unterschied zwischen einer Standard-Monochrom-Karte und einer Hercules-Karte festgestellt wird. Laut Hersteller-Anweisung wird eine Ansprechmöglichkeit, auf die nur Hercules-Karten antworten, benützt. Wenn wir über diese Stelle eine Antwort erhalten, heisst dies, dass die Hercules-Karte vorhanden sein muss. Um sicher zu gehen, dass hier keine falschen Antworten verwertet werden, wird der Vorgang fünf mal wiederholt.

Eine Video-Modalität, die Farbgrafik-Adapter voraussetzt, bedeutet, dass sowohl eine CGA- als auch eine EGA-Karte vorhanden sein können. die EGA-Karte ist in den niedrigen Video-Modalitäten 100 % mit der CGA-Karte kompatibel. Interrupt hex 10 stellt für die EGA-Karte neue Funktionen zur Verfügung.

EGA-Information

Register	BIOS Interrupt hex 10
AH	hex 12
BL	hex 10
Rückkehr	
BH	Farbmodalität 00 = Farbe 01 = monochrom
BL	EGA-Speicher 00 = 64 01 = 128 02 = 192 03 = 256

Wenn diese neue Funktion eine korrekte Meldung enthält, heisst dies einwandfrei, dass eine EGA-Karte installiert wurde. Es gibt heute weitere Video-Modalitäten, die nicht nur in den niedrigen Bereichen CGA-kompatibel sondern ebenfalls die EGA-Modalitäten erlauben und darüber hinaus noch weitere feinere Auflösungen grafisch oder farblich ermöglichen. Diese neuesten Video-Karten sind noch nicht sehr verbreitet. Da es kaum Programme hiefür gibt, wurde das Vorhandensein dieser Karten nicht untersucht. Ist eine EGA-Karte vorhanden, bedeutet dies, dass mindestens diese emuliert wird. Es könnte aber auch bedeuten, dass weitere noch bessere Leistungen von der installierten Karte erbringbar sind.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; HILFSROUTINEN
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

INCLUDE DEZIMAL.032

OB_DRUCK:
        SCHREIB_BIS_0   druck_t, 1400H
HOLE_ANTWORT:
        EINGABE_TAST
        GROSS_SCHREIB   AL
        VERGLEICH       AL, "N"
        WENN_UNGLEICH   OB_ENTER
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;
OB_ENTER:
        VERGLEICH       AL, ENTER
        WENN_UNGLEICH   OB_JA
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;
OB_JA:
        VERGLEICH       AL, "J"
        WENN_GLEICH     AUSDRUCK
        BEEP
        WEITER_MIT      HOLE_ANTWORT
AUSDRUCK:
        SCHREIB_BIS_0   vor_druck_t, 1400H
        EINGABE_TAST
        SCHREIB_BIS_0   null_druck_t, 1400H
        PRINT_SCREEN
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;

STELLUNG_CHAR:
        BEHALTE         DX
        CURSOR_NACH
        RUECKRUF        DX
        SCHREIB_DL
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;

STELLUNG_STRING:
        CURSOR_NACH
SCHREIBEN_STRING:
        VON_SI_NACH_AL
        REG_OB_NULL     AL
        WENN_GLEICH     STL_STRING_ENDE
        NACH_VON        DL, AL
        SCHREIB_DL
        WEITER_MIT      SCHREIBEN_STRING
STL_STRING_ENDE:
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;

RAHMEN:
        SCHREIB_BIS_0   rahmen_oben, 0101H
        NACH_VON        CX, 0DH
        NACH_VON        DL, 186
        NACH_VON        BP, 0201H
SEITEN:
        SCHREIB_CHAR    DL, BP
        SUMME           BP, 77
        SCHREIB_CHAR    DL, BP
        SUMME           BP,179
        SCHLEIFE        SEITEN
        SCHREIB_BIS_0   rahmen_unten, BP
        SCHREIB_BIS_0   muk_t, 0E04H
        SCHREIB_BIS_0   cp_t, 0E3CH
        ZURUECK         ;;;;;;;;;;;;;;;;;;;;;;;;;;;


CODE ENDS
END  BEGINN
```

```
;INFO.MAC
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ANZAHL_DISK_DL MACRO
               NACH_VON       DX, int_11
               AND            DL, 0C0H
               NACH_VON       CL, 02
               ROL            DL, CL
               ERHOEHE        DL
               ENDM           ;;;;;;;;;;;;;;;;;;;;
ANZAHL_FEST_DL MACRO
               REG_NULL       AX
               NACH_VON       ES, AX
```

```
                NACH_VON        DL, ES:[475H]
                NACH_VON        AX, CS
                NACH_VON        ES, AX
                ENDM            ;;;;;;;;;;;;;;;;;;
ANZAHL_PARL_AH  MACRO
                NACH_VON        AX, int_11
                AND             AH, 0C0H
                NACH_VON        CL, 02
                ROL             AH, CL
                ENDM            ;;;;;;;;;;;;;;;;;;
ANZAHL_SRLL_AH  MACRO
                NACH_VON        AX, int_11
                AND             AH, 0EH
                ROR             AH, 1
                ENDM            ;;;;;;;;;;;;;;;;;;
AUSSTATTUNG     MACRO
                INT             11H
                ENDM            ;;;;;;;;;;;;;;;;;;
BEEP            MACRO
                NACH_VON        AX, 0E07H
                INT             10H
                ENDM            ;;;;;;;;;;;;;;;;;;
CLS             MACRO
                NACH_VON        AH,0FH
                INT             10H
                NACH_VON        act_seite, BH
                NACH_VON        AH,8
                INT             10H
                NACH_VON        BH,AH
                REG_NULL        CX
                NACH_VON        DX,184FH
                NACH_VON        AX,0600H
                INT             10H
                ENDM            ;;;;;;;;;;;;;;;;;;
CURSOR_NACH     MACRO
                NACH_VON        BH, act_seite
                NACH_VON        DX,BP
                NACH_VON        AH,2
                INT             10H
                ENDM            ;;;;;;;;;;;;;;;;;;
DEZIMAL_AL      MACRO
                AAM
                SUMME           AX, 3030H
                XCHG            AH, AL
                STOSW
                ENDM            ;;;;;;;;;;;;;;;;;;
DOS_VERSION_AX  MACRO
                NACH_VON        AH,30H
                INT             21H
                ENDM            ;;;;;;;;;;;;;;;;;;
EGA_INFO_BX     MACRO
                NACH_VON        AH, 12H
                NACH_VON        BL, 10H
                INT             10H
                ENDM            ;;;;;;;;;;;;;;;;;;
EINGABE_TAST    MACRO
                NACH_VON        AX, 0C07H
                INT             21H
                ENDM            ;;;;;;;;;;;;;;;;;;
ERWEITERT_AX    MACRO
                NACH_VON        AH,88H
                INT     15H
                ENDM            ;;;;;;;;;;;;;;;;;;
EXPANDED_AX     MACRO
                NACH_VON        AH,42H
                INT             67H
                NACH_VON        AX,16
                MAL             DX
                ENDM            ;;;;;;;;;;;;;;;;;;
GROSS_SCHREIB   MACRO           register
                AND             register, 05FH
                ENDM            ;;;;;;;;;;;;;;;;;;
INT_ADRESSE     MACRO           interrupt
                NACH_VON        AH,35
                NACH_VON        AL, interrupt
                INT             21H
                ENDM            ;;;;;;;;;;;;;;;;;;
LAUFW_DATEN     MACRO
                NACH_VON        AH, 036H
                INT             21H
                ENDM            ;;;;;;;;;;;;;;;;;;
LAUFW_HOLEN     MACRO
                NACH_VON        AH, 019H
                INT             21H
                ENDM            ;;;;;;;;;;;;;;;;;;
```

Das DOS-Betriebssystem kann an und für sich maximal einen Grundspeicher von höchstens 640 KByte verwenden. Man war bis vor kurzem auch recht zufrieden, wenn man einen Computer besass, der einen Grundspeicher mit einer Ausstattung von 640 KByte hatte. Die technologische Entwicklung macht es heute jedoch möglich, normale Computer mit viel mehr Grundspeicher-Kapazität auszustatten. Jenseits der 640 KByte, die DOS verwalten kann, unterscheidet man zwischen expandierten und erweiterten Speichern. Auf die technischen Einzelheiten soll im Rahmen dieses Artikels nicht eingegangen werden. Diese Speicher-Erweiterungsmöglichkeiten werden heute schon von manchen Programmen genutzt und es ist daher wichtig zu wissen, wieviel Speicherbedarf oberhalb der von DOS ansprechbaren Grenze verfügbar ist.

Bei den expandierten Speichern muss man bei der Feststellung ob vorhanden und wieviel beinhaltend von den Hersteller-Empfehlungen ausgehen. Interrupts werden nicht nur von DOS oder BIOS benutzt, sondern stehen auch sowohl Herstellern als auch Programmierern zur Verfügung. Die meisten Hersteller folgen im Falle des expandierten Speichers der Empfehlung, den Interrupt hex 67 zur Ansteuerung dieses Speicherbereiches zu benutzen. Wenn ein expandierter Speicher vorhanden ist, wird an der jeweiligen Adresse festgestellt, ob eine entsprechende Signatur (EMM) da ist. Ist keine vorhanden, kann man schon davon ausgehen, dass keine Speicher dieser Art installiert wurden. Wenn die Signatur vorhanden ist, kann man eine Funktion dieses Interrupts benutzen; nämlich hex 42, die uns mitteilt (im Register AX), wieviel Paragraphen (ein Paragraph ist 16 Bytes) in diesem Speicherbereich verfügbar sind.

Der erweiterte Speicherbereich wird direkt von der Konfiguration her ermittelt. Ab Einführung der ATs wurde eine besondere Funktion des Interupts hex 15 dazu bestimmt, um die Kapazität des erweiterten Speichers wiederzugeben.

Erweiterter Speicher	
Register	BIOS Interrupt hex 15
AH	hex 88
Rückkehr	
AX	Anzahl der 1 KByte Speicherbereiche oberhalb 1 MB

Hier muss die Antwort ebenfalls mit Vorsicht interpretiert werden. Wie bei anderen Funktionen, die erst später hinzugekommen sind, sind mehrere Fehler-Antworten möglich. Wenn kein erweiterter Speicher vorhanden ist. Um die Funktion abzurufen, muss im AH der Wert hex 88 eingege-

Ermittlung der DOS-Version	
Register	Inhalt
AH	hex 30
Rückkehr	
AL	Hauptversion
AH	Unterversion

```
LAUFWERK_PAR    MACRO
                BEHALTE         ES
                NACH_VON        AH, 08
                INT             13H
                RUECKRUF        ES
                ENDM            ;;;;;;;;;;;;;;;;;;;
PORT_MLD_LT_AL  MACRO
                IN              AL, DX
                AND             AL, 80H
                ENDM            ;;;;;;;;;;;;;;;;;;;
PRINT_SCREEN    MACRO
                INT             05H
                ENDM            ;;;;;;;;;;;;;;;;;;;
PROGRAMM_ENDE   MACRO
                NACH_VON        AH,4CH
                INT             21H
                ENDM ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PRUEF_ERW_TST   MACRO
                REG_NULL        AX
                NACH_VON        ES, AX
                NACH_VON        AH,12H
                INT             16H
                VERGLEICH       AL,ES:[417H]
                ENDM            ;;;;;;;;;;;;;;;;;;;
REG_NULL        MACRO           register
                XOR             register, register
                ENDM            ;;;;;;;;;;;;;;;;;;;
REG_OB_NULL     MACRO           register
                OR              register, register
                ENDM            ;;;;;;;;;;;;;;;;;;;
SCHREIB_BIS_0   MACRO           string, ort
                ZEIGT_AUF       SI,string
                NACH_VON        BP, ort
                ABRUF           STELLUNG_STRING
                ENDM            ;;;;;;;;;;;;;;;;;;;
SCHREIB_AB_     MACRO           string
                ZEIGT_AUF       SI,string
                ABRUF           SCHREIBEN_STRING
                ENDM            ;;;;;;;;;;;;;;;;;;;
SCHREIB_CHAR    MACRO           char, ort
                NACH_VON        DL, char
                NACH_VON        BP, ort
                ABRUF           STELLUNG_CHAR
                ENDM            ;;;;;;;;;;;;;;;;;;;
SCHREIB_DL      MACRO
                NACH_VON        AH, 02
                INT             21H
                ENDM            ;;;;;;;;;;;;;;;;;;;
SPEICHER_FREI   MACRO
                NACH_VON    BX, 0FFFFH
                NACH_VON    AH, 48H
                INT         21H
                ENDM        ;;;;;;;;;;;;;;;;;;;
SPEICHER_NEU    MACRO
                NACH_VON    AH, 4AH
                INT         21H
                ENDM        ;;;;;;;;;;;;;;;;;;;
SPEICHER_TOTAL  MACRO
                INT         12H
                ENDM        ;;;;;;;;;;;;;;;;;;;
UEBER_DI_SI_CX  MACRO
                REPZ        LODSB
                ENDM        ;;;;;;;;;;;;;;;;;;;
UMSCHALT_GROSS  MACRO
                XOR         BYTE PTR ES:[417H],80H
                ENDM        ;;;;;;;;;;;;;;;;;;;
VERGL_DI_SI_CX  MACRO
                REPZ        CMPSB
                ENDM        ;;;;;;;;;;;;;;;;;;;
VERGLEICH_00    MACRO       register
                OR          register, register
                ENDM        ;;;;;;;;;;;;;;;;;;;
VIDEO_AL        MACRO
                NACH_VON    AH, 0FH
                INT         10H
                ENDM        ;;;;;;;;;;;;;;;;;;;
```

ben werden. Da das gleiche Register zur Rückmeldung der Kapazität benutzt wird, muss festgestellt werden, ob unter Umständen die Antwort ebenfalls in AH hex 88 lautet (die Funktion tat einfach nichts). Wenn nicht einmal die Funktion ansprechbar ist, kann man sicher sein, dass kein erweiterter Speicherbereich verfügbar ist.

Zuallerletzt sei auf die Ermittlung der DOS-Version hingewiesen.

Es wäre sinnvoll, wenn diese Ermittlung, wie im vorliegenden Falle gezeigt, öfters benutzt würde. Programme benutzen teilweise Funktionen und Interrupts, die erst ab einer bestimmten DOS-Version eingeführt werden.

Ausdruck der Konfiguration

Sicher ist es wünschenswert, die Konfiguration nicht nur am Bildschirm darzustellen, sondern einen Abdruck derselben erhalten zu können. Das Programm erledigt diese Aufgabe auf einfachste Weise. Zuerst wird noch am Bildschirm die Frage gestellt, ob der Abdruck gewünscht wird oder nicht. Wenn die Antwort «Ja» lautet, wird zuerst der Teil des Bildschirmes gelöscht, der die Frage beinhaltete. Der Interrupt hex 5 dient dazu, Bildschirm-Abdrucke zu erhalten. Dieser Interrupt wird normalerweise von der PRT-SC (Bildschirm-Abdrucktaste) gerufen. Um lästige Fehler zu vermeiden, wird vom Anwender verlangt, er soll den Drucker kontrollieren. Diese Aufforderung, die am Bildschirm erscheint, wird wiederum gelöscht und danach die Bildschirm-Abdruckfunktion gerufen. Man erhält danach auf dem Drucker eine genaue Abbildung des Bildschirmes, d.h. der Konfiguration.

Technische Bemerkung

Eine Reihe weiterer Konfigurationsmerkmale (technische Hinweise zur Beschaffenheit der Laufwerke, Geschwindigkeitsmerkmale, usw.) sind ermittelbar. Das vorliegende Programm versucht jedoch in einer möglichst kurzen Zeit, die jeweils effektiv benötigte Information zu geben. Technische Finessen, mit denen die meisten Anwender überhaupt nichts anzufangen wissen, werden nicht ermittelt. In diesem Programm zeigt sich nochmals der Vorteil der Assembler-Sprache. Man kann auf Maschinen-Ebene mit dem Computer kommunizieren, schnell die Information erhalten, die man braucht und trotzdem den Speicher nicht übermässig in Anspruch nehmen. In diesem Programm wurde die schon bekannte Technik der INCLUDE-Dateien benutzt. Obwohl das Programm knappe 2'800 Bytes in Anspruch nimmt, wurde nicht darauf verzichtet, es in einer ansprechenden grafischen Form auf den Bildschirm zu bringen.

Die Routine «DEZIMAL_32» wird in Original Macroassembler (ohne klärende Begriffe) wiedergegeben. Dies wegen der Möglichkeit, diese sehr nützlichen Programme in C- oder kompilierte BASIC-Programme einzubauen. □

HEXEDIT, ein Editor für die Bearbeitung von HEX-Dumps unter MS-DOS

Herkömmliche Dump-Programme bieten in der Regel keine Möglichkeit, die erzeugten Dumps zu bearbeiten. Gerade der Programmentwickler benötigt aber oft ein Hilfsmittel, um Dateien in hexadezimaler Schreibweise bearbeiten zu können. Das Programm HEXEDIT erlaubt sowohl die Bearbeitung einer Datei in hexadezimaler als auch in herkömmlicher Text-Schreibweise.

Daniel Aebi

Um Dateien in hexadezimaler Schreibweise bearbeiten zu können, werden oft Debugger wie z. B. das unter MS-DOS verfügbare Programm DEBUG verwendet. Diese Programme bieten jedoch meist eine Fülle von Möglichkeiten und sind deshalb in der Anwendung recht kompliziert. Das hier beschriebene Programm HEXEDIT soll die Lücke schliessen zwischen den einfachen Dump-Programmen, die nur das hexadezimale Auflisten einer Datei erlauben, und den Debuggern. Das Programm bietet folgende Möglichkeiten:

- Darstellen einer Datei in hexadezimaler und Text-Notation, dabei kann beliebig in der Datei umhergefahren (scrolling) werden.
- Verändern der Datei entweder in hexadezimaler oder Text-Schreibweise.
- Anspringen eines bestimmten Records oder Zeichens.
- Automatisches Vorwärts und Rückwärtsblättern.
- Suchen einer bestimmten Zeichenkombination.
- On-line Hilfe.

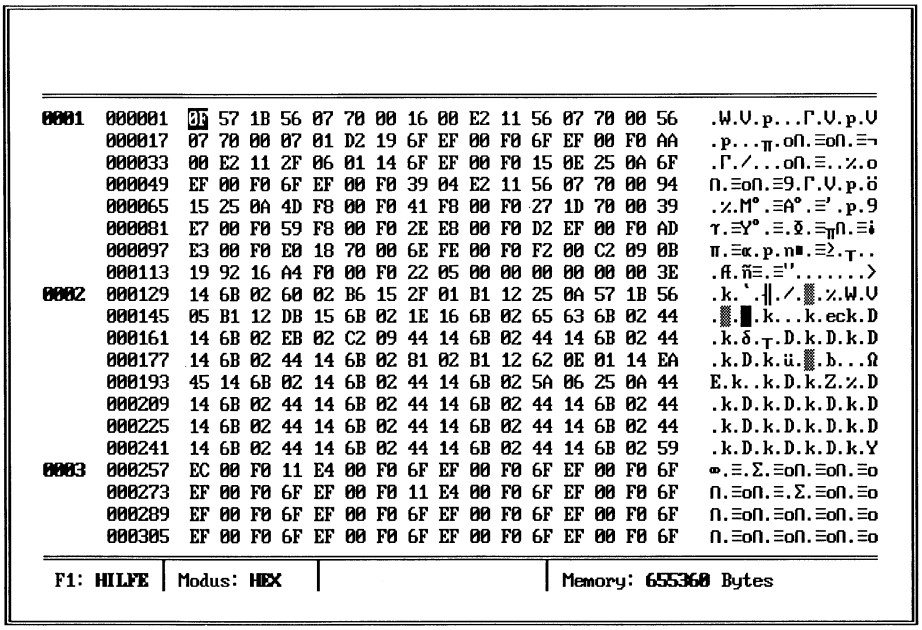


Bild 1

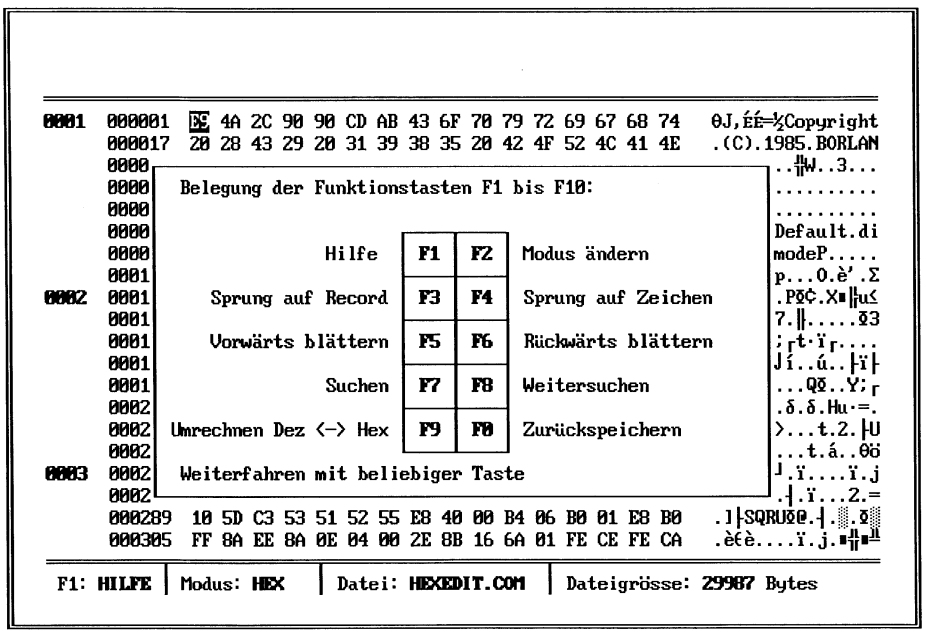


Bild 2

Und so läuft's ab...

Das Programm wird vom Betriebssystem aus gestartet, wobei der Name der zu editierenden Datei als Parameter angegeben werden kann. Wird kein Parameter angegeben, so kann mit dem Programm der Hauptspeicher untersucht werden.

Beispiel:

A>HEXEDIT HEXEDIT.COM

die Datei HEXEDIT.COM wird bearbeitet.

A>HEXEDIT

der Hauptspeicher wird bearbeitet.

Der Dateiname, der angegeben werden muss, darf auch eine Laufwerks- und Pfadangabe enthalten.

Nach dem Start des Programmes wird die entsprechende Datei in einen internen Puffer eingelesen und es erscheint der Editor (Bild 1).

Mit Hilfe der Pfeiltasten kann der Cursor bewegt werden. PgUp und PgDn dienen zum Blättern.

Die Taste Home bewirkt einen Sprung zum Dateianfang. Die Funktionstasten bewirken folgende Aktionen:

F1 Aufruf des Hilfe-Bildschirmes (Bild 2).

F2 Umschalten zwischen hexadezimaler und Textschreibweise.

F3 Sprung auf einen bestimmten Record (ein Record entspricht 128 Zeichen).

F4 Sprung auf ein bestimmtes Zeichen, dessen Position relativ zum Dateianfang angegeben werden muss.

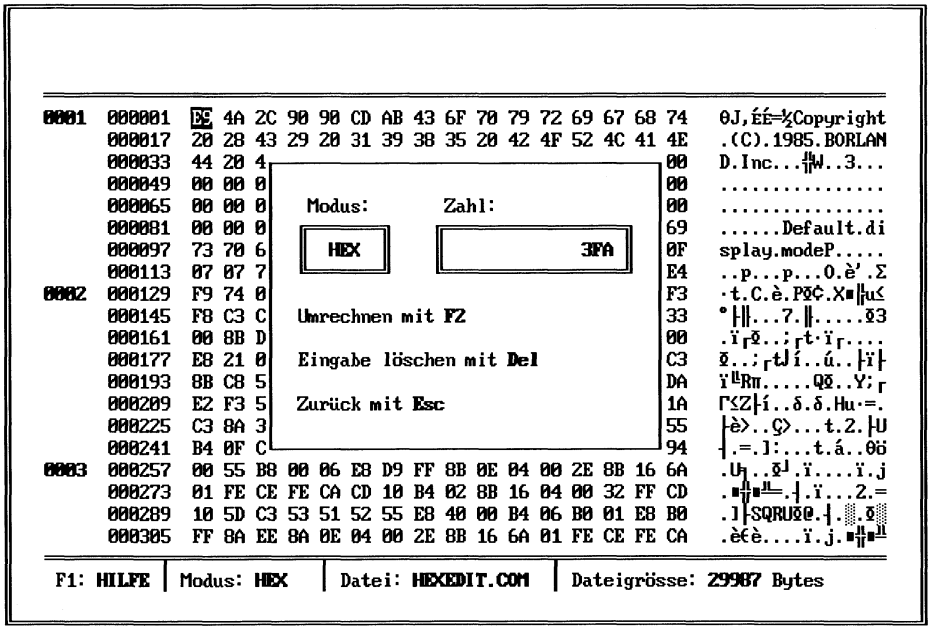


Bild 3

F5 Automatisches Vorwärtsblättern

F6 Automatisches Rückwärtsblättern

F7 Suchen einer Zeichenkombination

F8 Weitersuchen ab der ersten (mit F7) gefundenen Position.

F9 Aufrufen des Umrechners HEX-DEZ (Bild 3).

F10 Verlassen des Programmes und Speichern der durchgeführten Aenderungen.

Die Taste Esc bewirkt ein Verlassen des Editors ohne Speichern der Aenderungen. Die unterste Zeile ist eine Statuszeile, sie zeigt den aktuellen Editiermodus (Hexadezimal oder Text), den Namen der Datei und die Dateigrösse an.

Im Textmodus befindet sich der Cursor im rechten Teil. An der Position des Cursors kann ein beliebiges anderes Zeichen eingegeben werden.

Im Hex-Modus befindet sich der Cursor im linken Teil. An der Position des Cursors kann ein beliebiges anderes Zeichen im Hex-Format (Ziffern 0..9 und Zeichen A..F) eingegeben werden.

Listing- und Diskettenservice

Aus Platzgründen haben wir diesmal auf den Abdruck des umfangreichen Listings verzichtet. Interessenten stellen wir aber gerne das Listing zu (frankiertes und adressiertes C5-Rückantwortcouvert beilegen). Wer sich das Eintippen des Listings jedoch ersparen will, kann beim Autor für Fr. 50.- das Programm (Listing und ablauffähiges Programm) auf Diskette beziehen. Richten Sie Ihre Bestellung mittels einer Leserdienstkontaktkarte (am Schluss des Heftes) an den Verlag, der die Bestellung an den Autor weiterleitet. □

COMPUTER-SPLITTER

Design und Animation mit GEM

(590/fp) Die GEM-Familie wird zügig erweitert. Als jüngster Spross ist GEM Artline dazugestossen. Der Jüngling kann sehr gut zeichnen, gestalten und animieren, aber auch in der grafischen Behandlung von Texten ist er stark. GEM Artline versteht sich bestens mit Ventura und seinen Dateien aber auch mit PageMaker. Sämtliche Peripherie, die mit Ventura läuft, sollte dies auch auf Veranlassung von Artline tun - aber ohne PostScript. □

1 MBit SRAM

(478/eh) Statische RAM's mit einer Speicherkapazität von 1 MB sind zur Zeit noch von niemandem erhältlich. Dies wird sich aber demnächst ändern. Drei Firmen, zwei japanische und eine amerikanische, haben nun 1 MB-SRAM's angekündigt. Toshiba und Hitachi bringen solche Speicherbausteine, die mit einer neuen 0.8 µm-Technik produziert werden, Inova aus USA beschränkt sich auf die bereits bewährte 1,2 um-Technik. Inova wird wahrscheinlich ihre Speicherchips als erste anbieten können, ist aber auf unüblich grosse Gehäuseabmessungen angewiesen. Der Chip wird in einem 32-poligen Dual-In-Line-Gehäuse geliefert werden und zwischen 150 und 300 Dollar das Stück kosten. Dies bei einer Abnahme von 100 Stück. □

Neue Bildschirme von HP: VGA

(597/fp) Hewlett-Packard fährt nicht nur mit auf dem VGA-Zug, sie setzt die Lokomotive davor. Ein Kontroller-Bildschirm-Duo für VGA erlaubt die Wiedergabe von 800x600 Punkten und gleichzeitigen 256 Farben. Das VGA-Duo ist kompatibel mit sämtlichen «alten» Grafikmodi und verwaltet die Grafiken in einem eigenen Video-RAM von 256 KBytes oder 512 KBytes. Die monochromen Bildschirme zum VGA-Adapter von HP geben 64 Graustufen wieder. □

Peinlich

(595/fp) Entgegen aller anderslautenden Gerüchten ist nicht alles Gold, was Amerika. So z.B. in der Werbung. Da jettet ein hochmoderner Kampfbomber über die Wolken, elegant, schnell, mit Schriftzug ORACLE. Der Pilot wirft einen deutlich erkennbaren Blick auf einen darunter lodernd abstürzenden, einmotorigen Dreidecker. Schriftzug dBase. Gesehen im «PC Magazine». Peinlich. Im Zusammenhang mit aktuellen Ereignissen und dem dortigen Versagen amerikanischer Rettungsdienste: Geschmacklos. Auch in USA! □

Inside OS/2

(596/fp) Einer, der es wissen muss, hat ein Buch über OS/2 geschrieben: Inside OS/2, von Gordon Letwin, dem Projektleiter des OS/2-Teams (Microsoft Press, ISDN 1-55615-117-9). Das Buch erörtert die Geschichte, die Technik, die Funktionen und die Zukunft des Betriebssystems. Laut Rezensionen in der US-Fachpresse soll das Buch die komplexe Materie wissenschaftlich korrekt und dennoch allgemein verständlich darlegen. □

Mikroprozessor-Architekturen

Dieser Artikel beschreibt und bewertet die Methoden, die in modernen Mikroprozessoren zur Unterstützung der Speicherverwaltung eingesetzt werden. Erläutert werden ausserdem die Sicherheitsvorkehrungen und deren Verwirklichung. Vorgestellt werden die gängigsten 16- und 32-Bit-Prozessoren und wie sie die Speicherverwaltung realisieren. Der Begriff des hierarchischen Speicherns und grundlegende Anforderungen an einen Prozessor werden ebenso erläutert, wie die Adressen-Berechnungsmethoden: das Paging, die Segmentierung und die Kombination aus beiden. Nicht vergessen wird der Assoziativ-Cache sowie die verschiedenen Mapping-Schemata. Zuletzt wird noch auf die Unterstützung von virtuellem Speicherplatz eingegangen und die Möglichkeiten der Adressenfehler-Behandlung.

Oliver Rosenbaum

Die neue Generation von 16- und 32-Bit-Mikroprozessoren ist ausgereift und eignet sich im besonderen Masse zum Multiuser- und Multitasking-Betrieb. Gerade hier ist ein gesteigerter Bedarf an hardwareseitiger Unterstützung der Speicherorganisation. Die fortschreitende Disketten-Technologie besorgt die Deckung des wachsenden Bedarfes an sekundärem Speicherplatz und die Mikroprozessoren können einen grösseren Primär-Adressraum ansteuern (reeller Speicher), welcher seinerseits einen ebenfalls grösseren virtuellen Adressraum verwalten kann. Diese Prozessoren stellen zudem elegante Verwaltungsmechanismen bereit für den virtuellen Speicher.

In der Familie der 16-Bit-Prozessoren werden allerdings verhältnismässig unzulängliche Techniken zur Lösung von Speicherproblemen eingesetzt, die nicht effizient sind. Die 32-Bit-Prozessoren hingegen besitzen elegantere Lösungen in ihrer Architektur zur Unterstützung der Verwaltung von virtuellem Speicherraum.

Anforderung an die Speicherverwaltung

In der Regel bedient sich ein Mikroprozessor-System, welches die Speicherverwaltung unterstützt, der hierarchischen Speicher-Organisation (Bild 1).

Der Gesamtspeicher besteht also aus drei sehr unterschiedlichen Medien, welche die Niveaustufen bilden. Die Medien unterscheiden sich in der jeweiligen Kapazität, ihrer Zugriffsgeschwindigkeit und in den Kosten.

Auf der obersten Niveaustufe finden wir den High-Speed-Cachespeicher, welcher der teuerste (bezogen auf seine Kapazität) ist. Der Primärspeicher (reeller Speicher) ist zwar langsamer, dafür aber wesentlich billiger als der Cache. Er besitzt eine grosse Kapazität und bildet die mittlere Niveaustufe. Auf dem untersten Level wird der «Backstore» angesiedelt. Auf diesen werden Programme und Daten abgelegt, welche wegen ihres Umfanges nicht auf den beiden anderen Niveaustufen Platz finden.

Wenn ein Programm gestartet wird, werden der Programmcode und die Daten in den Primärspeicher und den Cachespeicher geladen. Letzterer enthält immer die kurzfristig benötigten Daten und aktuellen Programmcodes.

In der hierarchischen Speicherstruktur können die zugrundeliegenden Forderungen an die Speicherverwaltung wie folgt aufgezählt werden:

- Möglichkeiten zur Adressenumformung und Unterstützung der dynamischen Speicherzuteilung
- Unterstützung von virtuellem Speicherraum
- Schutz der Daten und Sicherheit der Speicheraufteilung

Der virtuelle Speichermechanismus erlaubt die Ausführung von Programmen, wenn nur wenige Blocks des Programmes im Hauptspeicher vorliegen, während der Rest des Programmes sich noch auf dem Sekundärspeicher (z.B. Diskette) befindet.

Es gibt zwei grundlegende Strategien der Mikroprozessor-Architektur zur Realisierung der Speicherverwaltung:

1) Die MMU (Memory Management Unit) ist auf dem CPU-Chip integriert.
2) Die MMU befindet sich nicht auf dem CPU-Chip, sondern ist auf einem separatem Chip untergebracht.

Beide Strategien sowie eine Liste von Mikroprozessor-Systemen, welche entweder nach Strategie 1 oder 2 arbeiten, sind in Bild 2 ersichtlich.

Die Hauptvorteile für die Memoryverwaltung auf dem CPU-Chip (Central Processing Unit) sind folgende:

- Verbesserung der Zugriffsgeschwindigkeit durch die direkte Verdrahtung
- Ein Maximum an Portabilität von Betriebssystemen und Anwenderprogrammen
- Bauteilverminderung auf dem Mainboard des Rechners

Andererseits verlangt die Integration der MMU auf dem CPU-Chip eine Anzahl von Transistorplätzen, welche für andere Funktionen verwendet werden könnten. Der Motorola MC68020, der eine Off-Chip-Memoryverwaltung besitzt (separater MMU-Chip) nutzt den dadurch einge-

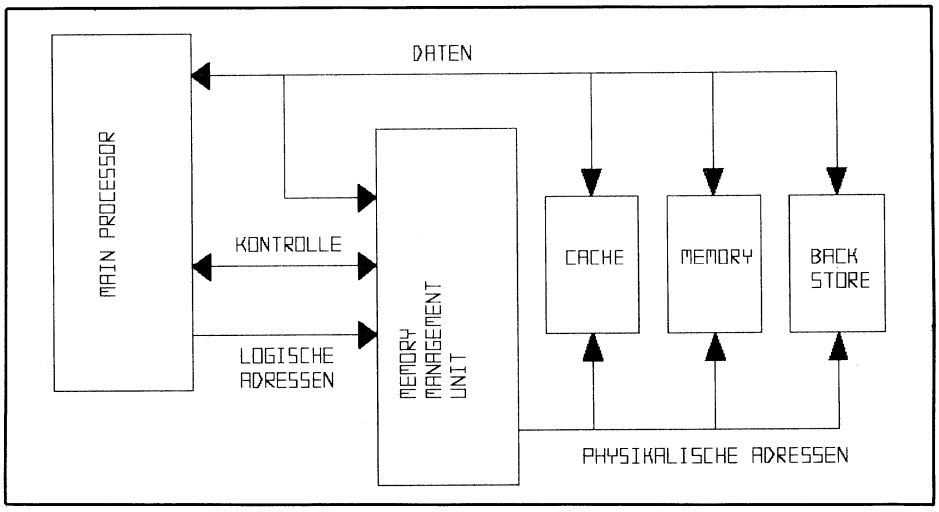


Bild 1 zeigt eine Mikroprozessor-System-Architektur mit hierarchisch organisiertem Speicher in drei Niveaustufen. Niveau 1 wird vom High-Speed-Cache gebildet, auf Niveau 2 befindet sich der Hauptspeicher des Systems und der Sekundärspeicher (Diskette, Festplatte o.ä.) bildet die 3. Niveaustufe.

sparten Platz auf dem Chip für die Realisierung eines Instruktions-Cache.

Ebenso wichtig wie die Wahl einer On-Chip- oder Off-Chip-MMU ist die Methode der Speicherorganisation. Grundsätzlich ist zwischen zwei verschiedenen Organisations-Schematas zu unterscheiden: linear oder segmentiert.

Die Wahl der Speicherverwaltungs-Hardware (MMU) und die Wahl der Methode für die Speicherorganisation sind immer im Zusammenhang zu sehen. In der Regel haben Prozessoren mit einer On-Chip-MMU eine Segmentierungs-Speicherverwaltung während bei Systemen mit getrennter CPU und MMU die lineare Adressierung vorzuziehen ist.

Das lineare Adressierungsschema beginnt immer bei Adresse 0 und fährt linear fort. Der Speicherraum kann später zum Beispiel softwareseitig strukturiert werden.

Im Segmentierungs-Adressenschema werden die Programme nicht als eine Reihenfolge von Instruktionen geschrieben, sondern als Module von Codes, Daten, Stacks usw. Der logische Adressenraum ist aufgeteilt in verschiedene lineare Adressenzwischenräume, jeder von veränderlicher Länge, welche vom Programmierer oder vom Compiler definiert werden kann.

Eine logische Adresse wird aus einer Kombination der Segmentnummer, welche einen Zeiger zum Block im Speicher darstellt und einem Code innerhalb des Segments berechnet.

Das Segmentierungs-Adressenschema unterstützt den Anwender bei der Speicherverwaltung. Der Anwender sieht das Memory als eine Sammlung von Segmenten, die von variabler Grösse sein können, gerade so wie sie benötigt werden für Programme, Unterprogramm-Module, Datenstrukturen usw.

Bild 3 zeigt das in verschiedenen Mikroprozessoren gebrauchten Speicher-Adressierungsschema. Es ist hier zu sehen, dass sowohl Zilog als auch Intel segementiert- und linearadressierende 32-Bit-Prozessoren anbieten (Z80000 und Intel 80386).

Im allgemeinen ist das lineare Adressierungsschema besser für die Manipulation grosser Datenstrukturen, während das Segmentierungs-Adressenschema die Programmierung erleichtert und dem Programmierer erlaubt, seine Software besser zu strukturieren. Die Segmentierung vereinfacht zudem den Schutz und die Verschiebung der Objekte im Speicher.

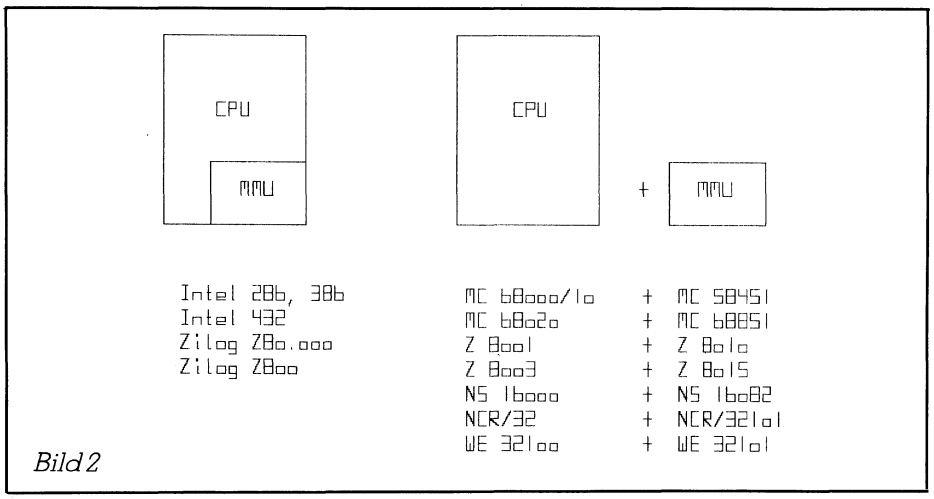


Bild 2

Als Beispiel für Adressensegmentierung sei hier der Intel 8086 genannt. Er besitzt vier 16-Bit Segmentregister, welche zu vier Objekten im Speicher zeigen: Code, Stack, Daten und einem extra Segment (wechselnde Daten).

Adressen-Umwandlungsmechanismus

Ungeachtet der Methode der Speicherorganisation muss der Prozessor einen Adressen-Umwandlungsmechanismus besitzen, um den virtuellen Speicherraum zu verwalten. Dieser Mechanismus liefert auch Methoden um Objekte im Speicher zu schützen (hierauf wird später noch eingegangen). Die Adressenumwandlung stellt die logischen Adressen als physikalische Speicheradressen dar (Mapping). Der Adressen-Umwandlungsmechanismus unterteilt den Speicher in Blocks und führt dann das Mapping von Blocks mit logischen Adressen in Blocks mit physikalischen Adressen durch.

Hierdurch ist es möglich, Programme, Daten, Stacks und andere Strukturen im Primärspeicher zu lokalisieren. Dieser Mechanismus bildet auch die Basis für den virtuellen Speicher, wo der logische Adressenraum grösser sein kann als der physikalische. Natürlich sind da noch andere wichtige Fähigkeiten, die ein Prozessor aufweisen muss zur virtuellen Speicherverwaltung - sie werden noch besprochen.

Die drei elementaren Umwandlungsmechanismen sind das Paging, die Segmentierung sowie eine Kombination aus Paging und Segmentierung. Bei Systemen, die mit dem Paging arbeiten, wird der Hauptspeicher (Primärspeicher) in Blocks von konstanter Grösse aufgeteilt, während bei der Segmentierungsmethode die Blocks verschiedene Grössen haben können; man spricht hier auch von Segmenten.

In Segmentierungssystemen wird ein Programm beim Laden nicht auf Positionen im logischen Adressenbereich gebunden. Beim Paging wird die Unterteilung des Programmes in logische Adressbereiche vom Programm vorgenommen, also nicht vom Programmierer und auch nicht vom Compiler.

Die einzelnen Pages sind normalerweise relativ klein im Verhältnis zur Gesamtspeicherkapazität. Die typische Pagegrösse liegt zwischen 256 und 2'048 Bytes, während Segmente 64 KBytes oder noch grösser sein können.

In Multiprogramming- und Time-Sharing-Systemen nutzen viele Anwender die Programme wie z.B. Editor, Compiler, Utilities, Programm-Bibliotheken usw. Sowohl das Paging- als auch das Segmentierungssystem liefern hierzu Teilungsmechanismen. Dies wird realisiert mit Hilfe von Einträgen in Mappingtabellen für die unterschiedlichen Prozesse, welche zum gleichen Seitenbereich zeigen (siehe hierzu Bild 6).

Beim kombinierten Paging-/Segmentierungsschema werden die Merkmale beider Methoden vermischt. Der Segmentierungsteil verwaltet den virtuellen Speicher, indem die Programme in Segmente aufgeteilt werden, während der Pagingteil des Systems den physikalischen Speicher verwaltet durch Aufteilung dessen in Pages. Jedes Segment besteht seinerseits wieder aus einer Anzahl von Pages.

Die Auswahl des Adressen-Umwandlungsmechanismuses ist entscheidend für die Speicherverwaltungstechniken, die vom Betriebssystem verwirklicht werden müssen. Der Pagingmechanismus ist z.B. sehr gut geeignet für die Plazierung und Verschiebung von Pages, weil alle Pages

gleich gross sind. Der Segmentierungsmechanismus benötigt komplizierte Algorithmen für die Plazierung der Segmente im verfügbaren Speicherraum. Bei der Segmentierung muss ein Segment erst vollständig im Primärspeicher plaziert sein, um ausgeführt werden zu können, da die kleinste Einheit das Segment selbst ist. Der verfügbare Speicherplatz muss dann fragmentiert werden in viele kleine Teilbereiche. Durch die bei der Fragmentierung auftretenden Probleme, sind die Pagingsysteme leistungfähiger in Bezug auf die Ausnutzung des Speicherplatzes. Durch die schon erwähnte immer gleiche Grösse der Pages, können sie problemlos getauscht werden und hinterlassen keinen unbrauchbaren, fragmentierten Speicherplatz; jedoch ist das Problem des durch die Fragmentierung verlorenen Speicherplatzes durch die grossen verfügbaren Speicherkapazitäten heute nicht mehr so gravierend.

Paging hat einen weiteren Vorteil: nicht alle Pages eines Programmes müssen permanent vorhanden sein, sondern nur diejenigen, welche momentan benötigt werden. Dieser Umstand reduziert die benötigte Zeit zum Tauschen der Pages im Speicher erheblich, allerdings muss nun öfter getauscht werden. Beim sogenannten «anticipatory Paging» (vorwegnehmenden Paging) wird über das Betriebssystem versucht, diejenigen Pages vorauszubestimmen, welche in unmittelbarer Zukunft gebraucht werden. Diese werden dann schon für den Prozess bereitgestellt. Im Zusammenhang mit einem schnellen anticipatory-Algorithmus werden die Ablaufprozesse dadurch schneller, weil die Pages schon im Hauptspeicher liegen, noch bevor sie gebraucht werden. Vieles spricht also für das Pagingsystem, vorallem das letztgenannte. In der Tat unterstützen alle gängigen ATs 32-Bit-Prozessoren sowie verschiedene 16-Bit'ler vollständig diese Technik, welche bereits jetzt für künftige Mikroprozessorentwicklungen zu einem Standard geworden ist.

Zur Verwirklichung der Adressenumformungen für die erwähnten Mechanismen gibt es zwei Techniken, die Adressenumformung mittels Tabellen sowie die registerorientierte Adressenumformung.

Tabellenumformungstechnik

Diese Art der Adressenumformung basiert auf Tabellen, die im Primärspeicher angelegt werden: für Pagingsysteme sind dies die PMT (Page-Map-Tabellen) und für segmentierende Systeme die SMT (Segment-Map-Tabellen).

Die Tabelleneinträge enthalten Informationen, die zur Umformung von logischen in physikalische Adressen benötigt werden sowie Daten für Zugriffsschutzzwecke und Daten zur Unterstützung der Algorithmen für die Plazierung und Verschiebung.

Bild 7 zeigt den Adressen-Umformungsmechanismus des Intel 286. Der Segment-Sucher zeigt zum Segment-Descriptor in der Segment-Tabelle. Die physikalische Adresse ist gebildet als Verkettung von der Primärspeicher-Adresse des Segmentes (im Segment-Descriptor) und dem Verschiebungsanteil, der in der logischen Adresse enthalten ist.

Diese Methode wird als «direktes Mapping» bezeichnet. Das direkte Mapping benötigt zur Umformung einer logischen zu einer physikalischen Adresse eine zusätzliche Operation, um die Basisadresse des Segmentes (oder Page) zu erhalten. Daher kann das direkte Mapping die Ausführungsgeschwindigkeit des Systems negativ beeinflussen. In modernen Mikroprozessor-Architekturen werden verschiedene Lösungen verwendet, um dieses Problem zu überwinden. Diese Lösungen werden noch erläutert.

Der Intel 286-Prozessor hat vier Segmentregister, die erweitert sind mit vier korrespondierenden 48-Bit Segment-Descriptor Cacheregistern.

Die Segmentregister werden durch das Programm geladen, während die Cacheregister durch die CPU geladen werden.

Das Laden des Cache wird in vier Schritten durchgeführt:

- Das Programm plaziert einen Zeiger im korrespondierenden Segmentregister.
- Der Prozessor addiert aus der Descriptortabelle den Zeigerindex zur Basisadresse, um einen Descriptor auszuwählen.
- Nachdem der Prozessor die Segment-Zugriffsrechte geprüft hat, kopiert er den Descriptor zum jeweiligen Segmentregister-Cache.
- Der Prozessor gebraucht die Information des Descriptors, um den Segment-Typ und die Grenzen des Segmentes zu prüfen, als auch um die effektive Adresse des Segments zu bilden.

Die beschriebene Technik, welche auf einem offenen Cacheregister basiert, beschleunigt das direkte Mapping, ist allein aber noch nicht leistungsfähig genug, da der Cache immer dann geladen werden muss, wenn die Kontrolle von einem Segment gleichen Typs zum anderen übertragen wird.

Eine sehr viel elegantere Lösung basiert auf einem speziellen assoziativen Cache mit 32 bis 64 Stellen, welcher die gebräuchlichsten Umformungswerte enthält.

Mit dieser Verbesserung kann der Umformungsprozess folgendermassen ausgeführt werden:

Prozessor	Adressierung linear	Adressierung segmentiert
Intel		
8086, 80286, 432		X
80286	X	X
Motorola		
68000, 68010, 68020	X	
National		
16032, 32032	X	
Zilog		
Z8000		X
Z80.000	X	X
AT&T		
WE32100	X	
NCR		
NCR/32	X	

Bild 3 zeigt die in verschiedenen Mikroprozessoren gebrauchten Speicher-Adressierungs-Schema

1) Die von der CPU empfangene virtuelle Adresse wird im Cache gesucht. Wenn die Adresse mit keiner der Einträgen im Cache übereinstimmt, wird die korrespondierende physikalische Adresse, welche sich im Cache befindet, von der CPU benutzt, um direkt auf den Primärspeicher zuzugreifen.
2) Wenn die virtuelle Adresse auf keinen der Cacheeinträgen passt, aber die Page oder das Segment im Hauptspeicher ist, dann wird die physikalische Adresse geladen, so wie es in den Umformungstabellen im Hauptspeicher festgelegt ist und daraufhin im Cache abgelegt. Die CPU kann nun auf den verlangten physikalischen Speicherraum zugreifen.
3) Ist die Page oder das Segment nicht im Hauptspeicher, so muss zunächst auf den Sekundärspeicher zugegriffen werden und umgespeichert werden. Die Umformungstabellen müssen geändert werden und die physikalische Adresse muss von der Umformungstabelle im Cache geholt werden.

Der assoziative Cachespeicher ist gewöhnlich wie ein TLB (Translation-Lookaside-Buffer) organisiert. Wenn eine logische Adresse umgeformt werden soll, wird gleichzeitig jeder Eintrag im TLB untersucht.

Eine grosse Anzahl von Studien mit Simulationsmodellen hat gezeigt, dass ein kleiner assoziativer Cache die Systemoperationen wesentlich beschleunigt, weil das Trefferverhältnis, die gebrauchte Information im Cache zu finden, bis zu 98 % erreicht. Viele Prozessoren haben dieses Adressenumformungsschema mit Hilfe der TLB-Methode verwendet. Hierzu gehören u.a.: Intel 80386, Zilog 80000 und die National-Familie NS16000 mit der NS16082 MMU.

Der TLB enthält in der Regel die Information, um die sechzehn zuletzt gebrauchten Pages umzusetzen. Für jede Speicherzuweisung wird die logische 22-Bit breite Adresse verglichen mit den im TLB gespeicherten Adressen-Zusätzen. Wenn der passende gefunden wird, kann der korrespondierende, physikalische Pageadressenbereich kombiniert werden mit dem 10-Bit breiten Zusatz aus der logischen Adresse und eine 32-Bit breite Adresse gebildet werden.

Wenn der TLB keine passende Information enthält, weist die CPU automatisch an, die Umformungstabellen im Hauptspeicher zu laden. Charakteristisch dabei ist, dass der gerade zuvor gebrauchte Algorithmus (LRU last-recently-used) für die Entscheidung verantwortlich ist, welche TLB-Einträge ersetzt werden wollen. Das Flussdiagramm in Bild 8 zeigt drei verschieden Kontrollpfade:

1) Wenn der Page-Descriptor im assoziativen Cache gefunden wurde.
2) Wenn der Page-Descriptor im Cache gefunden wurde, aber die Page im Hauptspeicher ist.
3) Wenn der Page-Descriptor nicht gefunden wurde im assoziativen Cache und die Page nicht im Hauptspeicher ist, wird die Adressenfehler-Routine aktiviert.

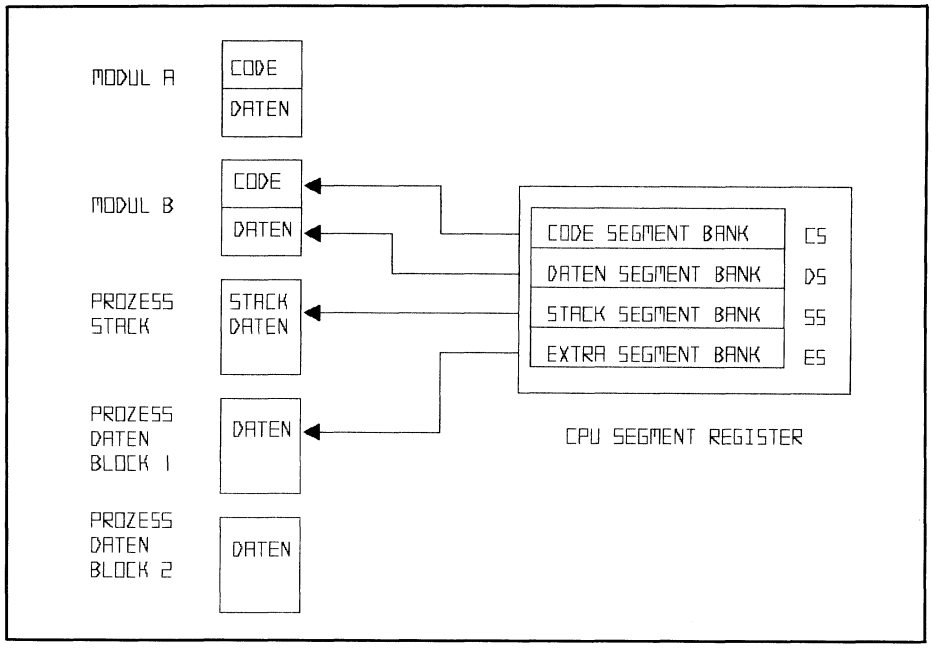


Bild 4 zeigt die Zuordnung der vier Register zum Speicherinhalt. Ein Umschalten von Modul B zu Modul A verlangt ein Zurückladen der einzelnen Segment-Register

Register-Umwandlungstechnik

Die Register-Umwandlungstechnik basiert auf individuell adressierbaren Hardwareregistern, welche die Umformungsparameter enthalten. Diese Technik ist im Motorola 68000 mit seiner 58451 MMU und im Zilog 8000 bzw. 800 verwirklicht und kann in zwei Kategorien klassifiziert werden: adressenzugängliche und inhaltadressierbare Register.

Im System, wo die Umformungsregister der Adresse zugänglich sind, weist das logische Adressenfeld auf das Register, welches die physikalische Basisadresse enthält.

Diese Technik findet sich im Z800 mit einer On-Chip-MMU. Die virtuelle Adresse besteht aus einem 4-Bit Register-Pointer und einem 12-Bit Offset.

Die beschriebene Technik partitioniert den 64 KByte grossen logischen Adressenraum. Dies verlangt das Einladen neuer Werte ins Umformungsregister nach jedem ausgeführten Task.

Die Register-Umformungstechnik ist unpraktisch bei grossen Systemen, da für jedes logische Segment (oder Page) ein Segmentregister benötigt wird. Die MC 68451 MMU z.B. kann separat 65'536x256 KByte Segmente im physikalischen Adressraum festlegen (im extremsten Fall), welche gleichviele adressenzugängliche Register verlangen würden für die Umformung der Adressen.

Da ist die zweite genannte Methode, die der inhaltadressierbaren Register, sehr viel besser geeignet für grosse Systeme. Sie findet man z.B. in der MC 58451 MMU und der Z8015 MMU.

Dynamische Speicherzuweisung

Zusätzlich zum Adressen-Umwandlungsmechanismus unterstützt die MC 58451 MMU die dynamische Speicherzuweisung. Mit diesem Mechanismus ist es möglich, während dessen Ausführung, Speicher einem Prozess zuzuteilen.

Im Motorola-Prozessor MC 58451 verwirklicht das sogenannte Binary-Buddy-System (ein Algorithmus) die dynamische Speicherzuweisung. Hier teilt der Algorithmus den physikalischen Speicher vollständig auf in viele Pufferspeicher von der Grösse 256 Bytes bis 256 KBytes. Der Algo-

rithmus behält diese Pufferspeicher bei und führt Pufferlisten für alle Serien gleichgrosser Pufferspeicher und ebenfalls Descriptoren für jeden einzelnen Pufferbereich. Wenn eine Anfrage nach Speicherplatz vorliegt, sucht der Algorithmus in der Liste nach verfügbarem Platz, um den am besten geeigneten Pufferspeicher zu finden. Wird die ideale Grösse nicht gefunden, weicht das System auf den nächst grösseren Pufferspeicher aus.

Single-Level- und Multi-Level-Mapping

Herkömmliche Mappingtechniken bestehen meist aus nur einem Mappingniveau: z.B. in den gängigen 16-Bit-Prozessoren: Intel 286, Zilog 8010 und Zilog 8015 MMU. Eine andere Möglichkeit wird beim NCR/32-Prozessor verwirklicht: es wird ein spezieller Adressenumformungs-Chip verwendet, der ATC (Adress-Translation-Chip). Dieser übernimmt die Adressenumformung auf der Grundlage eines Pagingsystems, jedoch ebenfalls mit einschichtigem Mapping. Dieser Chip enthält sechzehn Assoziativspeicher für gerade zuvor gebrauchte Pages.

Die Z8010 MMU, welche mit dem Prozessor Z8001 benutzt wird, verwendet eine einschichtige Segmentierung mit insgesamt 64 inhaltsadressierbaren Descriptorregistern. Hier werden die logischen Adressensegmente transferiert, während die Z8015 MMU in Pages umformt. Ein einschichtiges Mappingschema kommt zum Einsatz und 64 Page-Descriptorregister, welche inhaltsadressierbar sind.

Im Gegensatz hierzu unterstützt der WE32100 32-Bit-Prozessor mit seiner 32101 MMU je nach Bedarf Paging oder Segmentierung, der Anwender hat hier die Auswahl. Die MMU enthält einen On-Chip-Chachespeicher. Dieser besteht aus einem 32-stelligen Segment-Descriptor-Cache und einem Page-Descriptor-Cache mit 64 Stellen. Hier werden die gerade gebrauchten Segment- bzw. Page-Descriptoren vorgehalten.

Fast alle anderen 32-Bit-Prozessoren benutzen das Multilevel-Mapping-Schema, welches neue Gesichtspunkte in die Memoryverwaltung bringt. Die Vorzüge des Multilevel, gegenüber dem Single-Level-Mapping, sind:

- Die Schutzmechanismen können einfacher realisiert werden.
- Grössere Adressbereiche können verwaltet werden.
- Segment- und Page-Sharing ist möglich.

Intel's 432-Prozessor bedient sich z.B. einem 2-Level-Mapping und erreicht damit hervorragende Schutzmechanismen. Das Segment-Pointer-Register zeigt zum Zugriffssegment, in dem die Zugriffsrechte definiert sind. Der Zugriffs-Descriptor enthält einen Pointer zum Anfang des ausgewählten Segments im Hauptspeicher.

Weil die Zugriffsrechte unabhängig von den Segment-Descriptoren gespeichert sind, können sich mehrere Module das gleiche Segment teilen und zwar jedes von ihnen mit unterschiedlichen Zugriffsrechten. Das 2-Level-Mapping kann ausserdem die Anzahl der zugänglichen Segmente begrenzen.

Beim Single-Mapping hingegen (wie es der Intel 286 verwendet) kann jedes Programm jedes Segment durch einfachen Zugriff über die Segment-Tabelle adressieren.

Der Intel 386 bietet zwei Optionen, welche vom Anwender wählbar sind: Ein dem Intel 286 gleiches Segmentierungssystem oder ein Pagingsystem. Beim Pagingsystem wird das 2-Level-Mapping angewendet zusammen mit dem bereits erwähnten TLB (Translation-Lookaside-Buffer), der wie ein Chachespeicher arbeitet.

Die lineare virtuelle Adresse besteht aus drei Feldern (Directory, Tabelle, Offset). Die Adressenumformung wird nach folgenden Schritten durchgeführt:

1) Die Adresse wird vom TBL gesucht. Ist die Adresse gefunden, wird die Umformung direkt im TLB durchgeführt und direkt auf den Hauptspeicher zugegriffen.
2) Wenn die Adresse im TBL nicht gefunden wird, wird ein Verlustsignal erzeugt und die Umformung wird durch die CPU im 2-Level-Mapping ausgeführt.

Das 2-Level On-Chip Mapping ermöglicht eine schnelle Adressenumformung und Page-Tabellen können besser bearbeitet werden. Eine ähnliche Technik wird auf der NS 16082 MMU angewendet. Der gesamte physikalische Adressenbereich wird in 32'768 feste Pages aufgeteilt mit jeweils 512 KBytes.

Die virtuelle Adresse besteht aus 24 Bits, welche in drei Felder aufgeteilt sind: Index-1 und Index-2 des Page-Pointers und dem Offset. Ausserdem gibt es hier noch zwei CPU-Register für den Anwendermodus und für den Supervisormodus. Diese enthalten die korrespondierende Ablaufadresse der Page-Tabellen im Hauptspeicher.

Index-1 (8 Bits) vom Page-Pointer dient zum Festlegen auf einen der 256 Einträge in der Page-Tabelle. Der Inhalt der Page-Tabelle seinerseits verweist auf wiederum 256 Pointer-Tabellen, die jeweils 128 Einträge besitzen. Der Pointer dieser Tabelle wird kombiniert mit dem Index-2 (7 Bits) vom Page-Pointer.

Der so ausgewählte Eintrag enthält die tatsächliche Page-Nummer im Hauptspeicher. Das Offset-Feld dient zur Festlegung der Daten innerhalb der Page.

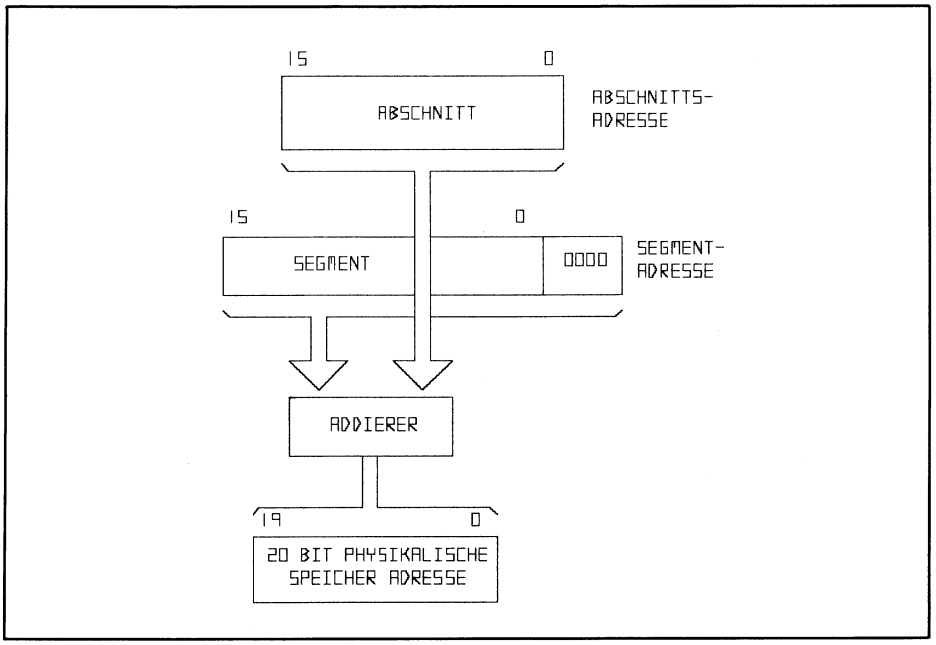


Bild 5 zeigt den Adressen-Berechnungsmechanismus, der im Intel 8086 eine 20-Bit breite physikalische Adresse hervorbringt

Die NS 16082 MMU enthält einen assoziativen Cache um die 32 zuletzt gebrauchten Page-Adressen vorzuhalten. Der Z80000-Prozessor arbeitet mit einem 3-Level-Schema basierend auf drei Umformungstabellen im Hauptspeicher. Dieser 3-Level-Adressenumformung unterteilt die 4 GByte des logischen (linearen) Adressenraumes in Pages von 1 KByte Grösse.

Die Rolle des Betriebssystems ist hierbei die Umformungstabelle im Hauptspeicher zu erzeugen und die vier On-Chip-Tabellen-Descriptorregister zu setzen (System-Instruktion, System-Daten, normale Instruktion und normale Daten). Anschliessend weist die CPU automatisch den Level-1 und den Level-2 sowie die Page-Tabellen zu, um die Adressenumformung durchführen zu können.

Der Prozessor enthält ausserdem einen assoziativen Speicher für das TBL, in dem die 16 zuletzt erwähnten Pages gespeichert werden. Wird eine logische Adresse nicht gefunden, wird der TLB automatisch von der CPU über die Umformungstabelle nachgeladen. Bild 9 fasst die Merkmale einiger 16- und 32-Bit Prozessoren bei der Adressumformung zusammen.

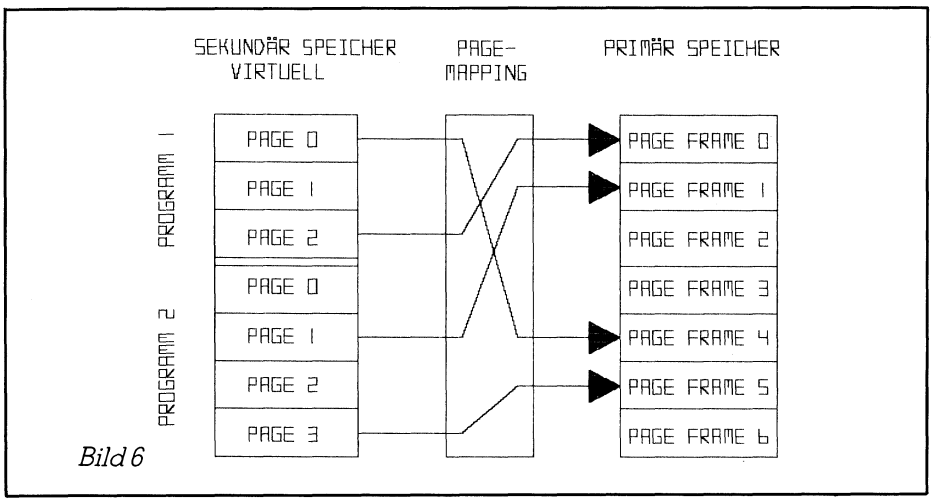


Bild 6

Die Technik der virtuellen Adressierung

Ein virtuelles Speichersystem erlaubt dem Anwender sehr grosse Programme auszuführen, die den physikalischen, tatsächlichen Speicherraum sprengen würden. Der Mikroprozessor muss hierzu die Fähigkeit besitzen, auf Pages oder Segmente im Sekundärmemory zugreifen zu können. Die virtuelle Speicherverwaltung lädt das gesuchte Segment (oder Page) in den Hauptspeicher, wenn es benötigt wird.

Ferner muss der Mikroprozessor ausser der schon beschriebenen Adressenumformung folgende Möglichkeiten unterstützen.

- Das System muss einen Page- oder Segmentfehler erkennen können (wenn die gesuchte Page nicht im Hauptspeicher vorliegt). Die MMU muss dann den Prozessor informieren, so dass die fehlende Page oder das fehlende Segment vom Sekundärspeicher geholt werden kann, und durch eventuell aktuelle Pages ersetzt werden können.
- Das System muss die Möglichkeit besitzen, eine gerade laufende Instruktion abbrechen zu können.
- Der Zustand der Maschine muss festgehalten werden, um nach der Beseitigung des Fehlers (fehlende Page z.B.) fortfahren zu können.
- Die Fehlerbehandlungsroutine des Betriebssystems muss aufgerufen werden, um die verlangte Page oder das Segment vom Sekundär- zum Primärspeicher zu transferieren.
- Die notwendige Information für das Betriebssystem muss bereitgestellt werden, um die Page- oder Segment-Plazierungs- oder Verschiebungsalgorithmen zu untersützen.
- Der zuvor zwischengespeicherte Zustand muss zurückgespeichert und der normale Prozess wieder aufgenommen werden (Instruktion-Neustart).

Trotz ihrer grossen Unterschiede liefern alle gängigen Mikroprozessoren Instruktionsabbruch und Neustartmöglichkeiten. Einige Beispiele werden noch gezeigt.

Die Fehlererkennung kann nur dann intern durchgeführt werden, wenn sich die MMU auf dem CPU-Chip befindet, ansonsten wird sie extern durchgeführt (bei Off-Chip-MMU).

Ein «intern erkannter Fehler» liegt z.B. dann vor, wenn auf nicht im Hauptspeicher vorhandene Daten oder Programmcodes zugegriffen werden soll. Die Adressfehlerroutine kommt daraufhin zum Einsatz. Wenn die Off-Chip-MMU eine Fehlersituation entdeckt, muss sie ein Signal zur CPU senden, welche ihrerseits dann die Routine startet («extern erkannter Fehler»).

Die CPU muss in jedem Fall bei Erkennung eines Zugriffsfehlers die Informationen über den internen Zustand «retten».

Gewöhnlich geschieht dies über den Stack. Normalerweise muss der Programmzähler (Startadresse der Instruktion), das Statusregister, die fehlerhafte Adresse, der Zugriffstyp, die aktuellen internen Register und verschiedenste interne Statusinformationen erhalten bleiben.

Die MMU muss auch dem Betriebssystem die Informationen über die Zugriffsaktivitäten bereitstellen (Plazierungs- und Verschiebe-Algorithmen). Gewöhnlich wird diese Information in

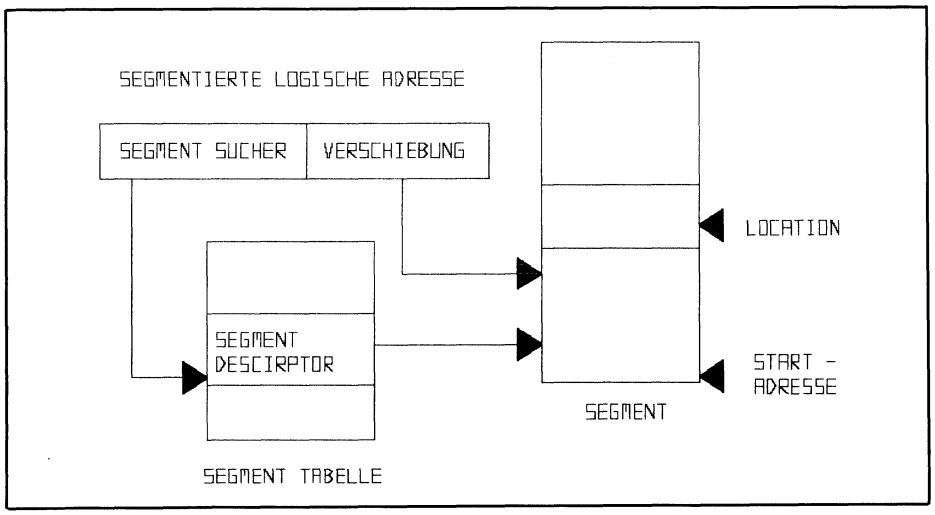


Bild 7 zeigt den Adressen-Berechnungsmechanismus des Intel 286

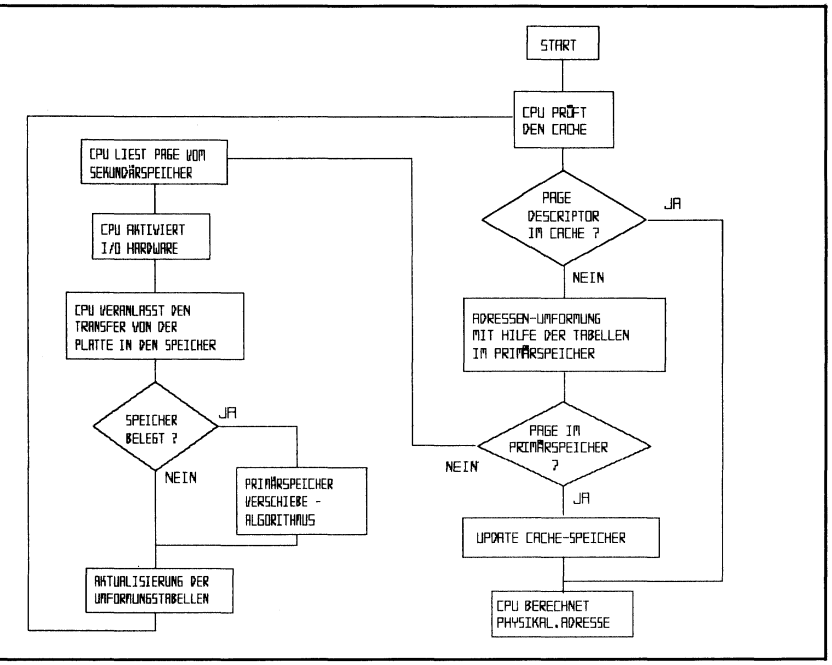


Bild 8

der Umformungstabelle gespeichert. In der Regel existieren dort drei Informations-Bits:

- Das «gültige» Bit wird vom Betriebssystem kontrolliert und zeigt an, ob ein Block (Page oder Segment) im Hauptspeicher vorhanden ist.
- Das Zuweisungs-Bit wird von der MMU gesetzt, um anzuzeigen, dass ein Zugriff auf einen Block im Hauptspeicher im Gange ist.
- Das Aenderungs-Bit wird bei jedem schreibenden Zugriff auf einen Block gesetzt. Dieses Bit zeigt an, ob der Block wieder zurückgeschrieben werden muss (gesichert im Sekundärspeicher), bevor er im Hauptspeicher überschrieben wird.

Der Intel 432-Prozessor besitzt sogar vier Aktivitäts-Bits in seinem Segment-Descriptor:

- Das gültige Bit zeigt an, ob das Segment im Hauptspeicher ist oder nicht (V).
- Das Speicherzuweisungs-Bit (S) wird wie oben beschrieben gehandhabt.
- Das Aenderungs-Bit (AL) zeigt an, ob die im Segment enthaltene Information geändert wurde.

Das Betriebssystem bedient sich hier der Bits V und S, wenn im physikalischen Speicherraum das gewünschte Segment nicht vorhanden war. Die AL- und AC-Bits werden vom Plazierungs- und Verschiebealgorithmus für die Entscheidung gebraucht, welche der bereits vorhandenen Segmente mit den angeforderten neuen getauscht werden können.

Zusätzlich können verschiedene andere Felder im Segment-Descriptor vom Betriebssystem dazu gebraucht werden, andere Informationen über das Segment abzulegen (Zugriffshäufigkeit usw.).

Die anderen Mikroprozessoren benutzen ähnliche Methoden des Betriebssystems für die Zugriffsaktivitäten. Eine allgemein gebräuchliche Page-Verschiebetechnik ist:

LRU Last Recently Used (die zuletzt gebrauchte)
LFU Last Frequently Used (die letzte häufig gebrauchte)
FIFO First in - First out (die zuerst eingelesene, wird zuerst ausgegeben)

Der bekannteste LRU-Algorithmus kann die vorhandenen Pages in vier Gruppen einteilen:

Gruppe 1: nicht zugewiesen (R = 0) und unverändert (M = 0)
Gruppe 2: nicht zugewiesen (R = 0) und geändert (M = 1)
Gruppe 3: zugewiesen (R = 1) und unverändert (M = 0)
Gruppe 4: zugewiesen (R = 0) und geändert (M = 1)

Die Pages der niederwertigsten Gruppe werden immer zuerst ersetzt und die der höchstwertigen Gruppen zuletzt. Das entsprechende Bit (R) wird immer dann gesetzt, wenn die Page zugewiesen wird. Das Betriebssystem überprüft periodisch das zugewiesene Bit. Immer wenn ein solches gesetztes Bit vom Betriebssystem angetroffen wird, wird die Häufigkeit des Zugriffs auf eine bestimmte Page gezählt. Bit (M) wird von der CPU bei schreibendem Zugriff auf die entsprechende Page gesetzt.

Soll ein Page im Speicher getauscht werden, so prüft das Betriebssystem dieses Bit, um festzustellen und entscheiden zu können, ob die Page im Sekundärspeicher ein Update erfahren soll, d.h. ob die Page aus dem Hauptspeicher wieder (in nun geänderter Form) auf die Disk zurückgespeichert werden soll.

Und nun noch ein wichtiges Merkmal eines Prozessors, der die virtuelle Speichertechnik beherrscht. Es geht hier um das Wiederaufnehmen der Operation nach einer Unterbrechung (Adressenfehler) oder anders ausgedrückt, nach dem vergeblichen Zugriff auf eine Page im Hauptspeicher.

Es gibt grundsätzlich zwei verschiedene Möglichkeiten der Realisierung dieses Problems, einerseits die Methode des Instruktions-Neustart und andererseits die Methode der Instruktions-Fortführung. Die Vor- und Nachteile dieser Methode werden im folgenden erläutert.

Prozessor	Adressenraum real	Adressenraum virtuell	Adressenumformungs-schema	Mapping-levels	Associative-Cache
Intel 80286	16M	1G	Segmentierung	1	4 Segment-Descriptor Register
Intel 432	16M	1T	Segmentierung	2	TLB-Cache-Speicher
Intel 80386	4G	64T	Paging und Segmentierung	2	TLB-Cache-Speicher
MC 68000	16M	4G	Wählbar vom Anwender	1	32 Inhalt-Adressen-Register
MC 68010 + 58451 MMU	16M	4G	Wählbar vom Anwender	1	32 Inhalt-Adressen-Register
MC 68020 + 68851 MMU	4G	?	Wählbar vom Anwender	1	32 Inhalt-Adressen-Register
Z 8001 + Z 8010 MMU	16M	4G	Segmentierung	1	64 Segement-Inhalt-Addressen-Register
Z 8003 + Z 8015 MMU	8M	4G	Paging (Seitengröße = 2K)	1	64 Page-Deskriptor-Regsiter
Z 800	16M	4G	Paging (Seitengröße = 4K)	1	16 Adressen-Zungangs-Register
Z 80.000	16M	4G	Paging (Seitengröße = 1K)	3	TLB-Cache-Speicher
NS 16032					
NS 32032 + 16082 MMU	16M	4G	Paging (Seitengröße = 512K)	2	32 Inhalt-Adressen Cache
NCR/32 + NCR 32101	16M	4G	Paging (Seitengröße = 1K)	1	16 Assoziativspeicher
WE 32100 + WE 32101 MMU	4G	?	Paging und Segmentation	1	64-Eingang-Page-Descriptor-Tabellen 32-Eingans-Segment-Descriptor-Tabelle

Bild 9

Methode des Instruktions-Neustart

Nachdem die Adressenfehlerroutine beendet ist, wird die Instruktion, in welcher der Adressenfehler aufgetaucht ist, wieder von Anfang an gestartet. Bild 10 erläutert die Ausführung des Mikrocode in dem Fall, wenn kein Adressfehler vorhanden ist und die Instruktions-Neustartmethode bei Auftreten eines Adressenfehlers.

Das Hauptproblem bei dieser Methode ist die Rekonstruktion des Maschinenzustandes vor der Fehlerbehandlung. Bestimmte Instruktionen verlangen ein Maximum an Daten zur Wiederherstellung (z.B. Input/Output-Operationen). Markantes Beispiel hierfür sind die extended-precision Arithmetik-Operationen. Hier sind die Ergebnisse (Outputs) der vorhergehenden Operationen in der Regel die Parameter für die folgenden Funktionen (Input) Diese liefert wieder Ergebnisse für die darauf folgenden Instruktionen usw.

Verschiedene Techniken werden zur Lösung dieses Problems herangezogen:

- Der Prozessor kann dem Anwender sichtbare Details vorenthalten, bis geklärt ist, dass kein Adressenfehler vorliegt.
- Alle Aenderungen dieser Details werden vom Prozessor aufgezeigt, wenn ein Adressenfehler auftritt. Mit dieser Information wird der Prozessor in der Lage sein, den Originalzustand wieder herzustellen.
- Der Prozessor fertigt sich Kopien aller Aenderungen an. Weil die Kopie immer den Originalzustand wiederspiegelt, ist es kein Problem mehr, den Ausgangszustand zu erreichen.

Die NS 16082 MMU sendet ein Abbruchsignal zur CPU (NS 16032 oder NS 32032), welches die CPU anhält. Alle benötigten Informationen werden daraufhin gerettet (Programmzähler, Maschinenstatus, Stack-Pointer und verschiedene andere Register). Die Rückkehr und Wiederaufnahme der Tasks erfolgt wie oben beschrieben.

Auch die Zilog-Prozessoren verwenden die Instruktion-Neustartmethode. Das Z8001/8015-System enthält ein spezielles Datenzählerregister, welches die erfolgreichen Datenzugriffe vor einem Adressenfehler zählt. Auch diese Information dient zur Wiedererlangung des Maschinenzustandes der vor dem Adressenfehler herrschte.

Die Z80000 und Z800-Prozessoren, welche ein On-Chip-MMU besitzen, gebrauchen eine verbesserte Instruktion-Neustarttechnik kompatibel zu ihrer Pipeline-Architektur.

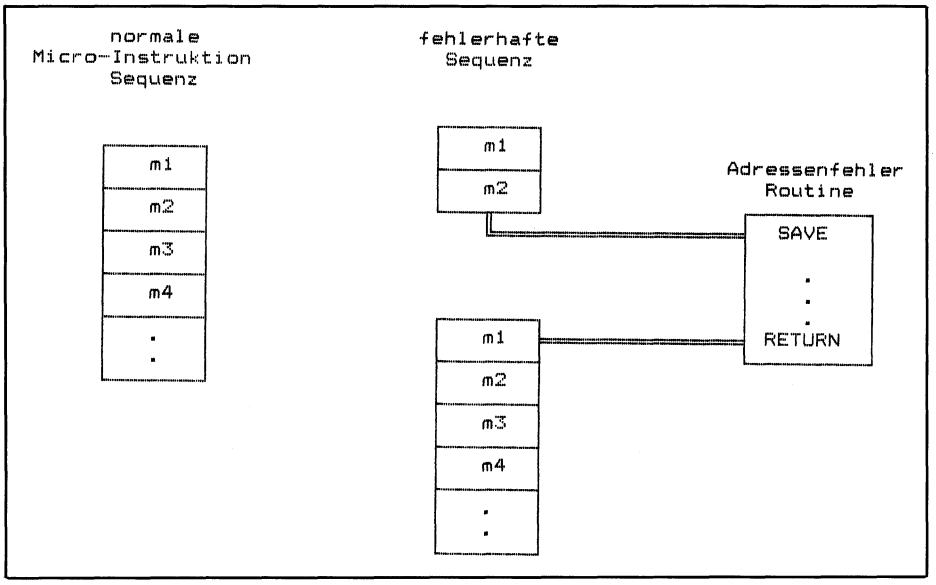


Bild 10 nimmt an, dass eine Maschinen-Instruktion aus verschiedenen Mikro-Instruktionen besteht: m1, m2, m3, m4 ... Ohne Adressenfehler werden diese Instruktionen nacheinander ausgeführt. Wenn die MMU einen Adressenfehler findet (hier in M2) wird die Kontrolle der Adressenfehlerroutine übergeben. Diese rettet zunächst den Zustand der Maschine. Daraufhin wird die Routine den Fehler behandeln, sie wird die verlangte Page oder das Segment vom Sekundärspeicher holen, den Zustand der Maschine anhand der zuvor geretteten Zustandsdaten wieder herstellen und die Instruktion erneut starten.

Der Z80000 arbeitet mit einem Sechs-Ebenen-Pipelining, was die Entdeckung eines Adressenfehlers noch vor dem Speicherzugriff ermöglicht. Die Adressenumformung ist in der dritten Pipeline-Ebene ausgeführt und die Ausführungs-Ebene kann gestoppt werden, bevor irgendein Registerinhalt geändert wurde.

Der Z800 gebraucht eine ähnliche Technik, jedoch mit einer Drei-Ebenen-Pipeline.

Auch die Intel-Prozessoren 286 und 386 verwenden den Instruktions-Neustart. Auch sie besitzen die Fähigkeit, den Adressenfehler vor Ausführung einer Instruktion zu erkennen, somit wird der Neustart der Instruktion wesentlich vereinfacht. Nach der Ausführung der Adressenfehlerroutine plaziert die CPU die Adresse der unterbrochenen Instruktion in den Instruktionszeiger und die Programmausführung kann fortgeführt werden.

Instruktion Fortführungs-Methode

Hierbei wird im Gegensatz zum vorher beschriebenen die Instruktion nicht von neuem begonnen (nach einem Adressenfehler), sondern an der Stelle des Abbruches wieder aufgenommen (nach Fehlerbehandlung).

Die Ausführung der Sequenz von Mikroinstruktionen nach dieser Methode ist in Bild 11 gezeigt.

Der Adressenfehler wurde hier in der Mikro-Instruktion m2 entdeckt. Daraufhin wurde die Kontrolle an die Adressenfehlerroutine übergeben. Nach Beendigung dieser Routine wird die Operation bei der Mikroinstruktion m3 wieder aufgenommen. Diese Art der Adressenfehlerbehandlung verläuft analog zu einer Interrupt-Operation auf Mikroinstruktions-Niveau.

Zur Durchführung dieser Methode muss der Prozessor fähig sein, den kompletten Zustand der Maschine beim Auftreten eines Adressenfehlers zu retten. Hierzu ist es notwendig, eine grosse Anzahl temporärer Registerinhalte zwischenzuspeichern. Mikroprozessoren, welche sich dieser Technik bedienen, haben daher in der Regel einen sehr grossen Adressenfehler-Stack, der alle diese Informationen aufnehmen kann (z.B. MC 68010).

Hier gibt es allerdings Probleme mit Instruktionen, die eine Ausführung ohne Unterbrechung verlangen. Hierzu zählen z.B. Multiprogramming- und Multiprozessing-Systeme.

Bei einem auftretenden Adressenfehler in einer dieser Instruktionen kann es nötig sein, diese neu zu starten. Daher muss der Prozessor die Fähigkeiten besitzen, solche Instruktionen zu erkennen, um entsprechend zu reagieren. Dieses Vorgehen ist sehr

zeitintensiv und verlangt eine grosse Flexibilität des Systems. Realisiert wird diese Methode lediglich im MC 68010 und MC 68020, alle andern Mikroprozessoren arbeiten mit der «Neustartmethode».

Schutz- und Sicherheits-mechanismen

Gerade beim Multitasking- und Multiuser-Betrieb wird von der Prozessor-Architektur ein hohes Mass an Unterstützung von Schutz- und Sicherheitsmechanismen zur Leistungssteigerung des Systems verlangt. Grundlegende Schutz- und Sicherheitsmechanismen umfassen folgende Bereiche: Speicherschutz, Programmschutz, Anwenderschutz und Datensicherheit.

Die Speicherschutzmechanismen sind verantwortlich für das Auffinden von Adressenfehlern vor Auftreten eines Defektes. Jede Instruktion muss daraufhin untersucht werden, ob sie die beabsichtigte Operation auch durchführen kann. Die Ueberprüfung obliegt der MMU. Entdeckt sie einen Adressenfehler, so wird die Adressenfehlerroutine aktiviert, welche den Fehler analysiert, gegebenenfalls diesen markiert und anschliessend zum unterbrochenen Programm zurückkehrt.

Programmschutzmechanismen sollen die Anwenderprogramme daran hindern, illegale Eingriffe in das Betriebssystem vorzunehmen. Sie sollen auch die Uebertragung zwischen den System-Modulen kontrollieren, um einen zuverlässigen Ablauf zu gewährleisten.

Der Anwenderschutz soll die Anwender bzw. deren Applikationen voreinander schützen.

Datensicherheit ist letztlich dann gegeben, wenn das System zu bestimmten Informationen nur beschränkten Zugriff gestattet und dies auch gewährleisten kann.

Bewährt haben sich zwei verschiedene Schutzsysteme für Programm- und Anwenderschutz, das hierarchische Schutzsystem (Ringschutz) sowie das nicht-hierarchische Schutzsystem.

Das hierarchische Schutzsystem besteht aus abgestuften Ringen oder Niveaustufen. Die Abstufung erfolgt von den am meisten privilegiertesten zu den am wenigsten privilegiertesten. In der Regel können Programme nur auf solche Daten zugreifen, die sich auf gleicher oder niederer Niveaustufe befinden. Dienstleistungen und Hilfen kann das Programm jedoch nur von höheren oder der gleichen Niveaustufen anfordern (siehe Bild 12).

Das Ringsystem ist in den Intel-Prozessoren 286 und 386 verwirklicht. Bild 13 zeigt die Anordnung der vier Privileg-Ringe. Verschiedene Prioritäten können verschiedene Programme (oder Segmente) innerhalb eines Systems zugewiesen werden. Dabei sind die grösseren Privilegien den wichtigsten Programme zugeteilt. Normalerweise erhält das Betriebssystem die höchsten Privilegien und ist damit automatisch vor Applikationsprogrammen geschützt. Die Programme können hier wesentlich schneller Betriebssystem-Dienste abrufen, als bei der Verwendung herkömmlicher Techniken (z.B. Zusammenschaltung von Betriebssystem und Applikation).

Der 2. und 3. Ring wird normalerweise für System-Dienste und Erweiterungen vorgehalten, während die Anwenderprogramme für gewöhnlich im am wenigsten priviligierten Ring angesiedelt werden. Die Trennung der Ringe untereinander übernimmt ein spezieller Stack für jeden Ring. Der Intel 286 und 386 benutzt zudem separate Descriptor-Tabellen.

Nicht-hierarchische Schutzsysteme definieren für jede Task eine Tabelle von Operationen. Hier wird festgelegt, welche Operation auf andere Tasks im System einwirken können und welche nicht. Um Operationen zuzulassen, die auf eine andere Task einwirken können, müssen die Einträge in der Tabelle übereinstimmen. Dieses System ist sehr komplex und heutige Prozessoren verwirklichen es noch nicht in ihrer Architektur, sondern im Betriebssystem.

Heutige Prozessoren liefern einige Schutzmerkmale, welche in elegante Software-Schutzsysteme eingebaut werden können. Der MC 68000, der Z8000 und der NS 16000 Prozessor haben jeweils zwei Operating-Modes (oder Privilegstufen) der CPU: Supervisor-Modus und Anwender-Modus. Im Supervisor-Modus kann die CPU auf ihren kompletten Instruktionssatz zurückgreifen, während ihr im Anwender-Modus nur ein Teil davon zur Verfügung steht. Beim Zilog-Prozessor werden diese Modi «System» und «Normal» genannt.

Normalerweise sind die Betriebssystem-Funktionen auf dem Supervisor (Ueberwacher)-Niveau angesiedelt, während die Applikationen auf dem Anwender-Niveau laufen; somit ist das Betriebssystem geschützt vor unerlaubten Zugriffen der Anwender-Programme. Aus diesem Grunde hat das Betriebssystem die Möglichkeit, sowohl den Prozessor selbst, als auch die «äusseren Funktionen» (z.B. I/O-Operationen) zu kontrollieren.

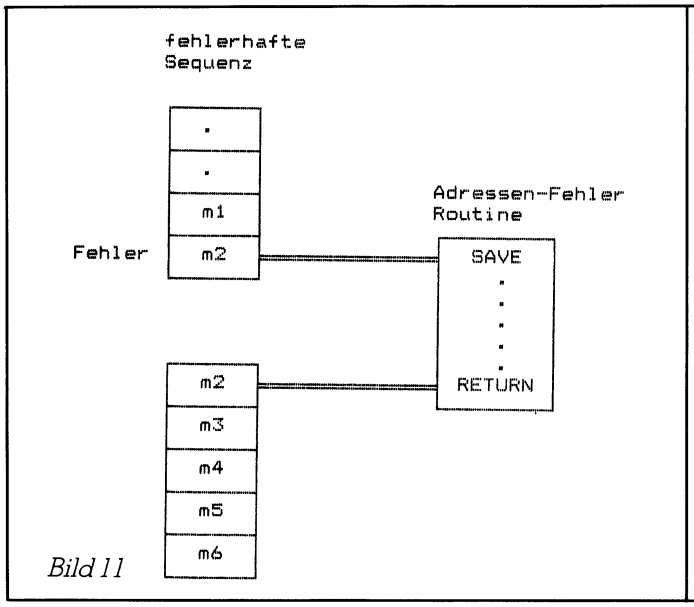


Bild 11

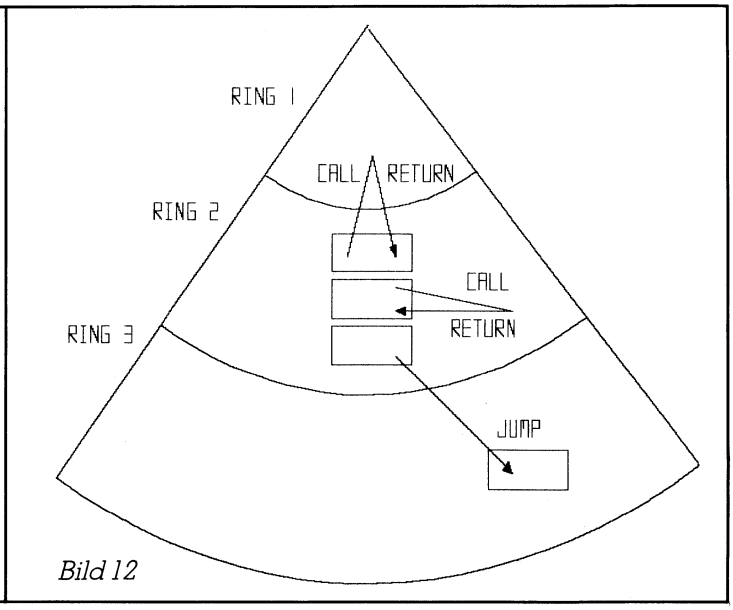


Bild 12

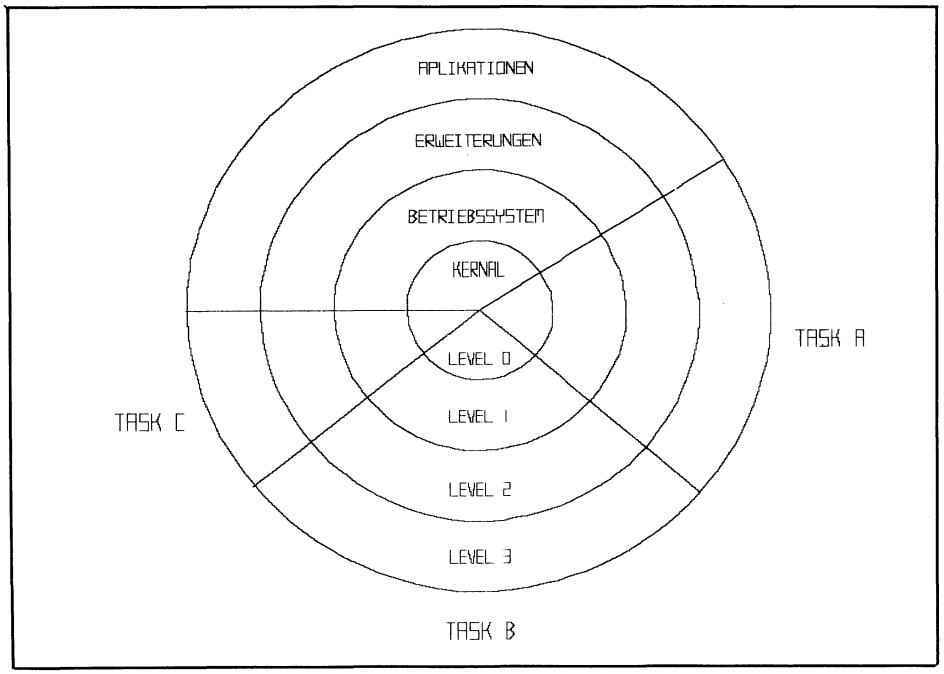


Bild 13 zeigt die Anordnung der vier Privileg-Ringe

Zusätzlich stellen diese Prozessoren separate Adressräume für jeden laufenden Prozess bereit und schützen somit einen Anwender vor dem anderen.

Der MC 68020 verwaltet sogenannte Zugangs-Level, welche eine Einteilung in bis zu 256 hierarchische Niveaustufen zulassen.

Die Sicherheit der Daten ist abhängig von den Zugriffsmöglichkeiten. Grundsätzlich wird einem Programm erlaubt, nur auf diejenigen Daten zuzugreifen, die hierfür bestimmt sind. Daher sollte jede Prozedur in einem geschützten Bereich laufen und somit Zugriffe nur innerhalb dieses Bereiches möglich sein. Diese Sicherheit wird dadurch erreicht, dass jedem Prozess bestimmte Zugangsrechte zu Pages oder Segmente verliehen werden. Die häufigsten Zugangsrechte sind:

1) Lesender Zugriff: Der Prozess kann jede Information einer Page oder eines Segmentes erreichen.
2) Schreibender Zugriff: Der Prozess kann eine Page oder Segment ändern und somit hier zusätzliche Informationen plazieren. Der Prozess kann ebenso alle hier enthaltenden Informationen zerstören.
3) Ausführender Zugriff: Der Prozess kann die Page oder das Segment wie ein Programm behandeln und es zur Ausführung bringen

Charakteristischerweise speichern heutige Prozessoren alle Rechte in Page- oder Segment-Descriptoren. Bevor ein Prozess auf ein Page oder ein Segment zugreift, werden dessen Zugangsrechte überprüft. Erst die Bestätigung dieser Rechte erlaubt den Zugriff.

Ungeachtet von der virtuellen Speicherverwaltung haben bei diesem System alle Anwender die gleichen Zugangsrechte zu Pages oder Segmenten, da die Rechte mit den Pages und den Segmenten verbunden sind, nicht aber mit den Anwendern.

Das Problem kann mit einem 2-Level-Mapping, wie es schon für den Intel 432-Prozessor beschrieben wurde, gelöst werden. Hier sind die Zugangsrechte mit den Anwendern verbunden, unabhängig von Segmenten oder Pages.

Fazit

In diesem Beitrag wurden nur Techniken und deren Realisation beschrieben und dort wo angebracht diese auch bewertet. Es geht hier nicht um die Wahl des «besten Mikroprozessors», dafür ist deren Architektur zu vielschichtig, um in allen Details verglichen werden zu können und hier wurde nur ein kleiner Bruchteil des Systems beleuchtet. Dennoch ist ein allgemeines Fazit zu ziehen: Heutige Mikroprozessor-Architekturen kommen langsam in die Jahre, d.h. sie veralten noch bevor sie vollständig ausgereift sind und sie sind zu vielschichtig. Gerade heutige Systemanforderungen verlangen immer mehr Leistung und innovative Techniken - aber auch vereinfachende Konzepte werden in Zukunft gefragt sein. □

COMPUTER-SPLITTER

CD-ROM-Leser von NEC

(598/fp) NEC hat interne und externe Laufwerke für CD-ROMs angekündigt. Die Applikationen von NEC (für die PC-Familie, PS/2 und Macintosh) umfassen die Laufwerke, die Adaptoren und Treiber-Software, mittels welcher sich die Laufwerke über normale DOS-Befehle ansprechen lassen. □

20 VAX-MIPS

(482/eh) Immer mehr amerikanische Hersteller bringen Prozessorbausteine auf den Markt, die auf der von SUN entwickelten SPARC-Technologie beruhen, so auch Cypress Semiconductors. Deren Prozessorchip wird mit einer Taktfrequenz von 33 MHz betrieben und ist in seiner Leistung vergleichbar mit einer VAX, die 20 Millionen Instruktionen pro Sekunde abarbeiten kann. □

Fernbehandlung

(602/fp) Da gibt es die ganz grossen LAN-Pakete oder das ganz kleine Desk-Link und wie sie alle heissen für den remote-Eingriff in andere PCs. Das müsste man jetzt noch per Telefon tun können. Kann man. PhoneBoot schaltet das ferne Computersystem ein und schaltet die Daten transparent durch. Mit Paketen aus dem genannten Bereich wird dann auf die Daten zugegriffen. Oder man betreibt auf die Ferne Diagnose und Therapie. Ob als Opfer von Problemen oder als Täter zu deren Beseitigung: Ideen für und Bedarf nach entsprechenden Dienstleistungen gibt es in der DOS-Welt noch genug! (Cybex Corporation, 2800 H. Bob Wallace Avenue, Huntsville, AL 35805 USA.) □

DOS 4.0 für die schweigende Mehrheit

(593/fp) Wird IBM nach 3.3 an DOS nicht mehr weiterarbeiten? Aber vielleicht Compaq? Wird OS/2 die Welt in zwei Hemisphären teilen? Könnte es sein, dass usw. usf. IBM hat der Gerüchteküche selber abrupt den Strom abgestellt und DOS 4.0 angekündigt. DOS 4.0 - in der englischen Version schon erhältlich - sprengt das 32-MB-Korsett auf Festplatten, hat eine grafische Benützeroberfläche und gut dreissig erweiterte oder neue Funktionen. DOS 4.0 läuft auf allen PCs und Personal Systemen. □

Es ist uns schrecklich peinlich. Dieses WITCH-DOS

müssen Sie unbedingt haben, können es auf dieser ganzen Welt jedoch einzig und allein bei uns kaufen, und sein Wert ist DM 9998.–. Weil dieser Stand der Dinge für Sie und für uns ganz unmöglich ist, machen wir einen Rabatt von DM 9800.– und bieten Ihnen das Programm zum Preise von DM/SFr.198.– an.

(Ein Barbar wäre, wer von allen Dingen den Preis weiß, und von keinem den Wert – Oscar Wilde)

WITCH-DOS ist die erste und weltweit einzige hohe Programmiersprache für Jobs zur Unterstützung von Anwenderprogrammen aus dem Hintergrund. Mit WITCH-DOS können Sie auf einfachste Weise Unentbehrliches in der Art von Sidekick, Norton-Commander, Superkey usw. herstellen. Bei andern können Sie das wärmende Bärenfell kaufen. Bei uns jedoch die Präzisions-Flinte namens WITCH-DOS.

1. MS-DOS schläft friedlich, während Sie sich in Lotus, dBase, Pascal usw. abrackern
1. WITCH-DOS greift Ihnen aber gerade dann hellwach unter die Arme

2. MS-DOS ist ein schöner Anfang
2. WITCH-DOS jedoch ist die gute Idee zu Ende gedacht

3. MS-DOS hat nur ein Minimum von Funktionen, z.B. «Notbehelf-Drucken»
3. WITCH-DOS bietet einen maximalen Befehlssatz

Einfaches Programmierbeispiel eines Jobs der die Wirkungen der y- und z-Tasten vertauscht

```
when («y», vert1) when («z», vert2) user
vert1:  key («z») user
vert2:  key («y») user
```

Erklärung: Während z.B. Lotus oder ein anderes Benützerprogramm läuft, achtet dieses WITCH-DOS-Programm darauf, ob die Taste «y» oder «z» getippt wird. Wenn ein «y» getippt ist, richtet der Computer seine Aufmerksamkeit auf WITCH-DOS, und führt jene Befehle aus, die auf die Sprungadresse «vert1:» folgen. WITCH-DOS betätigt die Taste «z». Mit dem Befehl «user» geht die Kontrolle wieder zum Benützerprogramm.

So einfach können Sie Ihre ganze Tastatur total umkrempeln. Sie sehen, WITCH-DOS ist nicht komplizierter als HK, BASIC oder PASCAL. Von einer (ebenfalls durchführbaren) Systemprogrammierung mit C unterscheidet sich WITCH-DOS jedoch wie ein Skilift vom Kraxeln (Kraxeln ist nicht etwa schlecht, aber anders, vor allem langsamer.)

Mitgelieferte gebrauchsfertige Quellcodes

Generator für vollautomatisch ablaufende Demos. Generator für Demos mit «Serienfotos» von Situationen auf Textbildschirmen. Record und Play von Tastenfolgen. Universaltabulator. Help-Windows für Programme. Lernprogramm mit Prüfung der korrekten Eingaben. Sowie viele Kleinprogramme.

Der Befehlssatz von WITCH-DOS

Bildschirmfenster, Balken-Menüs, Tastenzuordnungen, Bildschirm «von innen lesen», Tastenfolgen mit korrekten Zeitabläufen, Tasten sperren. Record und Play auch über mehrere Programme hinweg. Bit-Manipulationen. Peek und Poke. Rechnen. Bedingte Sprünge. Logische Verknüpfungen. Subroutinen. Chain zu andern Jobs. Modulbauweise (Montage mehrerer Jobs beim Compilieren). Serielle Dateien lesen und schreiben. Drucken, Kommunikation, Directory lesen usw.

ADD, AND, APPEND, ASC, BEEP, CALL, CHAIN, CHR, CLICK, CLR, CLRTAB, CONCAT, CONST, CREATE, DEC, DIR, DIV, EXIT, FILTER, FLUSH, GETKEY, GETNUM, GETSTR, GOTO, IF, IFCLR, IFDOS, IFEOF, IFEQ, IFGE, IFGT, IFKEY, IFLE, IFLT, IFNE, IFNOT, IFNOTDOS, IFNOTEOF, IFNOTKEY, IFNOTSCR, IFSCREEN, IFSET, IFSTREQ, IFSTRGE, IFSTRGT, IFSTRLE, IFSTRLT, IFSTRNE, IFTAB, INC, INITTAB, KEY, LOCASE, LOOP, MENU, MENUREAD, MOD, MOVE, MOVESTR, MULT, NOCLICK, NOFILTER, NOP, NOSPELL, NOTAB, NOWHEN, NUMTOSTR, ONLY, OPEN, OPENTEXT, OR, PEEK, POKE, PRINT, PRINTLN, RECORD, READ, READLN, REMOVE, RETURN, SCRCOLOR, SCREEN, SET, SETTAB, SHL, SHOW, SHR, SPEED, SPELL, STRTONUM, SUB, SUBSTR, TAB, TRIM, USER, USERKEY, VAR, WAIT, WHEN, WHENNOT, WHERE, WINDOW, WINDOWF, WRITE WRITELN, WRITESTR, XOR.

Ihre Lizenzrechte nach dem Kauf von WITCH-DOS

Sie können Ihre Programme als Quellcode oder in compilierter Form beliebig weiterverkaufen. Jeder Benützer muss jedoch seinerseits WITCH-DOS erwerben. Das versetzt ihn in die Lage, Ihre Programme zu benützen.

WITCH-DOS läuft mit Witchpen, Lotus, dBase III, Ventura, Norton usw. WITCH-DOS läuft nicht mit Word und Wordstar.

Info-Broschüre: gratis **Demodisk: 10 DM/10.– SFr.** **WITCH-DOS: DM 198.–/198.– SFr.**

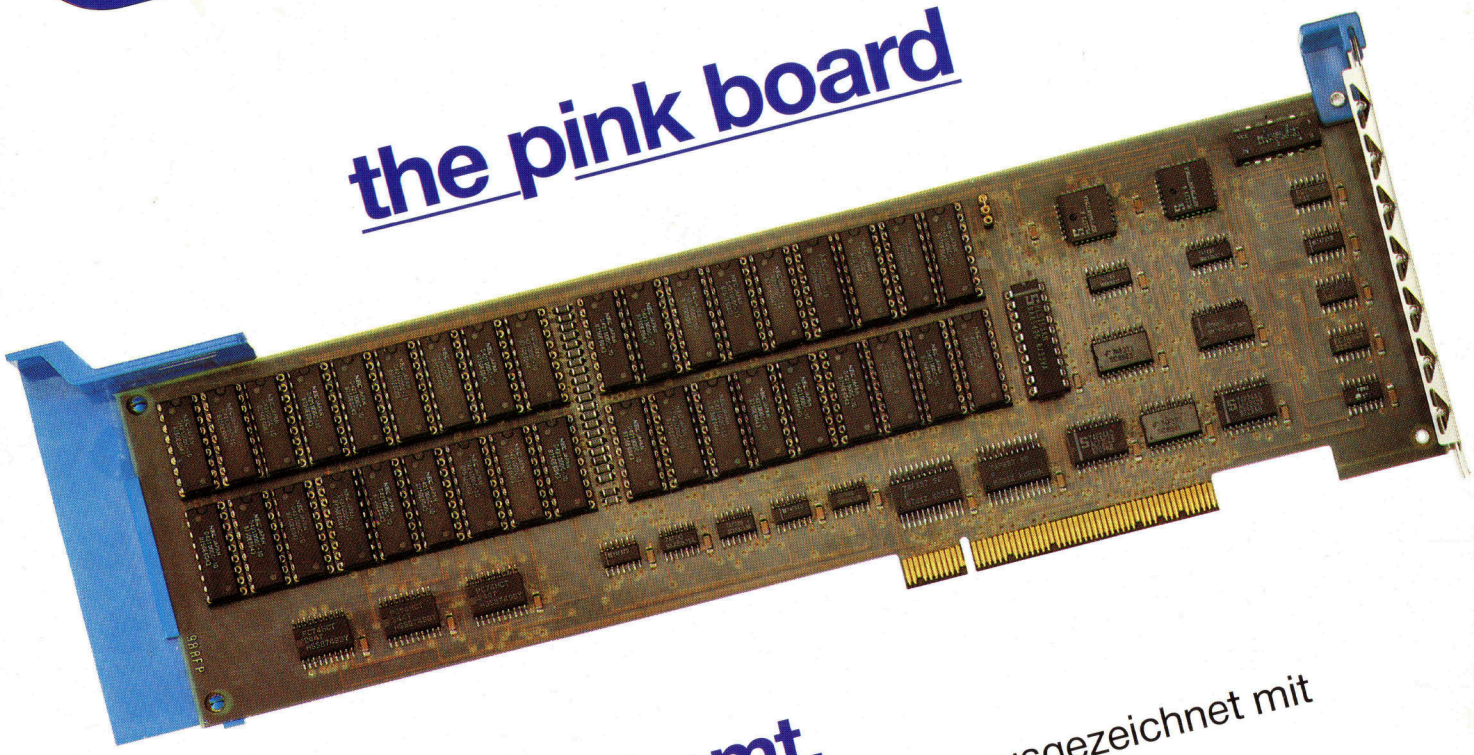
Die Benützer von WITCHPEN mal 5 (Preis DM 580.–/SFr. 495.–) sind Menschen mit einem besonders ausgeprägten Sinn für Eleganz, die bekommen WITCH-DOS natürlich geschenkt).

Hannes Keller Witch Systems AG

Hannes Keller Witch Systems AG, Eidmattstr. 36, 8032 Zürich, Schweiz, Tel. 01/251 14 15
Hannes Keller Witch Systems GmbH, Breitestr. 3, 7890 Waldshut, Deutschland, Tel. 07741/3065

Infosys

the pink board



Die Technik stimmt.

PS/2* Modelle 50, 60 und 80 vertragen sich ausgezeichnet mit „the pink board“.
OS/2* ist mit zusätzlichen 4 MB Speicher optimal bedient.

Der Preis stimmt.

Durch den Einsatz hochmoderner Technologie kann ein unglaublicher Preis realisiert werden.
„the pink board“ verwendet 1MB Chips.

Die Qualität stimmt.

„the pink board“ ist erneut ein Beweis für die Leistungsfähigkeit europäischer Computertechnik.
Auf Grund seiner „Made in Germany“ Eigenschaften ist „the pink board“ eines der ganz wenigen PS/2* boards, die nach USA exportiert werden.

„the pink board“ erhalten Sie inkl. 4 MB RAM für nur **Fr. 3’156.–**

*PS/2 und OS/2 sind eingetragene Warenzeichen der International Business Machines Corporation.

Computer Elektronik Infosys GmbH, Am Kümmerling 2, 6501 Bodenheim, Telefon 061 35/770
Computer Elektronik Infosys AG, Im Rötel 10a, CH-6301 Zug, Telefon 042/21 78 57
Lager und Vertrieb Schweiz: Oberdorfstrasse 11, CH-8953 Dietikon, Telefon 01/741 30 41