Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 83

Ein wöchentliches Sammelwerk

ROM

RAM

Zwerg im Computer

Die Zukunft hat begonnen

Wie Texte wachsen

Neue Schneider-Befehle

computer kurs

Heft 83

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.

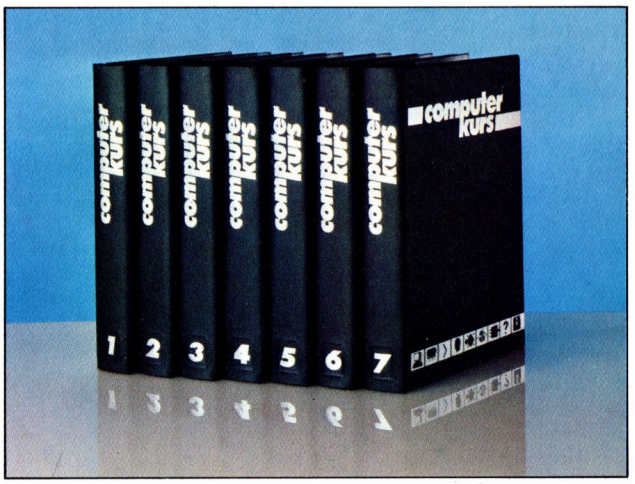
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

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

Mit sanfter Hand

Auch am Heimcomputer kann die Beherrschung des Zehnfingersystems wichtig sein. Was man früher als Geheimwissenschaft der Sekretärinnen abgetan hat, ist heute auch für Journalisten, Setzer und Programmierer unverzichtbar.

Die Fähigkeit, die Tastatur mit allen zehn Fingern zu bedienen, vereinfacht einen oft arbeitsintensiven, zeitraubenden Vorgang. Auch die Fehlerrate läßt sich beim „professionellen" Tippen deutlich senken.

Maschineschreiben wird traditionell mit Büchern oder durch Kurse erlernt, wobei die Lehrmethoden sehr unterschiedlich sind. Inzwischen gibt es eine ganze Reihe von Programmen, bei denen das Tippen direkt am Rechner geübt werden kann. Unabhängig von der verwendeten Methode gibt es vier Grundfähigkeiten, die für schnelles und richtiges Maschineschreiben unabdingbar sind:

- Das Beherrschen der Tastatur
- Augentraining am Bildschirm oder auf Papier
- Exaktheit
- Geschwindigkeit

Diese Fähigkeiten lassen sich nicht im Handumdrehen erlernen, sie erfordern Ausdauer und Disziplin. Ohne Training geht es dabei nicht – Buchstabenfolgen werden so lange angeschlagen, bis die Finger nahezu automatisch arbeiten.

Die normale DIN-Tastatur wird nach den ersten fünf Buchstaben der zweiten Tastenreihe auch als QWERTZ-Tastatur bezeichnet. Die einzelnen Buchstaben sind dabei entsprechend der Häufigkeit des Gebrauchs angeordnet. Beim Üben wird jede Tastenreihe in Buchstaben für die rechte und für die linke Hand eingeteilt.

Das Erlernen der Zehnfingertechnik beginnt meist mit dem Kennenlernen der ersten acht Tasten in der mittleren Buchstabenreihe. Die Finger sollen zu Anfang über diese Tasten in die „Grundstellung" gehen und nach Betätigung anderer Tasten auch wieder dorthin zurückkehren. Grundstellung für die linke Hand sind die Tasten asdf, für die rechte jklö. Wenn man diese Tasten problemlos greift, wird die Lage der anderen Buchstabentasten im Verhältnis zur Grundstellung eingeübt. Ungewohnt ist zu Anfang die Funktion der kleinen Finger, die wie alle anderen Finger auch einen eigenen Tastaturbereich abdecken sollen. Sind die Buchstabentasten bekannt, geht es weiter mit der Leertaste (mit den Daumen zu betätigen) und den Umschalttasten für die

Sight and Sound bietet Zehnfinger-Schreibkurse für Neulinge an und kann auf große Erfolge verweisen. Hier im Bild sehen Sie ein Lehrstudio, in dem die Schüler mit einer audiovisuellen Technik zunächst die Grundbegriffe und später dann das perfekte Schreiben auf der Schreibmaschine erlernen.

Großbuchstaben bzw. den Tasten mit zwei verschiedenen Zeichen.

Die Zifferntasten werden leider nur sehr selten in Lehrbüchern oder Schreibmaschinenkursen behandelt. Für den Computer-Anwender ist das ungünstig – beim Programmieren stellen gerade diese Tasten einen sehr wichtigen Teil des Keyboards dar. Wer das Arbeiten aus der Grundstellung erst einmal gemeistert hat, wird aber auch hierbei keine Probleme haben. Die Finger der linken Hand greifen einfach über die alphabetischen Tasten hinweg und decken die Ziffern Eins bis Sechs ab, die rechte Hand ist für die Sieben bis Null und die folgenden Symboltasten zuständig.

Augentraining

Eine interessante Weiterentwicklung des Arbeitens aus der Grundstellung kann auch die Nutzung der Grafikzeichen sein, falls diese direkt über die Tastatur eingegeben werden können. Sie sollten versuchen, auch die Grafikzeichen gleich mit in Ihr Übungsprogramm zu integrieren.

Das Augentraining auf dem Bildschirm bzw. auf dem Papier muß gleichzeitig mit dem Tippen erlernt werden. Die Abdeckung der Tasten mit Klebestreifen oder speziell geformten Hütchen kann dabei für Anfänger sehr nützlich sein – die Versuchung fällt weg, zwischendurch doch einmal auf die Tastatur zu blicken. Computer mit Bildschirm eignen sich besser zum Lernen als die klassische Schreibmaschine und Papier: Der Bildschirm liegt in Augenhöhe – das dämpft ein wenig den Wunsch, schnell einmal nach unten zu sehen. Außerdem lassen sich Fehler sofort beseitigen, was zu größerer Genauigkeit anhält.

Genauigkeit ist nur durch Konzentration und Übung erreichbar. Die Tasten müssen mit einem festen, schnellen Schlag betätigt und exakt getroffen werden. Außerdem soll der Anschlag regelmäßig sein und einen Rhythmus bilden, der einem nach und nach in Fleisch und Blut übergeht. Schon bald spürt man es, wenn beim Tippen ein falscher Buchstabe getroffen wurde. Der Schreibrhythmus trägt dazu bei, die Geschwindigkeit stetig zu steigern.

Die Anschlaggeschwindigkeit wird mit Tests ermittelt – auf Tempo wird erst dann hingearbeitet, wenn die ersten drei Ausbildungsstufen bewältigt sind. Professionelle Maschineschreiber erreichen bis zu 100 Wörter pro Minute. Der Anfänger kann zufrieden sein, wenn er auf 30 Wörter pro Minute kommt.

Es gibt eine Vielzahl unterschiedlicher Lehrbücher. Es haben sich mehrere Lehrmethoden herausgebildet, die aber alle auf den erwähnten Grundprinzipien beruhen. Inzwischen existieren auch Versionen, bei denen von Anfang an wortweise getippt werden soll. Weil in den ersten Versuchen kurze und sehr gebräuch-

Die Grafik zeigt die Grundstellung der Hände über der Tastatur und die den einzelnen Fingern zugeordneten Tasten. Dabei müssen die Zeige- und kleinen Finger den größten Bereich abdekken.

liche Wörter geschrieben werden, muß man sich nur sehr wenig um die Rechtschreibung kümmern und kann sich voll auf das Erlernen der Schreibtechnik konzentrieren.

Mit dem Schreibmaschinenkurs eines Anbieters, der auf den Theorien des amerikanischen Psychologen Skinner beruht, sollen bereits 11jährige Kinder in weniger als zehn Stunden das Zehnfingersystem gelernt haben. Bei dieser Methode werden die Seiten des Lehrbuchs selbst in die Maschine gespannt. Unter die Fragen des Textes sollen jeweils die entsprechenden Antworten getippt werden. Das entspricht etwa dem traditionellen Vorgehen, bei dem der Schüler Übungstexte genau abtippen soll. Mit zusätzlichen Illustrationen wird die korrekte Handstellung beim Schreiben erläutert.

Eine andere Schule verbindet die traditionellen Methode mit modernen audiovisuellen Techniken: „Sight and Sound" unterhält seine Trainingszentren in vielen Großstädten des In- und Auslandes. Hier wird mit einer Technik gearbeitet, die Leuchtanzeigen und Tonbandaufzeichnungen kombiniert. Bei der audiovisuellen Methode soll durch die gleichzeitige Stimulation mehrerer Sinne (Sehen und Hören) das sofortige Reagieren eingeübt werden. Auf einer großen Leuchttafel erscheint ein blinkender Buchstabe. Ein „Jetzt" vom synchronisierten Tonband bestimmt den Moment des Anschlages.

Mittelfinger

Zeigefinger

Ringfinger

Kleiner Finger

Software-Überblick

Computerprogramme zum Erlernen des Zehnfingersystems sind entweder spiel- oder textorientiert. Die auf Text gegründete Software legt das Schwergewicht auf Übung und Wiederholung. Spielorientierte Lernpakete nutzen eher die Grafik, schnelle Bewegungen und Soundeffekte. Welche von beiden sich zum Erlernen besser eignet, sollten Sie vorher ausprobieren.

Type Invaders
Herausgeber: Carswell Computers
Computer: Acorn B

Dieses spielorientierte Programm ist für Anfänger gedacht. Die einfache Grafik stellt angreifende Buchstaben dar, die durch korrektes Eintippen zerstört werden sollen. Es gibt zehn verschiedene Spielstufen, die von einfachen Großbuchstaben bis zu Wörtern mit fünf Lettern reichen. Auch Wechsel zwischen Groß- und Kleinbuchstaben sowie Ziffern ist vorgesehen. Vier Geschwindigkeiten sind vorgegeben. Erfahrene Tipper werden aber problemlos auch die schnellste Spielversion bewältigen. Trotzdem ist das Programm eine gute Übungsmöglichkeit für langsame Schreiber.

Sprintyper
Herausgeber: Micro Software International
Computer: Commodore VC 20

Das textorientierte Programm soll sowohl Anfängern wie Profis zu mehr Tempo verhelfen. Es verfügt über eine Bibliothek von 356 635 Übungssätzen. Zu Anfang wird ein einfacher Satz auf dem Bildschirm gezeigt, der so schnell wie möglich eingetippt werden soll. Ein tiefer Ton zeigt dabei einen Fehler an – er hört erst dann wieder auf, wenn der Fehler berichtigt ist. Nach dem Schreiben wird die benötigte Zeit, die Fehlerzahl und eine Rekordzeit ausgegeben. Sprintyper ist eher eine Geschwindigkeitsprüfung, die dem Anfänger nur wenig konstruktive Hilfestellung für den Ausbau seiner Fähigkeiten anbietet.

Typing Tutor II
Herausgeber: Microsoft
Computer: Apple IIe und Apple IIe+

Um das Programm zu starten, muß Applesoft im ROM vorhanden sein, es braucht mindestens 48K Speicher, ein Diskettenlaufwerk und DOS 3.3. Das menügesteuerte, textorientierte Übungsprogramm bietet eine Kombination aus Übungen, Lernabsätzen und Geschwindigkeitstests. Das wichtigste Merkmal ist ein spezielles Zeitmeßsystem, das 100mal pro Sekunde die Schreibgeschwindigkeit prüft und auch kleinste Verzögerungen feststellt. Anfänger schreiben zuerst einige Briefe zur Übung. Wenn dabei eine Geschwindigkeit von 30 Wörtern pro Minute überschritten wird, geht es mit einer schnelleren Übungsversion und neuen Briefen weiter.

System-unterbrechung

In der letzten Folge über den Motorola 68000 hatten wir uns die serielle Ein- und Ausgabe angesehen. Nun wenden wir uns den parallelen Schnittstellen zu und stellen Ihnen hier die Interruptbehandlung vor.

Sehen wir uns zunächst an, wie die serielle Ausgabe in der Subroutine „Meldung" aussieht. Die Adresse des Textes wird der Subroutine mit dem Adreßregister A3 übergeben:

```
LEA TEXT,A3     Pointer auf den Text der Meldung setzen
JSR MELDUNG     Meldung ausgeben
```

Das Modul gibt einen Text aus, der als

```
TEXT:DC.B       'Computer Kurs',$00
```

gespeichert wurde. DC.B ist eine Assembleranweisung, die Platz für den Text „Computer Kurs" reserviert. Das „leere" Byte $00 ist eine Endmarkierung, die anzeigt, daß kein weiterer Text folgt.

Die Routine zur byteweisen Ausgabe des Arrays TEXT arbeitet mit der Subroutine OUTCH. Dabei zeigt das ‚Ready'-Bit an, daß ACIA sendebereit ist und ein weiteres Zeichen ausgeben kann. Die Subroutine „Meldung" könnte so aussehen:

```
MELDUNG: MOVE.B (A3)+,DO   nächstes Byte der Meldung holen
         BEQ    FERTIG     auf Ende der Meldung testen
         JSR    OUTCH      Zeichen ausgeben
         BRA    MELDUNG    Schleife bis zum Textende wiederholen
FERTIG:  RTS
```

Die Bytes der Meldung werden in DO kopiert (das Datenregister, das OUTCH als Wertregister verwendet), wobei A3 ein Pointer mit nachinkrementierter indirekter Adressierung ist. Wenn das Byte auf Null steht, wird über das Label FERTIG der Rücksprung ausgelöst, anderenfalls gibt OUTCH das Zeichen aus.

Die parallele Datenausgabe (das gleichzeitige Senden von Datenbits) geschieht mit dem PIA-Chip. Da dieser Chip auf ein breites Anwendungsspektrum ausgelegt ist, muß er erst einmal auf die individuelle Hardwarekonfiguration eingestellt werden. Dies entspricht auf dem ACIA-Chip dem Setzen von Bitraten und Byteformaten.

Beim PIA-Chip müssen die acht parallelen Leitungen beider Chiphälften (A und B) als Ein- oder Ausgabeleitungen (d. h. die Datenrichtung) konfiguriert werden. Das Schreiben auf Bit 2 des Steuerregisters (CRA oder CRB) zeigt an, daß der Inhalt eines Datenrichtungsregisters (DDRA oder DDRB) gesetzt werden soll. Hier ein Beispiel:

```
CONFIGPIA: CLR.B   PIACRB      Bit des Steuerregisters auf Null setzen
           MOVE.B  #$FF,PIADDRB Datenrichtung auf Ausgabe setzen
           BSET    #2,PIACRB  Auf normales Datenregister zurückgehen
           CLR.B   PIACRA     Datenrichtungsregister für A adressieren
           CLR.B   PIADDRA    Datenrichtung auf Eingabe setzen
           BSET    #2,PIACRA  Auf normales Datenregister zurückgehen
```

Dabei wird die gesamte Seite A auf Eingabe und Seite B auf Ausgabe gesetzt. Die eigentliche Datenübertragung zu parallel angeschlossenen Geräten geschieht dann mit:

```
MOVE.B DO,PIADRA Inhalt von DO ausgeben
```

während folgende Zeile die Eingabe erledigt:

```
MOVE.B PIADRA,D1 Eingabe in D1 einlesen
```

Alle angesprochenen PIA-Adressen müssen zuvor wie folgt initialisiert werden:

```
PIADRA EQU $30051
PIACRA EQU $30053
```

Die Interpretation der von der PIA gelesenen oder geschriebenen Daten hängt von dem Ge-

rät ab, das mit den digitalen Leitungen des Chips verbunden ist. Es könnte zum Beispiel ein siebenteiliges Anzeigeelement für Dezimalzahlen angeschlossen sein (siehe Bild). Die Anzeige der Zahl 3 beispielsweise müßte die Elemente 1, 4, 2, 5 und das Element 3 mit Strom versorgen.

Es müssen aber nicht nur Daten zum Peripheriegerät übertragen, sondern auch dessen elektrische Funktionen gesteuert werden. Beim Einspeisen der Daten in die siebenteilige Anzeige übernehmen einige der freien Bits daher Steuerfunktionen. Da die Steuersignale oft einen elektrischen Taktpuls simulieren, muß auch das Steuerbit gesetzt und wieder auf Null gestellt werden:

```
BSET  #CONBITNO,PIADRA  Steuerbit setzen
JSR   DELAY             kurze Warteroutine
BCLR  #CONBITNO,PIADRA  Bit danach wieder auf Null stellen
```

Dieses kurze Modul erzeugt das Taktsignal auf dem digitalen Kanal, der CONBITNO (bei Adresse PIADRA) zugeordnet ist.

Interrupts sollen im allgemeinen helfen, die CPU optimal auszunutzen und eine Reaktion auf externe Ereignisse zu ermöglichen. So braucht die CPU während der Zeichenausgabe (z. B. beim Drucken mit OUTCH) nicht zu pausieren. Wir müssen dabei jedoch wissen, wann der Druck beendet ist und das nächste Zeichen ausgegeben werden soll.

Beim Warten auf Eingaben ist die Situation nicht ganz so einfach. Hier hängt die Schnelligkeit des Systems zwar viel von der Eingabegeschwindigkeit ab, aber auch von anderen Aufgaben, die die CPU ausführt, z. B. die parallele Druckausgabe während des Wartens.

Für diesen parallelen Vorgang muß die logische Reihenfolge der Ereignisse so organisiert werden, daß weder die Programmsteuerung noch Daten verlorengehen. Sehen wir uns die Abläufe einmal genauer an.

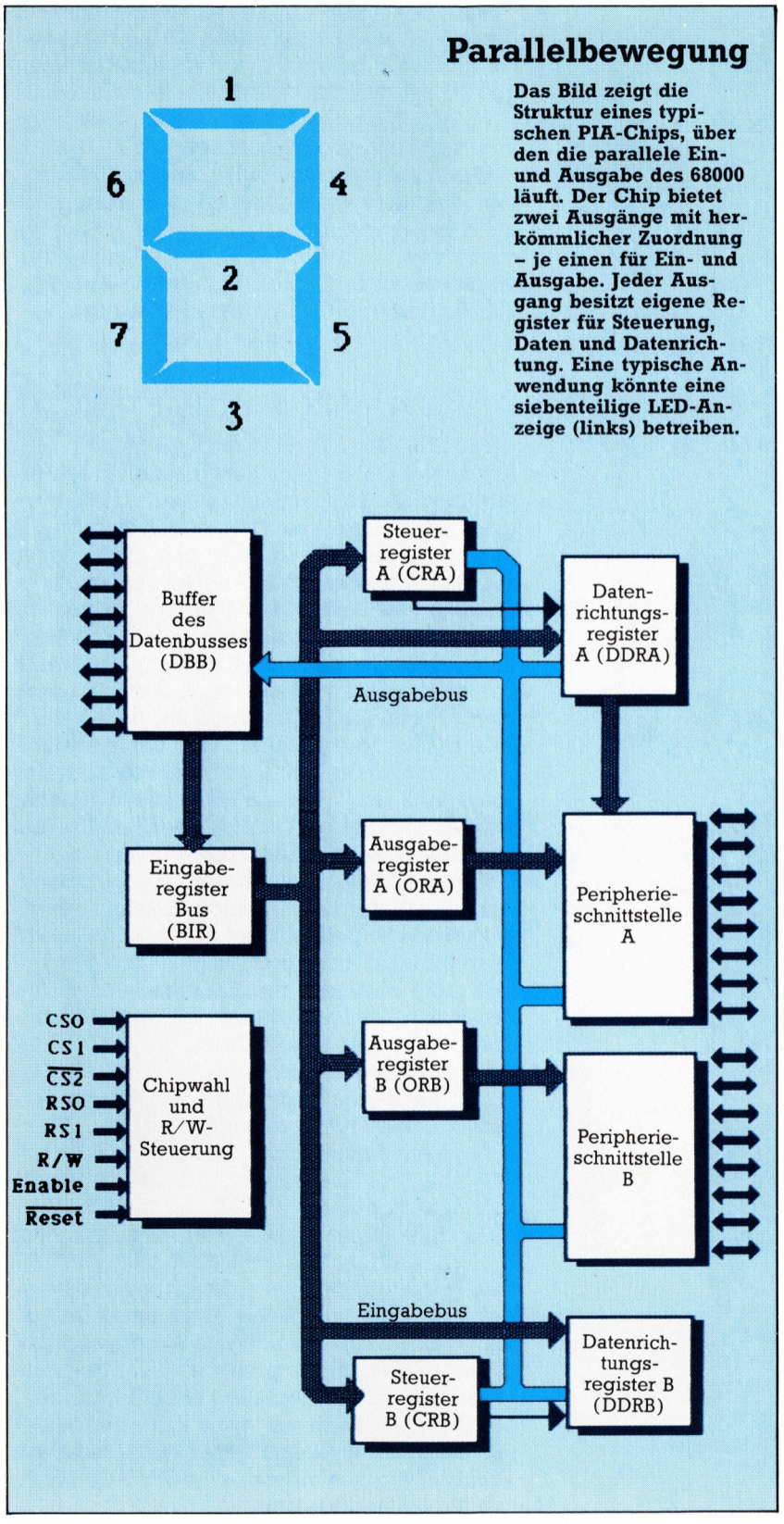
- Computerstatus sichern. Damit wird sichergestellt, daß Sie jederzeit problemlos und ohne Datenverluste zum Programm zurückkehren können. Zunächst müssen wir jedoch definieren, woraus der „Status" eines Computers überhaupt besteht.

Als Computerstatus ließe sich der gesamte Programm- und Datenbereich eines Anwenderprogramms mit Registern und Programmzähler definieren. Da Interrupts in der Praxis jedoch kaum die Anwenderprogramme ändern, genügt das Sichern des Programmzählers (PC) und des Statusregisters (SR). Diese beiden Register liefern genaue Informationen über den Status des unterbrochenen Programms.

- Interruptquelle identifizieren. Da fast alle Computer mit mehr als nur einem Peripheriegerät arbeiten, müssen wir feststellen, welche Bearbeitungsroutine gemeint ist.
- Die Interrupts anderer Quellen abschalten. Das Abschalten ist notwendig, da Interruptanforderungen, die während der Bearbeitung eines Interrupts entgegengenommen werden, zu Datenverlusten führen können. Diese Situation tritt jedoch nur ein, wenn die Bearbeitung des zweiten Interrupts länger dauert als die

Parallelbewegung

Das Bild zeigt die Struktur eines typischen PIA-Chips, über den die parallele Ein- und Ausgabe des 68000 läuft. Der Chip bietet zwei Ausgänge mit herkömmlicher Zuordnung – je einen für Ein- und Ausgabe. Jeder Ausgang besitzt eigene Register für Steuerung, Daten und Datenrichtung. Eine typische Anwendung könnte eine siebenteilige LED-Anzeige (links) betreiben.



Zeit, die zwischen zwei Datenübertragungen des ersten Interrupts liegt.

- Die Bearbeitungsroutine des Interrupts aktivieren. Diese Aufgabe kann automatisch oder über einen Spezialbefehl erfolgen.
- Rücksprung auf das unterbrochene Programm. Nach der Interruptbearbeitung wird der ursprüngliche Zustand wiederhergestellt.

Wie sieht dieser logische Ablauf nun auf dem 68000 aus? Zuerst wird der Maschinenstatus gesichert, indem PC und SR auf den Stack geschoben werden. Da dieser Mechanismus Interrupts schachteln kann, lassen sich auch Interruptroutinen unterbrechen.

Wenn die Interruptroutine Register verwendet, die für andere Zwecke reserviert sind, kann sie diese Register beim Einsprung sichern und beim Rücksprung wieder in ihren ursprünglichen Zustand versetzen. Auf dem 68000 erledigt MOVEM diesen Vorgang:

```
MOVEM D1,D3,—(SP) Datenregister auf den
                  Stack schieben
        |         Bearbeitungscodes für
                  Interrupts
MOVEM(SP)+,D1,D3 wiederherstellen
```

Auf dem 68000 ist auch das Feststellen der Interruptquelle sehr einfach, da jedem Interrupt eine eindeutige Speicherstelle ('Vektor' genannt) zugewiesen ist. Vektoren sind Pointer auf die Adressen der Bearbeitungsroutinen. Da mehrere Geräte den gleichen Vektor ansprechen können, müssen die entsprechenden Geräte abgefragt werden, um die Quelle der Interruptanforderung herauszufinden. Dies kommt auf dem 68000 jedoch nur selten vor.

Auch die Interruptprioritäten der einzelnen Geräte regelt der 68000 automatisch nach der vorgegebenen Hardwareeinstellung. Dabei dringen nur Interrupt von höherer Priorität (als der gerade bearbeitete) zur CPU vor.

Wenn ein Interrupt von der CPU bearbeitet werden soll, wird der PC mit dem Inhalt des Interruptvektors geladen und die entsprechende Bearbeitungsroutine aufgerufen. Die Routine lädt dann entweder die Eingabedaten in ein Register – und von dort in einen Zwischenspeicher – oder sendet Daten aus einem Buffer zum Ausgabegerät.

Der Rücksprung zum unterbrochenen Programm sieht zwar wie ein normaler Rücksprung aus einem Submodul aus, wird statt mit RTS aber mit RTE ausgeführt. Da RTE dabei automatisch den PC und SR aus dem Systemstack lädt, sollten alle Interruptroutinen mit RTE enden. Das bedeutet aber auch, daß A7 (der Pointer für den Systemstack) auf die Register des unterbrochenen Programms zeigen muß. Wenn Sie Daten auf dem Systemstack abgelegt haben, müssen Sie diese vor der Ausführung der RTE-Anweisung unbedingt wieder herunterziehen. Besser wäre es, einen anderen Stack zu verwenden. Der 68000 kann zur Veränderung von Adressen jedes Adreßregister als Stack einsetzen.

Sehen wir uns nun eine typische Interruptroutine an, die die Geräte einzeln abfragt („Polling"), um die Interruptquelle festzustellen. Unser Beispielsystem enthält auf „Ebene 4" (eine Hardwareleitung mit Priorität 4) zwei mögliche Interruptquellen. Die Geräte sind 1) eine externe Tastatur und 2) eine Echtzeituhr. Die Uhr zählt Sekunden und wird von anderen Teilen des (Software-)Systems als Taktgeber eingesetzt.

Um den Interruptvektor für Ebene 4 (bei $70) richtig setzen zu können, müssen wir ihn zuvor initialisieren:

```
MOVE.L  #EXTDEVS,$70   Vektor der Ebene
                       4 setzen
```

Natürlich werden auch andere Register gesetzt, beispielsweise die Stacks für Anwender und System:

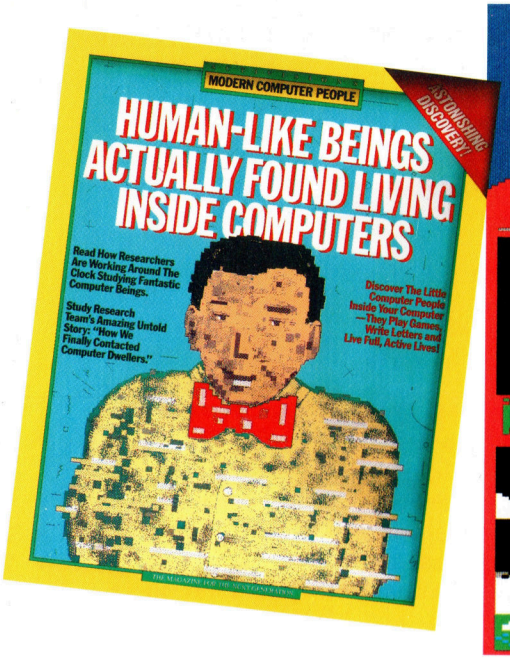
```
MOVE.L #STACK,SP       Systemstack set-
                       zen
MOVE.L #USERSTACK,A0   Anwenderstack
                       setzen
```

A7 steht nun auf der Adresse STACK und A0 auf USERSTACK.

Hier die Interruptbearbeitungsroutine:

```
EXTDEVS MOVEM.L  D0—D7,(A0)   Datenre-
                              gister si-
                              chern
        BTST     #7,PIACRA    feststel-
                              len, ob
                              Tastatur-
                              interrupt
                              vorliegt
        BNE      CHAR
CKCLK   BTST     #6,PIACRB    falls
                              nicht,
                              muß es
                              die Uhr
                              sein
        BNE      CLOCK
        BRA      WILD         anson-
                              sten liegt
                              ein Feh-
                              ler vor
CHAR    JSR      KEYBOARD     Tastatur-
                              routine
                              aufrufen
        BRA      CKCLK        die Uhr
                              könnte
                              ebenfalls
                              unterbro-
                              chen ha-
                              ben
CLOCK   JSR      SECS         Uhrrou-
                              tine auf-
                              rufen
WILD
        MOVEM.L  (SP)+,D0—D7  Datenre-
                              gister
                              wieder-
                              herstel-
                              len
        RTE
```

Diese Routine sichert alle Datenregister.



Software-Zwerge

Das Spiel ‚Little Computer People' der Firma Activision für den Commodore 64 paßt zwar in keine der üblichen Spiele-Kategorien, hat aber durchaus Chancen, ein großer Erfolg zu werden.

Wenn die Spiele-Software seit der Frühzeit der Heimcomputer auch beträchtliche Fortschritte gemacht hat, läuft meist doch alles nach dem gleichen bewährten Schema ab: Ob Arcade-, Abenteuer- oder Strategiespiel – stets muß man mit irgendwelchen bedrohlichen Herausforderungen fertigwerden, die sich der Programmierer ausgedacht hat. Bei „Little Computer People" ist die Zielsetzung aber ganz anders: Ihre Aufgabe besteht nur darin, für das Wohlbefinden einer kleinen Person zu sorgen, die sich im Rechner eingenistet hat.

Das Programm ist so geschrieben, daß die Spielfigur in jeder Kopie gewisse individuelle Züge aufweist – jedesmal sind Name, Kleidung und auch die Lebensgewohnheiten verschieden. Viel Mühe ist offensichtlich darauf verwendet worden, die kleinen Männchen mit einer „glaubhaften" Geschichte auszustatten: Der Software ist eine ausführliche schriftliche Information beigefügt, in der die ‚Entdeckung' dieser Wesen, ihre Verhaltensweisen, besonderen Vorlieben und Wohnumstände erläutert sind.

Nach dem Laden des Programms erscheint auf dem Bildschirm das Innere eines dreistökkigen Hauses mit allen üblichen Räumen. Hinter dem Schlafzimmer gibt es sogar ein verschwiegene Örtchen, in dem die Figur dann später ab und zu für ein paar Minuten verschwindet, wobei dezenterweise im unklaren bleibt, was darin vorgeht. Kurz nach dem Start des Programms betritt Ihr Rechnerbewohner das Haus und inspiziert die Räumlichkeiten. Wenn ihm diese zusagen, geht er kurz wieder hinaus und kehrt dann mit seinen Habseligkeiten einschließlich Hund zurück.

Das kleine Männchen muß nun stets mit Nahrung, Getränken und vor allem reichlich Unterhaltung versorgt werden. Ob Ihnen das gelingt, können Sie an seinem Gesichtsausdruck ablesen, der von hocherfreut bis jämmerlich reichen kann. Schaut der Arme verdrossen drein und ist grün im Gesicht, hat er nicht genug zu essen bekommen. Sie dürfen natürlich auch nicht vergessen, den Hund zu füttern.

Activision beteuert, die Sprache der kleinen Gäste sei immer noch nicht enträtselt und für die Kommunikation komme nur die Tastatur in Frage. Darüber müssen Sie also versuchen, das Kerlchen bei Stimmung zu halten, indem Sie ihm beispielsweise empfehlen, etwas zu essen, Feuer zu machen oder Klavier zu spielen. Wenn Sie ihn bitten, schreibt er Ihnen auch einen schönen Brief.

Little Computer People: für den Commodore 64
Produzent: Activision Inc., Box 7287 Mountain View, California/USA
Autoren: Richard Gold, David Crane und Sam Nelson
Datenträger: Cassette oder Diskette
Joystick: Nicht erforderlich

Die geistigen Väter der kleinen Leute im Computer haben keine Mühe gescheut, die Figuren so lebensecht wie möglich erscheinen zu lassen. Die mitgelieferte Anleitung im Stil eines populärwissenschaftlichen Magazins (Titel: ‚Menschliche Lebewesen im Computer entdeckt') ist hervorragend, und das Haus als Schauplatz der Handlung ist mit allem Komfort für das tägliche Leben ausgestattet. Es bietet mehr als andere Spiele.

Computerprodukte

In der Produktion spielen Computer heute eine sehr wichtige Rolle – nicht nur bei Planung und Entwurf: Rechner steuern heute den gesamten Ablauf bis zur Lieferung an den Endverbraucher.

Um effektiv produzieren zu können – von der Entwicklung eines neuen Produktes bis zu seiner Lieferung –, muß eine Vielzahl von Faktoren abgeglichen werden: Termine, Personal, Maschinen und Rohmaterial, Design und Lieferfristen sowie Dutzende anderer Bedingungen. In den frühen 80er Jahren waren nur die größten und teuersten Computer leistungsfähig genug, um erfolgreich in der Wirtschaft eingesetzt zu werden. Hier hat sich inzwischen ein Wandel vollzogen: Zum einen wurden die Computer leistungsfähiger, zum anderen sind viele Unternehmen erst heute auf die Möglichkeiten der Rechner aufmerksam geworden.

In neuerer Zeit trifft man gelegentlich auf Produkte, die ausschließlich mit Hilfe von Computern gefertigt wurden – vom Entwurf bis zur Lieferung an den Endverbraucher. Am weitesten ist die Entwicklung naturgemäß in der Computerbranche selbst fortgeschritten – die Hersteller von Rechnern und elektronischen Bauteilen waren die ersten High-Tech-Unternehmen.

Wenn Computer den gesamten Fertigungsablauf steuern, spricht man von „CIM" (Computer Integrated Manufacturing). Die vielfältigen Abkürzungen können in diesem Bereich allerdings sehr verwirren.

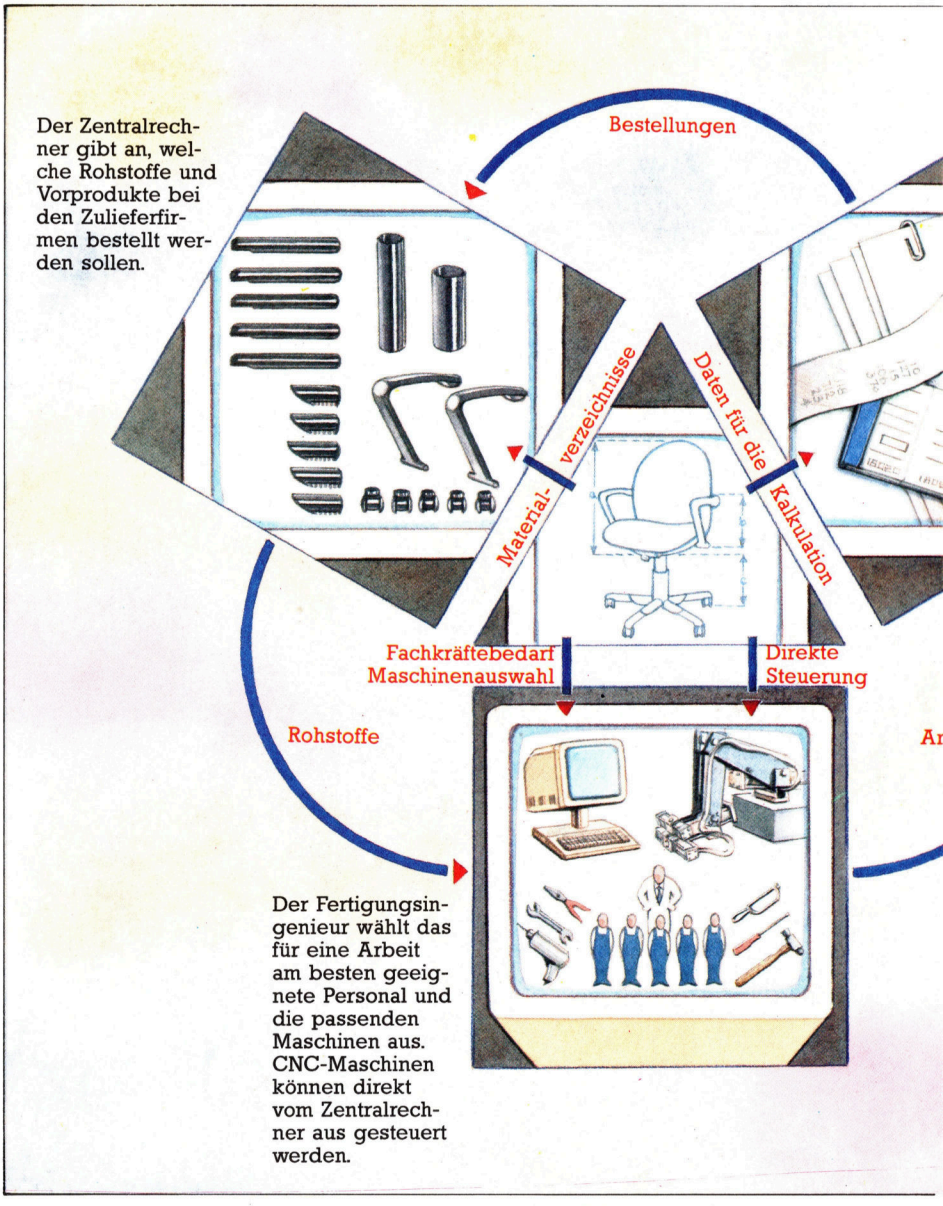
Termine

Große Industriefirmen sind auf Zusammenarbeit angewiesen – jede Abteilung ist für einen anderen Teil der Produktion zuständig. Die Planungsabteilung entwickelt eine Idee und stellt die technischen Spezifikationen zusammen, die zur Basis des gesamten Produktionsprozesses werden. Zur Vorbereitung müssen Bearbeitungswerkzeuge bestellt oder angefertigt werden, Werkzeugmaschinen sind neu zu programmieren. Die Lagerverwaltung muß sicherstellen, daß die nötigen Rohmaterialien oder Vorprodukte jederzeit in ausreichender Menge zur Verfügung stehen.

Dazu kommt die finanzielle Seite des Unternehmens. Bevor ein Produkt bestellt wird, wollen die Kunden natürlich den Preis kennen – der Hersteller muß exakt kalkulieren und genaue Zahlen nennen. Schließlich ist auch die Liefer- und Zahlungsfrist abzustimmen – Waren sollen termingerecht bereitstehen und auch pünktlich bezahlt werden. Die Personalsituation ist ebenfalls zu berücksichtigen: Fachleute mit besonderen Fähigkeiten und Kenntnissen sollen zum richtigen Zeitpunkt für die Arbeit frei sein.

Wie unterschiedlich die vorerwähnten Aufgaben auch immer scheinen mögen, sie basieren alle auf den gleichen Daten. Und das ist der Schlüssel zu CIM, bei dem ein zentraler Computer sämtliche wichtigen Informationen enthält. Alle Abteilungen können darauf zugreifen: Die Lager- und Einkaufsabteilung kann die Produktbeschreibung abrufen und prüfen, welche Materialien gebraucht werden, der Werksleiter sieht, welche Maschinen ein-

Computer können den Fertigungsprozeß viel stärker vernetzen als es früher möglich war. Das Konzept des computergestützten Entwurfs bildet die Basis jeder rechnerorientierten Fabrik, bei der die Elektronik die Steuerung vom Entwurf bis zur Fertigung verwaltet bzw. unterstützt.



Mit der CIM-Datenbank läßt sich ein Zeitplan für die gleichzeitige Fertigung verschiedener Produkte erstellen – die Rohstoffverwaltung, der Maschinen- und Personaleinsatz werden automatisch optimiert. Gleichzeitig informiert das System auch die Buchhaltung über die anfallenden Kosten. Die Menge der notwendigen Rohstoffe kann ebenso genau kalkuliert werden wie die Kosten von Arbeitseinsatz und Maschinenzeit. Aus diesen Informa-

gesetzt werden sollen, und die Rechnungsabteilung kann die Kosten kalkulieren – alle Daten sind in einer Datenbank gespeichert.

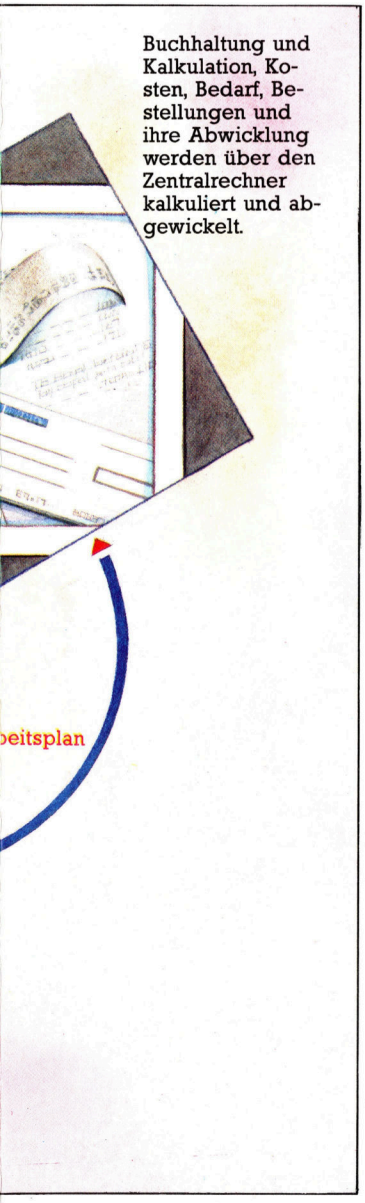
Simulation

Lassen Sie uns die nötigen Informationen genauer betrachten: Am wichtigsten ist natürlich die Beschreibung des Produktes selbst, die von der Planungsabteilung erstellt wird. Dazu werden seit Jahren mit großem Erfolg CAD-Systeme (CAD = Computer Aided Design) verwendet. Nur wenige Erzeugnisse bestehen aus einem einzigen Bauteil, im allgemeinen hat man viele zusammengesetzte Elemente.

Ausgereifte CAD-Systeme können Teil für Teil ein Gesamtbild des Produktes zusammensetzen, wobei Schicht auf Schicht gelegt wird, bis die Gesamtansicht vollständig ist – und das Ganze auch in drei Dimensionen. Aus diesem Bild kann ein geeignetes Programm für Simulation und Analyse bereits die späteren Eigenschaften des Produktes ermitteln.

Das BOM (Bill Of Materials = Materialverzeichnis) ist eine Hilfe für die Lagerverwaltung. Das BOM ist ein Verzeichnis aller Teile eines Produktes und kann wiederum zu einer Liste von Rohmaterialien aufbereitet werden, die für die Herstellung nötig sind. Die Lagerverwaltung muß die vorhandenen Materialien mit dem Bedarf abgleichen. Sie muß auch wissen, was während der Produktion nachzubestellen ist. Die Einkaufsabteilung ist darüber zu informieren, welche Waren von wem zu beziehen sind.

Der CAD-Entwurf dient auch den Fertigungstechnikern – sie erfahren, welche Anforderungen an Maschinen und Personal gestellt werden. Mit diesen Informationen ist es dann ein leichtes, die Anforderungen an Maschinenausrüstung und den Arbeitskräften zu ermitteln und einen Ablaufplan für die Fertigung zu erstellen.



Buchhaltung und Kalkulation, Kosten, Bedarf, Bestellungen und ihre Abwicklung werden über den Zentralrechner kalkuliert und abgewickelt.

Kalkulation

CIM soll aber nicht nur zur Herstellung einzelner Produkte beitragen, sondern die gleichzeitige Fertigung vieler Waren in den unterschiedlichsten Entwicklungsstadien synchronisieren. Situationen, in denen zwar Schrauben fertig sind, die entsprechenden Muttern jedoch nicht, soll es mit CIM nicht mehr geben.

tionen kann die für den Kunden wichtige Preisfrage bereits vor Anlaufen der Herstellung genau beantwortet werden.

Später gelangt der im Konstruktionsbüro erstellte Entwurf direkt zu einer rechnergesteuerten Werkzeugmaschine, welche die notwendigen Bearbeitungsschritte einleitet. Auf der Grundlage des Entwurfes hat sich inzwischen ein Netz von Informationen gebildet: Sowohl die Lagerverwaltung, die Fertigung, das Vertriebsbüro und sogar der Versand erhalten ihre Anweisungen und Informationen aus der gleichen Datenbank.

Aus manchen Industriezweigen ist der Computer-Einsatz heute nicht mehr wegzudenken. So ist etwa der Entwurf eines modernen Microprozessors so kompliziert, daß er ohne Rechner nicht mehr zu überblicken ist. Schaltungsentwicklung „per Hand" würde viel zu lange dauern, zudem könnte das von einem Zeichner angefertigte Schaltungslayout nicht einmal getestet werden, ohne den Chip tatsächlich herzustellen. Und das wäre so teuer, daß sich die Produktion nicht lohnen könnte. Um aber effektiv arbeiten zu können, muß die Datenbank erst an die nötigen Informationen kommen.

Industriesprache

Industrieller Computereinsatz wird mit einem Wust von Abkürzungen beschrieben. Einige davon haben wir hier zusammengestellt:

AMT Advanced Manufacturing Techniques = Neue Produktionstechniken: Allgemeiner Ausdruck für CAD, CAM und CAE.

CAD Computer Aided Design = Computergestützter Entwurf: Bezieht sich meist auf das reine Zeichnen mit Computerhilfe. Dabei können zwar Bilder entstehen, analysiert werden sie durch CAD aber meist noch nicht.

CAM Computer Aided Manufacturing = Computergestützte Produktion: Schließt alle Techniken ein, bei denen Rechner in den Produktionsprozeß integriert sind.

CAE Computer Aided Engineering = Computergestützte Analyse: Einsatz der Datenverarbeitung (meist CAD-Datenbanken) für die Analyse eines Entwurfs.

NC Numerical Control = Numerische Steuerung: Datengesteuerte Maschine zur Bearbeitung von Werkstücken.

CNC Computer Numerical Control = Computersteuerung: Weiterentwicklung der NC-Maschinen, die früher noch per Lochstreifen gesteuert wurden. Heute erfolgt die Steuerung direkt über angeschlossene oder integrierte Computer.

CIM Computer Integrated Manufacturing = Computergesteuerte Fertigung: Beschreibt den gesamten Prozeß der automatischen Steuerung von Fertigungsabläufen.

BOM Bill Of Materials = Materialverzeichnis: Liste aller Komponenten, die zu einem Produkt gehören.

Das fertige Blatt

Wir beenden unser Tabellenkalkulationsprogramm mit den Routinen zum Laden und Sichern der Daten und Formeln. Außerdem finden Sie eine Hilfstafel in dieser Folge.

Den letzten Teil unseres Kalkulationsprogramms bildet die Implementation der Routinen zum Laden und Sichern, mit denen wir Daten und Formeln auf Diskette und Cassette speichern können. Weil die vier Rechner, für die wir das Spreadsheet-Programm entwikkelten, Dateien unterschiedlich ablegen, ist für Acorn B, Commodore 64, Schneider CPC und Sinclair Spectrum jeweils eine eigene Version dieses Programmteils erforderlich.

In allen vier Versionen beginnen die Routinen zum Sichern und Laden von Daten bei der Zeile 7000 mit der Auflistung der verfügbaren Optionen in einem kleinen Menü:

1 — Laden eines Datensatzes
2 — Laden eines Formelsatzes
3 — Sichern eines Datensatzes
4 — Sichern eines Formelsatzes
'X' — Abbrechen

Aus diesem Menü ist ersichtlich, daß Sie Daten und Formeln voneinander unabhängig sichern und laden können. Diese Eigenschaft ermöglicht uns, Formelsätze für bestimmte Abläufe (z. B. Berechnung der Mehrwertsteuer) auf Datenträger zu sichern und von dort immer wieder abzurufen, um neue Daten zu bearbeiten. Andererseits möchten Sie vielleicht auf einen Datensatz verschiedene Formeln anwenden. Diese Daten schreiben Sie einfach auf Diskette oder Kassette und rufen sie von dort je nach Bedarf wieder ab.

Zeichenfeld

Die Implementation dieser Funktion ist relativ einfach. Alle Daten des Arbeitsblattes liegen in der zweidimensionalen Feldvariablen M(,) und die entsprechenden Formeln im eindimensionalen Zeichenfeld F$(). Die Inhalte dieser Felder schreiben wir in eine sequenzielle Datei. Beim Laden lesen wir einfach diese Datei wieder zurück in die Felder, wobei deren vorheriger Inhalt gelöscht wird.

An den Programmversionen für Acorn B- und Schneider CPC-Computer wollen wir beschreiben, wie Formeln und Daten gesichert werden. Die Befehle zum Eröffnen und Schließen einer Datei sind bei diesen Rechnern fast gleich. OPENIN öffnet eine Datei zum Lesen, OPENOUT zum Schreiben. Während der Acorn B eine Kanalnummer im OPEN-Befehl erwartet, benutzt der Schneider grundsätzlich den Kanal 9 für PRINT- und INPUT-Operationen mit Dateien.

Zum Sichern wird lediglich jedes Element des Feldes in einer FOR. . .NEXT-Schleife (oder beim zweidimensionalen Feld M(), in zwei verschachtelten Schleifen) mittels des PRINT-Befehls in die sequenzielle Datei übertragen. Die Laderoutine ist nur eine Umkehrung dieses Vorgangs.

Sequenzielle Dateien gibt es beim Spectrum nicht, doch erreichen wir das gleiche Ziel mit

```
SAVE I$ DATA F$ ()
```

wobei I$ der Dateiname und F$() das zu übertragende Feld ist. Dieser BASIC-Befehl erfüllt bereits unsere Anforderungen.

Bei der Commodore-Version sind einige Schwierigkeiten zu bewältigen: Das Problem tritt beim Schreiben des Inhaltes eines Zeichenfeldes auf.

Beim DIMensionieren eines Zeichenfeldes setzt das System alle Elemente auf „Null-Strings", repräsentiert durch CHR$(0). Wenn wir die Formeln unseres Kalkulationsblattes sichern und nicht alle Zellen eine Formel enthalten, werden die entsprechenden Elemente von F$() als CHR$(0) in die Datei geschrieben. Damit funktioniert das Auslesen der Datei mittels INPUT# nicht mehr!

Während zuvor eingegebene Formeln einwandfrei gelesen werden, hängt sich das System bei Erreichen eines „Null-Strings" auf. Diese recht unerfreuliche Eigenschaft umgehen wir, indem wir jedes Element des Zeichenfeldes prüfen und, wenn es leer ist, ein CHR$(1) anstelle des CHR$(0) schreiben. Die Eins stellt, ebenso wie die Null, keinen Buchstaben dar, doch der INPUT#-Befehl funktioniert wieder. Allerdings müssen wir auch die Laderoutine ändern und das Zeichen Eins wieder in den Null-String umwandeln (siehe Zeilen 7425 und 7230 des Commodore Listings).

Das Commodore Programm ist für den Betrieb mit einer Diskettenstation ausgelegt. Falls Sie eine Datasette benutzen, ändern Sie bitte die folgenden Zeilen:

```
7110 GOSUB 7500:OPEN1,1,0,I$
7219 GOSUB 7500:OPEN1,1,0,I$
7310 GOSUB 7500:OPEN1,1,1,I$
7410 GOSUB 7500:OPEN1,1,1,I$
```

Damit ist unser Tabellenkalkulationsprogramm fertig. Zu Anfang besprachen wir einige Unzulänglichkeiten des Programms, wie die auf fünf

Zeichen begrenzte Zellenbreite, daß nur numerische Daten zulässig sind und die Fehlererkennung zugunsten der Programmlänge recht simpel ist. Wie schon bei früheren Projekten ist nun Ihre Fertigkeit gefragt, um die vorhandenen Funktionen auszubauen oder neue hinzuzufügen.

Ende gut, alles gut

Wenn Sie nun Daten und Formeln laden und sichern können, haben sie die Möglichkeit, Ihren Rechner als Universalgenie für profihafte Tabellenkalkulationsprogramme einzusetzen. Wie schon bei früheren Projekten ist nun Ihre Fertigkeit gefragt.

Hilfe!

Die Subroutine ab Zeile 6000 ist für die Ausgabe einer Hilfstafel zuständig, die Ihnen in Kurzform anzeigt, mit welchen Tasten Sie die verschiedenen Funktionen aufrufen. Was die Funktionen bewirken, erklärt die Tafel nicht, daher beschließen wir diese Serie mit einer kleinen Übersicht der Funktionen und ihrer Arbeitsweise.

ENTER NEW FORMULA IN CELL: Sie können in die Zelle, in welcher der Cursor steht, eine Berechnungsformel eingeben. Formeln enthalten normalerweise Konstante und Zellvariable (vertreten durch Zellnamen, wie A1, B12) und beeinflussen die CALCULATE Funktion.

STORE CURRENT SHEET: Die Daten des Arbeitsblattes können Sie im Programm zwischenspeichern. Dies ermöglicht Ihnen, die Daten mit unterschiedlichen Formelsätzen zu berechnen, und ist gleichzeitig ein Schutz der Daten bei fehlerhaften Berechnungsformeln.

GET STORED SHEET: Setzt einen zwischengespeicherten Datensatz wieder in das Arbeitsblatt ein.

CALCULATE: Diese Funktion berechnet das Arbeitsblatt und setzt die Resultate in die vorgesehenen Zellen ein.

REPLICATE: Oft unterscheiden sich Formeln in einer Reihe oder Spalte des Arbeitsblatts nur durch die Zellnamen. Die **REPLICATE** Funktion setzt, beispielsweise nach dem Befehl ‚N1 (N2-N5)', die Formel aus der Zelle N1 in die Zellen N2 bis N5 und berechnet die in der Formel enthaltenen Zellnamen neu.

CLEAR SHEET: Löscht alle Daten aus dem Kalkulationsblatt.

TAB CURSOR: Neben der Steuerung des Cursors durch die Cursortasten kann jede Zelle auch direkt angewählt werden. Geben Sie nach dem Aufruf der Option einfach den Namen der gewünschten Zelle ein.

LOAD und **SAVE** Routinen: Hiermit rufen Sie das Laden/Sichern-Untermenü auf, von wo die Daten oder Formeln auf Diskette oder Cassette gesichert und zuvor gesicherte Informationen wieder in das Arbeitsblatt geladen werden.

Dateibehandlung und Hilfsfunktionen

Commodore 64

```
6000 REM ***** HELP SCREEN *****
6010 PRINT CHR$(147):PRINT
6020 PRINT:PRINT "        F1 - PRINTS THI
S HELP SCREEN "
6030 PRINT:PRINT "        F2 - ENTER NEW
FORMULA IN CELL   "
6040 PRINT:PRINT "        F3 - STORE CURR
ENT SHEET         "
6050 PRINT:PRINT "        F4 - GET STORED
 SHEET            "
6060 PRINT:PRINT "        F5 - CALCULATE
SHEET             "
6070 PRINT:PRINT "        F6 - CLEAR SHEE
T                 "
6080 PRINT:PRINT "        F7 - REPLICATE
FORMULA           "
6090 PRINT:PRINT "        F8 - LOADING/SA
VING SHEETS       "
6100 PRINT:PRINT "        'G' - TO TAB CU
RSOR            "
6110 GET A$:IF A$="" THEN 6110
6120 GOSUB 1000:GOSUB 1700:RETURN
7000 REM ***** LOAD/SAVE ROUTINES *****
7010 PRINT CHR$(147):PRINT:PRINT
7020 PRINT:PRINT "          F1 - LOAD DAT
A SHEET"
7030 PRINT:PRINT "          F3 - LOAD FOR
MULA ARRAY "
7040 PRINT:PRINT "          F5 - SAVE DAT
A SHEET "
7050 PRINT:PRINT "          F7 - SAVE FOR
MULA ARRAY"
7055 PRINT:PRINT "          'X' - TO EXIT
"
7056 PRINT:PRINT:PRINT
7060 GET A$:IF A$="" THEN 7060
7070 IF A$=CHR$(133) THEN 7100
7080 IF A$=CHR$(134) THEN 7200
7090 IF A$=CHR$(135) THEN 7300
7095 IF A$=CHR$(136) THEN 7400
7096 IF A$="X" THEN GOSUB 1000:GOSUB 170
0:RETURN
7100 REM **** LOAD DATA ****
7110 GOSUB 7500:I$="0:"+I$+",S,R":OPEN 2
,8,2,I$
7120 FOR I=1 TO 15:FOR J=1 TO 15
7130 INPUT#2,M(I,J)
7140 NEXT J,I
7150 CLOSE2
7160 GOTO 7010
7200 REM **** LOAD FORMULA ARRAY ****
7210 GOSUB 7500:I$="0:"+I$+",S,R":OPEN 2
,8,2,I$
7220 FOR I=1 TO 225
7230 INPUT#2,F$(I):IF F$(I)=CHR$(1)THEN
F$(I)=""
7240 NEXT I
7250 CLOSE2
7260 GOTO 7010
7300 REM **** SAVE DATA ****
7310 GOSUB 7500:I$="0:"+I$+",S,W":OPEN 2
,8,2,I$
7320 FOR I=1 TO 15:FOR J=1 TO 15
7330 PRINT#2,M(I,J)
7340 NEXT J,I
7350 CLOSE2
7360 GOTO 7010
7400 REM **** SAVE FORMULA ARRAY ****
7410 GOSUB 7500:I$="0:"+I$+",S,W":OPEN 2
,8,2,I$
7420 FOR I=1 TO 225
7425 IF F$(I)=""THEN PRINT#2,CHR$(1):GOT
07440
7430 PRINT#2,F$(I)
7440 NEXT I
7450 CLOSE2
7460 GOTO 7010
7500 INPUT "ENTER FILENAME:";I$
7510 RETURN
```

F1 — Prints HELP screen
F2 — Enter new formula in a cell
F3 — Store current sheet
F4 — Get stored sheet
F5 — Calculate sheet
F6 — Clear sheet
F7 — Replicate formula
F8 — Load/save routines
'G' — Tab cursor

Acorn B:

'H' — HELP screen
'F' — Enter new formula in cell
'S' — Store current sheet
'G' — Get stored sheet
'C' — Calculate sheet
'Z' — Clear sheet
'R' — Replicate formula
'T' — Tab cursor
'D' — Load/save routines

```
 1189 IF A$="D" THEN GOSUB 7000:REM LOAD
 SAVE ROUTINES
 6000 REM **** HELP SCREEN ******
 6020 CLS:PRINT TAB(6,2)"H - PRINT THIS
HELP SCREEN"
 6030 PRINT TAB(6,4)"F - ENTER NEW FORMU
LA IN CELL"
 6040 PRINT TAB(6,6)"S - STORE CURRENT S
HEET"
 6050 PRINT TAB(6,8)"G - GET STORED SHEE
T"
 6060 PRINT TAB(6,10)"C - CALCULATE SHEE
T"
 6070 PRINT TAB(6,12)"Z - CLEAR SHEET"
 6080 PRINT TAB(6,14)"R - REPLICATE FORM
ULA"
 6090 PRINT TAB(6,16)"T - TAB TO NEW CUR
SOR POS."
 6100 PRINT TAB(6,18)"D - LOADING/SAVING
 ROUTINES"
 6110 A$=GET$:GOSUB 1000:GOSUB 1700:RETU
RN
 7000 REM ***** LOAD / SAVE ROUTINES ***
**
 7010 CLS:PRINT TAB(6,4)"1 - LOAD DATA S
HEET"
 7030 PRINT TAB(6,6)"2 - LOAD FORMULA AR
RAY"
 7040 PRINT TAB(6,8)"3 - SAVE DATA SHEET
"
 7050 PRINT TAB(6,10)"4 - SAVE FORMULA A
RRAY"
 7055 PRINT TAB(6,12)"'X' - TO EXIT"
 7056 A$=GET$:A=VAL(A$)
 7060IF A>0 AND A<5 THEN ON A GOTO 7100,
7200,7300,7400
 7096 IF A$="X" THEN GOSUB 1000:GOSUB 17
00:RETURN
 7100 REM ***** LOAD DATA ******
 7110 GOSUB 7500:F=OPENIN I$
 7120 FOR I=1 TO 15:FOR J=1 TO 15:INPUT#
F,M(I,J):NEXT J,I
 7130 CLOSE#F:GOTO 7010
 7200 REM **** LOAD FORMULAE ****
 7210 GOSUB 7500:F=OPENIN I$
 7220 FOR I=1 TO 225:INPUT#F,F$(I):NEXT
 I
 7230 CLOSE#F:GOTO 7010
 7300 REM **** SAVE DATA ****
 7310 GOSUB 7500:F=OPENOUT I$
 7320 FOR I=1 TO 15:FOR J=1 TO 15:PRINT#
F,M(I,J):NEXT J,I
 7330 CLOSE#F:GOTO 7010
 7400 REM **** SAVE FORMULAE ****
 7410 GOSUB 7500:F=OPENOUT I$
 7420 FOR I=1 TO 225:INPUT#F,F$(I):NEXT
 I
 7430 CLOSE#F:GOTO 7010
 7500 INPUT TAB(0,22)"FILENAME"I$
 7510 RETURN
```

Schneider CPC

'H' — HELP screen
'F' — Enter new formula in cell
'S' — Store current sheet
'G' — Get stored sheet
'C' — Calculate sheet
'Z' — Clear sheet
'R' — Replicate formula
'T' — Tab cursor
'D' — Load/save routines

Vor dem Start CAPS LOCK drücken

```
1175 IF A$="Z" THEN GOSUB 5000:REM CLEAR
 SHEET
1177 IF A$="S" THEN GOSUB 5150:REM STORE
 SHEET
1178 IF A$="H" THEN GOSUB 6000:REM HELP
SCREEN
1189 IF A$="D" THEN GOSUB 7000:REM LOAD/
SAVE ROUTINES
6000 REM **** help screen ****
6020 CLS:LOCATE 6,3:PRINT"H - Print this
 HELP screen"
6030 PRINT:PRINT TAB(6)"F - Enter new FO
RMULA"
6040 PRINT:PRINT TAB(6)"S - STORE curren
t sheet"
6050 PRINT:PRINT TAB(6)"V - Get PREVIOUS
 sheet"
6060 PRINT:PRINT TAB(6)"C - CALCULATE sh
eet"
6070 PRINT:PRINT TAB(6)"Z - CLEAR sheet"
6080 PRINT:PRINT TAB(6)"R - REPLICATE fo
rmula"
6090 PRINT:PRINT TAB(6)"G - GO to new ce
ll"
6100 PRINT:PRINT TAB(6)"D - SAVE/LOAD ro
utines"
6110 WHILE INKEY$="": WEND
6120 GOSUB 1000:GOSUB 1700:RETURN
7000 REM **** LOAD/SAVE ROUTINES ****
7010 CLS:LOCATE 6,4:PRINT"1 - LOAD DATA
SHEET"
7020 PRINT:PRINT TAB(6)"2 - LOAD FORMULA
 ARRAY"
7030 PRINT:PRINT TAB(6)"3 - SAVE DATA AR
RAY"
7040 PRINT:PRINT TAB(6)"4 - SAVE FORMULA
 ARRAY"
7050 PRINT:PRINT TAB(6)"'X' - TO EXIT"
7055 A$="":WHILE A$="":A$=INKEY$:WEND
7056 A=VAL(A$)
7060 ON A GOTO 7100,7200,7300,7400
7096 IF A$="X" THEN GOSUB 1000:GOSUB 170
0:RETURN
7100 REM **** LOAD DATA ****
7110 GOSUB 7500:OPENIN I$
7120 FOR I=1 TO 15:FOR J=1 TO 15
7130 INPUT#9,M(I,J)
7140 NEXT J,I
7150 CLOSEIN:GOTO 7010
7200 REM **** LOAD FORMULAE ****
7210 GOSUB 7500:OPENIN I$
7220 FOR I=1 TO 225
7230 INPUT#9,F$(I)
7240 NEXT I
7250 CLOSEIN:GOTO 7010
7300 REM **** SAVE DATA ****
7310 GOSUB 7500:OPENOUT I$
7320 FOR I=1 TO 15:FOR J=1 TO 15
7330 PRINT#9,M(I,J)
7340 NEXT J,I
7350 CLOSEOUT:GOTO 7010
7400 REM **** SAVE FORMULAE ****
7410 GOSUB 7500:OPENOUT I$
7420 FOR I=1 TO 225
7430 PRINT#9,F$(I)
7440 NEXT I
7450 CLOSEOUT:GOTO 7010
7500 LOCATE 1,22:INPUT"FILENAME";I$
7510 RETURN
```

Sinclair Spectrum:

'H' — Prints HELP screen
'E' — Enter numeric data
'F' — Enter new formula
'C' — Calculate sheet
'S' — Store current sheet
'G' — Get stored sheet
'Z' — Clear sheet
'R' — Replicate formula
'T' — Tab cursor
'D' — Load/save routines

Vor dem Start CAPS LOCK drücken

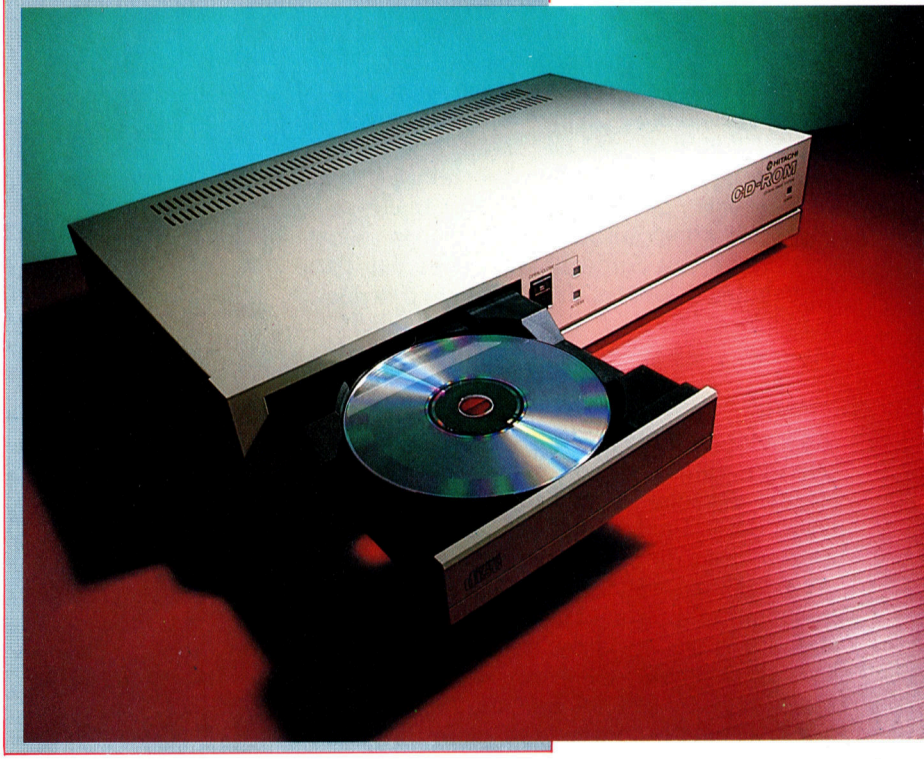
```
6000>REM PRINT HELP SCREEN
6010 CLS
6020 PRINT : PRINT "  H - PRINTS
 THIS HELP SCREEN"
6030 PRINT : PRINT "  E - ENTER
NUMERIC DATA"
6040 PRINT : PRINT "  F - ENTER
NEW FORMULA"
6050 PRINT : PRINT "  C - CALCUL
ATE SHEET"
6060 PRINT : PRINT "  S - STORE
CURRENT SHEET"
6070 PRINT : PRINT "  G - GET CU
RRENT SHEET"
6080 PRINT : PRINT "  Z - CLEAR
SHEET"
6090 PRINT : PRINT "  R - REPLIC
ATE FORMULAE"
6100 PRINT : PRINT "  T - TAB TO
 NEW CELL"
6110 PRINT : PRINT "  D - LOAD/S
AVE OPTIONS"
6120 LET A$=INKEY$: IF A$="" THE
N  GO TO 6120
6130 GO SUB 1000: GO SUB 1700: G
O TO 1100
7000 REM LOAD/SAVE OPTIONS
7010 CLS : PRINT
7020 PRINT "    LOADING AND SAVIN
G OPTIONS"
7030 PRINT
7040 PRINT : PRINT "       1 - LO
AD DATA SHEET"
7045 PRINT : PRINT "       2 - LO
AD FORMULA ARRAY"
7050 PRINT : PRINT "       3 - SA
VE DATA SHEET"
7055 PRINT : PRINT "       4 - SA
VE FORMULA ARRAY"
7056 PRINT : PRINT "       X - TO
 EXIT"
7060 LET A$=INKEY$: IF A$="" THE
N  GO TO 7060
7070 IF A$="1" THEN  GO TO 7100
7080 IF A$="2" THEN  GO TO 7200
7090 IF A$="3" THEN  GO TO 7300
7095 IF A$="4" THEN  GO TO 7400
7096 IF A$="X" THEN  GO SUB 1000
: GO SUB 1700: RETURN
7100 REM **** LOAD DATA *****
7110 GO SUB 7500
7120 LOAD I$ DATA M()
7130 GO TO 7010
7200  REM **** LOAD FORMULAE ***
*
7210 GO SUB 7500
7220 LOAD I$ DATA F$()
7230 GO TO 7010
7300 REM ***** SAVE DATA ***
7310 GO SUB 7500
7320 SAVE I$ DATA M()
7330 GO TO 7010
7400 GO SUB 7500
7420 SAVE I$ DATA F$()
7430 GO TO 7010
7500 PRINT AT 18,0
7510 INPUT "ENTER FILENAME";I$
7520 RETURN
```

Die Zukunft hat begonnen

Angesichts der rapiden Entwicklung der Microcomputertechnologie ist eine verbindliche Prognose über die Rechner des nächsten Jahrzehnts ausgesprochen schwierig. Dennoch wagen wir hier die Veröffentlichung eines Testberichts, wie er der Redaktion Anfang der 90er Jahre auf den Schreibtisch flattern könnte.

Laser lesen lassen

Der CD-ROM-Laserplattenspieler gilt heute als unerläßliches Peripheriegerät für jeden Mikro-Besitzer. Das Foto zeigt eins der ersten kommerziellen CD-ROM-Laufwerke, das Hitachi Mitte der achtziger Jahre mit einer Kapazität von 552 MByte und einer Übertragungsrate von 176 KByte/s herausbrachte



Die ausgehenden achtziger Jahre haben dem Heimcomputermarkt eine ganze Reihe von Leistungsmerkmalen beschert, die vorher nur bei Bürorechnern anzutreffen waren. Das betrifft sowohl die Software als auch die Hardware. Die gesteigerte Benutzerfreundlichkeit und Flexibilität der Systeme eröffnet dem Anwender eine Fülle interessanter Einsatzmöglichkeiten.

Der „Chestnut" ist der neuste Rechner aus der Kompakt-Serie von Computix für den privaten Anwender. Wie kürzlich durchgeführte Umfragen zeigten, wird die neue Heimcomputergeneration vom Markt voll akzeptiert: Der Benutzerkreis entspricht zahlenmäßig dem von HiFi-Anlagen. Das liegt nicht zuletzt daran, daß die Hersteller sich in der Produktgestaltung mehr als sonst üblich an den Bedürfnissen des Käufers orientieren.

Bei einem Preis von etwa 1500 Mark bietet der Chestnut alle wesentlichen Errungenschaften der letzten Jahre, nämlich einen integrierten Farbmonitor, ein doppelseitiges 3 ½-Zoll-Laufwerk, eine 10-MByte-Mini-Festplatte und ein Megabyte RAM-Kapazität als Grundausstattung. Eher konservativ ist allerdings die Entscheidung für den altgedienten 68000 von Motorola als CPU einzustufen. Mit Außenabmessungen von 300 × 240 × 120 mm folgt der Rechner dem Trend zu immer weniger Standflächenbedarf auf dem Tisch.

Die Tastenanordnung entspricht dem üblichen QWERTZ-Standard mit zehn zusätzlichen Funktionstasten und einem numerischen Block, der gleichzeitig als Cursorkreuz dient. Die Tastatur enthält ferner eine Reihe von Steuertasten wie ‚Ctrl' und ‚Alt', über die Sonderfunktionen ansprechbar sind.

Der Chestnut profitiert deutlich von den Fortschritten der Flachbildschirm-Technologie in letzter Zeit. Die Bildqualität ist hervorragend; vor allem entfallen die Verzerrungen an den Rändern.

Neben der Monitoreinheit finden Sie das Floppy-Laufwerk mit einer doppelseitigen Kapazität von einem Megabyte, darunter den Montageplatz für ein zweites Laufwerk. An der rechten Gehäuseseite sind der Anschluß für die Maus und eine Busschnittstelle vorgesehen, über die Zweitprozessoren, Video-Bildgreifer (Frame Grabber) und 1-MByte-RAM-Zusatzmodule zu betreiben sind.

Kommunikation

Die Rückwand enthält die üblichen Steckverbindungen für Peripheriegeräte und die Kommunikation. Ganz rechts ein Paar HiFi-Buchsen für den Anschluß einer Stereo-Anlage – damit können Sie die überlegenen Klangerzeugungs-Möglichkeiten voll ausreizen, die der Synthesizerchip Y/S2416 mit acht eingebauten

Oszillatoren und einem Tonumfang von fünf Oktaven bietet. In Verbindung mit einem richtigen Keyboard ist der Rechner uneingeschränkt polyphon nutzbar.

An den rückseitigen User Port können Sie außer einer Klaviatur auch Robotersysteme anschließen. Sie finden ferner ein Paar serielle MIDI-Buchsen. Trotz der gegenwärtigen Tendenz zum MIDI-P(Parallel)-Format hat sich Computix für das alte serielle System entschlossen, um die Kompatibilität mit den zahlreich verfügbaren MIDI-Lokalnetzpaketen nach alter Norm zu wahren.

Natürlich braucht der Synthesizer-Baustein nicht direkt programmiert zu werden, dafür läßt sich die eingebaute MIDI-Software verwenden. Der Synthesizerchip wird einfach als MIDI-Komponente spezifiziert und empfängt dann die entsprechenden Steuercodes.

Wie bei den meisten heutigen Micros steckt die ganze serielle Schnittstellenelektronik für das MIDI-System und das integrierte Telefonmodem in einem einzigen Baustein. Über das vorhandene Telefonnetz-Buchsenpaar sind sämtliche gängigen Kommunikationsangebote erreichbar. Die Software wird über das Fernsprechnetz direkt auf die Festplatte des Benutzers überspielt.

Wortschatz

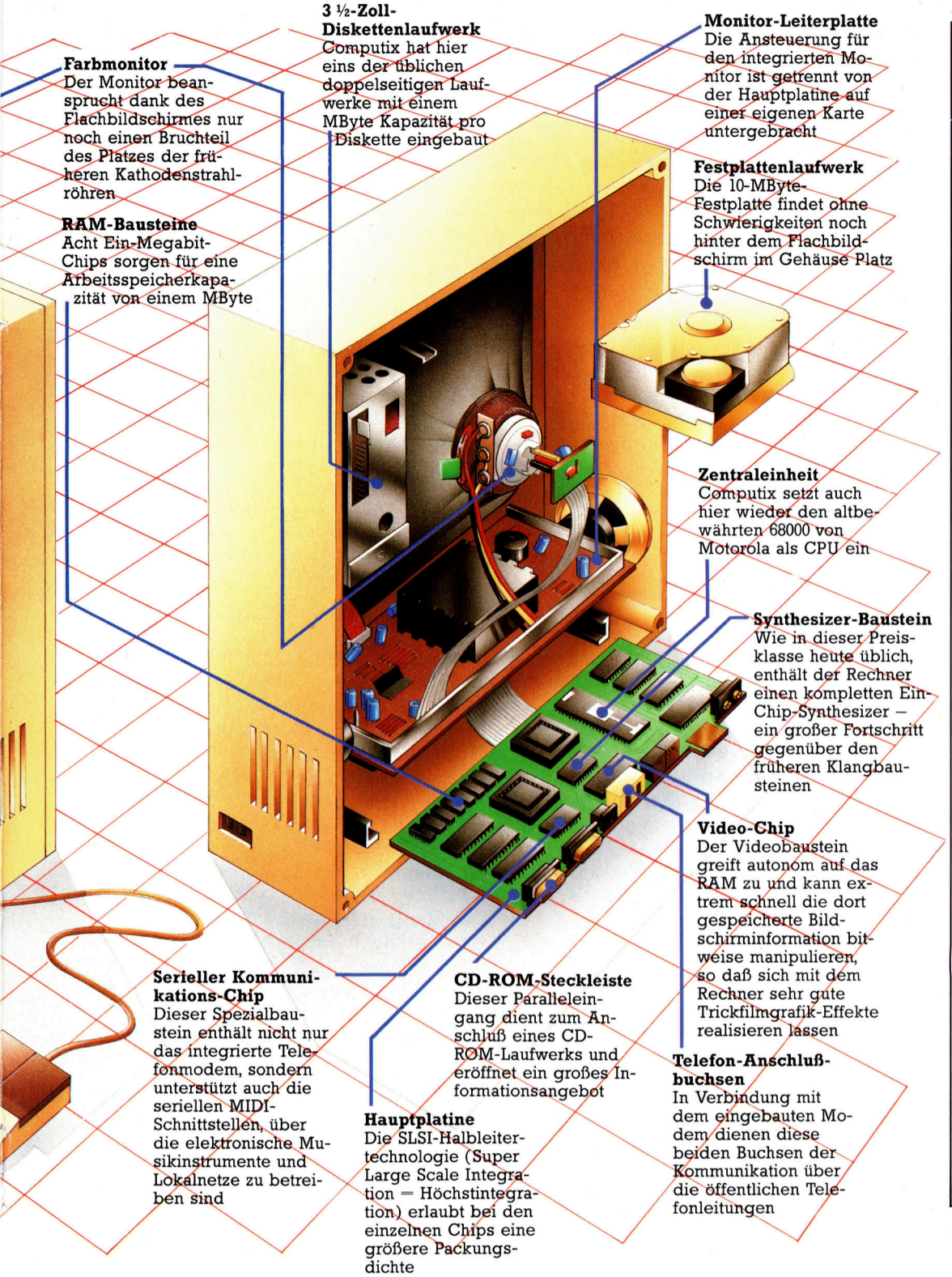
Wesentliche Pluspunkte der neuen Heimcomputergeneration stellen zweifellos die zuverlässige Spracherkennung und die Sprachsynthese dar – beides weniger durch technische Innovationen gefördert als durch den Preisverfall bei Chips während der letzten Jahre, trotz gestiegener Leistungsfähigkeit. Deshalb ließ sich die Speicher- und Verarbeitungskapazität so weit erhöhen, daß eine ausreichende Anzahl von Sprachschablonen zugänglich ist.

‚Mündliche' Anordnungen nimmt der Computer über ein Mikrofon und den rückseitigen Analogeingang zur Kenntnis, prüft die Schalldruckkurve auf Übereinstimmung mit einer der gespeicherten Schablonen und führt den betreffenden Befehl aus. Unter Verwendung des gleichen Schablonenvorrats erfolgt über den Synthesizerchip dann auch die Sprachausgabe. Der Wortschatz ist dabei schon recht beachtlich und verhilft dem Rechner zu einer begrenzten Dialogfähigkeit.

Für einen Drucker ist der übliche Centronics-Ausgang vorgesehen, und außerdem gibt es noch eine Parallelschnittstelle für ein CD-ROM-Laufwerk – sicher ein wesentliches Verkaufsargument, weil der Benutzer damit Zugang zu einem ungewöhnlich großen Informationsangebot erhält.

Der eingebaute hochauflösende Monitor kann über tausend verschiedene Farben gleichzeitig wiedergeben, und der Video-Blitter-Chip als schneller Bildschirm-Coprozessor sorgt für einen realistischen, ruckfreien Ablauf bewegter Szenen; die Grafik hält jedem Vergleich mit Konkurrenzprodukten in ähnlicher Preislage stand.

Das piktogrammgesteuerte Betriebssystem WIMPOS-3 unterscheidet sich von früheren Computix-Versionen im wesentlichen durch den vermehrten Einsatz der Spracheingabe beim Umgang mit den Fenstermenüs und Bildsymbolen. Allerdings arbeitete das Testmodell dabei nicht hundertprozentig zuverlässig, besonders wenn das Sprechtempo nicht bewußt



Farbmonitor
Der Monitor beansprucht dank des Flachbildschirmes nur noch einen Bruchteil des Platzes der früheren Kathodenstrahlröhren

RAM-Bausteine
Acht Ein-Megabit-Chips sorgen für eine Arbeitsspeicherkapazität von einem MByte

3 ½-Zoll-Diskettenlaufwerk
Computix hat hier eins der üblichen doppelseitigen Laufwerke mit einem MByte Kapazität pro Diskette eingebaut

Monitor-Leiterplatte
Die Ansteuerung für den integrierten Monitor ist getrennt von der Hauptplatine auf einer eigenen Karte untergebracht

Festplattenlaufwerk
Die 10-MByte-Festplatte findet ohne Schwierigkeiten noch hinter dem Flachbildschirm im Gehäuse Platz

Zentraleinheit
Computix setzt auch hier wieder den altbewährten 68000 von Motorola als CPU ein

Synthesizer-Baustein
Wie in dieser Preisklasse heute üblich, enthält der Rechner einen kompletten Ein-Chip-Synthesizer – ein großer Fortschritt gegenüber den früheren Klangbausteinen

Video-Chip
Der Videobaustein greift autonom auf das RAM zu und kann extrem schnell die dort gespeicherte Bildschirminformation bitweise manipulieren, so daß sich mit dem Rechner sehr gute Trickfilmgrafik-Effekte realisieren lassen

Telefon-Anschlußbuchsen
In Verbindung mit dem eingebauten Modem dienen diese beiden Buchsen der Kommunikation über die öffentlichen Telefonleitungen

Serieller Kommunikations-Chip
Dieser Spezialbaustein enthält nicht nur das integrierte Telefonmodem, sondern unterstützt auch die seriellen MIDI-Schnittstellen, über die elektronische Musikinstrumente und Lokalnetze zu betreiben sind

CD-ROM-Steckleiste
Dieser Paralleleingang dient zum Anschluß eines CD-ROM-Laufwerks und eröffnet ein großes Informationsangebot

Hauptplatine
Die SLSI-Halbleitertechnologie (Super Large Scale Integration = Höchstintegration) erlaubt bei den einzelnen Chips eine größere Packungsdichte

Computix Chestnut

ABMESSUNGEN:
ca. 300 × 240 × 120 mm

ZENTRALEINHEIT:
68 000 mit 8 MHz Taktfrequenz

SPEICHER:
1 MByte RAM, 128 KByte ROM

BILDSCHIRM:
Textformat 32 Zeilen zu 80 Zeichen, Grafikauflösung 640 × 256 Pixel, bis zu 1024 Farben gleichzeitig darstellbar

SCHNITTSTELLEN:
Buserweiterung, Maus-Anschluß, HiFi-Buchsen, User Port, je ein Paar MIDI- und Telefonanschlüsse, Analogeingang, CD-ROM- und Centronics-Parallelschnittstellen

SPRACHEN:
BASIC-System, C-Compiler

DISKETTENLAUFWERK:
3 ½-Zoll-Einzellaufwerk (doppelseitig, doppelte Dichte) mit einem MByte Kapazität; zweites Laufwerk auf Wunsch

BETRIEBSSYSTEM:
WIMPOS-3

DOKUMENTATION:
Die Dokumentation ist ausführlich und enthält alle erforderlichen Informationen über das System. Als echte Neuerung bietet Computix die Anleitungen außer in Buchform auch als CD-ROM an

STÄRKEN:
Der Chestnut wartet mit einer ganzen Reihe von Leistungsmerkmalen auf, die es bisher nur bei teureren Maschinen gab

SCHWÄCHEN:
Enttäuschend ist, daß der Rechner so wenig an wirklicher Innovation vorzuweisen hat. Er dürfte Schwierigkeiten haben, sich durchzusetzen

gezügelt wurde. Es erscheint nach wie vor empfehlenswert, die in Ehren ergraute Maus in die Hand zu nehmen, um das versehentliche Löschen von Dateien infolge Fehlinterpretation gesprochener Kommandos mit Sicherheit zu vermeiden.

Das mitgelieferte Softwarepaket auf der Festplatte beinhaltet das Textprogramm ‚ChestWrite', das vollständige Grafik-System ‚ChestPaint', das Spracherkennungs/Synthese-Paket ‚ChestTalk' zur Entwicklung eigener Sprachschablonen und ein kleines Expertensystem namens ‚ChestDoc' für die medizinische Selbstdiagnose. Die MIDI- und Telefonmodem-Software ist integraler WIMPOS-Bestandteil, und für die Programmentwicklung stehen ein BASIC-System und ein C-Compiler zur Verfügung.

Entgegen den lautstarken Werbesprüchen von Computix in den vergangenen Monaten ist Chestnut keine echte Neuerung im Bereich der Computertechnik.

Es kommen neue Schneider-Befehle

Zum Abschluß unserer Serie über das Schneider-Betriebssystem sehen wir uns die Systemerweiterungen an, mit denen neue Befehle an das Locomotive BASIC angefügt werden können.

Die Kernroutinen sind das Herz des Schneider-Betriebssystems. Sie erledigen die interne Verwaltung des Computers und steuern Ereignisse, Interrupts und den Speicher.

Die Kernroutinen werden zumeist mit Adressen zwischen &BCC8 und &BD10 angesprochen. Sie haben jedoch auch einen eigenen Jumpblock, der in einen oberen (&B9000 – &B921) und unteren Bereich (&0000 – &003B) geteilt ist. Im Gegensatz zum Haupt-Jumpblock des Firmwarebereiches läßt er sich jedoch nicht vom Anwender patchen. Einige dieser Routinen sind außerordentlich praktisch. Wir haben sie in einer Tabelle zusammengestellt.

Die Kernroutinen können unter anderem externe Befehle bearbeiten, die sich im RAM oder in den ROMs befinden. Befehle, die ins RAM geladen werden, heißen „Resident System Extension" (RSX). Externe Befehle können beliebig lang sein und jede Funktion übernehmen. So sind die AMSDOS-Befehle |DIR und |ERA beispielsweise Routinen zur Bearbeitung von Diskettendateien. Auch das gesamte BASIC arbeitet als RSX – geben Sie |BASIC ein, und beobachten Sie, was da passiert. Externe Befehle lassen sich auf zwei Arten anlegen. Erstens als Befehle, die beim Einschalten aus dem ROM geladen werden, und zweitens als programmgesteuerte Befehle, die aus RAM oder ROM geladen werden. In beiden Fällen werden die Befehle über eine Befehlstabelle angesprochen, deren Format im nebenstehenden Kasten beschrieben ist. Die Befehlsnamen können bis zu 16 Zeichen lang sein, wobei Bit 7 des letzten Zeichens gesetzt sein muß. Die Zeichen, die Sie in Ihren Computer eingeben, dürfen jeden Sieben-Bit-Code annehmen.

Anmeldung

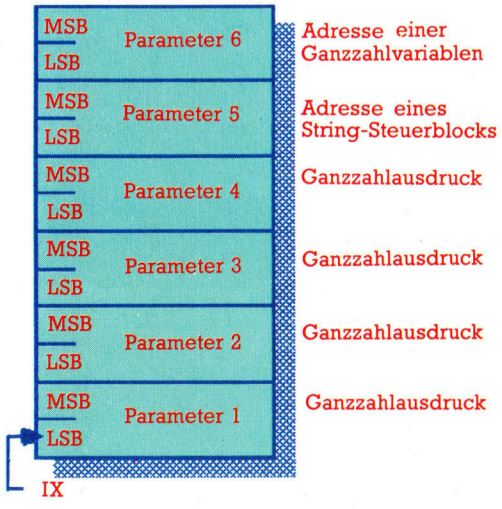
RSX-Befehle (d. h. ins RAM geladene Befehle) werden mit der Kernroutine KL_LOG_EXT „angemeldet" (d. h. ihre Existenz der Firmware angezeigt). Kl_LOG_EXT wird über &BCD1 mit der Adresse der Befehlstabelle in BC aufgerufen. HL muß die Adresse eines Vier-Byte-Arbeitsbereiches enthalten, den die Kernroutine braucht. Beide Adressen dürfen nur in den mittleren 32K des RAM liegen.

Bei Befehlen, die im ROM untergebracht sind, wird der Firmware das ROM und nicht die Befehlstabelle angemeldet. Das ROM muß dabei dem im Bild gezeigten Format entsprechen. Es können Vordergrund-, Hintergrund- oder Erweiterungs-ROMs definiert werden. Vordergrund-ROMs enthalten Programme, die beim Aufruf die Steuerung der Firmware übernehmen. Beispiele sind das BASIC-Rom oder die ROMs anderer Sprachen wie PASCAL.

Hintergrund-ROMs enthalten externe Befehle, die vom Vordergrundprogramm nicht angemeldet zu werden brauchen. In Erweiterungs-ROMs wird der Code von Vordergrundprogrammen oder externen Befehlen untergebracht, der nicht mehr in das entsprechende ROM paßt.

Jedes ROM wird im Bereich von 0 bis 251 adressiert, wobei bestimmte Typen Einschränkungen unterworfen sind. So können Vordergrund-ROMs alle Adressen annehmen, vorausgesetzt, die darunterliegenden Adressen sind von anderen ROMs belegt (die Kernroutine sucht die ROMs von Adresse 0 an aufwärts, bis sie den ersten freien Platz findet). Hintergrund-ROMs werden auf dem Commodore# PC 464 in den Adressen 1–7 eingebaut und in 1–15 auf den beiden Schneider-Computern 664 und 6128.

Die Adressierung der Parametertabelle



Beim Einsprung in eine Befehlsroutine, die vom BASIC aus aufgerufen wird, werden die Parameter in einem Parameterblock gespeichert. Das Register IX zeigt dabei auf die Basisadresse des Blocks, während das Register A die Anzahl der Parameter übergibt. Das Bild zeigt die Struktur eines Blocks, der nach der Ausführung von |COMMAND,5,24.6,a%, x,@b$, @x% angelegt wurde.

Wenn beispielsweise FORTH als ROM 0 eingebaut ist und sich bei ROM 1 ein Hintergrund-ROM befindet, sollte ein zweites Vordergrund-ROM bei der ROM-Adresse 2 installiert werden, um die Kontinuität sicherzustellen. Ein zweites Hintergrund-ROM könnte allerdings in die Adressen zwischen 3 und 7 (15 auf dem 664/6128) eingesetzt werden und ein Erweiterungs-ROM in jede beliebige Adresse.

Das Anschalten der Maschine meldet automatisch alle Vordergrund-ROMs an. Hintergrund-ROMs können dann entweder mit KL_INIT_BACK einzeln oder mit KL_ROM_WALK insgesamt initialisiert werden. Das Ansprechen des BASIC-ROMs initialisiert automatisch alle Hintergrund-ROMs.

Da die Erweiterungs-ROMs den gleichen Speicherbereich belegen wie das eingebaute BASIC-ROM (&C000 bis &FFFF), muß vor deren Einsatz das BASIC-ROM abgeschaltet und das gewünschte ROM angeschaltet werden. Es gibt mehrere Kernroutinen zum An- und Abschalten der oberen und unteren ROMs, mit denen ein Programmierer die Hardware nicht direkt ansprechen muß.

Nach der Anmeldung lassen sich die externen Befehle entweder direkt vom BASIC aus oder über die Kernroutine KI_FIND_COMMAND aufrufen. Dieser Aufruf durchsucht die RSX- und die Hintergrund-ROMs nach einem Befehlsnamen. Wird er gefunden, erscheint die Einsprungadresse in HL, und das Register C enthält die Nummer des Hintergrund-ROMs (wenn die Befehlsroutine im RAM liegt, kann C ignoriert werden). Wird kein Befehl gefunden, liefert die Routine im Übertrag ein FALSCH. KI_FIND_COMMAND wird über &BCD4 aufgerufen, HL enthält dabei die Adresse des Befehlsnamens, bei dessen letztem Zeichen Bit 7 gesetzt sein muß.

Speicherplatz

Externe Befehle lassen sich leicht vom BASIC aus aktivieren, indem vor den Befehlsnamen das Symbol | (Shift @) gestellt wird. BASIC kann dabei auch Parameter übergeben, die allerdings durch Kommas getrennt sein müssen. Hier ein Beispiel:

```
|COMMAND,1,2,3,x,y*3,@$,@X%
```

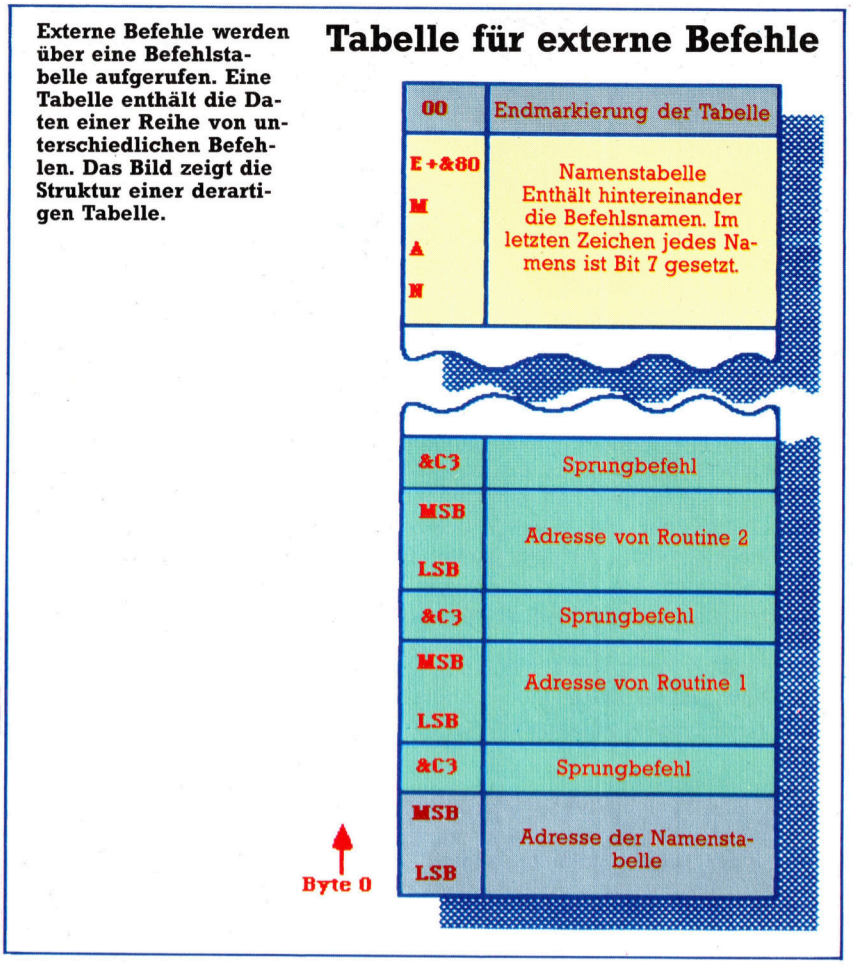
Die Firmware übergibt die Steuerung an die Befehlsroutine, wobei Register A die Anzahl der Parameter enthält und IX auf eine Tabelle mit Parameterinformationen zeigt. In dieser vorliegenden Tabelle sind jedem Parameter jeweils zwei Bytes zugeordnet, die folgendes Format besitzen:

Ganzzahl oder Ganzzahlausdruck — 16-Bit-Ganzzahlen mit Vorzeichen
Reelle Zahlen — 16-Bit-Ganzzahlen ohne Vorzeichen

Auch die Adressen von String- oder numerischen Variablen können mit dem Symbol @ übergeben werden. Ganzzahlvariablen hinterlassen dabei die Zwei-Byte-Adresse ihres Speicherplatzes und Stringvariablen die Adresse des String-Steuerblocks. Leider läßt sich nicht feststellen, ob ein Tabelleneintrag die Adresse eines String-Steuerblocks enthält oder eine absolute Ganzzahl. Hier muß die Befehlsroutine entscheiden, wie der Parameter interpretiert werden soll.

Beim Einsprung in die Befehlsroutine zeigt IX auf das LSB des letzten angegebenen Parameters (im obigen Beispiel ist dies die

Externe Befehle werden über eine Befehlstabelle aufgerufen. Eine Tabelle enthält die Daten einer Reihe von unterschiedlichen Befehlen. Das Bild zeigt die Struktur einer derartigen Tabelle.



Die Adressen der Kernroutinen

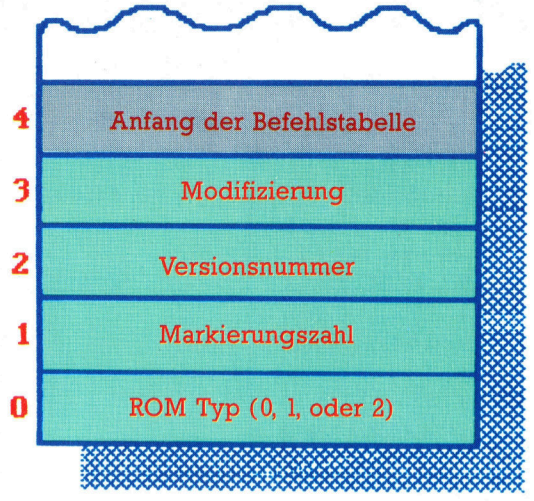
&BCC8 RESET	Reset für Kernroutinen, löscht Ereignisse, Warteschlangen etc.
&BCCB ROM WALK	Initialisiert alle Hintergrund-ROMs
&BCD1 LOG EXT	RSX-Befehlstabelle anmelden
&BCD4 FIND COMMAND	RSX suchen; Befehls adresse im Hintergrund- oder Vordergrund-ROM finden
&B90F ROM SELECT	Ein bestimmtes oberes ROM anwählen
&B912 CURR SELECTION	Testen, welches obere ROM gerade angewählt ist

Adresse von X%). Die Inkrementierung von IX indiziert dabei das LSB des vorletzten Parameters. Unsere Graphik zeigt, wie die Tabelle bei unserem Beispiel indiziert wird.

Oft empfiehlt es sich, zuerst den ersten Parameter anzusehen und dann die Parameter in Eingabereihenfolge abzurufen (nicht umgekehrt). Die Routine unseres ersten Listings ändert IX, das dann auf das LSB des ersten Parameters zeigt. Für den zweiten Parameter wird dann IX einmal dekrementiert. Für das LSB des zweiten Parameters muß IX dann nochmals dekrementiert werden.

RSX-Befehle sind für den Maschinencodeprogrammierer recht einfach einzusetzen, da sie nicht – wie bei vielen anderen Maschinen – über CALL- oder SYS-Befehle aufgerufen werden müssen. Aus diesem Grund hat der Befehl CALL auf den Schneider-Computern nur untergeordnete Bedeutung, es zahlt sich die Einrichtung von Systemerweiterungen aus.

Aufbau eines Erweiterungs-ROMs



Die Tabelle zeigt die Bytestruktur, mit der das Betriebssystem ein einzelnes Erweiterungs-ROM identifiziert und „anmeldet". Eine spätere Anfrage an das Betriebssystem liefert die Adressen der Befehlsroutinen, die in der Befehlstabelle des externen ROMs definiert sind.

Übergabe der Programmsteuerung

Parameterzeiger

Diese kurze Routine ändert IX beim Einsprung in eine externe Befehlsroutine, so daß sie auf den ersten Eingabeparameter zeigt:

```
        ;entry: A contains number of parameters
        ;       IX contains address of table
        ;exit:   IX   contains          the address
        ;        of   1st               entry in table
        ;       DE corrupt, other registers
        ;       preserved
        ;
5F              ld   e,a                ;copy number to E
1D              dec  e                  ;no of parameters-1
CB23            sla  e                  ;*2
1600            ld   d,0                ;get offset in DE
DD19            add  ix,de              ;add it to start
        ;IX now points to LSB of 1st entry
        ;in the parameter table
```

Firmwareaufruf

Dieses Listing enthält einen RSX, über den die Firmware (mit entsprechend gesetzten Registern) vom BASIC aus direkt aufgerufen werden kann. Die Befehlssyntax sieht folgendermaßen aus:

|FIRMCALL, <entry address>, [,<A> <HL> [,<BC>[, DE]]]

```
        kl_log: equ  #bcd1
01902C  logon:  ld   bc,commands        ;point to table
219E2C          ld   hl,scratch         ;and scratchpad
CDD1BC          call kl_log             ;log on command
C9              ret                     ;and that's it
952C    comman: defw name               ;point to name
C3A22C          jp   firmcall           ;entry point
4649524D name:  defm "FIRMCAL"          ;set bit 7 of...
CC              defb "L"+#80            ;..last byte of name
00              defb 0
        scratc: defs 4                  ;reserve data area
        ; |FIRMCALL, entry, a, hl, bc, de
5F      firmca: ld   e,a                ;copy number to E
1D              dec  e                  ;no of parameters-1
CB23            sla  e                  ;*2
1600            ld   d,0                ;get offset in DE
DD19            add  ix,de              ;add it to start
DDE5            push ix
E1              pop  hl
FE06            cp   6                  ; up to 5 parameters
D0              ret  nc                 ;too many, abort
B7              or   a                  ;any parameters?
C8              ret  z                  ; no, abort
47              ld   b,a                ;save count in B
23              inc  hl                 ;get MSB of address
7E              ld   a,(hl)             ;read it in
32E62C          ld   (entry+1),a        ;save it
2B              dec  hl                 ;LSB
7E              ld   a,(hl)
32E52C          ld   (entry),a
2B              dec  hl                 ;MSB of AF
2B              dec  hl                 ;LSB of AF
05              dec  b                  ;got enough?
2822            jr   z,call             ;yes, do the call
7E              ld   a,(hl)             ;else read in A
05              dec  b                  ;enough?
281E            jr   z,call             ;yes, do the call
2B              dec  hl                 ;MSB of HL
56              ld   d,(hl)
2B              dec  hl
5E              ld   e,(hl)
EB              ex   de,hl              ;get into HL
05              dec  b                  ;enough?
2816            jr   z,call             ;yes do the call
EB              ex   de,hl              ;get pointer back
2B              dec  hl                 ;MSB of BC
F5              push af                 ;save A for now
7E              ld   a,(hl)
2B              dec  hl                 ;LSB of BC
4E              ld   c,(hl)
05              dec  b                  ;done enough
47              ld   b,a                ;set up BC anyway
EB              ex   de,hl              ;and HL
2003            jr   nz,miss            ;get the last one
F1              pop  af                 ;restore the stack
1808            jr   call               ;and do the call
F1      miss:   pop  af
EB              ex   de,hl              ;get table back
D5              push de                 ;save hl value
2B              dec  hl                 ;MSB of DE
56              ld   d,(hl)
2B              dec  hl                 ;LSB of DE
5E              ld   e,(hl)
E1              pop  hl                 ;get HL back
C3      call:   defb #c3                ;jump instruction
0000    entry:  defw 0                  ;jump adress
        ;the jump address is patched
        ;into the routine
```

Text soll wieder länger werden

Wir schließen unsere Serie über Datenkompression mit einem Programm für 6502-Rechner und einem Treiberprogramm, mit dem sich Kompression und Expansion von BASIC aus steuern läßt.

Wer als 6502-Anwender keinen Assembler hat oder Zeit sparen möchte, kann das Unterprogramm für die Textkompression bzw. -Expansion auch mit dem 6502-Laderprogramm in den Speicher POKEn. Nach einmaligem Starten des Programms kann es gelöscht und durch das BASIC-Treiberprogramm ersetzt werden. Das Treiberprogramm ist sowohl für die 6502- als auch für die Z80-Version einsetzbar.

Das BASIC-Treiberprogramm fordert die Eingabe des zu komprimierenden Teststrings sowie einer Adresse an, ab der die Daten gespeichert werden sollen. Diese Adresse müssen Sie notieren, damit sie die Daten wiederfinden können – das Expansionsprogramm muß schließlich wissen, wo es mit der Arbeit beginnen soll.

Das Programm für den Z80 beginnt am Speicherplatz 30000. Wenn Sie einen Assembler haben, können Sie den Code durch Änderung des ORG-Befehls verschieben. Das BASIC-Treiberprogramm läßt sich für den Einbau in eigene Programme modifizieren.

Textexpansion mit dem 6502

Das folgende Assemblerprogramm muß vor den DATA-Tabellen in das Kompressionsprogramm eingefügt werden. Dieses Programm haben wir auf einem Commodore 64 erstellt. Für den Acorn B müssen Sie folgende Änderungen vornehmen:

- Die Adressen des Zero-Page-Zeigers (ZPTR1 bis ZPTR2) auf die Adressen &80 bis &87 legen.
- Das Befehlsformat LABEL*=*+2 auf .LABEL EQUW abändern.
- Befehlsformat LABEL *=*+1 auf .LABEL EQUB abändern.
- Das Befehlsformat .BYT in EQUS umwandeln.
- Adressen, die mit CALL abgerufen werden, hängen davon ab, wohin das Programm assembliert wurde. Die Labels START und EXPAND geben die CALL-Adressen der Kompression- und Expansionsprogramme an, die vom BASIC aus zugänglich sind.

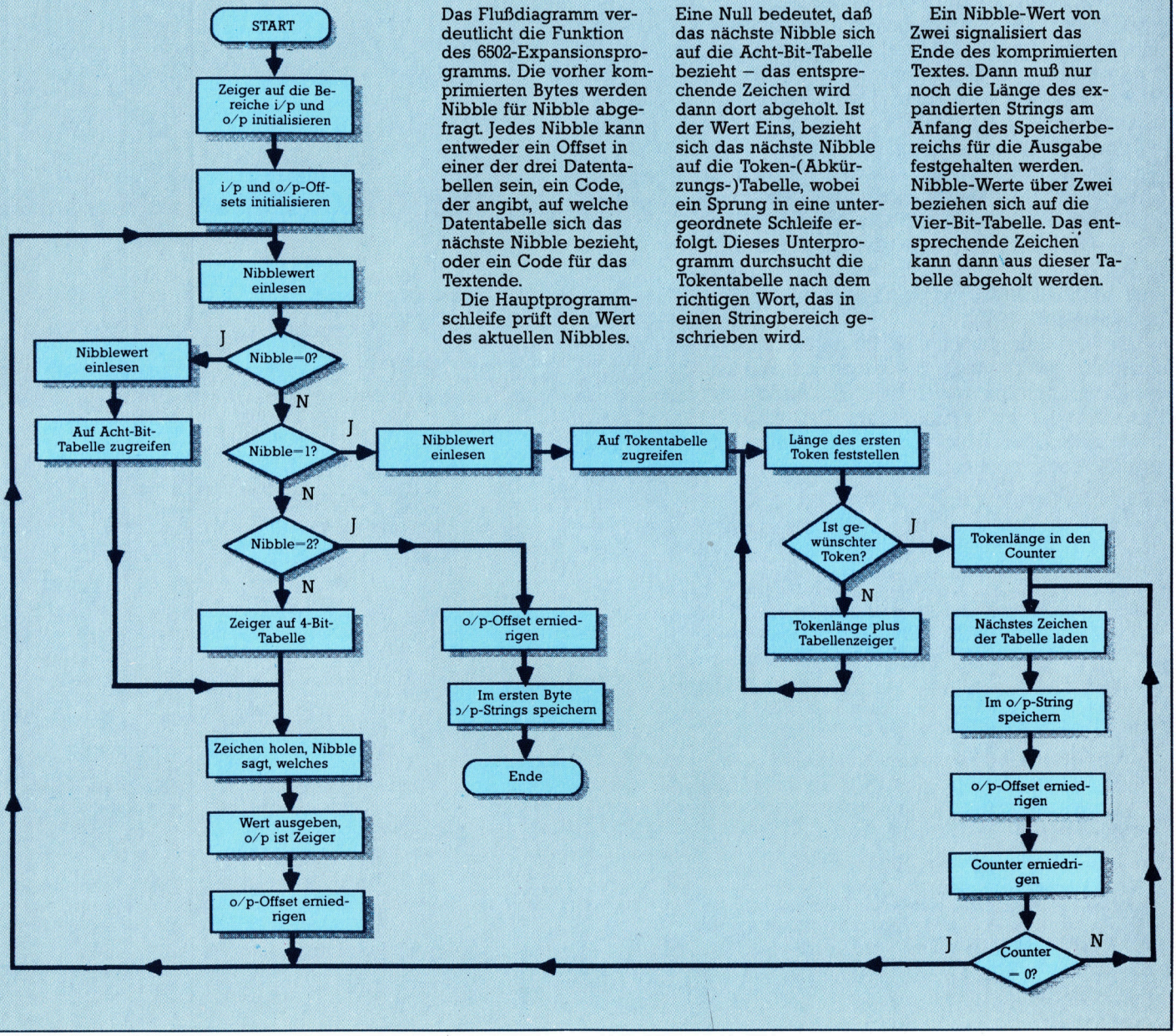
```
;
;++++ 6502 EXPAND ROUTINE +++++
EXPAND LDA INPUT
       STA ZPTR1
       LDA INPUT+1       ;SET UP 0 PAGE
       STA ZPTR1+1       ;PTRS TO POINT
       LDA OUTPUT        ;AT INPUT AND
       STA ZPTR2         ;OUTPUT AREAS
       LDA OUTPUT+1
       STA ZPTR2+1

       LDA #255          ;INITIALISE
       STA MASK          ;OFFSETS
       LDA #1            ;AND MASK
       STA LEN           ;INIT I/P OFFSET
       STA OUTOFF        ;INIT O/P OFFSET
;++ MAIN LOOP STARTS HERE ++
NEXNIB JSR GETNIB
       BEQ OUT8BT        ;0 NIBBLE MEANS 8BIT CODE
       CMP #2
       BEQ EXIT          ;IS IT END MARKER?
       BCC OUTTOK        ;IS IT TOKEN?
;++HANDLE 4BIT CODE ++
       TAY               ;NIBBLE VALUE INTO Y
       LDA #<TAB4BT
       STA ZPTR3         ;SET ZPTR3 TO
       LDA #>TAB4BT      ;START OF 4 BIT
       STA ZPTR3+1       ;TABLE
OUTCHR LDA (ZPTR3),Y     ;GET CHAR FROM TABLE
       LDY OUTOFF        ;SET UP Y FOR O/P
       STA (ZPTR2),Y     ;STORE IN O/P STRING
       INC OUTOFF        ;BUMP O/P OFFSET POINTER
       JMP NEXNIB        ;GO FOR NEXT NIBBLE
;++ HANDLE 8BIT CODE ++
OUT8BT JSR GETNIB        ;GET CODE NIBBLE
       TAY               ;PUT IT IN Y
       LDA #<TAB8BT      ;SET ZPTR3 TO
       STA ZPTR3         ;POINT AT START
       LDA #>TAB8BT      ;OF 8 BIT TABLE
       STA ZPTR3+1
       JMP OUTCHR        ;OUTPUT IT
;++ HANDLE TOKEN ++
OUTTOK JSR GETNIB        ;GET TOKEN PTR
       STA TABCNT        ;STORE IT AWAY
       LDA #<TOKTAB      ;SET ZPTR3 TO
       STA ZPTR3         ;LOOK AT TOKEN
       LDA #>TOKTAB      ;TABLE START
       STA ZPTR3+1
       LDX #0
       LDY #0
TOKA   LDA (ZPTR3),Y     ;GET LENGTH OF TOKEN
       CPX TABCNT        ;IF WE'VE GOT THE
       BEQ FOUND         ;ONE WE WANT -BRANCH
       CLC
       ADC #1
       ADC ZPTR3         ;SET ZPTR3 TO START
       STA ZPTR3         ;OF NEXT TOKEN
       LDA ZPTR3+1
       ADC #0
       STA ZPTR3+1
       INX
       BNE TOKA          ;THIS SHOULD ALWAYS BRANCH!
FOUND  TAX               ;PUT TOKEN LENGTH IN X
       LDY #1            ;INITIALISE Y
FOUND1 LDA (ZPTR3),Y     ;GET TOKEN CHAR
       STY TEMP          ;TEMP STORE Y
       LDY OUTOFF        ;GET O/P OFFSET
       STA (ZPTR2),Y     ;STORE CHAR IN O/P STRING
       INC OUTOFF        ;BUMP O/P OFFSET
       LDY TEMP          ;GET TOKEN OFFSET BACK
       INY
       DEX
       BNE FOUND1        ;IF NOT END, GET NEXT CHAR
       JMP NEXNIB        ;REJOIN MAIN LOOP
;++ HANDLE EXIT ++
EXIT   LDA OUTOFF        ;GET O/P OFFSET
       SEC
       SBC #1            ;REDUCE BY ONE
       LDY #0            ;AND STORE AT START
```

```
        STA (ZPTR2),Y    ;OF O/P STRING
        RTS              ;RETURN TO SYSTEM
;++ GET A NIBBLE SUBROUTINE ++
GETNIB  LDA MASK
        BNE RIGHT        ;IS MASK 255?
        LDA #255         ;SET UP MASK
        STA MASK         ;FOR NEXT TIME
        LDY LEN          ;GET I/P OFFSET
        LDA (ZPTR1),Y    ;GET I/P BYTE
        AND #15          ;MASK IT
        INC LEN          ;BUMP I/P OFFSET
        RTS
RIGHT   LDA #0           ;SET UP MASK
        STA MASK         ;FOR NEXT TIME
        LDY LEN          ;GET I/P OFFSET
        LDA (ZPTR1),Y    ;GET I/P BYTE
        LSR A            ;MOVE LEFT NIBBLE
        LSR A            ;DOWN INTO RIGHT
        LSR A            ;HALF OF BYTE
        LSR A
        AND #15          ;MASK IT
        RTS
```

BASIC-Treiberprogramm Z80/6502

```
100 REM ********************************
110 REM ** Program to compress/expand **
120 REM **                            **
130 REM * a 255-byte string (U/C only) *
140 REM ********************************
150 CLS
160 INPUT "6502 version (Y/N)";a$
170 IF a$="y" OR a$="Y" THEN cf=1
180 INPUT "Expand or Compress (E/C)?";a$
190 IF a$="" THEN GOTO 180
200 IF a$="e" OR a$="E" THEN ec=1:GOTO 2
60
210 IF a$="c" OR a$="C" THEN ec=0:GOTO 4
30
```

Expansion



Das Flußdiagramm verdeutlicht die Funktion des 6502-Expansionsprogramms. Die vorher komprimierten Bytes werden Nibble für Nibble abgefragt. Jedes Nibble kann entweder ein Offset in einer der drei Datentabellen sein, ein Code, der angibt, auf welche Datentabelle sich das nächste Nibble bezieht, oder ein Code für das Textende.

Die Hauptprogrammschleife prüft den Wert des aktuellen Nibbles. Eine Null bedeutet, daß das nächste Nibble sich auf die Acht-Bit-Tabelle bezieht – das entsprechende Zeichen wird dann dort abgeholt. Ist der Wert Eins, bezieht sich das nächste Nibble auf die Token-(Abkürzungs-)Tabelle, wobei ein Sprung in eine untergeordnete Schleife erfolgt. Dieses Unterprogramm durchsucht die Tokentabelle nach dem richtigen Wort, das in einen Stringbereich geschrieben wird.

Ein Nibble-Wert von Zwei signalisiert das Ende des komprimierten Textes. Dann muß nur noch die Länge des expandierten Strings am Anfang des Speicherbereichs für die Ausgabe festgehalten werden. Nibble-Werte über Zwei beziehen sich auf die Vier-Bit-Tabelle. Das entsprechende Zeichen kann dann aus dieser Tabelle abgeholt werden.

```
220 GOTO 180
230 REM
240 REM ***Expand a compressed string***
250 REM
260 PRINT "Where is the string to expand
?"
270 INPUT i
280 PRINT "Where do you want the output?
"
290 REM Be careful not to overwrite mach
ine code!
300 INPUT o
310 IF cf=0 THEN GOSUB 660:REM z80 versi
on
320 IF cf=1 THEN GOSUB 700:REM 6502 vers
ion
330 FOR x=1 TO PEEK(o)
340 PRINT CHR$(PEEK(o+x));
350 NEXT x
360 PRINT; "Continue (y/n)? ": INPUT a$
370 IF a$()"Y" AND a$()"y" THEN STOP
380 PRINT: GOTO 180
390 STOP
400 REM
410 REM ****** Compress a string ******
420 REM
430 PRINT "What shall I compress? Rememb
er that"
440 PRINT "the routine accepts upper cas
e letters only."
450 PRINT
460 INPUT "String to compress: ";a$
470 PRINT "Where shall I put it?"
480 REM See warning in line 290!
490 INPUT i
500 PRINT "Where do you want the output?
"
510 INPUT o
520 POKE i,LEN(a$)
530 FOR x=1 TO LEN(a$)
540 POKE i+x,ASC(MID$(a$,x,1))
550 REM poke i+x, code a$(x to x) on Spe
ctrum
560 NEXT x
570 GOSUB 660
580 PRINT "Output length ";PEEK (o)
590 PRINT "Compression achieved: ";100-P
EEK(o)/LEN(a$)*100;"%"
600 PRINT: "Continue (y/n)? ": INPUT a$
610 IF a$()"Y" AND a$()"y" THEN STOP
620 PRINT: GOTO 180
630 POKE 30003,INT(i/256)
640 POKE 30002,i-PEEK(30003)*256
650 POKE 30005,INT(o/256)
660 POKE 30004,o-PEEK(30005)*256
670 RETURN
680 IF ec=1 THEN CALL 30270:RETURN
690 CALL 30000:IF PEEK(30006)()0 THEN PR
INT "Compression Failed":STOP
700 REM 6502 Commodore version
710 POKE 49153, INT(0/256)
720 POKE 49152, i-PEEK
730 POKE 49155, INT(i/256)
740 POKE 49154, i-PEEK(49155)*256
750 IF ec=1 THEN sys 49163
760 sys 49518: IF PEEK(49156)()0 THEN PR
INT"compression failed":STOP
770 RETURN
```

BASIC-Ladeprogramm für 6502-Rechner

```
10 REM ********************************
15 REM ** COMMODORE 64 COMPRESSION   **
20 REM ** AND EXPANSION BASIC LOADER **
25 REM ********************************
30 FOR I=49163 TO 49807:READ A:POKEI,A
35 CC=CC+A:NEXT I
40 READ CS:IF CS()CC THEN PRINT"ERROR":STOP
100 DATA173,2,192,133,139,173,3,192
110 DATA133,140,173,0,192,133,141,173
120 DATA1,192,133,142,169,1,141,7,192
130 DATA160,0,177,139,141,6,192,169
140 DATA255,141,5,192,200,177,139,32
150 DATA39,193,208,54,32,129,192,240
160 DATA55,32,242,192,240,10,32,2,193
170 DATA72,169,0,32,65,193,104,32,65
180 DATA193,204,6,192,144,220,169,0
190 DATA141,4,192,169,2,32,65,193,173
200 DATA5,192,240,3,206,7,192,160,0
210 DATA173,7,192,145,141,96,169,255
220 DATA141,4,192,96,72,169,1,32,65
230 DATA193,104,32,65,193,76,79,192,72
240 DATA152,72,140,10,192,165,139,24
250 DATA109,10,192,133,251,165,140,105
260 DATA0,133,252,169,83,133,254,169
270 DATA194,133,255,169,0,141,8,192
280 DATA160,0,177,254,72,170,240,65
290 DATA165,254,24,105,1,133,254,165
300 DATA255,105,0,133,255,177,251,209
310 DATA254,208,22,200,202,208,246,104
320 DATA104,136,140,10,192,24,109,10
330 DATA192,168,104,173,8,192,162,0,96
340 DATA104,141,10,192,165,254,24,109
350 DATA10,192,133,254,165,255,105,0
360 DATA133,255,238,8,192,76,162,192
370 DATA104,104,168,104,162,255,96,72
380 DATA140,10,192,169,51,133,251,169
390 DATA194,133,252,104,76,15,193,72
400 DATA140,10,192,169,67,133,251,169
410 DATA194,133,252,104,160,0,209,251
420 DATA240,11,200,192,16,208,247,172
430 DATA10,192,162,255,96,152,172,10
440 DATA192,162,0,96,201,32,240,16,201
450 DATA44,240,12,201,46,240,8,201,65
460 DATA144,7,201,91,176,3,162,0,96
470 DATA162,255,96,72,140,10,192,172,7
480 DATA192,173,5,192,208,17,104,17
490 DATA141,145,141,238,7,192,172,10
500 DATA192,169,255,141,5,192,96,104
510 DATA10,10,10,10,145,141,172,10,192
520 DATA169,0,141,5,192,96,173,2,192
530 DATA133,139,173,3,192,133,140,173
540 DATA0,192,133,141,173,1,192,133
550 DATA142,169,255,141,5,192,169,1
560 DATA141,6,192,141,7,192,32,13,194
570 DATA240,28,201,2,240,106,144,37
580 DATA168,169,51,133,251,169,194,133
590 DATA252,177,251,172,7,192,145,141
600 DATA238,7,192,76,143,193,32,13,194
610 DATA168,169,67,133,251,169,194,133
620 DATA252,76,163,193,32,13,194,141,9
630 DATA192,169,83,133,251,169,194,133
640 DATA252,162,0,160,0,177,251,236,9
650 DATA192,240,16,24,105,1,101,251
660 DATA133,251,165,252,105,0,133,252
670 DATA232,208,233,170,160,1,177,251
680 DATA140,10,192,172,7,192,145,141
690 DATA238,7,192,172,10,192,200,202
700 DATA208,236,76,143,193,173,7,192
710 DATA56,233,1,160,0,145,141,0,173,5
720 DATA192,208,16,169,255,141,5,192
730 DATA172,6,192,177,139,41,15,238,6
740 DATA192,96,169,0,141,5,192,172,6
750 DATA192,177,139,74,74,74,74,41,15
760 DATA96,0,0,0,70,76,68,72,83,73,82
770 DATA78,79,65,84,69,32,67,77,85,71
780 DATA89,80,87,66,86,75,88,74,81,90
790 DATA44,46,3,84,72,69,4,84,72,73,83
800 DATA4,84,72,65,84,2,73,70,3,89,79
810 DATA85,2,77,69,3,87,65,83,2,72,69
820 DATA3,83,72,69,4,84,72,69,89,2,79
830 DATA70,2,73,84,2,73,83,3,70,79,82
840 DATA2,79,78,2,84,79,0,7
850 DATA76822:REM*CHECKSUM*
```

Ihren Einsatz bitte

Wir beenden unser 17+4-Programmierungs-Projekt mit den Wettroutinen und dem Titelbild. Die Wettroutinen ermöglichen den Starteinsatz und helfen Ihnen, diesen zu verdoppeln.

Das Wetten ist ein fester Bestandteil von 17+4. Das Programm stellt Ihnen einen Anfangseinsatz von 10 000 Pfund zur Verfügung. Das verbleibende Spielerkapital wird in der Variablen SK gehalten. Unsere Programmversion bedient sich des folgenden Wettsystems:

- Der Spieler muß zu Beginn jeder Runde 50 Pfund setzen, um in das Spiel einzusteigen.
- Diese Anfangswette wird automatisch vom Programm vorgenommen.

Nach Erhalt einer Karte kann der Spieler mittels einer Zusatzwette (bis zu 1000 £) eine zweite Karte kaufen.

Danach können weitere Karten an den Spieler ausgegeben werden, bis er stickt oder sein Blatt sprengt. Der Spieler kann auch seine Wette verdoppeln und bekommt eine weitere Karte.

Hat der Spieler das bessere Ergebnis, geht die Runde an ihn. In diesem Fall bekommt er seinen Wetteinsatz plus den gleichen Betrag zu seinem Kapital addiert. Ist sein Blatt niedriger als das der Bank, verliert er seinen Einsatz.

Während des Spiels werden die laufende Wette sowie das verbliebene Kapital auf dem Bildschirm angezeigt. Die Überschriften für diesen Kasten schreibt die Routine in Zeile 4200, die von der Initialisierungsroutine in Zeile 625 aufgerufen wird. In dieser Zeile wird auch die Variable BT auf Null gesetzt, die die Einsätze enthält.

In Zeile 4300 beginnt die Routine zur Ausgabe des Spieleinsatzes, die von verschiedenen Punkten des Programms aufgerufen wird. Vor Aufruf der Routine setzen wir die Variable SB auf die Höhe des neuen Einsatzes. Normalerweise wird SB zum gesetzten Betrag addiert und vom verbleibenden Einsatz SK subtrahiert. Da mehrere Programmteile die Routine benutzen, muß sie die Gültigkeit der neuen Wette prüfen. Verbleiben dem Spieler nach Abzug des Spieleinsatzes (angezeigt durch BG=1) weniger als £ 50, teilt ihm die Routine mit, daß sein Kapital unzureichend ist und ermöglicht ihm, das Spiel neu zu starten.

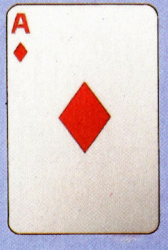
Versucht der Spieler seinen Einsatz zu verdoppeln, ohne das dafür notwendige Kapital zu haben, erscheint eine Mitteilung auf dem Bildschirm und die Wette wird nicht zugelassen. Der Test erfolgt durch Subtraktion des erforderlichen Betrages und Überprüfung des verbleibenden Kapitals in SK. Befindet sich dieser Wert im Minus-Bereich, wird die Wette zurückaddiert und die Routine verlassen.

Auch wenn der Spieler eine zweite Karte kauft und dabei mehr Geld wettet, als er besitzt, druckt die Routine eine entsprechende Meldung und reduziert die neue Wette auf die Summe des dem Spieler verbliebenen Kapitals.

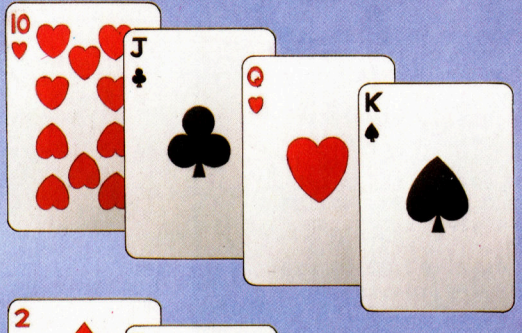
Die Routine löscht nun die zuvor gedruckten Wett- und Kapitalbeträge durch Überschreiben

Wettstrategien

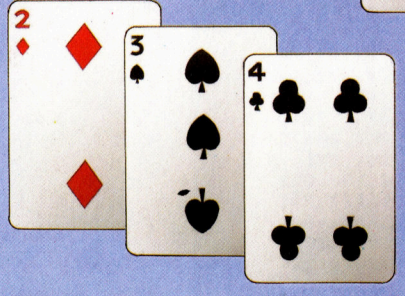
Alle erfolgreichen Wettstrategien basieren auf dem Prinzip „viel gewinnen, wenig verlieren". In unserer 17+4-Version ist der Kauf der zweiten Karte die kritische Wettphase. Wir machen hier einige Vorschläge, um Ihr Spiel zu verbessern.



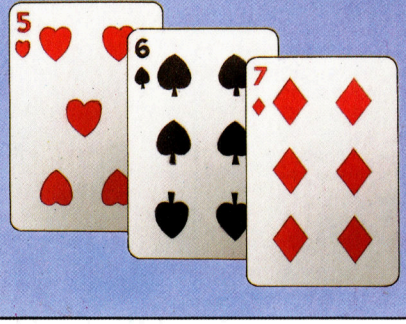
Einen sehr hohen Einsatz wert.
- Insgesamt 16 Karten im Stapel ermöglichen 17+4 oder Black-Jack.
- Asse zählen einen oder elf Punkte. Das schafft Varianten beim Aufdecken des Blattes.
- Möglicher Five-card-Trick.



Einen hohen Einsatz wert.
- Eine 2, 3 oder 4 als nächste Karte ermöglicht das Verbrennen der Karten.
- Eine hohe, zweite Karte (9, 10, Bube, Dame, König) ermöglicht einen guten Stick.
- Ein anschließendes As ergibt 17+4 oder Black-Jack.



Einen mittleren Einsatz wert.
- Niedrige Karten bieten die Chance des Five-card-Tricks.
- Ist die zweite Karte hoch, besteht die Möglichkeit, die Karten zu verbrennen.



Nur einen niedrigen Einsatz wert.
- Mittelwertige Karten sind problematisch, wenn als zweite Karte 9, 10, Bube, Dame oder König folgt.

mit Leerfeldern und zeigt anschließend die neuen Werte.

Zeile 72 setzt automatisch zu Beginn eines Spiels die Anfangswette von 50 Pfund. Die Zeilen 73 bis 80 erlauben dem Spieler eine zweite Karte zu kaufen, und überprüfen, daß die 1000-Pfund-Grenze nicht überschritten wird.

Wenn der Spieler „twist" spielt, um weitere Karten von der in Zeile 2700 beginnenden Routine zu erhalten, kann er den Einsatz mit der letzten Karte verdoppeln. Die Verdoppelungs-Routine beginnt in Zeile 2900 mit dem Aufruf der Spieleinsatz-Ausgabe-Routine. Falls sich der Spieler eine Verdoppelung nicht leisten kann (angezeigt durch CA=1), wird die Routine verlassen, und das Programm kehrt zum normalen Twist-Stick-Unterprogramm zurück. Ansonsten erhält der Spieler die letzte Karte,

Einsatzroutinen

Acorn B

```
   30 GOSUB 4000
   72 BG=1:SB=IS:GOSUB 4300:BG=0
   73GOSUB 700:PRINT"BUY A CARD (Y/N) ";
   74 AN$=GET$
   75 IF AN$<>CHR$(13) THEN PRINT AN$
   77 IF AN$<>"Y" THEN 85
   78 AN$="":GOSUB 700:INPUT "YOUR BET (
MAX £1000)"AN$
   79 IF VAL(AN$)>1000 THEN 78
   80 SB=VAL(AN$):GOSUB 4300
  210 GOSUB 700:PRINT"YOU WIN £";BT
  220 SB=-2*BT:BT=-SB:GOSUB 4300
  565 SK=10000:IS=50
  610 HP(1)=1:HP(2)=1
  625 BT=0:GOSUB 4200
 2740 IF AN$="D" THEN GOSUB 2900:IF CA=0
 THEN RETURN
 2900 REM **** DOUBLE ****
 2910 DB=1:SB=BT:GOSUB 4300
 2915 IF CA=1 THEN DB=0:RETURN
 2920 FL=0:PL=1:GOSUB 1300
 2930 GOSUB 800
 2940 DB=0:RETURN
 4000 REM
 4010 CLS
 4020 TX=13:TY=3:GOSUB 900:PRINT"BBC/ELE
CTRON"
 4030 PRINT TAB(TX+3);"PONTOON"
 4040 PRINT TAB(TX+5);"BY"
 4050 PRINT TAB(TX-7);"PETE SHAW & STEVE
 COLWILL"
 4060 TX=9:TY=8:GOSUB 900:PRINT"YOUR STA
KE IS £";SK
 4070 TX=10:TY=12:GOSUB 900:PRINT"PRESS
ANY KEY TO PLAY"
 4080 A$=GET$
 4090 RETURN
 4200 REM
 4210 COLOUR 4:TX=24:TY=18:GOSUB 900:PRI
NT"YOUR BET"
 4220 TX=24:TY=20:GOSUB 900:PRINT"REMAIN
ING STAKE"
 4230 RETURN
 4300 REM
 4302 CA=0:REM CAN'T AFFORD TO DOUBLE
 4305 IF SK>=50 OR BG=0 THEN 4310
 4306 AN$="":GOSUB 700:INPUT"OUT OF CASH
! PLAY AGAIN(Y/N) ";AN$
 4307 IF AN$="Y" THEN RUN
 4308 END
 4310 SK=SK-SB:BT=BT+SB
 4315 IF DB=1 AND SK<0 THEN GOSUB 700:PR
INT"YOU CAN'T AFFORD IT!"
 4317 IF DB=1 AND SK<0 THEN BT=BT-SB:CA=
1:RETURN
 4320 IF SK<0 THEN BT=SK+SB:SK=0:GOSUB 7
00:PRINT"YOU CAN ONLY AFFORD £";BT
 4340COLOUR 1:TX=24:TY=19:GOSUB 900:PRIN
T LEFT$(SP$,15)
 4345 TX=24:TY=19:GOSUB 900:PRINT "£";BT

 4350 TX=24:TY=21:GOSUB 900:PRINT LEFT$(
SP$,15)
 4355 TX=24:TY=21:GOSUB 900:PRINT "£";SK

 4360 RETURN
```

Schneider CPC

```
30 GOSUB 4000:REM title screen
72 bg=1:sb=is:GOSUB 4300:bg=0:REM print
bet
73 GOSUB 700:PRINT "buy a card (y/n)";
74 an$="":WHILE an$="": an$=INKEY$:WEND
75 IF an$<>CHR$(13) THEN PRINT an$
77 IF an$<>"y" THEN 85
78 an$="":GOSUB 700:INPUT"your bet (max
£1000)";an$
79 IF VAL(an$)>1000 THEN 78
80 sb=VAL(an$):GOSUB 4300:REM print bet
210 GOSUB 700:PEN black:PRINT"you win £"
;:PEN white:PRINT bt
220 sb=-2*bt:bt=-sb:GOSUB 4300:REM print
 bet
565 sk=10000:is=50:REM stakes
610 hp(1)=1:hp(2)=1
625 bt=0:sb=0:GOSUB 4200:REM print bet b
ox
2740 IF an$="d" THEN GOSUB 2900:IF ca=0
THEN RETURN
2900 REM **** double ****
2910 db=1:sb=bt:GOSUB 4300:REM print bet
2915 IF ca=1 THEN db=0:RETURN
2920 fl=0:pl=1:GOSUB 1300:REM deal
2930 GOSUB 800:REM evaluate
2940 db=0:RETURN
4000 REM **** title etc ****
4010 CLS
4020 tx=11:ty=3:GOSUB 900:PEN red:PRINT"
Amstrad CPC Range"
4030 PRINT TAB(tx+6)"Pontoon"
4040 PRINT TAB(tx+8)"By"
4050 PRINT TAB(tx+3)"Steve Colwill"
4060 tx=9:ty=8:GOSUB 900:PEN black:PRINT
"Your stake is £";sk
4070 tx=10:ty=12:GOSUB 900:PEN white:PRI
NT"Press a key to play"
4080 WHILE INKEY$="":WEND
4090 RETURN
4200 REM **** betting box ****
4210 tx=24:ty=18:GOSUB 900:PEN red:PRINT
"Your bet"
4220 tx=24:ty=20:GOSUB 900:PRINT"Remaini
ng stake"
4230 RETURN
4300 REM **** print stake ****
4302 ca=0:REM can't afford to double fla
g
4305 IF sk>=50 OR bg=0 THEN 4310
4306 an$="":GOSUB 700:PEN red:INPUT"Out
of cash! Play again (y/n)";an$
4307 IF an$=y" then run
4308 END
4310 sk=sk-sb:bt=bt+sb
4315 IF db=1 AND sk<0 THEN GOSUB 700:PEN
 red:PRINT"You can't afford it!":bt=bt-s
b:sk=sk+sb:ca=1:RETURN
4320 IF sk<0 THEN bt=sk+sb:sk=0:GOSUB 70
0:PEN red:PRINT"You can only afford £";b
t
4340 tx=24:ty=19:GOSUB 900:PRINT SPACE$(
15)
4345 tx=24:ty=19:GOSUB 900:PEN white:PRI
NT"£";bt
4350 tx=24:ty=21:GOSUB 900:PRINT SPACE$(
15)
4355 tx=24:ty=21:GOSUB 900:PRINT"£";sk
4360 RETURN
```

das Blatt wird ausgewertet und die Routine beendet.

Die Routinen entwickelten wir in der vorangegangenen Fortsetzung des Projekts. Es fehlt nur noch die Zeile, die, falls der Spieler gewinnt, die gewonnene Summe zum verbleibenden Einsatz addiert. Dazu fügen wir Zeile 220 in die Hauptprogrammschleife ein.

Die Programmlistings sind nun komplett. Zu Beginn des Spiels mischt das Programm automatisch den Kartenstapel und durch Drücken von SHIFT-S (beim Sinclair SYMBOL SHIFT-S).

Sinclair Spectrum

```
  30 GO SUB 4000: REM TITLE SCREEN
  72 LET BG=1: LET SB=IS: GO SUB 4300: L
ET BG=0: REM PRINT BET
  73 LET A$="": GO SUB 700: PRINT  INK 2
;"BUY A CARD (Y/N) ";
  74 LET A$=INKEY$: IF A$="" THEN  GO TO
 74
  75 IF A$<>CHR$ 13 THEN  PRINT A$
  77 IF A$<>"Y" THEN  GO TO 85
  78 LET A$="": GO SUB 700: INPUT "YOUR
BET (MAX £1000)"; LINE A$
  79 IF VAL A$>1000 THEN  GO TO 78
  80 LET SB=VAL A$: GO SUB 4300: REM PRI
NT BET
 210 GO SUB 700: PRINT "YOU WIN £";BT
 220 LET SB=-2*BT: LET BT=-SB: GO SUB 43
00: REM PRINT BET
 565 LET SK=10000: LET IS=50: REM STAKES
 610 LET P(1)=1: LET P(2)=1
 625 LET BT=0: GO SUB 4200: REM PRINT BE
T BOX
2740 IF A$="D" THEN  GO SUB 2900: IF CA=
0 THEN  RETURN : REM DOUBLE
2900 REM **** DOUBLE ****
2910 LET DB=1: LET SB=BT: GO SUB 4300: R
EM PRINT BET
2915 IF CA=1 THEN  LET DB=0: RETURN
2920 LET FL=0: LET PL=1: GO SUB 1300: RE
M DEAL
2930 GO SUB 800: REM EVALUATE
2940 LET DB=0: RETURN
4000 REM **** TITLE ETC ****
4010 CLS
4020 LET TX=10: LET TY=3: GO SUB 900: PR
INT " ZX SPECTRUM"
4030 PRINT TAB TX+3;"PONTOON"
4040 PRINT TAB (TX+5);"BY"
4050 PRINT TAB TX-7;"PETE SHAW & STEVE C
OLWILL"
4060 LET TX=5: LET TY=8: GO SUB 900: PRI
NT "YOUR STAKE IS £";SK
4070 LET TX=5: LET TY=12: GO SUB 900: PR
INT  FLASH 1;"PRESS ANY KEY TO PLAY"
4080 LET A$=INKEY$: IF A$="" THEN  GO TO
 4080
4090 RETURN
4200 REM **** PRINT STAKE ****
4210 PRINT AT 18,15;"YOUR BET"
4220 PRINT AT 20,15;"REMAINING STAKE"
4230 RETURN
4300 REM **** PRINT STAKE ****
4302 LET CA=0: REM CAN'T AFFORD TO DOUBL
E FLAG
4305 IF SK>=50 OR BG=0 THEN  GO TO 4310
4306 LET A$="": GO SUB 700: INPUT "OUT O
F CASH!  PLAY AGAIN (Y/N)";A$
4307 IF A$="Y" THEN  RUN
4308 STOP
4310 LET SK=SK-SB: LET BT=BT+SB
4315 IF DB=1 AND SK<0 THEN  GO SUB 700:
PRINT  FLASH 1;"YOU CAN'T AFFORD IT!"
4317 IF DB=1 AND SK<0 THEN  LET BT=BT-SB
: LET SK=SK+SB: LET CA=1: RETURN
4320 IF SK<0 THEN  LET BT=SK+SB: LET SK=
0: GO SUB 700: PRINT  FLASH 1;"YOU CAN O
NLY AFFORD £";BT
4340 LET TX=24: LET TY=18: GO SUB 900: P
RINT S$( TO 15)
4345 LET TX=24: LET TY=18: GO SUB 900: P
RINT "£";BT
4350 LET TX=24: LET TY=20: GO SUB 900: P
RINT S$( TO 15)
4355 LET TX=24: LET TY=20: GO SUB 900: P
RINT "£";SK
4360 RETURN
```

Commodore 64

```
 72 BG=1:SB=IS:GOSUB4300:BG=0:REM PRINT B
ET
 73 AN$="":GOSUB700:PRINT"BUY A CARD (Y/N
) ";
 74 GET AN$:IF AN$="" THEN 74
 75 IF AN$<>CHR$(13)THEN PRINT AN$
 77 IF AN$<>"Y" THEN 85
 78 AN$="":GOSUB700:INPUT"YOUR BET (MAX £
1000)";AN$
 79 IF VAL(AN$)>1000 THEN 78
 80 SB=VAL(AN$):GOSUB4300:REM PRINT BET
210 GOSUB700:PRINT CHR$(156);"YOU WIN £"
;CHR$(5);BT
220 SB=-2*BT:BT=-SB:GOSUB4300:REM PRINT
BET
565 SK=10000 :IS=50:REM STAKES
610 HP(1)=1:HP(2)=1
625 BT=0:GOSUB4200:REM PRINT BET BOX
2740 IF AN$="D" THEN GOSUB 2900:IF CA=0
THEN RETURN
2900 REM **** DOUBLE ****
2910 DB=1:SB=BT:GOSUB4300:REM PRINT BET
2915 IF CA=1 THEN DB=0:RETURN
2920 FL=0:PL=1:GOSUB1300:REM DEAL
2930 GOSUB800:REM EVALUATE
2940 DB=0:RETURN
4000 REM **** TITLE ETC ****
4010 PRINT CHR$(147):REM CLEAR SCREEN
4020 TX=13:TY=3:GOSUB900:PRINTCHR$(156);
"COMMODORE 64"
4030 PRINTTAB(TX+3);"PONTOON"
4040 PRINTTAB(TX+5);"BY"
4050 PRINTTAB(TX);"STEVE COLWILL"
4060 TX=9:TY=8:GOSUB900:PRINTCHR$(28);"Y
OUR STAKE IS £";SK
4070 TX=10:TY=12:GOSUB900:PRINTCHR$(5);"
PRESS A KEY TO PLAY"
4080 GET A$:IF A$=""THEN 4080
4090 RETURN
4200 REM **** BETTING BOX ****
4210 TX=24:TY=18:GOSUB900:PRINTCHR$(156)
;"YOUR BET"
4220 TX=24:TY=20:GOSUB900:PRINT"REMAININ
G STAKE"
4230 RETURN
4300 REM **** PRINT STAKE ****
4302 CA=0:REM CAN'T AFFORD TO DOUBLE FLA
G
4305 IF SK>=50 OR BG=0 THEN 4310
4306 AN$="":GOSUB700:PRINTCHR$(28);:INPU
T"OUT OF CASH! PLAY AGAIN (Y/N)";AN$
4307 IF AN$="Y" THEN RUN
4308 END
4310 SK=SK-SB:BT=BT+SB
4315 IFDB=1ANDSK<0THEN GOSUB700:PRINTCHR
$(28);"YOU CAN'T AFFORD IT!"
4317 IFDB=1ANDSK<0THENBT=BT-SB:SK=SK+SB:
CA=1:RETURN
4320 IF SK<0THENBT=SK+SB:SK=0:GOSUB700:P
RINTCHR$(28);"YOU CAN ONLY AFFORD £";BT
4340 TX=24:TY=19:GOSUB900:PRINT LEFT$(SP
$,15)
4345 TX=24:TY=19:GOSUB900:PRINTCHR$(5);"
£";BT
4350 TX=24:TY=21:GOSUB900:PRINT LEFT$(SP
$,15)
4355 TX=24:TY=21:GOSUB900:PRINTCHR$(5);"
£";SK
4360 RETURN
```

Korrekt übersetzt

In unserer letzten Folge über die Programmierung des 68000 sehen wir uns die Assemblervorgänge genauer an. Wir untersuchen, welche Rolle der Assembler in der Programmierung spielt.

In den letzten Folgen hatten wir schon mehrfach Assembleranweisungen erwähnt, um die Fähigkeiten des 68000 zu verdeutlichen. Dabei gingen wir davon aus, daß der Computer die Befehle so ausführt, wie sie im Assembler erscheinen. Natürlich kann die Maschine nur den binären Code verstehen, den die Assemblerbefehle auf höherer Ebene darstellen, wobei der Assembler diese Anweisungen in den Maschinencode übersetzt.

Für das Verständnis von Programmen ist es praktisch, sich die Ausführung auf der Ebene des Quellentextes vorzustellen. Diese Ebene kann weit oben liegen – zum Beispiel bei einem Anwendungspaket wie WordStar –, etwas niedriger – bei PASCAL-Programmen – oder noch maschinennaher auf der Ebene der Assemblerprogramme. Somit entsteht der Eindruck von zahlreichen „virtuellen" Maschinen auf unterschiedlichen Ebenen.

Programmausführungen

Jede Programmausführung – egal auf welcher Ebene – geschieht jedoch durch

- Übersetzung: Ein Programm auf einer Ebene wird in eine niedrigere Ebene übersetzt. So können Sie ein PASCAL-Programm auf Ebene 3 in den Assemblercode übersetzen, der auf Ebene 2 mit einem einfachen Compiler abläuft.
- Interpretation: Die Programme einer Ebene werden von einem weiteren Programm (‚Interpreter' genannt), das auf einer niedrigeren Ebene läuft, ‚gedeutet'. So interpretiert die Maschine beispielsweise die binären Bitmuster der Maschinenbefehle auf Registerebene und führt sie aus. Ebenso interpretiert ein Interpreter der Ebene 2 ein PASCAL-Programm der Ebene 3.

Uns interessiert besonders der Übersetzungsvorgang. Es gibt viele Übersetzungsarten – von Compilern bis zu Assemblern. Alle haben jedoch eins gemeinsam: Sie nehmen auf höheren Ebenen geschriebene Befehle entgegen und erzeugen auf einer niedrigeren Ebene den entsprechenden Code. Die ursprünglichen Quellanweisungen werden danach nicht mehr gebraucht und daher vom Übersetzer „weggeworfen" (Interpreter brauchen natürlich immer noch den ursprünglichen Programmtext).

Ziel eines Assemblers ist es, die Assemblerbefehle einer hohen Ebene in den Binärcode zu übersetzen, den die Maschine ausführen kann. Hier ein Beispiel:

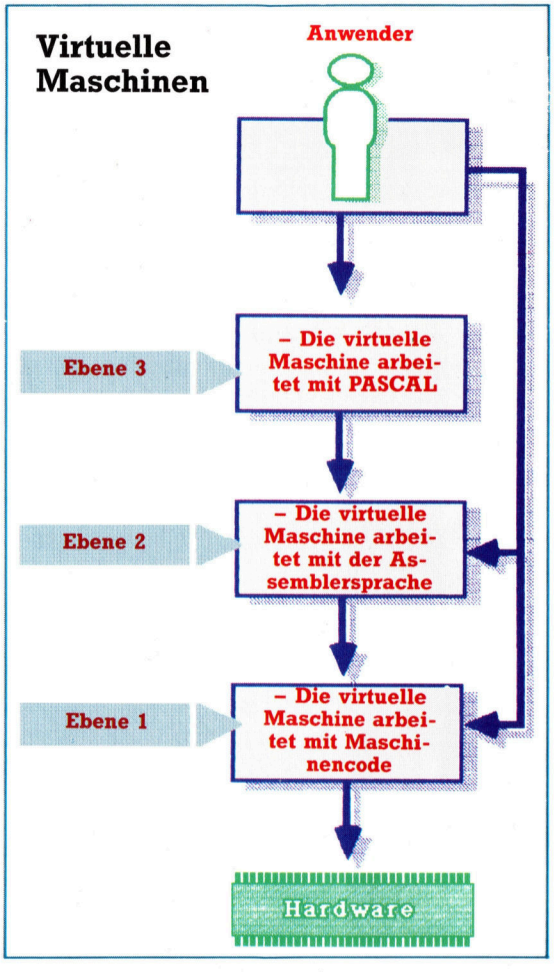
MOVE.W D3,D5
wird übersetzt in:
0011 101 000 000 011

0011 entspricht „Wort übertragen"; 101 000 entspricht „nach D5" und 000 011 „von D3".

Die Bitcodierung ist jedoch viel zu umständlich und fehleranfällig. Wir brauchen zumindest mnemotische Kürzel, um uns Befehle und Datenobjekte merken zu können. MOVE.W TOTAL,D4 sind symbolische Namen, die etwas über den Inhalt einer Speicherstelle aussagen (in diesem Beispiel TOTAL).

Bei der bitweisen Codierung entstehen auch leicht folgende Fehler:

- Befehle (Opcodes) falsch eingegeben
- Falsche absolute Adressierung



Ein Computer kann drei „virtuelle" Maschinen enthalten. Hier läuft auf Ebene 3 die Programmiersprache PASCAL, auf Ebene 2 der Assembler und auf Ebene 1 der Maschinencode, der die Hardware direkt steuert.

● Zuteilung einer falschen Bytezahl durch einen Befehl.

Da die manuelle Übersetzung von mehr als einer halben Befehlsseite kaum mehr zu bewältigen ist, bietet der Assembler mit seiner Übersetzung des Quellentextes in den Objectcode eine unschätzbare Hilfe.

Der Assemblervorgang liest zuerst die Befehle der Quelltextdatei (als ASCII-Text geschrieben) und legt eine Objektdatei an (oder liest sie ein). Anhand eines kurzen Assemblerprogramms (das eine arithmetische Berechnung mit den Elementen eines Arrays durchführt) wollen wir die einzelnen Übersetzungsschritte verfolgen.

Fehlermeldungen

In der ersten Spalte des Programmlistings stehen die Speicherstellen (LOC – siehe Überschriftzeile), gefolgt vom Inhalt der Speicherstellen. Der Rest der Zeile enthält den Quellentext mit den vom Assembler eingesetzten Befehlsnummern. Alle Fehlermeldungen beziehen sich auf diese Nummern. Wenn etwa Zeile 14 einen Fehler enthält, dann erscheint hier ein E, in der Fehlerliste eine Meldung.

Aufbau eines Assemblerlistings

```
LOC       OBJECT                    STMT          SOURCE STATEMENT
                                    1   *This program takes each element of an array and converts
                                    2   *each element to
                                    3   *                INPUT[I]:=2*INPUT[I]+3
                                    4
                                    5
001000                              6         ORG $1000
          =000C                     7   length:   equ 12
                                    8
001000    700C                      9   start:    moveq    #length,d0
001002    41F8  1024                10            lea      input,a0
001006    4244                      11            clr      d4
001008    4243                      12            clr      d3
                                    13
00100A    3618                      14  loop:     move.w   (a0),d3
00100C    C7FC  0002                15            muls     #2,d3
001010    D67C  0003                16            add.w    #3,d3
001014    30C3                      17            move     d3,(a0)+
001016    D843                      18            add      d3,d4
                                    19
001018    5340                      20            subq     #1,d0
00101A    6600  FFEE                21            bne      loop
00101E    31C4  103C                22            move     d4,sum
001022    4F40                      23            trap     #0
                                    24
001024    0001  0002  0003  0004    25  input:    dc.w     1,2,3,4,5
          0005
00102F    0006  0007  0008  0009    26            dc.w     6,7,8,9,10
          000A
001038    000B 000C                 27            dc.w     11,12
                                    28
00103C    0000                      29  sum:      dc.w     0
                                    30
                                    31            end

No errors found in this assembly

          SYMBOL                    DEFN          VALUE     REFERENCES
INPUT                               25            1024      10
LENGTH                              7             000C      9
LOOP                                14            100A      21
START                               9U            1000
SUM                                 29            103C      22
```

Auch die Binärdatei ist interessant. Sie enthält außer codierten Binärdaten auch Ladeinformationen, um die Daten überhaupt in den Speicher laden zu können. Da Adreß- und Inhaltsinformation in hexadezimal codierten Binärzahlen gespeichert werden, muß das Lademodul auch das Binärformat verarbeiten können. Zwar bestimmt jeder Assembler das Format selbst, doch ist die entsprechende Kompatibilität des Lademoduls eine der Grundvoraussetzungen für das Funktionieren.

Der letzte Teil des Listings ist die Symboltabelle mit den numerischen Werten, die den (im Programm deklarierten) Labels zugeordnet wurden. So hat INPUT (in Anweisung 25 definiert) beispielsweise den Wert 1024 (hex) und wird in Anweisung 10 angesprochen. In unserem kurzen Beispielprogramm lassen sich diese Daten zwar noch leicht dem Listing entnehmen, bei umfangreichen Programmen sind sie jedoch für die Fehlersuche unentbehrlich.

Die Hardware bestimmt, wohin der übersetzte Text ausgegeben wird. Wenn Sie beispielsweise auf einem Unix-System mit einem „Crossassembler" arbeiten, erscheinen Listing und Binärinformationen in Dateien. Vor Aufruf des Programms müssen die Binärdateien erst in das System des angesprochenen 68000-Prozessors geladen werden.

Ein Assembler kann natürlich auch gleich auf dem gewünschten Prozessor laufen, wenn dort ein Dateisystem vorhanden ist. Dabei werden die Binärcodes direkt in den Speicher geladen und dort assembliert, während ein Drukker das Listing direkt ausgibt.

Bei Einsatz des Assemblers können aber auch Schwierigkeiten auftreten. Normalerweise läuft der Assembliervorgang in zwei ‚Durchgängen' ab. Der erste Durchgang

● decodiert die mnemotischen Kürzel (MOVE, ADD, MULS etc.).

● zählt die Adreßbytes (und bereitet damit die Adreßvergabe vor).

● legt eine Symboltabelle an (die Auskunft über die Symbolwerte gibt).

● meldet Syntaxfehler (ADDQQ ist beispielsweise ein illegaler Befehl).

„Schnellübertragung"

Wenn der Assembler beim ersten Durchgang durch den Quellentext die Anweisung 9 liest, enthält das Label den Wert 41F8. Dies entspricht einer ‚Schnellübertragung' der Konstanten 12 (das Symbol LENGTH) auf DO im unmittelbaren Modus. Die Anweisung 10 (LEA) setzt die Speicherstelle 1002 auf 41F8, kann aber die Adresse von INPUT noch nicht festlegen. Die Speicherstelle 1004 wird daher freigelassen, bis nach dem zweiten Durchgang die Adresse von INPUT bekannt ist.

Im zweiten Durchgang setzt der Assembler alle vom Programm benötigten Adressen ein und gibt den binären Code und das Pro-

grammlisting aus. Fehler kommen dabei nur selten zum Vorschein, da sie zumeist im ersten Durchgang schon abgefangen wurden.

Der Assembler kann auch leicht Berechnungen durchführen und mit symbolischen Ausdrücken den Wert eines Operanden herausfinden. Hier als Beispiel die Einführungszeilen eines Listings:

```
LENGTH DC.W     (START-ARRAY)*4
ARRAY  DC.W     1,2,3,4,5
START  MOVE.W LENGTH,D3
```

Dabei wird die Länge des Arrays in Bytes berechnet (START-ARRAY mal vier) und im Programm eingesetzt (das Datenregister 3 wird gesetzt). Wenn Sie später im Programm die Länge des Arrays ändern wollen, wird LENGTH automatisch neu berechnet.

Auch bei der Programmdokumentation bietet der Assembler praktische Unterstützung. Sie können Kommentare in die Programme einfügen, die die Bedeutung der Befehle erläutern und Auskunft über Registervergabe und Programmaufbau geben. Hier ein Beispiel:

```
* Das Programm nimmt einen Eingabestring ent-
* gegen und prüft ihn.
* Texteingabe und Prüfung per ACIA
* B. Schreiber — Dezember 85
*
* Registervergabe
*
* Adreßregister
*    A1 Pointer auf Eingabestring
*    A2 Pointer auf gespeichertes Schlüsselwort
*    A3 Pointer auf aktuelles Zeichen
* Datenregister
*    D0 Eingabezeichen
*    D1 Ausgabezeichen
*
START JSR INIT          E/A initialisieren
      JSR READSTRING    Eingabestring lesen
      JSR VALIDATE      und prüfen
      JSR SIGNAL        korrekte Eingabe
                        anzeigen
      BRA START         zurück zum Anfang
```

Sie sehen, die Kommentare informieren ausführlich über Programmstruktur und die eingesetzten Register. Auf dieser Ebene wären weitere Kommentare vermutlich überflüssig, doch könnten Sie auf der Subroutinenebene vermutlich weitere Informationen erfahren. Wichtig ist, daß die entsprechenden Programme ausführlich kommentiert sind.

Der Assembler bietet aber noch weitere Hilfen. „Assembleranweisungen" wie TTL veranlassen, daß über jeder Listingseite eine Programmüberschrift gedruckt wird. Andere Anweisungen versorgen den Assembler mit Informationen. So können Sie mit der Anweisung END das Ende des Eingabetextes anzeigen oder mit ORG veranlassen, daß die Assemblierung bei einer bestimmten Adresse anfängt. Ein Programmodul enthält fast immer folgende Zeilen:

```
*    Dokumentationskommentare mit
*    Namen und Datum
     TTL Dies ist mein Programm
     ORG $1000    * Anfangsadresse
START  MOVE D1,D2  * einige Befehle
                   * weitere Befehle
END  * Endanweisung
```

Tabelle der Assembleranweisungen

ORG	Kurzform für ‚Origin'. Legt die Anfangsadresse des Codes fest
RORG	Wie ORG, jedoch für PC relativen Code
TTL	Druckt einen Titel auf jede Seite des Assemblerlistings
END	Kein weiterer Quellentext
EQU	Ordnet dem angegebenen Symbol einen Wert zu (z. B. LENGTH EQU 10+50*ARRAY1. Wenn ARRAY1 12 ist, dann wird LENGTH zu 610)
DC	‚Deklariere Konstante' — weist einen konstanten Wert (Byte, Wort oder Langwort) zu. Z. B. PARAM DC.W 12,23 setzt PARAM als Wort mit Wert 12; PARAM+2 als Wort mit Inhalt 23
DS	Reserviert einen Speicherbereich von nicht initialisierten Daten (z. B. STACK DS.B 100 reserviert einen Adreßbereich von 100 Bytes namens STACK.)
LLEN	Setzt die Zeilenlänge
PAGE	Gibt während des Listings einen Seitenvorschub aus

Gemeinsamkeiten

Die Assembler verschiedener Hersteller unterscheiden sich natürlich in Format und Abläufen voneinander. Dennoch folgen sie meistens dem folgenden Quelltextformat: Kommentare werden durch einen * am Zeilenanfang oder durch mindestens ein Leerzeichen hinter einem Befehl eingeleitet. Folgende Kommentare sind legal:

```
* Dies ist ein Kommentar
START ADD D1,D2 * und so weiter
```

Auch das Befehlsformat ist definiert. Es kann drei Felder enthalten:

- Label: Jeder als Label eingesetzte Name muß mit einem Buchstaben anfangen und insgesamt weniger als 30 alphanumerische Zeichen lang sein. Wenn das Labelfeld leer ist, muß an seiner Stelle mindestens ein Leerzeichen stehen.
- Opcode: Hier stehen die Befehlscodes des 68000.
- Operand: Zwischen Befehlscode und Operanden muß mindestens ein Leerzeichen stehen. Zwei Operanden dürfen nicht durch Leerzeichen voneinander getrennt sein.

Hier einige illegale Anweisungsbeispiele:

```
Dies ist kein Kommentar, da der * fehlt
MOVE  D1, D2     * Leerzeichen zwischen
                 * zwei Operanden
```

```
MOVE   8TOIT,D5  * Namen müssen mit einem
                 * Buchstaben anfangen und
                 * hinter den Operanden
                 * muß mindestens ein
                 * Leerzeichen stehen
RTS              * ist in Ordnung
RTS    D1        * D1 ist hier nicht möglich
* Ende der Fehlerbeispiele
```

Zum Schluß noch ein Wort über die alphanumerische Darstellung. Zahlen werden dem Assembler als Dezimalwerte übergeben, wenn davor kein $-Zeichen steht. So wird 1234 dezimal interpretiert und $1234 als Hexzahl. ASCII-Buchstaben müssen in Anführungsstriche eingeschlossen sein:

```
'Hier ist das Ende des Kurses!'
```

Sie sehen, der Assembler kann nicht nur Code zum 68000 schicken, sondern bietet mit dem Aufbau von Macros, den Assembleranweisungen und Fehlermeldungen auch viele Programmierhilfen. Auch den möglichen Einsatz von Kommentaren sollten Sie nicht unterschätzen, da Sie sich nach mehreren Monaten mit Sicherheit nicht mehr an alle Einzelheiten eines Programms erinnern können.

Der QL arbeitet mit einem 68008 Microprozessor, der dem 68000 entspricht, aber nur einen Acht-Bit-Datenbus besitzt. Da die beiden Chips den gleichen Befehlssatz einsetzen, können Sie die Befehle dieser Serie auch mit dem 68008 verwenden.

Flexibilität mit Macros

Für die Entwicklung umfangreicher Programme können im Quellprogramm, statt umfangreicher Codezeilen, „Macros" angegeben werden. Wenn die Assemblierung auf ein Macro trifft, wird der zuvor definierte Macrotext ausgegeben. Das Macro „ERROR" kann beispielsweise folgendermaßen definiert sein:

```
ERROR   MACRO
        MOVE.B    #3, ERRORLOC
        JSR       SIGNAL
        ENDMACRO
```

Zum Zeitpunkt der Assemblierung erzeugt das Wort ERROR nun folgende Befehle:

```
MOVE.B    #3,ERROR.LOC
JSR       SIGNAL
```

Sie können Macros – falls nötig – auch mit Parametern einsetzen. Hier ein Beispiel:

```
                TTL Macro Example
                ORG       $1000

ADDON   MACRO
        ADD       \1,D2
        ADD       D2,\2
        ENDM

        ADDON     AVAL,SUM
        ADDON     BVAL,SUM
        ADDON     SUM,TOTAL

AVAL    DC.W      0
BVAL    DC.W      0
SUM     DC.W      0
TOTAL   DC.W      0

        END
```

In diesem Beispiel beziehen sich \1 und \2 auf den ersten und zweiten Parameter. Bei der Assemblierung entsteht daraus folgender Code:

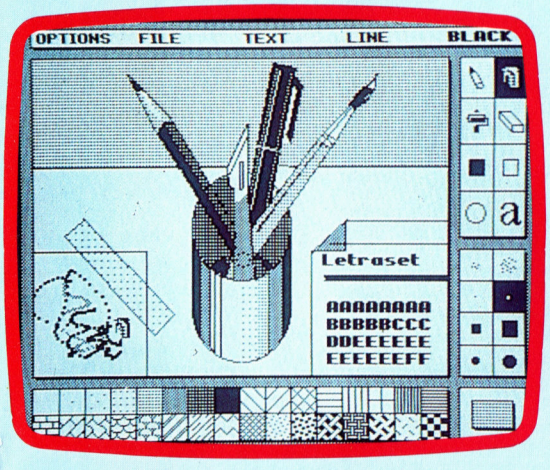
```
3                 ORG       $1000
4
5       ADDON     MACRO
6                 ADD       \1,D2
7                 ADD       D2,\2
8                 ENDM
9
10                ADDON     AVAL.SUM
11+               ADD       AVAL,D2
12+               ADD       D2,SUM
13                ADDON     BVAL,SUM
14+               ADD       BVAL,D2
15+               ADD       D2,SUM
16                ADDON     SUM,TOTAL
17+               ADD       SUM,D2
18+               ADD       D2,TOTAL
19
20      AVAL      DC.W      0
21      BVAL      DC.W      0
22      SUM       DC.W      0
23      TOTAL     DC.W      0
24                END
```

Das + hinter den Zeilennummern gibt die Zeilen an, die vom Macro-Prozessor eingesetzt wurden.

Fachwörter von A bis Z

WIMP = WIMP
WIMP bedeutet „Windows Icons Mouse Program", zu deutsch „maus- und Piktogramm-gesteuertes Fenstermenü-Programm". Die Bedienung erfolgt dabei nicht durch Eintippen von Kommandos, sondern über die Maus, indem ein Cursor-Symbol auf „Piktogramme" – die sinnbildlichen Darstellungen der Operationen – gesetzt wird.

Die WIMP-Technik besticht durch extreme Benutzerfreundlichkeit. Das Konzept geht auf Forschungsarbeiten der Firma Xerox zurück, die gegen Ende der siebziger Jahre zur Entwicklung der SMALL-TALK-Benutzerumgebung für das objektorientierte Programmieren führten. Den ersten kommerziellen WIMP-Systemen für Microcomputer stand die Fachwelt zuerst sehr skeptisch gegenüber, aber inzwischen ist bei der Mehrzahl der neuen Rechner diese Technik entweder bereits integriert oder nachträglich zu implementieren.



Die WIMP-Technik erregte mit der Vorstellung von Apples LISA erstmals Aufsehen und wurde dann durch den Macintosh allgemein bekannt.

Word = Wort
Ein „Wort" wird durch eine Gruppe zusammengehöriger Bits gebildet, die als Einheit adressierbar sind. Die Anzahl der Bits oder die „Wortlänge" ist generell durch die Verarbeitungsbreite der bei einem Rechner verwendeten CPU bestimmt; bei Micros beträgt sie normalerweise acht, 16 oder 32 Bits (d. h. ein, zwei oder vier Bytes).

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.

Word Processing = Textverarbeitung
Textverarbeitungssysteme erleichtern das Entwerfen von Schriftstükken und ihre Bearbeitung bis hin zum druckfertigen Formatieren des Textes. Daher sind neben den Editierfunktionen wie Einfügen, Löschen oder Umstellen von Textpassagen meist auch der automatische Zeilenumbruch und das Ausrichten der Zeilenränder vorgesehen.

Im privaten wie im kommerziellen Bereich ist die Textverarbeitung ein bevorzugtes Einsatzfeld für Microcomputer. Es gibt eine Riesenauswahl derartiger Systeme: von kompletten Spezialanlagen für einige zehntausend Mark über anspruchsvolle Softwarepakete für ein- bis zweitausend bis zum einfachen Texteditor für zwanzig Mark.

Workspace = Arbeitsbereich
Als „Workspace" reservieren sich viele Programmsysteme einen bestimmten Speicherbereich, in dem sie dann während des Ablaufs kurzfristig Daten sowie Variablenwerte ablegen.

Xerographic Printer = Xerografischer Drucker
Dieser Drucker hat eine mit einer Halbleiterschicht überzogene Platte, die elektrostatisch aufgeladen wird. Bei Belichtung ändert sich die Leitfähigkeit entsprechend der Bildstruktur, so daß sie mit Hilfe eines Pulvers auf Papier gebracht werden kann.

Zero Flag = Zero Flag
Das Statusregister der CPU enthält eine Reihe von Markierungs-Flipflops oder „Flags", die als Einzelbits bestimmte Zustände anzeigen. Das „Zero-Flag" wird immer dann gesetzt, wenn eine Akkumulatoroperation zum Ergebnis Null geführt hat.

Das Zero-Flag spielt in der Maschinenprogrammierung eine wichtige Rolle beim Vergleichen von Größen oder beim Überwachen von abwärtslaufenden Schleifenzählern. Die CPU fragt dann in Verbindung mit einem bedingten Sprungbefehl das Zero-Flag ab und verzweigt, wenn es gesetzt ist, zu einem anderen Programmteil.

Zero Page = Nullseite
Zur Angabe absoluter Adressen sind bei einem 8-Bit-Micro zwei Bytes erforderlich, nämlich ein höher- und ein niederwertiges. Das höherwertige Byte enthält die Nummer der „Seite" (Page), auf die sich die Adresse bezieht, das niederwertige legt die Position innerhalb der Seite fest. Jede Seite umfaßt 256 aufeinanderfolgende Speicherworte. Die erste Seite des Speichers wird als „Zero Page" bezeichnet, weil das höherwertige Adreßbyte dort Null ist. Zum Ansprechen der Nullseite genügt also eine Ein-Byte-Adresse – Befehle mit Nullseitenadressierung benötigen weniger Zugriffzeit als andere. Bei vielen Rechnern benutzt daher das Betriebssystem die Nullseite als Arbeitsbereich zur Ablage wichtiger Variablen und Adressen.

Als „Zero Page" gelten zwar eigentlich nur die ersten 256 Speicherbytes, aber einige Prozessoren kennen auch ein „Direct Page Register" (Seitenregister für die Direktadressierung) zum Eintragen einer beliebigen Seitennummer, auf die dann schneller zugegriffen werden kann.

Bildnachweise
2297: Tony Sleep
2298, 2310, 2318: Kevin Jones
2299, U3: Ian McKinnel
2303: Dimension Graphics
2304: Paul Bryant
2309: Liz Heany
2312, 2316, 2321: Caroline Clayton

Das letzte Heft +++ Das letzte Heft +++ Das letzte Heft +++ Das letzte Heft

computer kurs

Heft 84

INDEX