Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

# computer kurs

Heft 79

Ein wöchentliches Sammelwerk

Spielregeln 17 + 4

Neue Wege zum EDV-Job

Spaß mit MUD

Files komprimieren

# computer kurs Heft 79

## 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.

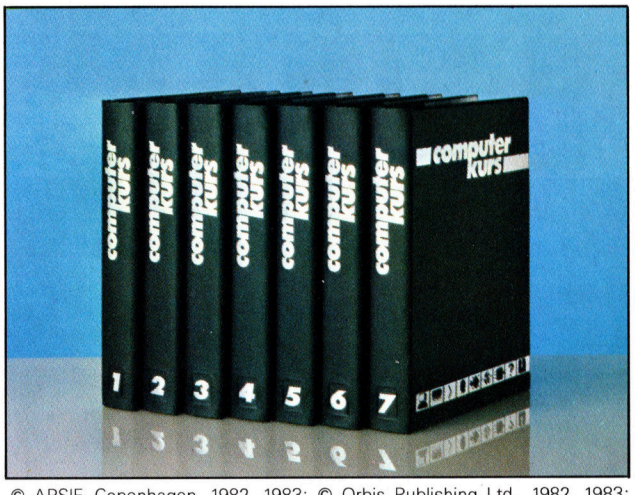
**Schweiz:** Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

## 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

# Spieler-träume

**Mit Spielprogrammen Karriere zu machen, ist der Traum vieler Computer-Freunde. Eine realistische Einschätzung des Marktes und der eigenen Programmierfähigkeiten ist dabei sehr wichtig.**

Im Virgin's Games Center in der Londoner Oxford Street kann man sehen, wohin der Markt der Computerspiele treibt – das riesige Angebot läßt sich nur noch mit einem Supermarkt vergleichen.

Als freier Mitarbeiter zu programmieren ist ein risikoreiches Geschäft. Die Bezahlung ist meist niedrig – es sei denn, ein Spiel wird zum Riesenerfolg. Top-Programmierer können mit Vorschuß und Gewinnbeteiligungen rechnen – für Anfänger sind solche Bedingungen kaum zu erwarten. Außerdem dauert es oft recht lange, bis überhaupt etwas gezahlt wird. Es vergehen Monate, bevor ein Spiel in den Verkauf geht, und die Gewinnbeteiligung wird meist halbjährlich ausgeworfen. Vom fertigen Spiel bis zum ersten Scheck dauert es also mindestens ein Jahr. Niemand kann garantieren, daß ein Spiel überhaupt angenommen wird – selbst wenn ein Vertrag vorliegt. Jeder Vertrag enthält Klauseln, die besagen, daß das Spiel den Anforderungen des Software-Hauses entsprechen muß. Spiele können aber sehr schnell aus der Mode kommen – das vielversprechende Konzept von gestern kann nach der Fertigstellung bereits überholt sein.

## Sprachkenntnisse

Die Mehrzahl der Software-Firmen im Bereich der Computer-Spiele arbeitet mit freien Mitarbeitern, es gibt aber auch die Möglichkeit einer festen Anstellung. Scharf kalkulierende Firmen setzen hier allerdings Talent und neue Ideen voraus. Eine Sicherheit bietet eine Festanstellung im Spielsoftware-Business allerdings nur selten, dafür hat man den Vorteil, in einem Team arbeiten zu können, und die Garantie der professionellen Vermarktung.

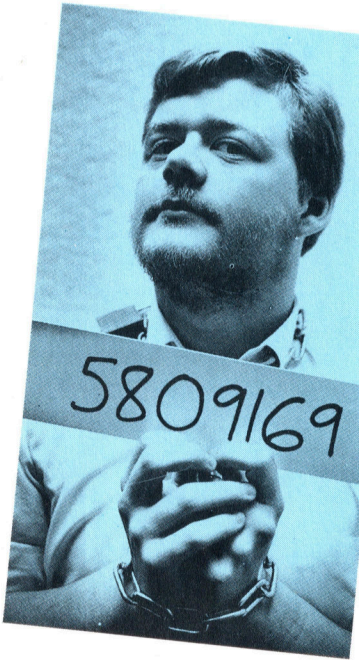
Software-Häuser gehen genau so schnell unter, wie sie auftauchen. Es gibt kaum gesicherte Zahlen, weil viele sogenannte Software-Häuser aus kaum mehr als einem Handelsnamen und einem „Gartenschuppen" bestehen. In der BRD gibt es mehrere Hundert Software-Häuser – die Hälfte davon überlebt nach der Gründung nicht einmal ein Jahr. Die Stabilität der Gesamtzahl täuscht eine Stabilität des Gewerbes vor – in Wahrheit bedeuten die Zahlen jedoch, daß nahezu die Hälfte der Unternehmen innerhalb eines Jahres stirbt!

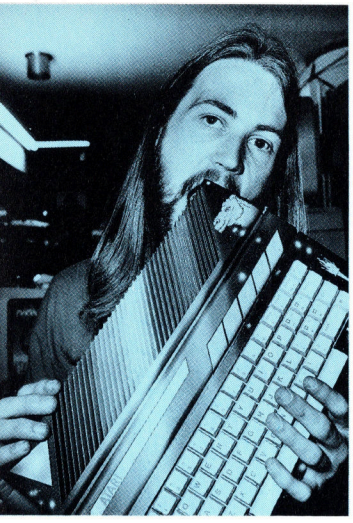
Weiterbildung und Aufstiegsmöglichkeiten sind von Firma zu Firma verschieden. Einige der größeren Unternehmen halten ernstzunehmende Weiterbildungskurse ab, in der Regel verdient die betriebliche Fortbildung ihren Namen aber nicht. Im Normalfall werden Programmierer in Teams zusammengefaßt, die unter der Leitung eines erfahrenen Gruppenleiters arbeiten. Dieser soll den jüngeren Kandidaten mit Rat und Tat zur Seite stehen. Einige Software-Firmen finanzieren ihren Angestellten auch den Besuch von Abendkursen – im allgemeinen tragen sie die Kursgebühren, der Kurs muß jedoch außerhalb der Arbeitszeit besucht werden.

In kleinen Firmen sind die Aufstiegsmöglichkeiten begrenzt – ein Direktor leitet das Unternehmen, die Programmierer entwickeln neue Software. Für den Verkauf von Programmen gibt es den Vertriebs- oder Marketingleiter, damit ist die Belegschaft dann auch schon komplett. In größeren Firmen führt der Aufstieg vom Jungprogrammierer über die Position des Gruppenleiters möglicherweise auch in die Geschäftsführung. Wer wirklich viel leistet, kann innerhalb von sechs Monaten Teamleiter werden. Wer weniger begabt ist, wird auch dann nicht gefördert, wenn er bereits mehrere Jahre im Unternehmen tätig ist.

Vorbildung und Erfahrung zählen wenig in diesem Geschäftsbereich. Der Unternehmer will Leistung sehen, also fertige Spielprogramme. Er braucht neue Ideen und ein Gespür für den Markt. Zwar geben die meisten Software-Firmen an, daß sie an Vorbildung interessiert sind, es wird aber sofort angemerkt, daß ein Mangel an Qualifikation einem talentierten Programmierer nicht im Wege steht.

Dave Nichols arbeitet ganztägig für die britische Ram-Jam Corporation. Seine Hauptaufgabe ist die Übertragung von Programmen, die ja nicht nur auf einem einzigen Computer laufen sollen.





**Den Profis unter den Heimcomputer-Spielern ist der Name Jeff Minter sicher ein Begriff: Seine Programme haben sich erfolgreich auf einem Markt mit starker Konkurrenz durchgesetzt. Jeff Minter ist Freiberufler. Auf lange Sicht kann die berufliche Zukunft eines Programmier-Einzelkämpfers aber höchst unsicher sein. Angestellte Programmierer können von der Weiterbildung und dem Gedankenaustausch mit Kollegen profitieren.**

Nicht selten wird ein ideenreicher 17-jähriger einem erfahrenen 40-jährigen vorgezogen.

Programmiert wird natürlich in jedem Fall in Maschinensprache. Normalerweise sind Kenntnisse über den 6502- oder Z80-Prozessor Voraussetzung, in letzter Zeit spielt auch der 68000 eine zunehmend wichtige Rolle. Mit BASIC ist hier überhaupt nichts anzufangen. Neben den "Sprachkenntnissen" sollte ein Bewerber auch gründlich über die zur Zeit aktuellen Computer informiert sein.

Zu diesen aktuellen Computern gehören in jedem Falle auch der Sinclair Spectrum, Commodore C64, Acorn B und Schneider CPC.

## Programmierer für Spiele:

**Vorbildung**
Eine formale Computerausbildung ist hilfreich, in diesem Job aber keine Bedingung.

**Computersprachen**
Maschinensprache, üblicherweise für Z80, 6502 oder 68000. Außerdem sollte man einen der gängigen Heimcomputer wirklich grundlegend kennen.

**Einkommen**
Extrem unterschiedlich. Ohne Hochschulabschluß liegt das Anfangsgehalt in einer Softwarefirma zwischen 1500 und 3000 Mark im Monat. Es hängt neben der Firmengröße auch von den Fähigkeiten des Anfängers ab. Angestellte mit Hochschulabschluß können industrieübliche Anfangsgehälter erwarten.

**Fortbildung durch die Firma**
Qualifizierte Fortbildung ist die Ausnahme. Die meisten Unternehmen beschränken sich auf das „Training on the Job", bei dem der erfahrenere Chefprogrammierer (Gruppenleiter) die Rolle des Lehrers übernimmt.

**Aufstiegsmöglichkeiten**
In kleineren Software-Firmen praktisch gleich Null. In größeren Unternehmen besteht die Chance, sich die Position eines Gruppenleiters zu erobern.

## Verkauf der Spiele

Falls Sie gerade ein Spiel geschrieben haben, das ähnlichen Erfolg wie Pacman, Defender oder gar Space Invaders haben könnte – warum wollen Sie damit nicht auf den Markt gehen? Diese kleine Checkliste bringt Sie auf den richtigen Weg.

**Vorsichtsmaßnahmen**
Die meisten Software-Häuser sind bedingungslos ehrlich – aber es gibt auch schwarze Schafe. Mit einigen kleinen Vorsichtsmaßnahmen können Sie sich schützen:

- Bringen Sie in der ersten Programmzeile einen Copyright-Vermerk unter. Falls möglich, sollten Sie so programmieren, daß sich dieser Vermerk nicht entfernen läßt. Falls das nicht geht, bringen Sie innerhalb des Programms weitere Copyright-Vermerke an.
- Installieren Sie Zeilen, die wie ein Teil des Programms wirken, aber keine echte Funktion besitzen. Wenn das Spiel ohne Ihre Erlaubnis auftaucht, können Sie durch das Vorhandensein dieser Zeilen Ihre Rechte leichter nachweisen.
- Deponieren Sie eine Kopie des Programms bei einem Anwalt oder einem Notar und lassen Sie sich das Datum der Hinterlegung schriftlich bestätigen.
- Bewahren Sie alles auf – frühere Programmversionen, Flußdiagramme, Programmteile und handschriftliche Notizen – alles, was Ihre Autorenschaft belegen kann.
- Bringen Sie Ihr Programm möglichst persönlich zur Softwarefirma und lassen Sie sich eine datierte Quittung geben. Falls Sie es mit der Post schicken, dann nur per Einschreiben mit Rückschein.

**Komplett liefern**
Softwarefirmen werden das Äußere des Spiels sicher gern abwandeln, einen neuen Titelscreen hinzufügen, eine Anleitung schreiben oder einen Schnell-Lader installieren. Es wird aber erwartet, daß das Programm selbst in jeder Beziehung vollständig, fehlerfrei und lauffähig ist.

**Stellen Sie sich gut dar!**
Softwarefirmen erhalten eine Vielzahl unaufgeforderter Zusendungen. Sorgen Sie dafür, daß Ihr Programm dabei positiv auffällt. Dazu können Sie

- eine kurze Beschreibung des Spiels geben
- klare Anweisungen für das Laden und den Spielverlauf beifügen
- für den Fall von Ladeschwierigkeiten mehrere Kopien schicken (Disketten sind Cassetten vorzuziehen!)
- ein Flußdiagramm mitliefern, aus dem die Struktur des Spiels deutlich wird
- detailliert alle Anforderungen beschreiben, die der Rechner zusätzlich zur Grundausstattung erfüllen muß, etwa 48 KByte RAM, Joysticks, ein Diskettenlaufwerk etc. Versehen Sie alles mit Ihrem Namen, Ihrer Adresse und der Telefonnummer, unter der Sie tagsüber zu erreichen sind.

**Legen Sie sich nicht fest!**
Schicken Sie Ihr Programm an möglichst viele Softwarefirmen und nehmen Sie nicht gleich das erste Angebot an. Wenn Sie ein Angebot haben, rufen Sie einige der Konkurrenzfirmen an und geben diese Information weiter, ohne aber ins Detail zu gehen. Dadurch könnte die Prüfung Ihres Spiels beschleunigt werden.

Unterzeichnen Sie keinen Vertrag ohne fachmännische Beratung. Sprechen Sie mit einem Anwalt. Bei vielen Anwälten ist ein halbstündiges Beratungsgespräch nicht übermäßig teuer. Die anfallenden Gebühren sind in jedem Fall gut angelegt!

# Richtig adressiert

**In der Reihe über den 68000-Chip untersuchen wir die Adressierung der Operanden. Dieser Bereich der Programmierung hat besonders für strukturierte Datentypen wie Records und Arrays große Bedeutung.**

In der letzten Folge dieser Reihe hatten wir die Adressierungsmöglichkeiten des 68000-Befehlssatzes untersucht. Dabei steht eine breite Palette von Adressierungsarten zur Verfügung, mit der sich Bytes, Computerworte oder Langworte leicht ansprechen lassen.

Der „Standardbefehl"

OPCODE Quelle, Ziel

zeigt, wie eine bestimmte Befehlsklasse die Operanden für Quelle und Ziel behandelt: Zuerst wird der Quellenoperand geholt (egal mit welchem Ablauf), dann mit dem Opcode-Vorgang bearbeitet und das Ergebnis schließlich an den Zieloperanden übergeben. So bestimmt beispielsweise der Befehl MOVEA D3,A6 daß der Inhalt von D3 (Quelle) auf A6 (Ziel) verschoben wird. Dabei gibt der Opcode MOVEA an, daß eine Adresse bewegt wird.

In diesem einfachen Beispiel wird nun die Operandenadresse direkt angegeben, doch gibt es am anderen Ende der Skala Adressierungsarten, die Operanden berechnen, indem sie den Inhalt eines Adreßregisters auf eine Offset-Zahl und ein Indexregister addieren (wie bei dem Z80-Befehl LD r,(IX+d)).

In unserem Befehlsmodell gibt es auch Opcodes mit nur einem Operanden, zum Beispiel:

OPCODE Quelle

Befehle dieser Art werden oft für Verzweigungen (wie BRA ZURUECK) eingesetzt, die nur einen Operanden brauchen, und zwar die Verzweigungsadresse (in diesem Fall „ZURUECK"). Und schließlich gibt es auch noch

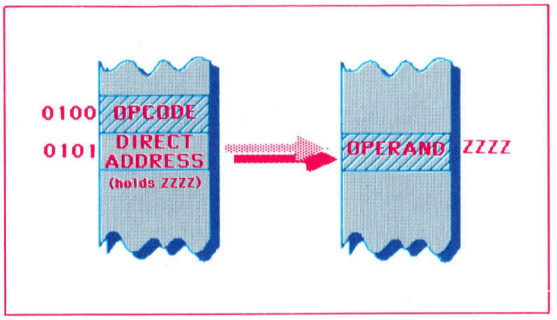
OPCODE

ohne jeden Operanden. Ein klassisches Beispiel dafür ist der Befehl NOP – die Leeranweisung. Mit diesem „Dummy"-Befehl lassen sich Programme sehr leicht im Speicher ändern oder patchen.

Bei all diesen Befehlsmodellen kann es also vorkommen, daß Adressen berechnet werden müssen. Welche Berechnungen ausgeführt werden, legt der Programmierer mit Anweisungen fest, die er aus dem Befehlssatz des 68000 und seinen Adressierungsarten auswählt.

Die meisten Computer verfügen über mindestens fünf Adressierungsarten (d. h. Methoden, die Operanden anzusprechen). Unser Bild zeigt die unterschiedlichen zwei Abläufe dieser Arten:

- **Direkte (oder absolute) Adressierung:** Dabei ist die Speicheradresse des Operanden im Befehl selbst untergebracht.
- **Indirekte (oder Pointer-)Adressierung:** Hier enthält der Befehl eine Speicheradresse, in der die Adresse des Operanden untergebracht ist.

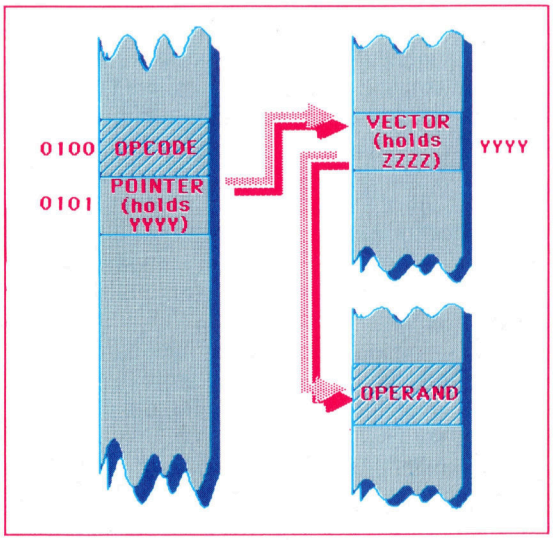



**In diesem Adressiermodus folgt dem Opcode die Adresse einer Speicherstelle mit den Operandendaten.**

Das Bild zeigt, daß der Opcode der direkten Adressierung direkt mit dem Operanden bei ZZZZ arbeitet. Zwar steht der Operand auch bei der indirekten Adressierung in ZZZZ, doch wird er hier über den Pointer in YYYY angesprochen.

Hier die anderen drei Adressierungsarten:

- **Unmittelbare Adressierung:** Der Befehl enthält eine Konstante. In MOVEQ #25,D3 wird die Konstante 25 unmittelbar angesprochen.
- **Registermodus:** Hier werden Register als Operanden eingesetzt und durch den Op-Code definiert. Die Operanden des Befehls MOVE D2,D4 sind die entsprechenden Datenregister D2 und D4.




**Bei der indirekten Adressierung greift der Opcode über eine „Vektoradresse" auf seinen Operanden zu. Die Adresse des Vektors steht unmittelbar hinter dem Opcode, während der Vektor die Adresse des Operanden enthält. Diese Adressierungsart ist besonders praktisch, wenn die Operandenadresse erst während des Programmablaufs bekannt wird. Es muß dann nur der Inhalt des Vektors berechnet werden.**

● **Implizite Adressierung:** Bei dieser Art der Adressierung liefert der Befehl selbst die Operanden. So sind beispielsweise bei RTS (Rücksprung von einer Subroutine) der Stack-Pointer und der Programmzähler die implizierten Operanden.

Interessanterweise hat der 68000 außer den bereits beschriebenen Adressierungsarten die Möglichkeit, Adressen relativ zum Programmzähler (auch „PC relativ" genannt) anzusprechen. Hier die Adressierungsarten im einzelnen:

● **Absolute Adressierung:** Kann jede beliebige Speicherstelle ansprechen. Die Adresse des Operanden steht hinter dem Befehl, zum Beispiel:

```
Adresse: Code:  Label:  Befehl:
1000     D678           ADD     DATA,D3
1002     2000
-
-
-
2000     0001   DATA    DC.W    1
```

Dabei wurde der Adresse $2000 der symbolische Name DATA gegeben. Der Befehl ADD mit seiner absoluten Quelladresse (DATA) und dem Ziel D3 wurde als D678 codiert, während die absolute Adresse DATA in $ 1002 (auch „Worterweiterung" genannt) gespeichert ist. Hier ein Beispiel mit nur einem Operanden:

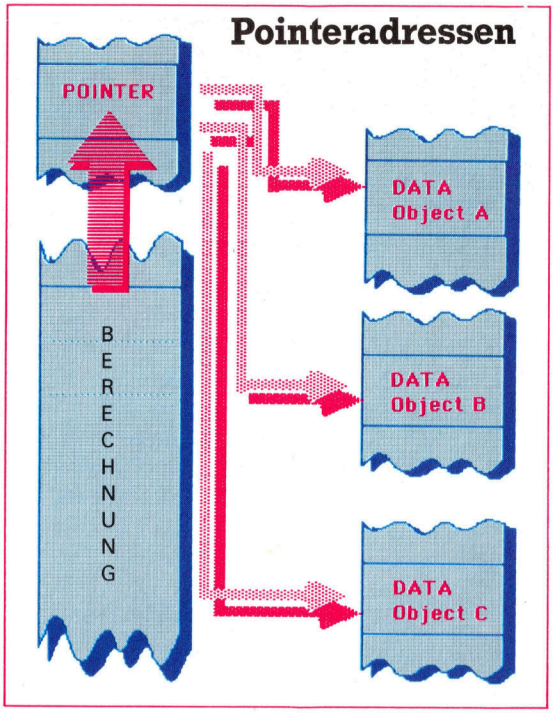
```
Adresse: Code:  Label:  Befehl:
1000     4278           CLR     COUNT
1002     3000
-
-
-
3000     0      COUNT   DS      1
```

Der Befehl CLR setzt dabei den Inhalt der Speicherstelle $3000 (COUNT) auf Null.

In der vorigen Folge hatten wir erwähnt, daß der PC von einem 32-Bit-Register dargestellt wird (von dem sich allerdings nur 24 Bits einsetzen lassen). Das bedeutet, daß die absolute Adresse, die den Operanden bezeichnet, nicht nur aus einem Wort – wie in den beiden letzten Beispielen – bestehen muß. Wie aber kann der Assembler feststellen, wieviel Platz er für die absolute Adresse reservieren soll? Da es Platzverschwendung wäre, wenn für jede absolute Adresse eine Langworterweiterung bereitgestellt würde, reserviert der Assembler – nachdem er die Operandenadresse festgestellt hat – eine Erweiterung in exakt der nötigen Länge.

Sie können den Assembler jedoch auch anweisen, für das ganze Programm lange oder kurze Erweiterungen je nach Bedarf zur Verfügung zu stellen.

Sehen wir uns einmal genauer an, wie in dem ADD-Beispiel die Langworterweiterung eingesetzt ist:




```
Adresse: Code:  Label:  Befehl:
1000     D679           ADD     HIDATA,D3
1002     0020
1004     0000
```

## Adressierungen

Hier ist $200000 die absolute Adresse von HIDATA. Beachten Sie, daß der Code für den ADD-Bereich des Befehls (D6) gleich geblieben ist, die Operandenadresse jedoch von $78 auf $79 verändert wurde. Wir werden in späteren Artikeln über den 68000-Assembler den Einsatz von Langworterweiterungen noch ausführlich erklären.

● **Registeradressierung:** Dies ist die einfachste Adressiermethode des 68000, da dabei nur die internen Register zum Einsatz kommen. So addiert ADD DO,D3 beispielsweise das Wort DO auf den Inhalt von D3.

Dieser Adressiermodus hat jedoch einige Einschränkungen. So können etwa Adreßregister nicht als Ziel eines Additionsvorganges angegeben werden – ADD DO,A4 ist nicht möglich. Mit dem Additionsbefehl ADDA läßt sich diese Einschränkung aber umgehen – ADDA DO,A4 ist legal.

Wenn Sie in den obigen Beispielen Langworte als Datenobjekte einsetzen wollen, brauchen Sie nur „.L" an den Befehl anzuhängen: ADD.L DO,D3 verarbeitet 32-Bit-Worte als Datenobjekte.

● **Indirekte Registeradressierung:** Diese Adressiermethode ist vermutlich die Wichtigste des 68000, da sie mit „Pointern" arbeitet. Auch lassen sich damit Stack-Vorgänge ausführen. Sehen wir uns zuerst die Pointer genauer an.

Programme müssen auf Datenobjekte – Bytes, Worte, Langworte oder strukturierte Objekte wie Arrays oder Records – „zeigen" können. Wenn beispielsweise ein Ablauf mehrfach für Daten des gleichen Typs ausgeführt werden soll, sind Pointer sehr praktisch. Das nebenstehende Bild zeigt, wie ein Pointer die einzelnen Elemente eines Records bezeichnet. Anfangs leitet er die Berechnung auf das Datenobjekt A, wird danach umgestellt und zeigt dann auf das nächste Datenobjekt.

Der Code des 68000 stellt den Pointer jedoch nicht direkt hinter den Befehl, sondern arbeitet mit einem Adreßregisterpointer. Dies mag in einigen Situationen etwas unbequem sein, doch stehen dafür acht Adreßregister zur Verfügung.

Hier ein Beispiel für eine einfache indirekte Adressierung:

```
LEA       INPUT,A0

MOVE.W (A0),D3
```

Der Befehl LEA lädt AO mit der Adresse eines Eingabedatenobjektes namens INPUT, das dann auf D3 kopiert wird (siehe Bild).
Wie läßt sich diese Adressiermethode für eine ganze Datenliste einsetzen? Zunächst brauchen wir eine „Nach-Inkrementierung". Hier wird der Pointer nach dem Zugriff auf das Datenobjekt inkrementiert und zeigt dann auf das nächste Element der Liste. Das nächste Beispiel zeigt, wie die Nach-Inkrementierung bei Computerworten arbeitet:

```
MOVE.W (A0)+,D3
```

Hier wird der Pointer nach dem Zugriff auf AO und dem Kopiervorgang auf D3 automatisch um Zwei inkrementiert. Die Adresse AO zeigt anfangs auf $2000, nach dem MOVE.W-Vorgang aber auf $2002. Für die Bearbeitung von Byteobjekten mit MOVE.B wird AO um Eins, bei MOVE.L um Vier inkrementiert.

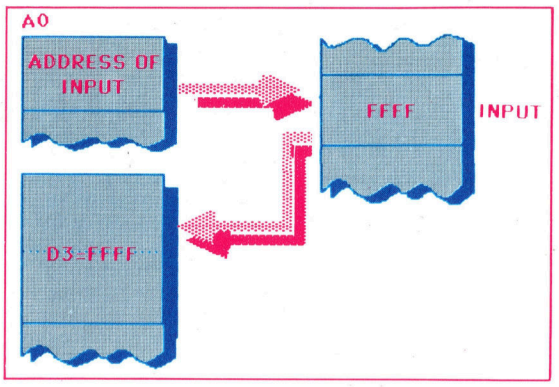
Wie flexibel diese Adressierungsart ist, zeigt sich deutlich in der folgenden Programmschleife zur Berechnung einer Liste:

```
LEA       INPUT,AO    Initialisieren
MOVE.W    (AO),D3     Jedes Element
                      kopieren
          :
          :
          Berechnung
          :
          :
          Auf Listenende
          prüfen
```

Bei jedem Ablauf der Berechnung deutet der Zeiger nach dem MOVE-Befehl automatisch auf das nächste Listenelement.

Es gibt jedoch auch die „Vor-Inkrementierung". Hier wird der Adreßpointer vor dem Zugriff auf das Datenobjekt dekrementiert.

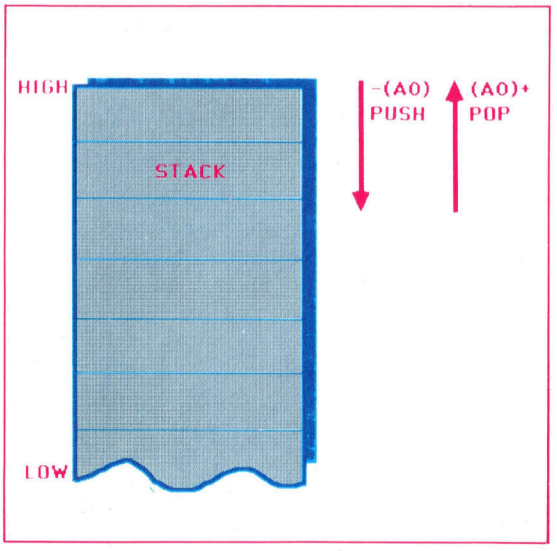
```
MOVE.W —(AO),D3
```




AO wird vor dem Kopieren des Datenobjektes auf D3 um zwei dekrementiert.

Die vor-inkrementierte indirekte Adressierung entspricht der indirekten Nach-Inkrementierung. Die Nach-Inkrementierung bearbeitet eine Liste in der Reihenfolge aufsteigender Adressen, während die Vor-Dekrementierung auf umgekehrte Weise vorgeht. Die vor- und nachgestellten Abläufe lassen sich jedoch nicht beliebig gegeneinander austauschen. So ist beispielsweise weder—(AO) noch (AO)+ möglich.

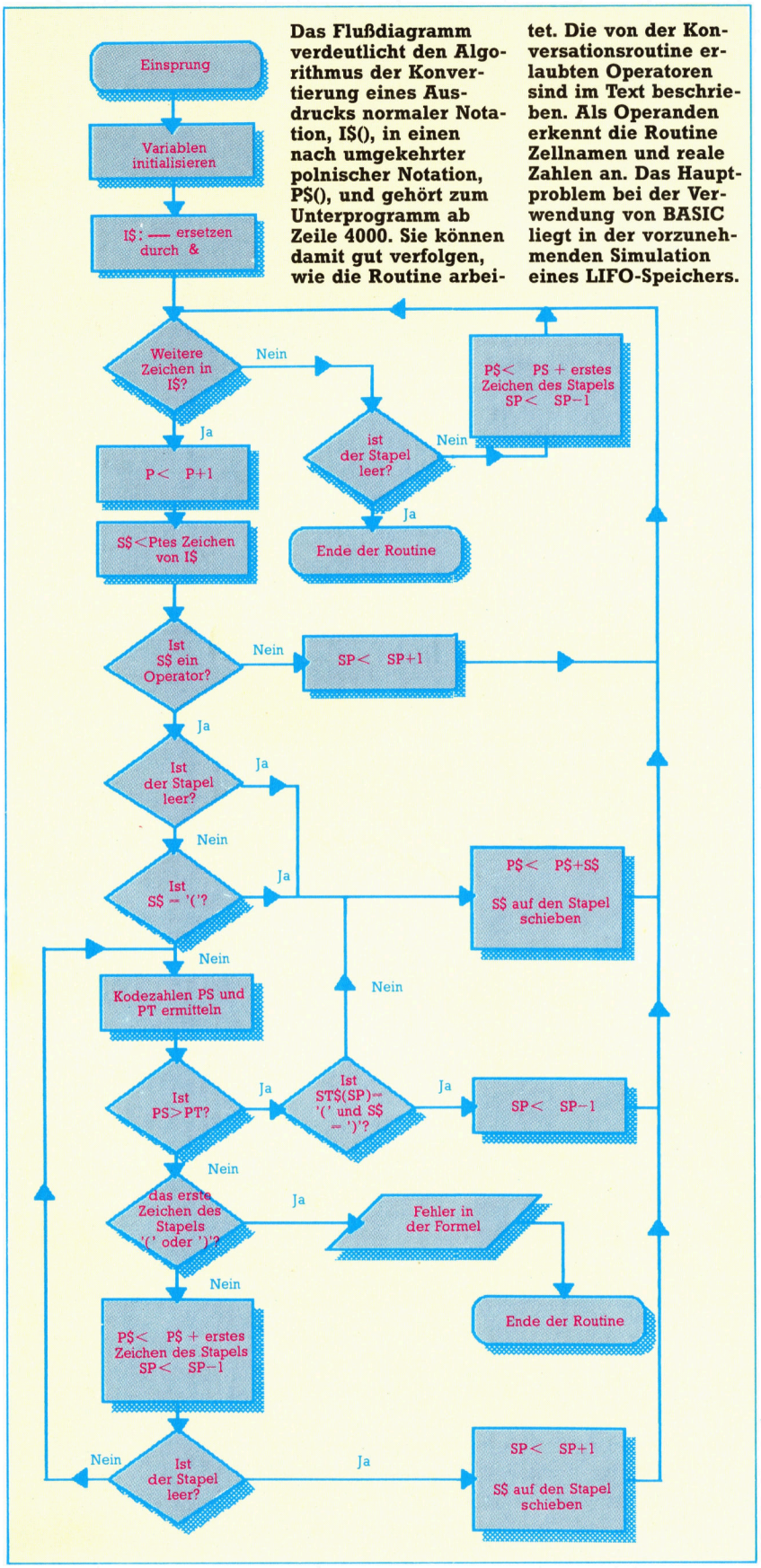
Sie mögen bereits bemerkt haben, daß die Adressierungsarten der Vor-Dekrementierung und des Nach-Inkrements den Stack-Vorgängen entspricht. Sie können damit Daten auf einen Stack „schieben" oder „herunterziehen", wobei ein Stack praktischerweise als eine Reihe von Adressen im Hi-Lo-Format verstanden wird. So legt MOVE DO,-(AO) Daten auf einen Stack, während MOVE (AO)+,DO sie wieder herunterzieht. Das Bild verdeutlicht diesen Vorgang des „Schiebens" und „Ziehens".




**Die vertrauten Z80-Befehle PUSH und POP lassen sich auf dem 68000 mit den außerordentlich flexiblen Befehlserweiterungen der Vor-Dekrementierung und Nach-Inkrementierung simulieren.**

Auf Stacks und ihren Einsatz gehen wir später in der 68000-Serie noch genauer ein, da diese praktischen Adressierarten leicht und gänzlich unkompliziert auf die entsprechenden Datenlisten zugreifen können.

# Formelauswertung




**Das Flußdiagramm verdeutlicht den Algorithmus der Konvertierung eines Ausdrucks normaler Notation, I$(), in einen nach umgekehrter polnischer Notation, P$(), und gehört zum Unterprogramm ab Zeile 4000. Sie können damit gut verfolgen, wie die Routine arbeitet. Die von der Konversationsroutine erlaubten Operatoren sind im Text beschrieben. Als Operanden erkennt die Routine Zellnamen und reale Zahlen an. Das Hauptproblem bei der Verwendung von BASIC liegt in der vorzunehmenden Simulation eines LIFO-Speichers.**

**Die wichtigste Eigenschaft der Tabellenkalkulationsprogramme ist das Berechnen der Zelleninhalte anhand von Formeln. In dieser Folge besprechen wir die Routinen zum Eingeben und Ausführen der mathematischen Formeln durch das Programm, unter Berücksichtigung der umgekehrten polnischen Notation.**

Beim Arbeiten mit einem Kalkulationsprogramm, wie auch bei vielen ähnlichen Anwendungen, sollten Sie berücksichtigen, daß der Computer Ihre Berechnungen in einer anderen Reihenfolge ausführt. Nehmen wir zum Beispiel an, Sie geben (A1+A2)*(A3—A4) in die Zelle A5 ein. Das Ergebnis errechnen wir nach der normalen Notation durch separates Ermitteln der Resultate der beiden Klammerinhalte und deren anschließender Multiplikation. Damit der Computer die gleichen Schritte nachvollzieht, müssen wir ihm folgende Instruktionen geben:

1. Nimm die Inhalte der Zellen A1 und A2.
2. Addiere diese Werte und lege das Resultat in einen Zwischenspeicher.
3. Nimm die Inhalte der Zellen A3 und A4.
4. Subtrahiere den Wert in A4 von dem in A3.
5. Multipliziere das Resultat mit dem Resultat der ersten Rechnung.
6. Lege das Endergebnis in die Zelle A5.

Sie könnten jetzt ein Programm schreiben, mit dem der Computer Ihre Anweisungen in dieser Reihenfolge abarbeitet, doch bei Verwendung vieler unterschiedlicher Formeln ist das sehr aufwendig. Besser ist eine Routine, die jede Formel so umsetzt, daß der Computer sie korrekt verarbeitet.

Das Problem beim Schreiben mathematischer Ausdrücke in der normalen Notation ist, daß der Computer sie nicht in der gleichen Reihenfolge verarbeitet. Um dies zu umgehen, benutzen wir die umgekehrte polnische Notation zum Schreiben unserer Formeln.

Die Umwandlung und Ausführung der Formeln in unserem Arbeitsblatt erfolgt in drei Hauptschritten: Zunächst wandelt der Rechner die Formel von der normalen Schreibweise in die UPN um, Schritt 2 prüft die gewandelte Formel auf richtige Schreibweise, und der dritte Schritt ist der eigentliche Rechenvorgang.

Die vom Programm erlaubten Operatoren sind +, —, *, / und ↑. Wenn Sie das —Zeichen zum Kennzeichnen eines negativen Wertes benutzen, benötigt das Programm zur Unterscheidung vom Subtraktionszeichen ein zu-

sätzliches Symbol, das &-Zeichen. Während der Konvertierung von normaler Notation in UPN weisen wir jedem Operator eine Kodezahl zu. Diese bestimmt die Reihenfolge, in der die Operatoren später verarbeitet werden. Division und Multiplikation haben zum Beispiel Vorrang vor Addition und Subtraktion:

| Operator | Kodezahl |
|---|---|
| = | 0 |
| ( | 1 |
| +-) | 2 |
| */& | 3 |
| ↑ | 4 |

In der endgültigen Version des Kalkulationsprogramms arbeitet die Routine zum Konvertieren von Normaler Notation auf UPN folgendermaßen: Nach Eingabe einer Formel, die sich auf mehrere Zellen bezieht, wird der resultierende Ausdruck in der eingegebenen Notation im F$()-Array abgelegt. Hier sind die Formeln aller Zellen gespeichert. Das entsprechende Element eines zweiten Zeichenfeldes, PS$(), das die umgewandelten UPN-Ausdrücke aufnimmt, wird dabei gelöscht.

Nach Anwahl der Funktion CALCULATE rechnet das Programm die einzelnen UPN-Zeichenketten des Feldes PS$() durch.

## String-Umwandlung

**BASIC-Dialekte**

**Acorn B:**

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

Zeile 50 entfällt.
4240 IF ST$(SP)="("OR ST$(SP)=")"THEN PRINT TAB(0,22);"ERROR IN FORMULA":RETURN

**Schneider CPC:**

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

Zeile 50 entfällt.
4240 IF ST$(SP)="("OR ST$(SP)=")"THEN LOCATE 1,22:PRINT"ERROR IN FORMULA":RETURN
Zeile 3140 erhält die neue Zeilennummer 3160.

**Commodore 64:**

```
50 CD$=CHR$(17):CU$=CHR$(145):CR$=CHR$(2
9):CL$=CHR$(157):CO$=CHR$(19)
```

**Nachtrag zur Feldinitialisierung UPN Konvertierungsroutine**

```
3140 DIM F$(255):DIM PS$(255)
```

**RPN Konvertierungsroutine**

```
4000 REM ******************************
4001 REM * NORMAL TO REVERSE POLISH  *
4002 REM * STRING CONVERSION         *
4003 REM ******************************
4005 FOR K=1 TO 20:ST$(K)=" ":NEXT K
4006 LET P=0:LET SP=0:P$=""
4010 LET I$=C$
4015 FOR K=1 TO LEN(I$)-1
4017 J$=MID$(I$,K,1):K$=MID$(I$,K+1,1)
4020 IF J$="(" AND K$="-" THEN I$=MID$(I
$,1,K)+"&"+MID$(I$,K+2)
4025 NEXT K
4030 IF P<=LEN(I$) THEN 4100
4040 REM *** EMPTY STACK *****
4050 IF SP=0 THEN PS$((J-1)*15+I)=P$:GOS
UB 4700:RETURN
4060 IF ST$(SP)="(" OR ST$(SP)=")" THEN
4080
4070 LET P$=P$+ST$(SP)
4080 LET SP=SP-1
4090 GOTO 4050
4095 STOP
4100 LET P=P+1
4110 LET S$=MID$(I$,P,1)
4120 IF S$="+" OR S$="-" OR S$="*" OR S$
="/" THEN 4200
4130 IF S$="^" OR S$="(" OR S$=")" OR S$
="&" THEN 4200
4140 LET P$=P$+S$
4150 GOTO 4030: REM CONTINUE TO END OF S
TRING
4200 IF SP=0 THEN 4400
4210 IF S$="(" THEN 4400
4220 GOSUB 4500:REM ** PRECEDENCE **
4230 IF PS>PT THEN 4300: REM PRECEDENCE
4240 IFST$(SP)="("ORST$(SP)=")"THENGOSUB
1950:PRINT"   ERROR IN FORMULA ":RETURN
4250 LET P$=P$+ST$(SP):ST$(SP)="":LET SP
=SP-1
4260 IF SP>0 THEN 4220
4270 SP=SP+1
4280 LET ST$(SP)=S$
4290 GOTO 4030: REM CONTINUE TO END OF S
TRING
4300 IF ST$(SP)="(" AND S$=")" THEN LET
ST$(SP)="":SP=SP-1:GOTO 4030
4400 LET SP=SP+1
4410 LET ST$(SP)=S$
4420 GOTO 4030
4500 REM *** PRECEDENCE EVALUATION ***
4510 LET T$=S$:GOSUB 4600
4520 LET PS=T
4530 LET T$=ST$(SP):GOSUB 4600
4540 LET PT=T
4550 RETURN
4600 IF T$="=" THEN T=0:RETURN
4605 IF T$="(" THEN T=1:RETURN
4610 IF T$="+" OR T$="-" OR T$=")" THEN
T=2:RETURN
4620 IF T$="*" OR T$="/" OR T$="&" THEN
T=3:RETURN
4630 IF T$="^" THEN T=4:RETURN
4650 T=0:RETURN
```

**Sinclair Spectrum:**

**Nachtrag zur Feldinitialisierung UPN Konvertierungsroutine**

```
3150 DIM F$(255,20):DIM H$(255,2
0):RETURN
```

**RPN Konvertierungsroutine**

```
4000>REM ************************
4001  REM * NORMAL TO REVERSE
*
4002 REM * POLISH STRING CONV. *
4003 REM ************************
4005 DIM S$(20)
4006 LET P=0: LET SP=0: LET P$="
"
4010 LET I$=C$
4011 FOR Z=1 TO LEN (I$)
4012 IF I$(Z)=" " THEN  GO TO 40
14
4013 NEXT Z
4014 LET I$=I$( TO Z-1)
4015 FOR K=1 TO LEN (I$)-1
4017 LET J$=I$(K): LET K$=I$(K+1
)
4020 IF J$="(" AND K$="-" THEN
LET I$=I$(1 TO K)+"&"+I$(K+2 TO
)
4025 NEXT K
4030 IF P<LEN (I$) THEN  GO TO 4
100
4040 REM **** EMPTY STACK ****
4050 IF SP=0 THEN  LET H$((J-1)*
15+I,1 TO )=P$: GO SUB 4700: RET
URN
4060 IF S$(SP)="(" OR S$=")" THE
N  GO TO 4080
4070 LET P$=P$+S$(SP)
4080 LET SP=SP-1
4090 GO TO 4050
4095 STOP
4100 LET P=P+1
4110 LET O$=I$(P)
4120 IF O$="+" OR O$="-" OR O$="
*" OR O$="/" THEN  GO TO 4200
4125 IF O$="^" OR O$="(" OR O$="
)" OR O$="&" THEN  GO TO 4200
4140 LET P$=P$+O$
4150 GO TO 4030
4200 IF SP=0 THEN  GO TO 4400
4210 IF O$="(" THEN  GO TO 4400
4220 GO SUB 4500: REM PRECEDENCE
4230 IF PS>PT THEN  GO TO 4300
4240 IF S$(SP)="(" OR S$(SP)=")"
 THEN  PRINT AT 20,0;"ERROR IN F
ORMULA": RETURN
4250 LET P$=P$+S$(SP): LET SP=SP
-1
4260 IF SP>0 THEN  GO TO 4220
4270 LET SP=SP+1
4280 LET S$(SP)=O$
4290 GO TO 4030: REM CONTINUE TO
 END OF STRING
4300 IF S$(SP)="(" AND O$=")" TH
EN  LET S$(SP)="": LET SP=SP-1:
GO TO 4030
4400 LET SP=SP+1
4410 LET S$(SP)=O$
4420 GO TO 4030
4500 REM * PRECEDENCE EVALUATION
 *
4510 LET T$=O$: GO SUB 4600
4520 LET PS=T
4530 LET T$=S$(SP): GO SUB 4600
4540 LET PT=T
4550 RETURN
4600 IF T$="=" THEN  LET T=0: RE
TURN
4610 IF T$="(" THEN  LET T=1: RE
TURN
4620 IF T$="+" OR T$="-" OR T$="
)" THEN  LET T=2: RETURN
4630 IF T$="*" OR T$="/" OR T$="
&" THEN  LET T=3: RETURN
4640 IF T$="^" THEN  LET T=4: RE
TURN
4650 LET T=0: RETURN
```

# Wortwechsel mit Unix

**Wie von einem Mehrbenutzersystem zu erwarten, bietet Unix eine Vielzahl von hilfreichen und komfortablen Dienstprogrammen. In C geschrieben, sind sie leicht über Systemkommandos aufrufbar. Der folgende Artikel befaßt sich mit einer Auswahl von Befehlen der Berkeley-Version 4.2.**

**Ohne besondere Vereinbarung erwarten unter Unix laufende Programme die Eingabedaten über die Tastatur, während Ergebnisse und Fehlermeldungen auf dem Bildschirm erscheinen. Mit Hilfe der Operatoren „<" und „>" läßt sich aber erreichen, daß bei Ein- und Ausgabe auch beliebige andere Geräte oder Dateien angesprochen werden können. Dieses Verfahren erlaubt auf einfache Weise die Herstellung von Datenfiles sowie die Ausführung von Unix-Befehlsfolgen, die in einer Kommandodatei abgelegt sind, durch die „Shell" als Kommandointerpreter.**

Da die Unix-Utilities und Tools in C geschrieben sind, bereitet die Kombination mit eigenen Programmen keinerlei Schwierigkeiten; die einheitliche Behandlung von Dateien und Ein/Ausgabegeräten unter Unix erhöht dabei noch die Benutzerfreundlichkeit. Jede Datei wird als eine Sequenz von einzelnen Bytes aufgefaßt – Unix liest und füllt die Dateien byteweise. Der Dateizeiger kann auf jedes definierte Byte gesetzt werden, um dort mit Lesen oder Schreiben zu beginnen.

Die Ein/Ausgabekanäle werden genau wie normale Dateien angesprochen. Sie finden eine Liste der zugehörigen Geräte (Devices) in der /dev-Directory. Die vorgesehenen Dateioperationen lassen sich am besten anhand der entsprechenden Kommandos erläutern, die wir hier zusammen mit einer kurzen Funktionsbeschreibung abdrucken:

- **creat (Dateiname, Zugriffserlaubnis):** Wenn bereits eine Datei gleichen Namens existiert, wird sie auf Null reduziert, andernfalls erzeugt das System eine neue Datei. Die Zugriffserlaubnis ist eine neunstellige Zahl, im allgemeinen durch drei Oktalziffern dargestellt, die Lese-, Schreib- und Ausführungsbefugnisse (mit je einem Bit) spezifiziert. Der Aufruf wird mit der Ausgabe eines ganzzahligen Dateideskriptors beantwortet.
- **open (Dateiname, Schreib/Lesefreigabe):** Eröffnet für ein Programm den Zugriff auf die genannte Datei, wobei als Schreib/Lesefreigabe eine „0" nur Lesen, eine „1" nur Schreiben und eine „2" beides zuläßt. Auch hier wird der Datei ein ganzzahliger Dateideskriptor mitgeteilt.
- **close (Dateiname):** Schließt die Datei für das betreffende Programm.
- **unlink (Dateiname):** Entfernt die Datei aus dem System.
- **read (Dateideskriptor, Buffer, Byteanzahl) bzw. write (s. o.):** Überträgt die angegebene Anzahl von Bytes zwischen der Datei laut Deskriptor und dem namentlich genannten Bufferspeicher.

Drei (fiktive) Dateien gelten für jedes Programm automatisch als eröffnet: „standardinput", „standardoutput" und „standarderror". Dabei ist mit standardinput die Tastatureingabe gemeint, während standardoutput und standarderror auf den Bildschirm wirken. Die input- und output-Ströme können mit Hilfe der Operatoren „<" und „>" aber auch beliebigen anderen Dateien oder Geräten zugeordnet werden. Jedes Unix-Kommando läßt sich in der Form ergänzen

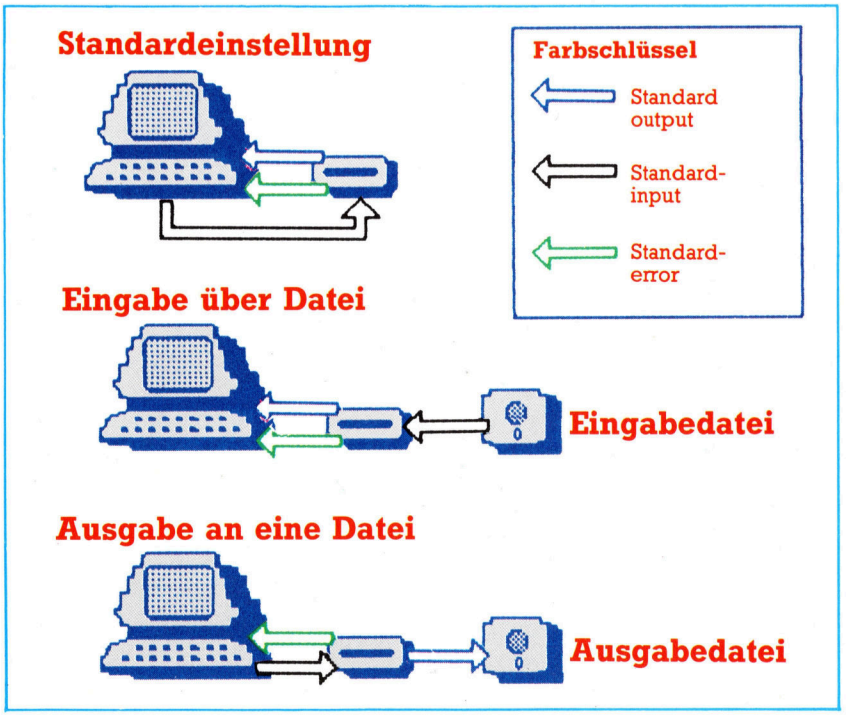
Befehl < Datei oder Gerät

Dann werden die Eingabedaten über die genannte Datei (bzw. das angegebene Gerät) statt von der Tastatur her erwartet. Ähnlich legt Zuweisung

Befehl > Datei oder Gerät

fest, daß die Ausgabe nicht zum Bildschirm, sondern zu dem angegebenen Gerät geleitet wird. Eine Variante dieses Kommandos ermöglicht es, die Ausgabe an den Inhalt einer bereits bestehenden Datei anzuschließen, statt eine neue hervorzubringen:

Befehl > Datei oder Gerät



Als Beispiel für den Umgang mit den oben angeführten Anweisungen soll eine Datei „lsfile" erzeugt werden, die alle Dateinamen einer Directory enthält. Diese Datei wird mittels des Kommandos „ls" (Auflisten der Directory) gefüllt:

```
ls > lsfile
```

Sie können auch eine Kommandodatei erstellen, durch die Sie dann mit einem Aufruf eine ganze Befehlsfolge ausführen lassen können. Der Unix-Kommandointerpreter (die „Shell") ist über den sh-Befehl anzusprechen; mittels

```
sh < Dateiname
```

gelangen die Kommandos der angegebenen Datei zur Ausführung.

Ein wesentlicher Faktor für die Leistungsfähigkeit von Unix ist das „Pipeline"-Verfahren – in allgemeiner Schreibweise

```
1.Befehl | 2.Befehl
```

Der Operator „|" bewirkt, daß der „standardoutput" der zuerst aufgerufenen Routine gleich über „standardinput" der zweiten Routine als Eingabe zugeführt wird. Die beiden Programme laufen verzahnt ab.

Ein Beispiel: Bei dem Kommando „ls" zum Auflisten der Directory ist das Zählen der Files nicht vorgesehen. Für das Zählen der Wörter in einer Datei gibt es aber das Kommando „wc" mit der Option „_w", so daß über

```
ls | wc _w
```

auf dem Bildschirm als „standardoutput" die Anzahl der Files erscheint.

Der Pipeline-Operator läßt sich u. a. in Verbindung mit dem „tee"-Befehl (T-Verzweigung) verwenden, um Ergebnisse zugleich an den Bildschirm und eine „echte" Datei zu senden, beispielsweise in der Form

```
ls | tee lsfile
```

Das Dateiverzeichnis gelangt auf diese Weise ins „lsfile" und auch auf den Bildschirm.

Unix ist sowohl ein Mehrbenutzer- als auch ein Mehrprogrammsystem. Wenn eine Befehlszeile mit dem „&"-zeichen abgeschlossen ist, schiebt der Kommandointerpreter die betreffenden Vorgänge in den Hintergrund.

Jeder Bearbeitungsvorgang oder „Prozeß" läßt sich beliebig durch Ctrl-Z unterbrechen und später an derselben Stelle wieder fortsetzen. Wenn Sie eine Reihe von Prozessen gleichzeitig in Gang haben, verlieren Sie unter Umständen die Übersicht; dann können Sie mit „ps" (Prozeß-Status) einen aktuellen Zustandsbericht oder mit „jobs" ein Verzeichnis aller im Hintergrund laufenden Prozesse anfordern.

Wird ein Prozeß unterbrochen oder in den Hintergrund geschoben, teilt Unix ihm eine persönliche Jobnummer (in eckigen Klammern) zu sowie eine Prozeßnummer, die sämtliche beim System angemeldeten Prozesse berücksichtigt. Durch den Befehl „fg %Jobnummer" kann ein gestopptes Programm im Vordergrund (Foreground=fg) wieder gestartet oder ein Hintergrundprogramm in den Vordergrund gebracht werden. Die Kommandos „stop %Jobnummer" und „bg%Jobnummer" dienen zum Stoppen eines Programms im Hintergrund (Background=bg) bzw. zum Wiederstarten nach dem Anhalten. Endgültig abgebrochen wird ein Prozeß durch „kill Prozeßnummer".

## Verkehrsregelung

```
%ls -l > lsfile
(Directory erscheint nicht auf dem Bildschirm)

%cat lsfile          (Ausgabe von lsfile)
total 41
-rw-r--r--    1 com-mcc      0  Oct 28 11:55 lsfile
drwxr-xr-x    2 com-mcc    512  Oct 21 11:11 mike
-rw-rw-r--    1 com-mcc    502  Sep 17 12:07 rec.c
-rwxr-xr-x    1 com-mcc  18432  Oct 21 11:02 receive
-rw-r--r--    1 com-mcc   1068  Oct 18 14:44 rx.p
-rwxr-xr-x    1 com-mcc  19456  Oct 21 11:03 transmit

%ls | sort -r > lsfile
(Directory wird gleich der sort-Routine übergeben, wobei die
option -r Sortieren entgegen der alphabetischen lsfile-Reihenfolge bewirkt)
%cat lsfile  (Ausgabe des umsortierten lsfile)
transmit
rx.p
receive
rec.c
mike
lsfile

%pc rx.p &   (Compilierung des Pascal-Quellprogramms rx.p im
[1] 1217      Hintergrund als Job [1] und Prozeß 1217)

%jobs
[1] + Running pc rx.p

%ps
PID   TT  STAT TIME COMMAND
1217 n09 I     0:00 pc rx.p
1218 n09 R     0:03 pc0 -o/tmp/p0001217 rx.p
(this last process was set up by the pascal compiler)
1220 n09 R     0:01 ps

%stop 1217   (Compilierungs-Prozeß gestoppt)
[1] +Stopped (signal) pc rx.p

%bg %1       (Compilierung über Jobnummer im Hinter-
no /          grund wieder gestartet)
[1] pc rx.p &

%fg %1       (Compilierung in den Vordergrund)
pc rx.p

Stopped      (Compilieren über Ctrl Z gestoppt)

%jobs
[1] +Stopped pc rx.p

%kill %1     (Endgültiger Abbruch des Prozesses)
[1] Exit 1 pc rx.p

%ps
PID   TT  STAT TIME COMMAND
1266 n09 R     0:01 ps
(außer ps selbst steht kein Prozeß mehr an)

%logout
```

# Duell mit Karten

**Herman Hollerith und James Powers leisteten entscheidende Beiträge zur Entwicklung der Lochkartentechnik.**

**James Powers Maschinen arbeiteten rein mechanisch und waren starr für eine bestimmte Anwendung konstruiert.**

**Herman Hollerith erfand den elektromechanischen Kartenleser, aus dem später die Tabelliermaschinen entwickelt wurden.**

Aus den Lochkartenmaschinen, die Herman Hollerith für die Auswertung der amerikanischen Volkszählung von 1890 konstruiert hatte, entstanden bald die „Tabelliermaschinen" als universelle Datenverarbeitungsgeräte. Bis zur Einführung der ersten Elektronenrechner in den fünfziger Jahren spielten die Tabelliermaschinen eine große Rolle für die zunehmende Rationalisierung in Handel und Industrie. In Pittsburgh/Pennsylvania (USA) experimentierte schon in den dreißiger Jahren ein führendes Kaufhaus mit einem Kundenkonten-System, bei dem 250 auf den Etagen verteilte Terminals über Telefonleitungen mit einer Tabellierer-Zentrale in Verbindung standen. Zur Warenauszeichnung dienten Lochetiketten, deren Daten an die Zentrale fernübertragen wurden, wo der Rechnungsbeleg erstellt wurde und über einen Fernschreiber die Freigabe an das Terminal erfolgte.

Den eigentlichen Anstoß zur Weiterentwicklung der Lochkartenmaschinen gab schon zwei Jahrzehnte vorher die Konkurrenzsituation: Mit Holleriths Monopol für die teuren Volkszählungsgeräte war es 1910 vorbei, als die zuständige Behörde James Powers als Zweitanbieter hinzuzog. Powers offerierte ein rein mechanisches System, das Holleriths Elektromechanik-Patente nicht verletzte. Der folgende Wettstreit zwischen beiden und ihren Firmen stellte für die Entwicklung der Datenverarbeitungstechnik einen Ansporn dar.

1902 entwarf Hollerith ein Steckbrett, mit dem man vorwählen konnte, welche Spalten der Lochkarten zusammengefaßt und als Summe ausgedruckt werden sollten. Dadurch boten Holleriths Maschinen eine gewisse Programmierbarkeit, die der Konkurrenz abging – die Geräte von Powers waren für eine bestimmte Anwendung konstruiert. Im Gegenzug ließ sich Powers 1924 ein System zur Darstellung alphanumerischer Information auf Lochkarten patentieren, das in jeder Spalte genau ein Loch zur Angabe einer Ziffer und eine Lochkombination für jeden Buchstaben verwendete. Hollerith reagierte darauf umgehend mit seinem 80-Spalten-Code, der heute noch Standard ist. Jede Spalte einer Karte enthält zwölf Stanzpositionen, die damals mit „Bürsten" aus Draht elektrisch abgetastet wurden: Wo die Karte gelocht war, bekam die Bürste Kontakt mit einer Metallplatte.

Die ersten Tabelliermaschinen konnten nur zählen oder addieren und beherrschten noch keine mathematischen Funktionen für die anspruchsvollere Verarbeitung. Das änderte sich bald infolge der technischen Fortschritte, und später wurden spezielle Tabelliermaschinen für die Analyse von Schwingungsvorgängen und astronomische Berechnungen eingesetzt.

## Lochkartenmaschinen

Auf dem Höhepunkt der Entwicklung zu Beginn der fünfziger Jahre gehörten zu einer Lochkartenanlage mindestens acht verschiedene Stationen. Das Ablochen der Daten erfolgte auf einem Kartenlocher mit Geschwindigkeiten bis zu 200 Karten pro Stunde. Für die Kontrolle der damaligen Datentypisten war eigens ein Prüflocher vorgesehen. Der Lochschriftübersetzer bedruckte die Karten am oberen Rand in Klarschrift, um die Handhabung zu erleichtern.

Die eigentliche Tabelliermaschine summierte spaltenweise die Informationen auf den Lochkarten (bis zu 9000 pro Stunde) und druckte die Ergebnisse in Tabellenform aus – daher der Name. Häufig war auch noch ein Rechenstanzer angeschlossen, der für kompliziertere Arithmetik wie Multiplikationen zuständig war. Der Mischer konnte zwei Lochkartenstapel inhaltlich vergleichen und auch mischen, und der Sortierer machte aus einem Lochkartenstapel bis zu 13 getrennte Haufen – einen für jedes der zwölf Löcher einer bestimmten Spalte und noch einen für ungelocht.

Die Programmierung der Tabelliermaschinen erfolgte über Steuerzeichen auf den Stanzpositionen 11 und 12, wobei die Steuerkarten zum besseren Auffinden im Stapel leuchtend bunt gefärbt waren. Stieß die Tabelliermaschine auf so eine Karte, bedeutete das eine neue Verarbeitungsvorschrift.

# Ereignisreich

**In der letzten Folge hatten wir uns angesehen, wie das Schneider-Betriebssystem mit Interrupts arbeitet. Dabei hatten wir auch „Ereignisse" untersucht, die „Software Interrupts" darstellen. In diesem Artikel zeigen wir, wie sich Ereignisse in der Maschinencodeprogrammierung einsetzen lassen.**

Das Schneider-Betriebssystem steuert Software-Ereignisse über einen „Ereignisblock". Dieser Block besteht aus sieben aufeinanderfolgenden Bytes, die sich mit der Routine KL_INIT_EVENT (bei $BCEF) an jeder beliebigen Stelle der mittleren 32K RAM anlegen lassen. Beim Aufruf der Routine muß das Registerpaar HL die Blockadresse enthalten, das Register B die Ereignisklasse (siehe Kasten), C die ROM-Wahl (0 bedeutet RAM) und DE die Adresse der Ereignisroutine. Unser Programmbeispiel zeigt diesen Ablauf.

Das Bild zeigt die Bitstruktur der im Register B gespeicherten Ereignisklasse. Der exakte Zeitpunkt, an dem ein Ereignis vom OS aufgerufen wird, hängt von Typ (synchron/asynchron) und der Priorität (normal/express) eines Ereignisses ab.

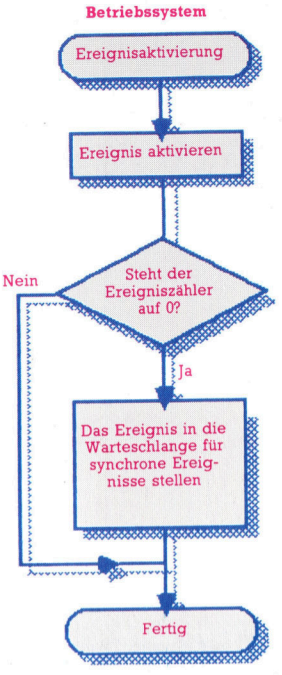
Das Betriebssystem behandelt synchrone und asynchrone Ereignisse unterschiedlich. Beide Ereignistypen lassen sich jedoch für jede Art von Ereignis verwenden.

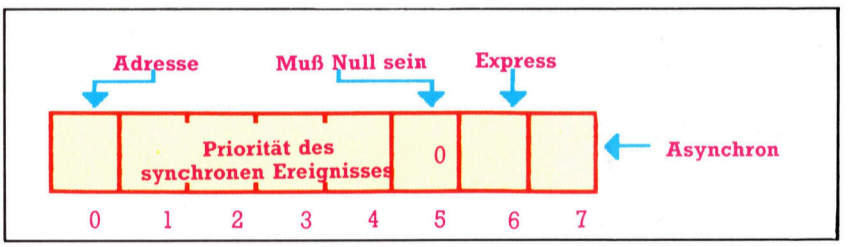
Der Ereigniszähler kann durch einen von vier Faktoren inkrementiert werden: vom Schnell-Interrupt, vom Ticker-Interrupt, vom Interrupt zur Bildauffrischung und von dem Jumpblock-Eintrag KL_EVENT. Den drei Timer-Interrupts sind Listen zugeordnet, in denen steht, welche Ereignisse beim Eintreten von Interrupts aktiviert werden. Die Routinen zur Anlage dieser Listen werden über die Jumpblock-Einträge aktiviert. Sie können entweder durch Initialisierung eines neuen Ereignisblocks Daten an die Liste anfügen oder einfach einen bereits bestehenden Block anhängen. Ereignisblöcke werden einzeln mit KL_INIT_EVENT initialisiert. Durch KL_EVENT lassen sich Ereignisse auslösen.

## Asynchrone Ereignisse

„Normale" asynchrone Ereignisse werden zunächst in eine eigene Warteschlange gestellt. Darin sind alle asynchronen Ereignisse gespeichert, die während eines externen Interrupts ausgelöst wurden.

Nehmen wir als Beispiel einen normalen asynchronen Ereignisblock, der vom Ticker-Interrupt aktiviert wird. Beim Auftreten eines Interrupts stellt das Betriebssystem zunächst fest, welche Ereignisse überhaupt ausgelöst werden müssen (in diesem Fall unser Ereignis) und prüft, ob der entsprechende Ereigniszähler über Null steht. Danach wird jedoch das Ereignis nicht gleich ausgelöst, sondern erst in die Warteschlange gestellt. Nach Ablauf aller OS-Aufgaben, die der Ticker-Interrupt normalerweise auslöst, werden die Ereignisroutinen nacheinander aus der Warteschlange abgerufen. Da zuvor der Ticker-Interrupt wieder eingeschaltet wurde, werden auch alle später eintretenden Ereignisaktivierungen gespeichert. Die Routine kann daher beliebig lang sein – es gehen keine Ereignisse verloren. Schließlich löst der Aufruf einer Ereignisroutine auch die Dekrementierung ihres Zählers aus.

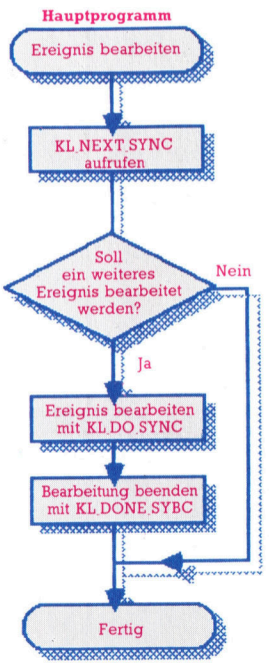



Asynchrone Ereignisse vom Typ „Express" (deren Zähler nach der Aktivierung über Null stehen) werden nicht erst in die Warteschlange gestellt, sondern sofort über ihre Ereignisroutinen aktiviert. Da dabei jedoch alle weiteren Interrupts abgestellt werden, können bei längeren Routinen externe Interruptanforderungen verlorengehen. Sie sollten diese Routine daher so kurz wie möglich halten – oder überhaupt nicht verwenden.

Bei synchronen Ereignissen entscheidet das Hauptprogramm den Zeitpunkt der Verarbeitung. Das Betriebssystem aktiviert daher das Ereignis nur und stellt es (bei einem Zähler über Null) je nach Priorität (siehe Ereignisblock) in die Warteschlange. Mit „Express" gekennzeichnete Ereignisse haben dabei Vorrang vor normalen Ereignistypen.

Zur Bearbeitung von Ereignissen führt ein Hauptprogramm bestimmte Schritte aus (siehe Flußdiagramm). Zunächst wird der Jumpblock-Eintrag KL_NEXT_SYNC aufgerufen, um festzustellen, welches Ereignis als nächstes in der Warteschlange steht. Das Ereignis wird dann über den Aufruf von KL_DO_SYNC ausgeführt. Dafür holt sich diese Routine die Adresse der




**Die beiden Flußdiagramme zeigen die Reihenfolge der Abläufe, die das Betriebssystem (oben) und das Hauptprogramm (unten) bei der Bearbeitung von synchronen Ereignissen ausführen.**

Ereignisroutine aus dem Ereignisblock und übergibt ihr die Steuerung. Nach Ablauf der Routine zeigt der Aufruf von KL_DONE_SYNC dem Betriebssystem an, daß die Bearbeitung beendet wurde. Wenn der Ereigniszähler nach der Dekrementierung über Null steht, wird der Ereignisblock wieder in die Liste der synchronen Ereignisse gestellt.

## Jumpblöcke

In manchen Situationen muß es möglich sein, das Eintreten bestimmter Ereignisse zu verhindern. Hierfür bietet das Betriebssystem eine ganze Reihe von Jumpblock-Einträgen. So setzt KL_DISARM_EVENT den Zähler im Ereignisblock eines asynchronen Ereignisses auf einen Minuswert und verhindert so, daß diese Routine weiterhin aktiviert wird.

Zum Abschalten synchroner Ereignisse stehen drei Einträge zur Verfügung. KL_SYNC_RESET löscht alle fälligen Ereignisse aus der synchronen Warteschlange, ändert aber deren Zähler nicht. Das Ereignis ist abgeschaltet, da der Ereigniszähler nur dekrementiert werden kann, wenn sich das Ereignis in der Warteschlange befindet. KL_DEL_SYNCHRONOUS löscht alle Ereignisse einer Warteschlange. KL_EVENT_DISABLE verhindert, daß normale synchrone Ereignisse in die Warteschlange gestellt werden, läßt die Aktivierung von Ereignissen aber weiterhin zu.

Obwohl Ereignisse auf den ersten Blick sehr kompliziert erscheinen, können Sie den Programmierer vor Problemen schützen, die bei Interrupts auftreten. Sie müssen jedoch genau darauf achten, daß die Daten der asynchronen Ereignisse vom Typ „Express" nicht mit dem Hauptprogramm kollidieren, sonst kann es zu folgenschweren Fehlern kommen.

## Hintergrund-Piepser

**Unser Programmbeispiel zeigt, wie mit den im Artikel beschriebenen Techniken Ereignisse angewendet werden. Dabei wird ein Ticker-Ereignis als Sekundenzähler angelegt, der wiederum veranlaßt, daß der Computer alle Sekunde ein „Beep" hervorbringt. Der Aufruf des Programms über seine Assembleradresse initialisiert die Ereignisse.**

**Das Piepen setzt sich auch bei einem laufenden BASIC-Programm im Hintergrund fort. Wenn das BASIC-Programm ebenfalls den Tongenerator anspricht, können seltsame Wirkungen entstehen.**

```
; This program demonstrates the use of several
; types of event to generate a simple clock that
; sounds a beeper every second.
;
INIT.EVENT:     EQU  £BCEF ; KL_INIT_EVENT
ADD.TICK:       EQU  £BCE9 ; KL_ADD_TICKER
KICK:           EQU  £BCF2 ; KL_EVENT
TXT.OUT         EQU  £BB5A ; TXT_OUTPUT
;
BLEEP:           EQU  £07
;
; First initialise the event blocks
;
                ORG  £8000
;
                LD   HL, TICK   ; start with ticker
                LD   B, £81     ; async, express
                LD   C, 0       ; no rom select
                LD   DE, FRAC   ; routine address
;
                CALL INIT.EVENT
;
; BC is preserved
;
                LD   HL, SEC    ; seconds counter
                LD   DE, SECND  ; routine address
                CALL INIT.EVENT
;
                LD   HL, BP     ; bleeper routine
                LD   DE, BEEP   ; routine address
                CALL INIT.EVENT
;
; now log on the Ticker Routine
;
                LD   HL, FRBLK  ; tick block address
                LD   DE, 1      ; count
                LD   BC, 1      ; recharge
                CALL ADD.TICK   ; log it in
;
                RET
;
;Event Processing Routines
;
FRAC:
; maintains a 1/50th of a second count
;
                LD   HL, SEC50  ; point to count
                LD   B, 50      ; one second?
                CALL TEST
                RET  NZ         ; no, then finish
                LD   HL, SEC    ; kick next second
                CALL KICK
                RET
;
SECND:
; This is called once a second - kicks a beep
                LD  HL, BP      ; kick beep event
                CALL KICK
                RET
;
BEEP:
; does a beep
                LD   A, BLEEP   ; send a bleep char
                CALL TXT.OUT
                RET
;
TEST:
; tests a counter
; B contains the required value on entry
; HL points to the count store location
;
                INC  (HL)       ; update it
                LD   A, (HL)    ; read the count
                CP   B          ; finished
;
                RET  NZ         ; no, return
                XOR  A          ; set zero true
                LD   (HL), A    ; reset the count
                RET             ; and go back
;
; now the event and tick blocks
;
FRBLK:          DEFS 6          ; tick block space
;
TICK:           DEFS 7          ; ticker event block
;
SEC:            DEFS 7          ; 1 sec event block
;
BP:             DEFS 7          ; beep event block
;
SEC50:          DEFB 0          ; 1/50 sec counter
```

# Unter der Haube

**Unsere Hardware-Serie wird mit der Betrachtung von drei Heimcomputern abgeschlossen. Dabei zeigen sich individuelle Unterschiede in der Architektur. Anhand vereinfachter Logik-Diagramme wird außerdem gezeigt, wie das Innenleben der meistbenutzten Gatterbausteine aussieht.**

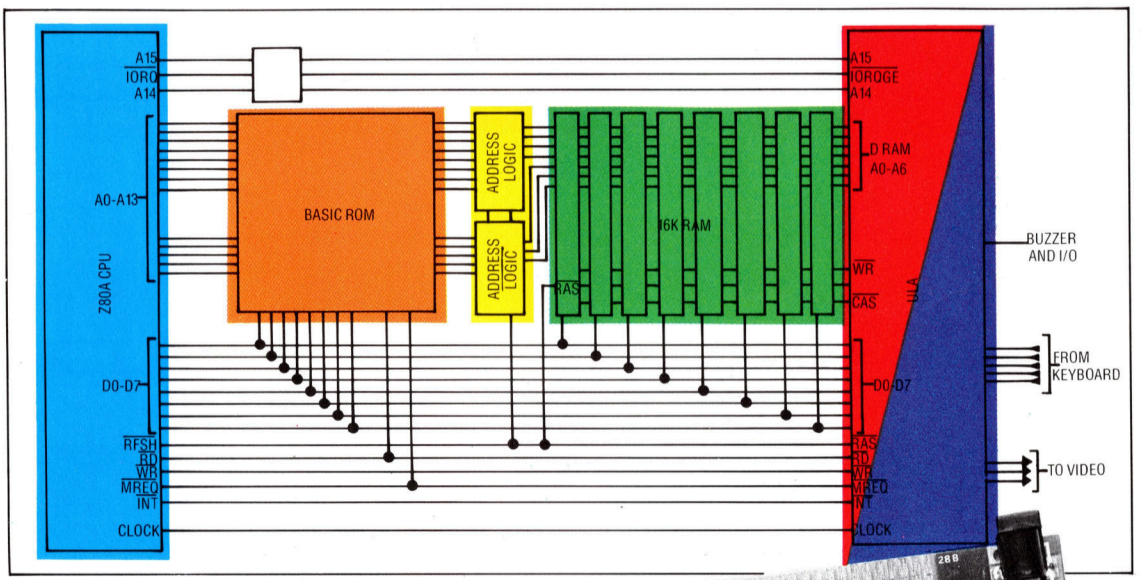
## Sinclair Spectrum

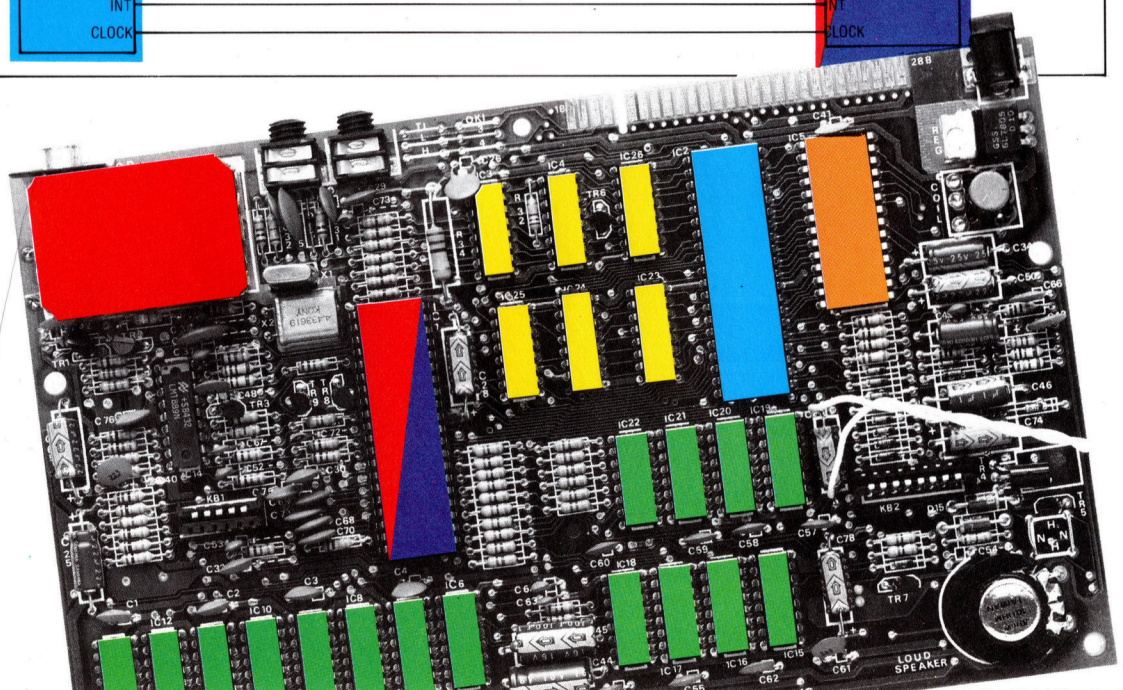
Der ZX Spectrum weicht in seiner Architektur erheblich von den anderen hier besprochenen Micros ab. Als CPU dient ein Z80A; für die meisten Systemfunktionen ist ein ULA (Uncommitted Logic Array) zuständig. Dieser maßgeschneiderte Chip steuert die Videosignale, die Speicherauffrischung und die elementare Ein/Ausgabe über Tastatur und Cassettenrecorder sowie den Minilautsprecher. Das untenstehende Blockdiagramm gilt für die ältere 16-KByte-Version, die späteren 48K-Modelle enthalten zusätzliche RAM-Bausteine.

Die Bildschirmwiedergabe erfordert in regelmäßigen Abständen schnelle Speicherzugriffe. Die ersten 16 K RAM mit der Videoinformation haben beim 48K-Spectrum direkte Verbindung mit dem ULA-Baustein, der deshalb unter normalen Umständen das Video-RAM unabhängig von der CPU ansprechen kann. Diese kann gleichzeitig mit dem BASIC-ROM oder den 32 K Zusatz-RAM kommunizieren. Kritisch wird es, wenn auch die CPU auf den 16K-Block mit dem Video-RAM zugreifen möchte, weil die dort ebenfalls gespeicherten BASIC-Systemvariablen benötigt werden.

**Farbschlüssel**

- Prozessor
- Dynamisches RAM
- ROM
- Ein/Ausgabe
- Videosignal-Erzeugung
- System- bzw. Adreßlogik



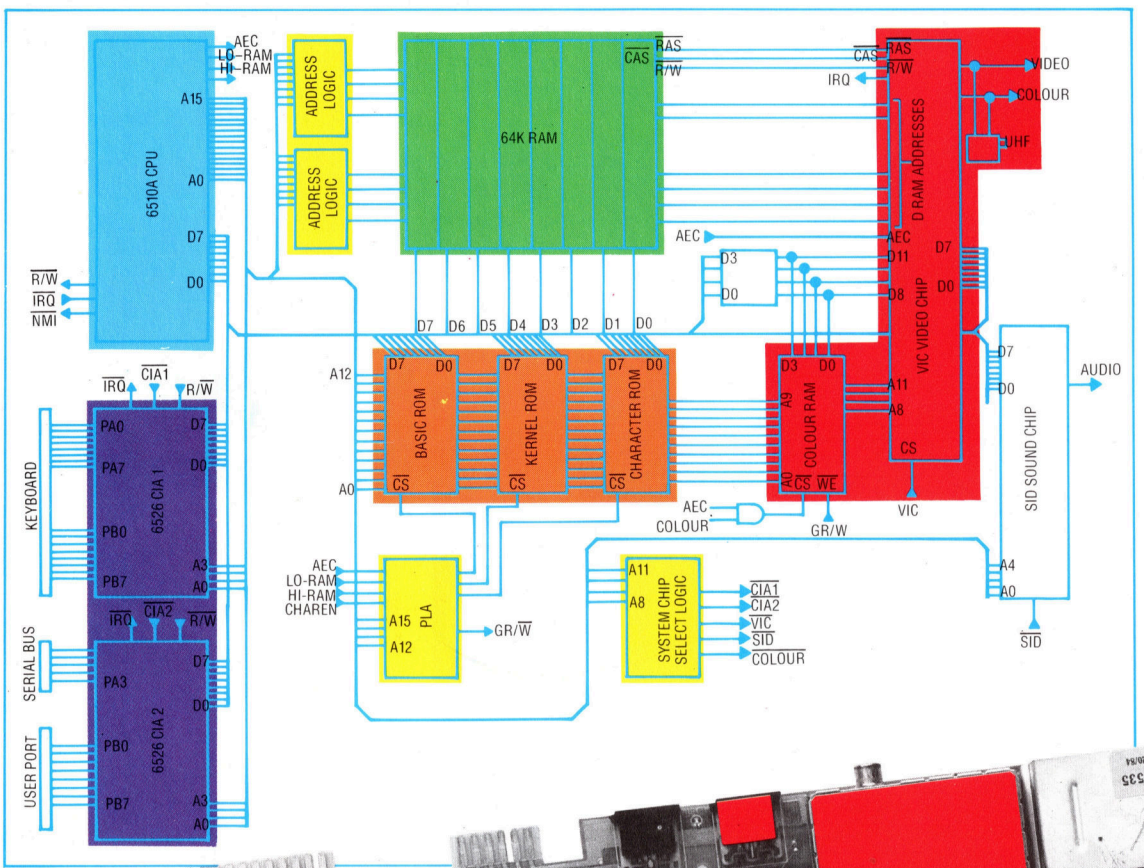



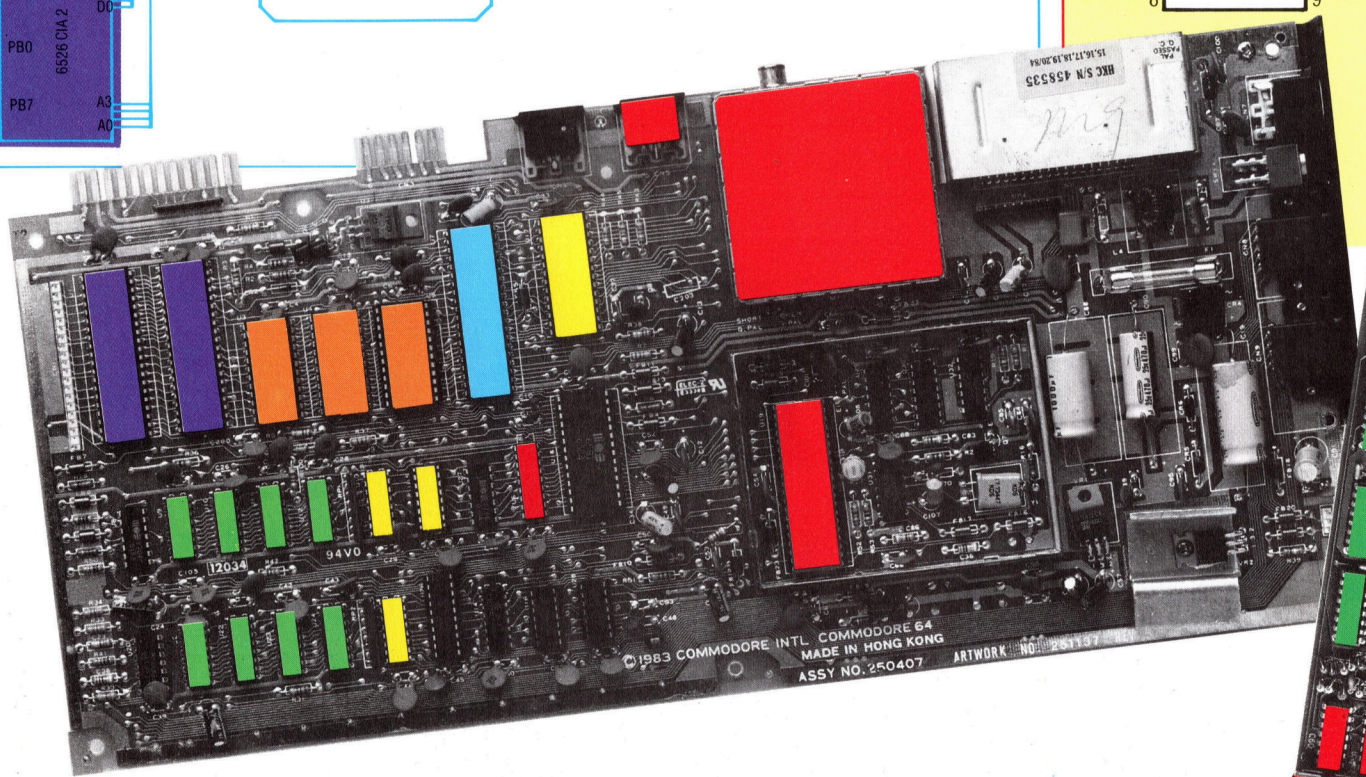
## Commodore 64

Der C 64 ist recht gut ausgestattet, mit 64 KByte RAM (dynamisch) in acht 64 KBit-Chips sowie mit drei 8 KByte-ROMs, die den BASIC-Interpreter, den Systemkern (Kernel) und den Zeichensatz (Character ROM) enthalten. Die CPU 6510 ist eine verbesserte 6502-Version, die das Einblenden der ROMs in den Adreßraum des Prozessors über ein integriertes Zusatzregister und einen speziellen PLA-Chip (Programmed Logic Array) ermöglicht.

Der C 64 hat zwei Schnittstellenbausteine (PIO = Peripheral Input/Output), einen davon für die Tastatur. Beim zweiten Chip wird ein Port für den seriellen Commodore-Bus verwendet, an dem Drucker, Diskettenlaufwerke und andere Peripheriegeräte hängen, der andere steht als User Port zur Verfügung.

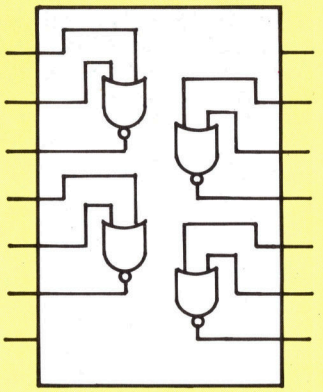
Der Video-Chip (VIC = Video Interface Controller) holt sich die Bildschirminformation direkt aus dem dynamischen RAM. Auch hier muß der Videochip die CPU anhalten, wenn er auf das RAM zugreifen will.



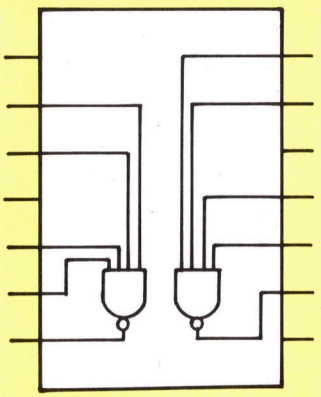




## Ganz logisch

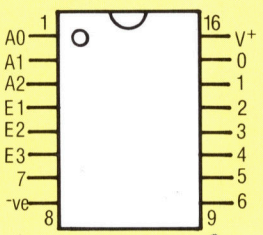
74LS00 (Vierfach-NAND mit je zwei Eingängen)



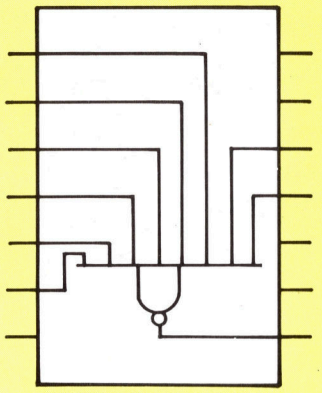
74LS20 (Zweifach-NAND mit je vier Eingängen)



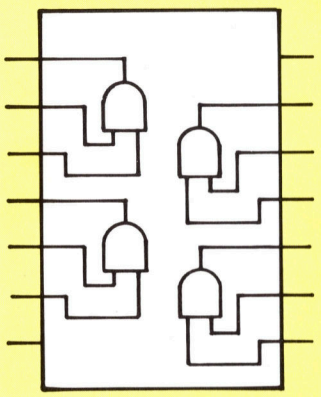
74LS138 (3-Bit-Decoder)



74LS30 (Einfach-NAND mit acht Eingängen)



74LS02 (Vierfach-NOR mit je zwei Eingängen)

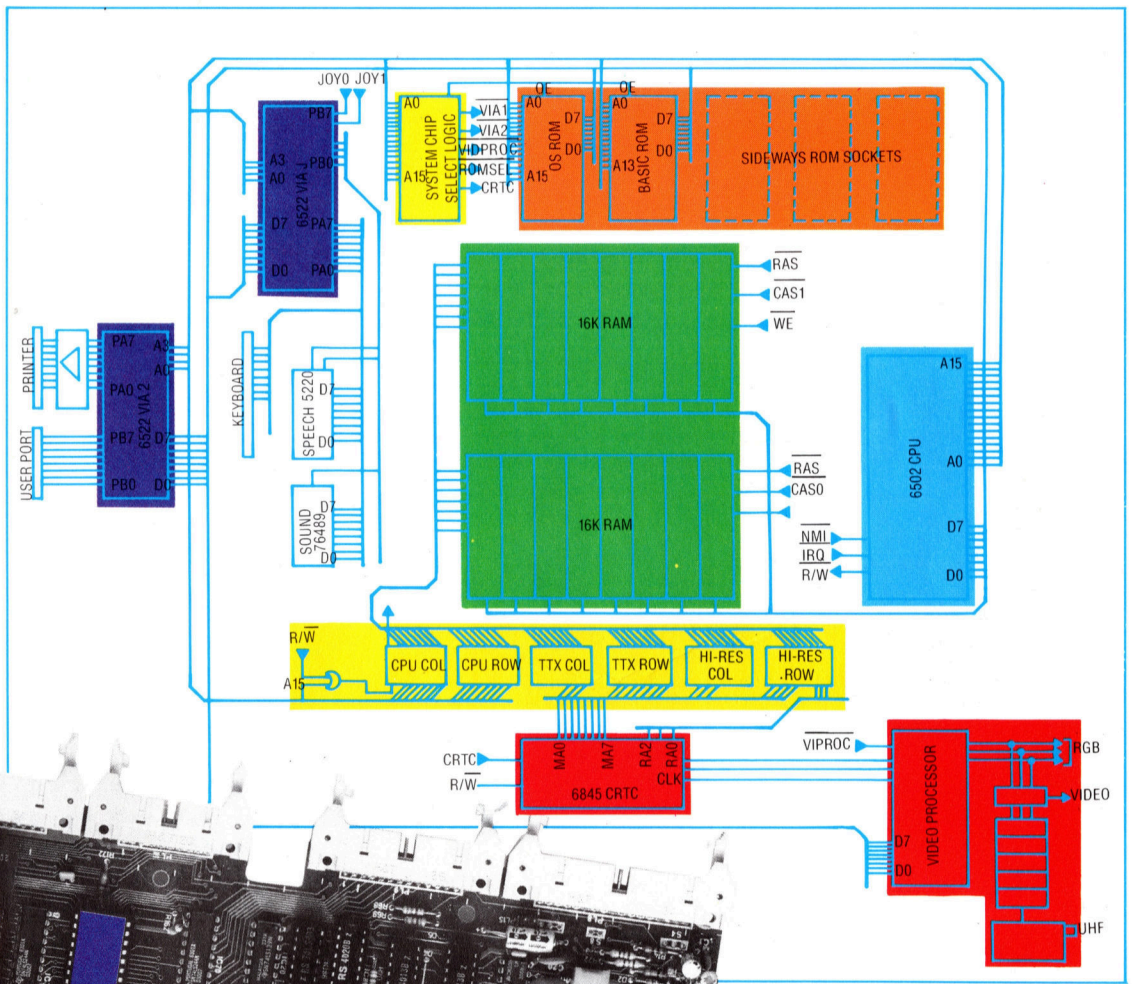


**Von den kleineren Bausteinen auf einer Microcomputer-Platine stammen viele aus der 74er-Serie. Sie enthalten diverse Gatterkombinationen und dienen zur Verknüpfung oder Decodierung von Adreß- und Steuersignalen.**

## Acorn B

Der Acorn B bietet vielseitige Möglichkeiten. Dennoch ist der Platine anzusehen, daß sie älteren Datums ist: Für 32 KByte RAM sind hier 16 dynamische RAM-Chips mit je 16 KBit nötig, während der Commodore 64 mit nur acht Bausteinen auf 64 KByte kommt. Für den wechselweisen Zugriff der CPU und der Video- bzw. Teletext-Chips auf das RAM sind sechs Adreßlogik-Bausteine vorgesehen. Die Grundausstattung enthält ferner zwei 8-KByte-ROMs und drei Leersockel für weitere ROMs.

Die Platine hat außerdem zwei PIOs und einen seriellen Schnittstellenbaustein. Ein PIO wird vom System für die Tastatur, die Klang- und Sprachsysthese-Bausteine, die Feuerknöpfe der Joysticks sowie für einige Videofunktionen wie das „Rollen" des Bildschirms (Scrolling) verwendet. Über den Port A des System-PIOs ist eine Art langsamer Datenbus eingerichtet; die CPU verkehrt mit der Tastatur und dem Tongenerator nicht unmittelbar. Beim zweiten PIO-Chip ist ein Port für den Paralleldrucker zuständig, der andere dient als User Port.

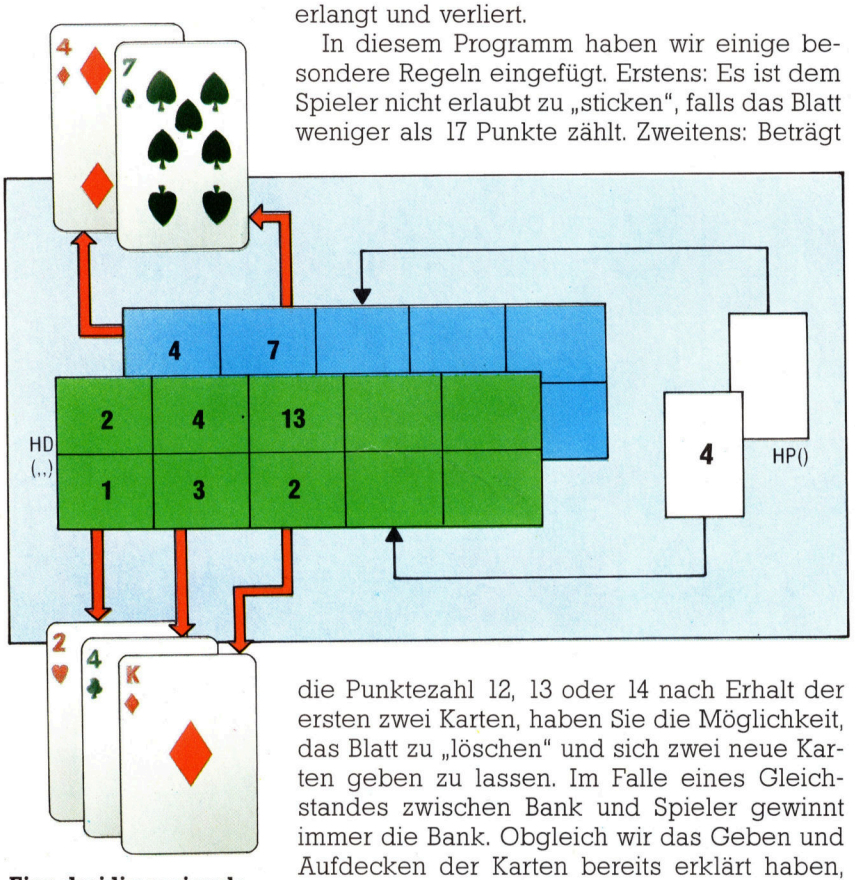


# Die Spielregeln

**Bis jetzt haben wir in unserem 17 + 4-Programmierprojekt die Routinen zum Aufdecken, Mischen und Geben der Karten zusammengestellt. Wir schauen uns jetzt an, wie das Blatt eines Spielers gespeichert und ausgewertet wird.**

In der einfachsten Form von 17 + 4 spielt ein Spieler gegen die Bank, die zwei Karten vom Stapel gibt: Eine aufgedeckt für den Spieler; die andere verdeckt für die Bank. Danach bekommen beide je eine aufgedeckte Karte. Nun muß der Spieler entscheiden, ob er „stick" spielt, d. h. 21 Punkte hat oder nahe genug daran ist, oder ob er „twist" spielt und eine weitere Karte nimmt und dabei mehr als 21 Punkte erlangt und verliert.

In diesem Programm haben wir einige besondere Regeln eingefügt. Erstens: Es ist dem Spieler nicht erlaubt zu „sticken", falls das Blatt weniger als 17 Punkte zählt. Zweitens: Beträgt die Punktezahl 12, 13 oder 14 nach Erhalt der ersten zwei Karten, haben Sie die Möglichkeit, das Blatt zu „löschen" und sich zwei neue Karten geben zu lassen. Im Falle eines Gleichstandes zwischen Bank und Spieler gewinnt immer die Bank. Obgleich wir das Geben und Aufdecken der Karten bereits erklärt haben, zeigten wir noch nicht, wie sich das Programm an die ausgegebenen Karten „erinnert". Um das Blatt des Spielers und das der Bank zu speichern, richten wir ein dreidimensionales Zahlenfeld HD(,,) ein. Da keine der Parteien mehr als fünf Karten erlangen kann, ist diese Feldvariable in Zeile 550 auf zwei Elemente (für die beiden Spieler) mal fünf (für die Karten) mal zwei (zum getrennten Speichern der Zahlen und Farben) dimensioniert. Als einen Zeiger auf das nächste freie Element der Kartenfelder beider Spieler benutzen wir ein zweites Array, HP(). In diesem Stadium müssen wir Zeile 1325 dem vorher entwickelten Kartenausgabeprogramm hinzufügen, so daß der Kartenwert, CN, und das Farbenzeichen, SU, in das Kartenfeld aufgenommen und der Zeiger auf das nächste Feld inkrementiert werden. Beachten Sie, daß die Variable PL bestimmt, welchem Blatt die ausgegebene Karte zugehören soll; ist ihr Wert gleich eins, bekommt der Spieler eine Karte, ist PL gleich zwei, wird die Karte dem Blatt der Bank hinzugefügt.




**Eine dreidimensionale Feldvariable HD(,,) speichert die Karten des Spielers und die der Bank, wie sie vom Stapel ausgegeben wurden. Die Feldvariable HP() dient als Zeiger auf die nächsten freien Elemente in den Kartenfeldern. Am Ende eines Spiels braucht das Programm die Kartenfeldelemente nicht einzeln zu löschen, es muß nur die Zeiger auf eins setzen.**

Die ersten vier Karten können jetzt ganz einfach ausgegeben werden, indem Sie in PL eine Eins oder Zwei eingeben und das Kartenausgabeprogramm aufrufen. Da die erste, an die Bank ausgegebene Karte verdeckt sein muß, benutzt das Kartenausgabeprogramm die Variable FL als Zeiger. Das Programm verzweigt in die Routine zum Darstellen einer Kartenrückseite, wenn FL den Wert Eins enthält. Die Daten (Farbe und Wert) der gezogenen Karte werden in das Kartenfeld übernommen.

Das Unterprogramm in Zeile 800 wurde als ein allgemein verwendbares Auswertungsprogramm entwickelt, das zwei Funktionen erfüllt. Es errechnet erstens den totalen Wert des überprüften Blattes und legt zweitens den Wert der Variablen EF zwischen eins und fünf, entsprechend einem von fünf möglichen Blatt-Typen, fest. Die dem Blatt entsprechende Kategorie ist relativ einfach zu ermitteln. Schwieriger hingegen ist die totale Punktzahl eines Blattes zu errechnen. Dabei sieht es auf den ersten Blick so aus, als hätte man nur die Zahlen auf den Karten eines Blattes zu addieren, plus der Werte der Bilderkarten (je 10), plus der Asse (1 oder 11). Das erste Problem ist schnell gelöst, indem wir den Kartenwert überprüfen. Ist er höher als zehn, addieren wir zehn zu dem Gesamtergebnis.

## Jetzt die Asse

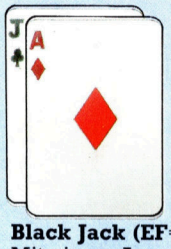
Das erste As des Blattes wird mir 1 oder 11 bewertet, alle nachfolgenden mit eins, oder das Blatt wird gesprengt.

Um dieser Regel zu entsprechen, arbeiten wir mit zwei Punktezählern: TT(PL,1), in dem das erste As einen Punkt zählt, und TT(PL,2) in dem das erste As elf Punkte zählt. In einer Schleife addieren wir die Punkte eines Blattes, wobei alle vorhandenen Asse einen Punkt zählen, und ermitteln dabei auch die Anzahl der

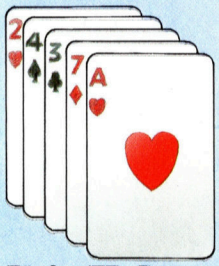
Asse. Bei Beendigung der Schleife überprüfen wir, ob ein oder mehrere Asse im Blatt vorhanden sind, und addieren dann 10 zu TT(PL,2). Der Rest des Bewertungsprogramms überprüft nun die Summen von TT(PL,1) und TT(PL,2) oder direkt das Kartenfeld, um den Typ des Blattes zu bestimmen, und setzt die Variable EF entsprechend.

## Auszählen

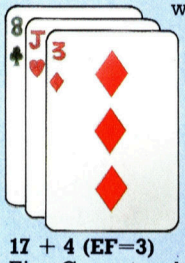
Es gibt fünf Kategorien, in denen alle möglichen Kartenkombinationen per Blatt zu erfassen sind. Die nachfolgende Übersicht erläutert die verschiedenen Blatt-Typen, die unser Programm in einer hierarchischen Ordnung erkennt.



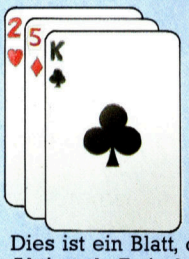
**Black Jack (EF=2)**
Mit einem As und einer Bildkarte erzielte 21 Punkte. Ein As und eine 10 gelten nicht als Black Jack.



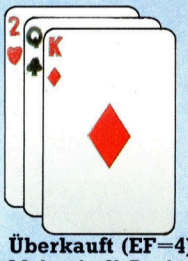
**Fünfer (EF=5)**
Fünf Karten mit zusammen 21 oder weniger Punkten.



**17 + 4 (EF=3)**
Eine Gesamtpunktzahl von genau 21



**Weniger als 21 (EF=1)**
Dies ist ein Blatt, das nach Ausführung einer Aktion als Zwischenergebnis dient und besagt, daß das Blatt gültig ist und die Punktzahl unter 21 liegt.



**Überkauft (EF=4)**
Mehr als 21 Punkte

## Auswertungs- und Berechnungsroutinen

**Sinclair Spectrum**

```
  60>LET FL=0:LET PL=1:GO SUB 1300:REM D
EAL CARD TO PUNTER
  70 LET FL=1: LET PL=2: GO SUB 1300: RE
M DEAL CARD TO BANK
  85 LET FL=0: LET PL=1: GO SUB 1300: RE
M DEAL CARD TO PUNTER
  90 LET FL=0: LET PL=2: GO SUB 1300: RE
M DEAL CARD TO BANK
  95 REM **** PUNTER'S TURN ****
 100 LET PL=1
 102 GO SUB 2300: REM BURN OPTION
```

**Kartenfeld dimensionieren**

```
 550>DIM D(2,5,2):DIM P(2):REM PUNTER'S
AND BANKER'S HANDS
 555 DIM T(2,2): REM SCORE TOTALS
 670>REM **** ERASE CARD DISPLAY ****
 675 LET X(PL)=EP: LET Y(PL)=0
 680 PRINT AT 0,0;: FOR I=1 TO 12: PRINT
 AT I,EP;S$( TO 7): NEXT I
 700 REM **** PREPARE FOR INPUT ****
 710 LET TX=0: LET TY=17: GO SUB 900: PR
INT S$( TO 31)
 720 LET TX=0: LET TY=17: GO SUB 900: RE
TURN
```

**Blattauswertungsroutine**

```
 800 REM **** EVALUATE HAND ****
 810 LET AV=1: FOR J=1 TO 2
 812 LET T(PL,J)=0
 815 FOR I=1 TO P(PL)-1
 820 IF D(PL,I,1)=1 THEN  LET T(PL,J)=T(
PL,J)+AV-1
 825 IF D(PL,I,1)>10 THEN  LET T(PL,J)=T
(PL,J)+10-D(PL,I,1)
 830 LET T(PL,J)=T(PL,J)+D(PL,I,1)
 840 NEXT I: LET AV=11: NEXT J
 852 IF (T(PL,1)<=21 OR T(PL,2)<=21) AND
 P(PL)>5 THEN  LET EF=5: RETURN
 854 IF D(PL,1,1)=1 AND D(PL,2,1)>10 THE
N  LET EF=2: RETURN
 855 IF D(PL,2,1)=1 AND D(PL,1,1)>10 THE
N  LET EF=2: RETURN
 856 IF T(PL,1)=21 OR T(PL,2)=21 THEN  L
ET EF=3: RETURN
 858 IF T(PL,1)<21 OR T(PL,2)<21 THEN  L
ET EF=1: RETURN
 860 IF T(PL,1)>21 AND T(PL,2)>21 THEN
LET EF=4: RETURN
2300>REM **** BURN OPTION ****
2305 GO SUB 800: REM EVALUATE
2310 IF T(PL,1)<>T(PL,2) OR T(PL,1)<12 O
R T(PL,1)>14 THEN  RETURN
2340 GO SUB 700: PRINT "BURN (Y/N) ";
2345 LET A$=INKEY$: IF A$="" THEN  GO TO
 2345
2347 IF A$<>CHR$ 13 THEN  PRINT A$
2350 IF A$<>"Y" THEN  RETURN
2360 LET P(1)=1: REM RESET HAND POINTER
2370 LET EP=0: GO SUB 670: REM ERASE CAR
DS
2380 LET FL=0: LET PL=1: GO SUB 1300: GO
 SUB 800: REM DEAL CARD TO PUNTER
2390 LET FL=0: LET PL=1: GO SUB 1300: GO
 SUB 800: REM DEAL CARD TO PUNTER
2400 GO TO 2300: REM BURN AGAIN?
```

**Acorn B**

```
  60 FL=0:PL=1:GOSUB 1300
  70 FL=1:PL=2:GOSUB 1300
  85 FL=0:PL=1:GOSUB 1300
  90 FL=0:PL=2:GOSUB 1300
  95 REM
 100 PL=1
 102 GOSUB 2300
```

**Kartenfeld dimensionieren**

```
 550 DIM HD(2,5,2),HP(2)
 555 DIM TT(2,2)
 670 REM ****  ERASE CARD DISPLAY ****
 675 X(PL)=EP:Y(PL)=0
 680 PRINT TAB(0,0);:FOR I=1 TO 12:PRIN
 T TAB(EP,I);LEFT$(SP$,19):NEXT I:RETURN
```

```
  700 REM
  710 COLOUR 1:TX=0:TY=23:GOSUB 900:PRIN
T SP$
  720 TX=0:TY=23:GOSUB 900:RETURN
```

**Blattauswertungsroutine**

```
  800 REM
  810 AC=0:AV=0:TT(PL,1)=0
  815 FOR I=1 TO HP(PL)-1
  820 IF HD(PL,I,1)=1 THEN AC=AC+1
  825 IF HD(PL,I,1)>10 THEN TT(PL,1)=TT(
PL,1)+10-HD(PL,I,1)
  830 TT(PL,1)=TT(PL,1)+HD(PL,I,1)
  840 NEXT I
  845 IF AC>0 THEN AV=10
  850 TT(PL,2)=TT(PL,1)+AV
  852 IF (TT(PL,1)<=21 OR TT(PL,2)<=21)
AND HP(PL)>5 THEN EF=5:RETURN
  854 IF HD(PL,1,1)=1 AND HD(PL,2,1)>10
THEN EF=2:RETURN
  855 IF HD(PL,2,1)=1 AND HD(PL,1,1)>10
THEN EF=2:RETURN
  856 IF TT(PL,1)=21 OR TT(PL,2)=21 THEN
 EF=3:RETURN
  858 IF TT(PL,1)<21 OR TT(PL,2)<21 THEN
 EF=1:RETURN
  860 IF TT(PL,1)>21 AND TT(PL,2)>21 THE
N EF=4:RETURN
 2300 REM
 2305 GOSUB 800
 2310 IF TT(PL,1)<>TT(PL,2) OR TT(PL,1)<
12 OR TT(PL,1)>14 THEN RETURN
 2340 GOSUB 700:PRINT"BURN (Y/N) ";
 2345 AN$=GET$
 2347 IF AN$<>CHR$(13) THEN PRINT AN$
 2350 IF AN$<>"Y" THEN RETURN
 2360 HP(1)=1
 2370 EP=0:GOSUB 670
 2380 FL=0:PL=1:GOSUB 1300:GOSUB 800
 2390 FL=0:PL=1:GOSUB 1300:GOSUB 800
 2400 GOTO 2300
```

### Schneider CPC

```
  60 fl=0:pl=1:GOSUB 1300:REM deal card to
   punter
  70 fl=1:pl=2:GOSUB 1300:REM deal card to
   bank
  85 fl=0:pl=1:GOSUB 1300:REM deal card to
   punter
  90 fl=0:pl=2:GOSUB 1300:REM deal card to
   bank
  95 REM **** punter's turn ****
 100 pl=1
 102 GOSUB 2300:REM burn option
```

**Kartenfeld dimensionieren**

```
 550 DIM hd(2,5,2),hp(2):REM punter and b
anker hands
 555 DIM tt(2,2):REM score totals
 670 REM **** erase card ****
 675 x(pl)=ep:y(pl)=0:tx=ep:ty=0
 680 FOR i=1 TO 12:GOSUB 900:PRINT SPACE$
(19)
 690 ty=ty+1:NEXT i:RETURN
 700 REM **** prepare for input ***
 710 PEN white:tx=0:ty=23:GOSUB 900:PRINT
  SPACE$(39)
 720 tx=0:ty=23:GOSUB 900:RETURN
```

**Blattauswertungsroutine**

```
 800 REM **** evaluate hand ****
 810 ac=0:av=0:tt(pl,1)=0
 815 FOR i=1 TO hp(pl)-1
 820 IF hd(pl,i,1)=1 THEN ac=ac+1
 825 IF hd(pl,i,1)>10 THEN tt(pl,1)=tt(pl
,1)+10-hd(pl,i,1)
 830 tt(pl,1)=tt(pl,1)+hd(pl,i,1)
 840 NEXT i
 845 IF ac>0 THEN av=10
 850 tt(pl,2)=tt(pl,1)+av
 852 IF (tt(pl,1)<=21 OR tt(pl,2)<=21) AN
D hp(pl)>5 THEN ef=5:RETURN
 854 IF hd(pl,1,1)=1 AND hd(pl,2,1)>10 TH
EN ef=2:RETURN
 855 IF hd(pl,2,1)=1 AND hd(pl,1,1)>10 TH
EN ef=2:RETURN
 856 IF tt(pl,1)=21 OR tt(pl,2)=21 THEN e
f=3:RETURN
 858 IF tt(pl,1)<21 OR tt(pl,2)<21 THEN e
f=1:RETURN
 860 IF tt(pl,1)>21 AND tt(pl,2)>21 THEN
ef=4:RETURN
2300 REM **** burn option ****
2305 GOSUB 800:REM evaluate
2310 IF tt(pl,1)<>tt(pl,2) OR tt(pl,1)<1
2 OR tt(pl,2)>14 THEN RETURN
2340 GOSUB 700:PRINT "Burn (y/n)";
2345 an$="":WHILE an$="":an$=INKEY$:WEND
2347 IF an$<>CHR$(13) THEN PRINT an$
2350 IF an$<>"y" THEN RETURN
2360 hp(pl)=1:REM reset hand pointer
2370 ep=0:GOSUB 670:REM erase cards
2380 fl=0:pl=1:GOSUB 1300:GOSUB 800:REM
deal card to punter
2390 fl=0:pl=1:GOSUB 1300:GOSUB 800:REM
deal card to punter
2400 GOTO 2300:REM burn again ?
```

### Commodore 64

```
  60 FL=0:PL=1:GOSUB 1300:REM DEAL CARD TO
   PUNTER
  70 FL=1:PL=2:GOSUB 1300:REM DEAL CARD TO
   BANK
  85 FL=0:PL=1:GOSUB 1300:REM DEAL CARD TO
   PUNTER
  90 FL=0:PL=2:GOSUB 1300:REM DEAL CARD TO
   BANK
  95 REM **** PUNTER'S TURN ****
 100 PL=1
 102 GOSUB 2300:REM BURN OPTION
 120 GOSUB 2600:REM TWIST ETC
 550 DIM HD(2,5,2),HP(2):REM PUNTER'S AND
  BANKER'S HANDS
 555 DIM TT(2,2):REM SCORE TOTALS
```

**Kartenfeld dimensionieren**

```
 670 REM **** ERASE CARD DISPLAY ****
 675 X(PL)=EP:Y(PL)=0
 680 PRINT CHR$(19);:FORI=1 TO 12:PRINTTA
B(EP);LEFT$(SP$,19):NEXT I:RETURN
 700 REM **** PREPARE FOR INPUT ****
 710 PRINT CHR$(5);:TX=0:TY=23:GOSUB900:P
RINTSP$
 720 TX=0:TY=23:GOSUB900:RETURN
```

**Blattauswertungsroutine**

```
 800 REM **** EVALUATE HAND ****
 810 AC=0:AV=0:TT(PL,1)=0
 815 FOR I=1 TO HP(PL)-1
 820 IF HD(PL,I,1)=1 THEN AC=AC+1
 825 IF HD(PL,I,1)>10 THEN TT(PL,1)=TT(PL
,1)+10-HD(PL,I,1)
 830 TT(PL,1)=TT(PL,1)+HD(PL,I,1)
 840 NEXT I
 845 IF AC>0 THEN AV=10
 850 TT(PL,2)=TT(PL,1)+AV
 852 IF (TT(PL,1)<=21 OR TT(PL,2)<=21)AND
  HP(PL)>5 THEN EF=5:RETURN
 854 IF HD(PL,1,1)=1 AND HD(PL,2,1)>10 TH
EN EF=2:RETURN
 855 IF HD(PL,2,1)=1 AND HD(PL,1,1)>10 TH
EN EF=2:RETURN
 856 IF TT(PL,1)=21 OR TT(PL,2)=21 THEN E
F=3:RETURN
 858 IF TT(PL,1)<21 OR TT(PL,2)<21 THEN E
F=1:RETURN
 860 IF TT(PL,1)>21 AND TT(PL,2)>21 THEN
EF=4:RETURN
2300 REM **** BURN OPTION ****
2305 GOSUB800:REM EVALUATE
2310 IF TT(PL,1)<>TT(PL,2) OR TT(PL,1)<1
2 OR TT(PL,1)>14 THEN RETURN
2340 GOSUB700:PRINT"BURN (Y/N) ";
2345 GET AN$:IF AN$=""THEN 2345
2347 IF AN$<>CHR$(13)THEN PRINTAN$
2350 IF AN$<>"Y" THEN RETURN
2360 HP(1)=1:REM RESET HAND POINTER
2370 EP=0:GOSUB670:REM ERASE CARDS
2380 FL=0:PL=1:GOSUB 1300:GOSUB800:REM D
EAL CARD TO PUNTER
2390 FL=0:PL=1:GOSUB 1300:GOSUB800:REM D
EAL CARD TO PUNTER
2400 GOTO 2300:REM BURN AGAIN?
```

# Kleinformat

**Der Speicherplatz von Heimcomputern reicht nicht für alle Anwendungen. Die Kompression von Texten kann sich oft lohnen. In diesem Abschnitt wollen wir Ihnen drei Methoden zum „Zusammendrücken" von Files vorstellen.**

Komprimierungstechniken lassen sich für eine Vielzahl unterschiedlicher Zwecke einsetzen. Wichtig sind sie speziell zur Programmierung umfangreicher Adventures für den Heimcomputer. Im kommerziellen Einsatz ist es die Ersparnis an Zeit und Telefongebühren, die den Anwender zur Kompression von Texten zwingt; die Datenkommunikation wird billiger. Aber auch die Minimierung des Platzbedarfs bei der Speicherung auf Disketten ist wichtig. Bei größeren Textfiles lassen sich Kompressionsraten bis zu 60 Prozent erzielen.

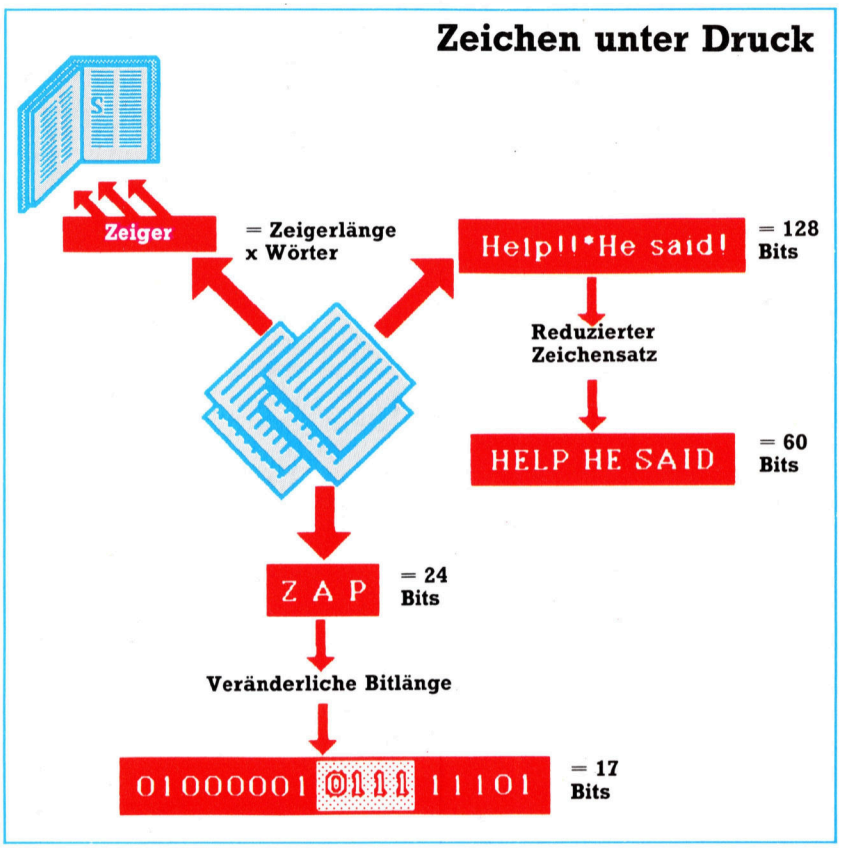
Für die Textkomprimierung gibt es drei Grundverfahren. Wir werden sie hier getrennt beschreiben, in der Praxis werden sie jedoch oft gemeinsam angewandt. Die erste und einfachste Form ist die Verwendung eines reduzierten Zeichensatzes. Bei den meisten Rechnern wird je ein Zeichen durch ein Byte dargestellt, weil so die Verarbeitung durch den Rechner am einfachsten ist. Bei acht Bits pro Byte gibt es 256 mögliche Zeichen. Zur Darstellung von Groß- und Kleinbuchstaben, Zahlen sowie der wichtigsten Interpunktionszeichen kommt man aber mit 96 Zeichen aus.

95 Zeichen lassen sich bereits mit sieben Bit codieren; damit wäre bei der Speicherung allerdings erst ein Platzgewinn von 12,5 Prozent (ein Achtel) zu erreichen. Man kann noch mehr Zeichen weglassen: +,*,< und > werden selten in Texten benutzt; auf sie kann verzichtet werden.

## Symbole

Aber auch das spart nicht genug, wenn man sich nicht zu drastischeren Einschnitten – etwa der ausschließlichen Verwendung von Kleinbuchstaben – durchringen will. Die Reduzierung des Zeichensatzes bietet also nur beschränkte Möglichkeiten. Allerdings wird diese Methode im Telex-Verkehr genutzt, wo sie um eine weitere Dimension gesteigert ist: Hier werden zwei Zeichen als Umschalt- (Shift-) Symbole verwendet, mit denen zwischen Großbuchstaben und einem Satz von Ziffern und Interpunktionszeichen gewechselt werden kann. Damit sind nur noch fünf Bit erforderlich, was immerhin 37,5 Prozent spart. Durch die Umschaltsymbole selbst wird ein Teil dieser Einsparung allerdings wieder verschenkt.

Ein zweiter Weg nutzt die Vergrößerung des Zeichensatzes. Diese Methode verwendet im ASCII-Satz nicht belegte Zeichen als „Tokens", sozusagen als Abkürzungen. Wenn in einem komprimierten Text ein solches Token auftaucht, wird seine Bedeutung aus einer Tabelle entnommen, in der gebräuchliche Wörter oder Sätze wie „der", „die", „oder", in kommerziellen Anwendungen beispielsweise „incl. Mehrwertsteuer" gespeichert sind. Bei sorgfältiger Wahl der Tokens lassen sich damit erstaunliche Verkürzungen erreichen; allerdings ist dazu ein gewisser Aufwand erforderlich, um die Tabelle zu erstellen. Die meisten Computer speichern auch BASIC-Programme im Token-Format, wobei alle Schlüsselwörter durch ein Byte




**Oft möchte man ASCII-Files komprimieren, um Übertragungszeiten oder Speicherplatz einzusparen. Für das „Zusammendrücken" (Squeezing) von Texten gibt es drei Grundtechniken, die sich auch kombinieren lassen. Dabei werden häufig vorkommende Begriffe in Wörterbüchern gespeichert, so daß ein Textfile nur noch aus einer Reihe von Zeigern, die die einzelnen Begriffe angeben, aufgebaut werden kann. Ein reduzierter Zeichensatz verzichtet auf überflüssige Symbole. Schließlich werden durch veränderliche Bitzahlen die häufiger vorkommenden Buchstaben mit weniger Bits dargestellt als die seltener auftretenden.**

### Häufigkeit

Viele Programme für die Textkomprimierung nutzen es aus, daß die Buchstaben eines normalen Textes in unterschiedlicher Häufigkeit auftreten. In der englischen Sprache gilt dabei diese Reihenfolge:
ETAONRISHDLFCMUGYPWBVKXJQZ

## Datenkompression extrem

Die Hoffman-Codierung sorgt dafür, daß Zeichenfolgen nur minimalen Speicherplatz brauchen. Als erstes wird dazu berechnet, wie häufig die einzelnen Zeichen auftreten; wenn etwa die Zeichenkette „ASSB" verkürzt werden soll, betragen die Häufigkeiten A:1, S:2 und B:1.

Im nächsten Schritt wird den häufigsten Zeichen eine möglichst kleine Anzahl von Bits zugeordnet. Danach werden die einzelnen Codes aneinandergereiht. Für die Decodierung arbeitet der Rechner einfach die Symbolkette ab und ersetzt jeden Code durch den entsprechenden Buchstaben. Anfang und Ende des Codes zu erkennen ist allerdings schwierig: Ist etwa A als 1 codiert, B als 0 und C als 10, dann ist nicht ohne weiteres ersichtlich, was der Code „10" bedeutet (entweder C oder auch AB).

Am besten wird so codiert, daß ein beim Lesen von links nach rechts entdeckter vollständiger Code immer den gewünschten Buchstaben ergibt, unabhängig von seiner Länge. Unser Beispiel (A,B,C) müßte daher als 1,00 und 01 codiert werden.

Das Beispielprogramm rechts läuft mit geringen Veränderungen auf fast allen Heimcomputern. (Wer mit dem Spectrum arbeitet, muß die Anweisung für die Stringdimensionierung ändern.) Dabei können Sie es entweder mit den vorgegebenenen Einstellungen arbeiten lassen oder eigene Auftretenshäufigkeiten und Zeichen festlegen. Experimentieren Sie ruhig mit verschiedenen Werten. Den größten Effekt erhalten Sie immer dann, wenn die Unterschiede in den Auftretenshäufigkeiten groß sind (so wie in den vorgegebenen DATA-Anweisungen).

```
10 INPUT "Do you wish to supply your own
 data? (y/n) ",i$
20 IF i$<>"y" AND i$<>"n" THEN GOTO 10
30 IF i$="n" GOTO 130
40 INPUT "Number of characters to encode
 ",n
50 IF n>255 THEN PRINT "Too many charact
ers...255 maximum": GOTO 40
60 DIM c$(n),f(n),r(2*n-1),t(2*n-1)
70 FOR i=1 TO n
80 PRINT "Enter character number ";i: IN
PUT s$
90 IF LEN(s$)>1 THEN PRINT "Only one cha
racter at a time...": GOTO 80
100 PRINT "Enter frequency of character
number ";i: INPUT f(i)
110 a$=a$+s$
120 NEXT i: GOTO 150
130 n=26: DIM c$(n),f(n),r(2*n-1),t(2*n-
1)
140 FOR i= 1 TO n: READ s$,f(i): a$=a$+s
$: NEXT i
150 FOR i=1 TO n: r(i)=f(i): NEXT i
160 FOR i=n+1 TO 2*n-1
170 z=9999:v=9999:k=0:g=0
180 FOR q=1 TO i-1
190 IF t(q)<>0 THEN GOTO 220
200 IF r(q)<z THEN g=k: v=z: k=q: z=r(q)
: GOTO 220
210 IF r(q)<v THEN g=q: v=r(q)
220 NEXT q
230 p=i
240 r(p)=z+v: t(k)=-p: t(g)=p
250 NEXT i
260 FOR i=1 TO n
270 c$(i)=""
280 p=i
290 IF t(p)=0 THEN GOTO 340
300 IF t(p)>0 THEN c$(i)="0"+c$(i): GOTO
 320
310 c$(i)="1"+c$(i)
320 p=ABS(t(p))
330 GOTO 290
340 NEXT i
350 FOR i=1 TO n
360 PRINT MID$(a$,i,1),r(i),c$(i)
370 NEXT i
380 END
390 DATA e,250,t,220,a,24,o,23,n,22,r,21
400 DATA i,20,s,19,h,18,d,17,l,16,f,15
410 DATA c,14,m,13,u,12,g,11,y,10,p,9
420 DATA w,8,b,7,v,6,k,5,x,4,j,3,q,2,z,1
```

symbolisiert werden.

Den Vorteil dieser Technik können Sie einmal durch Abspeichern eines nicht-„tokenisierten" Programms überprüfen. Beim Schneider CPC geht es direkt, beim Spectrum durch Speichern des geLISTeten Programms. Die Längenunterschiede sind wirklich erstaunlich. BASIC-Tokens liegen weitgehend fest, ein gutes Komprimierungsprogramm für lange Texte würde dagegen das vorliegende File durchsuchen, optimale Tokens herausfinden, danach komprimieren und den Text zusammen mit der erzeugten Token-Tabelle abspeichern.

Erfolg hat dieses Verfahren nur dann, wenn eine Vielzahl von Wendungen sehr häufig vorkommt. Es läßt sich aber bis zur Ebene einzelner Buchstaben treiben; damit sind wir beim Ansatz der dritten Komprimierungstechnik. In den beiden vorgenannten Methoden war die Anzahl der Bits für einen Buchstaben oder ein Token festgelegt. Mit einer veränderbaren Bit-Längen-Kompression (nach einem ihrer Begründer „Hoffman-Codierung" genannt) werden die am häufigsten vorkommenden Zeichen in weniger Bits gespeichert als die weniger häufigen.

Diese Methode führt dazu, daß für sehr selten vorkommende Zeichen mehr als 17 Bits nötig sind. Um die Höchstzahl der Bits zu reduzieren und außerdem auch noch Tokens nutzen zu können, wurde eine Verfeinerung entwikkelt: Dabei kommen nur Sequenzen von Vier-Bit-Datenworten zum Einsatz. In einem Vier-Bit-Wort lassen sich 16 Muster speichern. Drei davon werden genutzt, um Informationen vier Bit-Elemente zu geben. Die anderen 13 Bitmuster stellen die elf häufigsten Buchstaben sowie das Leerzeichen und das Zeichen für „Neue Zeile" dar.

Die drei Signalwerte geben an, ob es sich beim folgenden Vier-Bit-Wort um eines der 16 am häufigsten vorkommenden Wörter handelt oder ob die nächsten acht Bit eines der seltener vorkommenden Zeichen oder Wörter symbolisieren. Bei diesem Verfahren können alle 96 Zeichen sowie 205 Tokens für häufig vorkommende Wörter genutzt werden. So verkürzt sich ein File um die Hälfte.

# Einstieg in den EDV-Job

**Innerhalb der letzten 30 Jahre ist aus der datenverarbeitenden Industrie eine mächtige Branche mit unendlichem Bedarf an Arbeitskräften geworden. Dieser Sektor des Arbeitsmarktes bietet vielfältige Karrierechancen. Einige der Möglichkeiten stellen wir hier vor.**

Jede Woche erscheinen in Zeitungen und Fachzeitschriften Hunderte von Annoncen, in denen EDV-Jobs angeboten werden. Die angebotenen Gehälter liegen dabei zwischen 30 000 und 100 000 Mark im Jahr. Was ist nun wirklich an diesen Angeboten dran, wie sieht eine mögliche Karriere aus und wie kann man sich in der Datenverarbeitung etablieren?

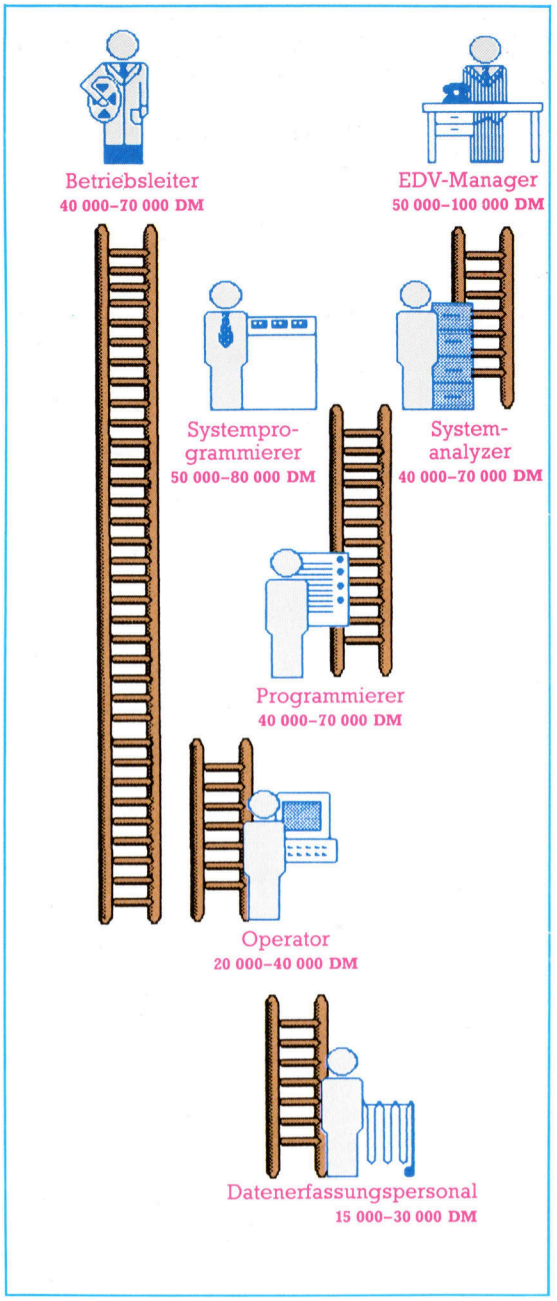
Zuerst klären wir, was EDV ist. Grundsätzlich beschäftigt sich die EDV mit der Manipulation, der Erstellung und Speicherung von Daten, die für eine bestimmte Anwendung gebraucht werden. Vieles in diesem Bereich ist Routine – etwa die Buchhaltung oder Lohnabrechnung eines Unternehmens.

Eine Tätigkeit in der EDV bedeutet noch nicht, daß Sie mit modernen Hard- oder Softwaresystemen arbeiten werden. Meist gilt der Microcomputer dem EDV-Spezialisten als Spielzeug – 90 Prozent der Arbeit werden mit Mini- oder Mainframe-Rechnern ausgeführt, die von Herstellern wie IBM, Nixdorf oder Wang sind.

Unsere Zusammenstellung möglicher Berufe in der EDV weist bereits auf die bedeutsameren Arbeitsgebiete hin. Diese Einteilung ist jedoch nicht feststehend. In manchen Betrieben kann durchaus dieselbe Person gleichzeitig Analyser und Programmierer sein.

## Karriereleiter

Auf der untersten Stufe finden wir den Angestellten für die Datenerfassung und -vorbereitung. Hier werden nur geringe Computer-Kenntnisse erwartet, die Arbeit kann ohne Vorbildung ausgeführt werden und wird dementsprechend niedrig bezahlt. Es geht dabei vorwiegend um das Erfassen von Daten. In kleineren Firmen werden diese Aufgaben noch vom Büropersonal nebenbei erledigt, wobei der

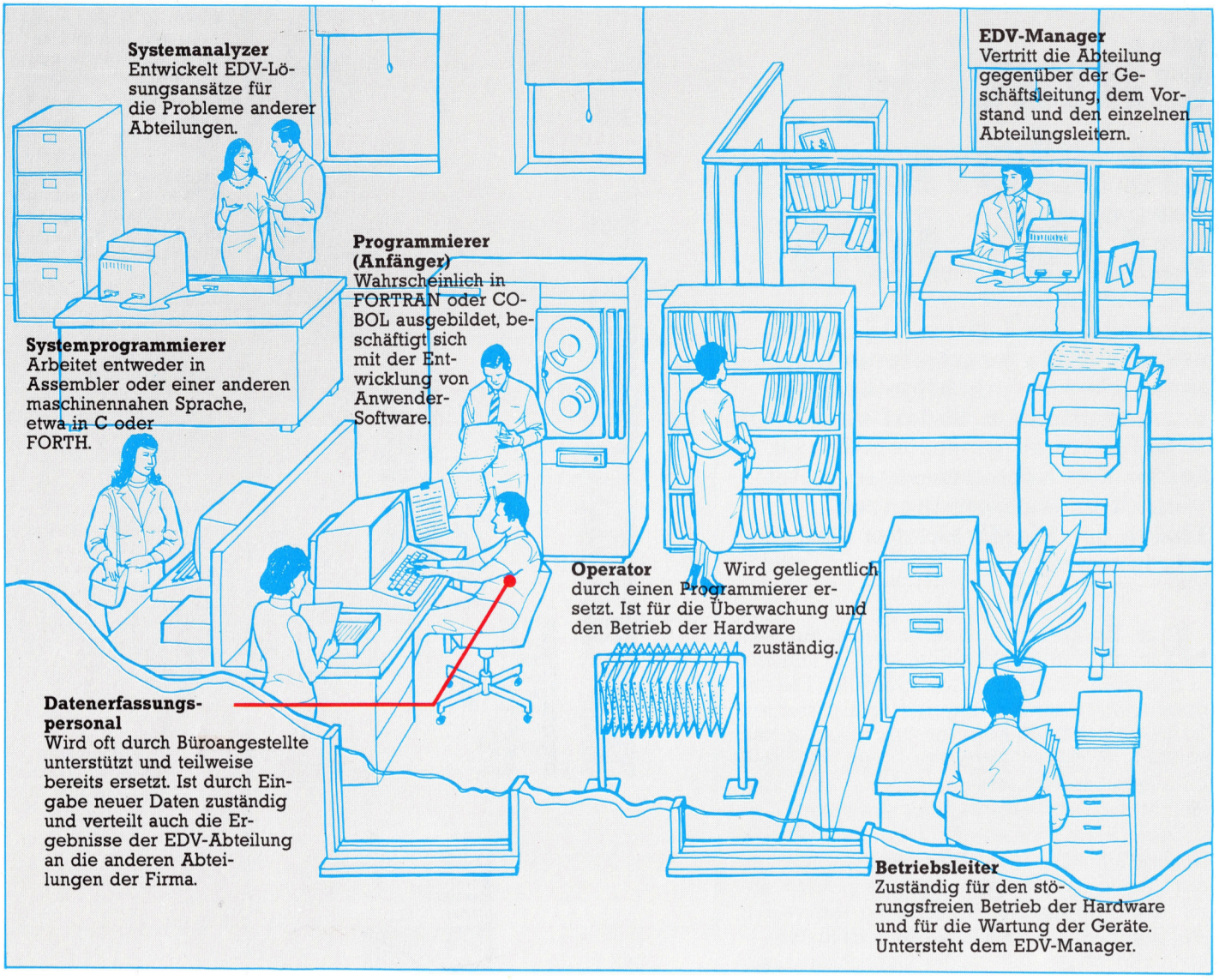



**In der EDV gibt es Tätigkeiten, die sich in ihren Anforderungen erheblich unterscheiden. Das reicht von der Funktion eines Abteilungsleiters bis zur Tätigkeit des Systemprogrammierers. Unser Bild zeigt die verschiedenen Berufswege und das Durchschnittsgehalt in den einzelnen Bereichen.**

Operator oder Programmierer vielleicht hilfreich zur Hand geht. Es ist fraglich, ob die Datenerfassung in dieser Form bereits zum EDV-Bereich gehört. Im allgemeinen ist es ohne eine zusätzliche Ausbildung nicht möglich, als Programmierer eingestellt oder beschäftigt zu werden. Der Operator hingegen kann angelernt werden.

Der Operator steht über dem Datenerfassungs-Personal und ist für den technischen Betrieb des Rechners zuständig. Im Gegensatz zum Programmierer besorgt der Operator die physikalische Abwicklung einzelner Arbeiten durch den Rechner. Er legt Bänder oder Speicherplatten ein, versorgt die Drucker mit Papier und bedient die Geräte.

Bisher war es möglich, vom Operator zum Programmierer aufzusteigen. Doch mit Vereinfachung der Bedienung und den sich daraus ergebenden geringeren Anforderungen an die Operatoren sowie durch die wachsende Zahl




**So sieht es in der EDV-Abteilung eines größeren Unternehmens aus. Die Bedeutung der EDV-Zentrale ist in den letzten Jahren durch die Verfügbarkeit dezentraler PCs weniger geworden. Trotzdem bleibt die Datenverarbeitung ein interessanter Arbeitsbereich, der immer noch einer Vielzahl von Interessenten Arbeit und Karrierechancen bieten kann.**

von ausgebildeten Programmierern hat sich dies geändert. Viele Operatoren versuchen auch heute noch aufzusteigen. Der Erfolg ihrer Bemühungen hängt jedoch von der Einstellung des Betriebsleiters ab. Wir raten zwar niemandem davon ab, als Operator tätig zu sein, ist das gewünschte Ziel jedoch der Beruf des Programmierers, so sollten Sie sich möglichst frühzeitig über die eventuellen Aufstiegsmöglichkeiten informieren.

Wer einen längeren Berufsweg in der EDV plant, steigt heute meist auf einer höheren Stufe ein – als Programmierer. Verglichen mit der Arbeit am Heimcomputer ist die Programmiertätigkeit jedoch anspruchsvoll und hochspezialisiert.

## Sprachschatz

Ein Neuling muß dazu entweder COBOL oder FORTRAN (evtl. auch PASCAL) erlernen, um anwenderspezifische Software entwickeln zu können. Es gibt jedoch in vielen Unternehmen einen zweiten EDV-Bereich, der vom Systemprogrammierer ausgefüllt wird. Systemsoftware liegt zwischen dem Betriebssystem des Computers und der anwenderspezifischen Software. Fast immer wird Systemsoftware in einer rechnerspezifischen Sprache geschrieben, meist in Assembler. Auch C gewinnt in letzter Zeit an Bedeutung.

Der nächste Schritt führt zum Systemanalyzer, der entscheidet, welche Informationen der Anwender benötigt bzw. für ihn nützlich sein könnten. Wer hier arbeiten will, muß mit Führungskräften reden können, die vom Computer nichts verstehen. Ihnen muß verständlich gemacht werden, was der Computer kann und was nicht. Noch immer werden Rechner von vielen Menschen gefürchtet und mit Mißtrauen betrachtet. Damit muß der Systemanalyzer umgehen können. Er muß Mißtrauen abbauen und neue Vorhaben genau planen und darstellen, ohne neue Furcht zu entfachen. Dazu gehört nicht nur Überzeugungskraft, sondern auch Geduld und diplomatisches Geschick.

Der Programmierer hat die Wahl zwischen zwei Berufswegen – als Software-Fachmann oder als Systemanalyzer. Die Wahl hängt von den Wünschen und Begabungen ab: Ein Ein-

zelgänger, der nicht viel Umgang mit anderen Menschen haben möchte, wird sicher als Programmierer glücklicher sein. Programmieren für Spezialgebiete kann lukrativ und intellektuell anspruchsvoll sein. Wer Teamarbeit bevorzugt und nicht nur am Bildschirm sitzen will, sollte sich besser als Systemanalyzer qualifizieren.

Der EDV-Abteilungsleiter hat drei verschiedene Aufgaben: Mit Hilfe des Systemanalyzers und des Programmierers sucht er die für seine Firma geeignete Hard- und Software und bewertet sie. Zudem ist er für die Mitarbeiter im EDV-Bereich und ihre Weiterbildung verantwortlich. Schließlich vertritt er die EDV-Abteilung auch nach außen und hält die Verbindung zur Geschäftsleitung.

## Viele Wege

Es gibt vier Wege zum Beruf des Programmierers: den Aufstieg vom Operator (eine Möglichkeit, die wir nicht unbedingt empfehlen möchten), den Einstieg über Kurse privater Schulen, den Fach- oder Hochschulabschluß Informatik oder eine Traineeausbildung, die meist ein Diplom voraussetzt. Im letzteren Fall haben auch Bewerber aus fachfremden Bereichen die Möglichkeit, ihr spezielles Berufswissen um EDV-Kenntnisse zu erweitern. Wir wollen diese Möglichkeiten betrachten.

Vom Staat geförderte Programmierkurse bieten eine gute Grundausbildung in einer der wichtigsten Programmiersprachen, meist COBOL. Zugang zu diesen Kursen hat man im allgemeinen nach einer Vorprüfung, bei der Kandidaten mit höherer Schulbildung meist begünstigt sind. Damit ist allerdings nur die erste Hürde genommen. Noch vor einigen Jahren garantierte ein erfolgreich abgeschlossener Programmierkurs einen Arbeitsplatz; das ist heute vorbei, und nicht selten folgt auf diese Ausbildung mehrmonatige Arbeitslosigkeit. Die Industrie lehnt gelegentlich Bewerber ab, die über staatlich geförderte Kurse zum Programmierer ausgebildet wurden. Das soll natürlich nicht heißen, daß ein solcher Kurs überflüssig ist, Tausende von Programmierern haben dadurch eine Stelle bekommen. Sie sollten sich aber vor Beginn des Kurses erkundigen, wie hoch die Erfolgsrate der gewählten Schule ist.

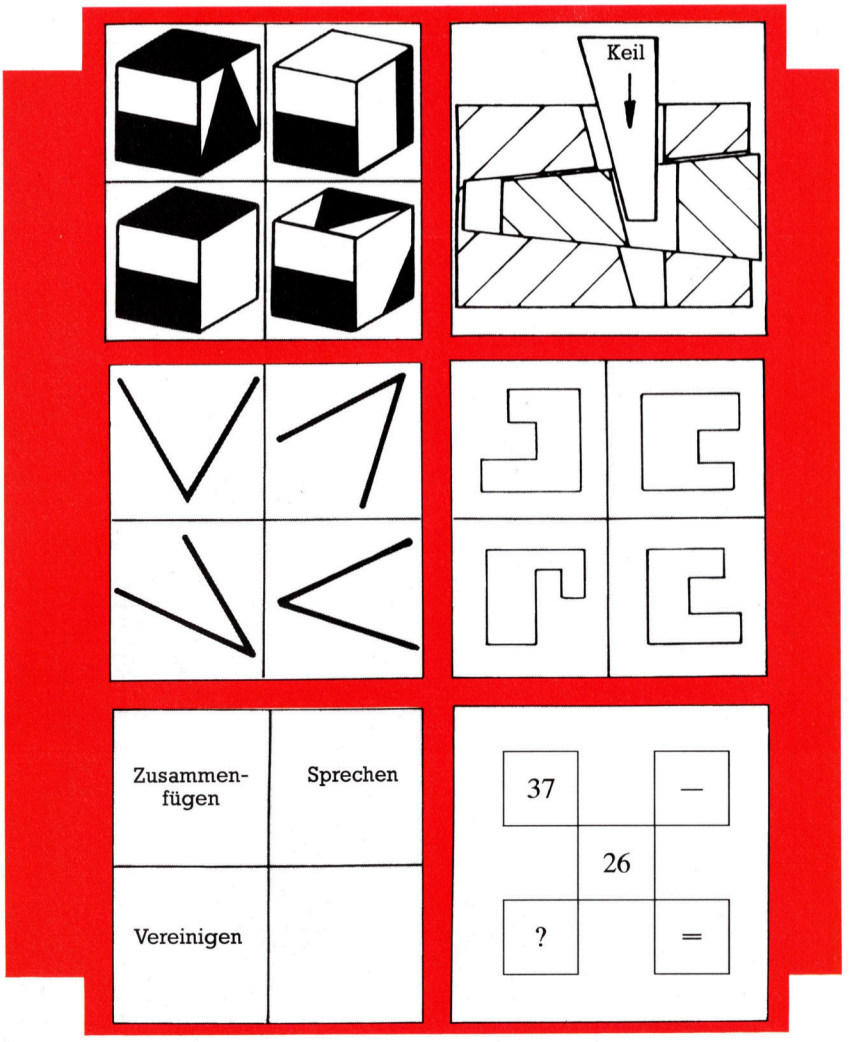
Der Beruf des Programmierers ähnelt immer mehr dem Ingenieurberuf. Informatiker haben die bessere Chancen, und auch Akademiker aus anderen Fachbereichen, wie Bewerber mit Lehrerausbildung, dringen in diesen Berufszweig vor. Wer an der Universität Mathematik oder Physik studiert hat, wird sich nicht übermäßig lange um eine Anstellung bemühen müssen. Ohne Eingangstest geht es allerdings nicht ab; mit einem Stück Papier gibt sich heute kaum noch ein Arbeitgeber zufrieden.

Die EDV steht in gewissen Grenzen auch älteren Interessenten mit Industrieerfahrung offen. Großfirmen haben bereits gute Erfolge mit Fertigungsingenieuren gemacht, die in den EDV-Bereich übergewechselt sind. Oft werden auch Teams aus Managern und Programmierern gebildet, um auf beiden Seiten das Verständnis für Anwenderwünsche und die Möglichkeiten der EDV zu fördern.

Der Aufstieg im EDV-Bereich hängt in hohem Maße davon ab, welche Fortbildungsmöglichkeiten geboten werden. Wer in einem speziellen Bereich der System-Software ausgebildet ist, verfügt über Fähigkeiten, die in den USA mit monatlich $4000 bis $7000 bezahlt werden. Auch wenn man auf eine Stelle angewiesen ist, sollte man immer die Vorstellungen des Arbeitsgebers bezüglich Fortbildung genau prüfen, bevor man ein Angebot annimmt. In einigen Firmen ist keine Weiterbildung möglich. Im allgemeinen wird erwartet, daß sich Angestellte neben der Arbeit weiterqualifizieren. Die Angst, daß ein qualifizierterer Mitarbeiter vielleicht zu einer anderen Firma wechselt, spielt eine Rolle. In der Regel ist die Fortbildung um so besser, je größer das Unternehmen ist.

Im nächsten Abschnitt soll untersucht werden, was der Einzelhandels- und Marketingbereich Ihnen bieten kann.

**Arbeitgeber verlassen sich heute immer weniger auf das reine Bewerbungsgespräch. Ausgefeilte Tests sollen dazu dienen, den geeigneten Kandidaten zu ermitteln. Die von Arbeitspsychologen entwickelten Testverfahren ähneln in mancher Hinsicht den bekannten Intelligenztests, gehen allerdings näher auf Fähigkeiten ein, die mit der späteren Beschäftigung verbunden sind. Unsere Zeichnung stellt vereinfacht einige Probleme dar, mit denen Bewerber in einem technischen Beruf konfrontiert werden. Von links oben nach rechts unten: räumliches Vorstellungsvermögen, Verständnis für mechanische Zusammenhänge, Wahrnehmungsfähigkeit, räumliches Wiedererkennen, sprachlicher Ausdruck und numerische Berechnungen.**



# Spaß mit MUD

**In Abenteuerspielen tritt der Spieler normalerweise gegen die Phantasie des Programmierers an. Anders bei MUD, dem englischen „Multi User Dungeon"-Projekt, das mehrere Spieler gleichzeitig auf einer Großrechenanlage spielen. Die Verbindung vom Heimcomputer oder Terminal zum Mainframe-Rechner läuft über das Telefon- und Datennetz.**

**In einem Mainframe-Adventure können viele Spieler gleichzeitig gegeneinander oder miteinander antreten. So können in dem Multi-User Dungeon bis zu 43 Anfänger, Krieger, Zauberinnen usw. auf Schatzsuche gehen oder sich gegenseitig den Rang von allmächtigen Hexen oder Zauberern streitig machen. Überall sind Schätze. Diese großen Abenteuer bieten auch ausführliche Beschreibungen der Spielumgebung.**

MUD läuft auf einer DEC-10-Anlage in der Universität von Essex, und Sie brauchen nur Ihren Computer, ein Terminalemulationsprogramm, ein Telefon, ein Modem und eine Benutzerkennung für das Datennetz (PSS in Großbritannien, Datex-P in Deutschland). Das Emulationsprogramm ermöglicht die Kommunikation zwischen Ihrem Computer und dem Mainframe über eine Telefonverbindung. Ob Sie ein selbstgeschriebenes oder ein fertig erworbenes Programm benutzen, steht Ihnen frei, nur sollte es 80 Zeichen per Zeile darstellen und den vollgeschriebenen Bildschirm scrollen können. Micronet Software erfüllt diese Anforderungen nicht und ist daher ungeeignet. Es ist möglich, Micros mit weniger als 80 Zeichen per Zeile zu benutzen, doch der Bildschirmaufbau ist dann unübersichtlich. Im Handel sind einige exzellente Programmpakete erhältlich, zum Beispiel Termi oder Communicator, beides ROM-Versionen, einzusetzen in den Acorn B. Das Modem muß mit dem öffentlichen Datennetz in 300/300, 1200/75 oder 1200/1200 Baud kommunizieren können. Ein handelsüblicher Akustikkoppler reicht dafür aus. Das deutsche Datex-Netz ist von vielen Städten zum Orts- oder Nahtarif per Telefon zu erreichen und ermöglicht kostengünstige Verbindungen zu weit entfernten Rechenzentren. Um in das englische PSS-Daten-

netz zu gelangen, wählen Sie die entsprechende Länderkennzahl.

MUD wird ständig aktualisiert, daher meldet sich das Programm mit dem Datum der letzten Änderung. Denken Sie sich nun einen Namen aus, unter dem Sie sich bei MUD anmelden möchten und den dann MUD in Ihr persönliches Datenpaket aufnimmt. Ein solches Datenpaket wird bei Ihrem ersten Besuch eingerichtet; Sie beginnen das Spiel mit der Spielstufe „Novice" (Anfänger). Die erreichbaren Spielstufen sind:

| Stufe | Punkte | Männlich | Weiblich |
|---|---|---|---|
| 1 | 0 | Novice | Novice |
| 2 | 400 | Warrior | Warrior |
| 3 | 800 | Hero | Heroine |
| 4 | 1600 | Champion | Champion |
| 5 | 3200 | Superhero | Superheroine |
| 6 | 6400 | Enchanter | Enchantress |
| 7 | 12800 | Sorcerer | Sorceress |
| 8 | 25600 | Necromancer | Necromancess |
| 9 | 51200 | Legend | Legend |
| 10 | 102400 | Wizard | Witch |

Wenn Sie Ihre Session „überleben", überträgt das Programm die erreichten Punkte auf Ihr Datenpaket, und Sie setzen das Spiel beim folgenden Besuch mit dieser Punktzahl fort. Der Grad der Schwierigkeit erhöht sich mit steigender Punktezahl. Ihren Punktestand vergrößern Sie durch Schätze, die Sie im Sumpf versenken, durch Lösen eines Problems, durch Zerstören der vielen Monster und durch Besiegen eines anderen Spielers. Wenn Sie von einem anderen Spieler getötet werden, löscht MUD Ihr Datenpaket, und Sie müssen das Spiel als Anfänger und ohne Punkte von vorn beginnen. Darum ist es gescheiter, Alternativen zum Kampf zu finden, denn Ihr Gegner könnte stärker sein als Sie.

## „Jez, Hallo"

Das Spiel beginnt „auf einer schmalen Landstraße". Die Eingabe „WHO" (wer) gibt Ihnen eine Liste der gleichzeitig mit Ihnen spielenden Personen auf Ihrem Bildschirm aus. Wenn Sie jemanden begrüßen möchten, schreiben Sie zum Beispiel „Jez, Hallo – Ich bin zum ersten Mal hier und könnte ein paar Ratschläge gebrauchen", und Ihre Nachricht erscheint auf dem Bildschirm des Spielers mit Namen Jez. Zum Ansprechen aller Spieler dient der Befehl „SHOUT".

Durch die Eingabe „HELP" erhalten Sie einige Hinweise, wie Sie sich bewegen können und dazu noch detaillierte Informationen über viele der Kommandos. Die Frage nach den Befehlen zur Bewegung beantwortet MUD etwa so: „Most simple movement commands are allowed, e.g. n, sw, west, up, jump, plus others you'll have to find out!" (Sehr einfache Bewegungsanweisungen sind erlaubt, z. B. n., sw, west, hoch, spring und andere, die Sie herausfinden müssen.)

Eine Liste der verfügbaren Befehle erhalten Sie mit „COMMANDS". Diese Liste ist sehr lang. Viele Terminalprogramme können den empfangenen Text zum späteren Zeitpunkt überarbeiten und auf Diskette sichern. Damit können Sie sich genaue Unterlagen über die „Computerlandschaft" erstellen und nach jedem Spiel erweitern. Die Übersicht enthält die Befehle einer früheren MUD-Version:

| | | |
|---|---|---|
| AutoWho,‹seconds› | Back | BERSERK |
| BRIEF | BUG | BYE |
| CONVERSE | DRop ‹item› | DRop ALL |
| EMPTY ‹bag› | EXITS | Flee ‹direction |
| FOLLOW ‹name› | Get ‹item› | Get ALL |
| GIve ‹item› TO ‹name› | | go ‹direction› |
| Help | HELP ‹name› | HINTS |
| HouRS | HUG ‹name› | INFO |
| Inventory | KEEP ‹item› | Kill ‹name› |
| KISS ‹name› | LEVEL | LOG |
| ‹level›, ‹message› | Look around | Look ‹bag› |
| Look ‹direction› | LOSE ‹name› | MEDITATE |
| NoPassWord | PassWord | PERSONA |
| ProNouns | QuickWho | Quit |
| "‹message› " | REFUSE ‹name› | |
| RETaliate ‹item› | SAVE | SCore |
| SHout,‹message› | SLEEP | SPELL |
| STeal ‹item› From ‹name› | | |
| tell ‹name›,‹message› | | UNKEEP |
| VERBOSE | WEIGH ‹item› | WHEN |
| WHO | WRITE ‹object›,‹message› | |

MUD ist ein großes, auf Text basierendes Abenteuer, mit sehr langen und detaillierten Beschreibungen der Umgebung. Wenn Ihnen die Szenerie bereits vertraut ist, können Sie die Beschreibungen mit „BRIEF" ausschalten. Für Anwender von Prestel und BTX ist die Teletextgrafik langsam und unzureichend, und Abenteuerspieler werden, obwohl Grafikabenteuer eine interessante Weiterentwicklung sind, Textabenteuer immer vorziehen. Ein rein auf Text basierendes Adventure erlaubt phantasievolle Verstrickungen, die mit Grafik nicht oder nur sehr aufwendig darzustellen wären, ähnlich wie ein Hörspiel im Radio fesselnder sein kann als die gleiche Geschichte im Fernsehen. Ein anderer Minuspunkt ist die unterschiedliche Grafik der Microcomputer. Für fast jedes Gerät müßte eine eigene Version erstellt werden. Viele MUD-Spieler arbeiten auf Acorn B, Apple oder Spectrum-Computern, doch andere benutzen einfache Terminals.

MUD soll demnächst auch im Handel erhältlich sein. Die Autoren des Urprogramms, Richard Bartle und Roy Trubshaw, schreiben an einer Version, die auf einem VAX Computer laufen und von Century Communications vertrieben werden soll. MUD soll weiterhin über Telefon zugängliche Datennetze gespielt werden, doch steht, bei entsprechender Nachfrage, auch eine Kabelversion in Aussicht.

# Geänderte Adressierungen

**In der vorigen Folge hatten wir uns die ersten drei Register-Adressierungsarten angesehen. In diesem Artikel beschäftigen wir uns mit den übrigen vier Adressiermethoden und ihren Unterschieden.**

Sehen wir uns kurz die bereits beschriebenen Adressierungsarten an:

**Absolute Adressierung:** Der Quell- oder Zieloperand ist eine Speicheradresse.

- **Register-Adressierung:** Quelle oder Ziel ist ein Register.
- **Indirekte Register-Adressierung (mit Pointer):** Der Pointer zeigt auf das Quell- oder Zielobjekt.
- **Indirekte Register-Adressierung (mit Nach-Inkrementierung oder Vor-Dekrementierung):** Eine Datenliste wird von oben nach unten oder umgekehrt durchgearbeitet.

Bei der „unmittelbaren Adressierung" ist eine Konstante direkt hinter dem Befehl untergebracht, so daß die Anweisung oft nur ein Computerwort belegt.

- **Unmittelbare Adressierung:** Hier ist der Quelloperand eine Konstante. MOVE.W #$43,D0 speichert die mit # gekennzeichnete Konstante $43 in D0 ($ bedeutet Hexadezimalformat). Diese Art der Adressierung heißt unmittelbar, da das konstante Computerwort direkt hinter dem Befehl steht. Bei Operanden mit Byte-Format belegt die Erweiterung natürlich nur ein Byte, bei Langworten vier Bytes.

Der unmittelbare Modus läßt sich gut für Konstantendeklarationen einsetzen, beispielsweise für eine Schleife mit vier Wiederholungen. Es gibt jedoch Situationen, in denen Sie eine Konstante besser in eine mit Namen gekennzeichnete Adresse laden, die dann absolut adressiert wird. Hier ein Beispiel:

```
MOVE    COUNT,D0
MOVE    COUNT,D5
COUNT   DC.W 4
```

Diese Methode ist flexibler, wenn der Wert von COUNT einmal geändert werden muß. Bei der absoluten Adressierung brauchen Sie nur den Inhalt von COUNT zu ändern, während bei der unmittelbaren Adressierung dafür erst mühsam alle Vorkommen der Konstanten gesucht werden müssen. Diese Suche ist weiterhin sehr fehleranfällig.

Auch bestimmte „Quick"-Befehle erlauben den unmittelbaren Modus, wobei Opcode und Konstante dann in einem Computerwort stehen. Die folgenden zwei Befehle

```
ADDQ    #3,D0
SUBQ    #1,D3
```

sind in einem Wort codiert und werden daher auch schneller ausgeführt. Beachten Sie dabei, daß die Konstanten bei ADDQ und SUBQ nur zwischen #1 und #8 liegen dürfen.

Auch der Befehl MOVE hat eine „Quick"-Version. Hier sind jedoch Konstanten im Bereich von −128 bis +128 erlaubt. MOVEQ #98,D4 speichert 98 (dezimal) in D4.

Die folgende Tabelle erhält ein Beispiel von jeder bisher behandelten Adressierungsart:

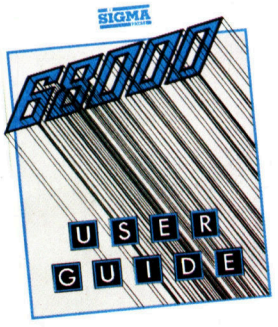
| | | | |
|---|---|---|---|
| 1 | LEA | OUTPUT,A1 | Absolut und Register |
| 2 | MOVE | A1,A2 | Register |
| 3 | MOVE | -(A1),D3 | Vor-Dekrementierung und Register |
| 4 | ADDQ | #3,D3 | Quick Unmittelbar und Register |
| 5 | MOVE | D3,(A2)+ | Register und Nach-Inkrementierung |
| 6 | ADD | #6,D3 | Unmittelbar und Register |
| 7 | MOVE | D3,(A2) | Register und Indirekt |
| 8 | INPUT | DC.W | #6 Konstante 6 |
| 9 | OUTPUT | DS.W | 2Platz für zwei Worte |

Die Speicherstelle INPUT enthält die Konstante 6, die Adresse OUTPUT hat zwei Wörter zur Verfügung.

Wie sieht nun der Inhalt von OUTPUT und OUTPUT+2 nach der Ausführung dieses Codes aus? Zeile 1 und 2 setzen A1 und A2, die dann auf OUTPUT zeigen. Zeile 3 läßt A1 auf INPUT zeigen, bevor der Operand adressiert wird. Danach wird 6 in D3 geladen. Zeile 4 addiert 3 auf D3, das dann 9 enthält.

Zeile 5 lädt D3 in OUTPUT, da A2 auf OUTPUT zeigt. Nun wird A2 um Zwei inkrementiert und zeigt so auf das zweite Wort von OUTPUT. Zeile 6 addiert 6 auf den Inhalt von D3 (D3 enthält damit den Wert 15), der dann in das zweite Wort von OUTPUT geladen wird.

- **Indirekte Adressierung (mit Adreßdistanzwert und Index):** Zunächst wollen wir untersuchen, was „Adreßdistanzwert" und „Index" überhaupt bedeuten. Auf dem 68000 bezieht sich der Adreßdistanzwert auf einen festen




**Das Anwenderhandbuch für den 68000 enthält kurze Beschreibungen der Befehle und Einsatzarten. Eine Reihe von Beispielprogrammen zeigen, wie sich die Möglichkeiten des 68000 optimal einsetzen lassen. Herausgeber: Sigma Press ISBN 1-85058-001-4.**

Offset (oder Distanz) von einem Basiswert aus, während Index per Register einen variablen Distanzwert angibt.

Das nebenstehende Bild zeigt den Unterschied zwischen diesen beiden Begriffen:
1) das Adreßregister „An" zeigt auf ein strukturiertes Datenobjekt namens Z;
2) die interne Substruktur Y wird über das Indexregister „Ri" angesprochen; während schließlich
3) auf das Datenelement X (der Struktur Y) mit dem festen Distanzwert zugegriffen wird.

Beispiele für den Einsatz sind:
1) Z: ein Array von Datensätzen oder ein zweidimensionales Array;
2) Y: ein einzelner Datensatz oder eine Arrayzeile;
3) X: das Daten- oder Arrayelement (Ganzzahl).

Da strukturierte Datenelemente dieser Art in modernen Hochsprachen wie PASCAL oder ADA häufig vorkommen, haben diese Zugriffsmethoden große Bedeutung. Natürlich gibt es keinen Grund, Assemblerprogramme nicht ebenfalls auf diese Weise zu strukturieren; der 68000 jedenfalls bietet diese Möglichkeit.

Doch zurück zu unserem strukturierten Datenobjekt. Sie haben sicherlich bemerkt, daß die Adresse von X aus der Summe von An+Ri+Adreßdistanzwert gebildet wird und An und Ri bei der Programmausführung geändert werden können. Sehen wir uns dazu einige konkrete Beispiele an. Die einfachste Methode ist die indirekte Adressierung mit Distanzwert:

```
MOVE.W DISP(A0),D1
```

DISP wurde zuvor als symbolischer Name definiert und mit einem Wert, zum Beispiel 6, belegt. Der Befehl adressiert daher die Quelle indirekt mit dem Distanzwert 6. Wenn beispielsweise A0 auf die Adresse $1000 zeigt, dann wird D1 mit dem Inhalt von $1006 geladen. Da der Befehl Wortformat hat, ist ein 16-Bit-Adreßdistanzwert möglich (von +32767 bis −32768).

Das nächste Beispiel zeigt die flexibelste Adressierungsart, die der 68000 bietet:

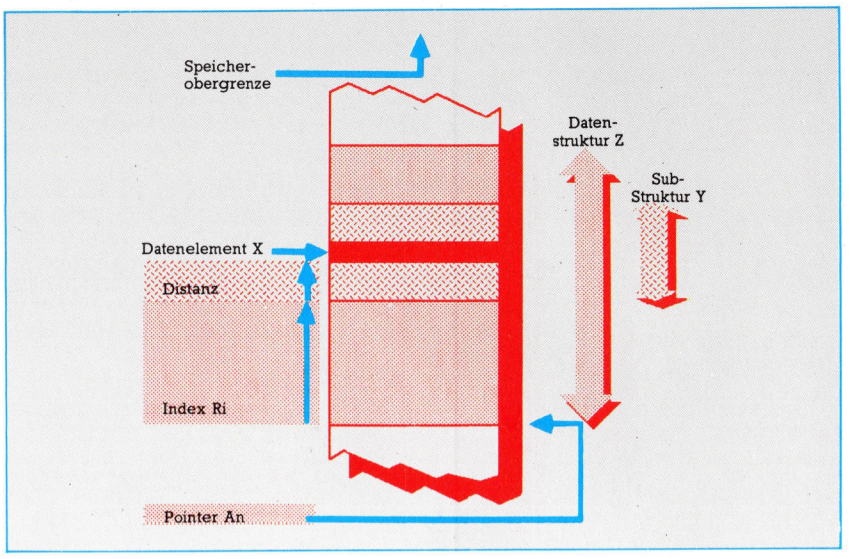
```
MOVE.W DISP(A0,D0,W),D1
```

Die Quelladresse wird durch Addieren des Basisregisters A0, des Indexregisters D0 und des festen Distanzwertes DISP bestimmt. DISP kann dabei jedoch nur eine vorzeichenbehaftete Ganzzahl im Acht-Bit-Format sein, während im Indexregister das volle 32-Bit-Format zur Verfügung steht.

Bei diesen beiden Adressierungsarten (indirekt und Distanzwert, mit oder ohne Index) wird der Distanzwert bei der Assemblierung festgelegt. Wenn Sie einen Distanzwert dynamisch ändern wollen, steht Ihnen

```
MOVE.W 0(A0,D1),D2
```

zur Verfügung, wobei D1 die Rolle des Distanzwertes übernimmt und der feste Distanzwert auf Null steht.




Der Befehlssatz des 68000 bietet diese flexiblen Adressierungsarten für die meisten der oft eingesetzten Befehle. Nachfolgend einige Adressierbeispiele mit Distanzwerten und Indizes (im Datenformat Byte):

```
1   LEA      LIST,A1
2   MOVEQ    #4,D1          D1:=4
3   MOVE.B   2(A1),D6       D6:=3
4   ADD.B    0(A1,D1),D6    D6:=8
5   MOVE.B   D6,4(A1,D1)    LIST+8:=8
6
7   LIST     DC.BQ          1,2,3,4,5,6,7,8,0
```

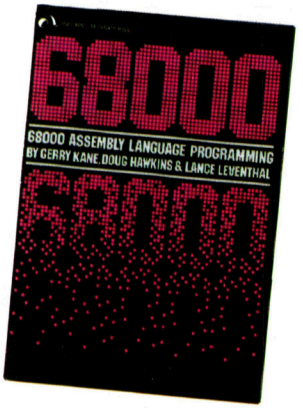
Zeile 1 veranlaßt, daß A1 auf LIST zeigt. Danach wird D1 auf 4 gesetzt. Zeile 3 addiert den Distanzwert 2 auf den Pointer in A1 und kann so das dritte Element von LIST in D6 laden. Danach wird mit dem Index 4 (in D1) der Inhalt von Element 5 auf D6 addiert und das Ergebnis in LIST+8 gespeichert.

● **Adressierung relativ zum Programmzähler (PC):** Bevor wir uns mit PC-relativen Adressiermethoden beschäftigen, sehen wir uns einige der Assembleranweisungen an. Assembleranweisungen erzeugen keinen ausführbaren Code, beeinflussen aber Faktoren wie Listenausgabe und Symboldefinitionen. Die Anweisung ORG (englisch „Origin" – Ursprung) bezieht sich auf die Adresse des Programmanfangs und die Art des erzeugten Codes.

Normalerweise wird eine Anfangsadresse mit

```
ORG $1000
```

angegeben. Damit ist die Startadresse (oder besser: die Ladeadresse für den Binärlader) auf $1000 gesetzt, so daß der darauffolgende Code in eine sequentiell aufsteigende Adresse geladen wird. Das Auftreten einer weiteren ORG-Anweisung setzt von diesem Punkt an natürlich eine neue Ladeadresse. Eine Variante ist




**Das Buch „68000 Assembly Language Programming", herausgegeben von Osborne/McGraw-Hill, enthält eine umfassende Einführung in die Programmierung des 68000. Erfahrene Programmierer werden die Teile über die Theorie der Assemblerprogrammierung überflüssig finden. Das Buch enthält jedoch alle wichtigen Informationen. Sehr praktisch sind die fettgedruckten Absatzüberschriften, durch die sich einzelne Informationen leicht finden lassen. ISBN 0-931988-62-4.**

```
ORG.L $2000
```

bei der alle Adreßbezüge als Langworte angelegt sind und absolute Adressen jeweils zwei Computerwörter belegen.

Die Anweisung RORG definiert eine Ladeadresse, von der aus alle Speicheradressen relativ zum Programmzähler angelegt sind. Diese Art Code läßt sich in jeden beliebigen Speicherbereich laden, da sich alle Adressen auf die Ladeadresse beziehen. Hier ein Bei-

```
                    RORG    $2000
2000 D47A START     ADD     CONST1,D2
2002 000A
2004 3202           MOVE    D2,D1
2006 6000           BRA     START
2008 FFFB
200A 4E40           TRAP    #0
200C 0026 CONST1    DC.W    38
```

Zur Adreßberechnung von CONST1 wird hier der 16-Bit-Distanzwert $A (dezimal 10) in Speicherstelle $2002 auf den PC addiert. Da der PC beim Laden des Distanzwertes auf $2002 steht, ist $200C die Adresse der Quelldaten. Es gibt keine speziellen Befehle oder Adressierungsarten, die PC-relativen Code erzeugen, die Anweisung RORG reicht aus.

## Der BRA-Befehl

Noch einige weitere Informationen über diesen Code. Der dem BRA-Befehl (unbedingte Verzweigung) zugeordnete Distanzwert befindet sich in $2008. Durch Addieren des PC ergibt sich die Operandenadresse von START. Das bedeutet, daß der BRA-Befehl immer PC-relativen Code erzeugt. Weiterhin wird der Befehl TRAP als Stopmarkierung eingesetzt. Bei Verwendung eines Monitors stellt er die Register dar und vereinfacht damit wesentlich die Fehlersuche.

Hier ein Beispiel, in dem die PC-relative Adressierung einen negativen Distanzwert ergibt:

```
                    RORG    $2000
2000 0026 CONST1    DC.W    38
2002 D47A START     ADD     CONST1,D2
2004 FFFC
2006 3202           MOVE    D2,D1
2008 6000           BRA     START
200A FFF8
200C 4E40           TRAP    #0
```

Der Bezug auf CONST1 in Adresse $2004 erhält einen negativen Distanzwert.

Diese Art der Adressierung läßt sich jedoch nur für den Quelloperanden einsetzen, bei

```
                    RORG    $2000
TOTAL               DC.W    0
START               ADD     D2,TOTAL
```

wird ein Assemblerfehler angezeigt. Diese Art der Adressierung eignet sich speziell für Code, der für feste ROM-Adressen bestimmt ist, zuvor aber im RAM getestet werden soll.

Es mag Ihnen wie eine Einschränkung vorkommen, daß dieser Adressiermodus nicht in Speicherstellen schreiben kann, doch steht Ihnen der „Bezug" auf absolute Adressen jederzeit zur Verfügung:

```
                    RORG    $1000
TOTAL               DC.W    0
                    RORG    $2000
CONST1              DC.W    38
START               ADD     CONST1,D2
                    MOVE    D2, TOTAL
```

Hier ist der Bezug auf TOTAL legal, da ein absoluter Adreßbereich angesprochen wird.

Die PC-relative Adressierung kann auch um einen Index mit Distanzwert erweitert werden. Hier ein Beispiel:

```
                    RORG    $3000
INDEX               DC.W    10
START               MOVEA   INDEX,A5
                    ADD     6(A5),D2
```

In diesem Fall müssen wir den Befehl MOVEA einsetzen, um A5 mit dem Inhalt der Speicherstelle INDEX laden zu können. LEA läßt sich nicht verwenden, da damit die Adresse von INDEX angesprochen würde und bei der PC-relativen Adressierung auch nicht erlaubt ist. Auch ein MOVE-Befehl wäre illegal, da ein Adreßregister nicht Zieloperand sein kann.

Doch zurück zu unserem Beispiel, in dem sich der Quelloperand des ADD-Befehls nach der Ausführung aus dem PC-Wert plus Indexregister (A5) plus Distanzwert (6) errechnet. Dabei liegt die Quellenadresse 16 Bytes hinter der Adresse mit dem Erweiterungswort 6.

Diese Adressierungsart wird allerdings dadurch eingeschränkt, daß der Distanzwert nur acht Bits des Opcodes belegen kann und damit auf Werte zwischen −128 und +127 beschränkt ist.

## Beliebige Ausführung

PC-relativer Code läßt sich an jeder beliebigen Stelle des Speichers ausführen. Die Adressiermethode wird daher hauptsächlich für die Entwicklung positionsunabhängiger Codes eingesetzt. Sie ist besonders wichtig bei Programmen mit vielen Modulen, in denen die Ladeposition eines Moduls erst beim Aufruf feststeht. Für große vielschichtige Programmiersysteme sind natürlich spezielle Speicherverwaltungssysteme nötig, doch für alle einfacheren Programmstrukturen hat die PC-relative Adressierung große Bedeutung.

- **Implizierte Adressierung:** Diese Methode ist unkompliziert, da die eingesetzten Register schon im Opcode enthalten sind. So beeinflußt beispielsweise RTS den PC und den Stackpointer, und BRA spricht den PC an.

# Fachwörter von A bis Z

**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.**

**Shell Sort = Shell Sort**
Dies Sortierverfahren wurde 1959 von dem holländischen Mathematiker Donald Shell entwickelt. Beim Shell Sort erfolgen Vergleiche und Vertauschungen zwischen den Listenelementen weiträumiger als etwa beim Bubble Sort.

Wenn mit dem Shell Sort beispielsweise ein Feld A mit acht Zahlen – etwas A=7,5,3,8,1,6,4,2 – geordnet werden soll, werden zunächst Zweiergruppen aus dem 1. und 5., dem 2. und 6. ... Element betrachtet, also (7,1), (5,6), (3,4), (8,2). Wenn in einem solchen Paar die größere Zahl vorn steht, erfolgt ein Platztausch im ursprünglichen Feld A, womit sich eine neue Liste B=1,5,3,2,7,6,4,8 ergibt. Darin werden nun Vierergruppen durch Verketten des 1., 3., 5. und 7. sowie des 2.,4. ....Elements erzeugt, also (1,3,7,4),(5,2,6,8). Wenn in einer Kombination die Reihenfolge nicht stimmt (d. h. bei 7,4 und 5,2), werden in der Liste B die Plätze gewechselt, die dadurch in die Form C=1,2,3,5,4,6,7,8 übergeht. Nun sind nur noch die Zahlen 5 und 4 zu vertauschen.

**Shift Register = Schieberegister**
In einem solchen Rechner-Register lassen sich Binärzahlen Stelle für Stelle nach links oder rechts verschieben, so etwa für die Seriell/Parallel-Wandlung: Die nacheinander übertragenen Bits füllen sukzessiv ein Schieberegister auf, das dann parallel ausgelesen werden kann. Die entgegengerichtete Umsetzung kann durch bitweises Leerschieben des geladenen Registers erfolgen. Das alles besorgt die Hardware allein; der Programmierer kommt erst mit Schieberegistern in Berührung, wenn er in Assembler- oder Maschinencode bestimmte Speicherinhalte durch Bitmanipulation verändern will.

Dabei bedeutet das Rechtsversetzen einer Binärzahl um eine Stelle eine Division durch zwei, das Linksverschieben (unter Anfügen einer Null) die Multiplikation mit zwei. Die Zweierkomplement-Darstellung einer negativen Zahl bleibt beim Rechtsschieben jedoch nur erhalten, wenn links eine Eins nachgefüllt wird. Die „arithmetischen" Schiebebefehle der CPU bewirken dies automatisch, so daß beim Rechtsschieben aus einer −16 wirklich eine −8 entsteht; beim „logischen" Schieben („Rotieren") wird das Bitmuster dagegen durch Nachziehen der am einen Ende herausgefallenen Stelle ergänzt.

**Simulation = Simulation**
Simulationsprogramme dienen dazu, reale Vorgänge auf einem Computer nachzubilden. Ein Beispiel wäre ein Simulationsprogramm für die aerodynamische Optimierung einer Pkw-Karosserie, das den Bau von Modellen und Windkanalmessungen weitgehend erübrigt.

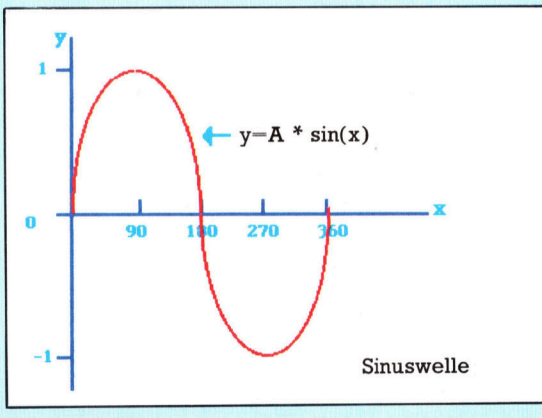
Eine weitere wichtige Anwendung sind rechnergesteuerte Simulatoren für spezielle Ausbildungszwecke, beispielsweise das Pilotentraining. Daran können auch Anfänger ohne Gefahr für teure Maschinerie oder gar Leib und Leben ihre Erfahrungen sammeln – selbst grobe Bedienungsfehler kosten nicht gleich Millionen oder den Kragen. Außerdem wird weder Kerosin vergeudet noch ein Flugzeug blockiert.

Simulationen ersparen aber nur dann Geld, wenn sie den Anforderungen der Wirklichkeit gerecht werden; für die Aussagekraft ist die exakte Nachbildung der Realität von größter Bedeutung. Bei der Simulationsplanung muß daher sorgfältig jeder in der Praxis relevante Faktor berücksichtigt werden.


**Flugsimulator-Programme gibt es auch für eine Reihe von Heimcomputern; dabei muß der „Pilot" mit dem Joystick oder den Keyboard-Tasten manövrieren.**

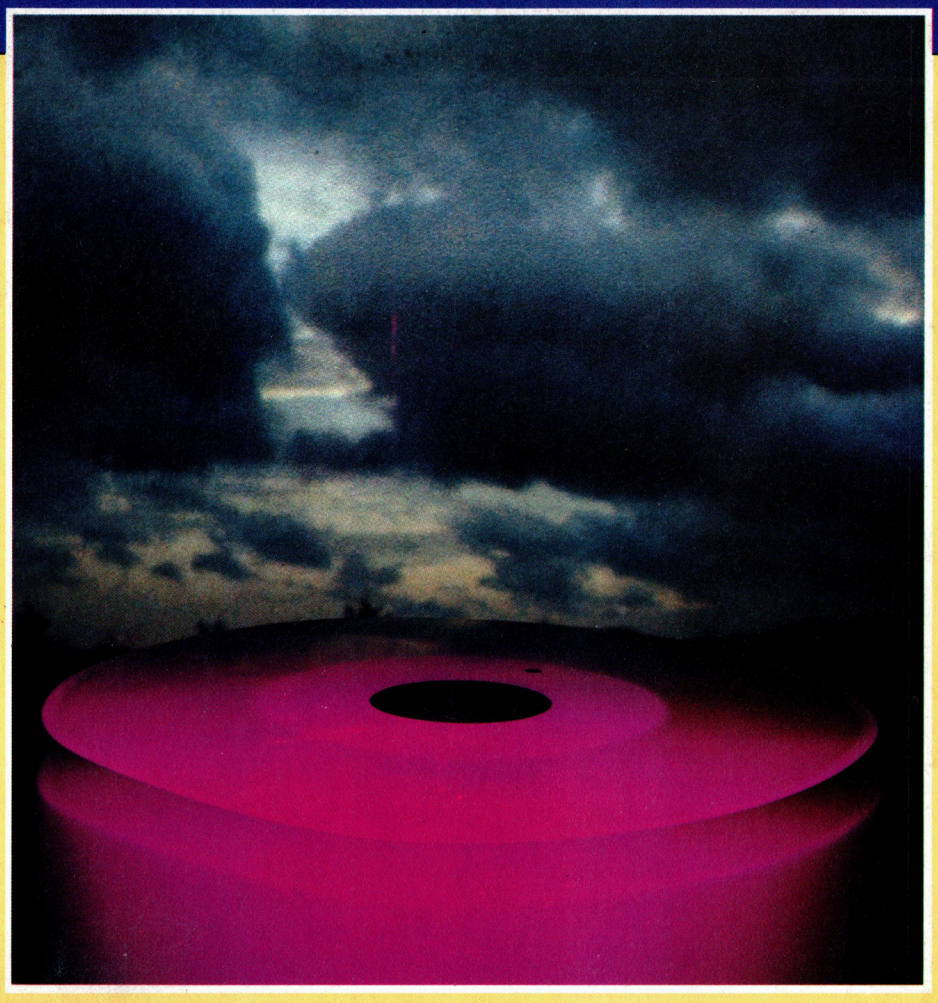
**Sine Wave = Sinuswelle**
Eine Sinuswelle wird mathematisch durch die Formel y=A * sin(x) beschrieben; der Sinus ist eine Winkelfunktion. Wenn Sie die Variable x kontinuierlich wachsen lassen, ergibt sich die unten wiedergegebene Kurve, die in charakteristischer Weise mit der „Amplitude" A um die x-Achse schwingt.

Sinuswellen werden oft benutzt, um das Übertragungsverhalten von Elektronik-Komponenten zu erfassen.



Sinuswelle

**Bildnachweise**
2186: Tony Sleep
2194: BBC Hulton PL, IBM
2208: Adrian Morgan
2192, 2195, 2203, 2205, 2210, 2211, 2212, U3: Caroline Clayton
2187, 2188, 2189: Mike Clowes
2200, 2201, 2206: Kevin Jones
2197, 2198, 2199: Chris Stevens



**Diskettenmonitore ermöglichen den „direkten Zugriff“ auf das Programm. Diese Form des direkten Zugriffs kann zum Ändern des Directory's, von Namen, Daten und Verknüpfungszeigern genutzt werden. Ebenso lassen sich verlorene Informationen wieder herstellen. Dies ist vielleicht der wichtigste Punkt. Der Direktzugriff ermöglicht es auch, zerstörte Sektorzeiger so einzurichten, daß die Sektorfolge einer Datei wieder stimmt.**

**+ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +**

# computer kurs

**Heft 80**


### Mama Bell
liefert seit Jahrzehnten Beiträge für Hard- und Software. Und sie schrieb ein Stück Computergeschichte.

### Kompaktklasse
Hier behandeln wir die Verkürzungstechnik, die „Textkompression“.

### Alle Möglichkeiten offen
sind beim UNIX Betriebssystem, das eine enorme Vielfalt an Dienstprogrammen bietet.

### Bildschirmtricks
können Sie mit dem Schneider zaubern. Wir zeigen Ihnen, wie Sie drei Module in Ihr eigenes Programm einbauen können.