Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 80

Service: Diskettenmonitor

Textkompressionen

Bildschirmsteuerungen

Portabler PC

Ein wöchentliches Sammelwerk

computer kurs

Heft 80

Inhalt

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

Computer Kurs ist ein wöchentlich erscheinendes Sammelwerk. Die Gesamtzahl der Hefte ergibt ein vollständiges Computer-Nachschlagewerk. Damit Sie jede Woche Ihr Heft erhalten, bitten Sie Ihren Zeitschriftenhändler, Computer Kurs für Sie zu reservieren.

Zurückliegende Hefte
Ihr Zeitschriftenhändler besorgt Ihnen gerne zurückliegende Hefte. Sie können sie aber auch direkt beim Verlag bestellen.

Deutschland: Das einzelne Heft kostet DM 3,80. Bitte füllen Sie eine Postzahlkarte aus an: Marshall Cavendish Int. Ltd. (MCI), Sammelwerk-Service, Postgiroamt Hamburg 48064-202, Postfach 105703, 2000 Hamburg 1, Kennwort: Computer Kurs

Österreich: Das einzelne Heft kostet öS 30. Bitte füllen Sie eine Zahlkarte aus an: Computer Kurs, Wollzeile 11, 1011 Wien, Postscheckkonto Wien 7857201 oder legen Sie Ihrer Bestellung einen Verrechnungsscheck bei. Kennwort: Computer Kurs.

Schweiz: Das einzelne Heft kostet sfr 3,80. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

Abonnement

Sie können Computer Kurs auch alle 2 Wochen (je 2 Ausgaben) per Post zum gleichen Preis im Abonnement beziehen. Der Abopreis für 12 Ausgaben beträgt DM 45,60 inkl. Mwst., den wir Ihnen nach Eingang der Bestellung berechnen. Bitte senden Sie Ihre Bestellung an: Marshall Cavendish Int. Ltd. (MCI), Sammelwerk Service, Postgiroamt Hamburg 86853-201, Postfach 105703, 2000 Hamburg 1, Kennwort: Abo Computer Kurs. Bitte geben Sie an, ab welcher Nummer das Abo beginnen soll und ob Sie regelmäßig für jeweils 12 Folgen einen Sammelordner wünschen.

WICHTIG: Bei Ihren Bestellungen muß der linke Abschnitt der Zahlkarte Ihre vollständige Adresse enthalten, damit Sie die Hefte schnell und sicher erhalten. Überweisen Sie durch Ihre Bank, so muß die Überweisungskopie Ihre vollständige Anschrift gut leserlich enthalten.

SAMMELORDNER
Sie können die Sammelordner entweder direkt bei Ihrem Zeitschriftenhändler kaufen (falls nicht vorrätig, bestellt er sie gerne für Sie) oder aber Sie bestellen die Sammelordner für den gleichen Preis beim Verlag wie folgt:

Deutschland: Der Sammelordner kostet DM 12. Bitte füllen Sie eine Zahlkarte aus an: Marshall Cavendish International Ltd. (MCI), Sammelwerk-Service, Postgiroamt Hamburg 48064-202, Postfach 105703, 2000 Hamburg 1, Kennwort: Sammelordner Computer Kurs.

Österreich: Der Sammelordner kostet öS 98. Bitte füllen Sie eine Zahlkarte aus an: Computer Kurs Wollzeile 11, 1011 Wien, Postscheckkonto Wien 7857201 oder legen Sie Ihrer Bestellung einen Verrechnungsscheck bei. Kennwort: Sammelordner Computer Kurs.

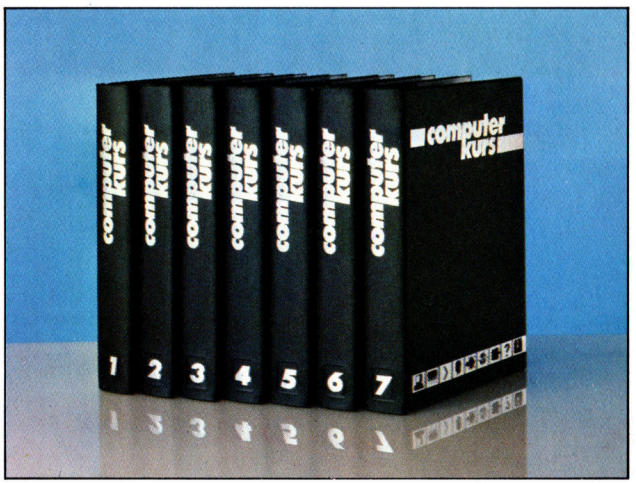
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

Alle 12 Hefte erscheint ein Teilindex. Die letzte Ausgabe von Computer Kurs enthält den Gesamtindex — darin einbezogen sind Kreuzverweise auf die Artikel, die mit dem gesuchten Stichwort in Verbindung stehen.

Redaktion: Peter Aldick (verantw. f. d. Inhalt), Gudrun Anderson, Joachim Knipp, Uta Brandl (Layout), Sammelwerk Redaktions-Service GmbH, Paulstraße 3, 2000 Hamburg 1

Vertrieb: Marshall Cavendish International Ltd., Heidenkampsweg 74, 2000 Hamburg 1



Druck: E. Schwend GmbH, Schmollerstraße 31, 7170 Schwäbisch Hall

Tolles Ding

Unter dem Namen „Dynabook“ wurde 1969 das Konzept eines universell zu gebrauchenden und portablen Computers entwickelt, noch bevor der Microprozessor erfunden war.

Alan Kay, ein junger amerikanischer Student der Computerwissenschaften, präsentierte 1969 sein Konzept eines universellen und transportablen Computers. Dieser „Dynabook“ genannte Computer sollte eine einfache Benutzerführung aufweisen, audiovisuelle Kommunikation unterstützen und über Datenbanken Zugriff auf öffentliche Informationen gewährleisten. Kays Dynabook war eine Zukunftsvision, denn 1969 waren Microprozessoren noch nicht erfunden, und die kleinsten Computer waren damals mindestens so groß wie ein Kühlschrank. Außerdem kosteten diese Ungetüme Summen, die mit fünf Nullen endeten.

1971 arbeitete Kay im Palo Alto Research Center (PARC) der Firma Xerox und war maßgeblich an der Forschungsgruppe beteiligt, die „Smalltalk“ erfand. Diese Computersprache sollte ursprünglich als Programmiersprache und Betriebssystem des Dynabook dienen. Das Xerox-Team baute jedoch den Dynabook nicht, denn selbst 1971, nach der Erfindung des Microprozessors, war die vorhandene Technik noch nicht leistungsfähig genug. Dennoch, die in „Smalltalk“ enthaltenen Ideen, wie z. B. Windows, Symbole oder Maussteuerung, wurden langsam von der Industrie aufgenommen und führten zu Appels Lisa und Macintosh und zum Atari 520ST.

Diese „historische“ Geschichte dient als Anregung, denn nun ist die Zeit gekommen, das Dynabook zu realisieren. Die benötigten Komponenten sind entweder schon erhältlich oder werden es in Kürze sein. Heute sieht Alan Kays Vision schon realistischer aus.

Dynabook sollte nach dem ursprünglichen Konzept nicht größer sein als ein Buch, batte-

Ursprünglich von Alan Kay bei Xerox entworfen, könnte das futuristische „Dynabook“ mit Hilfe der heutigen Technik gebaut werden – der hohe Preis steht dem jedoch noch im Wege. In Zukunft könnte dennoch ein solches Gerät eine unentbehrliche Hilfe werden, besonders für Gruppen (wie ältere Menschen zum Beispiel), die immer mehr in unserer sich schnell entwikkelnden, technischen Gesellschaft isoliert werden. Ein frei definierbares, berührungsempfindliches LCD-Display, Spracheingabe und ein Funknetzwerk würden es dem Anwender ermöglichen, Leistungen vom Wetterbericht bis zum persönlichen Bankservice in Anspruch zu nehmen. Bis sich dieser Service selbst finanziert, könnte das Grundgerät entweder kostenlos oder mit minimalen Gebühren von staatlichen Stellen vergeben werden.

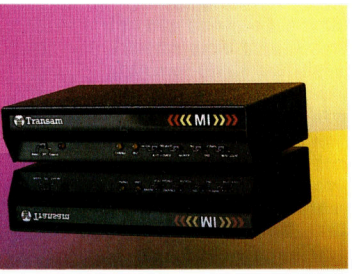
riebetrieben und somit voll portabel. Es sollte Text-, Graphik- und Soundfähigkeiten sowie ein Farbdisplay besitzen. Außerdem sollte es ein leistungsfähiges Kommunikationsmedium sein. Ein wirklich revolutionärer Computer darf nicht einfach nur portabel sein, sondern sollte eine dominierende Rolle spielen im Datenaustausch mit anderen Benutzern, auch über weite Strecken, so wie das Telefon jetzt. In Kürze sollten wir fähig sein, sowohl Briefe zusammen mit Bildern und Tönen zu übertragen als auch zu empfangen.

Fehlende Motivation

Aber dies ist nur der Anfang. Es sollte ein Zugriff auf öffentliche Dienstleistungen möglich sein: von Bibliotheken, deren Bücher und Programme wir auf unser eigenes System übertragen, bis hin zu Nachrichten, Wetterberichten, Unterhaltungs- und Schulungsangeboten. Abfrage des Kontostandes, Einsicht in Straßenpläne, Warenbestellung und Reisebuchungen sollten von jedem Ort aus vorgenommen werden können, unabhängig von Telefonen und Steckdosen.

Obwohl sich vieles des unter „technischen Anforderungen" Beschriebenen in der Entwicklungsstufe befindet und sehr teuer ist, dürfte die Finanzierung des Dynabooks nicht das Problem sein. Sorgfältig ausgeführt wäre es ein idealer Artikel für eine Massenproduktion (insofern er weltweit für jedermann erhältlich wäre). Bei solchen Produktionszahlen sollten Hardwarekosten keine große Rolle spielen. Die Vertriebsprobleme, die heute den Verkauf von Personalcomputern erschweren, würden sich von selbst lösen, könnten die Menschen die Computer wirklich sinnvoll benutzen – und Dynabook wäre nicht nur sinnvoll, sondern unerläßlich. Das größte Problem liegt dabei nicht in der Technologie. Es ist eher die fehlende Motivation, ein solches System einzuführen. Letztendlich ist es ein politisches Problem, denn nur Regierungen verfügen über die Möglichkeiten, ein solches Projekt durchzuführen. Heutzutage existiert bereits alles, was Dynabook tun könnte; vielfach durch öffentliche Mittel finanziert, realisiert mit unterschiedlichen, uneffizienten und teuren Systemen.

In Frankreich werden heute schon Telefonterminals bereitgestellt, da sie billiger als das Drucken und Verteilen von Telefonbüchern sind. Für die Dienstleistungen, weniger für die Hardware, müßten Gebühren erhoben werden, so wie für Telefon und Fernsehen.



Die Einführung eines Funknetzes Mitte der achtziger Jahre hat die Entwicklung des Radio-Modems beschleunigt. Das oben abgebildete Transam M1 ist ein intelligenter Modem, der, in Verbindung mit einem Telefon, automatische Wahl und Auto-Antwort ermöglicht. Weiterhin besitzt er noch eine 15polige D-Buchse zum Anschluß einer Sende- und Empfangsstation zum Senden und Empfangen von Daten über das Funknetz.

Technische Anforderungen

Lassen Sie uns auf die vorhandenen oder im Entstehen befindlichen Technologien blicken, die Dynabook ermöglichen könnten.

- **Verarbeitungseinheit:** Im Herzen der Maschine muß sich ein leistungsfähiger Prozessor befinden. 32-Bit-Prozessoren mit der Leistung größerer Computer sind heute schon als Microchips erhältlich. Jeder dieser Chips kann zwischen drei und zehn Millionen Befehle pro Sekunde ausführen.
- **Der Speicher:** Die bedeutenden Hersteller von Speicherchips produzieren mittlerweile 256-KBit RAM-Chips. Doch läuft bereits die Arbeit an der nächsten Generation, den Ein-Megabit-Chips, auf Hochtouren.
- **Bildschirm:** Einige Hersteller produzieren hauptsächlich flache, monochrome Flüssigkristallanzeigen, die einen IBM-PC-Graphik-Bildschirm mit 640 mal 200 Punkten emulieren. Lichtreflektierende Felder beleuchten diese Displays und beseitigen deren Hauptnachteil – den schwachen Kontrast und den begrenzten Blickwinkel. Japanische Hersteller, unter ihnen Epson, haben bereits Prototypen eines Farb-LCDs vorgestellt, während Toshiba und andere an selbstleuchtenden Plasma-Röhren arbeiten. Der Auftrag der Forschungsarbeit ist die Entwicklung eines Flachbildschirms für Fernseher und weniger eines Computerdisplays, welches Dynabook benötigt. Abzusehen sind auch berührungsempfindliche, flache Displays als Ersatz für die Tastatur.
- **Stromversorgung:** Die Stromversorgung ist noch das größte Problem. Die Leistungen, welche neue Displays und große Speicher verlangen, können herkömmliche Batterien, die auch in Hand-Held-Computern verwendet werden, sicherlich nicht erbringen. Die Batterietechnologie macht, im Vergleich zur Halbleitertechnik, nicht so große Fortschritte. Jedoch sind neue, leistungsfähigere Batterietypen im Entstehen, und gleichzeitig nimmt der Energieverbrauch der Chips immer mehr ab, da die Hersteller auf die sparsamere CMOS-Technik zurückgreifen.
- **Massenspeicher:** Die Compact Laser Disk (CD) ist unempfindlich gegen magnetische Einflüsse. Das Dynabook könnte Read-Only-CDs als Medium zum Vertrieb von „Büchern", auch spezieller Sachgebiete, benutzen. Eine andere Form des Massenspeichers ist die „Smart Card". Dies ist eine Speichereinheit von der Größe einer Scheckkarte, die für alles, vom Softwarevertrieb bis zum Abwickeln von Bankgeschäften, benutzt werden kann. In naher Zukunft könnten solche Karten zum Bezahlen von Waren oder Leistungen mittels Dynabook dienen.
- **Sound:** Es ist in letzter Zeit möglich, alle Komponenten eines Audio-Systems auf einem Chip unterzubringen. Ein Beispiel für den Einsatz dieser Chips ist der Sony Walkman. Sprache ist jedoch ein ganz anderes Kapitel. Synthetische Sprache läßt sich einfach herstellen, und moderne Synthesizer können sie ganz gut darbieten. Das Verstehen natürlicher Sprache befindet sich allerdings noch im Anfangsstadium.
- **Die Kommunikation:** Die restlichen Teile des Dynabook-Puzzles liegen im Bereich der Kommunikation. Telefon Modems werden weithin schon auf einem Chip angeboten, sie sind aber nicht gut genug für Dynabook. Um absolute Bewegungsfreiheit zu erreichen, muß die Verbindung kabellos durch Radiosender hergestellt werden. Das Dynabook-Konzept kann nur durch ein vielfach nutzbares, weltweites Kommunikationsnetz durchgesetzt werden. Großcomputer stellen die Netzknoten dar. Um sie zu erreichen, muß man zuerst eine Funkverbindung zu einer lokalen Computerstation herstellen. Diese sind über unterirdische Glasfaserkabel mit Netzknotenstellen verbunden, welche wiederum mit anderen Netzknoten und Satelliten in Verbindung stehen. Viele dieser Komponenten sind bereits installiert – die Weiterentwicklung stockt noch durch das Fehlen eines einheitlichen Verbindungsstandards.

Zu Befehl!

In den nächsten Folgen dieser Serie gehen wir ausführlich auf den Befehlssatz des Motorola 68000 ein. Zuerst untersuchen wir Übertragungsbefehle wie MOVE und PEA und sehen uns die arithmetischen Anweisungen an.

In den vorangegangenen beiden Folgen hatten wir gesehen, wie der Motorola 68000 Operanden adressiert, und dabei einen Einblick in Grundbefehle wie MOVE und ADD erhalten. Wir werden nun zunächst die Datenübertragungsbefehle untersuchen und uns dann den Arithmetikbefehlen für binäre Berechnungen zuwenden.

Wir beschreiben dabei die Anwendung der wichtigeren Befehle und gehen auch auf Hindernisse und wesentliche Einzelheiten ein. Wenn Sie viel mit dem 68000 arbeiten wollen, sollten Sie sich das Buch „Programmierung des 68000" oder ein anderes der in dieser Serie empfohlenen Handbücher zulegen.

Bevor wir die Befehle im einzelnen behandeln, sehen wir uns erst einmal den Inhalt des Statusregisters (SR) genauer an. Die eine Hälfte dieses Registers enthält die Bedingungscodes mit Informationen über das Ergebnis des letzten ausgeführten Befehls. Jeder Bedingungscode läßt sich als ein Ein-Bit-Speicher ansehen, der einem bestimmten arithmetischen Zustand zugeordnet ist. Hier die Codes im einzelnen:

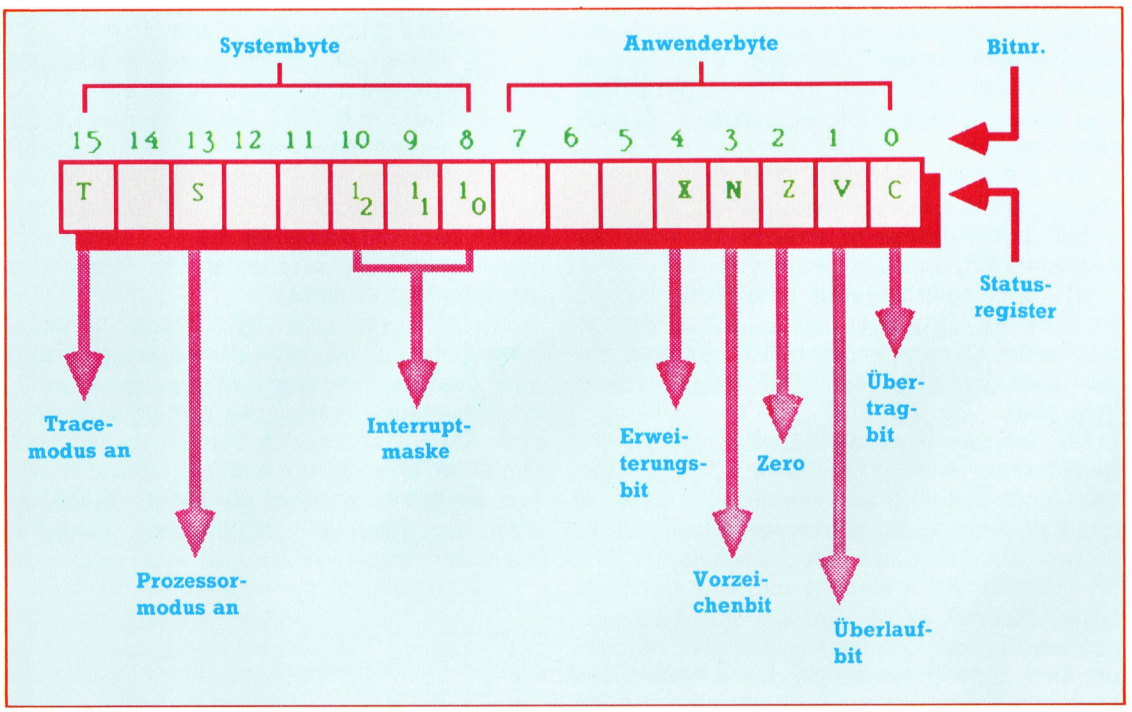
- **BIT 0: Übertrags-Bit oder C-Bit:** Dieses Bit wird gesetzt, wenn ein arithmetischer Vorgang einen Übertrag aus dem höchstwertigen Bit des Datenoperanden ergibt, zum Beispiel:

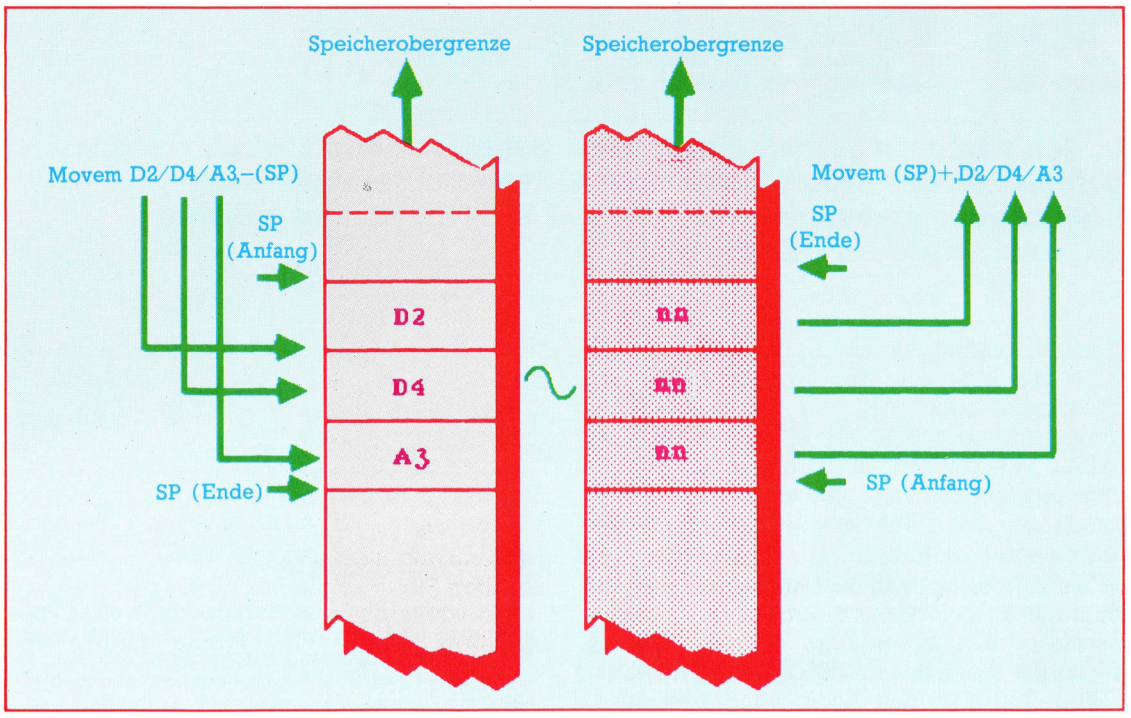
```
Die Addition 01100000
von          11100000
             ---------
ergibt       101000000
```

Statussymbole

Das Statusregister im 32-Bit-Format ist in einen System- und einen Anwenderbereich unterteilt. Unser Bild zeigt die Bedeutung der einzelnen Bits. Ein Statusbit gibt an, in welchem Prozessormodus der 68000 arbeitet, auf Supervisor- und Anwenderebene. Die Überwachung der verschiedenen Arbeitsabläufe ist im Mehrplatzbetrieb notwendig. Normalerweise werden auf dem 68000 alle Aufgaben im Anwendermodus (der über einen eigenen Stackpointer verfügt, aber die Statusbits nicht verändern kann) ausgeführt. Die Systemsoftware (beispielsweise die Sytemsteuerung) läuft im Supervisormodus ab.

Am Anfang werden Sie die Arbeit im Supervisormodus sicherlich einfacher finden, da darin alle Opcodes legal sind, während die Programmierung im Anwendermodus Einschränkungen unterworfen ist. Der 68000 verfügt weiterhin über die Betriebsart ‚Trace', mit der Sie zur Fehlersuche Programme schrittweise ablaufen lassen können. Dabei ruft der 68000 nach jedem Befehl automatisch eine Korrekturroutine auf.





Mit MOVEM können Sie in einem Befehl ganze Registerlisten übertragen. Das Bild zeigt, wie MOVEM mit der Vor-Dekrementierung und der Nach-Inkrementierung drei Register auf den Stack schiebt und wieder herunterzieht. Dabei wird auch die Position des Stackpointers vor und nach jedem Befehl angegeben.

Hier wird die 1 aus dem höchstwertigen Bit des Ergebnisses (mit Byte-Länge) übertragen.

Der Übertrag fließt jedoch nicht in das niederwertige Bit der nächsthöheren Struktur (in diesem Fall – des Computerwortes), sondern in das C-Bit des SR. Ob der Übertrag überhaupt Bedeutung hat, hängt allerdings direkt vom programmierten Ablauf ab. Bei Berechnungen mit höherer Präzision (z. B. wenn der Datenoperand aus mehreren Einheiten besteht) hat der Übertrag große Bedeutung.

- **BIT 1: Überlauf-Bit oder V-Bit:** Dieses Bit wird gesetzt, wenn das Ergebnis der Berechnung nicht in den Datenoperanden paßt. So erzeugt das Addieren von 1 auf 32767 (die höchste positive Ganzzahl bei einem 16-Bit-Computerwort) einen Überlauf im Datenoperanden. Das binäre Ergebnis läßt sich damit nicht korrekt darstellen.
- **BIT 2: Null-Bit oder Z-Bit:** Wird gesetzt, wenn das Ergebnis einer Berechnung Null ist.
- **BIT 3: Vorzeichen-Bit oder N-Bit:** Wird bei negativen Ergebnissen gesetzt.
- **BIT 4: Erweiterungs-Bit oder X-Bit:** Dieses Bit wird bei Vorgängen mit höherer Präzision eingesetzt. Es hat die gleichen Funktionen wie das C-Bit, wird aber von MOVE-Befehlen nicht verändert.

Der flexible Übertragunsbefehl MOVE kopiert Daten von einer Quelle zum Ziel. Er kann die Quelle beliebig adressieren und auch für das Ziel die meisten Adressiermethoden einsetzen (mit Ausnahme der Adreßregister, der PC-relativen Adressierung und des unmittelbaren Modus). Diese Gruppe von Adressierungsarten wird „datenverändernder Modus" genannt. Eine Untergruppe davon stellen die sogenannten „speicherverändernden Modi" dar, die alle Daten mit Ausnahme der Datenregister verändern können.

Beim MOVE-Befehl sind folgende Anweisungen illegal:

```
RORG $1000
MOVE D2,BETTY   PC-relativ ist für BETTY
                nicht möglich
MOVE D2,A2      Adreßregister lassen sich
                nicht als Ziel einsetzen
```

MOVE beeinflußt Bit N und Z des SR, Bit V und Z werden immer auf Null gesetzt.

Adreßregister können auf zwei Wegen als Zieloperand eingesetzt werden:

- MOVEA kopiert den Inhalt des Quelloperanden in das Adreßregister (Ziel).
- LEA kopiert die (normalerweise absolute) Quellenadresse in das Adreßregister.

Diese beiden Befehle haben keinen Einfluß auf die Bedingungscodes. Nach dem gleichen Prinzip arbeiten auch Spezialbefehle, die Daten ins Statusregister und den Stackpointer setzen oder von dort holen. Diese Befehle werden jedoch hauptsächlich in der Systemprogrammierung eingesetzt.

Ein weiterer, sehr brauchbarer Übertragungsbefehl ist MOVEM, der beliebige Daten- oder Adreßregister aus aufeinanderfolgenden Speicherstellen lesen oder dorthin schreiben kann. Damit lassen sich beispielsweise beim Eintritt in eine Subroutine alle Register, die dort verändert werden, mit einem Befehl sichern und nach dem Rücksprung wieder in ihren ursprünglichen Zustand versetzen:

```
Einsprung MOVEM   D2/D4/A3,PAD
                  (die Subroutine
                  arbeitet mit D2,D4
                  und A3)
Rücksprung MOVEM PAD,D2/D4/A3
```

Beim Sprung auf die Subroutine werden D2,D4 und A3 in PAD gesichert und nach Beendigung wieder zurückgeladen. Sie können die Register aber auch mit dem Stack sichern:

```
Einsprung MOVEM   D2/D4/A3,-(SP)
                  (Code der Subroutine)
Rücksprung MOVEM (SP)+,D2/D4/A3
```

Eine Variation des Befehls MOVE ist die Schnellübertragung (MOVEQ). Da der ganze Befehl in einem Computerwort untergebracht ist, eignet er sich besonders für das Setzen einer vorzeichenbehafteten Acht-Bit-Konstanten (+127 bis −128). MOVEQ initialisiert oft Datenregister als Schleifenzähler. Das Wort

```
MOVEQ #34,D2
```

überträgt die Zahl 34 in D2. Wenn das Q fehlt, nehmen einige Assembler nicht automatisch den schnellen Modus, sondern codieren MOVE #34,D2 in zwei Wörtern.

PEA (oder „effektive Adresse auf den Stack schieben") ist ein weiterer Datenübertragungsbefehl, der sich gut für Stackvorgänge eignet. So schiebt PEA BETTY die Adresse BETTY auf den Stapel und vermindert den Stackpointer.

Schließlich gibt es noch Befehle zum Austauschen von Registern. SWAP vertauscht die 16 unteren Bits eines Registers mit den 16 oberen Bits.

```
SWAP D2
```

tauscht das Computerwort in Bit 0 bis Bit 15 gegen das Wort von Bit 16 bis Bit 31. Der Befehl läßt sich gut für Berechnungen einsetzen, die in den Datenregistern mit Langwörtern arbeiten. Die Prozedur sieht so aus:

- Langwort in D2 laden
- Berechnung mit dem Wort ausführen
- SWAP D2
- Berechnung mit dem Wort ausführen
- SWAP Wort

Mit dem Tauschbefehl EXG lassen sich in bestimmten Kombinationen Langworte im 32-Bit-Format tauschen. Hier einige Beispiele:

```
EXG D2,D3 tauscht Datenregister
EXG A3,A4 tauscht Adreßregister
EXG D2,A5 tauscht Adreß- und Datenregister
```

EXG ist eigentlich ein SWAP mit 32-Bit-Worten.

Diese Befehle bilden die Grundlage aller mathematischen Berechnungen. Sie addieren Binärstellen auf Basisebene. Selbst wenn es in Ihrem Programm keine numerischen Vorgänge gibt, finden einfache arithmetische Vorgänge statt, wenn Zeichencodes umgewandelt oder Arrayindizes angelegt werden.

Der Befehl ADD addiert die Quelle mit dem Ziel und speichert das Ergebnis im Ziel. Das Datenobjekt kann jede mögliche Größe annehmen, es werden alle Bedingungscodes beeinflußt. So addiert ADD.W D2,D3 den Inhalt von D2 mit D3 und speichert das Ergebnis in D3.

Für den Quelloperanden können alle Adressierarten eingesetzt werden. Wenn das Ziel ein Adreßregister sein soll, müssen Sie allerdings mit ADDA arbeiten. ADDA D2, A4 addiert daher den Inhalt von D2 auf A4.

Für unmittelbare Daten gibt es den Spezialbefehl ADDI, der diese Informationen (alle Datenformate sind möglich) als Erweiterungswort im Ziel speichert (hier sind nur datenverändernde Modi erlaubt). ADDI #3423 BETTY addiert 3423 mit dem Inhalt von BETTY.

Wie bei MOVE gibt es auch eine schnelle Version von ADDI — ADDQ. Die Daten dürfen jedoch nur im Bereich von 1 bis 8 liegen. ADDQ 5,D2 addiert 5 auf den Inhalt von D2, wobei der ganze Befehl nur ein Wort belegt.

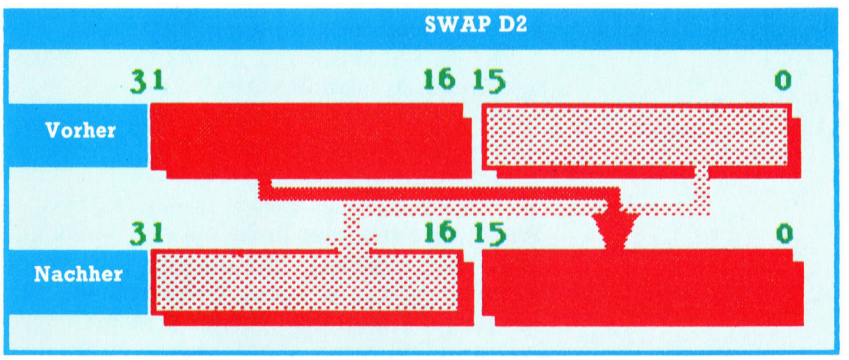
Alle bisher erwähnten Addierbefehle berücksichtigen keine Überträge. Für die Bearbeitung von Überträgen, besonders bei Berechnungen mit mehrfacher Präzision, müssen Sie den Befehl ADDX verwenden. ADDX D2,D4 addiert den Inhalt von D2, D4 und dem Erweiterungsbit in SR und speichert das Ergebnis in D4. Wenn D2 00000000 enthält, D4 00000001 und X auf 1 steht, dann ist der Inhalt von D4 nach Ausführung von ADDX 00000010. Mit ADD erhalten Sie in D4 den Wert 00000001.

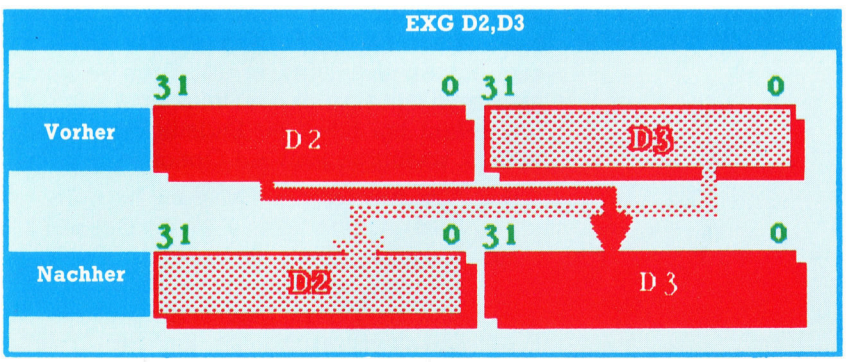
Die SUBtraktionsbefehle entsprechen den ADD-Anweisungen. Sie haben die gleichen Adreßbeschränkungen und setzen auch die gleichen Bits des SR. Die folgenden Beispiele sind alle legal:

```
SUB D2,D3     D2 von D3 subtrahieren und
              das Ergebnis in D3 speichern
SUBA #4,A3    4 von A3 subtrahieren
SUBI #200,D2  2 von D2 subtrahieren
SUBQ #1,D2    1 von D2 schnell subtrahieren
SUBX D2,D4    Speichert das Ergebnis von
              D4–D2–X in D4
```

Beachten Sie, daß der 68000er keinen separaten In- oder Dekrementierbefehl bietet. Sie müssen dafür die Versionen von ADD und SUB einsetzen, die nur ein Wort lang sind.

Die Arbeit mit Langwörtern kann recht unhandlich werden, wenn Sie mit Wort- oder Byteoperanden arbeiten. SWAP erleichtert diese Aufgabe, indem es die Registerwerte von Bit 0 bis Bit 15 gegen die von Bit 16 bis Bit 31 austauscht. Mit EXG lassen sich Langwörter gegeneinander austauschen.





Alle Möglichkeiten

Das Betriebssystem Unix bietet eine enorme Vielfalt an Dienstprogrammen, die in Verbindung mit dem Pipeline-Verfahren weitreichende Möglichkeiten eröffnen.

Unix-Kommandos haben das Format
Befehl-Name Optionen Parameter
Die einzelnen Elemente der Befehlszeile müssen voneinander durch mindestens einen Leerraum getrennt sein. Bei den Parametern handelt es sich im allgemeinen um Datei- oder Directory-Namen. Werden für die Ein- und Ausgabe keine Vorschriften gemacht, spricht das System automatisch die Tastatur ('Standardinput') bzw. den Bildschirm ('Standardoutput') an.

Eine 'Option' besteht aus einem Buchstaben oder Zeichen mit vorgesetztem Bindestrich. Nach einem Bindestrich können auch mehrere Optionen folgen. Für den Befehl 'ls' zum Auflisten eines Directory z. B. gibt es rund ein halbes Dutzend Optionen, darunter 'l' für das Auflisten im Langformat und 'a' für die Anzeige aller Einträge (einschl. Systemdateien). Parameter kann entweder eine Dateispezifikation oder ein Directory-Name sein; fehlt beides, wird das aktuelle Directory ausgegeben.

Das Auflisten des Directory /usr mit den Optionen 'l' und 'a' veranlassen Sie durch
ls –l –a /usr oder ls –la /usr
Falsche Schreibweise löst eine Fehlermeldung mit einem Hinweis auf die korrekte Form aus. Als Trennsymbol zwischen Befehlsgruppen innerhalb einer Zeile dient das Semikolon.

Der Befehl 'wc' zählt die Zeichen, Wörter und Zeilen in einer Textdatei.

–l: nur Zeilenzählung (lines)
–w: nur Wortzählung (words)
–c: nur Zeichenzählung (characters)
–p: Seitenzählung (pages; jeweils 66 Zeilen)

Wenn als Parameter mehr als ein Dateiname folgt, wird zusätzlich die Summe der Einzelwerte angezeigt. Ist keine Datei genannt, wird die Eingabe über die Tastatur erwartet.

Der Befehl 'head' (Kopf) veranlaßt die Ausgabe der Anfangszeilen einer Datei über 'Standardoutput' auf dem Bildschirm. Als einzige Option ist die Festlegung der Zeilenanzahl vorgesehen, etwa in der Form '–15'; danach muß mindestens ein Dateiname kommen. Der Befehl 'tail' filtert das Endstück einer Datei heraus; verfügbar sind die Optionen

+n: Ausgabebeginn n 'Einheiten' nach Dateianfang (hier kein '–' vor der Option!)
–n: Ausgabebeginn n 'Einheiten' vor Dateiende (Standardwert: n=10)
–l: Zeile als 'Einheit' (Standardeinstellung)
–b: Diskette-Blocklänge als 'Einheit'
–c: Zeichen als 'Einheit'
–r: Ausgabe in umgekehrter Reihenfolge

Zum Sortieren und zum Mischen dient das Kommando 'sort' mit den Optionen

–b: Ignorieren vorangehender Leerräume beim Vergleich
–d: Lexikografisches Sortieren (nur Buchstaben, Ziffern, Leerräume beachten)
–f: Groß- und Kleinschreibung gleichwertig
–n: Zahlen nicht ziffernweise, sondern stellenwertrichtig sortieren
–o: Ausgabe an Datei statt an 'Standardoutput'
–r: Sortieren in umgekehrter Reihenfolge

Als Parameter können ein oder mehrere Dateinamen aufgeführt sein; entsprechend wird entweder nur in einer Datei sortiert oder aber im ganzen, was eine Vermischung der betroffenen Dateien bedeutet.

Der Befehl 'cmp' (compare) veranlaßt das zeilenweise Vergleichen zweier Dateien und bewirkt in der Grundform nur die Anzeige von Byte- und Zeilennummer der ersten gefundenen Abweichung. Die einzige vorgesehene Option 'l' führt zur Ausgabe sämtlicher Differenzen zwischen den beiden Dateien. Ist als Parameter ein File allein genannt, wird die Vergleichsdatei über 'Standardinput' erwartet.

Die Anweisung 'comm' (common) sucht nach Gemeinsamkeiten zwischen zwei nach ASCII sortierten Dateien und gibt in getrennten Spalten (1) die Zeilen aus, die nur in der ersten, (2) solche, die nur in der zweiten und (3) alle, die in beiden Dateien vorkommen. Durch die Optionen 1, 2 und 3 lassen sich einzelne Spalten unterdrücken; in jedem Fall müssen zwei Dateinamen als Parameter folgen.

Auch die Routine 'diff' forscht zeilenweise nach Differenzen zwischen zwei Dateien und zeigt an, was zum Angleichen mit der ersten geschehen muß: für Anfügen erscheint ein 'a', für Ändern (change) ein 'c' und für Löschen (delete) ein 'd'. Danach folgen die nicht übereinstimmenden Zeilen mit vorgesetztem '<' für die erste und '>' für die zweite Datei. Zulässige Optionen:

–b: Ignorieren nachstehender Leerräume und der Länge von größeren Lücken
–e: Ausgabe einer Editor-Kommandodatei zur Korrektur der ersten Datei
–r: Nur bei Directories — erlaubt die rekursive Anwendung von 'diff' auf Subdirektories

Als Parameter kommt ein Paar von Datei- oder Directorynamen in Frage. Im Fall von Directories führt ‚diff' alle Dateien auf, die nur in einem Verzeichnis stehen, und danach diejenigen, die in beiden zu finden sind.

Der Befehl ‚uniq' bewirkt, daß in einer Textdatei benachbarte sich wiederholende Zeilen bis auf eine gelöscht werden. Die Optionen sind:

–u: Ausgabe nur einfach vorhandener Zeilen
–d: Ausgabe aller Mehrfachzeilen (je einmal)
–c: Anzeige der Wiederholungshäufigkeit in der Datei bei jeder Zeile

Das Kommando ‚lpr' übergibt eine oder mehrere Dateien an den Drucker. Da das Drucken bei Mehrprogrammbetrieb nicht direkt erfolgen kann, stellt das System die Daten in eine Warteschlange und arbeitet sie mit Hilfe eines ‚Spoolers' ab. Außer den Dateinamen als Parameter lassen sich in Abhängigkeit von der jeweiligen Anlage diverse Optionen angeben.

Der Befehl ‚lpq' teilt Einzelheiten über den aktuellen Stand der Drucker-Warteschlange (lineprinter-queue) mit, d. h. ob eine bestimmte Datei schon gedruckt ist. Jeder Druckauftrag in der Schlange läuft unter einer Nummer. Parameter oder Optionen sind bei diesem Befehl nicht vorgesehen.

Mit ‚lprm' (lineprinter-remove) läßt sich eine Datei aus der Druckerschlange entfernen, wenn als Parameter der Dateiname oder die über ‚lpq' mitgeteilte Jobnummer folgt. Wird statt dessen die Benutzerkennung (wie bei ‚login') angegeben, bewirkt das die Annullierung sämtlicher Druckaufträge des Benutzers.

Das Kommando ‚pr' dient zur Wiedergabe von Textfiles im Druckformat über den ‚Standardoutput'. Der umbrochene Text erscheint seitenweise, mit fünf Leerzeilen unten sowie dem Kopf aus Datum, Dateiname und Seitenzahl und zwei Leerzeilen. Als Optionen sind vorgesehen

–n: Ausgabe in n-spaltigem Satz
–m: Ausgabe mehrerer Dateien nebeneinander
–t: Unterdrücken von Kopf und Rand

Das untenstehende Dialogbeispiel soll anhand von drei Textdateien erläutern, wie vielseitig und effizient Sie mit diesen Unix-Kommandos arbeiten können.

Dateibearbeitung

%cat file1
Die Katze saß auf der Matte.
Mary hatte ein kleines Lamm.
Der flinke braune Fuchs springt über den faulen Hund.
Die Eule und die Katze zogen ans Wasser.

%cat file2
Die Katze saß auf dem Hund.
Mary hatte ein kleines Lamm.
Der flinke braune Fuchs springt über den faulen Hund.
Die Eule und die Katze zogen ans Wasser.

%cat file3
Die Katze saß auf der Matte.
Mary hatte ein kleines Lamm.
Der flinke braune Fuchs springt über den faulen Hund.
Diese Datei hat eine Zeile mehr.
Die Eule und die Katze zogen ans Wasser.

%wc file1 *(Zeilen, Wörter und Zeichen in file1)*
4 28 153 file1

%wc –w file1 *(nur Wörter zählen)*
28 file1

%head –2 file1 *(nur die ersten beiden Zeilen ausgeben)*
Die Katze saß auf der Matte.
Mary hatte ein kleines Lamm.

%tail +2 file1 *(von der zweiten Zeile an ausgeben)*
Mary hatte ein kleines Lamm.
Der flinke braune Fuchs springt über den faulen Hund.
Die Eule und die Katze zogen ans Wasser.

%tail –2 file1 *(die letzten beiden Zeilen ausgeben)*
Der flinke braune Fuchs springt über den faulen Hund.
Die Eule und die Katze zogen ans Wasser.

%sort file1 *(Zeilen alphabetisch sortieren)*
Der flinke braune Fuchs springt über den faulen Hund.
Die Eule und die Katze zogen ans Wasser.
Die Katze saß auf der Matte.
Mary hatte ein kleines Lamm.

%pr file3 *(Druckformatierung von file3)*
Die Katze saß auf der Matte.
Mary hatte ein kleines Lamm.
Der flinke braune Fuchs springt über den faulen Hund.
Diese Datei hat eine Zeile mehr.
Die Eule und die Katze zogen ans Wasser.

%sort file1 > file4 *(file1 und file2 sortieren und in neu*
%sort file2 > file5 *erzeugten Dateien ablegen)*

%cmp file1 file2 *(Vergleich zweier Dateien)*
file1 file2 differ char 23, line 1

%comm file4 file5
Mary hatte ein kleines Lamm.
Die Katze saß auf dem Hund.
Die Katze saß auf der Matte.
Die Eule und die Katze zogen ans Wasser.
Der flinke braune Fuchs springt über den faulen Hund.
(nur in file4/nur in file5/in beiden Dateien)

%diff file1 file2 *(Unterschiede zwischen den Dateien)*
1c1 *(‚c' bedeutet: Änderung erforderlich)*
<Die Katze saß auf der Matte.

>Die Katze saß auf dem Hund.

%sort file2 file3 > file7 (alphabetisches Mischen mit Ablegen des Ergebnisses in file7)
%cat file7
Der flinke braune Fuchs springt über den faulen Hund.
Der flinke braune Fuchs springt über den faulen Hund.
Die Eule und die Katze zogen ans Wasser.
Die Eule und die Katze zogen ans Wasser.
Die Katze saß auf dem Hund.
Die Katze saß auf der Matte.
Diese Datei hat eine Zeile mehr.
Mary hatte ein kleines Lamm.
Mary hatte ein kleines Lamm.

%uniq file7 *(Ausgabe ohne Wiederholungen)*
Der flinke braune Fuchs springt über den faulen Hund.
Die Eule und die Katze zogen ans Wasser.
Die Katze saß auf dem Hund.
Die Katze saß auf der Matte.
Diese Datei hat eine Zeile mehr.
Mary hatte ein kleines Lamm.

Richtig-stellung

In der letzten Folge unseres Spreadsheet-Programmierprojektes besprachen wir die Routinen zum Konvertieren der Zellinhalte in die umgekehrte polnische Notation. Diesmal entwickeln wir eine einfache Methode zum Prüfen der Eingaben auf Form- und Schreibfehler.

Wir stellten Ihnen bereits eine Methode zum Schreiben der Kalkulationsformeln vor, bekannt als die umgekehrte polnische Notation (UPN). Die Konvertierung von Formelausdrücken normaler Notation in UPN bringt zwei entscheidende Vorteile: Ein Ausdruck in UPN ist einfacher auf korrekte Schreibweise zu prüfen und, nach dieser Prüfung, viel einfacher zu berechnen.

Die meisten BASIC-Programmierer kennen die Fehlermeldung „SYNTAX ERROR", die immer wieder aus den verschiedensten Gründen erscheint. In vielen Fällen liegt ein simpler Tippfehler vor, oder wichtige Angaben wurden ausgelassen. Die gleichen Fehler können dem Anwender bei der Dateneingabe in ein Kalkulationsblatt passieren. Man kann versehentlich eine Klammer auslassen, oder die Anzahl der Operanden ist geringer als die der Operatoren. Es gibt eine simple Methode zum Prüfen eines Ausdrucks in UPN, die sicherstellt, daß die richtigen Komponenten in korrekter Reihenfolge vorhanden sind – also eine ordnungsgemäße Formel vorliegt.

Die Prüfung einer Formel auf etwaige Fehler ist eine maßgeschneiderte Aufgabe für einen Computer. Den Elementen, die zu einer Formel gehören, werden folgende Codezahlen zugeordnet:

- Binäre Operatoren (z. B. +,–) Codezahl –1
- Vergleichsoperatoren (z. B. &) Codezahl 0
- Operanden (z. B. A2,27) Codezahl 1

Anschließend arbeitet das Programm den UPN-Ausdruck Element für Element von rechts nach links ab und addiert dabei die Codezahlen zu einer Kontrollsumme. Wenn jede Zwischensumme kleiner oder gleich Null und die Endsumme gleich Eins ist, dann ist der Ausdruck fehlerfrei.

Nach Umwandlung der normal geschriebenen Formel (I$) in die umgekehrte polnische Notation (P$) prüft das Unterprogramm ab Zeile 4700, ob der Ausdruck in P$ Schreibfehler enthält. Die Routine arbeitet von links beginnend jedes Element in P$ ab und schiebt die jeweiligen Codezahlen (–1, 0, 1) auf einen Stapelspeicher, ST(). Nach Erfassen aller Elemente wertet das Programm ab Zeile 4800 den Stapel aus. Jede Codezahl wird vom Stapel ST() genommen und zur Kontrollvariablen P addiert. Falls während dieses Vorgangs der Wert in P Null überschreitet, bricht die Routine mit „ERROR BEFORE END" ab. Nach Auswertung aller Codezahlen sollte die Kontrollsumme gleich Eins sein. Ein anderer Wert führt zur Fehlermeldung „ERROR AT END".

So funktioniert die Fehlerprüfung

Zum Verdeutlichen des Prüfvorgangs nehmen wir ein einfaches Beispiel. Angenommen, wir beginnen mit dieser Formel in einer Zelle:

(A2+A3)*(B2–B3)

dann lautet der UPN-Ausdruck:

A2A3+B2B3–*

Die folgende Übersicht zeigt, wie die Formel von rechts beginnend abgesucht und die Codezahlen der Elemente zu einer Zwischensumme erfaßt werden:

Element	Codezahl	Zwischensumme
*	–1	–1
–	–1	–2
B3	1	–1
B2	1	0
+	–1	–1
A3	1	0
A2	1	1

Beim Absuchen eines Ausdrucks in umgekehrter polnischer Notation von rechts nach links sollte der Operator immer vor den dazugehörigen Operanden ermittelt werden. Daher müßte die Zwischensumme entweder negativ oder, nach Bearbeitung der beiden Operanden, gleich Null sein. Die Ausnahme hiervon ist die Endsumme – sie muß Eins sein –, weil in jedem Ausdruck ein Operand mehr als Operatoren vorhanden sein sollte.

Nehmen wir an, der letzte Operator, das Multiplikationszeichen, fehlt. Nach unserer Methode finden wir den Fehler schnell:

Element	Codezahl	Zwischensumme
–	–1	–1
B3	1	0
B2	1	1**ERROR**
+		
A3		
A2		

Da als Operanden in einer Formel auch Zellnamen gestattet sind, wie A2 oder L14, ist es wichtig, diese vor der Fehlerprüfung separat abzulegen. Weil solche Operanden in einem Ausdruck normaler Notation viel einfacher identifizierbar sind, speichert die Unterroutine bei Zeile 4900 die Namen der in I$ gefundenen Operanden in E$().

Testlauf

Durch eine kleine Änderung des Programms können Sie die Routinen zum Konvertieren in die umgekehrte polnische Notation und die anschließende Prüfung bereits in diesem Stadium testen. Geben Sie das abgedruckte Listing ein und ändern Sie Zeile 4010 wie folgt:

4010 LET I$="Ihre Formel in normaler Notation"

Danach geben Sie diese Befehlszeile im direkten Modus ein:

GOSUB 3000:GOSUB 4000:PRINT P$

Sinclair Spectrum:

```
4700>REM ************************
4701 REM * CHECK WELL FORMED    *
4702 REM * REVERSE POLISH       *
4703 REM ************************
4705 GO SUB 4900
4710 LET P=0: LET SP=1: LET L1=1
: LET U$=""
4720 LET P=P+1: IF P>LEN (P$) TH
EN  GO TO 4800
4730 LET T$=P$(P)
4740 IF T$="+" OR T$="-" OR T$="
*" THEN  LET S(SP)=-1: LET G$(SP
)=T$: LET SP=SP+1: GO TO 4720
4745 IF T$="/" OR T$="^" THEN  L
ET S(SP)=-1: LET G$(SP)=T$: LET
SP=SP+1: GO TO 4720
4747 IF L1=LP THEN  GO TO 4720
4750 LET U$=E$(L1)
4751 FOR Z=1 TO LEN (U$)
4752 IF U$(Z)=" " THEN  GO TO 47
54
4753 NEXT Z
4754 LET U$=U$(1 TO Z-1)
4760 IF U$<>P$(P TO P-1+LEN (U$)
) THEN  LET U$="": GO TO 4720
4770 LET S(SP)=1: LET G$(SP)=U$
4775 LET SP=SP+1
4780 IF L1<LP THEN  LET L1=L1+1
4785 LET U$=""
4790 GO TO 4720
4800 REM *** PROCESS STACK *****
4810 LET P=S(SP)
4820 FOR C=SP-1 TO 1 STEP -1
4830 LET P=P+S(C)
4840 IF P>0 AND C<>1 THEN  PRINT
 AT 20,0;"ERROR BEFORE END": RET
URN
4850 NEXT C
4860 IF P<>1 THEN  PRINT "ERROR
AT END  ": RETURN
4870 LET CP=SP-1: RETURN
4900 REM *** MAKE LIST OF OPREAN
DS** ***
4905 LET P=0: LET LP=1: LET T$="
": LET U$=""
4910 LET P=P+1: IF P>LEN (I$) TH
EN  GO TO 4995
4920 LET T$=I$(P)
4930 IF T$="+" OR T$="-" OR T$="
*" OR T$="&" THEN  GO TO 4960
4940 IF T$="/" OR T$="^" OR T$="
(" OR T$=")" THEN  GO TO 4960
4950 LET U$=U$+T$: GO TO 4910
4960 IF U$="" THEN  GO TO 4910
4970 LET E$(LP)=U$: LET LP=LP+1
4980 LET U$=""
4990 GO TO 4910
4995 IF U$="" THEN  RETURN
4997 LET E$(LP)=U$: LET LP=LP+1
4998 RETURN
```

BASIC-Dialekte

Acorn B:
Beachten Sie folgende Änderungen gegenüber der Commodore-64-Version:

```
4840 If P>1 AND C<>1 THEN PRINT
   TAB(0,22);"ERROR BEFORE END
         ":RETURN
4860 IF P<>1 THEN PRINT TAB(0,22);"
   ERROR AT END        ":RETURN
```

Schneider CPC:
Beachten Sie folgende Änderungen gegenüber der Commodore-64-Version

```
4840 IF P>1 AND C<>1 THEN LOCATE
   1,22:PRINT"         ERROR BEFORE
   END        ": RETURN
4860 IF P<>1 THEN LOCATE 1,22:PRINT"
   ERROR AT END        ":RETURN
```

Commodore 64:

```
4700 REM ******************************
4701 REM **** CHECK FOR WELL FORMED ***
4702 REM *** REVERSE POLISH STRING ***
4703 REM ******************************
4705 GOSUB 4900
4710 P=0:SP=1:L1=1:TE$=""
4720 P=P+1:IF P>LEN(P$) THEN 4800
4730 T$=MID$(P$,P,1)
4740 IF T$="+" OR T$="-" OR T$="*" THEN
ST(SP)=-1:G$(SP)=T$:SP=SP+1:GOTO 4720
4745 IF T$="/" OR T$="^" THEN ST(SP)=-1:
G$(SP)=T$:SP=SP+1:GOTO 4720
4746 IF T$="&" THEN ST(SP)=0:G$(SP)=T$:S
P=SP+1:GOTO 4720
4747 IF L1=LP THEN 4720
4750 LET TE$=E$(L1)
4760 IF TE$<>MID$(P$,P,LEN(TE$)) THEN TE
$="":GOTO 4720
4770 LET ST(SP)=1:LET G$(SP)=TE$
4775 LET SP=SP+1
4780 LET L1=L1-(L1<LP):TE$=""
4790 GOTO 4720
4800 REM *** PROCESS STACK ***
4810 LET P=ST(SP)
4820 FOR C=SP-1 TO 1 STEP -1
4830 LET P=P+ST(C)
4840 IF P>0 AND C<>1 THEN GOSUB 1950:PRI
NT "    ERROR BEFORE END    ":RETURN
4850 NEXT C
4860 IF P<>1 THEN GOSUB 1950:PRINT "
  ERROR AT END        ":RETURN
4870 LET CP=SP-1:RETURN
4900 REM ** MAKE LIST OPERANDS IN I$ **
4905 LET P=0:LET LP=1:LET T$="":LET TE$=
""
4910 FOR K=1 TO 20:LET E$(K)="":NEXT K
4915 LET P=P+1:IF P>LEN(I$) THEN 4995
4920 LET T$=MID$(I$,P,1)
4930 IF T$="+" OR T$="-" OR T$="*" OR T$
="&" THEN 4960
4940 IF T$="/" OR T$="^" OR T$="(" OR T$
=")" THEN 4960
4950 LET TE$=TE$+T$:GOTO 4915
4960 IF TE$="" THEN 4915
4970 LET E$(LP)=TE$:LP=LP+1
4980 LET TE$=""
4990 GOTO 4915
4995 IF TE$="" THEN RETURN
4996 LET E$(LP)=TE$:LET LP=LP+1
4997 RETURN
```

Bildschirmtricks

Das Schneider-Betriebssystem unterstützt eine Textanzeige, die völlig von der Grafikdarstellung getrennt ist. Wir untersuchen, welche Bildschirmsteuerungen interessant sind und zeigen drei Module, die sich in eigene Programme einbauen lassen.

Nützliche Adressen

Routine	Adresse	Anmerkung
TXT_WIN_ENABLE	BB66	H = links, D = rechts, L = oben, E = unten
TXT_GET_WINDOW	BB69	Ergebnis in den Registern wie oben
TXT_CLEAR_WINDOW	BB6C	Löscht Textfenster des aktuellen Stroms
TXT_SWAP_STREAMS	BBB7	In B und C stehen beide Stromnummern
TXT_STR_SELECT	BBB4	Angewählte Ströme in Register A
TXT_OUTPUT	BB5A	Zeichen steht in Register A
TXT_WR_CHAR	BB5D	Zeichen steht in Register A
TXT_RD_CHAR	BB60	Liest beim Cursor Zeichen und speichert es in Reg. A
TXT_GET_CURSOR	BB78	H = Spalte, L = Zeile
GRA_SET_ORIGIN	BBC9	Aufruf mit DE = X, HL = Y (Koordinaten)
GRA_SET_WIDTH	BBCF	Aufruf mit DE = links, HL = rechts (Koordinaten)
GRA_SET_HEIGHT	BBD2	Aufruf mit DE = oben, HL = unten (Koordinaten)
SCR_SET_BASE	BCO8	In A steht Hi-Byte der Basisadresse
SCR_SET_MODE	BCOE	In A steht Modusnummer (0, 1 und 2)

Wichtig: Für den Einsatz dieser Routinen sollten Sie sich unbedingt die „Amstrad Firmware Specification" besorgen.

Die Textanzeige steuert unter anderem das Lesen und Anzeigen von Zeichen, den Cursor und Fenster. Die Firmware von Schneider kann bis zu acht Fenster auf den Schirm bringen, jedes mit eigenen PEN- und PAPER-Farben, eigenem Cursor, transparentem oder undurchsichtigem Hintergrund und der Möglichkeit, Grafikzeichen darzustellen. Die Größe des Fensters wird mit TXT_WIN_ENABLE gesetzt, mit TXT_GET_WINDOW abgefragt und über TXT_CLEAR_WINDOW gelöscht.

Sie können die Parameter zweier „Ströme" mit TXT_SWAP_STREAMS auch gegeneinander austauschen. Die meisten Firmwareroutinen sprechen den jeweils aktiven Strom an, wobei TXT_STR_SELECT festlegt, welcher Strom aktiv ist.

Jedes Zeichen wird in einem Pixelquadrat von acht mal acht Pixeln ausgegeben. Wir haben die wichtigsten Anzeigeroutinen in der nebenstehenden Tabelle aufgeführt.

Die Bildschirmzeichen grenzen normalerweise aneinander und erscheinen an der aktuellen Cursorposition. Über einen Spezialmodus (Anwahl mit TXT_SET_GRAPHIC) können sie aber auch an der Position des Grafikcursors geplottet werden. Damit lassen sich Texte überlagern und an beliebigen Bildschirmpositionen darstellen.

Die Hauptroutine, TXT_OUTPUT, sichert zunächst alle Register und ruft dann TXT_OUT_ACTION auf. Die Routine, auf die dieses Modul zeigt, filtert die Steuerzeichen aus, bevor die darstellbaren Zeichen an TXT_WR_CHAR (bei grafischer Textausgabe GRA_WR_CHAR) übergeben werden. Da TXT_OUT_ACTION gepatcht werden kann, lassen sich auch andere Interpretationen der Steuercodes oder der Zeichendarstellung einsetzen.

Aktuelle Fenster

Beim Textbildschirm zeigt der Cursor auf die Position, an der das nächste Zeichen erscheinen soll. Einige Routinen können den Textcursor abschalten und andere Cursor an beliebige Bildschirmpositionen stellen oder löschen. Die wichtigsten dieser Routinen stehen ebenfalls in der Tabelle.

Mit dem Tabulatorprogramm können Sie Tabulatoren in gleichmäßigen Abständen setzen. Dabei wird zunächst mit TXT_GET_WINDOW die Größe des aktuellen Fensters festgestellt. Die Nummer der rechts außen liegenden Spalte befindet sich danach im Register D. Das Register wird umgewandelt und zeigt nun auf eine Position rechts dieser Spalte. Als nächstes wird mit TXT_GET_CURSOR die aktuelle Cursorposition geholt. Register H enthält nun die

Cursorspalte, die dann mit jeder möglichen Tabulatorposition verglichen wird, bis die nächste Tabulatoreinstellung gefunden ist. Nachdem diese Position bekannt ist, stellt ein Test sicher, daß das abzubildende Zeichenfeld auch über genug Platz verfügt. Ist dies der Fall, wird die Cursorspalte auf die nächste Tabulatoreinstellung gesetzt, wenn nicht, wird durch die Angabe einer Spalte außerhalb des aktuellen Fensters veranlaßt, daß der Cursor sich auf die darunterliegende Zeile bewegt.

Pixelweise

Jedem ASCII-Code von 0 bis 255 ist eine Matrix mit dem Bitmuster des jeweiligen Zeichens zugeordnet. Die Matrix besteht aus acht Bytes, wobei jedes Byte eine Zeile des Zeichens darstellt. Das Bild zeigt, wie die Zeichenmatrix für CHR$224 gespeichert ist.

Die Matrizen können über vier Firmwareroutinen festgelegt oder umdefiniert werden. Da alle Matrizen für die ASCII-Zeichen zwischen 0 und 247 standardmäßig im unteren ROM-Bereich untergebracht sind, lassen sie sich nicht beliebig umdefinieren. Die Zeichen 248 bis 255 werden automatisch ins RAM kopiert, um dort umgestellt zu werden. Mit der Routine TXT_SET_M_TABLE können Sie jedoch eine beliebige Anzahl Zeichen zwischen dem angegebenen Code und 255 umdefinieren und so einen völlig neuen Zeichensatz entwerfen.

Die Matrizen der umdefinierbaren Zeichen können Sie entweder mit einem Anwenderprogramm verändern oder über die Routine TXT_SET_MATRIX, die die acht Bytes, auf die das Registerpaar HL zeigt, in eine dem jeweiligen Zeichen entsprechende Matrix kopiert.

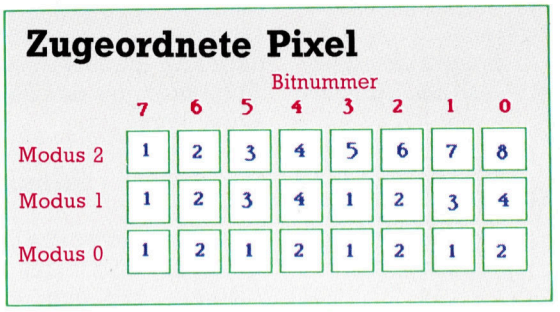
Mit TXT_GET_MATRIX können Sie die Adresse einer Zeichenmatrix feststellen. TXT_GET_M_TABLE holt das erste Zeichen und die Adresse einer Zeichentabelle.

Unser „Spin Print Programm" arbeitet mit TXT_GET_MATRIX und TXT_SET_MATRIX, um Zeichen in eine von vier Richtungen zu rotieren. Dabei wird Zeichen 248 auf die Matrix des darzustellenden Zeichens umdefiniert, dann die Matrix von Zeichen 248 rotiert und schließlich mit TXT_OUTPUT dargestellt.

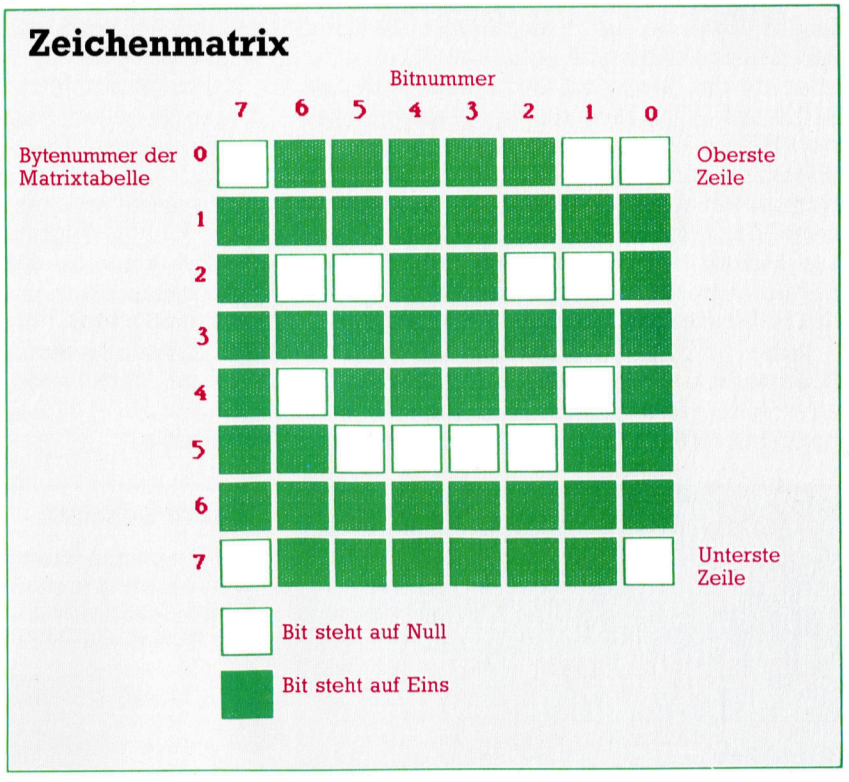
Der Grafikbildschirm läßt sich pixelweise steuern. Er arbeitet mit einem Grafikfenster, in dem die Vorder- und Hintergrundfarben unabhängig von denen des Textschirms gesetzt werden können.

Es gibt drei Koordinatensysteme, mit denen sich Pixelpositionen festlegen lassen. In Modus 2 beträgt die Auflösung 640 mal 200 Punkte, in Modus 1 320 mal 200 und in Modus 0 160 mal 200. Diese Daten werden „Basiskoordinaten" genannt. Jeder Bildschirmpunkt hat eine eindeutige Adresse, über die er unabhängig vom aktuellen Modus angesprochen werden kann. Die Modi bestimmen dabei nur, wieviel Platz ein bestimmtes Pixel beim Setzen eines Punktes belegt. Modus 2 setzt ein Pixel, Modus 1 zwei und Modus 0 vier.

Obwohl der Bildschirm vertikal nur eine Auflösung von 200 Punkten besitzt, arbeitet die Firmware mit 400 Punkten, um das Verhältnis 2 : 1 beizubehalten. Jeder vertikale Punkt bezieht sich daher nur auf ein halbes Pixel, wobei



Die Bytes des Bildschirmspeichers enthalten je nach Anzeigemodus die Daten für bis zu acht Pixel. Das Bild zeigt, welche Bits sich in den unterschiedlichen Modi auf welche Pixel beziehen.



Zeichen werden in Acht-Bit-Matrizen gespeichert, in denen die einzelnen Bytes je eine Pixelzeile des Zeichens darstellen. Gesetzte Bits zeigen an, daß ein Pixel in der aktuellen PEN-Farbe dargestellt werden soll. Auf Null stehende Bits geben die PAPER-(Hintergrund-) Farbe an. Unser Bild zeigt die Matrix für das ASCII-Zeichen 244.

die zwei Pixelhälften gleichzeitig gesetzt werden. So beziehen sich beispielsweise die beiden Pixelzeilen 10 und 11 auf die gleiche Bildschirmzeile. Durch dieses Verfahren hat jeder Bildschirmmodus eine Auflösung von 640 mal 400 Pixeln.

Die Fenstergröße wird mit GRA_SET_ORIGIN, GRA_SET_WIDTH und GRA_SET_HEIGHT gesetzt und mit GRA_GET_ORIGIN, GRA_GET_W_WIDTH und GRA_GET_W_HEIGHT abgefragt. Nach der Initialisierung eines Fensters beziehen sich die sogenannten „Anwender-Koordinaten" auf die neuen Daten. Der Grafikcursor läßt sich mit zwei Methoden positionieren: Beim „absoluten" Verfahren wird der Cursor auf die angegebene Anwenderkoordinate gesetzt, während die „relative" Methode den Cursor relativ zu seiner aktuellen

Position verschiebt. Die Tabelle zeigt die Positioniermethode.

Auf dem Grafikschirm lassen sich Pixel auf drei Arten plotten; einzeln, als Zeichen oder als Linien. Alle Plotroutinen können absolute Punkte oder Punkte relativ zur aktuellen Cursorposition anzeigen. Der Grafikcursor steht immer links von dem zuletzt geplotteten Punkt – außer wenn beim Plotten eines Zeichens die horizontale Position auf den Anfang des nächsten Zeichens verschoben und die vertikale Position beibehalten wird.

Das „Bildschirmpaket" ist ein Firmwarebereich, der maschinennahen Zugang zum Bildschirm bietet. Er steuert die Bildschirmadressierung, das Setzen von Farben und Modi, die Codierung und Decodierung, das Scrollen und vieles andere.

Der Bildschirmspeicher ist ein RAM-Block von 16 KByte, der auf jeder der vier 16K-Grenzen des Speichers anfangen kann. Sie sollten aber nur die Blöcke mit den Anfangsadressen $4000 und $C000 nehmen, da die anderen beiden Blöcke Firmwarebereiche enthalten und überschrieben werden könnten. Der Bildschirm kann mit SCR_SET_BASE zwischen zwei Speichern umgeschaltet werden. Mit dieser Technik läßt sich ein Bildschirminhalt anzeigen, während der andere vorbereitet wird. Durch Umschalten wechselt die Anzeige.

Jedes für den Bildschirm eingesetzte Speicherbyte enthält die Informationen für acht Anzeigepositionen. Dies mag je nach dem (mit SCR_SET_MODE gesetzten) Modus zwei, vier oder acht Pixeln entsprechen.

Das Bild mit den Pixelzuordnungen zeigt, wie die Bits eines Bildschirmbytes in verschiedenen Modi die Pixel festlegen. Zwar sieht das System auf den ersten Blick kompliziert aus, doch erscheinen durch die Rotation der Bytes automatisch die Bits für das nächste Pixel. Für das Setzen eines Pixels in einem Byte brauchen Sie nur über eine Maske das links außen liegende Bit zu setzen und das Byte so lange zu rotieren, bis das Bit an der gewünschten Position steht.

Maskenhaft

Den schnellsten Bildschirmzugriff haben Sie, wenn Sie den Bildschirmspeicher direkt ansprechen und nicht mit dem Text- oder Grafikschirm arbeiten. Hier bietet das Bildschirmpaket etliche Routinen, die diese Aufgabe erleichtern.

Unser „Schnellplot-Programm" zeigt, wie mit diesen Routinen Punkte auf den Schirm gebracht werden. Das Programm darf nicht mit einem Grafikschirm eingesetzt werden, der mit Fenstern arbeitet.

Die Routine fragt zunächst nach der Adresse des Bytes mit den Daten für die angegebene Bildschirmposition und dann nach der Maske, mit der alle Pixel eines Bytes mit der angegebenen Farbe definiert werden. Die Bearbeitung der Maske setzt die gewünschte Position zunächst auf INK Null und dann auf die neue INK-Farbe.

Plot mit 128K

Der PLOT-Befehl des Schneider 6128 wurde gegenüber dem CPC 464 etwas verfeinert:

Modus	Aufgabe
0 Normal	– setzt die Pixel auf die neue INK-Farbe
1 XOR	– setzt die Pixel auf die neue oder (XOR) aktuelle Farbe
2 AND	– setzt die Pixel auf die neue und (AND) die aktuelle Farbe
3 OR	– setzt die Pixel auf die neue oder (OR) die aktuelle Farbe

Besonders der XOR-Modus (exklusives OR) ist interessant, da Sie damit Linien plotten und löschen können, ohne die Hintergrundfarbe zu beeinflussen. Bei einer roten Hintergrundfarbe (PAPER 3) erzeugt das Plotten einer Linie (PEN 4) eine Linie in Magenta (PEN 7). Zum Löschen brauchen Sie die Linie nur ein zweitesmal mit PEN 4 zu plotten.

Diese Methode eignet sich besonders für die Sprite-Steuerung. Auf dem 6128 werden die Plot-Modi durch einen zusätzlichen Parameter des PLOT-Befehls angegeben. Diese Möglichkeit gibt es im BASIC 1.0 des CPC 464 noch nicht. Sie brauchen dort nur die Firmwareroutine bei &BC59 mit der Modusnummer im Register A aufzurufen.

Tabulatoren

Das Programm bewegt den Textcursor des aktuellen Fensters in die nächste Tabulatorenspalte. Es gibt keine Einsprungsbedingungen. Die Routine sichert alle Register.

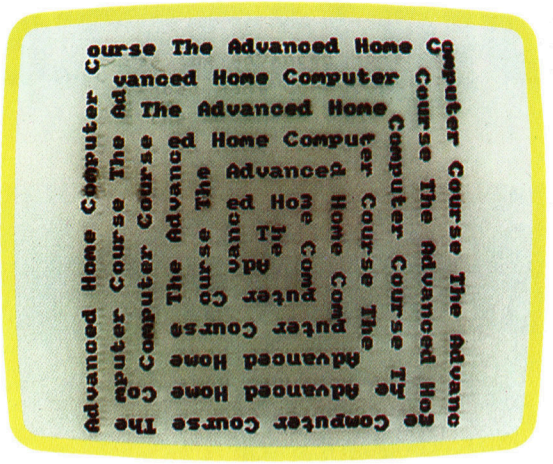
```
tab:            equ     15      ; tab stop column
get.window:     equ     #bb69   ; TXT_GET_WINDOW
get.cursor:     equ     #bb78   ; TXT_GET_CUSOR
set.column:     equ     #bb6f   ; TXT_SET_COLUMN
;
        push    af
        push    hl
        push    de
        call    get.window      ; get window size
        inc     d               ; logical rt col
        inc     d               ; right col limit
        call    get.cursor      ; get cursor posn
        ld      a, 1            ; start of line
;
loop:   add     tab             ; get next tab col
        cp      h
        jp      p, loop
;
        ld      h, a            ; save it
        add     tab             ; room for new col?
        cp      d               ; test for limit
        jr      nc, setcol      ; no,force newline
        ld      a, h            ; get back new col
setcol:
        call    set.column      ; set new column
        pop     de
        pop     hl
        pop     af
        ret
```

Spin Print

Das folgende Programm gibt ein Zeichen aus, dessen Zeichenmatrix um einen von vier Winkeln rotiert wurde. Dafür wird zunächst die Adresse der Zeichenmatrix festgestellt und Zeichen 248 (normalerweise der erste Eintrag in der Tabelle der anwenderdefinierten Zeichen) umdefiniert.

```
;
; Entry:        IX+0 = Character to display
;               IX+2 = Desired angle 1..4
;                    1 = upwards
;                    2 = backwards
;                    3 = downwards
;                    4 = normal
; Exit:
;               Carry True  - Character printed
;               Carry False - Failed to print char
;               AF, BC, DE, HL corrupt
;
txt.get.matrix: equ     #bba5
txt.set.matrix: equ     #bba8
txt.output:     equ     #bb5a
;
        org     8000h
;
start:  ld      a, (ix+0)       ; read character
        call    txt.get.matrix  ; get matrix addr
;
; Copy matrix into user definable character 128
; then find its address
;
        ld      a, 248          ; char to redefine
        call    txt.set.matrix  ; redefine matrix
        ret     nc              ; abort if unable
;
        ld      a, 248          ; char to rotate
        call    txt.get.matrix  ; find matrix
;
; Rotate the character matrix anti-clockwise
; in 90 degrees steps
;
        ld      (toprow), hl    ; save address
        ld      b, (ix+2)       ; read angle
;
lop3:   ld      d, 8            ; loop for 8 rows
;
lop2:   ld      e, 8            ; loop for 8 bits
;
lop1:   rlc     (hl)            ; next bit in c
        rla                     ; save in new row
        inc     hl              ; go to next row
        dec     e
        jr      nz, lop1        ; go for next row
;
        push    af              ; stack new row
        dec     d
        ld      hl, (toprow)    ; go for next bit
        jr      nz, lop2
;
; loop to unstack the new matrix
;
        ld      e, 8
lop4:   pop     af              ; get next row
        ld      (hl), a         ; move to matrix
        inc     h               ; go to next row
        dec     e
        jr      nz, lop4
;
        ld      hl, (toprow)
        djnz    lop3            ; again if needed
;
; Finally display the character on screen
;
        ld      a, 248          ; char to print
        call    txt.output
        ret                     ; finished
;
toprow: defw    0               ; start of matrix
```

Text – einmal andersherum
Mit unserem Programm zur Zeichenrotation können Sie Text vertikal, horizontal oder auch auf dem Kopf stehend ausgeben.

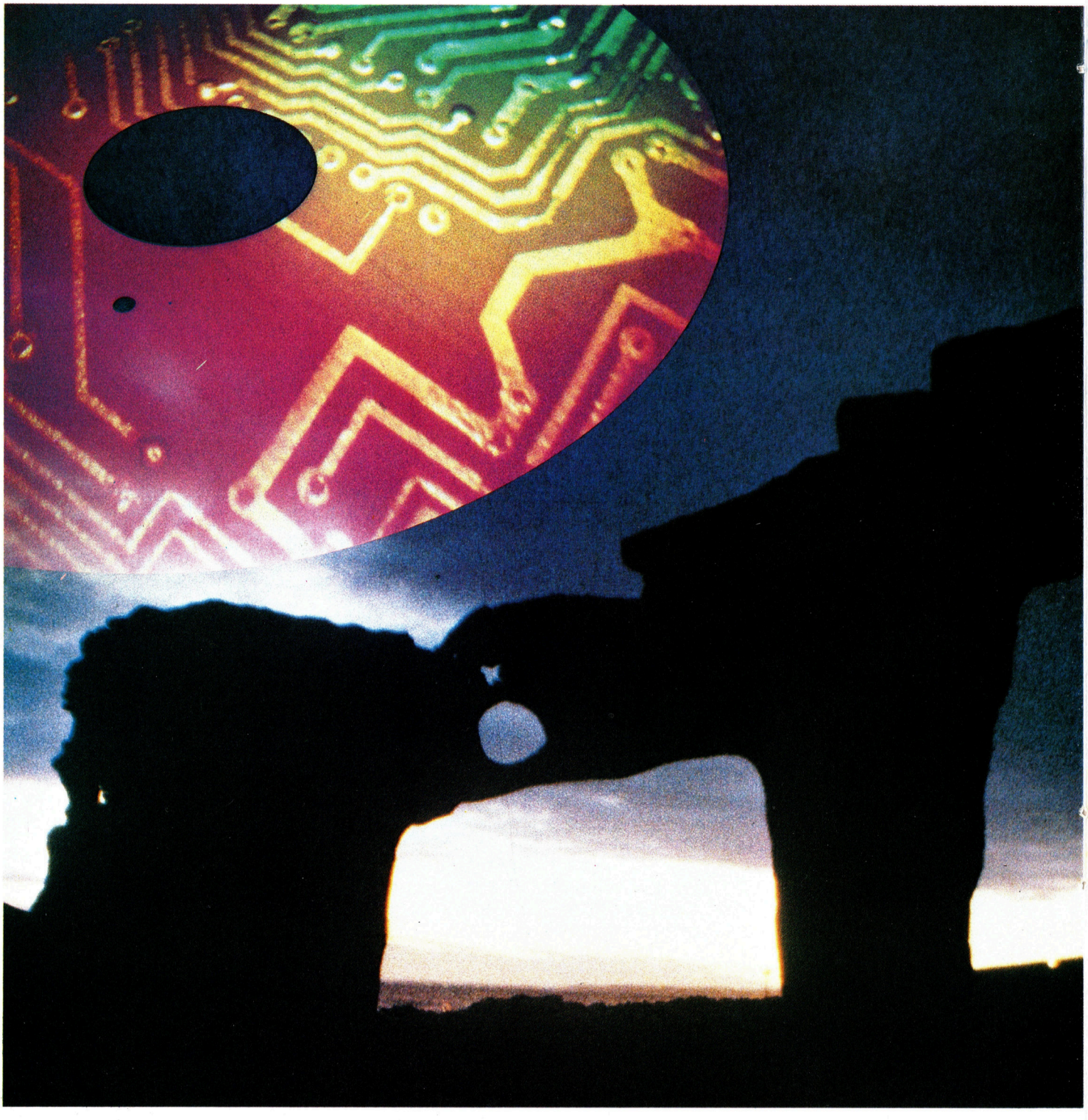


Ein Schnell-Plot-System

Diese Routine funktioniert in allen Bildschirmmodi. Per Firmware steuert sie eine Speicherstelle mit Informationen über den Bildschirmpunkt an und stellt so das neue Pixel dar.

```
; Entry:  HL contains the physical Y co-ordinate
;         (0-199)
;         DE contains the physical X co-ordinate
;         (0-639)
;         A contains the ink to set the point to
;
; Exit:   All registers preserved
;
dotpos:   equ  #bc1d      ; SCR_DOT_POSITON
encode:   equ  #bc2c      ; SCR_INK_ENCODE
;
plotter:
          push de   ; save the registers
          push hl
          push bc
          push af
;
          call dotpos     ; find the memory byte
          pop  af         ; get the ink back in a
          push af
;
; HL now contains the address of the byte
; and C contains the mask required to set only the
; required pixel on
;
          call encode     ; find the ink mask
;
; The mask to set all pixels to new ink is in a
;
          and  c          ; only want this pixel
;
          ld   b, a       ; save the new ink number
          ld   d, (hl)    ; get the current setting
          ld   a, c       ; get the pixel mask
          cmp  a          ; invert it
          and  d          ; set the pixel to ink 0
;
          or   b          ; mask in the new colour
          ld   (hl), a    ; and store it out
;
          pop  af         ; restore the registers
          pop  bc         ;
          pop  hl
          pop  de
;
          ret
;
```

Diskettenmonitor



Der Diskettenmonitor ermöglicht Ihnen den direkten Zugriff auf Diskettendateien. Sie können damit Datensätze ändern oder auch Daten retten, die bei einem Absturz des Rechners verlorengegangen sind.

Ihr Computer steuert das Lesen und Schreiben von Datensätzen über das „disk operating system" (DOS), manchmal auch „disk filing system" (DFS) genannt. Als Benutzer müssen Sie sich normalerweise nicht darum kümmern, wie oder wo Daten auf eine Floppy geschrieben werden – diese Aufgabe übernimmt das DOS/DFS für Sie.

Darüber hinaus bietet das DOS/DFS eine Möglichkeit, auf bestimmte Teile der Diskette zuzugreifen. Dadurch können Sie, ohne die Datei in Ihren Rechner zu laden, einzelne Daten auswählen, manipulieren oder ergänzen.

Sie können diese Form des „direkten Zugriffs" zum Ändern des Directory, von Namen, Daten und Verknüpfungszeigern nutzen. Auch verlorene Informationen lassen sich damit wieder herstellen. Der letzte Punkt ist vielleicht am nützlichsten. Direktzugriff ermöglicht nicht nur, gelöschte Datensätze zu restaurieren und offene Datensätze zu schließen, Sie können auch zerstörte Sektorzeiger so einrichten, daß die Sektorfolge einer Datei wieder stimmt.

Der Direktzugriff auf einzelne Informations-„Blöcke" läßt sich am besten mit einem Monitor vergleichen, der Teile des Speichers ausgibt oder ändert. Im Betrieb hat unser Programm große Ähnlichkeit mit einem Monitorprogramm.

Bevor man auf bestimmte Spuren oder Sektoren einer Diskette zugreifen kann, muß man ihre Position genau kennen. Das „Format" einer Diskette ist gewissermaßen der erforderliche Lageplan. Erzeugt wird er durch die Formatierungsroutinen im DOS/DFS.

In der Literatur wird bei der Beschreibung des Direktzugriffs durchgehend die hexadezimale Notation benutzt – auch wir halten uns daran.

Um mit dem Monitor zu arbeiten, brauchen Sie eine Beschreibung des Diskettenformats. Die Formate der einzelnen Rechner finden Sie weiter hinten im Artikel. Machen Sie unbedingt eine Sicherheitskopie, wenn die bearbeitete Diskette wertvolle Informationen enthält!

Eine zweite Bemerkung mag überflüssig scheinen, aber: Sie können keine Daten finden, die nicht mehr da sind. Eine verlorene oder gelöschte Datei zum Beispiel können Sie nur dann wieder herstellen, wenn die Daten vorher nicht überschrieben worden sind. Der Speicherplatz einer Datei wird nämlich beim Löschen für die erneute Nutzung freigegeben. Ist danach ein Schreibzugriff vorgenommen worden, können die alten Daten unwiederbringlich verloren sein. Sie könnten dann versuchen, wenigstens einen Teil zu retten, indem sie die Dateizeiger auf die noch nicht überschriebenen Sektoren setzen.

Commodore

Commodore 64 (und VC 20) arbeiten mit dem Laufwerk 1541. Das Laufwerk hat ein eingebautes Betriebssystem, zählt also zu den „intelligenten" Geräten. Sie können den Diskettenmonitor zwar auch für andere CBM-Laufwerke benutzen, Informationen über Spuren und Sektoren gelten aber nur für die 1541er.

Bei der 1541 ist die Diskette in 35 Spuren eingeteilt. Je nach Lage besteht eine Spur aus 17 bis 21 Sektoren. Es gibt insgesamt 683 Sektoren, von denen allerdings höchstens 664 benutzt werden können. Jeder Sektor nimmt 256 Bytes auf.

Das „Directory" (Inhaltsverzeichnis) befindet sich auf Spur 18. Ein Zugriff darauf erfolgt üblicherweise durch LOAD"$",8 und LIST.

Um Fehler im Directory zu korrigieren, werden Sie mit DMON sehr häufig auf Spur 18 zugreifen. Ein typisches „Rettungsprogramm" stellt gelöschte Datensätze wieder her, indem es die Dateizeiger korrigiert und dabei beschädigte Bereiche der Diskette überspringt.

Am Anfang des Directory auf Spur 18 steht die „block availability map" (BAM – Tabelle der verfügbaren Blöcke). In der BAM wird festgehalten, welche Blöcke zur Verfügung stehen. Sie wird nach jedem Zugriff auf den aktuellen Stand gebracht.

Die BAM ist folgendermaßen aufgebaut:
SPUR 18 ($12), SEKTOR 0

Byte	Inhalt
$00—$01	Spur und Sektor des ersten Directory-Blocks
$02	$41 (ASCII A zur Kennzeichnung des 1541-Formats)
$03	Null (wird nicht benutzt)
$04—$8F	BitweisesVerzeichnis der freien (1) bzw. belegten (0) Blöcke
$90—$FF	KOPF DES DIRECTORY
Format	
$90—$A1	Name der Diskette (mit Shift Space aufgefüllt – $A0)
$A2—$A3	ID-Kennung
$A4	Shift Space – $A0
$A5—$A6	Format-Typ ($32, $41–2A in ASCII)
£A7—$AA	Shift Space – $A0
$AB—$FF	Nicht benutzt ($00) bis auf den Text „BLOCKS FREE"

Das eigentliche Directory beginnt bei Spur 18, Sektor 1. Je nach Anzahl der Dateien können weitere Sektoren belegt werden. Dafür steht der Rest von Spur 18 zur Verfügung. Jeder Sektor kann bis zu acht Einträge aufnehmen.

SPUR 18($12), SEKTOR 1

Byte	Inhalte
$00—$01	Spur und Sektor des nächsten Directory-Blocks
$02—$1F	DETAILS DER ERSTEN DATEI
Format:	
$02	Dateityp (s. u.)
$03—$04	Spur und Sektor des ersten Datei-Blocks
$05—$14	Name der ersten Datei (aufgefüllt mit Shift Space – $A0)
$15—$16	Nur für REL-Dateien benutzt (Spur und Sektor des ersten Nebenblocks)
$17	Datensatzlänge der REL-Datei.

$18—$1B	Nicht benutzt
$1C—$1D	Spur und Sektor der neuen Datei beim Speichern/Ersetzen-Befehl (@)
$1E—$1F	Anzahl der Blöcke (nieder- und höherwertiges Byte)
$20—$21	Shift Space – $A0
$22—$3F	Details der zweiten Datei

(Format wie bei der ersten Datei)
Auf diese Art können bis zu acht Einträge in einem Block aufgenommen werden. Sind es weniger, dann wird der Rest des Blocks mit Nullen aufgefüllt.

Die beiden Bytes unmittelbar vor dem Dateinamen zeigen auf deren ersten Datenblock. Für die erste Datei sind das die Bytes $03 und $04. Ihre Werte (in hex) bezeichnen Spur und Sektor. $11 $01 etwa würde auf Spur 17 Sektor 1 verweisen. Der Aufbau eines typischen Datenblocks von 256 Bytes Länge ist sehr einfach. Die ersten beiden Bytes verweisen auf den nächsten Block, der Rest besteht aus Daten. Der letzte Block einer Datei beginnt mit $00, während das zweite Byte die Anzahl der belegten Bytes enthält.

```
10 PRINT"♥π":POKE53280,0:POKE53281,
   0:GOSUB2400
20 DIMA(255),S(35):HX$="0123456789
   ABCDEF":SE=1:TR=18
30 FORI=1TO17:S(I)=21:NEXT
40 FORI=18TO24:S(I)=19:NEXT
50 FORI=25TO30:S(I)=18:NEXT
60 FORI=31TO35:S(I)=17:NEXT
70 PRINT:GOSUB1000:PRINT"□"
80 A$=LEFT$(T$,1)
90 IFA$="P"THENGOSUB1200:GOTO70
100 IFA$="X"THENPRINT"BASIC":END
110 IFA$="$"THENGOSUB1500:GOTO70
120 IFA$>="0"ANDA$<="9"THEN
   GOSUB1600:GOTO70
130 IFA$="D"THENGOSUB650:GOTO70
140 IFA$="S"THENGOSUB1700:GOTO70
150 IFA$="E"THENGOSUB1800:GOTO70
170 IFA$="R"THENGOSUB1900:GOTO70
180 IFA$="W"THENGOSUB2100:GOTO70
190 IFA$="C"THENGOSUB2300:GOTO70
200 IFA$="H"THENGOSUB2400:GOTO70
210 PRINT".?UC?":GOTO70
650 OPEN15,8,15:OPEN8,8,8,"#":PRINT
   #15,"U1:"8;0;18;0:CLOSE15:CLOSE8
655 OPEN1,8,2,"$"
660 FORX=1TO141:GET#1,A$:NEXT
670 T$(0)="DELETED":T$(1)="SEQ":
   T$(2)="PROGRAM":T$(3)="USER":
   T$(4)="RELATIVE"
680 J=17:GOSUB940
690 N$=B$
700 J=2
710 GOSUB940
720 I$=B$
730 GET#1,A$
740 J=2
750 GOSUB940
760 O$=B$
770 FORL=1TO88
780 GET#1,A$
790 NEXT
800 PRINT"DISK NAME:"N$:PRINT"□□□
   □□□□□ID:"I$:PRINT"□□□□□□
   □OS:"O$"■"
810 PRINT"LENGTH","TYPE","NAME■"
820 FORP=1TO8
830 GET#1,T$,A$,A$
840 IFT$=""THENT$=CHR$(128)
850 J=15
860 GOSUB940
870 N$=B$
880 GET#1,A$,A$,A$,A$,A$,A$,A$,A$,A$,
   L$,H$
890 L=ASC(L$+CHR$(0))+256*ASC
   (H$+CHR$(0)):IFL=0THEN930
900 IFSTTHENCLOSE1:RETURN
910 PRINTL*256,T$(ASC(T$)-128),N$
920 IFP<8THENGET#1,A$,A$
930 NEXT:GOTO820
940 B$=""
950 FORL=0TOJ
960 GET#1,A$
970 IFA$<>CHR$(96)THENIFA$<>CHR$
   (160)THENB$=B$+A$
980 NEXT:RETURN
1000 T$="":PRINT".";
1010 PRINT"□▮▮";:GETA$:IFA$=""
   THEN1010
1020 IFA$=CHR$(13)THEN1100
1030 IFA$=CHR$(20)THEN1110
1040 IFLEN(T$)>10THEN1010
1050 IFA$="□"ORA$="$"THEN1090
1060 IFA$<"0"THEN1010
1070 IFA$>"Z"THEN1010
1090 T$=T$+A$:PRINTA$;:GOTO1010
1100 IFT$<>""THENRETURN
1110 IFT$=""THEN1010
1120 T$=LEFT$(T$,LEN(T$)-1)
1130 PRINTA$;:GOTO1010
1200 REM PRINT ROUTINE
1210 X$=MID$(T$,3,2):GOSUB1300:S=X
1220 X$=MID$(T$,6,2):GOSUB1300:F=X
1230 FORI=STOFSTEP9
1240 X=I:GOSUB1400:PRINTH$":";:FOR
   T=0TO8:IFI+T>255THENPRINT"**":
   RETURN
1250 X=A(I+T):GOSUB1400:PRINTH$
   "□";:NEXT
1260 FORT=0TO8:A=A(I+T):IFA<32OR
   A>91THENA=32
1270 PRINTCHR$(A);:NEXT:PRINT:NEXT:
   RETURN
1300 A$=LEFT$(X$,1);B$=RIGHT$(X$,1):
   FORI=1TO16
1310 IFA$=MID$(HX$,I,1)THENH=
   (I-1)*16
1320 IFB$=MID$(HX$,I,1)THENL=
   (I-1)
1330 NEXT:X=H+L:RETURN
1400 H=INT(X/16):L=(X-H*16)
1410 H$=MID$(HX$,H+1,1)+MID$(HX$,
   L+1,1):RETURN
1500 IFLEN(T$)=5THEN1540
1505 IFLEN(T$)<>3THENPRINT".?SX?";:
   RETURN
1510 X$=RIGHT$(T$,2)
1520 GOSUB1300
1530 PRINT".DEC"X:RETURN
1540 X$=RIGHT$(T$,2):GOSUB1300
1550 M=X:X$=MID$(T$,2,2):GOSUB1300:
   PRINT".DEC"256*X+M:RETURN
1600 V=VAL(T$):IFV>65535ORV<0THEN
   PRINT".??";:RETURN
1610 M=INT(V/256)
1620 N=V-M*256
1630 X=M:GOSUB1400:A$=H$:X=N:
   GOSUB1400:A$=A$+H$
1640 PRINT".HEX□"A$:RETURN
1700 PRINT"LAST TRACK:$";:X=TR:GOSUB
   1400:PRINTH$
1710 PRINT"□□□□SECTOR:$";:X=SE:
   GOSUB1400:PRINTH$
1720 RETURN
1800 X$=MID$(T$,3,2)
1810 GOSUB1300
1820 A=X:X$=MID$(T$,6,2):GOSUB1300:
   B=X
1830 A(A)=B:PRINT"OK":RETURN
1900 IFLEN(T$)=1THENGOSUB2000:
   RETURN
1910 X$=MID$(T$,3,2):GOSUB1300:IFX<1
   ORX>35THENPRINT".?IT?";:RETURN
1920 A=X:X$=MID$(T$,6,2):GOSUB1300:
   IFX<0ORX>S(A)THENPRINT".?IS?";:
   RETURN
1930 TR=A:SE=X:GOSUB2000
1940 RETURN
2000 OPEN15,8,15
2010 OPEN8,8,8,"#"
2020 PRINT#15,"U1:"8;0;TR;SE
2030 PRINT#15,"B-P:"8;0
2040 FORI=0TO255:GET#8,A$:IFST<>0
   ANDST<>64THENPRINT".?DR?":
   CLOSE8:CLOSE15:RETURN
2050 A(I)=ASC(A$+CHR$(0))
2060 NEXT
```

```
2070 CLOSE8:CLOSE15:PRINT"OK":RETURN
2100 IFLEN(T$)=1THENGOSUB2200:
   RETURN
2110 X$=MID$(T$,3,2):GOSUB1300:IFX<1
   ORX>35THENPRINT".?IT?";:RETURN
2120 A=X:X$=MID$(T$,6,2):GOSUB1300:
   IFX<0ORX>S(A)THENPRINT".?IS?";:
   RETURN
2130 TR=A:SE=X:GOSUB2200
2140 RETURN
2200 OPEN15,8,15
2210 OPEN8,8,8,"#"
2220 PRINT#15,"B-P:"8;0
2230 FORI=0TO255:PRINT#8,CHR$(A(I));:
   IFST<>0ANDST<>64THENPRINT
   ".?DW?":GOTO2250
2240 NEXT:PRINT#15,"U2:"8;0;TR;SE
2250 CLOSE8:CLOSE15
2260 RETURN
2300 OPEN15,8,15
2310 INPUT#15,A,B$,C,D
2320 PRINT"ERROR NO.:"A
2330 PRINT"□□□□□TYPE:□"B$
2340 PRINT"□□@□TRACK:"C
2350 PRINT"□□□SECTOR:"D
2360 CLOSE15:RETURN
2400 PRINT"■P□XX□XX□□ — PRINT
   MEMORY
2410 PRINT"D□□□□□□□□ —
   DIRECTORY
2420 PRINT"R□XX□XX□□ — READ
   FROM DISK
2430 PRINT"W□XX□XX□□ — WRITE TO
   DISK
2440 PRINT"E□XX□XX□□ — EDIT
   MEMORY
2450 PRINT"S□□□□□□□□□ — LAST
   SECTOR/TRACK
2460 PRINT"$□□□□□□□□□ — HEX
   TO DECIMAL
2470 PRINT"(NUMBER) — DECIMAL TO HEX
2480 PRINT"C□□□□□□□□□ —
   LAST ERROR
2490 PRINT"X□□□□□□□□□ — EXIT
   TO BASIC
2500 PRINT"H□□□□□□□□□ —
   PRINTS MENU
2510 PRINT"■IT□ — ILLEGAL TRACK
2520 PRINT"IS□ — ILLEGAL SECTOR
2530 PRINT"SX□ — SYNTAX ERROR
2540 PRINT"UC□ — UNKNOWN COMMAND
2550 PRINT"DR□ — DISK READ ERROR
2560 PRINT"DW□ — DISK WRITE ERROR
2570 RETURN
```

Wenn Sie das Programm gestartet haben, erscheint ein Menü mit elf Befehlen sowie eine Liste der Fehlermeldungen und ihre Erklärungen. Die Befehle werden durch die angegebenen Tasten eingegeben, wobei anstelle von XX eine numerische Eingabe erwartet wird.

Die Eingabe D gibt ein ausführliches Directory aus — Programmlängen werden in Bytes angegeben, und es werden sowohl die gelöschten als auch die gültigen Dateinamen dargestellt.

Wenn Sie einen Teil der Diskette lesen wollen, geben Sie zuerst ein R und dann die durch ein Leerzeichen getrennten Nummern von Spur und Sektor ein. Denken Sie daran, Hexzahlen zu benutzen! Versuchen Sie R 12 01 einzugeben, um das Directory so zu lesen, wie es auf der Diskette steht. Die gelesenen Daten werden im Diskettenbuffer gespeichert und können durch P Startbyte Endbyte auf den Bildschirm ausgegeben werden. Um den ganzen Block zu sehen, benutzen Sie P 00 FF. In der linken Spalte wird die Bytenumerierung, in der Mitte der Inhalt und rechts die Übersetzung in ASCII ausgegeben.

Zum Ändern eines Bytes geben Sie E, die Nummer des Bytes und seinen neuen Wert ein. Dieser Wert erscheint dann in der richtigen Position auf dem Bildschirm. Zurückschreiben können Sie die Daten mit W, gefolgt von Zielspur und Sektor.

Es gibt noch weitere Befehle: S gibt Spur und Sektor des letzten Zugriffs aus; $ Hexzahl wandelt in dezimal um, für die Umkehrung muß nur die Dezimalzahl selbst eingegeben werden. C gibt den Code der letzten Fehlermeldung aus, H springt ins Menü und X zurück zu BASIC.

Beim Löschen einer Datei wird das Dateiflag geändert. Dieses Flag steht unmittelbar vor dem Spur- und Sektorzeiger, der seinerseits unmittelbar vor dem Dateinamen steht. Mögliche Dateitypen sind DEL, SEQ, PRG, USR und REL.

Den Dateityp können Sie mit dem Diskettenmonitor sehr einfach feststellen. Die Dateitypen und ihre Hex-Werte haben wir zu einer Tabelle zusammengefaßt:

Dateityp	Geschl.	Offen	Geschützt
DELeted	$80	$00	–
SEQuentiell	$81	$01	$C1
PRoGramm	$82	$02	$C2
USeR	$83	$03	$C3
RELativ	$84	$04	$C4

Eine gelöschte Programmdatei führt also $02 als Dateiflag. Solange die zugehörigen Blöcke nicht über-

schrieben sind, kann sie wieder aktiviert werden – dazu wird der Wert einfach mit DMON von $02 auf $82 geändert.

Ganze Diskette löschen

Mit dem Wert $C2 (für andere Dateitypen siehe Tabelle) kann man die Datei vor versehentlicher Löschung bewahren. Im Directorylisting sind solche Dateien durch ein „<" neben dem Dateityp gekennzeichnet. Geschützte Dateien können nur beseitigt werden, wenn man mit NEW die ganze Diskette löscht. Natürlich geht es auch mit Hilfe des Monitors, indem man das Dateiflag auf „offen" stellt.

Manchmal sind Teile von Dateien bereits verschwunden. Sie müssen dann die ganze Datei mit Hilfe der Spur- und Sektorzeiger „durchkämmen", um den Umfang des Schadens festzustellen.

Dazu gehen Sie an den Anfang des Directory ($12 $01) und bestimmen Spur und Sektor des ersten Dateiblocks. Untersuchen Sie den Block mit DMON – wenn er „gesund" ist, also keine sinnlosen Daten enthält, können Sie den nächsten Block prüfen. Spur und Sektor finden Sie in den ersten beiden Bytes des aktuellen Blocks.

Wenn Sie an einen zerstörten Block geraten, prüfen Sie zuerst, ob sich der ursprüngliche Zeiger überschreiben läßt. Beim Lesen wird der zerstörte Block dann einfach übersprungen. Als nächstes wird dann das Programm aufgerufen, das die Daten benötigt, die Daten werden korrigiert und erneut gespeichert. Dazu können Sie auch eine neue Diskette verwenden.

Acorn B

Disketten für den Acorn haben entweder 40 oder 80 Spuren zu je zehn Sektoren, so daß auf jeder Seite 400 oder 800 Sektoren zur Verfügung stehen. Jeder Sektor bietet Platz für 256 Bytes.

Das Directory (man erhält es durch den Befehl *CAT) befindet sich auf Spur 0, Sektor 0 und 1. Das Format finden Sie in der Tabelle. Es gibt maximal 31 Einträge.

SPUR 0 SEKTOR 0 FORMAT

Byte	Inhalt
&00—&07	Die ersten acht Zeichen des Diskettennamens, aufgefüllt mit Leerzeichen
&08—&0E	Erster Dateiname, aufgefüllt mit Leerzeichen (höchstens sieben Zeichen)
&0F	Directory-Zeichen der ersten Datei
&10—&16	Zweiter Dateiname, aufgefüllt mit Leerzeichen (höchstens sieben Zeichen)
&17	Directory-Zeichen der zweiten Datei

Von diesen acht Byte langen Namen- und Directory-Blöcken sind maximal 31 vorhanden.

SPUR0 SEKTOR 1 FORMAT

Byte	Inhalt
&00—&03	Die letzten drei Zeichen des Diskettennamens, aufgefüllt mit Leerzeichen
&04	Anzahl der Schreiboperationen, die auf der Diskette durchgeführt werden
&05	8-Byte-Blockzähler (achtfache Anzahl der aktiven Dateien)
&06	Einzelne Bits gesetzt (s. u.)
&07	Anzahl der Sektoren (acht niederwertige Bits (LSB) einer 10-Bit-Zahl)
&08—&0F	SPEICHER-TABELLE DER ERSTEN DATEI

Dateiformat

&08—&09	Ladeadresse, LSB zuerst (0 für eine Datendatei und &1900 für ein BASIC-Programm).
&01—&08	Startadresse, LSB zuerst (0 für eine Datendatei, &8023 für BASIC I und &801F für BASIC II)
&0C—&0D	Dateilänge in Bytes, LSB zuerst
&0E	Einzelne Bits gesetzt (s. u.)
&0F	Start-Sektor (acht LSBs einer 10-Bit-Zahl)
&10—&17	SPEICHER-TABELLE DER ZWEITEN DATEI

Dateiformat wie oben.
Und so weiter bis maximal 31 Dateien.

Die Einzelbits in &06 und &0E haben folgende Bedeutung: Wenn Bit 5 und 4 in &06 gesetzt sind, wird das Programm automatisch geladen (!BOOT). Bits 1 und 0 sind die höchstwertigen Bits (MSB) einer 10-Bit-Zahl, der Rest steht in &07.

Bit 7 und 6 in &0E sind die MSBs der 18-stelligen Startadresse. Der Rest ist in &0A bzw. &0B untergebracht. Bits 5 und 4 sind die MSBs der Dateilänge (Rest in &0C, &0D), während Bits 3 und 2 diese Aufgabe für die Ladeadresse übernehmen. Bits 1 und 0 sind die MSBs des Startsektors (Rest in &0F).

Der Startsektor berechnet sich aus Byte &0F und Bits 1 und 0 in &0E: Wenn z. B. &38 in &0F steht, während die MSBs null sind, fängt die Datei bei &38/10=5.6 an, also bei Spur 5, Sektor 6. Enthält &0F &43 und ist Bit 0 in &0E gesetzt, dann fängt die Datei bei &143/10, Spur 32, Sektor 3 an.

Man sollte sich merken, daß die Ladeadresse (&08, &09) für eine Datei &0000 und für BASIC &1900 ist (LSB zuerst!). Die Startadressen sind &0000 für Dateien &8023 für BASIC.

Die Speichertabelle eines 12 067 Bytes langen BASIC-Programms könnte also folgendermaßen aussehen: 00 19 23 80 23 2F CC 02. In etwas anderer Form kann man diese Informationen auch mit dem Befehl *INFO*.* erhalten.

```
10 MODE3
20 DIM block 12
30 DIM buffer 255
40 printer=0
50 DR%=0
60 PROCload(DR%,0,0,1)
70 REPEAT
80 PROCprint
90 VDU28,0,24,79,21
100 PROCcommand
110 UNTIL com$="END□"
120 END
130 DEFPROCload(DR%,TR%,SCT%,RNW)
140 X%=block MOD 256:Y%=block DIV 256
```

```
150 A%=&7F
160 block?0=DR%
170 block!1=buffer
180 block?5=3
190 block?6=&4B+8*RNW
200 block?7=TR%
210 block?8=SCT%
220 block?9=&21
230 CALL&FFF1
240 ENDPROC
250 DEFPROCprint
260 CLS:VDU26
270 IF printer=1 THEN VDU2 ELSE VDU3
280 PRINT"Track□";TR%;"□□Sector□";
    SCT%;"□□Drive□";DR%
290 PRINT
300 FOR I%=0 TO 1
310 PRINT"□□□□□□□□bytes (in hex)
    □□□□□□□□□□□□ascii□□□";
320 NEXT
330 PRINT
340 line=32:pos=7:buff=buffer
350 FOR lin=1 TO line
360 L=LEN(STR$((lin-1)*8))
370 FOR I=1 TO 4-L:VDU32:NEXT
380 PRINT;STR$((lin-1)*8);"□□";
390 FOR linpos=0 TO pos
400 cont=linpos?buff
410 IF cont<&10 THEN PRINT"0";
420 PRINT;~cont;"□";
430 NEXT
440 PRINT"□";
450 FOR linpos=0 TO pos
460 cont=linpos?buff
470 asc=(cont>31 AND cont<127)
480 IF asc THEN PRINTCHR$cont; ELSE
    PRINT".";
490 NEXT
500 IF lin MOD2=0 THEN PRINT
510 buff=buff+8
520 NEXT
530 PRINT
540 VDU3
550 ENDPROC
560 DEFPROCcommand
570 INPUT"$"com$:comlen=LEN(com$):
    com$=com$+LEFT$("□□□□",4-
    comlen)
580 command=(INSTR("□□□□DRV TRK
    SCT INS PRT WRT END
    SHOW",com$,0))DIV4+1
590 ON command GOTO 600,610,620,630,
    640,650,660,680,670
600 PROCcommand:ENDPROC
610 PROCdrive:ENDPROC
620 PROCtrack:ENDPROC
630 PROCsector:ENDPROC
640 PROCinsert:ENDPROC
650 PROCprinter:ENDPROC
660 PROCload(DR%,TR%,SCT%,0):ENDPROC
670 PROCload(DR%,TR%,SCT%,1):ENDPROC
680 ENDPROC
690 DEFPROCdrive
700 INPUT"Drive□",DR%
710 ENDPROC
720 DEFPROCprinter
730 IF printer=1 THEN printer=0 ELSE
    printer=1
740 ENDPROC
750 DEFPROCinsert
760 REPEAT:INPUT"Offset from start
    ?...."os:UNTIL os>=0 AND os<256
770 REPEAT:INPUT"New value ?...."val$:val=
    EVAL(val$):UNTIL val>=0 AND val<256
780 buffer?os=val
790 ENDPROC
800 DEFPROCtrack
810 REPEAT
820 INPUT"Track□□?□"TR%
830 UNTIL TR%>=0 AND TR%<80
840 PROCload(DR%,TR%,SCT%,1)
850 ENDPROC
860 DEFPROCsector
870 REPEAT
880 INPUT"Sector□□?□"SCT%
890 UNTIL SCT%>=0 AND SCT%<10
900 PROCload(DR%,TR%,SCT%,1)
910 ENDPROC
```

Nach dem Start meldet sich der Monitor mit dem Inhalt von Spur 0, Sektor 0, Laufwerk 0 und der ASCII-Darstellung dieses Inhalts. Wollen Sie andere Teile der Diskette lesen, geben Sie DRV zur Auswahl des Laufwerks, TRK für die Spur und SCT für den Sektor ein. Versuchen Sie, zwischen den Sektoren 0 und 1 umzuschalten – Sie sehen zuerst die Dateinamen und dann die Speichertabelle.

Mit INS ändern

Zum Ändern eines Byte wird INS eingegeben. Danach kommt zuerst die Nummer des Bytes und dann sein neuer Wert. Dateinamen in Sektor 0 werden in ASCII gegeben. Werte für die Speichertabelle in Sektor 1 dagegen in hex.

Am Ende können Sie den Sektor mit WRT auf die Diskette schreiben.

Einen Ausdruck erhalten Sie mit PRT; ein zweites PRT schaltet den Drucker wieder ab. Nach einem Diskettenwechsel kann es vorkommen, daß Sie SHOW eingeben müssen, um den Inhalt der neuen Diskette zu lesen. Im allgemeinen funktioniert das aber automatisch. Nach getaner Arbeit geben Sie END ein.

Man kann DMON zwar für die „Wartung" von Dateien einsetzen – die wichtigste Anwendung ist aber die Rettung verlorener Dateien. Sie brauchen zuerst die Dateilänge und, noch wichtiger, den Startsektor. Wenn alles in Ordnung ist – d. h., wenn die Datei noch nicht zerstört ist –, erhalten Sie diese Information durch Eingabe von *INFO <Dateiname>.

Wenn Sie diese Information nicht notiert haben, dann haben sie ein ziemlich langweiliges Forschungsvorhaben vor sich, um Startspur und -sektor sowie die Länge der verlorenen Datei herauszufinden. Verwenden Sie dann die Umkehrung der zuvor beschriebenen Prozedur, um die Information über Spur und Sektor in ein hex-Byte und die beiden MSBs zu zerlegen.

Um die Datei zu retten, wählen Sie den nächsten freien 8-Byte-Block in Sektor 0 und füllen ihn mit einem neuen Dateinamen plus Directoryzeichen (7+1 Bytes), alles in ASCII. Vorsicht – nicht den Namen einer aktiven Datei überschreiben!

Dann rufen Sie mit DMON Sektor 1 auf und erhöhen &05 um 8, damit diese Datei akzeptiert wird.

Sie haben jetzt eine neue Datei angelegt, und die braucht ihre eigene Speichertabelle, also acht Bytes von &08 bis &0F. Sie gehört in die ersten &08 bis &0F. Sie gehört in die ersten freien acht Bytes, die dem Namen entsprechen. Die ersten beiden Bytes sind die Ladeadresse – &0000 (Daten) bzw. &1900 (BASIC) –, und das zweite Paar ist die Startadresse.

Mit dem Monitor schreiben Sie die Dateilänge in die nächsten zwei Bytes (das LSB zuerst). Kennen Sie die Länge noch nicht, setzen Sie provisorisch &00 &10 ein.

Etwas problematisch ist das Setzen des siebten Bytes, dessen LSBs den Startsektor der Datei angeben, die Sie retten wollen. Denken Sie daran, daß sich diese Information bereits auf der Diskette befindet, daß die Datei aber erst gerettet werden kann, wenn der Startsektor gefunden und sein Wert in Byte 7 eingetragen worden ist.

Wenn Sie damit fertig sind, verlassen Sie den Diskettenmonitor. Wußten Sie die Dateilänge nicht, geben Sie jetzt PRINT LOMEM ein, und notieren Sie das Ergebnis. Dann laden Sie das Programm oder den Datensatz und prüfen, ob Sie erfolgreich waren. Fragen Sie noch einmal LOMEM ab – die Differenz zwischen den beiden Werten ist die Dateilänge, die Sie jetzt in &0C &0D der Speichertabelle eintragen können. Um ganz sicher zu gehen, sollten Sie die Datei jetzt auf eine neue Diskette kopieren.

Dragon

Die Dragon-Diskettenstation besitzt ein eigenes Interface, welches auch das DOS enthält.

Die Disketten sind in 40 oder 80 Spuren mit 18 Sektoren à 256 Bytes aufgeteilt.

Das Directory, das die Verwaltung der Dateien vornimmt, befindet sich auf den Spuren 16 und 20. Beide Spuren sind identisch, aber Spur 16 wird für das Directory benutzt, Spur 20 für das System.

Spur 16 Sektor 3

Byte	Inhalt
1	Dateiflag, z. B. 00 für aktive Datei, 02 für eine geschützte Datei, 81 für gelöschte Dateien
2–9	Dateiname, aufgefüllt mit Nullen
10–12	Dateibeschreibung, z. B. BAS, BAK
13–14	MSB und LSB einer 16-Bit-Zahl für Startspur und -sektor.
15	Gesamtanzahl der benutzten Sektoren
16–24	Verweist auf verknüpfte Dateien
16–17	Höher- und niederwertige Bytes, verweisen auf den nächsten Startsektor
18	Anzahl der benutzten Sektoren
19–21	dito für den nächsten Abschnitt
22–24	dito für den letzten Abschnitt
25	Anzahl der belegten Bytes im letzten Sektor.

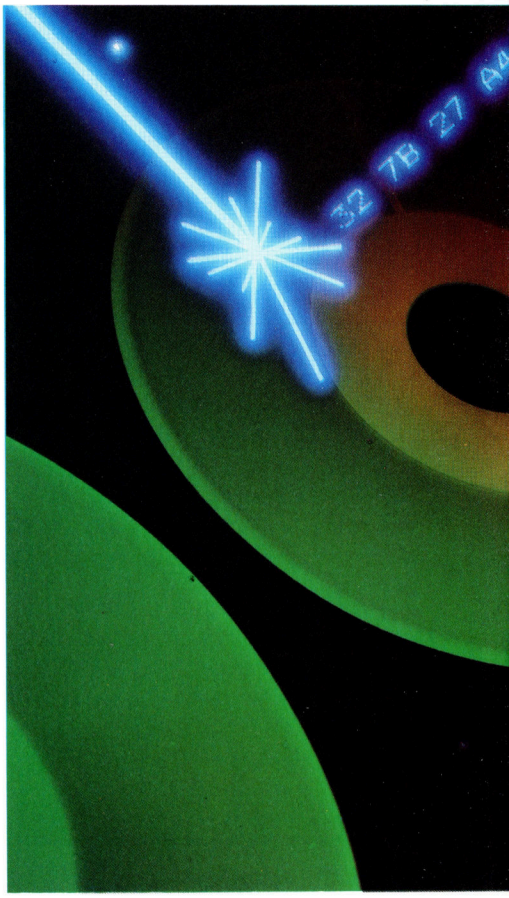
Die Sektoren 1 und 2 von Spur 16 und 20 informieren den Computer darüber, welche Sektoren belegt sind. Jedes Byte repräsentiert acht Sektoren. Für jeden Sektor gibt es ein Bit, gezählt wird von Spur Null, Sektor 1. Bei einer frisch formatierten Diskette sind alle Bits auf Null gesetzt, für jeden belegten Sektor wird eine 1 eingetragen, die beim Löschen wieder auf Null gesetzt wird.

Für ein BASIC-Programm könnte der Directory-Eintrag etwa folgendermaßen aussehen:

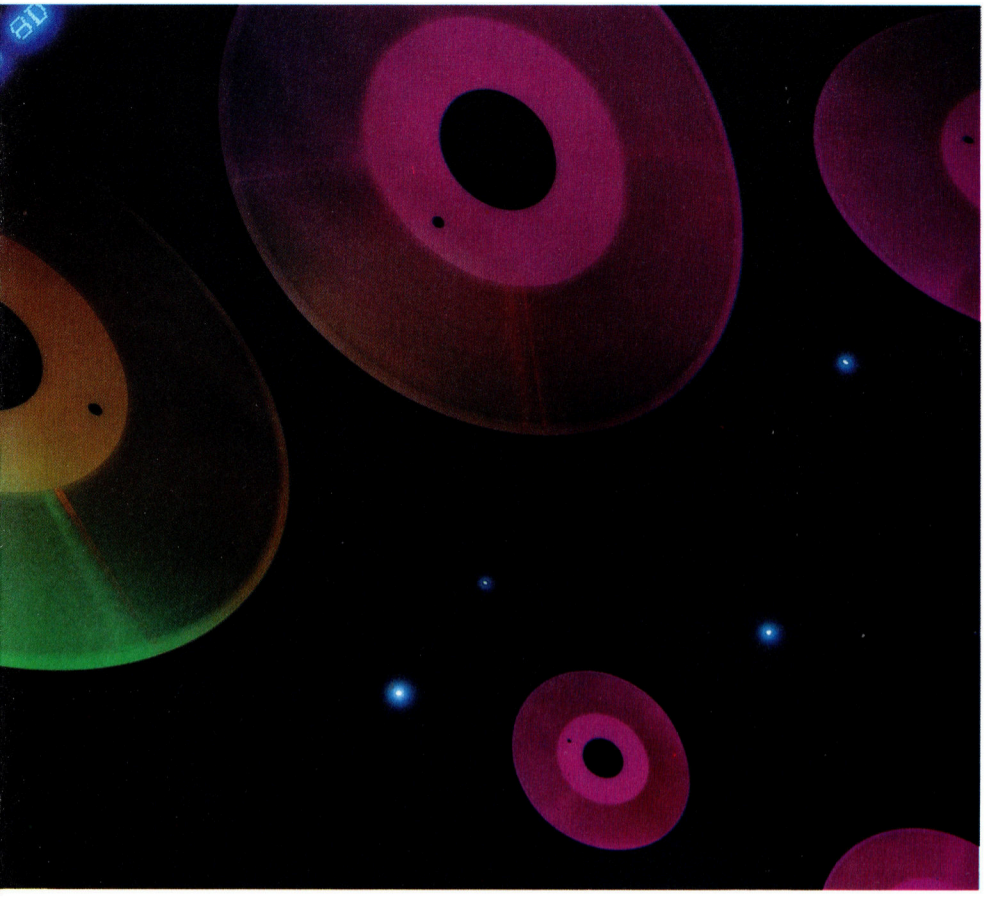
00 A B C 00 00 00 00 00 B A S 01 44
18 00 FC
09 00 00 00 00 00 00 DB

Das ist eine Datei namens „ABC", die bei &H0144 anfängt, &H18 Sektoren lang ist, mit einer Erweiterung, die bei &H00FC anfängt und neun Sektoren lang ist. Im letzten Sektor sind &HDB Bytes belegt. Um den Anfang der Datei zu finden, rechnen Sie einfach &H0144=324 dezimal durch 18 aus (Anzahl der Sektoren pro Spur). Das Ergebnis ist 18 ohne Rest, die Datei fängt also in Spur 18, Sektor 1 an. Ihre Länge beträgt &H18+&H8 Sektoren +&HDB Bytes, also insgesamt 8411 Bytes.

```
10 CLEAR5000:DIMA$(1),D$(1),D(160):
   C$="↑"+CHR$(10)+CHR$(8)+
   CHR$(9)+"AH"+CHR$(13)+"□":D=1
20 CLS:PRINT@13,"menu"
30 PRINT@106,"lOAD SECTOR":PRINT
   @170,"vIEW/EDIT SECTOR":PRINT@234,
   "sAVE SECTOR":PRINT@298,"cATALOGUE"
40 R$=INKEY$:IFR$=""THEN40
50 R=INSTR("LVSC",R$):IFR=0 THEN40
60 IFSL=0AND(R=2ORR=3) THENPRINT:
   PRINT"NO SECTOR LOADED":FORK=1
   TO2000:NEXT:GOTO20
70 CLS:ON R GOSUB1000,2000,3000,4000
80 GOTO20
1000 SL=1:GOSUB5000
1010 SREADD,T,S,A$(0),A$(1)
1020 RETURN
2000 F=1:H=1:CLS:PRINT"aSCII OR hEX
   LISTING ?"
2010 R$=INKEY$:IFR$<>"A"AND
   R$<>"H" THEN2010
2020 AS=0:IFR$="A" THENAS=1
2030 IFF=0 THEN2050
2050 PK=96:CP=1535:IFAS=1 GOSUB
   2320 ELSEGOSUB2280
2050 POKECP,PK:CP=1024+Y*32+X*3:
   PK=PEEK(CP):POKECP,239
2060 PRINT@321,"TOP BYTE=";H
```



```
2070 R$=INKEY$:IFR$=""THEN2070
2080 R=INSTR(C$,R$):IFR=0 THEN2070
2090 F=0:ON R GOTO 2100,2110,2120,
   2130,2140,2150,2160,2170
2100 Y=Y-1:GOTO2210
2110 Y=Y+1:GOTO2210
2120 X=X-1:GOTO2210
2130 X=X+1:GOTO2210
2140 AS=1:GOTO2040
2150 AS=0:GOTO2040
2160 RETURN
2170 PRINT@384,"INPUT NEW CONTENTS
   (HEX)□";:INPUTH$
2180 V$=CHR$(VAL("&H"+H$)):P=H+
   Y*11+X
2190 MID$(A$(P/128),P+128*(P>128),
   1)=V$
2200 F=1:GOTO2030
2210 IFY<0 THENH=H-44:Y=0:F=1
2220 IFY>7 THENH=H+44:Y=7:F=1
2230 IFX<0 THENX=10:Y=Y-1:IFY<0
   THENH=H-11:Y=0:F=1
2240 IFX>10 THENX=0:Y=Y+1:IFY>7
   THENY=7:H=H+11:F=1
2250 IFH=-10ORH=-43 THENH=1:
   F=0:ELSEIFH<1 THENH=1:F=1
2260 IFH=179ORH=212 THENH=168:
   F=0 ELSEIFH>168 THENH=168:F=1
2270 GOTO2030
```



```
2280 CLS:FORJ=H TOH+87 STEP11:FOR
   T=ØTOIØ
2290 PRINTRIGHT$("Ø"+HEX$(ASC
   (MID$(A$(J/128),J+T+128*
   ((J+T)>128)))),2);"□";
2300 NEXT:PRINTCHR$(8);:NEXT
2310 RETURN
2320 CLS:FORJ=H TOH+87 STEP11:FOR
   T=ØTOIØ
2330 G=ASC(MID$(A$(J/128),J+T+128*
   ((J+T)>128))):IFG<32 THEN2350
2340 PRINT"□";CHR$(G);"□";:GOTO2360
2350 PRINTLEFT$("Ø"+HEX$(G),2);"□";
2360 NEXT:PRINTCHR$(8);:NEXT:RETURN
3000 CLS:PRINT"SAVE TO SAME SECTOR
   (Y/N) ?"
3010 R$=INKEY$:IFR$<>"Y"AND
   R$<>"N" THEN3010
3020 IFR$="Y" THEN3040
3030 CLS:GOSUB5000
3040 PRINT:PRINT"ARE YOU SURE (Y/N) ?"
3050 R$=INKEY$:IFR$<>"Y"
   ANDR$<>"N" THEN3050
3060 IF R$="N" THENRETURN
3070 SWRITED,T,S,A$(Ø),A$(1)
3080 RETURN
4000 GOSUB5050
4010 PRINT#PR,TAB(14);"START□□NO."
4020 PRINT#PR,"□□NAME□□TYPE□
   TR□□SC□SECS□LEN"
4030 FORJ=ØTO15:SREAD1,16,J+3,D$(Ø),
   D$(1)
4040 FORK=1TO250 STEP25
4050 GOSUB6000
4060 IFASC(V$)<>Ø ANDASC(V$)<>2
   THEN4120
4070 PRINT#PR,MID$(V$,2,8);TAB(8);".";
   MID$(V$,10,3);
4080 TS=-1:FORP=13TO22 STEP3:
   V=256*ASC(MID$(V$,P))+ASC(MID$
   (V$,P+1)):EB=ASC(MID$(V$,P+2)):
   TS=TS+EB:IF EB=Ø THEN4110
4090 IFP<>13 THENPRINT#PR
4100 PRINT#PR,TAB(12);INT(V/18);TAB(16);
   1+V-18*INT(V/18);TAB(20);ASC(MID$
   (V$,P+2));
4110 NEXTP:PRINT#PR,TAB(24);256*TS+
   ASC(MID$(V$,25))
4120NEXTK,J:R$=INKEY$:IFPR=-2 THEN
   4140
4130 R$=INKEY$:IFR$="" THEN4130
4140 RETURN
5000 INPUT"TRACK NUMBER (Ø-39)□";T
5010 INPUT"SECTOR NUMBER
   (1-18)□";S
5020 INPUT"DRIVE NUMBER (1-4)□";D
5030 IFD>4ORD<1ORT>39ORT<ØOR
   S>18ORS<1 THEN5000
5040 RETURN
5050 PR=Ø:IF(PEEK(65314)AND1)=1 THEN
   RETURN
5060 PRINT"OUTPUT TO PRINTER (Y/N) ?"
5070 R$=INKEY$:IFR$<>"Y"AND
   R$<>"N" THEN5070
5080 IFR$="Y" THENPR=-2
5090 RETURN
6000 V$=MID$(D$(K/128),K+128*
   (K>128),25):IFLEN(V$)<25 THENV$=
   V$+MID$(D$(1+K/128),1,25-LEN(V$))
6010 RETURN
```

Tippen Sie das Programm ein oder laden Sie es, und legen Sie dann die Diskette ein, mit der Sie arbeiten wollen. Zum Üben sollten Sie aber keine wichtige Diskette wählen. Nach RUN bietet das Menü vier Möglichkeiten: Sektor laden, Sektor ausgeben/editieren, Sektor schreiben und Katalog. Geben Sie zuerst C ein. Sie erhalten dann ein ausführliches Inhaltsverzeichnis mit allen Dateien. Die Liste enthält die Dateinamen, Typ, Startspur und -sektor sowie die Länge in Bytes. Besteht eine Datei aus mehreren verknüpften Abschnitten, werden auch Startspuren und -sektoren der einzelnen Abschnitte ausgegeben.

Geben Sie als nächstes L ein, um einen Sektor zu laden, etwa das Directory, Spur 16, Sektor 3. Nach V können Sie mit A oder H zwischen einer Darstellung in hex oder ASCII wählen. Sie sollten jetzt in der Lage sein, die Zahlen oder Buchstaben mit dem vorher dargestellten Directory zu vergleichen.

Mit den Pfeiltasten können Sie den Cursor auf ein Byte setzen, das Sie ändern wollen. Überschreiben Sie es einfach mit der neuen Hexzahl und schließen Sie die Eingabe mit einem Leerzeichen ab. Nur ein Teil des Sektors paßt auf den Schirm. Um den Rest zu sehen, bewegen Sie den Cursor einfach in die unterste Zeile; die Bildschirmanzeige scrollt dann nach oben. Dabei wird die Nummer des aktuellen Bytes fortlaufend angezeigt.

Vernünftigerweise sollten Sie dieses Programm auf eine neue Diskette speichern und es auf der alten noch einmal löschen. Das ist nötig, weil Sie zwar den Dateinamen wieder gesetzt haben, nicht aber die Bits in Sektor 1 und 2 des Directory.

Kompaktklasse

Aus den im vorhergehenden Kapitel behandelten Verkürzungstechniken soll ein vielseitiges Programm zur „Textkompression" entwickelt werden. Unser Programmvorschlag arbeitet mit Vier-Bit-Blöcken. Den Assembler-Teil finden Sie in dieser Ausgabe. Weitere Ergänzungen dazu folgen im nächsten Heft.

Die bei unserem Programm eingesetzte Methode ist eine verkürzte Fassung des Vier-Bit-Algorithmus. Der Text wird dabei in Vier-Bit lange Blöcke umgesetzt, die entweder eines der am häufigsten verwendeten Zeichen oder einen Code mit Informationen über den folgenden Vier-Bit-Block darstellen. In der Tabelle rechts wird erläutert, was die einzelnen „Nibbles" (Vier-Bit-Blöcke) bedeuten.

In der vorliegenden Form komprimiert das Programm nur Texte aus Großbuchstaben, Leerzeichen, Kommas und Punkten. Es können aber auch bis zu 16 Tokens (Abkürzungen) verarbeitet werden.

Beim Einsatz muß der Anwender die Adresse des zu komprimierenden Strings eingeben und auch die Adresse, an der der neue String gespeichert werden soll. Der String wird von einem Zählbyte eingeleitet, auf das die Buchstaben folgen. Die Stringlänge ist auf 255 Bytes begrenzt.

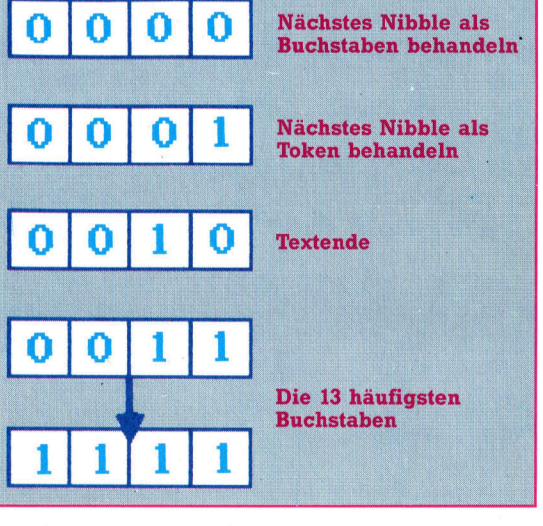
Wer in Maschinensprache programmieren kann, wird mit der Erweiterung keine besonderen Schwierigkeiten haben. Eine wirkungsvolle Ausbauvariante könnte etwa die Erzeugung einer Token-Tabelle mit 255 Elementen sein. Damit könnten neben Groß- und Kleinbuchstaben auch zusätzliche Tokens verarbeitet werden. Wenn dies vor der Prüfung auf Acht-Bit-Tokens geschieht, kann das restliche Programm bis auf die zusätzliche Tabelle unverändert bleiben.

Die Tabelle für diese Variante sollte so aufgebaut sein, daß die Vier-Bit-Werte die häufigsten Buchstaben in Kleinschrift sowie das Leerzeichen und das Symbol für „neue Zeile" darstellen. In der Acht-Bit-Tabelle stehen dann nur die weniger gebräuchlichen Kleinbuchstaben. Seltene Kleinbuchstaben stehen zusammen mit den Großbuchstaben und Tokens in der neuen Tabelle.

Die Programmadresse ORG kann je nach Rechner abgewandelt werden. Das Programm muß assembliert und der übersetzte Code auf Cassette oder Diskette gespeichert werden.

Unser Programm übersetzt Text in Vier-Bit-Blöcke, sog. „Nibbles". Die einzelnen Nibbles haben unterschiedliche Bedeutung (siehe Tabelle). Hier sind nur die Werte 0 bis 2 eingesetzt. Der Einsatz des Wertes 3 könnte beim Erweitern des Programms ein zusätzliches Signal definieren. Die 3 würde dabei als Zeiger einer Tabelle mit 255 Elementen fungieren. Dadurch wird die Darstellung von Kleinbuchstaben und eine größere Zahl von Tokens möglich.

Kompressions-Codierung



Programm zur Textkompression

Unser Programm für die Textkompression erscheint in zwei Abschnitten. In dieser Folge haben wir das Assembler-Listing für Z80-Rechner abgedruckt. Die Version für den Prozessor 6502 folgt im nächsten Heft. Außerdem versorgen wir Sie auch mit BASIC-Ladeprogrammen für beide Versionen. Das Programm läuft, auch wenn Sie keinen Assembler haben.

```
        org 30000

        jr      start           ; leap round the data space

string: dw      0       ; load with input string address
output: dw      0       ; enter address for output string
status: db      0       ; status return, zero signals ok
mask:   db      0       ;
len:    db      0       ; storage space for input length

start:  ld      hl,(string)     ; get address of string to
                                ; compress
        ld      a,(hl)          ; get input string length
        ld      (len),a         ; store it away
        inc     hl              ; point at start of string
        ld      de,(output)     ; point at output space
        push    de              ; save start of output
        inc     de              ; leave 1 space for count
        ld      (output),de     ; save output address till
                                ; we get something to store
        ld      a,255
        ld      (mask),a        ; handle left nibble first

nchar:  ld      a,(hl)          ; get byte to work on
        call    check           ; make sure it's within range
        jr      nz,badchar      ; jr if character not allowed
        call    token           ; check if start of a token
        jr      z,gottoken      ; found a token so process it
        call    fourbit         ; check for a 4 bit character
        jr      z,got4bit       ; character is 4 bits so jump
        call    eightbit        ; get 8 bit character value
        push    af              ; no need to check - save value
        ld      a,0             ; while '8 bit char' indicator..
        call    writenib        ;..output
```

```
        pop     af              ; restore 2nd nibble of 8 bit val
got4bit:call    writenib        ; output 2nd nibble of 8 bit, or
                                ; 1st nibble of 4 bit, value
rejoin: inc     hl              ; point at next input character
        ld      a,(len)         ; get back current length
        dec     a               ; minus one for char just proc'd
        ld      (len),a         ; put back new length
        and     a
        jr      nz,nchar        ; jr to proc next char if exists
        ld      (status),a      ; signal finished ok, a will = 0
        ld      a,2             ; 2 = end of compressed text
        call    writenib        ; write it out
        ex      de,hl           ; final output address into hl
        pop     de              ; get output string address
        ld      a,(mask)        ; get mask value
        and     a
        jr      z,nodec         ; if mask zero then final value ok
        dec     hl              ; ignore final byte, it's null
nodec:  and     a               ; clear carry for subtraction
        sbc     hl,de           ; get length
        ld      a,l             ; get length - not more than 255!
        ld      (de),a          ; store count at begin of output
        ret                     ; return

badchar:pop     de              ; clear stack
        ld      a,255
        ld      (status),a      ; signal failure
        ret

gottoken:
        push    af              ; save token nibble
        ld      a,1             ; signal token coming
        call    writenib
        pop     af              ; restore nibble
        call    writenib        ; output token
        jr      rejoin

        ; subroutine to check for start of new token
token:  push    hl              ; save input address
        ex      de,hl           ; put input address into de
        ld      hl,tktable      ; point hl at table of tokens
        ld      a,(len)         ; get remaining length into a
        ld      c,a             ; and transfer it to c
        ld      b,0fh           ; initialize token number
tok1:   ld      a,(hl)          ; get length of current token
        and     a
        jr      z,notfound      ; if length=zero then endoftable
        push    de              ; save these pointers in case
        push    hl              ; they are needed
        inc     hl              ; point to 1st char in token
        push    bc              ; save token number
        ld      b,a             ; count into b
tok2:   ld      a,(de)          ; get input character
        cp      (hl)            ; compare with token character
        jr      nz,nxttoken     ; if no match forget this token
        inc     hl              ; point to next character
        inc     de
        dec     b               ; reduce remaining chars count
        jr      nz,chkchars     ; if b isn't 0 then jump
        ld      a,c
        ld      (len),a         ; store it away
        pop     bc              ; token found - restore token no.
        pop     hl              ; clear stack
        pop     hl              ;
        pop     hl              ;
        ex      de,hl           ; put new input address into hl
        dec     hl              ; and dec to keep main loop in step
        xor     a               ; set zero flag to signal success
        ld      a,b             ; get token number into a
        ret
chkchars:
        dec     c               ; reduce remaining character count
        jr      nz;tok2         ; if still chars to check go back
nxttoken:
        pop     bc              ; restore remaining chars+tok no.
        dec     b               ; next token number
        pop     hl              ; get back pointer to curr tok. no
        ld      a,(hl)
        ld      e,a             ; length into de
        ld      d,0
        inc     hl              ; step over count
        add     hl,de           ; hl now point at next tokens count
        pop     de              ; restore input address
        jr      tok1            ; jump to check next token

notfound:
        pop     hl              ; restore original input address
        ld      a,(hl)          ; restore character
        or      a               ; reset zero flag to signal failure
        ret


        ; subroutine to check for 4 bit characters

fourbit:push    hl              ; save input address
        ld      hl,tab4bit      ; point hl at character table
        call    tabscan         ; scan table for value
        pop     hl              ; restore input address
        ret                     ; go back

        ; subroutine to check for 8 bit characters
eightbit:
        push    hl              ; save input address
        ld      hl,tab8bit      ; point at 8 bit character table
        call    tabscan         ; scan table
        pop     hl              ; restore address
        ret                     ; return

        ; general purpose table scanning routine

tabscan:ld      b,0fh           ; 16 characters in each table
                                ; (one is numbered 0)
tabsc2: cp      (hl)            ; check character
        jr      z,tabsc3        ; return with 0 set if match
        inc     hl              ; check next table element
        djnz    tabsc2          ; loop till no chars in table
        or      a               ; reset 0 flag to signal no match
                                ; a contains char code <> 0
        ret                     ; go back
tabsc3: ld      a,b             ; put nibble into a
        ret

        ; routine to check for valid characters
        ; only space  , . ; : and upper case letters allowed
check:  cp      ' '             ; check for space first
        ret     z
        cp      ','             ; amd comma
        ret     z
        cp      '.'             ; finally full stop
        ret     z
        cp      'A'             ; assume contiguous alphebet
        jr      c,nogood        ; jump if 'less than' A
        cp      'Z'+1
        jr      nc, nogood
        ld      c,a
        xor     a               ; set zero to signal character ok
        ld      a,c
        ret
nogood: ld      a,255
        and     a               ; reset zero to signal failure
        ret

writenib:
        ld      c,a             ; save nibble
        ld      a,(mask)        ;
        ld      de,(output)     ; get address of byte to write to
        and     a               ; check mask value
        jr      nz,left         ; jump if nibble needs shifting
        ld      a,(de)          ; get old nibble
        or      c               ; insert new nibble
        ld      (de),a          ; and store it back
        inc     de              ; point to next byte
        ld      (output),de     ; save address
        ld      a,255
        ld      (mask),a        ; signal left nibble next time
        ret
left:   ld      a,c             ; get value into a
        sla     a                       ; shift it across
        sla     a
        sla     a
        sla     a
        ld      (de),a          ; amd store it
        xor     a
        ld      (mask),a        ; signal other nibble next time
```

```
        ret

; 4 Bit text expansion program for Z80, machine independent

expand: ld      hl,(string)     ; string contains address of
                                ; string to expand
        inc     hl              ; point hl at start of string
        ld      (string),hl     ; put back new string start
        ld      de,(output)     ; point at start of output space
        push    de              ; save for end
        ld      a,255
        ld      (mask),a        ; handle left nibble first

nextnib:call getnib             ; get next input nibble
        cp      2               ; is this the end
        jp      z,fin           ; if so jump to finish up
        jp      nc,exp4bit      ; if nibble > 2 process 4 bits
        and     a
        jp      z,exp8bit       ; if nibble=1 process 8 bit char
        call    getnib          ; get nibble to expand into token
        call    tokscan         ; find the one we want
tokloop:ld      b,a             ; count into b
tklp1:  ld      a,(hl)          ; get token character
        inc     hl              ; point to next character
        call    outchar         ; output to new string
        djnz    tklp1           ; loop for all the token chars
        jr      nextnib         ; jump back for next input nibble
tokscan:ld      hl,tktable
        ld      b,a             ; initialise b
        ld      a,0fh
        sub     b
        ld      b,a             ; count into b
        inc     b
toksc1: ld      a,(hl)          ; get lenght of this token
        inc     hl              ; point at first char in token
        dec     b               ; reduce count
        ret     z               ; return if it's the one we want
        ld      e,a             ; step over this token
        ld      d,0
        add     hl,de
        jr      toksc1          ; jump back

fin:    ld      hl,(output)     ; get end of output string in hl
        pop     de              ; start of the output string
        and     a               ; clear carry
        sbc     hl,de           ; length in hl
        ld      a,l             ; length mustn't be > 255
        ld      (de),a          ; put count in right place
        ret                     ; return to system

exp8bit:call    getnib          ; get next nibble for 8 bit char
        ld      hl,tab8bit      ; point hl at correct table
        call    index           ; get the right character
        call    outchar         ; put it in the output string
        jr      nextnib         ; go back for more

exp4bit:ld      hl,tab4bit      ; point at 4 bit table
        call    index           ; find character
        call    outchar         ; output it
        jr      nextnib         ; round again

index:  ld      e,a             ; initialise e
        ld      a,0fh
        sub     e               ; get offset into a
        ld      e,a             ; put into e
        ld      d,0
        add     hl,de           ; index into table
        ld      a,(hl)          ; get character
        ret

outchar:ld      de,(output)     ; get last address outputted to
        inc     de              ; point to new address
        ld      (de),a          ; store character
        ld      (output),de     ; put back new address
        ret

getnib: ld      a,(mask)        ; get mask value
        ld      hl,(string)     ; get address of input byte
        and     a
        ld      a,(hl)          ; get byte
        jr      nz,shfta        ; jump if left nibble required
        and     0fh             ; mask right nibble
        inc     hl              ; point to new byte
        ld      (string),hl     ; store for next time
        ld      c,a
        ld      a,255           ; signal left nibble next time
        ld      (mask),a
        ld      a,c
        ret
shfta:  sra     a               ; shift right nibble ove
        sra     a
        sra     a
        sra     a
        and     0fh             ; mask it off
        ld      c,a
        xor     a               ; signal right nibble next time
        ld      (mask),a
        ld      a,c
        ret

        ; table of values

        ; four bit table

tab4bit:db      ' '             ; space character
        db      'E'
        db      'T'
        db      'A'
        db      'O'
        db      'N'
        db      'R'
        db      'I'
        db      'S'
        db      'H'
        db      'D'
        db      'L'
        db      'F'
        db      0               ; dummy values to fill table
        db      0
        db      0

        ; 8 bit characters

tab8bit:db      'C'
        db      'M'
        db      'U'
        db      'G'
        db      'Y'
        db      'P'
        db      'W'
        db      'B'
        db      'V'
        db      'K'
        db      'X'
        db      'J'
        db      'Q'
        db      'Z'
        db      ','
        db      '.'
        ; token table
tktable:db      3,'THE'
        db      4,'THIS'
        db      4,'THAT'
        db      2,'IF'
        db      3,'YOU'
        db      2,'ME'
        db      3,'WAS'
        db      2,'HE'
        db      3,'SHE'
        db      4,'THEY'
        db      2,'OF'
        db      2,'IT'
        db      2,'IS'
        db      3,'FOR'
        db      2,'ON'
        db      2,'TO'
        db      0               ; end of table marker

        end
```

Mama Bell

Die Bell-Laboratories liefern seit Jahrzehnten entscheidende Entwicklungsbeiträge für die Hard- und Software. Und sie schrieben ein Stück Computergeschichte.

Schon vor hundert Jahren amüsierte sich Queen Victoria über eine neue Erfindung, dank der sie von der Isle of Wight im Ärmelkanal aus mit ihren Ministern im fernen London sprechen konnte. Vom ersten Telefongespräch der Königin bis hin zur weltweiten Satellitenkommunikation bedurfte es aber noch erheblicher Anstrengungen, wobei der Computer quasi als Abfallprodukt entstand. Zur Förderung der Fernsprechtechnik hatte die „American Telephone & Telegraph Company" (AT&T), eine Gründung Graham Bells, bereits 1925 in Murray Hill/New Jersey (USA) die „Bell Laboratories" ins Leben gerufen, die später unter dem Namen „Mama Bell" bekannt wurden.

Der Schwerpunkt dieser Einrichtung mit über zwanzigtausend Mitarbeitern liegt in der Grundlagenforschung. Die Wissenschaftler des Labors werden von den Problemen der Tagesentwicklung mit Bedacht abschirmt. Hervorragende Fachleute können tun und lassen, was Ihnen für ihre Forschung wichtig erscheint.

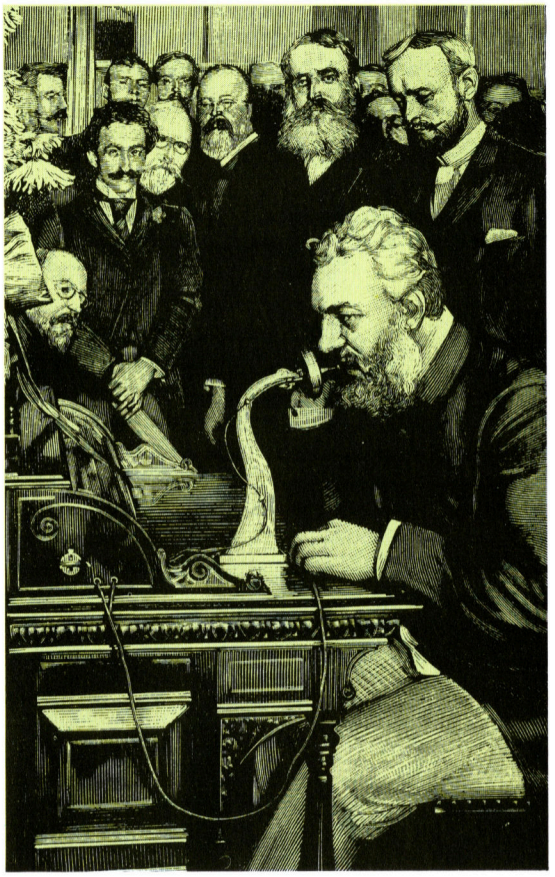
Nobelpreise

Kommen dann dabei innovative Entdeckungen heraus, hat sich der Einsatz des Geldes für das Unternehmen gelohnt. Diese Philosophie brachte für die forschenden Wissenschaftler tatsächlich mehrere Nobelpreise und zwanzigtausend Patente auf den verschiedensten Gebieten ein. Für die Entwicklung des Computers waren viele Arbeiten der Forscher dieses Unternehmens von großer Bedeutung.

In den dreißiger Jahren wurde zunehmend die Fernsprechtechnik weiterentwickelt; vor allem wurde die Vermittlungs-Automatisierung vorangetrieben. Der Verbindungsaufbau erfolgte bereits mit Hilfe eines digitalen Wählcodes, der als eine Folge von Rechteckimpulsen beim Ablauf der Wählscheibe erzeugt und an die Zentrale gesendet wurde. Die Information übernahm dort zunächst ein Relaisregister, bis die Verbindung über ein Kreuzschienensystem durchgeschaltet war; die Wählimpulse mußten gezählt und in Koordinaten auf elektromechanischen Schaltfeldern umgesetzt werden. Mit diesem Vermittlungssystem standen die wesentlichen Rechnerkomponenten eigentlich schon bereit – es mußte nur noch der rechte Mann kommen.

Dieser Mann war der Bell-Mathematiker Georg Stibitz, dem die Analogie zwischen dem „Zählen" der Wählimpulse und dem Vorgehen beim Addieren auffiel. Auf seinem Küchentisch bastelte er aus einem alten Koordinatenwähler und ein paar Relais die ersten elektromechanischen Computer-Funktionsgruppen.

Stibitz tat sich dann mit dem erfahrenen Vermittlungs-Ingenieur Samuel B. Williams zusammen, der schon seit Jahrzehnten die Kreuzpunktschalter im Kopf hatte, und baute mit ihm den „Complex Number Calculator". Dieses Gerät war eine Rechenmaschine für „komplexe" Zahlen, die ja bekanntlich imaginäre Anteile enthalten und bei der Lösung von Polynomgleichungen gebraucht werden. Der Bau des Apparates begann 1937, und man benötige 450 Relais sowie zehn Kreuzschienenfelder, bis er am 8. Januar 1940 erstmals arbeiten konnte. Er arbeitete mit binärer Codierung und benötigte für die Division bei zwei achtstelligen Zahlen eine halbe Minute. Im September 1940 wurde diese Rechengeschwindigkeit der Fachwelt vorgeführt.



Die Bell Laboratories sind nach dem Sprachheillehrer Alexander Graham Bell benannt, der als schottischer Auswanderer in Amerika 1876 das erste brauchbare Telefon erfand. Das Prinzip der elektrischen Tonübertragung hatte schon 1861 der Deutsche Philipp Reis vorgeführt. Nach der Überlieferung bestand das erste Telefongespräch Bell's aus dem nachfolgenden bedeutenden Satz, den er an den Assistenten im Keller gerichtet hatte: „Kommen Sie mal rauf, Mr. Watson, ich brauche Sie hier!"

Befehlsempfänger

Wir behandelten die Datenübertragungsbefehle und einfache Arithmetikanweisungen des Motorola 68000. Bevor wir weitergehen, müssen wir herausfinden, wie die Vergleichsbefehle des 68000 arbeiten.

Der Befehl CMP (compare – vergleichen) ist der SUB-Anweisung sehr ähnlich. Sehen Sie sich folgende Programmstruktur im Pseudo-Code an:

```
if Eingabezeichen = 'N' then
   Array-löschen
end
```

Die IF-Entscheidung wird in der Implementierungsphase des Programms als

```
CMP.B #'N',D0      Vergleiche Eingabebyte
                   mit 'N'
BNE UNGLEICH       Auf UNGLEICH verzweigen, wenn Eingabebyte
                   ungleich 'N'
```

Der CMP-Befehl subtrahiert zwar die Quelle vom Ziel, verändert aber nur die Bedingungscodes. Der Vorgang hat keinen Einfluß auf das Ziel (wie beispielsweise SUB). Wenn das Ergebnis der Subtraktion von D0 (das heißt 'N') nicht gleich Null ist, dann veranlaßt die bedingte Verzweigung – BNE – eine Verzweigung auf das Label UNGLEICH.

Hier weitere Beispiele mit anderen Datenattributen:

```
CMP.W SPEED,D3     Vergleiche den Wortinhalt der Speicherstelle SPEED mit D3
CMP.L D1,D2        Vergleiche die beiden Langwörter in D1 und D2
```

Die Ziele dieser beiden Beispiele sind Datenregister. Dabei kann zur Adressierung der Quelle jeder beliebige Modus eingesetzt werden. Wenn Sie eine andere Art Ziel adressieren wollen, müssen sie andere CMP-Befehle verwenden, zum Beispiel:

```
CMPA BETTY,A3      Vergleiche den Inhalt der Speicherstelle
                   BETTY mit A3
```

Dabei lassen sich zwar alle Adressierungsarten für Quellen einsetzen, doch sind im unmittelbaren Format nur datenverändernde Modi möglich:

```
CMPI #3, (A4)      Vergleiche, ob A4 als Pointer auf 3 zeigt
```

Schließlich verdient noch die Variante CMPM besondere Aufmerksamkeit. CMPM (A2)+, (A3)2 vergleicht beispielsweise den Inhalt der Speicherstellen, auf die A2 und A3 zeigen, und nach-inkrementiert dann die Pointer. Damit kann ein einziger Befehl die gespeicherten Schlüsselwörter mit den Zeichen eines Eingabebuffers vergleichen:

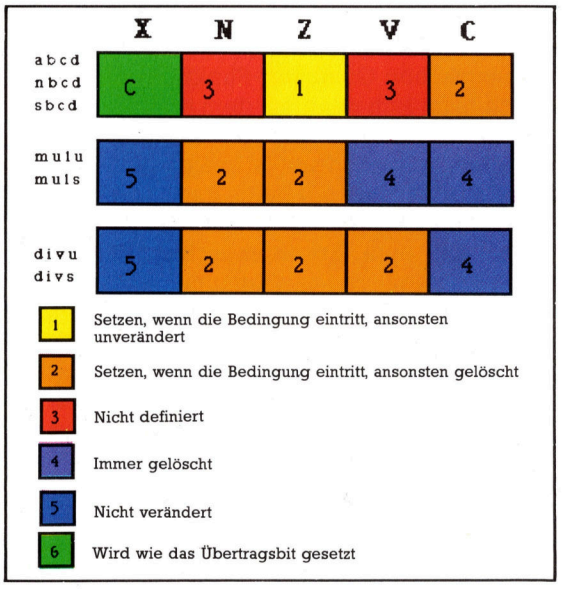
```
LEA KEYS,A2        Den ersten Pointer auf die Schlüsselwörter setzen
LEA BUFFERS,A3     Bufferpointer anlegen
CHECK CMPM.B       Die beiden Speicherstellen vergleichen
BEQ CHECK          Prüfen, bis ungleich
```

Das gesamte Programm belegt nur sieben Wörter.

Bei CMPM ist nur die Adressierung mit Nach-Inkrementierung möglich. Für andere Formate können Sie einen Operanden in ein Datenregister laden und dann CMP einsetzen.

NEG und EXT sind zwei einfache, aber sehr wichtige Arithmetikbefehle. NEG (Negation) zieht das Datenregister des Operanden von Null ab, das heißt, es bildet aus dem Registerinhalt den negativen Wert des Zweierkomplements (hier sind keine weiteren Adressierungsmethoden möglich). Wenn Sie mit D0 = 1111 1010 (dezimal –6) NEG.B D0 ausführen, dann enthält D0 am Ende 0000 0110 (dezimal 6). Mit diesem Befehl wird oft der absolute Wert von Datenoperanden gebildet: Zuerst wird festgestellt, ob ein negativer Wert vorliegt, und dann negiert. Eine erweiterte Form dieses Befehls (NEGX) schließt das X-Bit in den laufenden Vorgang mit ein.

Mit EXT (Vorzeichenerweiterung) wird das Vorzeichenbit des Datenoperanden auf die nächstgrößere Operandengröße erweitert.



Das nebenstehende Bild zeigt die Auswirkungen der DIV-, MUL- und BCD-Befehle auf das Statusregister. Fehlt ein Bedingungscode, so bedeutet das jedoch nicht, daß das entsprechende Flag auch gelöscht ist. Wenn Sie einen (von einem vorangehenden Ablauf gesetzten) Bedingungscode testen, erhalten Sie falsche Werte.

EXT.W D0 (mit D0 = 11110101) ergibt FFFA (hex). Dieser Befehl läßt sich gut für Zahlen mit doppelter Genauigkeit einsetzen und besonders für große Operanden bei Multiplikations- und Divisionsbefehlen.

Bevor wir uns den komplizierteren arithmetischen Anweisungen zuwenden, müssen wir untersuchen, was die binären Bitmuster der Datenbytes überhaupt bedeuten. So könnten wir die Zahl 1001 0110 (z. B. in D0 gespeichert) als Ganzzahl mit dem Werk −106 ansehen (der Wert entsteht, wenn Sie die Zahl umkehren und Eins addieren). Wenn die Zahl jedoch nicht vorzeichenbehaftet ist und der gesamte Binärbereich von positiven Ganzzahlen belegt wird, dann ergibt sich hexadezimal 96 (dezimal 150). Zahlen ohne Vorzeichen sind praktisch, wenn nur ein großer positiver Zahlenbereich gebraucht wird. So läßt sich beispielsweise der Adreßbereich eines Computers als ein Bereich positiver Zahlen ohne Vorzeichen ansehen. Wenn Sie diese Zahlen nicht mit Operatoren im Zweierkomplement bearbeiten, gibt es keine Probleme.

Illegale Bitfolgen

Und es gibt noch eine weitere Interpretation der Bitmuster: das BCD-System (binär codierte Dezimalzahlen). Bei diesem praktischen Datencode werden Dezimalzahlen in jeweils vier Bits codiert. Unser Beispiel (D0 = 1001 0110) würde daher den BCD-Wert von 96 ergeben (1001 = 9 und 0110 = 6).

Selbst große Zahlen lassen sich leicht in BCD-Zahlen umwandeln und umgekehrt. Die Dezimalzahl 9631 sieht im BCD-Code so aus: 1001 0110 0011 0001.

Ebenso einfach wird die Rechengenauigkeit definiert. Dabei sind beim BCD-Code bestimmte Bitfolgen illegal. Da BCD nur die Zahlen von 0 bis 9 codiert, gibt es keine Verwendung für die Codes zwischen 1010 (dezimal 10) und 1111 (dezimal 15).

Bei der Behandlung der Arithmetikbefehle des 68000 werden Sie sehen, wie wichtig dabei die Interpretation der Bitmuster ist. Untersuchen wir zunächst den Ablauf einer binären Multiplikation. Wenn zwei 16-Bit-Operanden multipliziert werden, kann das Bitmuster des Ergebnisses 32 Bits lang sein. Für eine Wortlänge von n ist das Produkt der größten Zahl, $2^{(n-1)}$, die doppelte Länge des ursprünglichen Wortes oder $2^{2(n-1)}$.

Binäre Multiplikationsbefehle haben daher zwei 16-Bit-Operanden und ein 32-Bit-Ergebnis. Die Verarbeitung von Zahlen mit und ohne Vorzeichen geschieht über zwei unterschiedliche Befehle – MULU (multiplizieren ohne Vorzeichen) und MULS (multiplizieren mit Vorzeichen). Beide Befehle multiplizieren die Operanden und erzeugen in dem Zielregister ein Ergebnis im 32-Bit-Format, wobei der Operand nur mit Datenadressierungsarten angesprochen werden kann, damit der Computer den Befehl ausführt.

MULU #20,D0 für D0 = XXXX 0003 (X kann 0 oder 1 sein) ergibt D0 = 0000 003C. Dabei wird das gesamte 32-Bit-Datenregister eingesetzt, obwohl es in diesem Fall nicht nötig ist.

Auf ähnliche Weise erhalten Sie bei MULU #$10,D0 für D0 = XXXX FFFF das Ergebnis D0 = 000F FFF0. Der Befehl MULS ergibt D0 = FFFF FFF0, da sich das vorzeichenbehaftete Ergebnis auf 32 Bits erstreckt. Die Resultate lassen sich leicht prüfen, da der Multiplikator 10 (Hex) einer vierfachen Linksverschiebung oder einer Multiplikation mit 16 entspricht.

Auch die Bedingungscodes sind bei der Multiplikation wichtig. So werden die Flags N und Z je nach Ergebnis gesetzt und die Bits V und C auf Null gestellt. Obwohl Sie bei Operanden mit Wortlänge und einem Langwortergebnis nie einen Überlauf erhalten können, lohnt sich die Feststellung, ob das Ergebnis die Wortlänge überschreitet (damit es, falls nötig, auf die von Ihnen gewünschte Wortlänge gekürzt werden kann).

Das folgende Programmbeispiel zeigt, wie der Befehl MULS für die Umwandlung von dezimalen ASCII-Codes in Binärwörter eingesetzt wird. Als erstes muß das Eingabezeichen in Binärformat umgewandelt und dieses Bitmuster dann in einen binären Akkumulator addiert werden. Vor der Addition muß jedoch sichergestellt sein, daß die vorige Eingabe mit 10 multipliziert wurde (da sie im Dezimalformat erfolgt). Hier der Code:

```
SUB.B #'0',D0   Dezimalen Zeichencode in
                Binärformat wandeln
MULS #10,D5     Die alte Summe mit dezimal
                10 multiplizieren
ADD.W D0,D5     Neuen Wert addieren
```

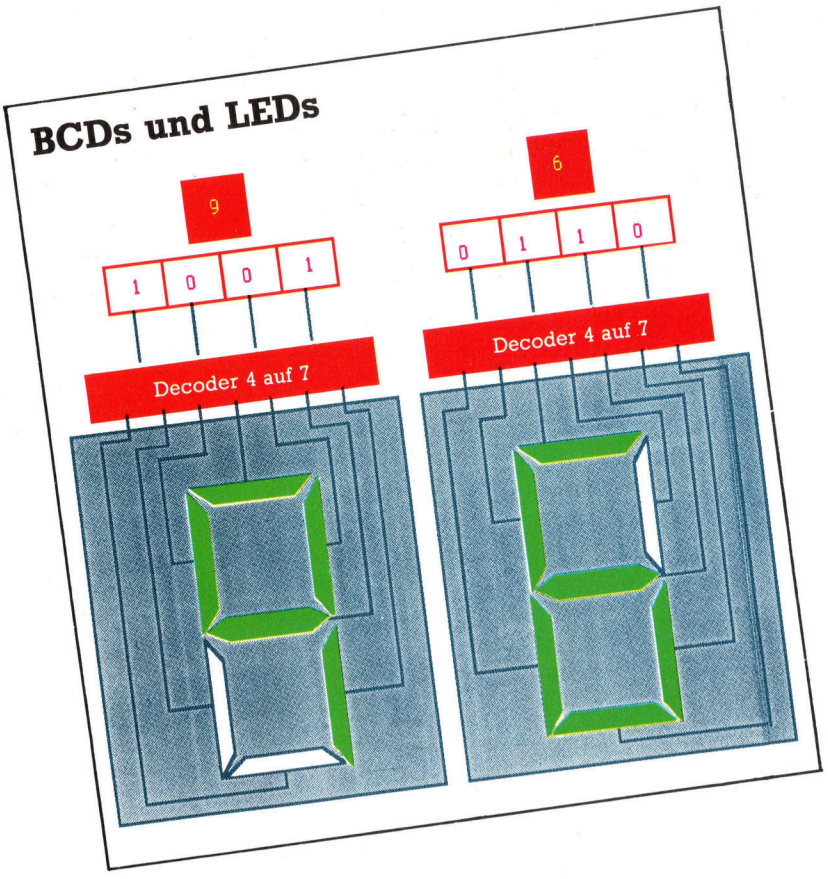
In diesem Beispiel befindet sich das Eingabezeichen in D0 und das neue Binärwort mit dem binären Zeichencode in D5.

Gute Vorzeichen

Auch für die Subtraktion gibt es zwei Befehle – DIVU für Zahlen ohne Vorzeichen und DIVS für Zahlen mit Vorzeichen. Beide Befehle nehmen die Ganzzahl mit 32-Bit-Format im Zieldatenregister und teilen sie durch den Quelloperanden im 16-Bit-Format. Das Ergebnis wird in den niederwertigen 16 Bits des Zieldatenregisters gespeichert, während der Rest in den höherwertigen 16 Bits erscheint. Bei D0 = 0000 0005 (Hex) ergibt

```
DIVU #3,D0
```

D0 = 0002 0001 (das heißt 1 Rest 2). Da der Zieloperand 32 Bits lang ist, sollte, falls nötig, zuvor das entsprechende Langwort gelöscht oder mit EXT.L eine Langworterweiterung angegeben werden. Da Überläufe auftreten können (das Ergebnis der Teilung eines 32-Bit-Wortes durch Eins kann durchaus das 16-Bit-Format überschreiten), muß in bestimmten Fäl-



Der Befehlssatz des 68000 enthält nicht nur Codes für Zahlen mit und ohne Vorzeichen, sondern auch für die Verarbeitung von BCD-Formaten (binär codierte Dezimalzahlen). Diese Zahlen sind besonders praktisch, wenn die größtmögliche mathematische Genauigkeit gefordert ist. Auch in anderen Anwendungsbereichen, wie beispielsweise in der Steuerung von siebenteiligen LED-Anzeigen, hat das BCD-Format Vorteile. So ist die (im Bild gezeigte) Decodierung von vier auf sieben Bits einfacher als die Umwandlung von Binärzahlen.

len von Ihnen auch das Überlaufbit berücksichtigt werden.

Hier ein einfacher Algorithmus, der den Durchschnitt aller Zahlen einer Wortliste bildet. Die Liste endet mit einer Null:

```
                ORG $1000
        LEA     LIST1,A0        Adresse
POINTER                         anlegen
                                Ergebnisregister
        CLR.L   D0              löschen
        CLR.L   D1              Schleifenzähler
                                löschen
LOOP    ADDQ    #1,D1           Wortzahl zählen
        ADD.W   (A0)+,D0        Summe bilden
        TST     (A0)            Auf Ende der
                                Liste prüfen
        BNE     LOOP
        DIVU    D1,D0           Summe durch
                                Zähler teilen
        TRAP    0
LIST1   DC.W    1,2,3,4,0
```

Im Initialisierungsteil des Programms werden Langwörter gelöscht, da die Summe das 32-Bit-Format hat. D1 ist der Zähler, und A0 zeigt auf das Wort, das mit nach-inkrementierter Adressierung auf D0 addiert wird. Danach wird das nächste Wort von LIST1 mit TST getestet. Dieser Befehl setzt nur die Bedingungscodes für BNE, ändert den Operand aber nicht.

Der Befehl TST ist nötig, da das vorangegangene ADD die Bedingungscodes nach dem Ergebnis der Addition des Datenoperanden auf D0 setzt, nicht aber nach dem Wert der Daten, die über den Pointer A0 geladen werden. Der Befehl BNE (verzweige bei ungleich Null) veranlaßt einen Rücksprung auf LOOP, wenn das nächste Listenelement nicht Null ist. Nach dem Auftreten einer Null enthält D0 die Summe und D1 die Zahl der Elemente, die nicht Null sind.

Nach vier Schleifendurchgängen stehen in den Datenregistern folgende Werte:

D0 = 0000 000A (oder dezimal 10)
D1 = 0000 0004

Nach Ausführung von DIVU enthält D0 den Wert 0002 0002 (10 / 4 = 2 Rest 2).

Ein letzter Hinweis: Eine Teilung durch Null löst ein „Trap" (ein Softwareinterrupt im Systemmonitor) aus, da Unendlichkeit sich in 16 Bits nicht darstellen läßt. Sie können statt des Trap auch prüfen, ob der Divisor Null ist. Hier ein Beispiel:

```
TST   D1
BEQ   ERROR
DIVU  D1,D0
```

Wie praktisch das BCD-Format für die Codierung von Dezimalzahlen sein kann, hatten wir schon erwähnt. Hierbei entspricht eine legale BCD-Stelle einer Hexadezimalstelle (4 = binär 0100 = 4 hex = 4 BCD). BCD-Konstanten werden daher auf die gleiche Weise gesetzt wie Hex-Konstanten:

```
MOVE.B #$54,D0 speichert BCD 54 in D0
```

Doch hier endet die Ähnlichkeit auch schon. Wenn Sie zwei BCD-Stellen mit binärer Arithmetik addieren, erhalten Sie ein falsches Ergebnis

```
0100    1001    (BCD 49) binär addiert
0000    0001    (BCD 1)
---------------
0100    1010
```

Dabei entsteht für die niederwertige BCD-Stelle ein illegaler BCD-Code. Der 68000 bietet für diesen Zweck einen Satz von BCD-Befehlen: ABCD (Addition), SBCD (Subtraktion) und NBCD (Negation).

Der BCD-Befehl ABCD addiert das Quellenbyte (je zwei BCD-Zahlen) mit dem Bit X auf das Zielbyte und speichert dort auch die Summe. Hier sind als Adressierungsarten jedoch nur Datenregisterpaare und vor-dekrementierte Datenregisterpaare erlaubt. Bei D0 = 44 BCD und D1 = 01 BCD und nach Ausführung des Befehls

```
ABCD D0,D1
```

enthält D1 45 BCD. Wenn das Bit X des SR gesetzt ist (entspricht dem Übertrag von BCD) würde das Ergebnis 46 lauten. In ähnlicher Weise ergibt die Addition von 1 auf 99 in D0 den Wert 00 mit dem gesetzten Bit im Statusregister. Zwar lassen sich nur Byteoperanden angeben, doch ermöglicht Bit X eine hohe Rechengenauigkeit, da Überläufe in höherwertige Bytekomponenten übernommen werden.

Die Bedingungscodes X, C und Z werden bei allen BCD-Befehlen gesetzt. Beachten Sie jedoch, daß der Befehl NBCD (negiere BCD) beim Zieloperanden datenverändernde Adressierungsarten zuläßt.

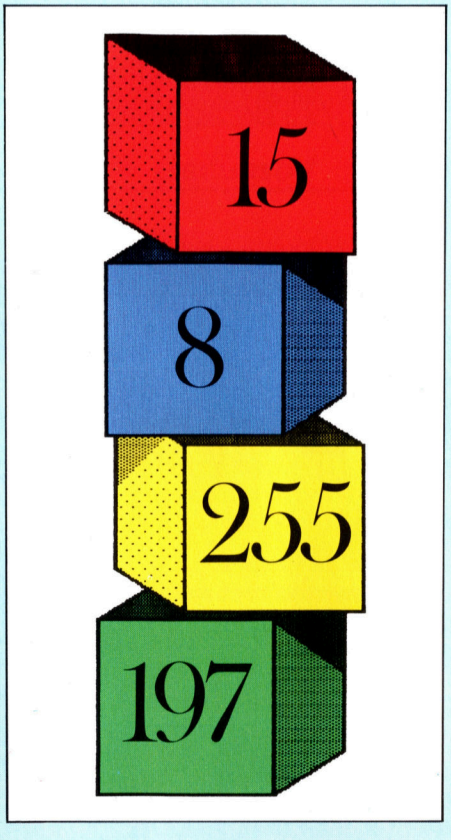
Fachwörter von A bis Z

Source Code = Quellprogramm
Ein Quellprogramm ist in einer Compiler- oder Assemblersprache geschrieben. Daraus muß vor Ablauf im Rechner durch Übersetzen in den Maschinencode ein „Objektprogramm" erzeugt werden. Das Quellprogramm, z. B. in PASCAL oder COBOL, wird zunächst einer syntaktischen Prüfung unterzogen; treten dabei keine Fehler auf, folgt die Compilierung. Auch ein Assemblerprogramm mit seinen mnemotechnischen Befehlen und symbolischen Adressen stellt ein Quellprogramm dar, das durch Assemblieren in die Maschinensprache umgesetzt wird.

Split Screen = Fenstertechnik
Bei dieser Technik wird der Bildschirm in zwei oder mehr Felder unterteilt, so daß oben auf dem Schirm ein Menü stehen kann, während darunter der Text dargestellt wird. Die untere Schirmhälfte läßt sich dabei unabhängig von der oberen bearbeiten, und der Benutzer kann auch meist beliebig zwischen den Fenstern hin und her springen. Das ist beispielsweise beim Editieren langer Schriftstücke sehr praktisch, weil ältere Passagen parallel zum Neuentwurf überarbeitet werden können. Besonders bei Abenteuerspielen finden Sie jetzt immer häufiger auch einen geteilten Grafikschirm.

Spooler = Spooler
Systemprogramme, die den Datenfluß an Peripheriegeräte (meist Drucker oder externe Speicher) auffangen und nach dem Warteschlangenprinzip in eigener Regie verwalten, heißen „Spooler". Sie sind vor allem deshalb zweckmäßig, weil die Ausgabegeschwindigkeit des Prozessors sehr viel größer ist als die Aufnahmefähigkeit der Peripherie. Wenn Daten dann zunächst in einen Puffer gesteckt werden, braucht die CPU nicht auf das externe Gerät zu warten. Zumeist arbeitet der Spooler unabhängig von der CPU und läßt sich direkt vom zugehörigen Peripheriegerät mitteilen, wann wieder neue Daten gesendet werden können.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.



„Stacks" oder Stapelspeicher spielen in der Datenverarbeitung für das kurzfristige Deponieren von Informationen eine wichtige Rolle. Sie sind besonders einfach zu handhaben. Beim LIFO-Stack ist wie beim Würfelstapel aus dem Baukasten der Zugriff nur von oben her möglich. Die Reihenfolge der gespeicherten Größen bleibt vom Eintragen bis zum Auslesen unverändert, was für die Verwaltung der Rücksprungadressen bei verschachtelten Unterprogrammaufrufen ideal ist. Eine einfache Einsatzmethode für nicht so routinierte Anwender.

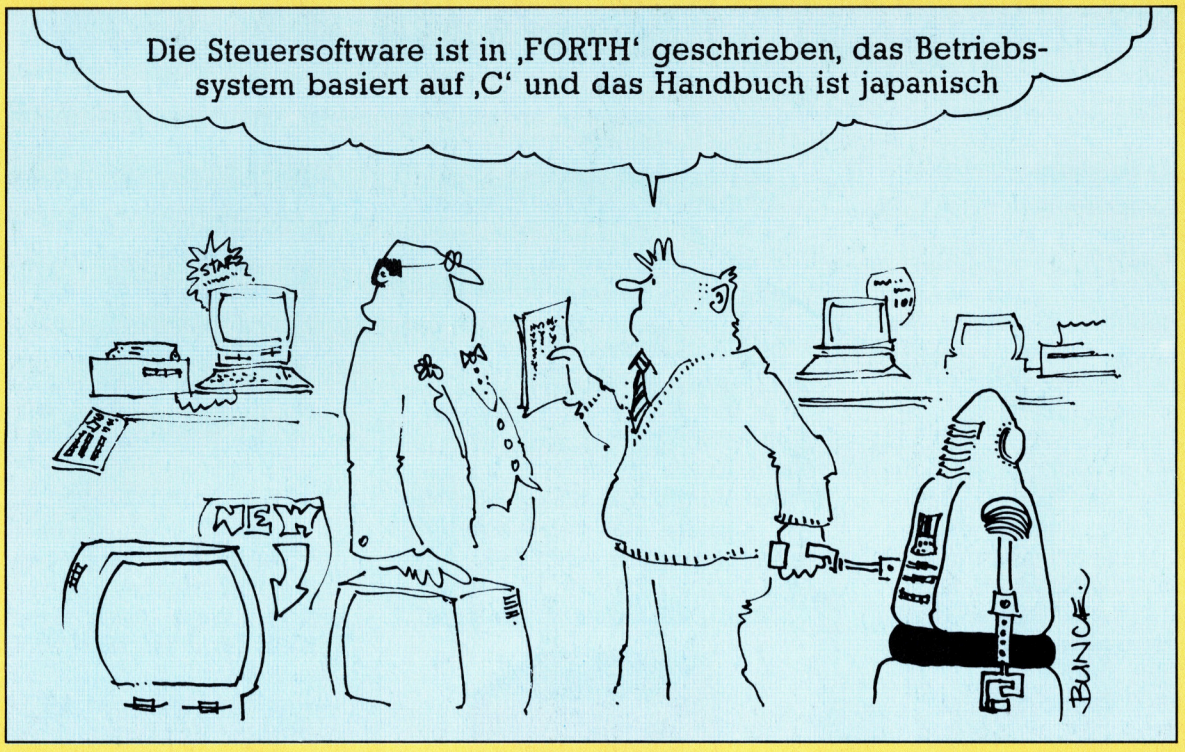
Square Wave = Rechtecksignal
Ein elektrisches Signal, das periodisch zwischen zwei Spannungspegeln hin- und herspringt, wird als „Rechtecksignal" bezeichnet. Eine solche Impulsfolge ist als Muster von binären Nullen und Einsen interpretierbar und dient zur Informationsübertragung, beispielsweise zwischen CPU und der Peripherie.

Stack = Stapelspeicher
Der „Stack" ist als spezieller Speicherbereich der CPU eines Computers zugeordnet und wird zur kurzfristigen Ablage von Registerinhalten und anderen Informationen benutzt. Die Kapazität des Stack und seine Lage im Arbeitsspeicher hängt vom jeweiligen Prozessor ab: Beim 6502 liegt der Stack auf der Nullseite (Zero Page) und faßt nur 256 Byte, beim Z80 dagegen können ihm beliebige Adressen und bis zu 64 KByte zugeteilt sein. Der Stack ist aber stets in gleicher Weise organisiert und geordnet.

Jeder Prozessor verfügt über besondere Stack-Befehle zum Eintragen (PUSH) und Entnehmen (POP) von Information. Beim üblichen LIFO-Stack werden die Daten „aufeinander" gestapelt; die letzten Einträge werden zuerst ausgelesen.

An diesem Prinzip kommt man beim Umgang mit dem Stack kaum vorbei: Die tieferstehende Information ist nur zugänglich, wenn der Stack von oben her weit genug abgearbeitet ist. Für den Zugriff hält ein bestimmtes CPU-Register, der „Stack Pointer" (Stapelzeiger), stets als aktuelle Adresse die der zuletzt angesprochenen Speicherzelle bereit. Der Stapelzeiger wandert beim Füllen und Leeren des Stack automatisch auf- bzw. abwärts, kann aber seitens des Programmierers bei Bedarf auch verschoben werden. Stack-Manipulationen sind nur im Maschinencode vorgesehen, aber es gibt sie auch in FORTH.

Bildnachweise
2213: Kevin Jones
2214: Liz Heany
2215–2217, 2240, U3: Caroline Clayton



+ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +

computer kurs

Heft 81

Die Quahl der Wahl

haben die meisten Hobby-Anwender, wenn sie die richtige Sprache für ihre Programme suchen. Eine gründliche Analyse des Vorhabens hilft ihnen, die Wahl zu erleichtern.

. . . hat man Töne

kann man da nur sagen. Ja, wir haben, und wir zeigen Ihnen, wie Sie sie erzeugen können. Das Schneider-CPC-Betriebssystem enthält dazu viele Möglichkeiten.

Tee oder Computer

ist eine ungewöhnliche Geschichte, die an einem ungewöhnlichen Ort entstand.

Jedem das Seine

möchte man hier sagen. Und damit Sie auch dazu kommen, bietet Ihnen UNIX eine Benutzerschnittstelle „nach Wunsch".