
STELLMACH
STRAUCH
Das Aufsteiger-buch
C 64
Ein DATA BECKER Buch

AMIGA

Stellmach/Strauch

Amiga
Das Aufsteigerbuch

DATA BECKER

ISBN 3-89011-134-4

Copyright © 1987 DATA BECKER GmbH
Merowingerstraße 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Vorwort

Gehören Sie auch zu denjenigen, denen ihr alter C64 oder C128 nicht mehr ausreicht, weil Ihnen die Grafik zu langweilig, der Prozessor zu langsam, der Speicherplatz zu klein und die Tastatur zu sehnenzermürbend ist? Haben auch Sie sich einen Amiga 500, 1000 oder 2000 geleistet, der mit solchen Nachteilen endlich Schluß macht? Dann haben auch Sie sicher schon festgestellt, daß ein solch komplexer Computer wie der Amiga mit seinen vielfältigen Möglichkeiten zahlreiche Hürden mit sich bringt, deren Überwindung sowohl dem Anfänger als auch dem ausgebufften Freak Schwierigkeiten bereitet. Dieses Buch soll Ihnen helfen, sich schneller mit Ihrem neuen Gerät zurechtzufinden.

Während andere Einsteigerbücher sich oft in langatmigen Erklärungen darüber verstricken, was überhaupt ein Computer ist, sind wir davon ausgegangen, daß Sie schon Erfahrungen mit anderen Computern, speziell dem C64, gesammelt haben. Auf diese Weise konnten wir uns auf die interessanten Fakten konzentrieren, anstatt Ihnen den Unterschied zwischen einem Bit und einem Byte darlegen zu müssen. Dieses Buch ist somit eher ein Umsteiger- als ein Einsteigerbuch. Auch wenn Ihr alter Rechner nicht ein C64 oder C128 war, können Sie mit diesem Buch wesentlich mehr anfangen als mit Büchern, die für absolute Computerneulinge gedacht sind.

Dieses Buch verhilft Ihnen zu einem erfolgreichen Start und sorgt dafür, daß Sie gerade die Fehler vermeiden, die durch die Gewöhnung an ein System wie den C64 entstehen. Denn besonders die neue Benutzeroberfläche hebt sich doch stark von der alten ab. Daher haben wir besondere Mühe darauf verwendet, Besonderheiten des CLI und der Workbench aufzudecken, die speziell C64-Usern Probleme bereiten dürften.

Ein Schwerpunkt dieses Buches liegt auf der Beschreibung der aktuellen Programmiersprachen des Amiga. Hierbei war es unser Ziel, jedem einen schnellen Einstieg in die Programmierung zu ermöglichen. Schließlich führt der Weg zu einem guten Programm auf dem Amiga über das Erlernen einer neuen Program-

miersprache. Wir vermitteln Ihnen einen schnellen, jedoch nichtsdestoweniger gründlichen Einblick in die Sprachen AmigaBASIC, 68000-Assembler und C. Wir können und wollen im Rahmen dieses Buches keinen Programmierkurs liefern. Das würde auch der Idee widersprechen, sich nicht an völlig Unerfahrene zu wenden. Vielmehr erleichtern wir Ihnen die Entscheidung, welche Programmiersprache für Sie die richtige ist.

Neben dieser Einführung haben wir auch einen Effizienzvergleich der verschiedenen Programmiersprachen vorgenommen. Hieran können Sie ersehen, was Ihr neuer Rechner nun wirklich gegenüber dem alten mehr "bringt".

Speziell für die Maschinenspracheprogrammierer haben wir ein Kapitel verfaßt, das einen Vergleich der unterschiedlichen Hardwarekonzeptionen von C64 und Amiga durchführt. Jedoch sollte es auch für BASIC-Programmierer von Interesse sein, wie sich beispielsweise die Grafikmöglichkeiten des C64 von denen des Amiga unterscheiden.

Ein anderer Teil des Buches bezieht sich auf die Datenübertragung vom C64 zum Amiga. Bestehende BASIC-Programme, Bilder oder Texte, die Sie auf dem C64 erstellt haben, brauchen Sie so nicht völlig neu zu schreiben bzw. zu malen, sondern Sie können sie problemlos auch auf Ihrem Amiga weiterverwenden. Einige ausgefeilte Programme nehmen Ihnen dabei die Arbeit der Übertragung und Konvertierung ab.

Wir sind davon überzeugt, Ihnen mit diesem Buch ein Werkzeug an die Hand zu geben, das es Ihnen erspart, die hier gelieferten Informationen in Dutzenden von Systemhandbüchern, Zeitschriften und ähnlichen Publikationen zusammensuchen zu müssen. Im Vertrauen darauf, daß dieses Buch Ihnen zu einem schnellen und sicheren Einstieg verhelfen wird, wünschen wir Ihnen viel Erfolg!

Die Autoren

Inhaltsverzeichnis

1.	**Der schnelle Einstieg**	**13**
1.1	Der neue Rechner	13
1.2	Allgemeines zum Arbeiten mit Disketten	20
1.3	Das CLI	23
1.3.1	Aufruf des CLI	23
1.3.2	Befehlsverarbeitung des CLI	24
1.3.3	Ablage von Files in Ordnern	26
1.3.4	Erstellen einer CLI-Arbeitsdiskette	28
1.3.5	Weitere wichtige Befehle	32
1.3.6	Tips und Tricks beim Arbeiten mit dem CLI	34
1.3.7	Multitasking	37
1.4	Workbench-Hilfen	38
1.4.1	Voreinstellungen mit Preferences	39
1.4.2	Die Uhr	44
1.4.3	Der Notizblock	44
1.4.4	Icons und Icon-Editor	45
1.4.5	Sonstige Hilfsprogramme	47
1.5	RAM-Erweiterung oder zweites Diskettenlaufwerk?	48
2.	**AmigaDOS**	**51**
2.1	Die Diskettenstruktur	51
2.2	Filetypen und Fileparameter	52
2.4	Befehlsdateien	58
2.5	Systemdateien	62
3.	**Vom Microsoft BASIC V2 zum AmigaBASIC**	**67**
3.1	Unterschiede in der Konzeption der Interpreter	67
3.2	Unterschiede der BASIC-Editoren	69
3.3	Änderungen im Programmtextaufbau	73

3.4 Der Befehlssatz 76
3.4.1 Was ist gleich oder ähnlich geblieben? 76
3.4.2 Befehle für allgemeine Ein-/Ausgabe 83
3.4.3 Befehle für die Dateiverwaltung 87
3.4.4 Grafik mit AmigaBASIC 94
3.4.5 Pull-Down-Menüs 103
3.4.6 Sonstige Befehle: Sound und Joystick-Abfrage 109
3.5 Benutzung der Betriebssystemroutinen 111
3.5.1 Öffnen und Benutzen von Libraries 111
3.5.2 Wichtige DOS-Systemroutinen für BASIC-Programme 115
3.6 Einbinden von Maschinenprogrammen in BASIC-Programme 121

4. Programmierung in Maschinensprache auf dem Amiga 129

4.1 Die Programmierung des MC68000 130
4.1.1 Die verwendbaren Datenformate 130
4.1.2 Die internen Register - Aufbau und Funktionen 131
4.1.3 Die Befehlsnotation des MC68000 135
4.1.4 Die Adressierungsarten 136
4.1.5 Der Befehlssatz des MC68000 146
4.1.6 Aufruf der Betriebssystemroutinen 154
4.1.7 Die Speicherverwaltung des Amiga 158
4.2 Programmierung der Custom-Chips 160
4.2.1 Die Aufgaben der Chips 160
4.2.2 Direkter Speicherzugriff - Direkter Registerzugriff 163
4.2.3 Beschreibung der Grafik-Hardware 166
4.2.4 Sounderzeugung mit dem Amiga 172

5. Die Alternative zu BASIC und Maschinensprache: C 177

5.1 Einführung 177
5.2 Das erste C-Programm 179
5.3 Variablen und Datentypen 180

5.4 Schleifen und Verzweigungen 184
5.5 Zuweisungen und Ausdrücke 192
5.6 Datenfelder und Zeiger 196
5.7 Funktionen 200
5.8 Der Präprozessor 206
5.9 Strukturen und Varianten 209
5.10 Nutzung der Bibliotheksfunktionen 212

6. Wie schnell ist der Amiga wirklich? 215

6.1 Planung der Benchmark-Tests 215
6.2 Die Benchmark-Programme 216
6.3 Die Ergebnisse 229

7. Datenübertragung vom C64 zum Amiga 233

7.1 Bau und Betrieb eines Verbindungskabels 233
7.2 Welche Arten von Datenübertragung sind sinnvoll? 237
7.3 Das Übertragungsprogramm 238
7.4 Bedienung und Arbeitsweise des Übertragungsprogramms 269
7.5 Textdateien und Dateien aus Dateiverwaltungen 273
7.6 Übertragung von Bildern 275

8. Alte Hardware am neuen Gerät 279

8.1 Monitor 279
8.2 Drucker 280
8.3 Floppy 285

Glossar 287

Stichwortverzeichnis 307

1. Der schnelle Einstieg

Können Sie sich noch daran erinnern, welche Probleme und Schwierigkeiten Sie in der Anfangszeit mit Ihrem C64 oder C128 überwinden mußten, um mit dem Gerät arbeiten zu können? Das begann mit der Handhabung des Editors über das Arbeiten mit der Diskettenstation bis hin zu diversen Programmiertricks. Lösen konnten Sie solche Probleme sicherlich meistens nur durch Herumprobieren, den Rat von Bekannten oder das Studium von Fachzeitschriften. Damit Ihnen das beim Amiga nicht wieder passiert, haben wir dieses Buch geschrieben. Dabei kommt Ihnen zugute, daß Sie kein Computerneuling mehr sind. Das nun folgende Kapitel setzt sich mit der allerersten Problemquelle auseinander, die einem bei jedem Computer begegnet, nämlich mit der Benutzeroberfläche.

1.1 Der neue Rechner

Die Unterschiede vom C64 zum Amiga beginnen schon beim Einschalten. Der C64 meldet sich sofort mit einem blinkenden Cursor, da sowohl das Betriebssystem als auch das BASIC, das beim C64 die Benutzerschnittstelle beinhaltet, in ROM-Bausteinen enthalten sind, die ihren Inhalt auch nach dem Ausschalten nicht verlieren.

Der Amiga dagegen will als allererstes eine Diskette von Ihnen haben. Der Amiga 1000 verlangt zuerst eine Kickstart-Diskette. Kickstart ist das eigentliche Betriebssystem des Rechners, das alle wichtigen Funktionen übernimmt, angefangen von der Tastatureingabe über die Bildschirmausgabe bis zur Verwaltung der Diskettenlaufwerke und des Druckers. Als Amiga-1000-Besitzer haben Sie normalerweise auch zwei verschiedene Kickstart-Disketten erhalten. Sie sollten nach Möglichkeit immer Kickstart 1.2 verwenden, da diese gegenüber der Kickstart 1.1 wesentliche Verbesserungen enthält. Kickstart 1.1 ist nur deshalb mitgeliefert worden, da es einige wenige Programme gibt, die sich mit dem neuen Betriebssystem nicht vertragen.

Nach dem Booten (Laden) der Kickstart befindet sich Ihr Amiga im gleichen Zustand wie ein Amiga 500 oder 2000 nach dem Einschalten. Er will nun die Workbench-Diskette haben. An dieser Stelle können Sie aber auch irgendeine andere bootfähige (automatisch ladbare) Diskette ins Laufwerk schieben. Zum Arbeiten werden Sie aber zunächst die Workbench benötigen. Die Workbench (engl. Werkbank) ist ein sehr mächtiges Hilfsprogramm, das den Umgang mit Disketten und Dateien erleichtert.

Nachdem Sie die Workbench-Diskette eingelegt haben und der Amiga alles fertig eingeladen hat, kommt die Maus ins Spiel. Sie ist innerhalb der Workbench das zentrale Eingabe-Instrument. Wie Sie sicherlich schon selbst bemerkt haben, können Sie mit der Maus einen Zeiger, den sogenannten Mauszeiger, frei auf dem Bildschirm bewegen. Die linke Maustaste, auch Auswahltaste genannt, dient zum Anwählen der verschiedenen Symbole. Dies geschieht entweder durch ein kurzes Drücken der Taste (Anklicken), sofern der Mauszeiger auf das gewünschte Symbol zeigt, oder durch ein zweimaliges Drücken in kurzem Abstand (Doppelklick).

Mit der rechten Maustaste, die den Namen Menütaste trägt, können Sie die sogenannten Pull-Down-Menüs nutzen. Dazu halten Sie diese Taste gedrückt und bewegen den Mauszeiger in die Nähe des Balkens am oberen Rand, der Menüleiste. Unter jedem der in der Menüleiste angezeigten Menütitel erscheint dann eine Reihe von Menü-Unterpunkten. Bewegen Sie den Mauszeiger auf einen dieser Punkte und lassen Sie dann die Menütaste los, so wird der entsprechende Punkt ausgewählt.

Nach dem Laden der Workbench ist auf dem Bildschirm das Disketten-Symbol der Workbench-Diskette zu sehen. Man bezeichnet es auch als Piktogramm oder Icon. Icons sind alle diejenigen Symbole, die Disketten oder Dateien repräsentieren und die man mit der Maus quasi nehmen und verschieben kann. Dies geschieht, indem Sie mit dem Mauszeiger auf das entsprechende Icon zeigen, dann die Auswahltaste gedrückt halten und die Maus mit dem Icon an die gewünschte Stelle schieben.

Durch einen Doppelklick auf das Diskettensymbol öffnet sich ein Fenster oder Window. Der Reihe nach erscheinen dann innerhalb dieses Fensters weitere Icons, die Dateien auf der Diskette bezeichnen. Das Fenster zeigt demnach das Inhaltsverzeichnis (Directory) der Diskette. Alle Dateien lassen sich, genau wie das Diskettensymbol, durch einen Doppelklick aktivieren. Dabei können unterschiedliche Funktionen ausgelöst werden, je nachdem, um was für eine Datei es sich handelt. Es gibt zum Beispiel die sogenannten Ordner, kenntlich gemacht durch ein Schubladensymbol. Sie enthalten weitere Dateien, worunter sich auch zusätzliche Ordner befinden können. Ordner dienen dazu, zusammengehörige Files unter einem Namen zusammenzufassen. Durch einen Doppelklick erscheint ein weiteres Fenster, das die Dateien im Ordner anzeigt. Ordner werden daher auch als Unter-Directories bezeichnet.

Wenn man eine Programmdatei doppelt anklickt, wird das entsprechende Programm gestartet. Probieren Sie doch einmal folgendes aus: Öffnen Sie zuerst das Fenster der Workbench-Diskette und dann das Fenster des Ordners Demos. Ihnen werden dann vier Demonstrationsprogramme zur Auswahl geboten, die Sie durch einen Doppelklick aufrufen können. Jedes der Programme benutzt wiederum ein eigenes Fenster. Sie werden übrigens bemerken, daß man auch mehr als ein Programm gleichzeitig ablaufen lassen kann, indem man nämlich einfach nacheinander zwei verschiedene Programme startet. Man kann sogar ein- und dasselbe Programm mehrfach ablaufen lassen. Jedes mal entsteht ein neues Fenster, und in allen Fenstern läuft gleichzeitig das Programm. Diese Eigenschaft des Amiga nennt sich Multitasking, und wir werden später noch genauer darauf zu sprechen kommen. Beim zweimaligen Start des gleichen Programmes liegen die beiden Fenster allerdings genau übereinander, was sich durch Verschieben eines Windows ändern läßt. Dazu aber mehr in den nächsten Absätzen.

Eine wichtige Sache, die durch das Multitasking bedingt ist und eine Fehlerquelle gerade für Amiga-Neulinge darstellt, ist das Anwählen des richtigen Fensters für die Eingabe. Da Sie ja nur eine Tastatur besitzen, ist es erforderlich, eindeutig festzulegen, auf welches Fenster die Eingaben wirken. Dies geschieht durch

Hineinklicken mit der Maus in das entsprechende Fenster. Die Namen aller anderen Fenster werden jeweils in Geisterschrift angezeigt, nur das aktuelle Eingabefenster hat seinen Namen in normaler Schrift. Auch wenn Sie bisher noch keine Fenster kennengelernt haben, für die eine Tastatureingabe notwendig war, so werden Sie doch in Zukunft noch öfter darüber stolpern.

Fenster gehören sowieso zu den wichtigsten Ein-/Ausgabe-Elementen des Amiga. Wie jeder durch Ausprobieren sehr schnell bemerken wird, kann man mit solchen Windows (engl. Fenster) eine ganze Menge machen. Dazu dienen die sogenannten Gadgets (sprich: Gädschets). Dieser nur schwer zu übersetzende Ausdruck (deutsch: kleine Vorrichtung oder Dingsbums) bezeichnet die verschiedenen Symbole am Rand und in den Ecken der Fenster. Nicht alle Fenster enthalten die gleichen Gadgets, aber die im folgenden beschriebenen Gadgets trifft man relativ häufig an.

Da wäre zunächst das Close-Gadget (Schließsymbol) in der linken oberen Ecke. Damit lassen Sie ein Fenster wieder vom Bildschirm verschwinden, Sie schließen es. Auf diese Weise können Sie auch die Demonstrationsprogramme abbrechen, indem Sie einfach das zugehörige Close-Gadget anklicken.

Weiterhin gibt es die Verschiebeleiste des Fensters, also der obere Balken, der auch den Namen enthält. Bewegen Sie den Mauszeiger einfach auf diese Leiste, halten Sie die Auswahltaste gedrückt, und verschieben Sie das Fenster dorthin, wohin Sie es haben wollen. Ähnlich funktioniert das Größen-Gadget in der Ecke rechts unten. Zeigen Sie mit dem Pfeil auf das Symbol, und halten Sie die linke Maustaste gedrückt. Jetzt können Sie problemlos die Fenstergröße verändern.

Wenn Sie ein Fenster so klein machen, daß nicht mehr der gesamte Inhalt auf einmal angezeigt werden kann, so können Sie durch die Rollbalken am rechten und am unteren Rand bestimmen, welcher Ausschnitt des Inhalts sichtbar sein soll. Dazu halten Sie wiederum die Auswahltaste gedrückt und verschieben den ausgefüllten Balken innerhalb seines Rechtecks. Am besten

ist es, wenn Sie das einmal selbst ausprobieren. Dies gilt auch für alle anderen Funktionen. Auf diese Weise lernen Sie wesentlich schneller, mit den Bedienungselementen umzugehen.

Die beiden Symbole rechts oben sind die Vordergrund-/ Hintergrund-Gadgets. Da sich Fenster auch überlappen können, ist es wichtig, angeben zu können, welches Fenster vorne und welches hinten liegt. Klicken Sie das linke dieser beiden Gadgets an, wird das Fenster in den Hintergrund geschaltet, beim rechten in den Vordergrund.

Zwischen den Disketten-Windows und den Ordner-Windows besteht übrigens ein kleiner Unterschied. Die Disketten-Windows enthalten nämlich am linken Rand einen Balken, der anzeigt, wieviel Speicherplatz schon auf der Diskette belegt und wieviel noch frei ist. Der Balken wächst von unten nach oben, wobei am einen Ende ein E für empty (engl. leer), am anderen ein F für full (engl. voll) steht.

Mit Dateien kann man natürlich noch mehr machen als sie sich ansehen und sie starten. Man kann sie beispielsweise umbenennen, kopieren oder löschen, oder die Position ihrer Icons verändern. Da vieles sich aber aufgrund der Menüführung der Workbench von selbst erklärt, wollen wir nur kurz die wichtigsten Funktionen zusammenfassen.

Beim Umbenennen einer Datei kommen zum ersten Mal die Pull-Down-Menüs zum Zug. Wie Sie sicherlich schon bemerkt haben, verändert sich ein Icon farblich, wenn man es nur einmal anklickt, es ist dann angewählt. Sofern Sie den Namen eines Icons ändern wollen, wählen Sie es an, und halten Sie dann die Menütaste gedrückt. Fahren Sie mit dem Mauszeiger unter den Menütitel Workbench, und wählen Sie aus dem angezeigten Menü die Option Rename. Es erscheint dann ein Balken mit dem alten Namen und ein Cursor. Jetzt können Sie den neuen Namen eingeben beziehungsweise den alten Namen abändern. Genauso lassen sich auch Diskettennamen ändern. Für diese Operationen müssen Sie natürlich den Schreibschutz von der Diskette entfernt haben, auf der das Umbenennen stattfindet.

Beim Kopieren von Dateien innerhalb einer Diskette schieben Sie einfach das entsprechende Icon vom Fenster des Quell-Ordners ins Fenster des Ziel-Ordners. Dabei entsteht kein Duplikat der Datei, sondern sie wird bloß verschoben. Duplikate entstehen nur, wenn Sie die Datei anwählen und Duplicate aus dem Workbench-Menü verwenden. Genauso können Sie sich auch einen neuen Ordner für eigene Dateien schaffen. Erstellen Sie dazu eine Kopie des leeren Ordners Empty, und geben Sie diesem dann einen anderen Namen.

Löschen von Dateien sollten Sie mit der Workbench immer so bewerkstelligen, daß Sie die entsprechende Datei nehmen und auf das Trashcan-(Mülleimer-)Icon legen. Der Mülleimer ist eigentlich nichts anderes als ein Ordner, in den die Datei kopiert wird. Man kann den Trashcan deshalb auch wie einen normalen Ordner öffnen und Dateien aus ihm in einen anderen Ordner kopieren. Daher verringert sich der Speicherplatz auf der Diskette auch nicht, wenn eine Datei im Mülleimer abgelegt wird. Wirklich leerer wird die Diskette erst dann, wenn Sie das Trashcan-Icon anwählen und aus dem Disk-Menü den Punkt Empty Trash benutzen. Dann lassen sich die Dateien allerdings nicht mehr zurückholen.

Beim Kopieren von einer Diskette zur anderen enstehen ebenfalls Duplikate, da die Dateien auf der Quelldiskette nicht gelöscht werden. Auch hier nehmen Sie einfach das Datei-Icon und legen es im Fenster der Zieldiskette ab. Bei zwei Laufwerken klappt das problemlos, wenn Quell- und Zieldiskette sich in je einem der beiden Diskettenlaufwerke befinden. Sie können dann auch mehrere Dateien kopieren, indem Sie die Shift-Taste gedrückt halten und der Reihe nach alle gewünschten Icons anwählen. Beim letzten Icon halten Sie dann die Auswahltaste gedrückt und können nun alle Icons auf einmal verschieben. Natürlich können Sie auch vollständige Ordner mit Inhalt kopieren.

Das Ganze hat aber einen Haken, sofern Sie nur ein Laufwerk besitzen. Für jede Datei, die Sie kopieren, müssen Sie dann mindestens zweimal die Diskette wechseln. Um diesem Manko abzuhelfen, gibt es die RAM-Disk. Bei einem Amiga 500 und

2000 sollten Sie schon das RAM-Disk-Icon bemerkt haben, das neben dem Workbench-Icon zu sehen ist. Bei einem Amiga 1000 ist dieses Icon meistens nicht vorhanden, es läßt sich aber installieren (siehe Kapitel 1.3). Es handelt sich dabei um einen Teil des Speichers, der sich wie eine Diskettenstation verhält. Natürlich geht der Inhalt der RAM-Disk verloren, sobald man den Rechner ausschaltet. Kopieren Sie bei einem Laufwerk zuerst die gewünschten Dateien auf die RAM-Disk, wechseln Sie dann die Disketten, und kopieren Sie nun die Dateien von der RAM-Disk auf die Zieldiskette. Abschließend löschen Sie die Dateien aus der RAM-Disk, indem Sie alle Icons im RAM anwählen (mit SHIFT) und dann die Option Discard aus dem Workbench-Menü benutzen.

Kopieren kompletter Disketten geschieht durch Anwählen des Icons der zu kopierenden Diskette und Aufruf der schon beschriebenen Funktion Duplicate. Der Amiga sagt Ihnen dann alles weitere. Dies funktioniert auch mit nur einem Diskettenlaufwerk ohne zuviel Aufwand. Die Zieldiskette braucht nicht wie bei vielen anderen Computern vorher formatiert zu werden. Wenn Sie allerdings eine neue Diskette für Ihre eigenen Programme benutzen wollen, muß sie durch einen Formatiervorgang entsprechend vorbereitet werden. Dazu schieben Sie die leere Diskette ins Laufwerk und wählen nach Erscheinen des Disketten-Icons dieses an. Danach rufen Sie den Menüpunkt Initialize aus dem Diskmenü auf.

Probieren Sie ruhig aus, was sich mit der Workbench alles anstellen läßt. Wenn Sie danach aber etwas anderes machen wollen, so genügt es, beim Amiga 1000 die beiden Amiga-Tasten (links und rechts von der Leertaste) und die Ctrl-Taste, beim Amiga 500/2000 die Amiga-Taste, die Commodore-Taste und die Ctrl-Taste gleichzeitig zu drücken. Dadurch wird ein Reset ausgelöst, wodurch der Amiga in den Zustand vor dem Laden der Workbench zurückgesetzt wird. Man braucht also seinen Amiga nicht auszuschalten, wenn man ein neues Programm benutzen will. Benutzen Sie aber niemals diese Tastenkombination, solange die Lampe des Diskettenlaufwerks noch leuchtet.

Als Ex-C64-Besitzer kennen Sie vielleicht diverse Reset-Schalter, die das gleiche beim C64 bewirkten. Man hatte damit manchmal das Problem, daß einige Programme resetgeschützt waren. Mit solchen Spielereien werden Sie auf dem Amiga aber wahrscheinlich nichts zu tun bekommen, obwohl prinzipiell das Sperren des Resets möglich ist.

Vielleicht haben Sie auch schon einmal eine Diskette zur Verfügung gehabt, die offensichtlich nicht leer war, was sich am Speicherbelegungsbalken ablesen ließ, die aber trotzdem keine sichtbaren Programme enthielt. Die Workbench kann nämlich keine Programme anzeigen, die kein Icon besitzen. Daher werden wir uns in Kapitel 1.3 mit dem sogenannten CLI auseinandersetzen, daß einem eine wesentlich größere Kontrolle über alle Vorgänge auf der Diskette erlaubt. Das CLI erreichen Sie über den System-Ordner, der ein Icon mit dem Namen CLI enthält. Wenn Sie dieses Icon doppelt anklikken, erscheint das sogenannte CLI-Fenster, das Sie am besten zuerst auf eine für die Arbeit sinnvolle Größe bringen. Das CLI ist im Gegensatz zur Workbench tastaturorientiert, dazu aber mehr in Kapitel 1.3. Vorher sollten Sie sich aber einmal Kapitel 1.2 ansehen, das Ihnen einige wichtige Tips im Umgang mit 3½-Zoll-Disketten gibt. Interessieren Sie sich dagegen mehr für komplexere Funktionen und Hilfsprogramme der Workbench, so werfen Sie doch einen Blick in Kapitel 1.4. Hier finden Sie sicherlich das, was Sie suchen.

1.2 Allgemeines zum Arbeiten mit Disketten

Bevor Sie überhaupt mit dem Amiga zu arbeiten anfangen, sollten Sie unbedingt eine Sicherheitskopie Ihrer Workbench-Diskette erstellen. Klicken Sie dazu am besten schon beim ersten Start der Workbench das Workbench-Icon an, und wählen Sie den Menüpunkt Duplicate. Alles weitere sagt Ihnen der Rechner. Bewahren Sie die Original-Workbench-Disk an einem sicheren Ort auf, und arbeiten Sie nur mit der Kopie. Dies schützt sie vor einer unbeabsichtigten Zerstörung.

3½-Zoll-Disketten beim Amiga haben den Vorteil, daß sie sich wesentlich leichter vor dem Überschreiben schützen lassen als 5¼-Zoll-Disketten. Unsere Empfehlung: Nutzen Sie das! Am besten gewöhnen Sie sich an, jede Diskette, auf der Sie nicht regelmäßig Daten abspeichern, mit einem Schreibschutz zu versehen. Falls Sie sich nicht sicher sind: Der Schreibschutz ist aktiviert, wenn sich der Schalter oben befindet und das Loch sichtbar ist. Sollten Sie einmal vergessen haben, vor einem Speichervorgang den Schreibschutz zu entfernen, so macht das überhaupt nichts. Der Amiga sagt Ihnen das, und Sie können nun die Diskette herausziehen, den Schreibschutz entfernen und die Diskette wieder einlegen. Der Amiga arbeitet dann korrekt weiter.

Übrigens werden die Disketten beim Amiga grundsätzlich auf beiden Seiten bespielt. Beim C64 hat fast jeder vernünftige Mensch, der nicht zuviel Geld besaß, seine Disketten mit einer zweiten Schreibschutzkerbe versehen, so daß er sie auch umgekehrt ins Laufwerk schieben konnte. Der Amiga dagegen besitzt zwei Schreib-/Leseköpfe, einen für jede Diskettenseite. Beide Seiten werden auch gemeinsam verwaltet, das heißt, es gibt zum Beispiel nur ein Directory und einen Schreibschutzschalter für beide Seiten. Wer einen C128 mit der Floppy 1571 besitzt, kennt das, obwohl im C64-Modus das Umdrehen der Diskette noch erforderlich war.

Bei einseitigen No-Name-Disketten, also Disketten, die nicht das Label eines bestimmten Herstellers tragen, ist die Ausfallquote etwa genauso hoch wie bei 5¼-Zoll-Disketten. Es kann also schon unter 50 Disketten mal eine dabei sein, die sich nicht problemlos bespielen läßt. Das Amiga-Laufwerk ist im Gegensatz zur 1541/70/71 für Double Density-Disketten (DD, 135 tpi) gedacht, da es auf jeder Diskettenseite 80 Tracks verwaltet. Trotzdem haben wir auch schon mit Single Density-Disketten gearbeitet, was zu keinerlei Problemen geführt hat. Besser ist aber in jedem Fall, Double Density zu verwenden. Diese Disketten kosten meistens auch nur unwesentlich mehr als die anderen.

Gerade wenn Sie nicht mit den teuersten Disketten arbeiten, ist es sicherer, bei allen Kopierprogrammen, die einem eine Verify-Option anbieten, diese auch auszunutzen. Seltsamerweise kopiert der Duplicate-Befehl der Workbench eine Diskette grundsätzlich ohne Verify. Wenn Sie im Besitz eines Kopierprogrammes sind, das Verify anbietet, sollten Sie dieses vorziehen, auch wenn der Verifiziervorgang etwas mehr Zeit benötigt.

Bei Speichervorgängen arbeitet der Amiga mit einem Trackbuffer. Das bedeutet, er hat grundsätzlich einen kompletten Track im Ram. Wenn Schreibvorgänge ablaufen, werden die Daten zuerst in den Buffer geschrieben. Wird der Track gewechselt oder werden eine Zeitlang keine Daten abgelegt, so schreibt der Amiga den Track auf die Diskette zurück. Dadurch kann es zu der kuriosen Situation kommen, daß sich nach einem Speicherbefehl der Computer zurückmeldet, obwohl das Diskettenlaufwerk noch gar nicht angesprungen ist. Erst etwa zwei Sekunden später beginnt das Laufwerk, den Track zu schreiben. Seien Sie hier also vorsichtig, und ziehen Sie niemals die Diskette zu früh aus dem Diskettenschacht, insbesondere nicht, bevor die Lampe erloschen ist.

Es kann manchmal etwas länger dauern, bis der Amiga erkennt, daß Sie eine neue Diskette eingelegt haben. Wenn Sie nach einem solchen Einlegen einen Diskettenzugriff durchführen wollen und nicht abwarten, bis das Laufwerk kurz angelaufen ist, beschwert sich der Amiga, Sie hätten keine Diskette eingelegt. In den meisten Fällen brauchen Sie aber nicht das Feld Retry anzuklicken. Es genügt fast immer, bloß abzuwarten, bis die Diskette erkannt wird. Der Grund für die große Zeitspanne liegt in der Tatsache, daß das Laufwerk das Signal, ob eine Diskette eingelegt wurde, nur dann an den Rechner liefert, wenn ein Track-Wechsel durchgeführt wird. Zu oft wollten die Entwickler von Commodore einen solchen Wechsel aber nicht durchführen, da das dem Benutzer durch ein unerwünchtes Klacken unangenehm auffällt. Das ist auch der Grund, warum bei nicht eingelegter Diskette das Laufwerk die bekannten Geräusche von sich gibt. Warum man nicht einfach zwei Track-Wechselsignale so schnell hintereinander sendet, daß der Kopf gar nicht bewegt

wird, ist allerdings unverständlich. Unserer Meinung nach sollte das nämlich ausreichen, um das Diskwechsel-Signal abfragen zu können, und zwar ohne den Verschleiß des Laufwerks zu erhöhen.

1.3 Das CLI

Die Workbench sieht zwar sehr schön aus und ist auch sehr leicht zu handhaben. Gerade für einen schnellen Einstieg ist aber die Kenntnis des CLI viel wichtiger. Zum ernsthaften Arbeiten benötigt man es nämlich viel öfter als die Workbench.

1.3.1 Aufruf des CLI

Was ist das überhaupt: das CLI? CLI steht für Command-Line-Interface, und es handelt sich dabei um den Teil des Betriebssystems, der Befehle per Tastatur entgegennimmt und ausführt. Das CLI gehört zum AmigaDOS (Disk-Operating-System), also zu der Systemeinheit, die die Diskettenlaufwerke verwaltet. Beim C64 hatte man alle DOS-Befehle von BASIC aus zur Verfügung. Beim Amiga dagegen sind DOS und BASIC streng voneinander getrennt. Das DOS bietet aber Möglichkeiten, quasi Programme zu erstellen, die nur aus DOS-Befehlen bestehen. Darauf kommen wir aber später zurück.

Wie kommt man ins CLI? Die vom Handbuch vorgeschlagene Methode funktioniert natürlich immer, auch wenn sie prinzipiell sehr umständlich ist. Man lädt die Workbench und öffnet das Fenster der Workbench-Diskette. Dann öffnet man den Ordner System und klickt zweimal das CLI-Icon an. Schneller geht es zweifellos, indem man schon während des Boot-Vorgangs, also dann, wenn man seine Workbench-Diskette das erste Mal eingelegt hat, beim Erscheinen des Fensters die Tastenkombination Ctrl D drückt. Das hat aber den Nachteil, daß nicht alle Befehle, die das DOS vor dem eigentlichen Laden der Workbench normalerweise ausführt, wirklich ausgeführt werden. Beispielsweise fehlt Ihnen wahrscheinlich nach Ctrl-D die deutsche Tastaturbelegung. Daher ist es am besten, einen dritten Weg einzu-

schlagen. Erstellen Sie doch einfach eine Kopie Ihrer Workbench-Diskette, und ändern Sie sie so ab, daß die Workbench erst gar nicht geladen wird. Wie das genau geht, beschreiben wir etwas später. Man sollte nämlich vorher verstanden haben, wie das CLI arbeitet.

Sobald sich das CLI mit dem Cursor meldet, können Sie beliebige Eingaben machen. Wenn Sie versuchen, die Cursortasten zu benutzen, werden Sie feststellen, daß das leider nicht möglich ist. Man kann also nicht hingehen und einen Befehl, bei dem man sich ganz zu Anfang der Zeile vertippt hat, einfach nur an der entsprechenden Stelle korrigieren. Sie müssen schon mit der Backspace-Taste alle Zeichen bis zum Fehler löschen. Genausowenig können Sie einfach eine Zeile höher fahren, um einen schon benutzten Befehl nocheinmal zu verwenden. Dieser an sich lästige Umstand spart dem Betriebssystem eine Menge Speicherplatz, da es schon ausgeführte Zeilen nicht zwischenzuspeichern braucht, wenn das Eingabefenster verkleinert wird.

1.3.2 Befehlsverarbeitung des CLI

Nach dem Aufruf des CLI haben Sie ein CLI-Fenster mit einem Cursor zur Verfügung. Das CLI selbst ist eigentlich, kurz gesagt, "dumm". Es kennt keine eigenen Befehle, sondern kann im Prinzip nur Programme starten. Sie geben nur den Programmnamen an, und das CLI sucht das Programm auf der eingelegten Diskette, lädt und startet es. Der Trick dabei ist, daß alle DOS-Befehle des CLI, die man beispielsweise benötigt, um sich das Inhaltsverzeichnis einer Diskette anzusehen, selbst Programme sind. Wenn der Befehl dir eingegeben wird, lädt das CLI zuerst das Programm mit dem Namen dir in den Speicher und ruft es auf. Dieses Programm wiederum zeigt dann das Directory der aktuellen Diskette.

Eine kurze Anmerkung zwischendurch. Klein- und Großschrift können beliebig vertauscht werden. Dies gilt sowohl für Befehlsaufrufe als auch für die dahinter angegebenen Werte. Zum Beispiel passiert bei dir das gleiche wie bei Dir oder DIR.

Ein Fehler, der fast jedem passiert, der das erste Mal mit dem CLI arbeitet, ist der folgende: Angenommen, Sie wollen sich das Inhaltsverzeichnis einer anderen als der Workbench-Diskette anschauen. Sie legen also diese Diskette ins Laufwerk und geben den Befehl dir ein. Anstatt daß der Computer Ihnen aber nun das Directory der gewünschten Diskette anzeigt, fordert er Sie auf, die Workbench-Disk einzulegen. Wenn Sie das tun, erhalten Sie statt des Inhaltsverzeichnisses, das Sie sehen wollten, das der Workbench-Disk. Auch die Eingabe von dir df0: führt zu keinem anderen Ergebnis. Woran das liegt, erkennt man sofort, wenn man sich überlegt, was eigentlich bei Eingabe des dir-Befehls vor sich geht.

Der Amiga merkt sich ein sogenanntes aktuelles Verzeichnis, (engl. current directory), in dem zuallererst nach einem Befehl gesucht wird. Dieses Verzeichnis ist normalerweise entweder allgemein DF0: oder das Haupt-Directory der Workbench-Diskette. Der Rechner sucht nun bei Eingabe von dir innerhalb des aktuellen Verzeichnisses nach einem Programm mit dem Namen dir. Da er ein solches Programm nicht finden kann, versucht er es noch im Unter-Directory c auf der Workbench-Disk, weil dort normalerweise alle DOS-Befehle stehen. Unter-Directories werden gleich noch erklärt. Wichtig ist nur, daß in dem Moment, wo der Befehl dir geladen und gestartet wird, sich die Workbench-Diskette im Laufwerk befindet. Da nicht genau angegeben wurde, von welcher Diskette das Directory angezeigt werden soll, sondern nur das Laufwerk, erscheint auf dem Bildschirm das falsche Inhaltsverzeichnis. Gibt man statt dessen dir Diskettenname: ein, will der Rechner zwar zuerst schon wieder die Workbench haben, fordert aber nach dem Laden von dir dazu auf, die richtige Diskette einzulegen.

Was macht man aber, wenn man den Diskettennamen nicht kennt? Man kann schließlich nicht von jedem erwarten, daß er die Namen aller seiner Disketten im Kopf hat. Wenn man zwei Diskettenlaufwerke besitzt, ist das alles kein Problem. Man legt die Workbench-Disk in Laufwerk df0, die andere Diskette in Laufwerk df1 und gibt ein: dir df1:. Wer aber nur eine Diskettenstation besitzt, muß sich anders behelfen. Ihr Amiga enthält nämlich ein quasi von der Software geschaffenes Gerät, das sich

wie ein Diskettenlaufwerk verhält: die RAM-Disk. Es handelt sich dabei um einen Teil des Speichers, der mit den normalen Diskettenbefehlen verwaltet werden kann. Man muß nur statt der Laufwerksbezeichnung df0: die Bezeichnung ram: verwenden. In der RAM-Disk lassen sich beliebig Programme und Daten ablegen und wieder löschen, ganz genau wie bei einer richtigen Diskettenstation.

Wie löst die RAM-Disk nun unser Problem? Geben Sie folgendes ein: copy c/dir ram:. Dadurch wird der Befehl dir in die RAM-Disk kopiert. Legen Sie nun die Diskette ein, deren Inhaltsverzeichnis Sie interessiert. Durch ram:dir rufen Sie den Befehl dir aus dem RAM heraus auf. Sollten Sie jetzt immer noch die Aufforderung zum Einlegen der Workbench-Disk erhalten, versuchen Sie es mit ram:dir df0:. Jetzt funktioniert aber alles mit Sicherheit.

Die RAM-Disk hatten wir auch schon im Kapitel 1.1 in Verbindung mit der Workbench angesprochen. Falls sich Ihre Workbench nicht mit einem RAM-Disk-Icon meldet, so können Sie die entsprechende Startup Sequence (siehe Kapitel 1.3.4) abändern und den Befehl dir ram: einbauen. Beachten Sie, daßß das beim Amiga 1000 nur mit der Kickstart 1.2 möglich ist.

Sie finden das sicherlich alles sehr umständlich im Vergleich zu dem, was Sie bisher von Ihrem alten Rechner her kannten. Zweifellos war es einfacher, nach dem Einschalten des Rechners direkt LOAD$,8 tippen zu können, ohne erst mit mehreren Disketten herumhantieren zu müssen. Trotzdem kann man es auch beim Amiga so einrichten, daß man sich das andauernde Diskettenwechseln erspart. Bevor wir das weiter erklären, sollten Sie vorher noch etwas über Unter-Directories erfahren.

1.3.3 Ablage von Files in Ordnern

Wenn Sie ein wenig mit der Workbench herumexperimentiert haben, sollten Sie eigentlich schon Ordner, auch genannt Drawer, Unter-Directories oder Schubladen, kennengelernt haben. Beispielsweise gehört der System-Ordner dazu, den Sie von

der Workbench aus anklicken, um das CLI zu aktivieren. Ein Ordner ist ein Name, unter dem Programme, Daten und weitere Ordner abgelegt sein können. Jeder Ordner verhält sich demnach genau wie das Haupt-Directory der Diskette.

Wie sahen denn Ihre Disk-Inhaltsverzeichnisse auf Ihren C64-/C128-Disketten aus? Waren die verschiedenen Files, von denen jeweils mehrere zu einem Programm gehörten, wild im Directory verteilt, oder haben Sie sich die Mühe gemacht, die Dateien der Reihenfolge nach zu sortieren, nach Möglichkeit sogar noch durch Trennstriche deutlich sichtbar unterschieden? Wenn letzteres der Fall war, dann sollten Sie eigentlich das Handikap bemerkt haben, das auftritt, wenn man beim C64/C128 versucht, eine Datei innerhalb einer ganz bestimmten Gruppe von schon bestehenden Files unterzubringen. Um dann Ordnung zu schaffen, benötigt man unbedingt ein Directory-Sortierprogramm, mit dem man regelmäßig für die gewünschte Reihenfolge sorgen mußte. Die Ordner auf dem Amiga kennen keine solchen Probleme. Sie gruppieren zusammengehörige Files problemlos mit der Unterstützung aller Betriebssytemfunktionen.

Mit dem Befehl dir df0: erhält man neben den Files, die sich im Haupt-Directory der Diskette befinden, zudem noch alle Ordner innerhalb dieses Directories, gekennzeichnet durch (dir). Jede Schublade in der Workbench hat eine Entsprechung als Unter-Directory, die man vom CLI aus erreichen kann. Wollen Sie den Inhalt eines solchen Ordners erfahren, so geben Sie den Befehl dir Ordnername ein. Ein File innerhalb eines Ordners erreichen Sie über die Bezeichnung Ordnername/Filename. Geben Sie beispielsweise dir c ein. Sie erhalten eine Auflistung aller Files im Ordner c. Hierbei handelt es sich um sämtliche DOS-Befehle. Beispielsweise können Sie den dir-Befehl auch durch c/dir oder df0:c/dir aufrufen.

Ein wichtiger Befehl im Zusammenhang mit Ordnern ist cd. Der Amiga verwaltet nämlich ein aktuelles Diskverzeichnis (current directory). Dieses Verzeichnis ist entweder ein Unter-Directory oder das Haupt-Directory einer Diskette. Durch cd Ordnername oder cd df0: beziehen sich jetzt alle Befehle auf das entsprechende Directory. Das bedeutet, daß alle Dateien, die in

irgendwelchen Befehlen verwendet werden, im festgelegten Directory gesucht oder abgelegt werden. Man kann das dadurch unterdrücken, daß man den vollständigen File-Namen, begonnen mit dem Gerät, das man ansprechen will, angibt. Ein Beispiel: Nach cd devs erhält man nach Eingabe von dir den Inhalt des Ordners devs, nach Eingabe von dir df0: jedoch wie gewohnt das Haupt-Directory. dir c versucht nach obigem cd-Befehl, das Unter-Directory c im Ordner devs anzuzeigen. Da sich c aber nicht innerhalb von devs befindet, sondern im Haupt-Directory, führt dies zu einer Fehlermeldung. Korrekt funktioniert dagegen der Befehl dir df0:c.

Cd / wechselt von einem Unter-Directory in das darüberliegende Directory, cd // geht zwei Ebenen hinauf und so weiter.

Cd hat aber einen Nachteil, den man kennen sollte. cd ohne Angabe eines Parameters zeigt das aktuelle Directory an. Sofern man das CLI nicht auf dem Umweg über die Workbench gestartet hat und das aktuelle Directory noch nicht verändert wurde, führt der Aufruf von cd zur Anzeige DF0:. Nach cd df0: liefert die Eingabe von cd jedoch die Anzeige Diskettenname:. Vorher war das aktuelle Diskverzeichnis immer das der gerade eingelegten Diskette, nachher muß man bei jedem Diskettenwechsel immer den cd-Befehl neu aufrufen. Es gibt keine Möglichkeit, wieder als aktuelles Verzeichnis allgemein df0: anzuwählen. Wenn Sie zum Beispiel den dir-Befehl ins RAM kopiert haben, sich nun mit ram:dir das Inhaltsverzeichnis einer Diskette ansehen wollen und der Rechner trotzdem die Workbench-Disk verlangt, dann liegt das daran, daß das aktuelle Diskverzeichnis nicht mehr df0: lautet.

1.3.4 Erstellen einer CLI-Arbeitsdiskette

Gerade wenn man nur ein Laufwerk besitzt, ist es sehr lästig, wenn man jedesmal nur über die Workbench ins CLI kommt und dann erst die Befehle, die man benötigt, ins RAM kopieren muß. Es wäre doch ganz praktisch, wenn der Rechner automatisch ins CLI springen würde und dabei alle Befehle, die man

regelmäßig braucht, in die RAM-Disk laden würde. Dann könnte er zudem noch dem DOS mitteilen, wo es die Befehle suchen soll (man will ja nicht immer ram: davor schreiben), und auch noch alles weitere erledigen, was Sie gerne von ihm vor dem CLI-Start erledigt hätten. Die Lösung des Problems ist die Erstellung einer CLI-Arbeitsdiskette, die Sie anstelle der Worbench-Disk verwenden können.

Wie erstellt man sich eine solche CLI-Disk? Man beginnt natürlich damit, daß man sich eine weitere Kopie der Workbench-Disk erstellt. Also: Workbench-Icon einmal anklicken und Menüpunkt Duplicate auswählen. Dann ruft man das CLI auf und benutzt wie im weiteren erklärt den ed-Befehl.

Im Ordner s der Workbench-Diskette befindet sich ein File mit dem Namen Startup Sequence. Es enthält eine Liste all der Befehle, die direkt während des Boot-Vorgangs, also während des Einlesens von der ersten Diskette nach einem Reset, ausgeführt werden. Mit ed s/startup-sequence ruft man den Texteditor des AmigaDOS auf und kann nun die startup-sequence problemlos ändern. Wie Sie mit dem ed-Editor arbeiten und welche Befehle er ihnen zur Verfügung stellt, finden Sie in Ihrem Handbuch. Für das Ändern der startup-sequence genügt es aber im Prinzip zu wissen, daß man mit ESC-X den ed wieder verläßt und das bearbeitete File abspeichert, während ESC-Q zu einem Verlassen ohne Abspeichern dient. Zudem sollte man noch die Funktion Ctrl-B kennen, die eine Zeile löscht. Mit diesen drei Tastenkombinationen kommt man im allgemeinen aus.

Sind Sie nun im Editor und haben Sie die startup-sequence vor sich, so entfernen Sie zuerst die Zeilen mit den Befehlen loadwb und endcli. Loadwb ist der Befehl, der die Workbench startet und endcli derjenige, der das CLI-Fenster schließt. Wenn Sie sowohl CLI als auch Workbench gleichzeitig zur Verfügung haben möchten, können Sie loadwb auch stehenlassen. Damit hätten wir schon eine Diskette, die sofort im CLI bleibt, ohne in die Workbench zu springen.

Der nächste Schritt ist das Kopieren der Befehle in die RAM-Disk. Sie wollen sicherlich nicht alle Befehle in der RAM-Disk

haben, sondern nur diejenigen, die Sie wirklich regelmäßig benötigen. Am besten ist es, wenn Sie zuerst den Befehl copy ins RAM kopieren. Wenn Sie danach nämlich den copy-Befehl aus dem RAM heraus starten, können Sie sich eine Menge Zeit sparen. Bringen Sie also der Reihe nach folgende Befehle im Text unter:

```
copy c/copy ram:
ram:copy c/dir ram:
ram:copy c/delete ram:
...
```

Wenn Sie auf diese Weise alle Befehle kopiert haben, die Sie benötigen, müssen Sie dem Rechner noch mitteilen, daß er sie in der RAM-Disk finden kann. Anderenfalls wären Sie dazu gezwungen, immer ram: vor Ihren Befehlen einzugeben. Welche Befehle es gibt und was sie bewirken, wird noch genauer in Kapitel 2 erläutert.

Wollen Sie, daß der Rechner seine Befehle statt im c-Ordner in der RAM-Disk sucht, dann geben Sie assign c: ram: ein. Wollen Sie dagegen, daß der Rechner, falls er einen Befehl in der RAM-Disk nicht findet, noch im C-Ordner nachsieht, dann verwenden Sie statt dessen path ram:. Hierdurch wird die RAM-Disk in die Befehlssuche miteingegliedert. Falls es Sie interessiert, meine persönliche CLI-Disk enthält folgende Startup-Sequence:

```
echo "A500  CLI-Disk 30.5.1987 23:09 *N"
addbuffers df0: 22
path sys:system
setmap d
failat 30
copy c/copy ram:
ram:copy c/dir ram:
ram:copy c/makedir ram:
ram:copy c/rename ram:
ram:copy c/ed ram:
ram:copy c/type ram:
ram:copy c/delete ram:
ram:copy c/newcli ram:
ram:copy c/endcli ram:
ram:copy c/assign ram:
```

```
ram:copy c/cd ram:
ram:copy c/protect ram:
ram:copy sys:system/format ram:
ram:copy c/list ram:
ram:copy c/info ram:
ram:copy c/relabel ram:
ram:copy c/install ram:
ram:dir RAM:
Path RAM:
```

Diese Startup-Sequence enthält noch einige, hier nicht erklärte Befehle, die im Rahmen dieser kurzen Einführung nicht erläutert werden brauchen. Einiges wurde auch so belassen, wie es in der Original-Sequenz schon vorgegeben war.

Wenn Sie Ihre Startup-Sequence fertig erstellt haben, speichern Sie sie mit ESC und X ab. Falls Sie alles richtig gemacht haben, so können Sie Ihre CLI-Disk nach einem Reset problemlos laden. Sie schieben sie einfach statt der vom Rechner geforderten Workbench-Diskette ins Laufwerk. Sobald alles geklappt hat, können Sie alle Dateien von der Diskette entfernen, die Sie nicht benötigen. Mit delete Dateiname1 Dateiname2 Dateiname3 ... können Sie einzelne Files löschen. Wollen Sie dagegen einen vollständigen Ordner entfernen, so lautet der Befehl delete Ordnername all.

Welche Dateien Sie bedenkenlos entfernen können, wollen Sie wissen? Nun, da wären zunächst einmal alle mit .info endenden Dateien, die nur die Icons enthalten. Ohne Workbench sind diese Dateien natürlich unnötig. Weiterhin werden Sie die Ordner Demos, Empty, Expansion, Utilities und Trashcan auf Ihrer CLI-Disk kaum benötigen. Auch für das Voreinstellungsprogramm Preferences reicht es aus, wenn es sich auf Ihrer Workbench-Disk und nicht auf Ihrer CLI-Disk befindet. Im Ordner devs befindet sich ein Unterordner printers, in dem alle Druckeranpassungen vorhanden sind. Diejenigen, die Sie nicht benötigen, können Sie selbstverständlich löschen.

1.3.5 Weitere wichtige Befehle

Ohne zu weit in die Beschreibung der DOS-Befehle eindringen zu wollen, geben wir Ihnen hier eine kurze Einführung in die wichtigsten Befehle, die man zum Arbeiten unbedingt benötigt.

Den Dir-Befehl kennen Sie ja bereits. Man erfährt mit ihm allerdings nichts über den freien Speicherplatz auf der Diskette und die Länge der einzelnen Files. Der Befehl info verrät einem unter anderem, wieviel Bytes auf der eingelegten Diskette schon belegt und wie viele noch frei sind. Außerdem erfährt man den Diskettennamen. list zeigt wie dir das Inhaltsverzeichnis der Diskette an, nur nicht in sortierter Form, dafür aber mit den Längen der Dateien. Übrigens, wenn Sie den Inhalt aller Ordner und Unterordner auf einer Diskette erfahren wollen, so probieren Sie es doch einmal mit dir opt a.

Gerade zur Arbeit mit Public-Domain-Software (Software ohne Urheberrechtsschutz) kommt man nur selten ohne eine CLI-Disk aus. Solche Software hat fast immer eine mitgelieferte Anleitung, die als Datei auf der Diskette steht. Dateien, die Anleitungen enthalten, tragen meistens den Namen ReadMe, Liesmich oder eine ähnliche Bezeichnung. Man kann sich solche Dateien natürlich dadurch ansehen, daß man Sie mit dem Befehl ed in den Speicher holt. Oftmals genügt aber auch der Anzeigebefehl type. Type File-Name druckt das entsprechende Daten-File Zeichen für Zeichen auf dem Bildschirm aus. Beispielsweise können Sie durch

```
type README
```

die Datei ReadMe lesen. Anhalten kann man den Ausdruck über den Druck einer beliebigen Taste wie zum Beispiel die Leertaste, fortsetzen über die Backapace-(Pfeil links-) oder die Return-(Enter-)Taste. Ein Abbruch ist durch Ctrl-C möglich. Durch type >prt: File-Name wird das entsprechende File auf dem Drucker gelistet. Passen Sie aber auf, daß Sie keine Programm-Files mit type Programmname auszudrucken versuchen, sonst erhalten Sie nur wirre Zeichen auf dem Bildschirm oder Drucker.

Was benötigt man noch zum Arbeiten mit Disketten? Delete und copy kennen Sie auch schon. Sofern Sie nur ein Laufwerk haben, können Sie natürlich ein Daten-File mit copy so kopieren: Copy Diskname1:File-Name Diskname2:i. Besser ist aber in jedem Fall, die RAM-Disk zu benutzen, weil man sonst viel zu oft die Disketten wechseln muß. Also zuerst copy df0:File-Name ram: und dann copy ram:File-Name df0:. Danach können Sie mit delete ram:File-Name den von der Datei belegten Speicherplatz wieder freigeben. Beim Kopieren läßt sich gleichzeitig der Dateiname ändern. copy alter_Name neuer_Name kopiert die Datei alter_Name, wobei die Kopie den Namen neuer_Name trägt. Passen Sie aber auf, daß im Diskverzeichnis, in das Sie eine Datei hineinkopieren, nicht schon eine Datei gleichen Namens existiert. Diese wird dann nämlich ohne Warnung überschrieben. Beim Arbeiten mit einer 1541/70/71 mußte für das Überschreiben vor Dateinamen ein @: angegeben werden. Eine solche Sicherung kennt der Amiga nicht. Dies gilt nicht nur für den Copy-Befehl, sondern auch für alle anderen Speicherbefehle, egal, ob Sie sich gerade im CLI, im BASIC-Interpreter oder in einer Textverarbeitung befinden.

Wichtig ist außerdem noch rename alter_Name neuer_Name, was dem R:-Befehl der Floppies 1541/70/71 entspricht. Zusätzlich gibt es noch den Befehl relabel alter_Name neuer_Name, der den Diskettennamen ändert. makedir Ordnername erstellt einen neuen Ordner, den man dann für eigene Dateien verwenden kann. Ein Befehl, den man beim C64 oft vermißt hat, ist protect. protect Dateiname schützt eine Datei softwaremäßig vor Schreibzugriffen. Beim C64 war das zwar auch möglich, doch hatte das DOS keinen Befehl, um das Schützen zu ermöglichen. Man mußte also die entsprechende Manipulation quasi von Hand auf der Diskette durchführen. protect Dateiname d hebt den Schreibschutz wieder auf.

Formatieren kann man Disketten mit dem Format-Befehl. Der steht übrigens nicht im C- sondern im System-Ordner. Format Drive df0: Name Diskettenname formatiert eine Diskette im Laufwerk df0. Außerdem wird der Ordner Trashcan angelegt,

was sinnvoll ist, wenn man mit der Diskette unter der Workbench arbeiten will. Der Zusatz NOICONS hinter dem Diskettennamen beim FORMAT-Befehl verhindert dies.

1.3.6 Tips und Tricks beim Arbeiten mit dem CLI

Vielleicht ist Ihnen schon aufgefallen, daß es mit den bisher beschriebenen Befehlen scheinbar nicht möglich ist, Dateinamen zu verarbeiten, die Leerzeichen enthalten. Beim C64 war das ohne weiteres möglich, da alle File-Namen immer in Anführungszeichen standen und Leerzeichen nicht als Trennzeichen verwendet wurden. AmigaDOS akzeptiert aber ohne weiteres auch Dateinamen in Anführungszeichen, man muß es halt nur wissen. 'makedir neuer Ordner' ist also ein durchaus erlaubter und sinnvoller Befehl.

Vermissen Sie Jokerzeichen? Die Floppies 1541/70/71 kennen zwei Jokerzeichen: Das Fragezeichen, das für ein beliebiges anderes Zeichen steht, und den Stern, der eine beliebige Folge von Zeichen ersetzt. Mit diesen Zeichen kann man im Prinzip zwei Sachen machen. Es ist möglich, beim Load- und Verify-Befehl das erste Programm suchen zu lassen, das dem angegebenen File-Namen entspricht. Beispielsweise wird durch Load"D*",8 das erste Programm geladen, das mit dem Buchstaben D beginnt. Der S:...-Befehl (Scratch) des DOS dagegen löscht mit Open 1,8,15,"S:D*" nicht nur das erste Programm, sondern alle Programme, die mit dem Buchstaben D beginnen.

Jokerzeichen kennt AmigaDOS ebenfalls, allerdings nur in der Bedeutung, daß damit mehrere Dateien gleichzeitig erfaßt werden können. Es ist demnach nicht möglich, ein Programm, das man durch Angabe des Namens starten will, durch Jokerzeichen abzukürzen. Man kann aber ohne weiteres bei den Befehlen delete oder copy Jokerzeichen verwenden. Hier nun die wichtigsten in Kürze:

Das Fragezeichen hat genau die gleiche Bedeutung wie beim C64. Es ersetzt ein einzelnes Zeichen. Das Nummernzeichen # steht für null- oder mehrfaches Auftreten des nachfolgend an-

gegebenen Zeichens oder Musters. Beispielsweise werden durch #A die Dateien A, AA, AAA usw. bezeichnet. Sinnvoll ist die Kombination #?, die für eine beliebige Anzahl beliebiger Zeichen steht, also prinzipiell das gleiche macht wie der Stern beim C64. delete D#? löscht demnach alle Files, die mit dem Buchstaben D beginnen. Beim Amiga ist allerdings nun auch folgendes erlaubt: Delete #?.info löscht alle Files, die die Endung .info haben.

Will man Jokerzeichen beim List-Befehl verwenden, so muß das in folgender Weise geschehen: List p dateiname. Anderenfalls sucht der List-Befehl nach einer Datei, die wirklich das Fragezeichen oder das Nummernzeichen im Namen enthält. Um diese beiden Zeichen auch bei anderen Befehlen innerhalb von Dateibezeichnungen zu verwenden, setzt man das Hochkomma ' davor. Dadurch wird die Joker-Bedeutung aufgehoben.

Zum Arbeiten reicht diese kurze Beschreibung der Jokerzeichen. Zwar kennt der Amiga noch einige mehr, aber diese benötigt man so gut wie nie.

Kennen Sie schon den Assign-Befehl? Mit ihm ist es möglich, neue Gerätenamen zu definieren. AmigaDOS trifft keine direkte Unterscheidung zwischen Gerätenamen wie df0:, Diskettennamen wie Workbench_1.2: oder selbstdefinierten Namen wie C:. Man kann mit dem Assign-Befehl Disketten, Ordner, aber auch Dateien mit Gerätenamen ausstatten. Das Betriebssystem hat schon einige Namen vordefiniert, die sich durch Eingabe von assign ohne Parameter abfragen lassen. Beispielsweise werden DOS-Befehle nicht direkt im C-Ordner gesucht, sondern auf dem Gerät C:, das wiederum nichts anderes ist als der C-Ordner. Sie können nun hingehen, und mit assign C: df0:neuerOrdner einen anderen Ordner definieren, der als Befehlsordner fungiert.

Interessanter wird der Assign-Befehl, wenn man zwei Programme von verschiedenen Disketten auf einer einzige zusammenlegen will. Oft laden diese Programme nämlich Dateien unter Angabe des Diskettennamens nach. Da aber eine Diskette normalerweise nur einen Namen haben kann, benötigt jedes Programm, das so arbeitet, eine eigene Diskette, richtig? Falsch!

Mit dem Assign-Befehl definiert man einfach einen zweiten Diskettennamen, so daß die Diskette unter beiden Namen erreichbar ist. Heißt eine Diskette zum Beispiel Beckermat, und Sie wollen ein darauf befindliches Programm, das den Disknamen zum Nachladen verwendet, mit auf eine andere Diskette überspielen, so bauen Sie wie beschrieben in die Startup Sequence den Befehl assign Beckermat: df0: ein. Jetzt läßt sich die Diskette auch unter dem Namen Beckermat erreichen. Wie Sie sehen, nimmt der Assign-Befehl keinerlei Namensveränderungen auf der Diskette vor, sondern nur im Speicher.

Noch ein Trick zum Assign-Befehl. Sollten Sie einen Amiga 2000 mit einem internen und einem externen Laufwerk besitzen, so trägt das externe Laufwerk die Bezeichnung df2: und nicht df1:. Letztere ist nämlich dem zweiten internen Laufwerk vorbehalten. Manchmal kann es aber ganz nützlich sein, ein Laufwerk df1: zur Verfügung zu haben. Legen Sie dazu die Diskette, mit der Sie in df2: arbeiten wollen, in dieses Laufwerk, und geben Sie ein: assign df1: df2:. Dadurch wird dem Amiga vorgegaukelt, er hätte ein Laufwerk df1: zur Verfügung. Dieser Trick macht manchmal Schwierigkeiten, da sich Laufwerk df1: nur mit der beim Assign-Befehl eingelegten Diskette betreiben läßt.

Haben Sie sich schon einmal gefragt, warum das CLI seine Befehle gerade aus dem Ordner c holt? Dann geben Sie doch den Befehl path ein. Auf dem Bildschirm erscheint dann der sogenannte Suchpfad, das heißt alle Verzeichnisse, in denen nach einem Befehl gesucht wird. Zuoberst steht dort immer Current directory, also das aktuelle Verzeichnis. Darunter folgt dann das Gerät C:, das ja, wie Sie inzwischen wissen, nichts anderes als der Ordner c ist. Sie können nun weitere Verzeichnisse in den Suchpfad eingliedern, zum Beispiel mit path RAM:. Dadurch sucht das CLI nach einem eingegebenen Befehl nicht nur im aktuellen Verzeichnis und im C-Ordner, sondern auch noch auf der RAM-Disk. Dabei werden zusätzliche Verzeichnisse immer zwischen Current directory und C: eingefügt. Zurücksetzen läßt sich der Suchpfad durch path reset, wodurch alle zusätzlichen Verzeichnisse wieder aus der Liste gestrichen werden.

1.3.7 Multitasking

Ein Schlagwort, das man beim Amiga oft hört, ist Multitasking. Damit ist das gleichzeitige Abarbeiten mehrerer Programme gemeint. Was kann man damit vom CLI aus anfangen?

Der Befehl newcli eröffnet ein neues CLI-Window. Man hat nun zwei Fenster, in denen man DOS-Befehle ausführen oder Programme starten kann. Zwischen beiden Fenstern kann man hin- und herschalten, indem man jeweils mit der Maus in das entsprechende Fenster hineinklickt. So können Sie sich beispielsweise ein Directory einer Diskette im ersten CLI-Window anzeigen lassen und danach auf das zweite Fenster umschalten, damit Ihnen das Directory bei weiteren Befehlen nicht wieder verlorengeht. Schließen läßt sich das Fenster mit dem Befehl endcli.

Eine interessante Eigenschaft der beiden Fenster ist es, daß Programme, die in einem Fenster gestartet wurden, gleichzeitig mit denen des anderen Fensters ablaufen können. Erstellen Sie doch beispielsweise mit newcli ein zweites CLI-Window, und lassen Sie sich mit dir das Directory anzeigen. Wenn Sie schon auf das zweite Fenster umschalten, ohne abzuwarten, bis der Befehl vollständig ausgeführt wurde, so werden Sie bemerken, daß Sie Befehlseingaben machen können, während gleichzeitig der Dir-Befehl weiterarbeitet. Oder Sie wollen eine lange Textdatei auf dem Drucker ausgeben. Wenn Sie mit zwei CLI-Fenstern arbeiten, so können Sie problemlos weiterarbeiten, auch während der Type-Befehl noch einige Minuten beschäftigt ist. Am geschicktesten gehen Sie dabei so vor, daß Sie zuerst das File in die RAM-Disk kopieren, also copy Filename ram:, und dann den Befehl type >prt: ram:Filename aufrufen. So können Sie bedenkenlos die Diskette aus dem Laufwerk entfernen, ohne daß der Type-Befehl Ihnen Schwierigkeiten bereitet.

Es gibt noch einen anderen Befehl, der das Multitasking unterstützt. Er lautet run und kann einen oder mehrere Befehle hintereinander laden und ausführen. Dabei wird allerdings kein neues Fenster eröffnet, was sowohl Vor- als auch Nachteile hat. Nach run dir df0: beispielsweise wird das Directory angezeigt, wobei aber der Cursor eingabebereit bleibt. Man kann dann

mitten in der Befehlsausführung einen anderen Befehl eingeben, beispielsweise list df0:. Dies würde aber zu einem ziemlichen Durcheinander führen, da beide Befehle, sowohl list als auch dir, ihre Bildschirmausgaben im selben Fenster tätigen würden. Interessanter ist dagegen eine Anwendung wie run type >prt: Filename. Da dieser Befehl keine Daten auf dem Bildschirm ausgibt, benötigt er auch kein eigenes Fenster. Außerdem ist es wesentlich einfacher, diese Run-Anweisung einzugeben, als wenn man mit den Befehlen newcli und endcli herumhantieren und zwischen verschiedenen Fenstern umschalten muß.

Eine Möglichkeit des Run-Befehls sollten Sie noch kennengelernt haben, bevor wir dieses Kapitel beenden. Man kann nämlich mehrere Befehle oder Programme mit einem einzigen Befehlsaufruf hintereinander ablaufen lassen. Dazu geben Sie folgendes ein:

```
run Befehl1+
Befehl2+
...
Befehlx
```

Der Run-Befehl testet jedesmal, ob sich am Schluß eines ihm übergebenen Befehls ein +-Zeichen befindet. Falls ja, wartet er mit der Befehlsausführung, bis Sie einen Befehl ohne das Plus-Zeichen eingeben. Erst dann arbeitet er einen Befehl nach dem anderen ab.

1.4 Workbench-Hilfen

Während wir in Kapitel 1.1 nur kurz das Wichtigste über die Workbench zusammengefaßt haben, wollen wir in diesem Kapitel einen tieferen Einblick nehmen und uns insbesondere mit den mitgelieferten Hilfsprogrammen auseinandersetzen. Dabei setzen wir voraus, daß Sie sich schon ein wenig mit Icons, Windows und Pull-Down-Menüs angefreundet haben.

1.4.1 Voreinstellungen mit Preferences

Ein sehr hilfreiches Programm ist Preferences. Mit ihm können Sie alle möglichen Einstellungen vornehmen, angefangen von den Bildschirmfarben über die Mausgeschwindigkeit bis hin zum Druckertreiber. Die damit eingestellten Werte können Sie dann abspeichern, wobei sie im (nur vom CLI aus sichtbaren) Ordner devs im Daten-File System-Configuration abgelegt werden. Beim Booten einer Diskette, die diese Datei im richtigen Ordner enthält, werden dann automatisch alle Voreinstellungen getroffen. Das gilt unabhängig davon, ob die Programme auf der Diskette mit der Workbench zusammenarbeiten oder nicht.

Beim Starten des Preferences-Programmes meldet sich zuerst das Hauptfenster, von dem aus sich schon das meiste einstellen läßt. Durch die konsequente Verwendung von sinnvollen Symbolen erklären sich viele Funktionen von selbst.

Wenn Sie alles, was Sie ändern wollten, geändert haben, gibt es drei Möglichkeiten, das Programm wieder zu verlassen. Entweder Sie klicken das Feld Save an, wodurch die Einstellungen abgespeichert werden und nach jedem Einladen wieder zur Verfügung stehen oder Sie benutzen Use, wodurch die Einstellungen zwar erhalten bleiben, jedoch nicht abgespeichert werden. Wenn Sie dagegen alles verwerfen wollen, was Sie bisher eingestellt haben, so wählen Sie die Option Cancel. Prinzipiell das gleiche findet auch beim Feld Last Saved statt. Hier wird auch alles auf den letzten abgespeicherten Stand zurückgesetzt, Sie bleiben allerdings im Preferences-Programm. Auch für den Fall, daß Sie eine ungewünschte Voreinstellung versehentlich abgespeichert haben, bietet das Programm Ihnen eine Möglichkeit: Reset All schaltet alle Einstellungen auf den Wert zurück, der auf der unveränderten Original-Workbench-Diskette eingestellt war.

Beginnen wir mit der Farbeinstellung. Das Betriebssystem des Amiga stellt einem normalerweise vier verschiedene Farben zur Verfügung. Im unteren Bereich des Fensters sehen Sie vier verschiedenfarbige Quadrate, von denen immer eines umrandet ist. Die umrandete Farbe kann jeweils verändert werden. Jede Farbe

besteht aus drei Grundfarben, nämlich Rot, Grün und Blau. Unter den Quadraten befinden sich Balken, mit denen Sie die Anteile der Grundfarben in der entsprechenden Farbe einstellen können. Durch Anklicken des Rechtecks Reset Colors stellen Sie wieder die Originalfarben ein.

Der Schalter CLI bestimmt, ob beim Öffnen des System-Ordners ein anklickbares CLI-Icon erscheinen soll oder nicht. Die Option Text dient zum Umschalten zwischen den beiden Darstellungsarten 60-Zeichen-pro-Zeile und 80-Zeichen-pro-Zeile. 60-Zeichen sollten Sie dann wählen, wenn Sie keinen Monitor, sondern nur einen Fernseher besitzen.

Datum und Uhrzeit stellen Sie mit den danebenstehenden Pfeilsymbolen ein. Klicken Sie zuerst die entsprechende Stelle beziehungsweise Ziffer des Datums oder der Zeit an, die Sie ändern wollen. Danach dienen die Pfeile zum Herauf- und Herunterzählen.

Die Balken Key Repeat Delay und Key Repeat Speed dienen zur Regelung der Wiederholfunktion der Tastatur. Key Repeat Delay ist die Zeit, die man eine Taste gedrückt halten muß, bis die Tastaturwiederholfunktion einsetzt, und Key Repeat Speed die Geschwindigkeit, mit der die Zeichen wiederholt werden. Am einfachsten regeln Sie diese Werte, indem Sie vor Aufruf von Preferences das CLI-Fenster öffnen und es in den Bildschirmbereich unterhalb des Preferences-Fensters schieben. Jetzt können Sie sofort ausprobieren, wie sich die Tastatur verhält. Denken Sie aber immer daran, vor irgendwelchen Tastatureingaben in das CLI-Fenster hineinzuklicken, da sonst nämlich dort nichts passiert. Am Ende schließen Sie das CLI-Fenster wieder durch endcli.

Der Schalter Workbench Interlace erhöht die vertikale Bildschirmauflösung auf das Doppelte. Da dies aber auf einem normalen Monitor je nach Farbeinstellung zu einem leichten bis stärkeren Flackern führt, sollten Sie diese Option nur dann nutzen, wenn Sie einen Monitor mit langer Nachleuchtdauer be-

nutzen. Interlace wird auch nur dann wirksam, wenn man diese Schalterstellung abspeichert und danach die Workbench neu bootet.

Zur Maus gibt es zwei Voreinstellungsmöglichkeiten. Der linke Balken mit den Zahlen 1, 2 und 4 regelt die Geschwindigkeit, mit der sich der Mauszeiger über den Bildschirm bewegt. Dabei ist 1 die höchste Geschwindigkeit mit der geringsten Genauigkeit, während sich bei 4 der Zeiger am genauesten bewegen läßt, was aber auch den meisten Platz auf dem Tisch beansprucht. Der rechte Balken daneben bestimmt den maximalen zeitlichen Abstand, den zwei Klicks voneinander haben dürfen, damit sie noch als Doppelklick erkannt werden.

Das Feld in der Mitte dient zum Zentrieren des Bildschirms. Verschieben Sie einfach das darin enthaltene Symbol wie ein Icon, bis der Bildschirm zentriert ist.

Change Serial liefert Ihnen einen neuen Fensteraufbau mit Parametern zum Betrieb der seriellen (RS232-) Schnittstelle. Da Sie, wenn Sie die serielle Schnittstelle nutzen wollen, sich mit deren Parametern wohl schon auskennen, sollte Ihnen klar sein, was die einzelnen Werte bedeuten. Abschließend können Sie die Einstellungen mit OK oder Cancel. Cancel macht dabei alle Änderungen wieder rückgängig.

Change Printer erlaubt Ihnen Zugriff auf die Druckervoreinstellungen. Wählen Sie zuerst Ihren Druckertyp. Sollte sich Ihr Drucker nicht unter den angegebenen befinden, probieren Sie es mit Epson und Generic. Links neben der Druckerauswahl können Sie festlegen, ob Sie Ihren Drucker über den parallelen (Centronics-) Port oder über den seriellen (RS232-) Port betreiben.

Paper Size (Papiergröße) führt zu keinerlei Problemen, wenn Sie Custom wählen und als Length je nach Blattlänge 66 oder 72 angeben. Dabei wird ein Spacing-Wert von 6 lpi (6 Zeilen pro Inch) vorausgesetzt. Die Anzahl der Zeichen in jeder Zeile wird sowieso durch die Einstellungen Left Margin (linker Rand) und Right Margin (rechter Rand) bestimmt. Die hier verwendeten

Maximalwerte hängen von der Pitch (der Zeichendichte) ab. Bei einer Standard-Papierbreite steht 10-Pica für bis zu 80 Zeichen, 12-Elite für bis zu 96 Zeichen und 15-Fine für bis zu 136 Zeichen pro Zeile.

Paper Type unterscheidet zwischen Fanfold (Endlospapier) und Single (Einzelblatt). Quality meint entweder Draft-Qualität, also normaler Ausdruck, oder Letter, also NLQ oder LQ (Briefqualität).

Mit Graphic Select stellen Sie Werte für den Grafikausdruck ein. Aspect bestimmt, ob eine Grafik normal (Horizontal) oder seitlich um 90 Grad gedreht (Vertical) gedruckt werden soll. Image steht für normalen (Positive-) Ausdruck oder invertierten (Negative-) Druck. Shade wählt zwischen Farbausdruck (Color) auf einem Farbdrucker oder Grauschattierungen (Gray Scale) zur Farbimitierung. Eine dritte Möglichkeit ist Black and White (Schwarz/Weiß), wodurch alle hellen Farben weiß und alle dunklen Farben schwarz gedruckt werden. Bei welcher Helligkeit die Grenze liegt, hängt von Threshold ab. Hier ist der Wert 2 interessant, bei dem nur die dunkelste Farbe auf dem Bildschirm schwarz wird.

Eine letzte Option aus dem Hauptfenster haben wir noch ausgelassen, nämlich Edit Pointer. Wenn Ihnen das Aussehen des Mauszeigers nicht gefällt, wählen Sie diesen Punkt. Sie haben dabei vier Farben zur Auswahl, von denen Sie die drei Vordergrundfarben verändern können. Innerhalb des Zeigers muß ein Punkt angegeben werden, der die Pfeilspitze kennzeichnet. Dies geschieht mit Set Point. Ok und Cancel haben wieder die übliche Funktion.

Normalerweise können Sie mit dem Preferences-Programm nur die Voreinstellungen einer Diskette ändern, auf der sich dieses Programm auch befindet. Was hindert uns aber, auch auf Boot-Disketten, die keine Workbench geschweige denn Preferences zur Verfügung stellen, trotzdem das File System-Configuration anzulegen? Die folgende Beschreibung geht davon aus, daß Sie nur ein Diskettenlaufwerk besitzen.

Für die Änderung wird das CLI benötigt. Falls Sie damit noch unsicher sind, werfen Sie doch einmal einen Blick in Kapitel 1.3. Öffnen Sie also zuerst von der Workbench ein CLI-Fenster und kopieren Sie die Befehle dir, assign und makedir in die Ram-Disk, außerdem das Preferences-Programm:

```
copy c/dir ram:
copy c/assign ram:
copy c/makedir ram:
copy preferences ram:
```

Danach schieben Sie die Diskette, auf der die Konfigurationsdatei angelegt werden soll, ins Laufwerk. Kontrollieren Sie zunächst mit ram:dir df0:, ob sich auf der Diskette schon ein Ordner mit dem Namen devs befindet. Wenn nein, dann erstellen Sie halt mit ram:makedir df0:devs einen. Als nächstes geben Sie folgendes ein:

```
ram:assign DEVS: df0:devs
ram:assign SYS: df0:
```

Jetzt können Sie endlich Preferences durch den Befehl ram:preferences starten. Falls Sie ein schon bestehendes Konfigurations-File verändern wollen, wählen Sie zuerst das Feld Last Saved an. Nachdem Sie mit allen Einstellungen fertig sind, können Sie mit Save das Programm beenden. Damit sind Sie fertig und können die Diskette booten.

Nehmen Sie solche Änderungen nur auf Disketten vor, von denen Sie sich sicher sind, daß sie normal mit dem AmigaDOS zusammenarbeiten. Falls hier Zweifel bestehen, so fertigen Sie zuerst eine Sicherheitskopie der entsprechenden Diskette an und testen hinterher, ob alle Programme auf der Diskette noch richtig funktionieren. Natürlich müssen auf der Diskette noch genügend Bytes frei sein. System-Configuration kommt mit einem einzigen Datenblock aus.

1.4.2 Die Uhr

Die im Preferences eingestellte Uhrzeit wäre relativ sinnlos, wenn es keine Uhr gäbe. Durch einen Doppelklick auf das Clock-Icon wird sie aktiviert. Die Uhr besitzt eigene Pull-Down-Menüs. Man kann zwischen diesen Menüs und den Workbench-Menüs umschalten, indem man entweder in das Clock-Fenster oder in den Workbench-Hintergrund mit der Auswahltaste hineinklickt.

Die Uhr-Menüs erklären sich selbst. Sie enthalten eine Umschaltung zwischen Analog- und Digitalanzeige, eine Umschaltung zwischen 12- und 24-Stundenanzeige und eine Alarmzeiteinstellung. In der linken unteren Ecke des Fensters ist ein Gadget, das den Sekundenzeiger ein- und ausblendet.

Die Uhr läuft aufgrund der Multitasking-Fähigkeiten des Amiga im Hintergrund vieler Programme problemlos mit. Der Alarm bleibt leider nur wirksam, solange das Uhrenfenster geöffnet ist.

1.4.3 Der Notizblock

Der Notizblock (Notepad) befindet sich im Ordner Utilities. Es handelt sich dabei um ein kleines Textprogramm, das die Möglichkeit zum Mischen von Schriften und Schriftarten enthält. Es eignet sich besonders gut, Anleitungen, Hilfestellungen und Tips zu Programmen, die unter der Workbench laufen, auf einer Diskette unterzubringen. Für längere Texte oder Programmtexte fehlen dem Notizblock allerdings einige wichtige Eigenschaften.

Auch der Notizblock verwendet eigene Menüs, aus denen Sie so ziemlich alle wichtigen Funktionen abrufen können. Einige Menüpunkte haben am Ende ein Amiga- und ein weiteres Zeichen. Dadurch werden sogenannte shortcuts (Abkürzungen) angezeigt, die für eine Tastenkombination stehen, die statt eines Menüaufrufs verwendet werden kann.

Nachdem Sie einen Text fertiggestellt haben und mit Save oder Save As abgespeichert haben, erhält dieser ein eigenes Icon.

Sollten Sie das Fenster des Ordners geöffnet haben, der das Icon enthält, so verändert sich zunächst nichts. Erst wenn Sie dieses Fenster schließen und dann wieder öffnen, ist das Icon zu sehen. Beim doppelten Anklicken wird dann der komplette Notizblock geladen, und zwar in der Größe, mit der man das Dokument abgespeichert hat.

Blättern kann man im Text mit den beiden Gadgets links unten und rechts oben. Das Symbol links unten schaltet eine Seite weiter, während das Symbol rechts oben, das auch die Seitennummer enthält, eine Seite zurückgeht.

1.4.4 Icons und Icon-Editor

Mit Icons haben Sie ja inzwischen schon oft genug gearbeitet. In diesem Kapitel erfahren Sie etwas über das Erstellen eigener Icons und das Abändern bestehender.

Beginnen wir mit dem Festlegen der Position, an der ein Icon erscheinen soll. Ordnen Sie dazu alle Icons in einem Fenster durch Verschieben mit der Maus so an, wie Sie es für sinnvoll halten. Wählen Sie dann mit gedrückter Shift-Taste alle Icons, deren Position Sie verändert haben, an. Nun rufen Sie aus dem Menü Special den Menüpunkt Snapshot auf. Dadurch wird die Position auf der Diskette abgespeichert.

Wenn Sie Kapitel 1.3 aufmerksam gelesen haben, ist Ihnen aufgefallen, daß sich die Icons in Dateien befinden, die den gleichen Dateinamen tragen, wie die Dateien, die sie repräsentieren, nur ergänzt um das Anhängsel .info. Um diese Dateien zu verändern oder zu erzeugen, gibt es das Programm IconEd, den sogenannten Icon-Editor. Er befindet sich im Ordner System auf der Workbench-Diskette.

Bevor Sie mit dem Icon-Editor zu arbeiten beginnen, müssen Sie wissen, daß es fünf verschiedene Typen von Icons gibt, nämlich Disketten-Icons, Ordner-Icons, Programm-Icons, Projekt-Icons und Müll-Icons. Disketten-, Ordner- und Programm-Icons erklären sich schon durch ihren Namen. Projekt-Icons sind zum

Beispiel die Icons der Notepad-Texte. Wenn man so ein Icon anklickt, wird nicht nur der entsprechende Text in den Speicher geladen, sondern auch noch das dazugehörige Programm, wie zum Beispiel Notepad selbst. Müll-Icons werden üblicherweise nur für den Trashcan-Ordner benötigt. Sie unterscheiden sich von normalen Ordner-Icons dadurch, daß man sie nicht in einen anderen Ordner hineinschieben darf.

Nach dem Start des Icon-Editors warnt dieser Sie zuerst einmal in bezug auf die Icon-Typen. Als Vorgabe erhalten Sie das Icon des Icon-Editors selbst. Der Icon-Typ, den sich der Icon-Editor merkt, ist immer der des zuletzt geladenen Icons. Nachdem Sie ein eigenes Icon zusammengebastelt haben, wollen Sie es natürlich abspeichern. Achten Sie dabei unbedingt auf den Icon-Typ, und versuchen Sie nicht, zum Beispiel einem Ordner ein Programm-Icon zu verpassen.

Am einfachsten geht das Editieren von schon bestehenden Icons. Wählen Sie dazu aus dem Menü Disk den Punkt Load Data, und geben Sie am besten den kompletten Dateinamen, beginnend mit df0:, an. Nach Fertigstellung des neuen Icons speichern Sie es dann mit Save Data wieder unter dem gleichen Namen ab. Auf diese Weise haben Sie keine Probleme mit dem Icon-Typ.

Wollen Sie dagegen eigene Maschinen- oder C-Programme mit Icons versehen, so laden Sie zuerst ein Icon des richtigen Typs, löschen das Bild im Speicher und malen dann ein neues. Beim Abspeichern geben Sie dann den Namen des entsprechenden Programms an, auch wiederum vollständig mit df0: am Anfang. Was Sie bei der Erstellung von Programmen sonst noch beachten müssen, die sich durch ein Icon starten lassen sollen, kann an dieser Stelle nicht mehr beschrieben werden. Wir verweisen Sie dabei auf das demnächst erscheinende DATA-BECKER-Buch Amiga Systemprogrammierung.

Nach dem Abspeichern erscheint das veränderte Icon erst dann in seinem Fenster, wenn letzteres neu geöffnet wird. Sie müssen das entsprechende Window also zuerst einmal schließen, falls es noch geöffnet ist. Sollten Sie mehrere Icons erstellt und abge-

speichert haben, so kann es sein, daß sich die Icons überdecken. Gerade in diesem Fall sollten Sie unbedingt die Bilder richtig anordnen und Snapshot anwählen.

Vielleicht haben Sie auch schon einmal Icons gesehen, die, wenn Sie angewählt werden, nicht einfach invertiert werden, sondern die ein zweites Bild anzeigen und so zum Beispiel eine Bewegung imitieren. Um selbst solche Icons zu erzeugen, benötigen Sie das Programm IconMerge. Dieses Programm wird nur bei den Amiga 500 und 2000 auf der Diskette Amiga Extras mitgeliefert und befindet sich im Ordner Tools. Malen Sie sich also mit dem Icon-Editor zwei Icons und speichern Sie sie unter zwei verschiedenen Namen ab. IconMerge fragt Sie nach den Namen der beiden Icons und erstellt aus ihnen ein drittes Icon, dem Sie den Namen des zugehörigen Programms geben sollten. Beliebte Motive für diese Art Icons sind eine Diskette, deren Verriegelung sich öffnet, oder eine Ordner-Schublade, die herausgezogen wird.

1.4.5 Sonstige Hilfsprogramme

Im Ordner Utilities befindet sich auch der Taschenrechner (engl. Calculator). Die angezeigten Tasten lassen sich mit der Maus anklicken, einfacher ist es jedoch, direkt die Tastatur zu benutzen. Alles funktioniert genau wie bei einem richtigen Taschenrechner, daher sollten Sie problemlos damit zurechtkommen.

GraphicDump aus dem System-Ordner ist ein Hardcopy-Programm für Ihren Drucker. Nach seinem Aufruf haben Sie zehn Sekunden Zeit, alle Fenster, die Sie nicht ausgedruckt haben möchten, zu schließen. Danach erstellt Ihnen GraphicDump gemäß den Einstellungen aus Preferences einen Ausdruck des kompletten Bildschirms.

Die Routine NoFastMem ist nur für Besitzer eines Amiga mit einem Megabyte Speicher gedacht. Sie sperrt die oberen 512 KByte, um so einige Programme, die sich nicht mit der Speicher-Erweiterung vertragen, trotzdem zum Laufen zu bringen.

1.5 RAM-Erweiterung oder zweites Diskettenlaufwerk?

Wenn man eine Zeitlang mit dem Amiga gearbeitet hat, wird man feststellen, daß diverse Peripherie wie zum Beispiel ein zweites Diskettenlaufwerk oder eine Speicher-Erweiterungskarte sehr hilfreich sein können. Meistens hat man aber nicht sofort das Geld für beides und muß sich dann für eines der beiden Geräte entscheiden. Bei dieser Entscheidung will Ihnen dieses Kapitel helfen.

Einige der ersten Amiga 1000, die auf dem Markt erschienen, wurden nur mit 256 KByte RAM ausgeliefert. Sollten Sie noch ein solches Gerät besitzen, so ist Ihnen unbedingt anzuraten, sich eine Speicher-Erweiterung auf mindestens 512 KByte zu kaufen. Mindestens die Hälfte bis drei Viertel der auf dem Markt angebotenen Software benötigt nämlich diese Speicherkonfiguration.

Beim Kauf einer RAM-Erweiterung auf ein Megabyte (1024 KByte) oder mehr sollten Sie vorher bedenken, was sie Ihnen bringt. Sie können dadurch natürlich einerseits mehr Programme gleichzeitig benutzen und brauchen nicht zu befürchten, daß Ihre RAM-Disk zu schnell überfüllt ist. Diese Art von Anwendungen sind aber meistens nicht so sinnvoll, und die RAM-Disk läßt sich auch fast immer durch ein zweites Laufwerk einsparen.

Wirklich sinnvoll wird eine Speicher-Erweiterung erst dann, wenn Sie sehr große Datenmengen verwalten wollen. Das können zum Beispiel eine große Anzahl von Texten sein, digitalisierte Tondaten oder mehrere Grafikbilder in hoher Auflösung. Ein Malprogramm wie Deluxe Paint II ist beispielsweise nicht in der Lage, ein Grafikbild in der Auflösung 640 * 400 mit 16 Farben auf einem 512-KByte-Rechner zu verwalten, da ein solches Bild alleine schon 128 KByte benötigt und für die Korrekturfunktion zwei solche Bilder im Speicher neben dem Programm selbst Platz haben müssen. Auch CAD-Programme schlucken sehr viel Speicherkapazität. Bei Ihnen wirkt sich eine Speicher-Erweiterung zudem noch geschwindigkeitssteigernd aus.

Für viele andere Fälle ist aber ein zweites Diskettenlaufwerk wesentlich sinnvoller. Zum Beispiel kann man die Fähigkeiten der gängigsten Compiler der Programmiersprache C meistens erst dann voll nutzen, wenn man zwei Diskettenlaufwerke besitzt. Hierbei hat man dann den Compiler mit allen Hilfsdateien im ersten Diskettenlaufwerk, während sich die Arbeitsdiskette mit dem Programm im zweiten Laufwerk befindet. Diese Art der Arbeitsweise wird auch von vielen anderen Programmen gleichermaßen gehandhabt. Außerdem erspart man sich beim Kopieren von Dateien und Disketten das Herumhantieren mit der RAM-Disk. Man braucht dann auch zum Arbeiten mit dem CLI nicht erst alle Befehle ins RAM zu kopieren, sondern kann direkt loslegen, indem man die Workbench-Diskette in einem Laufwerk und die Arbeitsdiskette im anderen läßt.

Wie Sie sehen, lassen sich die Anwendungsgebiete der beiden Peripheriegeräte gut voneinander abgrenzen, was einem die Entscheidung erleichtern sollte. Einen anderen wichtigen Entscheidungsgrund stellt der Preis dar. Dazu ist zu sagen, daß eine Speicher-Erweiterung für den Amiga 500 momentan etwas billiger, eine Speicher-Erweiterung für den Amiga 1000 etwas teuerer als ein zweites Diskettenlaufwerk ist, immer vorausgesetzt, es handelt sich um eine Speicher-Erweiterung auf ein Megabyte. Die dabei zugrunde gelegten Preise beziehen sich nicht auf die Commodore-Preisliste, sondern auf die Preislisten von freien Händlern. Deren Preise liegen nämlich meistens deutlich unter den Preisen des Originalherstellers, ohne daß deswegen die Geräte schlechter wären. Sie sollten sich beim Kauf Ihres Gerätes ruhig vorher einmal auf dem Markt umsehen. Dadurch können Sie ohne weiteres einige 100 Mark sparen.

Es hängt also davon ab, was Sie mit Ihrem Amiga in erster Linie machen wollen, wenn es darum geht, welche Peripherie Sie sich zuerst anschaffen. Wir hoffen, Ihnen durch unseren Rat die Entscheidung erleichtert zu haben.

2. AmigaDOS

In diesem Kapitel wollen wir uns etwas genauer mit AmigaDOS auseinandersetzen. Die wichtigsten Grundbefehle des CLI kennen Sie ja bereits. Deshalb hält sich dieses Kapitel damit nicht mehr auf. Vielmehr haben wir uns auf Fragen konzentriert, die dann auftauchen, wenn man schon (oder besser: wenn man nur (?)) ein DOS wie das des C64 kennengelernt hat. Dies betrifft sowohl die Diskettenstruktur als auch die durch das DOS unterstützten Ein-/Ausgabefunktionen. Außerdem wollen Sie doch sicherlich wissen, welche der auf der Workbench-Diskette mitgelieferten Dateien wofür vom DOS benötigt werden. Alles das finden Sie in den nun folgenden Abschnitten.

2.1 Die Diskettenstruktur

Eine 3½-Zoll-Diskette beim Amiga wird auf jeder Diskettenseite in 80 Spuren (Tracks) eingeteilt. Die zwei gegenüberliegenden Spuren auf den beiden Seiten faßt man unter dem Begriff Zylinder zusammen. Wenn man von einem bestimmten Track redet, benutzt man die Zylindernummer und die Nummer des Schreiblesekopfes der entsprechenden Seite (0 oder 1). Jeder Track enthält elf Sektoren à 512 Byte Daten, egal, ob der Track außen oder innen auf der Diskette liegt. Rein rechnerisch bietet die Diskette daher 880 KByte Daten Platz.

Sie haben sicher auch schon bemerkt, daß es zwei Arten von Disketten gibt: selbstbootende und nicht-selbstbootende. Der CLI-Befehl install macht aus einer nicht-selbstbootenden Diskette eine selbstbootende. Dabei wird in Sektor Null, Zylinder Null, Kopf Null eine bestimmte Kennung geschrieben. Dieser Sektor heißt deshalb auch Boot-Sektor. In diesem Block beginnt ab dem zwölften Byte ein Maschinenprogramm, das beim Booten geladen und gestartet wird. Auch dieses wird vom Install-Befehl so eingerichtet, daß der Boot-Vorgang korrekt abläuft.

Auf Track 40 beginnt das Directory der Diskette. Es enthält jedoch nicht alle Namen der zugehörigen Dateien, sondern nur die

Nummern von Sektoren, in denen die Namen und weitere Informationen zu den Dateien stehen. Da diese Blöcke kreuz und quer über die Diskette verteilt sein können, benötigt der Amiga relativ viel Zeit, um das Inhaltsverzeichnis anzuzeigen.

Neben den üblichen 512 Byte Daten hat jeder Sektor auch noch einen Header, der dem Anwender normalerweise nicht zugänglich ist. In diesem sind Informationen verzeichnet, die das DOS benötigt, um den Sektor auf der Diskette wiederzufinden. Die Zeiger, die angeben, welche Blöcke zu einer Datei gehören, sind dagegen in den 512 Bytes untergebracht. Dies sollte Ihnen bekannt vorkommen, wenn Sie sich schon einmal näher damit beschäftigt haben, wie die Daten auf Ihren 5¼-Zoll-Disketten untergebracht wurden.

Der Amiga hat sogar eine Block-Availability-Map, kurz BAM, die anzeigt, welche Sektoren schon belegt sind und welche noch beschrieben werden können. Auch das Programm Disk-Validator von der Workbench-Diskette macht nichts anderes als der Validate-Befehl der Floppies 1541/70/71: es schreibt die BAM neu. Nur geschieht dies beim Amiga automatisch, wenn ein File nicht ordnungsgemäß geschlossen wurde, während man beim C64/C128 erst das V:-Kommando zur Floppy senden mußte.

2.2 Filetypen und Fileparameter

Bei den alten Commodore-Rechnern wurde immer zwischen verschiedenen Filetypen unterschieden. Es gab dort sequentiell aufgebaute PRG-, SEQ- oder USR-Dateien, bei denen man immer an die Reihenfolge gebunden war, in der die Daten in den Dateien untergebracht waren, und es gab die relativen REL-Dateien, bei denen man seine Daten in Datensätze gleicher Größe unterteilen mußte, wodurch man allerdings auch beliebige Daten in der Mitte eines Files sofort finden und abändern konnte.

Anders dagegen der Amiga. Hier gibt es nur einen Dateityp, der sowohl sequentiellen als auch relativen Zugriff erlaubt. Das dabei verwendete Prinzip ist eben so einfach wie effektiv. Die

Files selbst haben eine normale Blockverkettung, das bedeutet, jeder Block auf der Diskette enthält einen Zeiger auf den nachfolgenden Block der Datei. Zusätzlich existiert aber noch eine Liste darüber, welche Blöcke in welcher Reihenfolge durch die Datei belegt werden. Beim Zugriff auf das Daten-File kann der Amiga aus einer gegebenen Bytenummer sofort berechnen, welcher Block das gesuchte Datum enthält. Die Unterteilung in Datensätze bleibt dem Anwender überlassen. Dieses Konzept ist natürlich wesentlich flexibler als das des C64/C128. Daher gibt es auch kaum Programme, die ihre Daten nicht in Dateien ablegen, sondern den direkten Zugriff auf die Sektoren benutzen. Dies war ja oft ein Ärgernis beim C64, insbesondere dann, wenn ein Programm die von ihm benutzten Blöcke nicht in der BAM markiert hatte. Man durfte dann nämlich keinerlei Schreibzugriffe mehr auf der Diskette tätigen, da man sich nie sicher sein konnte, ob nicht Daten überschrieben wurden.

Im Directory einer C64-/C128-Diskette waren neben dem Filetyp manchmal noch zwei andere Markierungen zu sehen. Der Stern markierte nicht ordnungsgemäß geschlossene Files. So etwas kann aber beim Amiga nicht vorkommen, da hier, wie schon erläutert der Disk-Validator automatisch aufgerufen wird, wann immer es nötig ist.

Das <-Zeichen hinter dem Filetypen zeigte an, ob eine Datei vor Schreibzugriffen geschützt war. Der Amiga kennt eine solche Markierung nicht, sondern gleich vier. Dazu sollten Sie sich einmal ein Directory mit dem CLI-Befehl List ansehen. Hinter den meisten Dateien stehen die Buchstaben rwed. Sie sagen aus, daß das File gelesen (read), überschrieben (written), ausgeführt (executed) und gelöscht (deleted) werden kann. Wenn eine der Optionen nicht besteht, steht anstelle des entsprechenden Buchstabens ein Strich. Der CLI-Befehl protect kann diese Flags ändern. Dazu geben Sie einfach protect File-Name Flags ein, wobei Flags für die Buchstaben steht, deren Optionen Sie gesetzt haben möchten. Alle anderen werden dann gelöscht. Zum Beispiel schützt protect Preferences rwe die Datei Preferences vor unbeabsichtigtem Löschen.

2.3 Geräte unter AmigaDOS

AmigaDOS verwaltet weit mehr Geräte (engl. devices) als nur die Diskettenlaufwerke. Von einer Harddisk mal ganz abgesehen, fallen in den Aufgabenbereich des DOS genauso die Druckeransteuerung oder einfache Tastatureingabe/Bildschirmausgabe über Fenster.

Der Begriff Gerät muß nicht unbedingt für ein tatsächlich am Rechner angeschlossenes Peripheriegerät stehen. AmigaDOS simuliert auch einige Geräte, wie zum Beispiel einen genormten Drucker. Folgende Gerätebezeichnungen kennt AmigaDOS normalerweise:

DF0:, DF1:, DF2:, DF3:

Die Laufwerksbezeichnungen der angeschlossenen Laufwerke sollten Sie ja inzwischen kennen.

RAM:

Auch die RAM-Disk wurde in Kapitel 1 schon weitestgehend erklärt.

DH0:

Dies ist die Bezeichnung für die Harddisk. Außer in Geschwindigkeit und Speicherkapazität besteht von der Softwareseite her kein wesentlicher Unterschied in der Bedienung zu einer normalen Diskettenstation.

PAR:

Auf der Rückseite Ihres Amiga befindet sich eine als Parallelport bezeichnete Schnittstelle, manchmal auch Centronics-Schnittstelle genannt. Über das Gerät PAR: lassen sich Daten über diesen Port ein- und ausgeben. Wenn Sie zum Beispiel Ihren Drucker dort angeschlossen haben, können Sie mit PAR: direkt Zeichen an ihn senden, ohne daß die Zeichen in irgendeiner Weise abgeändert werden.

SER:

Neben dem parallen (Centronics-) Ausgang hat Ihr Amiga auch noch eine serielle RS232-Schnittstelle. Hier läßt sich ebenfalls ein Drucker oder auch ein Modem anschließen. Für eine serielle Schnittstelle müssen aber vorher die Übertragungsrate und noch einige andere Parameter festgelegt werden. Dies läßt sich über das Voreinstellungsprogramm Preferences bewerkstelligen. Auch für SER: gilt, daß alle Zeichen unverändert an einen Drucker geschickt werden können.

PRT:

Dieses Gerät ist für die Druckerausgabe vorgesehen. Im Gegensatz zu PAR: und SER: nimmt PRT: bestimmte Druckersteuersequenzen entgegen und wandelt sie in die entsprechenden Sequenzen des mit Preferences voreingestellten Druckers um. Die Steuercodes, die PRT: versteht, sind genormt und befinden sich im Anhang Ihres mitgelieferten Handbuchs. Jedes Programm, das mit PRT: arbeitet, läßt sich somit auf einfachste Weise an jeden Drucker anpassen, für den eine Druckertreiberroutine auf der Workbench-Diskette vorhanden ist. Das Programm, das sich hinter PRT: verbirgt, benutzt entweder SER: oder PAR: zur Ausgabe. Auch hier entscheidet die Preferences-Einstellung. Leider hat Commodore kein Programm mitgeliefert, mit dem man neue Druckertreiber erstellen kann. Mit dem Erscheinen eines solchen Programms dürfte aber in Kürze zu rechnen sein, da hier ein echter Bedarf besteht.

CON:

AmigaDOS kann zwei verschiedene Arten von Fenstern für die Tastatureingabe bzw. Bildschirmausgabe öffnen. Da wären zunächst die Fenster, die sich über das Gerät CON: ansprechen lassen. Immer, wenn eine Datei mit dem Namen CON:x1/y1/x2/y2/Fenstername öffnet wird, erscheint ein neues Fenster auf dem Bildschirm. x1/y1 und x2/y2 sind die Koordinaten der gegenüberliegenden Ecken links oben und rechts unten des Fensters. Wenn nun Zeichen an dieses Gerät gesendet werden, so erscheinen sie innerhalb des Fensters. Eingaben, die Sie innerhalb des Fensters tätigen, werden erst dann an das Pro-

gramm gesendet, das die Eingaben erwartet, sobald Sie die Return-Taste drücken. Sie können dadurch eine Zeile eingeben und eventuell erst noch mit der Backspace-Taste korrigieren, ehe die Zeichen gesendet werden. Probieren Sie doch einmal im CLI folgendes Beispiel aus:

```
type con:10/10/600/100/Fenster
```

Der Type-Befehl holt sich jetzt seine Zeichen nicht aus irgendeinem Daten-File, sondern aus dem neuen Fenster, und bringt die Zeichen ins CLI-Fenster. Oder wollen Sie einen kurzen Text schnell auf Ihrem Drucker zu Papier bringen? Probieren Sie es doch einmal mit

```
copy con:10/10/600/200/Hallo prt:
```

Abbrechen können Sie die Eingabe jeweils mit Ctrl \. CON:-Windows haben aber einen Nachteil, den Sie sicherlich schon beim CLI, das ja selbst ein CON:-Window benutzt, festgestellt haben: Alles läuft zeilenorientiert ab, man kann die Cursortasten nicht verwenden. Auch die Funktionstasten sind völlig wirkungslos. Um diesen Mangel zu beheben, gibt es eine zweite Art von Fenstern, die sogenannten RAW:-Windows. Bevor wir aber auf sie zu sprechen kommen, vorher noch etwas zu einem ganz bestimmten CON:-Fenster, nämlich dem CLI-Fenster selbst. Dieses spricht man über das Stern-Zeichen an, und zwar ohne irgendwelche Zusätze. Die Druckerausgabe des letzten Beispiels läßt sich daher auch ohne ein zusätzliches Fenster realisieren:

```
copy * prt:
```

Hierbei wird auf das neue Fenster verzichtet, und die Eingaben werden aus dem normalen CLI-Fenster übernommen.

RAW:

RAW-Windows werden genauso eröffnet wie CON:-Windows. Bei der Ausgabe von Zeichen in dem Fenster verhält sich ein RAW:-Window auch nicht anders. Jedoch arbeitet die Tastatureingabe deutlich anders. Die eingegebenen Zeichen erscheinen nicht auf dem Bildschirm, und jede gedrückte Taste wird sofort

an das die Zeichen empfangende Programm weitergeleitet. Auch für die Cursortasten und die Funktionstasten werden Codes gesendet. Das Ganze hat den Zweck, daß das Programm, das mit dem RAW-Window arbeitet, die Bildschirmausgabe selbst gestalten kann, wodurch zum Beispiel Scrolling durch einen Text möglich wird. RAW:-Windows sind natürlich wesentlich flexibler als CON:-Windows, haben jedoch den Nachteil, daß das sie benutzende Programm sich zwangsläufig selbst um mehr kümmern muß. Ein Beispiel für ein RAW:-Window, ist der CLI-Editor ed. Wenn Sie sich den Unterschied zu CON-Windows noch einmal an einem Beispiel klarmachen wollen, probieren Sie es doch einmal mit:

```
copy RAW:10/10/600/100/Raw-Fenster *
```

Der Abbruch geschieht hier nicht durch Ctrl \, sondern durch Ctrl C. Machen Sie sich am besten einmal selbst klar, was hier genau passiert, insbesondere dann, wenn Sie zwischen RAW:- und CLI-Fenster mit der Maus hin- und herschalten.

NIL:

Wenn dieses Gerät zur Ausgabe verwendet wird, gehen alle ankommenden Zeichen ins Nichts, sie verschwinden einfach. Dies kann ganz nützlich sein, wenn man eine unerwünschte Bildschirmausgabe unterdrücken will.

Welche Geräte gerade angeschlossen oder aktiviert sind, können Sie durch den CLI-Befehl assign erfahren. Geben Sie assign ohne irgendwelche Parameter ein. Unter dem Punkt devices stehen die gesuchten Bezeichnungen.

Um die aufgeführten Geräte richtig nutzen zu können, muß man eine Besonderheit des CLI kennen, auf die hier noch nicht eingegangen wurde. Man hat nämlich die Möglichkeit, Eingaben und Ausgaben, die normalerweise über die Tastatur bzw. den Bildschirm ablaufen, auf ein anderes Gerät umzulegen. Dazu gibt man als erstes Argument hinter einem Befehl einen Gerätenamen mit vorangestelltem < oder > an. < wird zur Änderung

des Eingabegeräts benutzt, > zur Änderung des Ausgabegeräts. Das Standardbeispiel dafür ist das Umleiten einer Bildschirmausgabe auf den Drucker, beispielsweise beim Type-Befehl:

```
type >prt: Dateiname
```

Beachten Sie, daß nicht der Type-Befehl das Argument >prt: auswertet, sondern daß dies schon vom CLI durchgeführt wird. Dadurch kann das Umleiten für jedes Programm benutzt werden, auch für selbstgeschriebene.

Wofür sind CON:- und RAW:-Fenster nützlich? Vom CLI aus benötigt man sie nur selten. In BASIC kann man auch fast immer auf sie verzichten. In C werden Bildschirmausgaben im aktuellen CLI-Fenster dargestellt. Wenn man aber C-Programme schreiben will, die sich von der Workbench aus starten lassen oder wenn einem ein CON:-Fenster zur Ausgabe nicht flexibel genug ist, so sollte man mit eigenen Fenstern arbeiten. Auch von Maschinensprache aus ist es oft einfacher, CON: und RAW: als Ausgabegeräte zu verwenden. Man benötigt dann nur die Betriebssytemroutinen Open, Close, Read und Write. Diese finden Sie auch in diesem Buch kurz erklärt, nämlich in Kapitel 3.5.2.

Hier aber noch eine Anwendung für das CLI. Wenn man das Multitasking des Amiga mit dem Befehl run benutzen will, kann man das Problem umgehen, daß sich Programme gegenseitig mit der Ein- und Ausgabe stören. Benutzen Sie den Dir-Befehl in Verbindung mit run doch einmal so:

```
run dir >con:10/10/600/100/Directory
```

2.4 Befehlsdateien

Beim C64/C128 konnte man Programme zur Ansteuerung von DOS-Funktionen nur indirekt über BASIC oder Maschinensprache schreiben. Anders dagegen der Amiga. Das CLI bietet die Möglichkeit, Programme, die nur aus DOS-Befehlen be-

stehen, zu erstellen. Dazu existieren sogar einige Befehle, die den Ablauf solcher Befehlsdateien steuern können. Was sich damit alles anstellen läßt, zeigen wir Ihnen in diesem Kapitel.

Eine Befehlsdatei haben Sie bereits kennengelernt: die Startup-Sequence. Sie besteht einfach aus einer Anzahl von DOS-Befehlen, die der Reihe nach abgearbeitet werden. Jeder Befehl steht in einer eigenen Zeile, und zwar genau so, wie Sie ihn auch über die Tastatur eingeben würden. Das Erstellen einer solchen Datei geschieht mit dem Editor ed.

Sie können nun hingehen und Befehlsdateien für den eigenen Bedarf erstellen. Der Sinn und Zweck dieser Dateien ist es, sich Tipparbeit zu ersparen. Ausgeführt wird eine solche Datei mit dem Befehl execute Dateiname. Falls Sie oft mit execute arbeiten, empfiehlt es sich, diesen Befehl in e oder ex umzubenennen.

Ein Beispiel: Sie wollen eine Befehlsdatei einrichten, die Ihnen einige wichtige Befehle in die RAM-Disk kopiert. Die Datei könnte dann so aussehen:

```
copy c/copy ram:
ram:copy c/dir ram:
ram:copy c/cd ram:
ram:copy c/delete ram:
ram:copy c/list ram:
ram:copy c/info ram:
path ram:
```

Angenommen, Sie haben mit ed die Datei erstellt und unter dem Namen ramdisk abgelegt, dann läßt sie sich nun durch execute ramdisk starten.

Befehlsdateien werden natürlich wesentlich effektiver, wenn man eine Möglichkeit hat, Dateinamen mit an die aufgerufene Datei zu übergeben. Zum Beispiel wäre eine Routine praktisch, die ein File zum Drucker schickt, ohne dabei das aktuelle CLI-Fenster zu blockieren. Die Befehlsdatei ist folgendermaßen aufgebaut:

```
.key Datei
run type >prt: <Datei>
```

Diese unter dem Namen print abgespeicherte Datei wird durch execute print Dateiname aufgerufen. Die Anfangszeile .key Datei definiert ein Label Datei. Wenn nun in der Befehlsdatei irgendwo der Text <Datei> vorkommt, wird dieser durch den Parameter ersetzt, der beim Aufruf hinter dem Befehlsdateinamen nach execute steht.

In Verbindung mit Compilern und Assemblern fertigt man sich oft Befehlsdateien an, die den Übersetzer mit vorgegebenen, oft verwendeten Optionen starten. Die Befehlsdateien nennt man dann auch Batchdateien. Dafür kann es von Interesse sein, ob der Übersetzer den Übersetzungsvorgang korrekt abschließt oder ob ein Fehler gemeldet wird. Ein Fehlertest kann natürlich auch bei anderen Arten von Befehlsdateien sinnvoll sein. Manchmal sollte man einen Fehler auch besser dadurch abfangen, indem man testet, ob eine Datei existiert. Hier ein Beispiel:

```
.key Datei
if EXISTS <Datei>
copy <Datei> ram:
else
echo "Die Datei existiert nicht"
endif
```

Die hier verwendete Anweisung if testet verschiedene Bedingungen, je nachdem, welches Schlüsselwort mit ihr angegeben wird. Exists mit nachfolgendem Dateinamen kontrolliert, ob die angegebene Datei vorhanden ist. Wenn dies der Fall ist, werden alle Befehle zwischen if und else ausgeführt. Sind die Daten nicht vorhanden, dann werden die Befehle zwischen else und endif ausgeführt. Der Else-Teil ist optional. Es können auch mehrere If-Anweisungen geschachtelt werden. echo gibt, wie am Beispiel zu sehen, einen Text auf dem Bildschirm aus.

Jeder Befehl kann eine Fehlernummer zurückliefern, wenn bei seiner Ausführung etwas schiefgelaufen ist. Die Fehlernummer des jeweils letzten Befehls merkt sich das CLI. Mit if Warm, if Error und if Fail läßt sich kontrollieren, ob ein Fehler aufge-

treten ist. Die Befehle nach if Warn werden dann ausgeführt, wenn die letzte Fehlernummer größer oder gleich fünf war, nach if Error ab der Fehlernummer zehn und nach if Fail ab der Fehlernummer 20. Dazu muß aber gesagt werden, daß die Ausführung einer Befehlsdatei normalerweise durch einen Fehlercode größer oder gleich 10 abgebrochen wird. Die Abbruchs-Grenze läßt sich mit dem Befehl failat auf einen beliebigen Wert erhöhen, was für if Error und if Fail unbedingt erforderlich ist. Ein Beispiel:

```
failat 100
...
if error
skip fehler
endif
...
lab fehler
echo "Fehler aufgetreten"
```

Ein Fehler durch den letzten Befehl vor der If-Anweisung mit einer Fehlernummer kleiner 100 führt nicht zu einem Abbruch des DOS-Programms, sondern dazu, daß der Befehl skip fehler ausgeführt wird. Dieser entspricht etwa dem Goto-Befehl von BASIC. Er springt nämlich zu einer durch lab festgelegten Marke. Skip kann aber nur vorwärts springen, wodurch ausgeschlossen wird, daß Befehlsdateien Schleifen enthalten.

Der Befehl quit beendet eine Befehlsdatei, wahlweise auch mit Rückgabe einer Fehlernummer. Also quit 10 für einen Abbruch mit Fehlercode 10.

Noch ein interessanter Befehl für Befehlsdateien: Ask gibt einen Text aus und erwartet dann als Eingabe yes oder no bzw. y oder n. Wenn yes eingegeben wird, wird der Fehlercode 5 gesetzt. Auf diese Weise wird die Befehlsdatei nicht unterbrochen, aber man kann mit if Warn abfragen, welche Wahl der Benutzer getroffen hat. Auch hierfür ein Beispiel:

```
ask "yes/no?"
if warn
echo "yes"
else
echo "no"
endif
```

2.5 Systemdateien

Schon in Kapitel 1 haben wir angedeutet, daß Sie längst nicht alle Dateien Ihrer Workbench-Diskette wirklich benötigen. Auf allen bootfähigen Disketten muß aber ein gewisser Satz an Dateien vorhanden sein. Daher wollen wir uns hier einmal die Workbench-Diskette vornehmen und erklären, wofür man welche Files benötigt. Sie können sich dann Platz schaffen, indem Sie auf (einer Kopie!) der Diskette alle Files löschen, die Sie für unnötig halten.

Da der Amiga mit verschiedenen Versionen der Workbench ausgeliefert wird, unterscheiden sich die Workbench-Disketten leicht voneinander. Im großen und ganzen sind sie aber gleich aufgebaut. Bei den weiteren Erklärungen beziehen wir uns auf die Workbench-Version V33.53.

Um wirklich alle Dateien zu betrachten, können Sie natürlich nicht von der Workbench-Ebene an die Diskette herangehen, sondern müssen das CLI benutzen. Beginnen wir mit den Dateien, die in keinem Ordner stehen. Da wäre zunächst die Datei Clock. Sie enthält das Uhrenprogramm, das aber ohne die Erweiterungskarte mit der batteriegepufferten Echtzeituhr nicht mehr als eine Spielerei ist und daher bedenkenlos gelöscht werden kann. Das nächste Programm ist Preferences. Wenn Sie sich sicher sind, daß Sie die abgespeicherten Voreinstellungen nicht mehr ändern wollen, können Sie es löschen.

Alle weiteren nicht in Ordnern befindlichen Dateien sind .info-Dateien, das heißt, sie enthalten Icons. Wenn Sie eine Datei wie Clock gelöscht haben, können Sie natürlich auch die zugehörige Datei Clock.info entfernen. Sofern die entsprechende Diskette nicht in Verbindung mit der Workbench verwendet werden soll,

können Sie selbstverständlich auch alle anderen .info-Dateien löschen. Das gleiche gilt für den Ordner Trashcan. Auch er ist nur für das Arbeiten mit der Workbench von Interesse, da der Delete-Befehl des CLI nicht mit ihm zusammenarbeitet.

Gehen wir als nächstes zum Ordner C über. Wie Sie sicher schon wissen, enthält er die CLI-Befehle. Hier müssen Sie selbst entscheiden, welche der Befehle Sie benötigen und welche nicht. Seien Sie aber vorsichtig, da in der Startup-Sequence der Workbench auch einige ungebräuchliche Befehle, wie zum Beispiel BindDrivers aufgerufen werden. Auf einige Befehle können Sie aber mit Sicherheit verzichten. Unser Vorschlag:

Ask
Break
ChangeTaskPri
DiskChange
Edit
Fault
Join
Lab
Mount
Quit
SetDate
Skip
Status
Wait

Ohne die Uhrenkarte brauchen Sie auch den Befehl SetClock nicht. Kontrollieren Sie aber besser die Startup-Sequence, und ändern Sie diese, falls der Befehl darin vorkommt. Wenn allerdings von vornherein feststeht, welche CLI-Befehle benötigt werden, können Sie auch alle anderen löschen.

Der Ordner Demos enthält nichts anderes als Demonstrationsprogramme, die spätestens nach dem zweiten Ansehen langweilig werden und daher ersatzlos gestrichen werden können.

Im Ordner System befinden sich einige weniger oft benötigte Hilfsprogramme. Ob Sie das Hardcopy-Programm GraphicDump,

den Icon-Editor IconEd, das Kopierprogramm Diskcopy oder den Sprachsynthesizer Say dort belassen, bleibt Ihnen selbst überlassen. CLI wird nur gebraucht, falls Sie von der Workbench-Ebene wieder auf die CLI-Ebene zurückkehren wollen. NoFastMem ist bei einem 512-KByte-Gerät völlig nutzlos, da es nichts anderes macht, als die Speicher-Erweiterung softwaremäßig zu deaktivieren.

Die drei Dateien Disk-Validator, Port-Handler und RAM-Handler des Ordners l sollten nicht verändert werden. Disk-Validator erstellt automatisch, wenn notwendig, eine neue Blockbelegungstabelle der Diskette. RAM-Handler verwaltet die RAM-Disk, während Port-Handler für die Geräte SER: und PAR: zuständig ist.

Der nächste Ordner ist devs. Hier sind in den Unterordnern keymaps und printers die Tastatur- und Druckertreiber untergebracht. Sie benötigen natürlich nur die Treiber für die von Ihnen verwendete Tastaturbelegung und den Drucker, den Sie mit Preferences eingestellt haben. System-Configuration wurde schon öfter angesprochen. Diese Datei wird von Preferences erzeugt und enthält alle möglichen Voreinstellungen. Die Dateien printer.device, parallel.device und serial.device sollten dann auf der Diskette bleiben, wenn Sie einen Drucker, ein Modem oder ein Midi-Interface an Ihrem Amiga betreiben. Narrator.device wird für die Sprachausgabe gebraucht. MountList ist ein reines Text-File, das Sie sich mit type anzeigen lassen können. Es enthält Informationen zum Betrieb eines 5¼-Zoll-Diskettenlaufwerks am Amiga, wird aber von keinem Programm in irgendeiner Form benötigt.

Der Ordner s enthält die Ihnen schon bekannte Startup-Sequence. Falls Sie eine Harddisk besitzen, sollten Sie die alte Startup-Sequence löschen und das File Startup-Sequence.hd in Startup-Sequence umbenennen.

Im Unter-Directory t speichert der Bildschirmeditor ed eine Sicherheitskopie des Files ab, das er beim Abspeichern eventuell überschreibt. Solange man mit ed arbeitet, sollte man t auf der Diskette belassen.

Die Schriftsätze aus dem Ordner fonts sind nur dann von Interesse, wenn man nicht mit dem vom Betriebssystem vorgegebenen Zeichensatz arbeitet. Das Notepad macht zum Beispiel davon Gebrauch.

Libraries sind Sammlungen von Betriebssystemroutinen. Nicht alle Libraries stehen im Kickstart-ROM. Einige seltener gebrauchte Routinen sind im Ordner libs. Diskfont.library, icon.library, info.library und version.library werden aber vom Betriebssystem bzw. der Workbench benötigt. Mathtrans.library und mathieeedoubbas.library sind spezielle Fließkommarechenroutinen, die nur von wenigen Programmen aufgerufen werden. Translator.library wird nur vom Sprachsynthesizer gebraucht und ist daher relativ unwichtig.

Der leere Ordner Empty hat die Funktion, beim Arbeiten mit der Workbench auf einfache Weise neue Ordner zu erzeugen. Ohne Workbench kann der Ordner natürlich gelöscht werden.

Die Programme Notepad und Calculator aus dem Ordner Utilities kennen Sie ja schon. Zum Schluß bleibt nur noch der Ordner Expansion über, der für Dateien zum Betrieb einer Harddisk vorgesehen ist. Ist eine Harddisk nicht vorhanden, kann er natürlich gelöscht werden.

3. Vom Microsoft BASIC V2 zum AmigaBASIC

In diesem Kapitel vergleichen wir die BASIC-Interpreter von C64 und Amiga. Zudem erhalten Sie eine schnelle Einführung in die Besonderheiten des AmigaBASIC. Wir gehen natürlich davon aus, daß Sie BASIC in den wichtigsten Grundzügen kennengelernt haben. Falls Sie BASIC überhaupt noch nicht kennen oder Ihnen der Ihnen hier gebotene Einblick nicht weit genug geht, dann werfen Sie doch einmal einen Blick in eines der DATA-BECKER-Bücher AmigaBASIC oder Amiga Tips und Tricks. Hier finden Sie mit Sicherheit alles, von den ersten BASIC-Befehlen bis zu komplexen Hilfsprogrammen.

3.1 Unterschiede in der Konzeption der Interpreter

Der erste Unterschied der BASIC-Interpreter von C64 und Amiga, der jedem sofort auffällt, ist die Tatsache, daß AmigaBASIC von einer Diskette nachgeladen wird, während beim C64 das Microsoft BASIC V2 sofort nach dem Einschalten zur Verfügung steht. Was zuerst als Nachteil des Amiga erscheint, ist jedoch auf den zweiten Blick ein ungeheurer Vorteil. Zwar benötigt man auf dem Amiga etwas mehr Zeit, um loslegen zu können, jedoch ist ein Interpreter im RAM wesentlich flexibler als einer im ROM. Man kann nämlich einerseits damit rechnen, daß diverse Fehler und Schwächen, die der AmigaBASIC-Interpreter aufweist, noch nachträglich geändert werden können. Andererseits belegt der Interpreter keinen Speicherplatz, sobald man ihn nicht mehr braucht.

Beim C64 waren einige DOS-Funktionen speziell für den BASIC-Interpreter konzipiert. Man denke dabei nur an das Laden des Directories einer Diskette in den Arbeitsspeicher. Durch die klare Trennung von DOS und BASIC auf dem Amiga wird kein Programmierer dahingehend eingeengt, sich mit diesem Interpreter beschäftigen zu müssen, wenn er seine Programme lieber in einer anderen Programmiersprache schreiben möchte. Alle wichtigen Systemroutinen sind wirklich da, wo sie hinge-

hören, nämlich im Betriebssystem. Beim C64 hingegen war jeder Maschinensprache-Programmierer, der beispielsweise mit Fließkommaroutinen arbeiten wollte, darauf angewiesen, sich diese Routinen aus dem BASIC-Interpreter heraussuchen zu müssen.

Eine Sache, die jedem C64-Besitzer sofort nach dem Laden des Interpreters ins Auge sticht, ist der vermeintlich sehr geringe Programmspeicher von nur 25000 Bytes. Schließlich hat selbst der C64 trotz seines im Vergleich zum Amiga relativ kleinen Hauptspeichers von 64 Kilobyte mehr Speicherplatz für BASIC-Programme frei. Des Rätsels Lösung liegt auch hier im flexiblen Konzept des Amiga. Der Speicherplatz, der dem BASIC zur Verfügung steht, ist nicht wie beim C64 auf einen festen Wert beschränkt, sondern kann beliebig bis zur vollen Nutzung des Systemspeichers vergrößert werden. Der dazu notwendige Befehl CLEAR wird ausführlich auf den Seiten 6-17 und 9-40 des mitgelieferten BASIC-Handbuchs beschrieben. Dieses System hat den Vorteil, daß BASIC-Programme jederzeit von Speicher-Erweiterungen Gebrauch machen können und daß andererseits kurze BASIC-Programme nicht zuviel Speicher verschwenden. Schließlich ist der Amiga ein Multitasking-Computer, bei dem sich gleichzeitig ablaufende Programme den vorhandenen Speicherplatz teilen müssen. Ein Nachteil des CLEAR-Befehls soll aber nicht verschwiegen werden: Will man Programme laden oder erstellen, die mehr als 25000 Bytes belegen, muß man unbedingt darauf achten, daß man genügend Speicher reserviert hat. AmigaBASIC ist in dieser Beziehung nämlich nicht so gegen Abstürze gesichert, wie man es erwarten könnte. Seien Sie also vorsichtig, sonst kann es Ihnen passieren, daß Sie Ihr Programm nicht mehr abspeichern können, weil der Interpreter ununterbrochen die Fehlermeldung "Out of memory!" von sich gibt.

Noch ein Wort zur Geschwindigkeit des Interpreters. Zweifellos ist der AmigaBASIC-Interpreter schneller als der Interpreter im C64 (wenn man von der Bildschirmausgabe einmal absieht). Er ist aber bei weitem nicht so schnell, wie er eigentlich sein müßte, bedenkt man, daß man es mit einem (angeblich) vollständig in 68000-Assembler geschriebenen Programm zu tun hat. Insbesondere der Editor zeichnet sich durch quälende Langsamkeit aus. Woran das liegt, weiß Commodore. Wir dagegen können

nur vermuten, daß es sich bei dem Interpreter doch um ein C-Programm handelt. Man sollte daher bei BASIC-Programmen auf nebenherlaufende andere Programme soweit es geht verzichten. Diese bremsen nämlich sowohl die Ablaufgeschwindigkeit als auch die Window- und Menüaufbauten dermaßen stark ab, daß das Arbeiten mit BASIC völlig unerträglich wird. Es bleibt nur zu hoffen, daß Commodore oder andere Firmen diesen Mangel rechtzeitig erkennen und für Abhilfe sorgen.

3.2 Unterschiede der BASIC-Editoren

Um ein Programm zum Laufen zu bringen, muß man es selbstverständlich erst einmal eingeben. Wie gut sich das bewerkstelligen läßt und wie gut sich ein Programm auf Fehler hin testen läßt, hängt vom Editor ab.

Beim C64 hat jede Zeile eine Zeilennummer, die sowohl die Position der Zeile im Text angibt, als auch dem GOTO-Befehl eine Markierung liefert, auf die er sich beziehen kann. Trotzdem hat der Editor des C64-BASIC Elemente der Bildschirmorientiertheit. Man kann schließlich in einem Programmteil, den man mit dem LIST-Befehl auf den Bildschirm geholt hat, frei mit dem Cursor herumfahren und die Zeilen in beliebiger Weise ändern. Um die Eingabe zu beschleunigen, steht einem die Möglichkeit frei, die Befehle als Abkürzungen einzugeben.

Ein beliebtes Mittel zur übersichtlichen Gestaltung von Programmen ist das Einrücken von Programmzeilen, die innerhalb von Schleifen liegen. Auf dem C64 läßt sich so etwas nur durch einen umständlichen Trick erreichen, nämlich durch Setzen eines Doppelpunktes nach der Zeilennummer und dem Einfügen einer entsprechenden Anzahl von Leerzeichen in jeder einzurückenden Zeile.

Die einzige Unterstützung, die das Microsoft-BASIC V2 zum Austesten von Programmen bietet, ist das Setzen von Unterbrechungspunkten (Breakpoints) mit Hilfe der STOP-Anweisung. Hierdurch hat man die Möglichkeit, Variableninhalte zu bestimmten Zeitpunkten abzufragen.

Beim Amiga dagegen sieht die Sache ganz anders aus. Hier wird zuerst einmal zwischen Direktmodus-Eingaben und Änderungen am Programmtext streng unterschieden. Bei den Direktmodus-Eingaben hat man zur Korrektur leider nur die Backspace-Taste zur Verfügung, die der DEL-Taste beim C64 entspricht. Man kann mit dem Cursor nicht mehr Änderungen mitten in der Zeile vornehmen, und man kann auch keine schon einmal getippten Befehle noch einmal verwenden. Wenn man sich einmal völlig vertippt hat, kann man mit Ctrl-C die Eingabe einer Zeile ohne ihre Ausführung abbrechen. Dies hat die gleiche Wirkung wie die Eingabe von Shift-Return beim C64.

Die Nachteile der Direktmodus-Eingabe fallen kaum ins Gewicht, wenn man die Vorteile des eigentlichen Programmeditors dagegen stellt. Zeilennummern gibt es zwar noch, jedoch wird kein vernünftiger Mensch von ihnen Gebrauch machen. Commodore hat die Zeilennummern auch wohl nur deshalb integriert, um das Umschreiben von bestehenden BASIC-Programmen anderer Computer auf den Amiga zu erleichtern. Der Editor benötigt diese Zeilennummern nicht. Man hat nämlich wie bei einem Textprogramm immer den gesamten Programmtext vor sich, in dem man beliebig mit dem Cursor herumfahren kann, und der auch automatisch mitscrollt, sobald der Cursor den Rand erreicht hat. Es gibt auch keine INSERT-Taste mehr, sondern der Editor fügt grundsätzlich alle eingegebenen Zeichen in den Text ein, ohne bestehende Zeichen zu überschreiben. Dieses Konzept wäre an sich sehr gut, wenn es da nicht den schon erwähnten Haken gäbe: das schleichende Tempo. Zum Glück gibt es einige Tricks, mit denen man diese Schwäche einigermaßen ausgleichen kann.

Um auf dem Bildschirm mit dem Cursor schnell eine bestimmte Position zu erreichen, braucht man nicht zu warten, bis die Cursorsteuertasten ihren Dienst getan haben. Meistens geht es nämlich schneller, indem man die entsprechende Position direkt mit der Maus anklickt. Zusammen mit der Shift-Taste kann man mit Hilfe der Cursortasten den Programmtext schnell durchblättern. Mit der Alt-Taste zusammen dienen die Cursortasten zum Anspringen von Zeilenanfang und Zeilenende beziehungsweise Textanfang und Textende. Die TAB-Taste kann jedoch

nicht zum schnellen Springen innerhalb einer Zeile verwendet werden, wie es im ED-Editor der Fall ist. Mit ihr werden grundsätzlich zwei Leerzeichen eingefügt. Ebenso kann die Return-Taste nicht mitten im Text zum Anspringen des Anfangs der nächsten Zeile benutzt werden, da Return ein Zeichen ist, das in den Text eingefügt wird. Dieses Zeichen läßt sich auch mit der Backspace-Taste löschen, wie Sie es vielleicht auch schon vom ED-Editor her kennen.

Was macht man aber, wenn man innerhalb eines langen Programms eine mitten im Text stehende Routine editieren will, ohne sich erst durch zwanzig Seiten hindurchzublättern? Da der Amiga als Ersatz für Zeilennummern Text-Labels zur Verfügung stellt, kann man hier prinzipiell genauso vorgehen wie beim C64. Man benutzt einfach den LIST-Befehl, gibt aber statt einer Zeilennummer das entsprechende Label an. Das setzt natürlich voraus, daß der gesuchte Programmteil auch Label enthält. Wer jedoch seine Programme einigermaßen sinnvoll in Unterprogramme unterteilt und diese Unterprogramme zudem mit aussagekräftigen Labels versieht, sollte keine Schwierigkeiten haben, jederzeit die gewünschte Stelle wiederzufinden. Was der Hersteller allerdings nicht berücksichtigt hat, ist das Auffinden von Unterprogrammen, die mit der SUB-Anweisung definiert werden. Hier muß man halt ohne den LIST-Befehl zurechtkommen oder eben ein Label suchen, von dem man weiß, daß es sich in der Nähe des UnterProgramms befindet.

Der AmigaBASIC-Editor unterstützt auch die übersichtliche Programmgestaltung. Da wäre zuerst einmal das Einrücken mit der TAB-Taste. Programmteile innerhalb von Schleifen oder von IF-THEN-ELSE-END-IF-Klauseln werden wesentlich übersichtlicher, wenn man sie einrückt. Hat man einmal eine Zeile eingerückt, werden alle folgenden Zeile ebenfalls eingerückt, bis man das Einrücken mit Backspace rückgängig macht. Sie sind es zwar vieleicht nicht gewöhnt, aber wir möchten Sie in Ihrem eigenen Interesse bitten: Nutzen Sie die Ihnen hiermit zur Verfügung gestellte Hilfe aus. Es ist beim Amiga wirklich kein Aufwand, alles korrekt einzurücken, und es erhöht die Lesbarkeit Ihrer Programme enorm. Eine weitere Hilfe bei der Programmgestaltung ist die automatische Umsetzung von BASIC-

Schlüsselwörtern in Großschrift. Wenn Sie alles in Kleinschrift eingeben, können Sie sofort sehen, ob Sie sich bei einem BASIC-Befehl verschrieben haben, weil dann das entsprechende Wort nicht in Großbuchstaben umgesetzt wird.

BASIC-Befehle lassen sich leider nicht abkürzen. Dies trägt allerdings zur Fehlervermeidung bei, da man bei einem solch großen Befehlssatz, wie AmigaBASIC ihn bietet, sonst leicht durch die Abkürzung einen anderen Befehl angibt, als man eigentlich will. Der einzige abkürzbare Befehl ist PRINT, für den man genau wie beim C64 ein Fragezeichen verwenden kann. Man kann auch nicht mehr beliebig BASIC-Befehle, Variablen und Zahlen hintereinander schreiben, ohne dazwischen Leerzeichen zu setzen. Der Grund ist einfach der, daß AmigaBASIC beliebige Variablennamen erlaubt, also auch solche, die BASIC-Befehle enthalten. Beispielsweise darf in dem Variablennamen Zeilen die Silbe len nicht mit der BASIC-Funktion LEN verwechselt werden. Damit der Interpreter aber die Variablennamen von den Befehlen unterscheiden kann, müssen die Befehle eindeutig erkennbar sein, was durch das Setzen der Leerzeichen bewerkstelligt wird. AmigaBASIC ist also nichts für Tippfaule.

Wichtige Programmierhilfen, die man beim C64 nur von BASIC-Erweiterungen her kennt, sind der Trace-Modus und die Einzelschrittabarbeitung. Beim Trace-Modus wird während des Programmlaufes der gerade abgearbeitete Befehl, im LIST-WINDOW umrandet, angezeigt. Sie haben natürlich nur etwas davon, wenn das LIST-WINDOW auch sichtbar ist. Gerade beim Arbeiten mit eigenen Bildschirmen (Screens) hat diese Option nur dann Sinn, wenn genügend vom Workbench-Screen sichtbar ist, so daß das LIST-WINDOW angezeigt werden kann. Da der Interpreter beim Programmstart das LIST-WINDOW in den Hintergrund schaltet, sollten Sie es bei der Verwendung des Trace-Modus über die Vordergrund/Hintergrund-Symbole wieder nach vorne bringen. Ein- und Ausschalten läßt sich der Trace-Modus einerseits über das RUN-Menü, andererseits über die Befehle TRON und TROFF.

Interessanter beim Austesten von Programmen ist die Einzelschrittabarbeitung. Durch Betätigen der (rechten) Amiga-Taste zusammen mit T wird jeweils die nächste Anweisung im Programm ausgeführt. Bezüglich des LIST-WINDOWS gilt das gleiche, was schon zum Trace-Modus gesagt wurde. Sie können beispielsweise aus einem laufenden Programm mit Ctrl-C, Amiga-Punkt, der STOP-Option aus der Menüleiste oder der STOP-Anweisung im Programm selbst herausspringen, um dann an dieser Stelle das Programm in einzelnen Schritten weiter fortzusetzen. Durch Eingabe von END im Direktmodus können Sie die Einzelschrittabarbeitung auch am Programmanfang beginnen. Sie haben dann die Möglichkeit, vor und nach jeder Anweisung alle Variablen abzufragen. AmigaBASIC stellt Ihnen damit ein äußerst mächtiges Hilfsmittel zur Verfügung, Ihre Programme auf Fehler hin zu untersuchen. Auch hier können wir Ihnen nur empfehlen: nutzen Sie es! Sie kommen damit sehr viel schneller zu einem brauchbaren Ergebnis.

3.3 Änderungen im Programmtextaufbau

Formeln und Anweisungen werden zwar im Prinzip beim AmigaBASIC auf die gleiche Weise verwendet wie beim C64, es gibt jedoch einige Unterschiede, die man kennen sollte.

Einige wichtige Unterschiede wurden schon im vorhergehenden Kapitel beschrieben. Sie sollen hier nur kurz angesprochen werden. Da wäre zuerst einmal die Verwendung von Sprungmarken (Labels) anstelle von Zeilennummern. Sprungmarken müssen am Anfang einer Zeile stehen. Sie dürfen maximal 40 Zeichen beinhalten und müssen mit einem Doppelpunkt enden. Befehle müssen aus den schon erwähnten Gründen unbedingt durch Leerzeichen von Variablennamen und Zahlen abgetrennt sein.

Genau wie Sprungmarken dürfen Variablen maximal 40 Zeichen lang sein, das heißt, der Interpreter ignoriert alles, was nach dem vierzigsten Zeichen kommt. Sie dürfen im Gegensatz zum 64er BASIC-Schlüsselworte enthalten, sofern man nicht versucht,

BASIC-Schlüsselworte selbst als Variablen zu verwenden. Vorsicht ist übrigens bei der Verwendung von deutschen Umlauten innerhalb von Variablennamen geboten. Sie sind schlicht und einfach nicht erlaubt.

Beim C64 gibt es eigentlich nur drei Datentypen, wenn man die durch den DIM-Befehl definierten Felder nicht gesondert als Datentypen auffaßt. Es handelt sich dabei um Zeichenketten (Strings), Ganzzahl- (Integer-) und Fließkommavariablen. Strings werden dabei durch das Dollar-Zeichen gekennzeichnet, Ganzzahlvariablen durch das Prozentzeichen. Beim Amiga gibt es diese Datentypen ebenfalls, zusätzlich existieren noch zwei weitere. Das AmigaBASIC unterscheidet nämlich kurze und lange Ganzzahlvariablen sowie Fließkommavariablen von einfacher und doppelter Genauigkeit. Kurze Ganzzahlvariablen entsprechen dem, was man vom C64 her bereits kennt. Sie bestehen intern aus zwei Byte, wodurch sich ein gültiger Wertebereich von -32768 bis +32767 ergibt. Schon beim C64 hatte man das Problem, daß sich diese Variablen nur bedingt als Zeiger auf Speicheradressen eignen, da Adressen ab 32768 nicht mehr dargestellt werden können. Man mußte daher auf Fließkommavariablen ausweichen. Beim Amiga dagegen gibt es lange Ganzzahlvariablen. Sie werden mit einem Und-Zeichen (&) am Ende gekennzeichnet. Da Sie eine interne Länge von vier Byte besitzen, können sie Werte von -2147483648 bis +2147483647 aufnehmen. Daher eignen sie sich auch hervorragend als Adreßzeiger, da selbst der riesige Adreßbereich des 68000-Prozessors im Amiga damit vollständig abgedeckt wird.

Einfach genaue Fließkommazahlen und -variablen entsprechen nicht völlig denen des C64. Beim 64er wurden diese nämlich intern durch fünf Byte dargestellt, was etwas genauer ist als die Vier-Byte-Darstellung des Amiga. Normalerweise reicht diese Genauigkeit allerdings auch aus. Sollte das einmal nicht der Fall sein, so gibt es beim Amiga noch die Möglichkeit, doppelt genaue Fließkommavariablen zu verwenden. Diese besitzen eine interne Größe von acht Byte. Sie werden durch ein Nummernzeichen (#) am Ende gekennzeichnet. Einfach genauen Fließkommavariablen kann auch ein Ausrufezeichen angehängt werden, was allerdings nicht unbedingt erforderlich ist.

Die Endungen !, #, % und & können nicht nur bei Variablen, sondern auch bei Zahlen verwendet werden, um den entsprechenden Datentyp kenntlich zu machen. 1234& oder 43.1# sind also erlaubte Konstruktionen. Wenn man Fließkommazahlen in Exponentialdarstellung verwenden will, muß man bei einfach genauen Zahlen den Exponenten wie gewohnt durch ein E von der Mantisse trennen. Doppelt genaue Fließkommazahlen benötigen statt dessen ein D. Auch Hexadezimalzahlen und Oktalzahlen lassen sich verarbeiten. Ihnen wird ein &H beziehungsweise ein &O vorangestellt. Bei Oktalzahlen genügt auch ein &. Folgende Zahlen haben demnach den gleichen Wert:

```
&123 = &O123 = &H53 = 83
```

Im Gegensatz zum C64 wird eine Rechnung mit Ganzzahlvariablen auf dem Amiga schneller abgearbeitet als eine mit Fließkommawerten. Dies liegt daran, daß der C64 keine Integer-Rechenroutinen besitzt, sondern Ganzzahlwerte erst in Fließkommawerte umwandelt, um damit zu rechnen. Hinterher werden diese Werte dann wieder zurückgewandelt. Was beim C64 nicht möglich ist, jedoch beim Amiga erhebliche Zeitvorteile bringt, ist das Verwenden von Ganzzahlvariablen für FOR...NEXT-Schleifen. Die Befehlszeile:

```
FOR i%=1 TO 10000:NEXT
```

wird mehr als doppelt so schnell ausgeführt als:

```
FOR i=1 TO 10000:NEXT
```

Was die Umwandlung von Zahlen verschiedener Datenformate angeht, so ist alles erlaubt, was einigermaßen sinnvoll ist. Das bedeutet, nur eine Zuweisung einer zu großen Zahl an eine Variable, die Zahlen solcher Größe nicht aufnehmen kann, führt zu einer Fehlermeldung. Für negative Zahlen muß es natürlich der Absolutbetrag sein, der nicht zu groß ist. Ein Sachverhalt, der mit der Umwandlung von Fließkommazahlen in Integer-Zahlen zu tun hat, ist beim Amiga anders als beim C64. Der Befehl PEEK(100.5) liefert beim C64 den Inhalt der Speicher-

zelle 100, beim Amiga dagegen den Inhalt der Speicherzelle 101. Der Amiga rundet nämlich Fließkommazahlen bei allen Befehlen, die ganze Zahlen verlangen. Der C64 dagegen schneidet die Nachkommastellen einfach ab.

3.4 Der Befehlssatz

Nach so vielen Erklärungen zum neuen Editor und zum Programmtextaufbau kommen wir jetzt endlich zum interessanten Teil: dem Befehlssatz. Ehemalige C128-Besitzer, die ja durch ihr BASIC 7.0 schon sehr verwöhnt waren, werden sich in mancher Hinsicht umstellen müssen, da viele Befehle andere Namen bekommen haben, bestimmte Befehle nun ganz anders ablaufen und wiederum andere Befehle völlig fehlen. Ex-C64-User haben es da einfacher. Ihr BASIC V2 ist größtenteils im AmigaBASIC integriert. Wo jedoch die Unterschiede liegen und welche Befehle völlig neu hinzugekommen sind, sind die Themen der folgenden Kapitel.

3.4.1 Was ist gleich oder ähnlich geblieben?

Wie schon gesagt, finden sich die meisten Befehle des C64-BASIC im AmigaBASIC wieder. Unterschiede gibt es in erster Linie bei den Ein-/Ausgabebefehlen. Auf diese Befehle werden wir allerdings in den Kapiteln 3.4.2 und 3.4.3 noch einmal gesondert eingehen. Die Befehle, an denen sich etwas geändert hat oder die um interessante Zusätze erweitert wurden, sind im folgenden aufgelistet. Alle anderen Befehle, wie zum Beispiel die komplette Stringverwaltung, funktionieren im AmigaBASIC unverändert weiter.

CLOSE, OPEN

Aufgrund der Tatsache, daß der Amiga völlig anders mit Dateien und Geräten umgeht als der C64, ist es ganz natürlich, daß hier gravierende Unterschiede bestehen. Wir verweisen hier auf das Kapitel 3.4.3 über die Dateiverwaltung von BASIC aus.

CMD

Dieser Befehl existiert nicht. Er ist nicht mehr erforderlich. Hier gilt der Verweis auf Kapitel 3.4.2.

CLR

CLR ist durch CLEAR ersetzt worden. CLEAR alleine macht genau das gleiche wie CLR beim C64. Jedoch können Sie mit Hilfe der CLEAR-Anweisung auch die Größe des zur Verfügung stehenden Arbeitsspeichers verändern.

DEF FN

Alle Funktionsdefinitionen des C64-BASIC funktionieren unverändert weiter. Endlich ist es aber möglich, Funktionen beliebigen Datentyps, also auch Stringfunktionen, zu definieren. Dies geschieht durch Angabe der Typspezifikation im Funktionsnamen wie bei einer Variablen oder einer Zahl. Zudem ist die Anzahl möglicher Parameter, die in Klammern hinter dem Funktionsnamen angegeben werden, nicht auf einen einzigen Wert beschränkt. Es können beliebig viele Argumente angegeben werden, oder auch überhaupt keins. Diese Argumente brauchen zudem auch nicht numerisch sein, sondern es kann sich auch genausogut um Zeichenketten handeln. Zum Beispiel ist folgendes erlaubt:

```
"DEF FN einzeichen$(a$,n)=MID$(a$,n,1)"
```

FRE

Beim C64 ist es egal, welches Argument man der FRE-Funktion mitgibt. Sie liefert immer den noch zur Verfügung stehenden freien BASIC-Speicher. Beim Amiga tut sie das auch, solange man nicht die Argumente -1 oder -2 benutzt. FRE(-1) gibt Auskunft über den gesamten im System noch zur Verfügung stehenden Speicher, der momentan von keinem Programm benutzt wird. FRE(-2) liefert den Wert des momentan freien Stapelspeichers für BASIC-Programme, um unter anderem Rücksprungadressen abzulegen. Der Aufruf von FRE hat allerdings noch einen anderen, schon vom C64 her bekannten Effekt,

nämlich den, daß eine sogenannte Garbage Collection durchgeführt wird. Es handelt sich dabei um ein Aufräumen des Zeichenkettenspeichers. Wenn man beim C64 große String-Felder dimensioniert hat, kann die Garbage Collection Minuten, wenn nicht sogar Stunden dauern. Beim Amiga braucht man das nicht mehr zu befürchten. Hier benötigt die Garbage Collection allerhöchstens einen Augenblick.

GET, GET#

Diese Befehle dienen nur zur Verwaltung von relativen Dateien oder zum Zugriff auf den Grafikbildschirm, nicht jedoch zur Eingabe von Zeichen von der Tastatur oder aus einer Datei. Um Zeichen von der Tastatur einzulesen, verwendet man statt dessen die Funktion Inkey$, mit der sich allerdings nur Stringvariablen einlesen lassen. Beispielsweise wird aus der C64-Anweisung GET a$ beim Amiga a$=Inkey$. Als leistungsfähigeren Ersatz für GET# bietet AmigaBASIC die Funktion INPUT$. Genaueres hierzu finden Sie wiederum in Kapitel 3.4.3.

GOTO

Die beim C64 erlaubte, aber ungebräuchliche Form GO TO des GOTO-Befehls gibt es beim Amiga nicht mehr. GOTO funktioniert genau wie GOSUB nicht nur mit Zeilennummern, sondern auch mit Sprungmarken (Label). Man kann aber in AmigaBASIC aufgrund der vernünftigen Kontrollstrukturen fast immer ohne GOTO auskommen. Sie sollten aus Gründen der Übersichtlichkeit GOTO wirklich nur dann verwenden, wenn Sie eine Schleife in der Mitte verlassen wollen oder wenn dadurch eine wirkliche Vereinfachung herbeigeführt wird. GOTO kann nämlich sonst zum hervorragenden Instrument zur Schaffung von Verwirrung oder gar Chaos werden.

IF...THEN

Diese Befehle funktionieren zuerst einmal wie gewohnt. Sogar die Konstruktion IF...GOTO ist erlaubt. Hinzugekommen ist allerdings der ELSE-Befehl, der vielen von Ihnen schon aus BASIC-Erweiterungen oder vom C128 her bekannt sein dürfte.

Die hinter ELSE angegebenen Anweisungen werden immer dann ausgeführt, wenn die hinter IF angegebene Bedingung nicht erfüllt ist. Interessant beim Schreiben strukturierter Programme ist die Form des IF...THEN-Befehls, bei der direkt hinter der THEN-Anweisung kein weiterer Befehl steht. Dann werden nämlich alle folgenden Zeilen bis zur nächsten ELSE- oder END IF-Anweisung als zur THEN-Anweisung zugehörig angesehen. Entsprechendes gilt für die ELSE-Anweisung. Manchmal sind umständliche GOTO-Anweisungen beim C64 nur deshalb nötig, weil in der Zeile, in der der IF...THEN-Befehl steht, nicht mehr genug Platz ist. Dieses Problem kann hier nicht mehr auftreten.

INPUT

Neben der vom C64 her bekannten Verwendung des INPUT-Befehls gibt es beim Amiga ein paar kleine Zusätze, die einige der Nachteile, die dieser Befehl beim C64 schon immer gehabt hat, ausgleichen. Das manchmal unerwünschte Fragezeichen, das bei der Ausführung dieses Befehls auf dem Bildschirm erscheint, kann dadurch unterdrückt werden, daß man statt des Semikolons nach dem Ausgabetext ein Komma verwendet. Wenn direkt nach der INPUT-Anweisung ein Semikolon steht, springt der Cursor nach der Eingabe nicht in die nächste Zeile. Dies entspricht dem PRINT-Befehl mit Semikolon am Ende. Will man Strings mit vorhergehenden oder nachfolgende Leerzeichen oder Kommas eingeben, steht man vor dem gleichen Problem, das man schon vom alten 64er her kennt. Man kann dieses Problem einerseits genauso wie beim C64 lösen, nämlich indem man die Zeichenkette in Anführungszeichen eingeschlossen eingibt. Andererseits kann man auch den Befehl LINE INPUT verwenden, mit dem sich nur eine einzige Zeichenkette einlesen läßt, diese allerdings vollständig.

LOAD, SAVE, VERIFY

Mit den Befehlen LOAD und SAVE läßt sich im Prinzip genauso arbeiten wie beim C64, nur ohne Angabe einer Gerätenummer. Statt dessen wird das entsprechende Diskettenlaufwerk im FILE-NAMEN nach den üblichen Regeln spezifiziert. Die Anweisung VERIFY gibt es nicht mehr, man benötigt sie auch

nicht unbedingt, da das Diskettenlaufwerk nach jedem Schreibzugriff diesen automatisch verifiziert. LOAD arbeitet allerdings in einer Hinsicht anders als beim C64. Wird dieser Befehl nämlich innerhalb eines Programms verwendet, so muß, um das angegebene Programm nach dem Einladen sofort zu starten, nach dem FILE-NAMEN der Zusatz ,R angefügt werden. Variablen können auf diese Weise nicht übergeben werden. Dazu dient statt dessen der Befehl CHAIN. Genaueres hierzu finden Sie wiederum im Kapitel 3.4.2. Der SAVE-Befehl bietet übrigens die Möglichkeit, Dateien im ASCII-Format, also für jedes Textprogramm lesbar, abzuspeichern. Dazu geben Sie hinter dem FILE-NAMEN die Endung ,A an. Zudem können Sie Programme auch geschützt abspeichern, in dem Sie die Endung ,P verwenden. Ein solches Programm kann weder gelistet noch verändert werden. Hierbei ist allerdings Vorsicht geboten. Gehen Sie immer sicher, daß Sie ein geschütztes Programm auch noch in einer ungeschützten Version besitzen. Sonst nämlich kann es Ihnen passieren, daß Sie Ihr eigenes Programm nicht mehr ändern können.

Als Ergänzung zu LOAD bietet AmigaBASIC noch den Befehl MERGE an. Dieser hängt die angegebene Datei an das im Speicher befindliche Programm an. Dazu muß das Programm auf der Diskette unbedingt im ASCII-Format mit SAVE abgespeichert worden sein.

PEEK

An dieser Funktion hat sich zwar nichts geändert. Beim Amiga hat man aber als zusätzliche Hilfe die Befehle PEEKW und PEEKL zur Verfügung.(W = Word = zwei Byte; L = Longword = vier Byte.) Sie liefern dementsprechend 16- bzw. 32-Bit-Werte zurück. Als Argumente sind nur gerade Zahlen erlaubt. Beispiel: Folgende Befehle erfüllen bei beiden Rechnern den gleichen Zweck:

```
C64:   B=PEEK(A)+PEEK(A+1)*256
Amiga: B=PEEKW(A)
```

PI

Der Amiga kennt weder ein PI-Zeichen noch einen Ersatz dafür. Wenn Sie die Zahl Pi in einem Ihrer Programme oft benötigen, dann definieren Sie doch am Anfang eine Variable dafür:

```
PI=3.1415927
```

POKE

Hier gilt das gleiche wie bei PEEK, nur daß die Befehle POKEW und POKEL heißen.

RESTORE

Mit dieser Anweisung läßt sich der DATA-Zeiger nicht nur auf das erste DATA-Element zurückstellen. Durch RESTORE Zeilennummer oder RESTORE Sprungmarke wird der DATA-Zeiger auf das erste DATA-Element der angegebenen Zeile zurückgestellt. RESTORE ohne irgendeine Angabe klappt aber auch.

Return

Auch hier ist beim AmigaBASIC die Angabe einer Sprungmarke erlaubt. Dadurch wird das Unterprogramm verlassen, wobei das Programm nicht hinter der aufrufenden GOSUB-Anweisung fortgesetzt wird, sondern an der angegebenen Zeile.

ST

Die Statusvariable ST existiert nicht mehr. Fehler beim Lesen oder Schreiben führen grundsätzlich zu einer Fehlermeldung. Um aber das Dateiende beim Einlesen zu erkennen, gibt es jetzt die Funktion EOF (END Of FILE). EOF(Dateinummer) liefert den Wert "wahr" (-1), wenn das Dateiende erreicht wurde, ansonsten den Wert "falsch" (0).

SYS, USR

Statt SYS oder USR gibt es beim Amiga den Befehl CALL, um ein Maschinenprogramm zu starten. Um die Übergabe von Parametern mußte der Programmierer sich beim SYS-Befehl auf dem C64 selbst kümmern, und bei der USR-Funktion konnte nur eine Fließkommazahl übergeben werden. Ganz anders beim AmigaBASIC. Hier werden alle Werte, die beim CALL-Befehl angegeben werden, automatisch auf den Stapel gelegt. Da man auch Zeiger auf Fließkommavariablen oder Zeichenketten übergeben kann, hat das Maschinenprogramm problemlosen Zugriff auf Werte jeglicher Art. In Kapitel 3.6 werden wir speziell noch auf das Einbinden von Maschinenprogrammen in BASIC-Programme eingehen.

TI, TI$

Beide Systemvariablen gibt es noch in leicht abgewandelter Form. TI$ heißt jetzt TIME$ und besteht auch nicht mehr aus sechs, sondern aus acht Zeichen. Hinzugekommen sind nämlich die zwei Doppelpunkte, die Stunden, Minuten und Sekunden voneinander abtrennen. Das bedeutet, die Zeit 100435 beim C64 wird beim Amiga zu 10:04:35. Man kann auch nicht die Uhr durch Zuweisung an TIME$ neu stellen. Dies funktioniert nur über die Preferences-Einstellung der Workbench. TI hat seine Entsprechung in TIMER. TIMER liefert allerdings einen Fließkommawert in Sekundeneinheiten, während TI 60stel-Sekunden angibt. Die genaue Entsprechung wäre demnach: TI=INT(TIMER*60). Als Ergänzung zu TIME$ gibt es jetzt auch DATE$. DATE$ liefert das Datum in der Form mm-tt-jjjj. Auch dieser Wert läßt sich nur durch die Preferences-Voreinstellung ändern. TIME$ und DATE$ werden besonders dann interessant, wenn man die von Commodore angebotene Echtzeituhr besitzt, die auch dann weiterläuft, wenn der Rechner ausgeschaltet ist.

WAIT

Dieser Befehl existiert nicht. Man kann aber auch ganz gut ohne ihn auskommen. Als Ersatz bietet AmigaBASIC höchstens den SLEEP-Befehl, der auf das Eintreten diverser Ereignisse wie z.B. das Betätigen einer Maustaste wartet.

3.4.2 Befehle für allgemeine Ein-/Ausgabe

Natürlich gibt es beim Amiga die bekannten Befehle PRINT und INPUT zur Ein- und Ausgabe. Die Unterschiede zum C64 fangen allerdings schon an, wenn man ein Programmlisting auf dem Drucker ausgeben will. Dieses Kapitel beschreibt die Möglichkeiten der Ein- und Ausgabe des Amiga.

Vorher noch ein kurzer Hinweis: Ihnen ist ja vom CLI her sicherlich bekannt, daß beim Amiga Geräte- und Ordner-Bezeichnungen im FILE-NAMEN mitangegeben werden. AmigaBASIC kennt neben üblichen Gerätebezeichnungen wie DF0: oder RAM: noch vier weitere, die nicht vom DOS aus verwendbar sind. Sie lauten:

SCRN: Bildschirm

KYBD: Tastatur

LPT1: Drucker

COM1: RS232-(serielle) Schnittstelle mit Parameterangabe

Beginnen wir mit der Datenausgabe auf dem Bildschirm. Der PRINT-Befehl des C64 gibt einem die Möglichkeit, Farb- und Cursorsteuerzeichen zu verwenden. Darauf müssen wir beim Amiga zuerst einmal verzichten. Wie man trotzdem eine hervorragende Grafik, farbige Bildschirmmasken und anwenderfreundliche Menüs erzeugt, wird in den Kapiteln 3.4.4 und 3.4.5 erklärt. Wir beschränken uns hier auf die einfachen Befehle.

Beim Amiga gibt es zwar keine Cursorsteuerzeichen mehr, der Cursor läßt sich nur mit der LOCATE-Anweisung an eine bestimmte Position auf dem Bildschirm setzen. LOCATE Zeile, Spalte, LOCATE Zeile und LOCATE, Spalte sind erlaubte Schreibweisen. Die erste Zeile und die erste Spalte tragen die Nummer Eins und nicht Null.

Will man die aktuelle Cursorposition erfahren, so ruft man die Funktionen POS und CSRLIN auf. x=POS(0) liefert die Spaltennummer, y=CSRLIN die Zeilennummer.

Zum Löschen des Bildschirms dient der Befehl CLS. Das gleiche bewirkt auch der Aufruf PRINT CHR$(12);.

Als Ergänzung zum Befehl PRINT gibt es noch die beiden Befehle WRITE und PRINT USING. WRITE funktioniert fast wie PRINT, nur daß Kommas, die zwischen Variablen stehen, mitausgedruckt werden. Außerdem werden Strings in Anführungszeichen ausgegeben und Zahlenwerte ohne führendes Leerzeichen. PRINT USING dient zur formatierten Ausgabe von Variablen. Dazu wird vor den zu druckenden Daten ein Formatstring angegeben, das die Anordnung des Ausdrucks bestimmt. Es würde im Rahmen dieser kurzen Zusammenfassung zu weit gehen, alle Möglichkeiten aufzuzählen, die dieser Befehl bietet. Sie finden eine ausführliche Erklärung in Ihrem BASIC-Handbuch auf Seite 9/191.

Die Druckerausgabe ist beim Amiga sehr einfach. Man braucht keinen Kanal zum Drucker zu öffnen. Zu den Befehlen PRINT, PRINT USING und LIST existieren nämlich die Gegenstücke LPRINT, LPRINT USING und LLIST, die nicht auf den Bildschirm, sondern auf den Drucker wirken. LPOS(n) ist die Entsprechung zu POS(n) und gibt die Position des zuletzt gedruckten Zeichens an.

Beim C64 wandert der Cursor, wenn in einer Zeile das letzte Zeichen ausgedruckt wird, an den Anfang der nächsten. Beim Amiga ist das normalerweise nicht der Fall, da der Bildschirm so aufgefaßt wird, als hätte er keine rechtsseitige Begrenzung. Mit der WIDTH-Anweisung läßt sich aber ein rechter Rand

definieren, bei dessen Erreichen ein Wagenrücklaufcode ausgegeben wird. Für den Drucker beispielsweise ist eine Randbegrenzung von 80 voreingestellt. Mit WIDTH läßt sich zudem angeben, welche Abstände die Tabulatorstops besitzen, die von PRINT oder LPRINT angesprungen werden, wenn ein Komma als Trennzeichen verwendet wird. Folgende Syntax ist für WIDTH erlaubt:

WIDTH Breite, Tabulatorabstand

Die Druckbreite wird für den Bildschirm festgesetzt. Eines der beiden Argumente kann natürlich auch weggelassen werden.

WIDTH LPRINT Breite, Tabulatorabstand

Entsprechend für den Drucker.

WIDTH Gerätekennung, Breite,Tabulatorabstand

Entsprechend für ein Gerät (wie z. B. SCRN:)

WIDTH#Dateinummer, Breite, Tabulatorabstand

Entsprechend für ein Daten-File, das mit dem OPEN-Befehl öffnet wurde. Dazu finden Sie mehr im folgenden Kapitel.

Wenn als Breite beim WIDTH-Befehl eine 255 angegeben wird, wird keine Randbegrenzung angenommen. Dies ist z.B. die Voreinstellung für den Bildschirm.

Nach den Befehlen für die Ausgabe fehlen noch die für die Eingabe. Die Besonderheiten des INPUT-Befehls sowie der LINE-INPUT-Befehl wurden aber schon im Kapitel 3.4.1 beschrieben. Der einzige Eingabebefehl, der noch fehlt, ist INPUT$. INPUT$ ist eine Stringfunktion, die eine vorgegebene Anzahl von Zeichen von der Tastatur oder aus einem mit OPEN geöffneten File liest. Der File zugriff wird im nächsten Kapitel erklärt. Die Tastatureingabe sieht so aus: Nach dem Aufruf von a$=INPUT$(n) erwartet der Amiga eine Eingabe von n Zeichen.

Diese Zeichen werden nicht auf dem Bildschirm angezeigt, und die Eingabe kann nicht mit Return beendet werden. Ein Abbruch ist nur mit Amiga-Punkt oder Ctrl-C möglich.

Zur allgemeinen Ein-/Ausgabe zählen auch die Befehle LOAD, SAVE und MERGE. LOAD eignet sich aber nicht besonders für die sogenannte Overlay-Technik, bei der es sich um das wechselseitige Nachladen von Programmteilen, um Speicherplatz zu sparen handelt. LOAD übergibt nämlich im Gegensatz zum 64er keine Variablen an das nachgeladene Programm. Beim C64 muß zudem darauf geachtet werden, daß der nachgeladene Programmteil kürzer ist als der, der schon im Speicher steht. Außerdem startet ein nachgeladenes Programm dort grundsätzlich am Anfang.

Der Amiga dagegen besitzt den Befehl CHAIN, der die Overlay-Technik unterstützt. Folgende Befehlsaufrufe stehen zur Verfügung:

```
CHAIN "File-Name",Zeilennummer,ALL
```

Das nachgeladene Programm wird ab der angegebenen Zeilennummer gestartet. Die Angabe eines Labels statt einer Zeilennummer ist leider nicht erlaubt. Wird die Zeilennummer weggelassen, so wird das Programm am Anfang gestartet. Der Zusatz ALL sagt aus, daß alle Variablen des aufrufenden Programms an das nachgeladene übergeben werden. Will man nur einen Teil oder überhaupt keine der Variablen übergeben, so läßt man diesen Zusatz weg. Welche Variablen dann tatsächlich übergeben werden, wird mit der COMMON-Anweisung bestimmt. COMMON x,y,a() markiert z.B. die Variablen x und y sowie das Feld a für die Übergabe.

```
CHAIN MERGE "File-Name",Zeilennummer,ALL,DELETE Bereich
```

Das nachzuladende Programm wird, wie bei MERGE, ab der angegebenen Zeile des aufrufenden Programms überlagert. Hier sollte die Zeilennummer sinnvollerweise immer angegeben werden. Das nachzuladende Programm muß als ASCII-Datei abgespeichert worden sein (siehe Kapitel 3.4.1, LOAD/SAVE). Der

Zusatz DELETE gibt einen Bereich an, der nach der Überlagerung gelöscht wird. Die Bereichsangabe geschieht dabei wie beim LIST-Befehl. Er kann genauso wie der Zusatz ALL weggelassen werden.

3.4.3 Befehle für die Dateiverwaltung

Wie schon angemerkt, funktioniert die Dateiverwaltung auf dem Amiga etwas anders als auf dem C64. Trotzdem sind zumindest von der BASIC-Ebene aus wesentliche Strukturen erhalten geblieben, gerade was Art und Zugriffsmöglichkeiten der Files betrifft. Da Dateiverwaltung ein unentbehrliches Hilfsmittel ist, sollen in diesem Kapitel gerade den C64-Anwendern die Unterschiede zu dem, was sie bisher gekannt haben, erklärt werden.

Beim C64 oder C128 ist alles noch ein wenig einfacher. Da die Diskettenstationen 1541 bzw. 1570/71 "intelligente" Laufwerke sind, benötigt man zur Ansteuerung im Prinzip nur vier Befehle. Man braucht eine Anweisung, um einen Datenkanal zu eröffnen, eine, um Daten zum Laufwerk zu senden, eine, um Daten vom Laufwerk zu holen, und eine, um den Datenkanal wieder zu schließen. Damit hat man alle Möglichkeiten im Griff. Man kann so beispielsweise Files auf der Diskette löschen, indem man den Befehl S:NAME über den Kanal schickt. Um die Befehlsausführung kümmert sich das Laufwerk. Dies hat den ganz großen Vorteil, daß alle Floppy-Befehle von jeder Ebene, auf der man programmiert, erreichbar sind, also sowohl von BASIC als auch von Maschinensprache oder von jeder anderen Programmiersprache, die die standardmäßigen Ein-/Ausgabebefehle zur Verfügung stellt. Man braucht sich also nicht umzustellen, wenn man weiß, wie die Entsprechungen der BASIC-Befehle OPEN, CLOSE, PRINT# und GET# in der jeweiligen anderen Programmiersprache lauten.

Beim Amiga dagegen befindet sich, wie schon an anderer Stelle erklärt, der Diskcontroller direkt im Rechner. Daher sind die eigentlichen Befehle zur Diskettenverwaltung direkt im Betriebssystem des Amiga verankert. Solche Betriebssystemroutinen stellt aber einem normalerweise eine Programmiersprache nicht direkt

zur Verfügung, mit Ausnahme von C und natürlich Assembler. Um wieder auf das Beispiel eines zu löschenden Files zurückzukommen: Der Befehl, um ein Daten-File von einer Diskette zu entfernen, ist in BASIC ein anderer als vom CLI aus und von diesem wieder ein anderer als in C oder Assembler. Man kann diesem Manko abhelfen, da es möglich ist, CLI-Befehle von BASIC oder Assembler aus aufzurufen, darüber finden Sie allerdings mehr in Kapitel 3.5.2.

Beginnen wir also mit den einfachen Befehlen, die noch nichts mit komplexer Dateiverwaltung zu tun haben. Wir gehen allerdings davon aus, daß Sie sich schon ein wenig mit der Bedienung des AmigaDOS auskennen und wissen, daß mit DF0: das eingebaute Laufwerk gemeint ist. Falls das nicht der Fall ist, finden Sie diese Informationen im Kapitel 1 dieses Buches. Nun aber die Befehle:

Files

Dieser Befehl zeigt das aktuelle Directory einer Diskettenstation an. Im Gegensatz zu LOAD$,8 beim C64 wird hier natürlich nicht der Programmspeicher gelöscht. Files funktioniert genau wie DIR oder LIST im CLI, der FILE-NAME muß aber in Anführungszeichen stehen. Files df0:Basicdemos zeigt also den Inhalt des Ordners Basicdemos auf der Diskette im eingebauten Laufwerk an.

CHDIR

Mit CHDIR wird ein anderes aktuelles Disk-Verzeichnis angewählt. Dieser Befehl ist das Gegenstück zum CD-Befehl des CLI. Wenn Sie beispielsweise Ihre selbsterstellten BASIC-Programme nicht auf der Diskette abspeichern wollen, auf der sich der Interpreter befindet, so sollten Sie vorher CHDIR df0: eingeben. Dadurch wird das Verzeichnis der im Laufwerk DF0 eingelegten Diskette zum aktuellen Verzeichnis, und Sie sparen sich die Angabe df0: bei jedem SAVE-Befehl.

NAME

NAME df0:alter_Name AS df0:neuer_Name ändert den Namen der Disk-Datei alter_Name in neuer_Name um. Die Entsprechung im CLI ist RENAME. NAME ändert allerdings auch den Namen des zugehörigen Info-Files.

KILL

Dieser Befehl löscht ein File auf der angegebenen Diskette. DELETE ist das CLI-Gegenstück dazu. Es können keine Jokerzeichen angegeben werden. Außerdem wirkt KILL genau wie NAME auch auf das zugehörige Info-File.

Beim C64 gibt es die Möglichkeit, Disketten zu initialisieren, zu validieren und zu formatieren. Beim Amiga gibt es die ersten beiden Möglichkeiten nicht, da sich das Betriebssystem automatisch um die entsprechenden Aufgaben kümmert. Um beim Amiga eine Diskette zu formatieren, können Sie jederzeit die Initialize-Anweisung aus dem Workbench-Menü aufrufen, sogar dann, wenn gerade ein BASIC-Programm abläuft. Aus diesem Grund gibt es keinen Formatier-Befehl im AmigaBASIC.

Nachdem wir kurz die grundlegenden Befehle, die man zum Arbeiten mit Dateien benötigt, erläutert haben, können wir nun zur eigentlichen Dateiverwaltung übergehen. Bei den folgenden Erklärungen gehen wir davon aus, daß Sie schon ein wenig Dateiverwaltung auf ihrem C64 betrieben haben.

Laut Handbuch gibt es beim Amiga zwei Arten von Dateien, nämlich sequentielle und relative (Direktzugriffsdateien). Dies entspräche dem, was wir bereits auf dem C64 kennengelernt haben. Hier gibt es ja die sequentiell aufgebauten PRG-, SEQ- und USR-Dateien und die relativen REL-Dateien. Beim C64 sind diese beiden Dateiarten auch wirklich völlig unterschiedlich aufgebaut. Das Betriebssystem des Amiga kennt aber tatsächlich nur einen einzigen Dateityp. Das hat zur Folge, daß man eine Datei, die man als sequentielle erzeugt hat, genausogut mit den Befehlen für den relativen Zugriff wieder einlesen kann und umgekehrt. Andererseits hängt die Datensatzlänge einer Direktzugriffsdatei nicht davon ab, mit welchem Längenwert man

diese Datei erzeugt hat, sondern davon, mit welchem Wert man sie zum Lesen eröffnet. Meistens ist es natürlich sinnvoller, jeweils die gleiche Satzlänge zu verwenden.

Wie eröffnet man nun eine Datei? Es gibt die Möglichkeit, ein File zur Eingabe (input), Ausgabe (output), zum Anfügen an schon bestehende Daten (append) und als Direktzugriffsdatei (relativ) zum gleichzeitigen Schreiben und Lesen zu öffnen. Dem Daten-File, das durch seinen Namen angegeben wird, wird eine Dateinummer (logische File-Nummer) zugeordnet. Sie merken schon: im Prinzip ist alles genau wie beim C64. Der zugehörige Befehl sieht dann z.B. so aus:

Beim sequentiellen Zugriff:

```
Entweder: OPEN "NAME" FOR OUTPUT AS #1
oder    : OPEN "O",#1,"NAME"
(Beide Schreibweisen sind gleichwertig.)
```

Statt OUTPUT kann auch INPUT oder Append verwendet werden, je nachdem, ob die Datei zum Schreiben, Lesen oder zum Anhängen von Daten verwendet wird. Bei der zweiten Schreibweise muß dann statt des O ein I oder ein A stehen. #1 gibt die File-Nummer an, die wie beim C64 den Bezug des OPEN-Befehls zu allen weiteren Datei-Befehlen herstellt.

Beim Direktzugriff:

```
Entweder: OPEN "NAME" AS #1 LEN 100
oder    : OPEN "R",#1,"NAME",100
(Auch hier: beides ist gleichwertig.)
```

Durch Weglassen des FOR-Teils bei der ersten Schreibweise bzw. durch Verwenden des Buchstabens R bei der zweiten wird kenntlich gemacht, daß es sich um eine relative Datei handelt. Die Länge eines Datensatzes beträgt in diesem Beispiel 100. Sie kann im Gegensatz zum C64 zwischen 1 und 32767 liegen. Außerdem kann und sollte sie auch bei einem Lesezugriff auf eine Datei mitangegeben werden, da der Amiga sonst einen Vorgabewert von 128 annimmt. Sie kann übrigens auch bei sequentiellen Dateien angegeben werden, was sich auf die Größe des

intern verwendeten Datenpuffers auswirkt. Ein größerer Datenpuffer erhöht natürlich die Geschwindigkeit des Zugriffs, geht aber zu Lasten des Speicherplatzes.

Geschlossen werden alle Dateien mit dem CLOSE-Befehl. Bei den obigen Beispielen müßte er also CLOSE #1 oder CLOSE 1 lauten. Auch beim OPEN-Befehl ist das Nummernzeichen (#) optional, man sollte es aber zur Verdeutlichung ruhig verwenden.

Für sequentiellen Filezugriff gibt es natürlich die Befehle PRINT#, PRINT#...USING, INPUT# und LINE INPUT#, die hier wohl nicht mehr erklärt zu werden brauchen. GET# allerdings fehlt in der Bedeutung, in der man diesen Befehl normalerweise kennt. Dafür gibt es die Funktion INPUT$, die auch mehr als nur ein Zeichen aus einer Datei lesen kann. Ein Beispiel für ihre Anwendung:

```
a$=INPUT$(10,#1)
```

Hier werden 10 Zeichen aus der Datei mit der Nummer 1 gelesen und der Variablen a$ zugewiesen. Damit hat man an sich schon alles zusammen, was man zur Verarbeitung sequentieller Dateien benötigt. Der Amiga bietet einem aber noch ein paar andere Befehle an, um seine Daten durch die Gegend zu schieben. Hier eine kurze Zusammenfassung:

EOF

Der Aufruf der Funktion EOF(Dateinummer) liefert den Wert "wahr" (-1) zurück, wenn alle Daten einer sequentiellen Datei gelesen wurden, ansonsten den Wert "falsch" (0).

LOC

LOC(Dateinummer) ist eine Funktion, die bei relativen Dateien die Nummer des zuletzt gelesenen oder geschriebenen Datensatzes angibt. Da der Amiga auch bei sequentiellen Dateien einen Datenpuffer anlegt, der einem Datensatz bei einer relativen Datei entspricht (siehe: Eröffnen einer Datei), liefert diese

Funktion bei sequentiellen Files die Anzahl der gelesenen/geschriebenen Datenbytes dividiert durch die Pufferlänge. Wenn man bei seiner Datei keine Pufferlänge angegeben hat (was ja eigentlich auch nicht nötig ist), so gibt der Amiga einen Wert von 128 Bytes vor.

LOF

LOF(Dateinummer) liefert die Länge des entsprechenden Daten-Files in Bytes.

WRITE#

Dieser Befehl funktioniert fast genauso wie der PRINT#-Befehl. Er vereinfacht allerdings das spätere Einlesen der Daten mit INPUT#, da er Kommas zwischen Variablen ausgibt, wenn diese durch Kommas statt durch Semikolons getrennt werden. Außerdem werden Zeichenketten in Anführungszeichen eingeschlossen, wodurch Leerzeichen und Kommas in diesen Zeichenketten per INPUT# problemlos eingelesen werden.

Bei Direktzugriffsdateien werden die Daten in Datensätze gleicher Länge unterteilt. Mit der Anweisung GET #Dateinummer,Datensatznummer wird ein Datensatz in den vom BASIC eingerichteten Puffer gelesen. Läßt man die Datensatznummer weg, wird der auf den zuletzt gelesenen Datensatz folgende Satz geholt. Das Gegenstück zu GET ist PUT. PUT #Dateinummer,Datensatznummer schreibt den Pufferinhalt in den angegebenen Datensatz. Normalerweise läßt man hier die Satznummer aber weg, sofern man einen gelesenen und modifizierten Datensatz auch an die Stelle ins File zurückschreiben will, an der er vorher stand.

Wie greift man nun auf den Datenpuffer zu? Zuerst einmal kann man, genauso wie bei einer sequentiellen Datei, mit den Befehlen PRINT#, WRITE#, INPUT$ usw. arbeiten. Sinnvoller und komfortabler aber ist es, mit Hilfe der FIELD-Anweisung Puffer-Variablen zu definieren, über die dann der Datensatz angesprochen wird. Folgendes Beispiel zeigt, wie das funktioniert:

```
FIELD #1,10 AS A$,5 AS B$,25 AS C$
```

A$ wird hierdurch zu einem String der Länge 10, B$ zu einem der Länge 5 und C$ zu einem der Länge 25. Wird mit einem GET-Befehl ein Datensatz von einer Mindestlänge 40 eingelesen, so sind die Bytes des Satzes sofort in den Variablen A$, B$ und C$ verfügbar. Will man Daten in den Datensatz schreiben, darf man jedoch keine normale Zuweisung an diese Stringvariablen verwenden, sondern nur Zuweisungen mit den Befehlen LSET und RSET. Diese bewirken fast das gleiche wie der Befehl LET. LSET nimmt eine Stringzuweisung vor, bei der die zugewiesenen Daten linksbündig in die Variable eingetragen werden. Die Stringlänge der Variablen wird dabei nicht verändert, das heißt, zuviele Daten werden ignoriert, zuwenige Daten durch Leerzeichen ergänzt. RSET weist die Daten dementsprechend rechtsbündig an den String zu.

Die FIELD-Anweisung arbeitet nur mit String-Variablen. Wollen Sie allerdings Zahlenwerte in Ihren Datensätzen unterbringen, so müssen diese zuerst in Strings umgewandelt werden. Beim Auslesen der Datensätze wandeln Sie die Strings wieder in Zahlen zurück. Anstatt aber mit den Funktionen STR$ und VAL zu arbeiten, verwenden Sie besser die Funktionen MK?$ und CV?. Das Fragezeichen steht für einen der Buchstaben D,I,L oder S. Die MK?$-Funktionen liefern einen String, der die Bytes der internen Darstellung der als Argument angegebenen Zahl enthält. Das bedeutet, eine kurze Ganzzahl wird grundsätzlich in einen String der Länge zwei umgewandelt, eine einfach genaue Fließkommazahl immer in einen String der Länge vier usw. Die Funktionen CV? bilden das Gegenstück dazu, das heißt, sie wandeln einen String der entsprechenden Länge in eine Zahl zurück. Im folgenden nun die vollständige Auflistung dieser Funktionen:

a$=MKI$(n) Kurze Ganzzahl in String der Länge zwei wandeln

a$=MKL$(n) Lange Ganzzahl in String der Länge vier wandeln

a$=MKS$(n) Einfach genaue Zahl in String der Länge vier wandeln

a$=MKD$(n)	Doppelt genaue Zahl in String der Länge acht wandeln
n=CVI(a$)	Zwei-Byte-String in kurze Ganzzahl wandeln
n=CVL(a$)	Vier-Byte-String in lange Ganzzahl wandeln
n=CVS(a$)	Vier-Byte-String in einfach genaue Zahl wandeln
n=CVD(a$)	Acht-Byte-String in doppelt genaue Zahl wandeln

Wenn Sie innerhalb einer sequentiellen Datei Zahlenwerte ablegen wollen, so können Sie natürlich einerseits mit den Befehlen PRINT# und INPUT#, wie schon beschrieben, arbeiten. Platzsparender ist es jedoch, genau wie bei einer relativen Datei, die Zahlen vorher mit den MK?$-Funktionen in Strings umzuwandeln. Beispielsweise belegt die Zahl kurze Ganzzahl 30000 sechs Bytes, wenn man sie mit der PRINT#- oder WRITE#-Anweisung in ein File schreibt ,nämlich fünf Ziffern und ein Trennzeichen wie Komma oder Zeilenende. MKI$(30000) benötigt nur zwei Bytes. Man spart in der Tat auch das Trennzeichen, da man beim Einlesen weiß, daß eine kurze Ganzzahl genau zwei Zeichen, nicht mehr und nicht weniger, enthält. Das Einlesen muß dann aber unbedingt mit der Funktion INPUT$ erfolgen.

3.4.4 Grafik mit AmigaBASIC

Beim C64 war es eine beliebte Methode, Menüs und Bildschirmmasken mit Hilfe der auf der Tastatur zur Verfügung gestellten Grafikzeichen aufzubauen. Solche Grafikzeichen wird der C64-Anwender auf dem Amiga zuerst einmal vermissen. Trotzdem gibt das AmigaBASIC einem mehr Möglichkeiten zur grafischen Gestaltung, als man jemals auf dem C64 zur Verfügung gehabt hat. Man muß nur wissen, wie diese Möglichkeiten ausgenutzt werden können.

Zum Verständnis der meisten Grafikbefehle ist es unbedingt erforderlich, einige wesentliche Hardware-Konzepte des Amiga verstanden zu haben. Sie finden dazu eine ausführliche Erklärung in Kapitel 4. Hier das Wichtigste in Kürze:

Der Amiga besitzt keinen Textbildschirm, der mit der Zeichendarstellung des C64 vergleichbar wäre, sondern nur eine Grafikdarstellung, die der Bit-Map-Darstellung des 64ers entspricht. In diesen Grafikbildschirm werden die Bytes, aus denen ein Zeichen besteht, einzeln hineingeschrieben, während beim C64 für jedes Zeichen auf dem Bildschirm ein Byte im Bildschirmspeicher steht. Für den Programmierer bedeutet das, daß sich Grafik und Text beim Amiga beliebig mischen lassen, was für die flexible Fensterverwaltung sowieso unerläßlich ist.

Die Farbverwaltung des Amiga stellt alles in den Schatten, was man vom C64 her kennt. Zwar hat man auf dem normalen Arbeitsschirm nur vier Farben zur Verfügung, wohingegen der C64 alle seine 16 Farben anbietet. Nichts hindert einen aber daran, sich einen Grafikbildschirm mit bis zu 32 verschiedenen Farben zusammenzustellen. Dabei läßt sich die Farbe jedes Punktes unabhängig von allen anderen Punkten auf dem Bildschirm bestimmen. Die 32 Farben können aus einer Farbpalette von 4096 Farben bestimmt werden. Dies alles geschieht in einer Auflösung von mindestens 320 mal 256 Punkten. Beschränkt man sich auf maximal 16 Farben, so kann man auch mit einer Auflösung von 640 Punkten in der Breite arbeiten, während man, um beim C64 überhaupt Punkte einigermaßen unabhängig von der Farbe anderer Punkte setzen zu können, die Auflösung auf 160 mal 200 reduzieren muß. Weiterhin läßt sich beim Amiga die vertikale Auflösung durch den sogenannten Interlace-Modus auf 512 Zeilen erhöhen. Auf einem normalen Farbmonitor stört das hierbei auftretende Flackern doch merklich. Dieser Modus ist aber hervorragend für professionelle Anwendungen geeignet, wenn man einen Schwarzweiß- oder Farbmonitor mit langer Nachleuchtdauer besitzt. Es gibt noch andere Darstellungsmodi mit mehr Farben, die allerdings nicht ohne weiteres von BASIC aus erreichbar sind.

Ein wichtiges Element der grafischen Darstellung auf dem Amiga sind die sogenannten Screens. Jeder Screen stellt praktisch einen eigenen Bildschirm dar, mit einer eigenen Grafikauflösung, eigenen Farben und eigenen Windows. Screens können genau wie Windows Vordergrund-/Hintergrund-Symbole enthalten. Wird ein Screen an seiner oberen Leiste herunter-

gezogen, verschieben sich auch alle Windows, die zu diesem Screen gehören. Ein Screen, mit dem jeder schon einmal gearbeitet hat, ist der Workbench-Screen.

Kommen wir nun zu den Grafikbefehlen des AmigaBASIC. Alle diese Befehle beziehen sich normalerweise auf das Standard-Ausgabefenster. Man kann aber auch eigene Fenster einrichten, dazu aber später. Bei vielen Befehlen brauchen nicht unbedingt immer alle Parameter angegeben zu werden. Es müssen aber genügend Kommas gesetzt werden, so daß eindeutig wird, welcher Wert welchen Parameter bezeichnet.

Wie gesagt kann jeder Screen bis zu 32 verschiedene Farben enthalten. Diese Farben sind in Farbregistern mit Nummern von null bis 31 abgelegt. Der normale Workbench-Screen enthält vier Farben und benutzt demnach die Farbregister null bis drei. Da diese Farben aus einer Palette von 4096 Farben ausgewählt werden können, muß es einen Befehl geben, der diese Auswahl trifft. Er trägt den Namen PALETTE und wird folgendermaßen benutzt:

```
PALETTE Farbnummer, Rotanteil, Grünanteil, Blauanteil
```

Farbnummer ist die Nummer, die angibt, welches der 32 Farbregister geändert wird. Für den Workbench-Screen darf man hier Werte von null bis drei einsetzen. Null ist normalerweise die Hintergrundfarbe, eins die Vordergrundfarbe. Die Rot-, Grün- und Blauanteile sind Werte zwischen 0.00 und 1.00, die die prozentualen Anteile der Grundfarben Rot, Grün und Blau in der gewählten Farbe bezeichnen.

Der COLOR-Befehl legt fest, welches Farbregister beim PRINT-Befehl für die Vordergrundfarbe und welches Farbregister für die Hintergrundfarbe zuständig ist. Damit läßt sich beispielsweise invertierte Schrift erzeugen. Es gibt auch einige andere Befehle, die die hiermit angegebenen Farben benutzen. Der Aufruf von COLOR geschieht so:

```
COLOR Vordergrundfarbe, Hintergrundfarbe
oder: COLOR Vordergrundfarbe
oder: COLOR , Hintergrundfarbe
```

Folgendes Beispielprogramm zeigt einige Möglichkeiten, die der COLOR-Befehl bietet:

```
COLOR 2,0:print "Farbe 2"
COLOR 3,0:print "Farbe 3"
COLOR 2,3:print "bunt"
COLOR 0,1:PRINT "revers"
COLOR 1,0:PRINT "normal"
```

Die im folgenden erklärten Grafikbefehle LINE, PSET, PRESET, CIRCLE, PAINT und AREA benötigen alle Koordinatenangaben in der Form (x,y). Normalerweise geben diese Koordinaten die Pixel-Spalte bzw. -Zeile innerhalb des aktuellen Ausgabefensters an. Vor diese Koordinaten läßt sich aber auch das Schlüsselwort STEP setzen. Die Koordinaten beziehen sich dann auf den letzten gezeichneten Punkt. Wir werden das nicht mehr explizit bei jedem Befehl hinzuschreiben. Ein Beispiel:

```
PSET (100,100)
PSET STEP (-10,20)
```

Der zweite PSET-Befehl bezieht sich auf den Punkt (90,120). Was PSET macht, wird weiter unten erklärt.

Um Linien oder Rechtecke zu zeichnen, gibt es den Befehl LINE. Folgende Möglichkeiten zur Benutzung gibt es:

```
LINE (x1,y1)-(x2,y2), Farbnummer
```

Es wird eine Linie in der entsprechenden Farbe vom Punkt mit den Koordinaten x1,y1 zum Punkt x2,y2 gezogen. Wenn die Farbnummer weggelassen wird, verwendet AmigaBASIC die mit COLOR eingestellte Vordergrundfarbe. Läßt man das erste Koordinatenpaar weg, wird die Linie beginnend beim letzten gezeichneten Punkt gezogen. Damit lassen sich auf einfache Weise zusammenhängende Linienzüge zeichnen.

```
LINE (x1,y1)-(x2,y2), Farbnummer,B
LINE (x1,y1)-(x2,y2), Farbnummer,BF
```

Hiermit werden Rechtecke gezeichnet. Die Koordinaten geben zwei gegenüberliegende Eckpunkte des Rechtecks an. Wenn es sich um ausgefüllte Rechtecke handeln soll, wird die Kennung BF verwendet, ansonsten nur B. Natürlich gilt auch alles das, was zum ziehen vonLinien gesagt wurde.

Beispiel 1:

```
REM "Beispielprogramm für Liniengrafik"
WHILE 0=0              :REM "Diese Bedingung ist immer wahr"
  FOR j=1 TO 3               :REM "Farbnummern durchzählen"
    FOR i=0 TO 639 STEP 4 :REM "x-Koordinate des Endpunktes"
      LINE (319,186)-(i,0),j          :REM "Linie zeichnen"
      IF INKEY$<>"" THEN abbruch      :REM "Test auf Taste"
    NEXT i
  NEXT j
WEND
abbruch:
END
```

Dieses Programm zeichnet ein Linienmuster in wechselnden Farben auf dem Bildschirm. Abgebrochen wird es durch einen beliebigen Tastendruck. Wenn Sie das Programm nicht mit RUN vom Ausgabefenster aus starten, klappt der Abbruch nur dann, wenn Sie vorher mit der Maus in dieses Fenster hineinklikken.

Beispiel 2:

```
REM "Menüdemo mit Rechtecken"
CLS
LINE (24,12)-(190,76),2,BF :REM "ausgefülltes Rechteck"

LINE (24,12)-(190,28),1,B  :REM "vier nicht ausgefüllte"
LINE (24,28)-(190,44),1,B  :REM "Rechtecke"
LINE (24,44)-(190,60),1,B
LINE (24,60)-(190,76),1,B
```

```
COLOR 1,2
LOCATE 3,5:PRINT "1. Dateien ansehen"  :REM "Beispieltexte"
LOCATE 5,5:PRINT "2. Datei löschen"
LOCATE 7,5:PRINT "3. Datei umbenennen"
LOCATE 9,5:PRINT "4. Programm beenden"
COLOR 1,0
```

Dieses Programm demonstriert, wie man mit LINE-Befehlen professionell wirkende Bildschirmmasken erstellen kann.

Welche Befehle bietet AmigaBASIC noch für die Grafik? Da wären zunächst die Befehle PSET und PRESET. PSET (x,y) setzt einen Punkt in der aktuellen Vordergrundfarbe, PRESET (x,y) setzt einen in der aktuellen Hintergrundfarbe. PSET(x,y),Farbe und PRESET(x,y),Farbe setzen einen Punkt in der angegebenen Farbe.

CIRCLE (x,y),r,Farbe,Start,Ende,Bildverhältnis

Zeichnet einen Kreis mit dem Radius r. Die Farbe kann weggelassen werden, dann wird die Vordergrundfarbe verwendet. Start und Ende sind zwei Winkel im Bogenmaß, die angeben, welcher Ausschnitt des Kreises wirklich gezeichnet werden soll. Normalerweise kann man sie aber weglassen, wodurch ein vollständiger Kreis gezeichnet wird. Wird das Bildverhältnis angegeben, kann man aus dem Kreis eine Ellipse machen. Dieser Wert gibt das Breiten-/Höhenverhältnis der Ellipse an.

PAINT (x,y),Farbe,Randfarbe

Malt eine Fläche in der angegebenen Farbe aus. Randfarbe ist die Farbe, die als Randbegrenzung der Fläche verwendet wird. Leider ist es nicht möglich, eine Fläche auszufüllen, die von mehr als einer Farbe begrenzt wird. Wie üblich, kann die Farbe weggelassen werden. Gibt man keine Randfarbe an, wird als Begrenzung dieselbe Farbe benutzt, mit der die Fläche ausgefüllt wird.

AREA (x,y)

Dient dazu, ein Polygon (d.h. einen geschlossenen Linienzug) zu definieren, der dann mit dem AREAFILL-Modus ausgefüllt wird. Der Amiga merkt sich alle mit AREA angegebenen Punkte seit der letzten AREAFILL-Anweisung und füllt dann den so definierten Linienzug mit der durch COLOR festgelegten Vordergrundfarbe aus, sobald man den Befehl AREAFILL oder AREAFILL 0 aufruft. Verwendet man dagegen den Aufruf AREAFILL 1, so wird das Polygon invertiert. Für den vierfarbigen Workbench-Screen bedeutet das, aus Farbe null wird Farbe drei, aus Farbe eins wird Farbe zwei und umgekehrt. Man sollte natürlich mindestens drei AREA-Anweisungen aufrufen, bevor man AREAFILL benutzt.

Bisher sind wir immer davon ausgegangen, daß Flächen vollständig ausgefüllt und Linien immer durchgehend gezeichnet werden. Mit der PATTERN-Anweisung läßt sich das ändern. PATTERN Linienmaske,Flächenmaske gibt an, welche Punkte in der Vordergrundfarbe und welche in der Hintergrundfarbe gezeichnet werden. Linienmaske ist ein 16-Bit-Wert, in dem jedes gesetzte Bit einen Punkt in der Vordergrundfarbe, jedes gelöschte Bit einen Punkt in der Hintergrundfarbe bedeutet. Dieser Wert ist für die Befehle LINE und CIRCLE relevant. Flächenmaske ist der Name eines mit DIM dimensionierten Feldes, das eine Anzahl von solchen 16-Bit-Masken enthält. Diese Anzahl muß unbedingt eine Potenz von zwei sein, also 1, 2, 4, 8, 16 usw. Da das erste Feldelement die Nummer Null hat, muß man mit dem zugehörigen DIM-Befehl ein Feld der Größe 0, 1, 3, 7 usw. vereinbaren. Die Flächenmaske bezieht sich auf alle flächenfüllenden Befehle wie PAINT oder AREAFILL.

Nachdem wir Ihnen die wichtigsten Grafikbefehle vorgestellt haben, kommen wir nun auf Windows (Fenster) und Screens (Bildschirme) zu sprechen. Von BASIC aus lassen sich maximal vier eigene Screens öffnen. Der dazu benötigte Befehl sieht so aus:

```
SCREEN n, Breite, Höhe, Tiefe, Modus
```

"n" ist die Nummer des Screens (von eins bis vier). Modus gibt die verwendete Auflösung an:

Modus	Auflösung
1	320 Spalten * 256 Zeilen
2	640 Spalten * 256 Zeilen
3	320 Spalten * 512 Zeilen
4	640 Spalten * 515 Zeilen

Breite und Höhe geben die tatsächliche Breite und Höhe des Screens an, die nämlich geringer sein können als die mit Modus vereinbarte Bildschirmauflösung. Tiefe gibt die Anzahl der Bitplanes (siehe Kapitel 5) des Screens an, woraus die Anzahl darstellbarer Farben resultiert:

Tiefe	Farbanzahl
1	2
2	4
3	8
4	16
5	32 (nur möglich bei Auflösung von 320 Spalten)

Die Anweisung SCREEN CLOSE n schließt den Screen n und alle Windows, die zu diesem Screen gehören.

Nun aber zu den Windows. Folgende Anweisung definiert ein Window:

```
WINDOW n, "Titel", (x1,y1)-(x2,y2), Typ, Screennummer
```

"n" ist wie bei SCREEN eine dem Fenster zugeordnete Nummer, die hier aber größer als zwei sein sollte, da das normale Ausgabefenster die Nummer eins trägt. Titel erklärt sich selbst und kann auch entfallen. (x1,y1) und (x2,y2) bezeichnen die gegenüberliegenden Ecken des Fensters. Läßt man sie weg, erhält das Fenster entweder die Größe des gesamten Screens, oder, falls dieses Fenster schon auf dem Bildschirm existiert, es behält seine ursprünglichen Werte.

Typ ist einer der folgenden Werte oder eine Summe dieser Werte:

Wert	Bedeutung
1	Fenstergröße kann mit der Maus verändert werden.
2	Fenster kann mit der Maus verschoben werden.
4	Fenster enthält Vorder- und Hintergrundsymbole.
8	Fenster kann mit der Maus geschlossen werden.
16	Fensterinhalt wird zwischengespeichert und wieder angezeigt, wenn das Window von einem anderen Window überlagert wurde.

Wird die Screen-Nummer angegeben, erscheint das Fenster auf dem entsprechenden selbstdefinierten Screen, ansonsten erscheint es auf dem Workbench-Screen. Dem Workbench-Screen ist NICHT die Nummer eins zugeordnet, wie im Handbuch behauptet wird.

WINDOW CLOSE n schließt das entsprechende Fenster wieder.

Beim Öffnen eines Windows wird einerseits das Fenster selbst, andererseits der Screen, zu dem dieses Window gehört, in den Vordergrund gebracht. Arbeitet man beispielsweise mit mehreren Screens und will man den Workbench-Screen nach vorne bringen, so genügt der Befehl WINDOW 1. Hierdurch wird praktisch das Standard-Ausgabefenster neu definiert, aber unter Beibehaltung aller bestehenden Parameter wie Größe oder Typ.

Nach dem Öffnen wirken nun alle Ausgabebefehle wie PRINT oder LINE auf dieses Fenster. Will man dagegen ein schon bestehendes Window zum aktuellen Ausgabefenster machen, ohne es dabei in den Vordergrund zu holen, benutzt man den Befehl WINDOW OUTPUT n.

Folgendes Beispielprogramm demonstriert, wie man einen Screen und ein Window öffnet, um die Farbgrafik des Amiga richtig auszunutzen:

```
SCREEN 1,320,230,5,1  :REM "32 Farben, 320*230 Punkte"
WINDOW 2,"Grafikdemo",(0,0)-(311,216),16,1
REM "Das Window ist etwas kleiner als der Screen,"
REM "da die Window-Begrenzungen auch Platz brauchen."
FOR i%=0 TO 15
  col=1/16*i%
  PALETTE i%,col,col,col:REM"grau"
  PALETTE i%+16,col,0,0 :REM"rot"
NEXT i%
FOR i%=0 TO 216 STEP 64
  FOR j%=0 TO 15
    LINE (0,i%+j%)-(311,i%+j%),j%
    LINE (0,i%+j%+16)-(311,i%+j%+16),15-j%
    LINE (0,i%+j%+32)-(311,i%+j%+32),16+j%
    LINE (0,i%+j%+48)-(311,i%+j%+48),31-j%
  NEXT j%
NEXT i%
WHILE INKEY$="":WEND
SCREEN CLOSE 1
```

Sobald Sie eine Taste drücken, wird das Programm beendet. Vorsicht! Falls Sie das Programm mit Ctrl-C oder Amiga-Punkt abbrechen, können Sie den Workbench-Screen nicht mehr mit der Maus erreichen. In diesem Fall hilft nur noch Amiga-L, um das LIST-Fenster und damit den Workbench-Screen in den Vordergrund zu bringen.

Diese Einführung in die Grafikbefehle des AmigaBASIC ist natürlich noch nicht vollständig. Sie ist aber vollständig genug, um jedem (Ex-)C64-Anwender einen schnellen Einstieg in die Grafik-Programmierung des Amiga zu ermöglichen. Weitergehende Informationen finden Sie selbstverständlich in den schon erwähnten Büchern AmigaBASIC und AmigaBASIC für Fortgeschrittene. Speziell zum Thema Grafik finden Sie bei DATA BECKER auch noch die Bücher Amiga Supergrafik und Amiga 3-D-Grafikprogrammierung.

3.4.5 Pull-Down-Menüs

Wie Sie schon im vorhergehenden Kapitel sehen konnten, ist es in AmigaBASIC ein leichtes, ansprechende Menüs mit den Grafikbefehlen zu erzeugen. Interessanter ist allerdings, wie man

selbst die bekannten Pull-Down-Menüs erzeugt, also die Menüs, die man aus der Menüleiste mit der Maus herunterzieht.

Normalerweise sind in der Menüleiste des BASIC die vom Interpreter angebotenen Menütitel Project, Edit, Run und Windows vorhanden. Man kann diese Menüs durch eigene ersetzen. Dazu dient die Anweisung MENU:

```
MENU Nummer, Menüpunkt, Status, "Titel"
```

Nummer ist eine Zahl zwischen eins und zehn und gibt die Nummer des Menütitels in der Menüleiste an.

Menüpunkt ist eine Zahl zwischen null und 19 und gibt die Nummer des Menüunterpunktes an, den man durch die MENU-Anweisung definiert. Wenn eine Null angegeben wird, wird der Menütitel neu festgelegt, und alle Menüpunkte werden gelöscht, sofern ein Titel angegeben wird. Will man dagegen nur den Status (s.u.) des Menüs ändern, ohne die Unterpunkte zu löschen, läßt man den Titel weg.

Status ist eine Zahl zwischen null und zwei. Eine Eins aktiviert das gesamte Menü oder den entsprechenden Menüpunkt auf normale Weise. Bei einer Null erscheint der Text in Geisterschrift, und das Menü (der Menüpunkt) läßt sich nicht anwählen. Eine Zwei dagegen versieht den entsprechenden Menüpunkt mit einem Haken als Kennzeichen. In diesem Fall muß der Titel des Menüpunktes mit mindestens zwei, besser drei Leerstellen beginnen. Dieser Statuswert ist nicht bei Menütiteln in der Menüleiste erlaubt, sondern nur bei den Unterpunkten.

Titel gibt den Titel des Menüs oder des Menüpunktes an, je nach Menüpunkt. Wie schon gesagt, kann man ihn weglassen, sofern man nur den Status eines Menüs oder Menüpunktes ändern will.

Beim Einrichten eines Menüs sollten Sie nicht versuchen, Teile der bestehenden Menüs des BASIC-Interpreters zu verändern. Dies kann nämlich unter ungünstigen Umständen zu einem vollständigen Systemabsturz führen. Besser ist es, wenn Sie diese

Menüs vollständig durch eigene ersetzen. Es kann natürlich sein, daß Sie weniger als vier Menüs benötigen. In diesem Fall geben Sie den überschüssigen Menütiteln einfach eine leere Zeichenkette als Namen. Um also beispielsweise das Windows-Menü zu löschen, benutzen Sie einfach den Befehl:

```
MENU 4,0,0,""
```

Wenn Sie wieder auf die normalen BASIC-Menüs zurückgreifen wollen, können Sie alles mit dem Befehl MENU RESET rückgängig machen.

Folgendes Beispiel richtet ein Menü ein, das von einem Textprogramm stammen könnte:

```
REM "Menü eines TextProgramms"
MENU 1,0,1,"Datei"
MENU 1,1,1,"Neu"
MENU 1,2,1,"Laden"
MENU 1,3,1,"Speichern"
MENU 1,4,1,"Drucken"
MENU 1,5,1,"Abbruch"

MENU 2,0,1,"Optionen"
MENU 2,1,1,"   Auto-Trenn"
MENU 2,2,1,"   Steuerzeichen anzeigen"
MENU 2,3,1,"   60 Zeichen/Zeile"

MENU 3,0,0,"":MENU 4,0,0,""
```

Hier werden zwei Menüs eingerichtet. Das Datei-Menü bietet fünf Unterpunkte für die wichtigsten Datei-Funktionen. Das Optionen-Menü bietet drei Unterpunkte, bei denen Platz für den Haken als Markierung gelassen wurde.

Ein solches Programm ist natürlich noch nicht vollständig. Man kann zwar nun mit der Maus in die Menüleiste zeigen, während man die rechte Taste gedrückt hält. Wenn man aber versucht, einen Menüpunkt auszuwählen, passiert nichts. Um die Menüauswahl zu erkennen, gibt es die Befehle ON MENU GOSUB, MENU ON/OFF/STOP und die MENU-Funktion.

ON MENU GOSUB Unterprogramm

Hiermit wird ein Unterprogramm definiert, das abgearbeitet wird, sobald irgendein Menüpunkt ausgewählt wird. Das Programm unterbricht dann seinen Ablauf, egal, wo es sich gerade befindet, und führt zuerst dieses Unterprogramm aus. Danach wird das laufende Programm wieder an der Stelle fortgesetzt, an der es unterbrochen wurde. Das Unterprogramm wird aber nicht schon bei der ON MENU GOSUB-Anweisung ausgeführt, wie man vielleicht annehmen könnte.

MENU ON

Diese Anweisung ist notwendig, um die Unterbrechung, die man mit ON MENU GOSUB definiert, zuzulassen. Ohne sie passiert bei der Menüauswahl nichts.

MENU OFF

Sobald man keine Menüauswahl mehr wünscht, führt man diese Anweisung aus. Sie bewirkt das Gegenteil von MENU ON.

MENU STOP

Diese Anweisung bewirkt fast das gleiche wie MENU OFF. Wird allerdings nach MENU STOP noch ein Menü ausgewählt, so merkt sich der Interpreter dieses und ruft das mit ON MENU GOSUB definierte Unterprogramm bei der nächsten MENU ON-Anweisung auf. MENU STOP wird auch automatisch ausgeführt, sobald eine Menüauswahl zum Aufruf des entsprechenden Unterprogramms führt. Dadurch wird eine Unterbrechungsschachtelung verhindert, die sonst zum Chaos führen könnte. Nach Verlassen des Unterprogramms wird die Menü-Unterbrechung wieder zugelassen, sofern nicht im Unterprogramm die Anweisung MENU OFF oder MENU STOP gestanden hat.

Zum Erkennen, welcher Menüpunkt gewählt wurde, dient die MENU-Funktion. MENU(0) liefert die Nummer des zuletzt gewählten Menütitels. Wurde seit Programmbeginn oder seit dem

letzten MENU(0)-Aufruf kein Menü gewählt, liefert diese Funktion den Wert Null. MENU(1) liefert die Nummer des zuletzt gewählten Menüpunktes.

Folgendes Beispielprogramm ist die Ergänzung zu dem oben abgedruckten Beispiel für ein Textprogramm-Menü:

```
at=0:sa=0:zz=0          '"Flags für Optionen"

ON MENU GOSUB menueauswahl
MENU ON

quit=0                  '"Wert Null für falsch"
WHILE NOT quit
WEND
MENU OFF
MENU RESET
END

menueauswahl:
  ON MENU(0) GOTO datei,optionen

datei:
  ON MENU(1) GOTO neu,laden,speichern,drucken,abbruch
  neu:
    CLS:PRINT "neu":RETURN
  laden:
    CLS:PRINT "laden":RETURN
  speichern:
    CLS:PRINT "speichern":RETURN
  drucken:
    CLS:PRINT "drucken":RETURN
  abbruch:
    CLS:PRINT "Abbruch":quit=-1:RETURN

optionen:
  ON MENU(1) GOTO trennen,steuerzeichen,zeichenzeile
  trennen:
    at=NOT at
    IF at THEN MENU 2,1,2 ELSE MENU 2,1,1
    RETURN
```

```
steuerzeichen:
  sa=NOT sa
  IF sa THEN MENU 2,2,2 ELSE MENU 2,2,1
  RETURN

zeichenzeile:
  zz=NOT zz
  IF zz THEN
    MENU 2,3,1,"   80 Zeichen/Zeile"
  ELSE
    MENU 2,3,1,"   60 Zeichen/Zeile"
  END IF
  RETURN
```

Dieses Programm demonstriert einige oft benutzte Techniken in Verbindung mit der Menüabfrage. Da wäre zuerst die Variable quit. Nachdem die Menüabfrage initialisiert wurde, wartet das Programm in einer Schleife darauf, daß quit einen anderen Wert als Null annimmt. Das eigentliche Programm ist die Unterroutine menueauswahl. Hier wird durch ON MENU(0) GOTO... die Auswahl getroffen, welches Menü aufgerufen wurde und welche Unterroutine entsprechend diesem Menü benutzt werden soll. Beachten Sie bitte den Unterschied: ON MENU(0) GOTO... ist das normale ON...GOTO..., das Sie schon vom C64 her kennen und bei dem tatsächlich eine Verzweigung ausgeführt wird. ON MENU GOSUB... dagegen entspricht NICHT dem ON...GOSUB... vom C64, da hier nicht sofort eine Verzweigung stattfindet, sondern erst dann, wenn ein Menüpunkt ausgewählt wird.

Jedes der den Menüs zugeordneten Unterprogramme enthält eine Abfrage, welcher Menüpunkt gewählt wurde, und für jeden Menüpunkt ein entsprechendes Unterprogramm. Wird einer der Unterpunkte aus dem Optionen-Menü gewählt, kann man das Ergebnis erst dann sehen, wenn man sich dieses Menü noch ein zweites Mal ansieht. Bei den ersten beiden Punkten wird jeweils der Haken gesetzt oder gelöscht, beim dritten Punkt wechselt die Anzeige von 60 Zeichen/Zeile auf 80 Zeichen/Zeile und wieder zurück.

Wir haben Ihnen in diesem Kapitel alle Möglichkeiten des BASIC in bezug auf Pull-Down-Menüs gezeigt. Was leider nur über die Nutzung von Betriebssystemfunktionen möglich ist, ist das Verwenden von Untermenüs innerhalb von einzelnen Menüs. Auch läßt sich das Amiga-Zeichen nicht innerhalb der Menüs unterbringen, wie es beispielsweise im Interpreter-Menü zu sehen ist. Trotzdem sind die Menübefehle des AmigaBASIC schon sehr hilfreich und in den meisten Fällen ausreichend.

3.4.6 Sonstige Befehle: Sound und Joystick-Abfrage

Beim C64 hatte man immer das Problem, Sound-Erzeugung und Joystick-Abfragen nur über komplizierte POKE-Sequenzen verwirklichen zu können. Im AmigaBASIC sieht das ganz anders aus. Hier gibt es dafür einfache und komfortable Befehle.

Der Amiga hat vier Tonkanäle, also eine Stimme mehr als der C64. Im Gegensatz zum 64er ist die Lautstärke jeder Stimme unabhängig von der jeder anderen. Die möglichen Frequenzen liegen im Bereich von 20 bis 15000 Hertz. Zudem kann die Wellenform jeder Stimme völlig frei definiert werden, während man beim C64 nur vier verschiedene Wellenformen zur Verfügung hatte, sofern man das Rauschen als Wellenform bezeichnet.

SOUND Frequenz,Dauer,Lautstärke,Stimme

Spielt einen Ton entsprechend den angegebenen Parametern. Die Frequenz wird in Hertz angegeben, die Dauer in Einheiten von 0.055 Sekunden. Der für Dauer angegebene Wert darf 77 nicht überschreiten. Lautstärke ist ein ganzzahliger Wert zwischen null (unhörbar) und 255 (sehr laut). Läßt man die Lautstärke weg, wird der Wert 127 verwendet. Stimme gibt die Stimmennummer von null bis drei an. Auch dieser Wert kann entfallen, wodurch dann die Stimme null ausgewählt wird.

Oft will man mehrere Töne gleichzeitig über verschiedene Kanäle spielen. Dazu dienen die Anweisungen SOUND WAIT und SOUND RESUME. Nach SOUND WAIT werden alle

SOUND-Anweisungen in einer Warteschlange abgelegt und erst dann abgespielt, wenn eine SOUND RESUME-Anweisung ausgeführt wird.

Die Wellenform einer Stimme läßt sich mit Hilfe der WAVE-Anweisung definieren. WAVE Stimme,Amplitudenfeld legt für die angegebene Stimme eine Wellenform entsprechend dem Feld Amplitudenfeld fest. Amplitudenfeld enthält mindestens 256 Werte im Bereich von -128 bis 127, die, wie der Name schon sagt, die Amplitude der Welle definieren. Statt des Feldes kann auch das Schlüsselwort SIN angegeben werden, wodurch eine Sinuswelle definiert wird.

Das nachfolgende Beispielprogramm demonstriert, wie man den WAVE-Befehl benutzt, und spielt dann eine selbst definierte Welle im Vergleich zu einer Sinuswelle:

```
DIM welle%(500):REM "500 Amplitudenwerte"
FOR i%=0 TO 500
  welle%(i%)=127*SIN(1/(.001+.003*i%))
NEXT i%
WAVE 0,welle% : REM "Welle für Stimme Null festlegen"
SOUND 500,20  : REM "Ton spielen"
FOR i%=1 TO 7000:NEXT
WAVE 0,SIN    : REM "Sinuswelle festlegen"
SOUND 500,20  : REM "Ton spielen"
```

Beim C64 hatte man die Möglichkeit, für jede Stimme eine Hüllkurve zu definieren, was den Klang wesentlich beeinflußte. Beim Amiga muß man die Hüllkurve von Hand programmieren, was allerdings wesentlich komplexere Hüllkurven erlaubt.

Zur Joystick-Abfrage gibt es zwei Befehle, je nachdem, ob man die Richtung oder den Feuerknopf eines Joysticks abfragen will. v=STICK(n) liefert die Richtungsinformation von einem der beiden Joystick-Anschlüsse entsprechend folgender Tabelle:

n	Information
0	Joystick 1 waagerecht
1	Joystick 1 senkrecht
2	Joystick 2 waagrecht
3	Joystick 2 senkrecht

Der Wert v hat dabei folgende Bedeutung:

```
-1  Joystick unten bzw. links
 0  Joystick nicht bewegt
 1  Joystick oben bzw. rechts
```

v=STRIG(n) liefert Informationen über den Feuerknopf beider Joysticks:

```
n          Bedeutung
0  v=1,    wenn Knopf von Joystick 1 seit letzter Abfrage
           gedrückt wurde, sonst v=0.
1  v=1,    wenn Knopf von Joystick 1 gerade gedrückt wird.
2  wie n=0, nur für Joystick 2
3  wie n=1, nur für Joystick 2
```

3.5 Benutzung der Betriebssystemroutinen

Beim C64 ist es eine beliebte Methode, daß man in seinen BASIC-Programmen diverse POKE- und SYS-Befehle benutzt, um so die Programme deutlich zu beschleunigen oder Funktionen auszuführen, die sonst überhaupt nicht möglich wären. Gemeint ist hier der Zugriff auf Betriebssystemadressen und -routinen. Beim Amiga ist der Zugriff auf Adressen des Betriebssystems von BASIC aus nur bedingt möglich und relativ kompliziert. Sehr einfach und viel effektvoller ist dagegen die Benutzung der Systembibliotheken (Libraries).

3.5.1 Öffnen und Benutzen von Libraries

Libraries sind Sammlungen von Betriebssystemroutinen. Jede LIBRARY hat einen Namen, der die Endung .library besitzt. Um auf eine dieser Bibliotheken zugreifen zu können, muß sie mit dem Befehl LIBRARY eröffnet werden. Dabei wird automatisch eine Sprungtabelle im Speicher angelegt, die die Einsprünge in die entsprechenden Routinen enthält. Den Zeiger auf den Beginn dieser Sprungtabelle merkt sich der Interpreter.

Normalerweise stehen Betriebssystemroutinen im Kickstart-ROM. Einige Bibliotheken sind aber extern auf der Workbench-Diskette und müssen erst nachgeladen werden. Das Betriebssystem holt sich diese Routinen aus dem Ordner libs der Workbench-Diskette. Es handelt sich dabei nur um Routinen, die nicht so oft benötigt werden.

Beim Eröffnen einer LIBRARY von BASIC aus muß auf der Diskette, die sich gerade im Laufwerk befindet, eine Datei mit dem LIBRARY-Namen und der Endung .bmap statt .library vorhanden sein. Trägt die Bibliothek beispielsweise den Namen graphics.library, so muß die zugehörige Datei den Namen graphics.bmap tragen. Diese .bmap-Dateien erfüllen zwei Funktionen.

Die erste Funktion dieser Dateien ist die Zuordnung von Namen zu den Betriebssystemroutinen. Jede .bmap-Datei enthält Zuordnungen von Routinennamen zu OFFsets auf die schon erwähnte Sprungtabelle. Diese Namen sind üblicherweise standardisiert.

Die meisten Betriebssystemroutinen benötigen Parameter in bestimmten Prozessorregistern (siehe Kapitel 4). In den .bmap-Dateien ist daher verzeichnet, welche Parameter beim Aufruf einer Routine in welchen Registern übergeben werden.

Auf Ihrer Amiga-Extras-Diskette sollte sich ein Ordner mit den sogenannten .fd-Dateien und ein Programm ConvertFD befinden. Für jede LIBRARY existiert eine .fd-Datei, und ConvertFD kann aus diesen Dateien die entsprechenden .bmap-Dateien generieren. Auf diese Weise erhalten die Betriebssystemroutinen die von Commodore vergebenen Standardnamen. Eine Ausnahme sollte man allerdings kennen: Alle Routinen, deren Namen mit BASIC-Befehlen kollidieren, erhalten ein x vor ihrem normalen Namen. Drei der .bmap-Dateien brauchen Sie nicht mehr zu erzeugen, da sie schon im Ordner basicdemos stehen. Es handelt sich dabei um die Dateien exec.bmap, dos.bmap und graphics.bmap.

Folgendes Beispiel soll demonstrieren, wie der Zugriff auf die Bibliotheken erfolgt:

```
CHDIR "df0:"
LIBRARY "basicdemos/exec.library"

DECLARE FUNCTION AllocMem&(bytesize,requirements) LIBRARY
REM "FreeMem liefert keinen Wert zurück"

PRINT"freier Systemspeicher vorher  :";FRE(-1)
Speicherblock&=AllocMem&(1000,0)
PRINT"nach dem Belegen des Speichers:";FRE(-1)
CALL FreeMem(Speicherblock&,1000)
PRINT"freier Systemspeicher nachher :";FRE(-1)

LIBRARY CLOSE
```

Dieses Programm geht davon aus, daß Sie Ihre Amiga-Extras-Diskette ins Laufwerk eingelegt haben und daß sich im Ordner basicdemos die Datei exec.bmap befindet.

Der Befehl LIBRARY eröffnet, wie schon gesagt, eine Systembibliothek. In diesem Programmbeispiel wird die sogenannte EXEC-LIBRARY eröffnet. Sie enthält die wichtigsten Routinen zur Verwaltung des Speichers und des Multitasking.

Das Beispielprogramm benutzt zwei Routinen aus der exec-LIBRARY, nämlich AllocMem und FreeMem. AllocMem reserviert eine gewünschte Anzahl von Bytes im Speicher, während FreeMem diesen Speicher wieder freigibt. Eigentlich könnte man diese Routinen direkt benutzen, nachdem der LIBRARY-Befehl ausgeführt wurde. AllocMem liefert aber einen Wert zurück, nämlich die Adresse, an der der gewünschte Speicherblock beginnt. In diesem Falle muß erst der Datentyp des Wertes festgelegt werden. Dazu dient die Anweisung DECLARE FUNCTION.

DECLARE FUNCTION NAME(...) LIBRARY

Funktioniert ähnlich wie DEF FN. NAME ist der Routinenname und sollte als Endung eine Typspezifikation enthalten, genau so wie sie bei Variablen oder DEF-FN-Funktionen verwendet wird. Der Klammerausdruck hinter NAME wird zwar normaler-

weise ignoriert. Man sollte aber der Übersichtlichkeit halber die Namen der an diese Routine übergebenen Parameter hier hineinschreiben. Der Zusatz LIBRARY muß unbedingt mitangegeben werden. Er macht deutlich, daß im Gegensatz zu DEF FN die definierte Routine nicht im Programm selbst zu finden ist, sondern in einer Systembibliothek.

Im Beispielprogramm wird die Routine AllocMem als Funktion vom Typ lange Ganzzahl definiert. FreeMem wird nicht noch einmal gesondert deklariert, da diese Routine keinen Wert zurückliefert.

Der Aufruf der Routinen geschieht auf unterschiedliche Weise, je nachdem, ob die Routine als Funktion deklariert wurde (DECLARE FUNCTION) oder nicht. AllocMem beispielsweise kann einfach so aufgerufen werden, als handele es sich dabei um eine normale BASIC-Funktion. FreeMem dagegen wird normalerweise mit dem Schlüsselwort CALL aufgerufen. In vielen Fällen kann man CALL auch weglassen, was aber manchmal zu Zweideutigkeiten führt. Besser ist es, wenn Sie dieses Wort immer hinzuschreiben.

Was machen die beiden Beispielroutinen nun genau?

S&=AllocMem&(bytesize,requirements)

Sucht im freien Systemspeicher nach einem Speicherblock der Länge bytesize. requirements gibt an, welche Art von Speicher gewünscht wird und kann von uns in diesem Beispiel bedenkenlos auf Null gesetzt werden. Der Wert, den die Variable s& erhält, ist die Anfangsadresse des reservierten Speichers. Ab dieser Adresse haben Sie dann bytesize Bytes zur freien Verfügung. Diese können Sie von BASIC aus nur über die PEEK- und POKE-Befehle ansprechen.

CALL FreeMem(memoryblock,bytesize)

Gibt den mit AllocMem belegten Speicherplatz wieder frei. Memoryblock ist die Startadresse des freizugebenden Bereiches, bytesize die Länge. Geben Sie hier grundsätzlich die Werte an,

die Sie bei der Funktion AllocMem benutzt bzw. zurückerhalten haben, sonst kann es zu einem Systemabsturz kommen.

Im Beispielprogramm oben wird ein Speicherblock von 1000 Bytes belegt und wieder freigegeben. Kontrolliert wird das Ganze mit der Funktion FRE(-1), die den freien Systemspeicher anzeigt.

Der Befehl ganz am Schluß, LIBRARY CLOSE, schließt alle geöffneten Libraries wieder. Beachten Sie bitte auch, daß Sie nur maximal fünf Libraries gleichzeitig geöffnet haben können.

3.5.2 Wichtige DOS-Systemroutinen für BASIC-Programme

In diesem Kapitel stellen wir Ihnen einige Beispiele für DOS-Systemroutinen vor, mit denen sich gerade von BASIC aus wirklich etwas Sinnvolles anfangen läßt. Natürlich konnten wir nur eine sehr kleine Auswahl an Routinen treffen, da zu mehr in diesem Buch kein Platz war. Mehr ist aber auch nicht unbedingt sinnvoll, da die meisten anderen Routinen nur für Maschinenprogrammierer interessant sind oder Ihnen sowieso schon durch anwenderfreundlichere BASIC-Befehle zur Verfügung stehen.

Wir haben uns in diesem Kapitel auf zwei Probleme konzentriert, die im BASIC des C64 nicht existieren, jedoch im AmigaBASIC mit normalen BASIC-Befehlen nicht zu lösen sind. Da wäre zunächst einmal der Filezugriff auf beliebige Files, also auch auf solche, die nicht von BASIC aus erzeugt wurden. Dieser Zugriff ist zwar theoretisch mit dem normalen OPEN-Befehl möglich, allerdings wird hier immer noch ein .info-File erzeugt, was im Endeffekt als sehr störend wirken kann. Weiterhin hat man beim C64 die Möglichkeit, alle DOS-Befehle von BASIC aus zu erreichen. Beim Amiga sind nur die wichtigsten direkt zu erreichen. Wir werden hier zeigen, wie man die CLI-Befehle von BASIC aus benutzt.

Eines noch vorweg: Für einige der im weiteren erklärten Routinen wird die SADD-Funktion benötigt. SADD(a$) liefert die

Startadresse der in der Variablen a$ abgelegten Zeichenkette. Wie bei Adressen üblich, ist dieser Wert eine lange Ganzzahl.

Folgende Routinen werden im weiteren besprochen:

```
xOpen     : beliebiges Daten-File eröffnen
xClose    : Daten-File schließen
xRead     : Daten aus einem File lesen
xWrite    : Daten in ein File schreiben
Seek      : bestimmte Stelle im Daten-File anwählen
Execute   : CLI-Befehl aufrufen
```

Für alle diese Routinen muß mit dem LIBRARY-Befehl die Bibliothek dos.library geöffnet werden. Außerdem sollten die Routinen xOpen, xRead, xWRITE, und Seek mit DECLARE FUNCTION als Funktionen vom Typ lange Ganzzahl, also mit der Endung &, deklariert werden. Execute kann dagegen als Funktion beliebigen Typs, am besten als einfach genaue Zahl ohne Endung, angegeben werden. Nun aber zu der angekündigten Routinen-Beschreibung:

xOpen

```
filehandle&=xOpen&(name,acessMode)
```

Eigentlich heißt der Befehl OPEN, aber dieser Name ist schon durch einen BASIC-Befehl belegt.

Funktion

Mit xOpen läßt sich ein beliebiges Daten-File eröffnen. Der zurückgelieferte Wert filehandle wird benötigt, um bei den Befehlen zum Zugriff auf das File (xRead, xWrite, Seek, CLOSE) anzugeben, welches File gemeint ist. Ist filehandle& null, ist ein Fehler beim xOpen-Aufruf aufgetreten. NAME ist die Adresse eines Strings, der den Namen des zu öffnenden Files enthält. Der String muß als letztes Zeichen ein Null-Byte (CHR$(0))

enthalten. Als Name sind alle Namen erlaubt, die unter dem normalen DOS erlaubt sind, also zum Beispiel auch CON:..., jedoch nicht die nur unter BASIC verfügbaren wie SCRN:.... acessmode ist einer der beiden Werte 1005 (ModeOldfile) oder 1006 (ModeNewfile). ModeOldfile öffnet ein bestehendes File für Lese- und Schreibzugriffe, während ModeNewfile ein neues File zum Lesen und Schreiben öffnet. Sollte ein File des angegebenen Namens schon existieren, so wird es bei ModeNewfile gelöscht.

Der Vorteil von xOpen gegenüber OPEN ist die größere Flexibilität, die dieser Befehl ermöglicht. Es wird beispielsweise kein .info-File erzeugt, wenn man eine Datei anlegt. Man hat mit der Seek-Routine relativen Filezugriff, ohne auf eine bestimmte Datensatzlänge festgelegt zu sein. Grundsätzlich sollte man xOpen immer dann verwenden, wenn man mit Dateien arbeitet, die nicht mit den normalen BASIC-Befehlen erzeugt wurden oder die auch von anderen (Nicht-BASIC-) Programmen benutzt werden sollen.

xClose

```
xClose(filehandle&)
```

Eigentlich heißt der Befehl CLOSE, aber dieser Name ist schon durch einen BASIC-Befehl belegt.

Funktion

Schließt ein mit xOpen öffnetes File.

xRead

```
readlength&=xRead&(filehandle&,buffer&,length&)
```

Eigentlich heißt der Befehl Read, aber dieser Name ist schon durch einen BASIC-Befehl belegt.

Funktion

xRead liest aus der zu filehandle& gehörenden Datei length& Bytes und schreibt sie an der Adresse buffer& in den Speicher. Für buffer& kann beispielsweise eine Stringadresse angegeben werden. Der String muß dann aber schon eine Mindestlänge von length& Zeichen haben. readlength& ist die Anzahl der tatsächlich gelesenen Zeichen. Für den Fall, daß das Dateiende erreicht wurde, stimmt readlength& nicht mit length& überein. Sollte beim Einlesen ein Fehler aufgetreten sein, hat readlength& den Wert -1.

xWrite

```
writelength&=xWrite&(filehandle&,buffer&,length&)
```

Eigentlich heißt der Befehl WRITE, aber dieser Name ist schon durch einen BASIC-Befehl belegt.

Funktion

xWrite ist das Gegenstück zu xRead und schreibt length& Bytes ab der Adresse buffer& in das Daten-File, das zu filehandle& gehört. writelength& stimmt normalerweise immer mit length& überein, außer, wenn ein Fehler aufgetreten ist. Dann nämlich hat writelength& den Wert -1.

Seek

```
oldposition&=Seek(filehandle&,position&,offset&)
```

Funktion

Seek positioniert innerhalb eines Files auf ein bestimmtes Byte. position& gibt dabei an, auf welches Byte der Schreib-/Lesezeiger gesetzt werden soll, und zwar relativ zum Filebeginn, File-Ende oder zur aktuellen Position. Welcher dieser drei Bezugspunkte genommen wird, entscheidet offset&. Hat offset& den Wert -1, so bezieht sich position& auf den Filebeginn. Beim Wert 0 darf position& nur negative Werte annehmen, da hier Bezug zum File-Ende genommen wird. Der Wert 1 schließlich läßt position& sich auf die aktuelle Position beziehen. oldposition& ist die Position im File vor dem Seek-Aufruf. Hat oldposition& den Wert -1, so ist ein Fehler aufgetreten. Dies kann zum Beispiel der Fall sein, wenn man versucht, über das File-Ende hinaus zu positionieren.

Execute

```
success=Execute(string&,infile&,outfile&)
```

Funktion

Execute ruft einen CLI-Befehl auf. string& ist die Adresse einer Zeichenkette, die genauso aufgebaut ist wie ein Befehlsaufruf im CLI. Sie muß mit einem chr$(0) beendet sein. infile& und outfile& sind Filehandle-Werte. infile& bezeichnet ein File, aus dem weitere Eingabewerte geholt werden, hat aber normalerweise den Wert Null, da fast alle CLI-Befehle ihre Parameter schon aus dem Befehlsstring erhalten. outfile& ist dementsprechend das Filehandle eines mit xOpen erzeugten Ausgabefiles. Leider kann man die Ausgabe nicht auf das normale

BASIC-Ausgabefenster umleiten. Man kann sich aber mit xOpen ein eigenes Ausgabefenster schaffen und das zugehörige Filehandle an Execute übergeben. Hat success den Wert Null, so ist ein Fehler beim Befehlsaufruf aufgetreten, anderenfalls hat alles geklappt.

Folgendes Beispielprogramm demonstriert, wie man den CLI-Befehl dir von BASIC aus aufruft. Es wird davon ausgegangen, daß sich die Datei dos.bmap auf der Amiga-Extras-Diskette im Ordner basicdemos befindet.

```
CHDIR "df0:"
LIBRARY "basicdemos/dos.library"

ModeOldfile&=1005&
ModeNewfile&=1006&
DECLARE FUNCTION xOpen&(xName,acessMode) LIBRARY
DECLARE FUNCTION Execute(string,file,file) LIBRARY

a$="con:10/10/500/200/Directory"+CHR$(0):REM "Fensterwerte"
filehandle&=xOpen&(SADD(a$),ModeNewfile&)

a$="sys:c/dir df0:"+CHR$(0):REM "Befehl 'dir' von Workbench"
success=Execute(SADD(a$),0,filehandle&)

WHILE INKEY$="":WEND
xClose(filehandle&)

LIBRARY CLOSE
```

Zu Anfang wird wie üblich die DOS-LIBRARY eröffnet. Ihre Amiga-Extras-Diskette muß sich zu Programmbeginn in Laufwerk df0 befinden. Dann werden die Betriebssytemroutinen xOpen und Execute als BASIC-Funktionen deklariert. xOpen eröffnet danach ein Ausgabefenster über das Gerät con:. Wenn jetzt Execute den dir-Befehl aufruft, werden Sie aufgefordert, die Workbench-Diskette einzulegen. Nachdem dieses geschehen ist, holt sich das Programm aus dem Ordner c den entsprechenden Befehl und führt ihn aus. Um das Programm abzubrechen, müssen Sie in das BASIC-Ausgabefenster hineinklicken und eine Taste betätigen.

3.6 Einbinden von Maschinenprogrammen in BASIC-Programme

Um beim C64 Maschinenunterprogramme in BASIC-Programmen zu verwenden, mußte man verschiedene Tricks kennen, die man sich erst durch Fachlektüre anlesen oder durch eigenes Herumprobieren herausfinden mußte. Dazu gehörte beispielsweise, die Speicherstelle, an der man seine Programme ablegen konnte, wie man sie nach dem Programmstart wieder in den Speicher bekommt und wie man Werte zwischen dem BASIC-Programm und dem Maschinenprogramm austauscht. Alle diese Probleme tauchen genauso beim Amiga auf. Wir erklären Ihnen in diesem Kapitel, wie Sie sie auf elegante Weise lösen.

Wir gehen davon aus, daß Sie schon 68000-Assembler-Programme schreiben können und wissen, wie Sie Ihren Assembler oder Monitor bedienen. Wenn das nicht der Fall sein sollte, dann werfen Sie doch einmal einen Blick in Kapitel 4, das eine kurze Einführung in 68000-Assembler enthält.

Beginnen wir mit der Frage: Wo legt man seine Maschinenprogramme ab? Wir haben schon öfter gesehen, daß der Amiga ein Multitasking-Computer ist, bei dem alle Programme in beliebigen Speicherbereichen ablaufen können. Dies wird normalerweise durch eine Betriebssystemroutine bewerkstelligt, die beim Laden die Programme an den entsprechenden Bereich anpaßt. Dazu müssen die Programme in einem ganz bestimmten Format, als sogenannte LOAD-Module, auf der Diskette vorliegen. Es gibt aber auch die Möglichkeit, seine Programme so zu schreiben, daß sie nicht verändert werden brauchen, um trotzdem an jeder Stelle des Speichers ablauffähig zu sein (sogenannter PC-relativer Code). Dieser Code läßt sich leicht in BASIC-Programmen innerhalb von DATA-Zeilen unterbringen, hat aber den Nachteil, daß er nicht direkt von diversen Assemblern erzeugt wird. Diese erzeugen nämlich normalerweise LOAD-Module, die direkt vom CLI aus gestartet werden können.

Wie bekommt man ein als LOAD-Modul abgelegtes Programm von BASIC aus in den Speicher? Zu diesem Zweck existieren

zwei Routinen in der DOS-LIBRARY, die uns die Verwaltung der Module abnehmen. Hier die Beschreibung:

LoadSeg

```
segment&=LoadSeg&(filename&)
```

Funktion

LoadSeg lädt ein Programm in den System-Speicher und paßt es an den entsprechenden Speicherbereich an. Dabei wird der benötigte Speicher automatisch reserviert. filename& ist die Adresse eines den File-Namen enthaltenden Strings, der wie üblich als letztes Zeichen ein CHR$(0) enthalten muß. segment&*4+4 liefert die Startadresse des Maschinenprogramms.

UnLoadSeg

```
UnLoadSeg(segment&)
```

Funktion

UnLoadSeg gibt den durch LoadSeg belegten Speicher wieder frei. Diese Routine sollte grundsätzlich immer am Ende eines BASIC-Programms aufgerufen werden, wenn vorher LoadSeg benutzt wurde. segment& ist der von LoadSeg zurückgelieferte Wert. Andere Werte können zum Systemabsturz führen.

Ein Beispiel: Die Assembler-Routine mprogramm soll durch ein BASIC-Programm geladen und gestartet werden. Folgendes Programm übernimmt diese Aufgabe:

```
LIBRARY "dos.library"
DECLARE FUNCTION LoadSeg&(filename) LIBRARY
REM "UnLoadSeg liefert keinen Wert zurück"

a$="mprogramm"+CHR$(0)
segment&=LoadSeg&(SADD(a$))

programmstart&=segment&*4+4
Call programmstart& :REM "Programm aufrufen"

UnLoadSeg(segment&)
LIBRARY CLOSE
```

Das Programm geht davon aus, daß die Dateien dos.bmap und mprogramm im aktuellen Directory zur Verfügung stehen. Worauf noch nicht hingewiesen wurde, ist, daß sich mit dem Befehl CALL auch Maschinenprogramme starten lassen. Wenn ein solches Programm mit der Instruktion RTS abschließt, wird das BASIC-Programm ganz normal weiter ausgeführt. Es ist fast genauso wie beim Befehl SYS im C64-BASIC, nur darf beim CALL-Befehl keine Formel, sondern nur ein Variablenname angegeben werden. Wie man mit dem CALL-Befehl Werte zwischen dem BASIC- und dem Maschinenprogramm austauscht, werden wir noch später besprechen.

Jetzt aber zu der Methode, Maschinenprogramme, die sowieso schon frei verschiebbar sind, in BASIC-Programme einzugliedern. Will man beispielsweise Programme in DATA-Zeilen unterbringen, sind frei verschiebbare Programme wesentlich einfacher zu handhaben als andere. Beim C64 ging man dabei so vor, daß man die Daten einfach mit POKE-Befehlen in einen freien Teil des Speichers brachte. Wie wir schon in Kapitel 3.5 gezeigt haben, ist es beim Amiga kein Problem, sich einen freien Speicher vom Betriebssystem zuteilen zu lassen. Theoretisch wäre es somit kein Problem, Programme genau wie beim C64 dort hineinzupoken. Einfacher ist es jedoch, ein komplettes Maschinenprogramm in einen String zu packen, der ja beim Amiga bis zu 32767 Zeichen enthalten darf. Die Startadresse des Maschinenprogramms ist dann die Anfangsadresse des Strings und läßt sich über die Funktion SADD ermitteln.

Zuerst müssen Sie allerdings eine sogenannte Binärdatei Ihres Programms erstellen. Dabei handelt es sich einfach um die hintereinander abgelegten Bytes, aus denen Ihr Programm besteht, ohne irgendwelche Informationen über die Startadresse. Solche Dateien lassen sich natürlich nicht mehr wie LOAD-Module vom CLI aus aufrufen. Das Erstellen können Sie nur dann vornehmen, wenn Sie einen Maschinensprache-Monitor besitzen, mit dem Sie LOAD-Module laden und beliebige Speicherbereiche abspeichern können. Vielleicht bietet aber auch Ihr Assembler die Möglichkeit, Programme als Binärdateien abzulegen. Wichtig ist nur, daß Sie das fertig assemblierte Programm direkt aus dem Speicher auf die Diskette übertragen. Wie Sie das machen, ist von Monitor zu Monitor verschieden. Entnehmen Sie die dazu notwendigen Schritte bitte Ihrem Monitor- bzw. Assembler-Handbuch.

Wie und wohin lädt man nun von BASIC aus eine Binärdatei in den Speicher? Es gibt dazu mehrere Alternativen. Sie können einerseits die Datei jedesmal bei Programmstart mit OPEN und INPUT$ einlesen. Angenommen, Ihr Programm heißt mprogramm und hat eine Länge von 200 Bytes. Das Einlesen und Starten des Programms geschieht dann so:

```
OPEN "i",#1,"mprogramm":REM "Programm-File öffnen"
prog$=INPUT$(200,#1)   :REM "Daten einlesen"
CLOSE #1               :REM "Programm-File schließen"
prostart&=SADD(prog$)  :REM "Startadresse ermitteln"
CALL prostart&         :REM "Programm aufrufen"
```

Eine andere Möglichkeit besteht darin, wie beim C64 das Programm in DATA-Zeilen zu übertragen. Die dazu benötigte DATA-Wandler-Routine können wir problemlos in BASIC schreiben:

```
REM Datawandler

 INPUT"Name der Binär-Datei:",nl$
 INPUT"Name der DATA-Datei :",ns$

 OPEN"i",#1,nl$ :REM Lesedatei öffnen
 OPEN"o",#2,ns$ :REM Schreibdatei öffnen
```

```
WHILE NOT EOF(1)
  PRINT#2,"DATA ";
  FOR i%=1 TO 12
    PRINT #2,USING "###";ASC(INPUT$(1,#1));
    IF EOF(1) OR i%=12 THEN abbruch
    PRINT#2,",";
  NEXT i%
abbruch:
  PRINT #2,""
WEND

CLOSE #2
CLOSE #1
```

Diese Routine liest eine Datei ein und erzeugt daraus eine ASCII-Textdatei, die nur aus DATA-Zeilen besteht. Diese kann dann mit MERGE an ein bestehendes Programm angefügt werden. Die Routine, die die Daten dann in einen String einliest, muß man allerdings noch selbst schreiben.

Betrachten wir als Beispiel folgendes kurzes Maschinenprogramm:

```
        move.l  #$00030000,D0    ;Zähler laden
loop:   move.b  D0,$dff180       ;Lowbyte in Farbregister
        subi.l  #$00000001,D0    ;Zähler verringern
        bne     loop             ;verzweige, wenn nicht null
        rts                      ;Rücksprung
```

Das Programm macht nichts anderes, als für einen kurzen Moment die Hintergrundfarbe sehr schnell zu verändern.

Der DATA-Wandler erzeugt aus der Binärdatei dieses Programms folgendes:

```
DATA  32, 60,  0,  3,  0,  0, 19,192,  0,223,241,128
DATA   4,128,  0,  0,  0,  1,102,  0,255,242,  78,117
```

Das vollständige Programm, das diese Daten einliest und das Maschinenprogramm startet, sieht so aus:

```
p$=""
READ a
WHILE a<>-1
  p$=p$+CHR$(a):READ a
WEND
programm&=SADD(p$)
CALL programm&

DATA  32, 60,  0,  3,  0,  0, 19,192,  0,223,241,128
DATA   4,128,  0,  0,  0,  1,102,  0,255,242, 78,117
DATA -1
```

Der Befehl CALL kann genau wie beim Aufruf von LIBRARY-Routinen dazu genutzt werden, Werte an das jeweilige Maschinenprogramm zu übergeben. Erlaubt sind dabei kurze und lange Ganzzahlen. Die Werte werden auf dem Stapel abgelegt. Folgendes Beispiel macht klar, wie das geschieht:

CALL programm&(Wert1&,Wert2&,Wert3&)

Erzeugt folgendes Bild auf dem Stapel:

```
OFFSET  Wert
 0(SP)   Rücksprungadresse
 4(SP)   Wert1&
 8(SP)   Wert2&
12(SP)   Wert3&
```

Will man Werte vom Maschinenprogramm zurück an das BASIC-Programm übergeben, so muß man zuvor dem Maschinenprogramm die Adressen von Variablen mitteilen. Das Maschinenprogramm kann dann die Variableninhalte verändern. Dazu gibt es zwei BASIC-Funktionen: SADD(Stringvariable) haben wir schon kennengelernt. Sie liefert die Adresse des in einer Zeichenkettenvariablen abgelegten Strings. VARPTR (Variable) ist die zweite Funktion. Sie liefert die Adresse einer numerischen Variablen. VARPTR kann zwar auch auf Stringvariablen angewendet werden, liefert dann aber die Adresse, an der sich die Stringlänge und ein Zeiger auf den String befinden. VARPTR darf nur auf schon existente Variablen angewendet werden.

Durch die Übergabe von Adressen von String- oder Fließkommavariablen wird es auch möglich, von der Maschinenebene auf solche Werte zugreifen zu können, obwohl über den CALL-Befehl normalerweise nur Ganzzahlen übergeben werden können.

Da Datenfelder grundsätzlich zusammenhängend im Speicher stehen, kann man mit VARPTR auch die Anfangsadresse eines Datenfeldes dadurch ermitteln, daß man sich die Anfangsadresse des ersten Feldelementes holt. Vorsicht! Sobald neue Variablen im Programm benutzt und definiert werden, ändern sich die Startadressen aller Felder.

Folgendes Beispiel demonstriert die Datenübergabe: Eine Maschinenroutine soll zählen, wie viele Variablen in einem Ganzzahlfeld den Wert Null haben. Beim CALL-Befehl müssen dazu drei Werte übergeben werden: die Adresse des Datenfeldes, die Feldlänge und die Adresse einer Variablen zur Rückgabe des ermittelten Wertes. Hier zuerst das BASIC-Programm:

```
REM "Parameterübergabe an Maschinenprogramm"

REM "Maschinenprogramm einlesen"
pro$=""
READ a
WHILE a<>-1
  pro$=pro$+CHR$(a):READ a
WEND

REM "Feld definieren"
DIM feld%(100)

REM "Feld zufällig initialisieren"
RANDOMIZE TIMER
FOR i%=0 TO 100
  IF RND>.5 THEN
              feld%(i%)=1
            ELSE
              feld%(i%)=0
            END IF
NEXT i%

anzahl%=0:REM Variable schaffen
```

```
REM Programm aufrufen
Zaehlen&=SADD(pro$)
CALL Zaehlen&(VARPTR(feld%(0)),100&,VARPTR(anzahl%))

PRINT "Anzahl:";anzahl%

DATA  32,111,  0,  4, 32, 47,  0,  8, 66, 65, 74, 88
DATA 102,  2, 82, 65, 81,200,255,248, 32,111,  0, 12
DATA  48,129, 78,117,-1
```

Das in diesem BASIC-Programm enthaltene Maschinenprogramm sieht so aus:

```
          move.l  4(A7),A0   ;Feldadresse holen
          move.l  8(A7),D0   ;Feldgröße holen
          clr.w   D1         ;Null-Zähler auf null
loop:     tst.w   (A0)+      ;Feldelement auf Wert testen
          bne.s   weiter     ;Verzweige, wenn ungleich null
          addq.w  #1,D1      ;Null-Zähler erhöhen
weiter:   dbf     D0,loop    ;Verzweige, wenn nicht fertig
          move.l  12(A7),A0  ;Variablenadresse für Rückgabe
          move.w  D1,(A0)    ;Wert in Variable schreiben
          rts                ;Rücksprung
```

Das BASIC-Programm schreibt zuerst ein Datenfeld zufällig mit Nullen und Einsen voll. Dann wird die Variable anzahl% durch eine Zuweisung geschaffen, da die VARPTR-Anweisung sonst einen Fehler melden würde. Beim Aufruf werden die drei benötigten Werte übergeben, so daß der Stapel danach so aussieht:

```
 4(SP)  Adresse des Datenfeldes
 8(SP)  Feldgröße
12(SP)  Adresse der Variablen anzahl%
```

Das Maschinenprogramm ermittelt in einer Schleife die Anzahl der Nullen im Datenfeld und schreibt diese in die Variable anzahl%.

Wie wir gesehen haben, ist es beim Amiga wesentlich einfacher als beim C64, Maschinenprogramme in BASIC-Programme einzubinden. Insbesondere die Parameterübergabe war beim C64 wesentlich schwerer zu realisieren.

4. Programmierung in Maschinensprache auf dem Amiga

Dieses Kapitel soll Ihnen zeigen, wie Sie mit Hilfe der Maschinensprache die Leistungsfähigkeiten des Amiga voll ausnutzen können.

Im ersten Teil dieses Kapitels wird die Programmierung des Amiga-Prozessors (MC68000) beschrieben, bezugnehmend auf den 6510 Prozessor, der im C64 Verwendung findet. Der zweite Teil befaßt sich anschließend mit der weiteren Amiga-Hardware. Es werden die Aufgaben der Custom-Chips genannt, und es wird gezeigt, wie Sie diese mit eigenen Programmen direkt ansteuern können.

Nach diesem Kapitel werden Sie in der Lage sein, Ihre eigenen kleinen Maschinenprogramme zu schreiben. Diese können dann zum Beispiel in ein BASIC- oder C-Programm eingebunden werden.

Zum Verständnis der nachfolgenden Ausführungen müssen Kenntnisse der 6510-Progammierung vorausgesetzt werden, da eine grundlegende Einführung den Rahmen dieses Buches sprengen würde.

Hinweis: (zur Programmeingabe) Alle Programme dieses Kapitels wurden mit dem Kuma-SEKA-Assembler geschrieben und danach getestet. Da uns der wesentlich leistungsfähigere Assembler PROFIMAT Amiga von DATA BECKER leider noch nicht zur Verfügung stand, nennen wir Ihnen hier die nötigen Änderungen, damit Sie auch den PROFIMAT verwenden können.

- Gleichheitszeichen müssen durch EQU ersetzt werden
- Labels müssen am Zeilenanfang beginnen

- Befehle dürfen nicht am Anfang beginnen
- Am Ende des Programmtextes muß eine END-Anweisung stehen

4.1 Die Programmierung des MC68000

4.1.1 Die verwendbaren Datenformate

Da der MC68000 ein 32/16-Bit-Prozessor ist, weist er zusätzlich zu den Datenformaten des 6510 (Bit, BCD, Byte) das Wort (16 Bit) und das Langwort (32 Bit) auf. Als Standarddatenformat wird das Wort verwendet. Somit beherrscht dieser Prozessor folgende Datenformate:

Datenformat	Datenlänge	Verwendung
Bit	1 Bit	Bitmanipulationen
BCD	4 Bit	BCD-Arithmetik
Byte	8 Bit	unterschiedlich
Wort	16 Bit	unterschiedlich
Langwort	32 Bit	unterschiedlich

Diese Datenformate werden wie folgt im Speicher abgelegt:

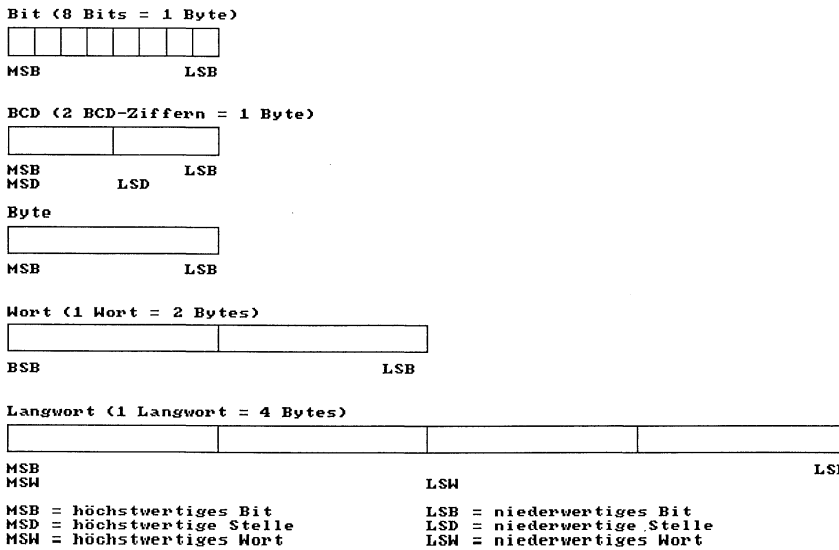

Abbildung 4.1: Die Datenformate des MC68000

Sie werden sicherlich gemerkt haben, daß der MC68000 16-Bit-Daten anders im Speicher ablegt als der 6510. Er legt im Gegensatz zum 6510 zuerst das HI-Byte und dann das LO-Byte ab. Dadurch kann der Zugriff auf die Daten beschleunigt werden.

4.1.2 Die internen Register - Aufbau und Funktionen

Der MC68000 besitzt im Vergleich zum 6510 eine Vielzahl universell verwendbarer Register. Diese Eigenschaft macht auch Programmiersprachen wie C für die Software-Entwicklung interessant, da die Laufzeit durch geschickte Ausnutzung aller Register wesentlich verkürzt werden kann.

Folgende Register stehen zur Verfügung:

Register	Datenlänge
Datenregister 0-7 (D0-D7)	32 Bit
Adreßregister 0-7 (A0-A7)	32 Bit
Programmzähler (PC)	32 Bit (24 Bit benutzt)
Statusregister (SR)	16 Bit

Die Datenregister sind vergleichbar mit dem Akkumulator des 6510. In ihnen können 1-, 4-, 8-, 16- und 32-Bit-Daten verarbeitet werden. Bei der Verwendung von 8- und 16-Bit-Daten werden nur die niederwertigsten Bits bearbeitet, die anderen bleiben unverändert. Befehle, die ihr Ergebnis in ein Datenregister schreiben, verändern die Flags des Statusregisters.

Mögliche Operationen mit Datenregistern:

- Datentransport
- Integer-Arithmetik
- BCD-Arithmetik
- Logische Verknüpfungen
- Verschiebung und Rotation
- Bitmanipulationen

In Verbindung mit den Adreßregistern ist ein Großteil der Adressierungsarten möglich (siehe Kapitel 4.1.5 Adressierungsarten). Es ist aber auch möglich, Daten in diesen Registern zu speichern. Eine 1- und 4-Bit Verarbeitung wie bei den Datenregistern ist nicht zulässig. Wird in ein Adreßregister das Ergebnis einer Operation gespeichert, bleiben die Flags unverändert.

Adreßregister A7 bildet eine Ausnahme, es dient dem Prozessor als Stackpointer. A7 zeigt immer auf das letzte Datum des Stacks, der mit fallenden Adressen wächst. Da der Stackpointer eine Länge von 32 Bit hat, kann der Stack an einer beliebigen, vom Programmierer gewählten Stelle des Speichers liegen.

Der Programmzähler umfaßt ebenfalls 32 Bit, da aber nur 24 Adreßleitungen verwendet werden, ergibt sich daraus ein direkt ansprechbarer Adreßraum von 16 MByte.

Nun ist noch das Statusregister zu beschreiben. Es ist unterteilt in ein Systembyte und ein Anwenderbyte. Unter dem Condition-code-Register versteht man das Anwenderbyte des Status-registers. Die Flags sind wie folgt zugeordnet:

Statusregister

Anwenderbyte (Conditioncode-Register)

Bit 0	C - Carry (Übertrag)
Bit 1	V - Overflow (Überlauf)
Bit 2	Z - Zero (Null)
Bit 3	N - Negative (Negativ)
Bit 4	X - Extend (Erweiterung)
Bit 5-7	unbenutzt

Systembyte

Bit 8-10	I0-I2 - Interrupt-Maske
Bit 11-12	unbenutzt
Bit 13	S - Supervisor-Bit
Bit 14	unbenutzt
Bit 15	T - Trace-Bit

Das Carry-Flag C wird auf Eins gesetzt, wenn bei der Addition ein Überlauf stattfindet, weil dann das Ergebnis nicht mehr in das entsprechende Datenformat paßt. Mit Hilfe des Carry-Flags können Zahlen größer als 32 Bit addiert werden. Bei der Subtraktion wird das C-Flags gesetzt, falls die abzuziehende Zahl größer ist als die, von der abgezogen wird. Schließlich wird das herausgeschobene Bit bei Schiebe- oder Rotationsbefehlen im Carry-Flag zwischengespeichert.

Das Überlauf-Flag V wird bei der Überschreitung des Zahlenbereichs gesetzt. Dies ist bespielsweise bei der Addition zweier positiver Zahlen mit negativen Ergebnis (Zweierkomplement) der Fall.

Das Zeroflag Z wird gesetzt, wenn das Ergebnis einer Operation gleich null ist. Dies gilt auch für Vergleichsbefehle.

Das Negativ-Flag N repräsentiert das höchstwertigste Bit im Ergebnis. Somit ist es bei einem negativen Ergebnis (Zweierkomplement) gesetzt.

Die bisher besprochenen Flags waren bereits vom C64 her bekannt. Das Break- und das Dezimal-Flag existieren beim MC68000 nicht mehr, da sie überflüssig geworden sind. Die Amiga-Hardware ist in der Lage, einen Software-Interrupt zu erkennen, und durch eine Trennung zwischen arithmetischen und BCD-Befehlen entfällt das Dezimal-Flag. Dafür sind aber einige neue Flags hinzugekommem.

Da wäre zunächst das Extendflag X, das sich meistens so verhält wie das C-Flag. Es gibt aber einige Befehle, die das C-Flag beeinflussen, das X-Flag jedoch unverändert lassen. Dazu zählen zum Beispiel Vergleichsbefehle. Auf diese Weise wird es möglich, einen Übertrag über mehrere Befehle hinweg im X-Flag zu halten und erst dann auf ihn zuzugreifen.

Im Gegensatz zum 6510 besitzt der MC68000 nicht ein Interrupt-Flag, sondern gleich drei. Mit Hilfe dieser drei Bits ist er in der Lage, von sieben Interrupt-Ebenen bis zu sechs auszumaskieren. Die höchste, nicht auszumaskierende Interrupt-Ebene ist vergleichbar mit dem NMI (non-maskable interrupt) des 6510. Für jeden dieser Interrupts steht ein freidefinierbarer Vektor zur Verfügung.

Eine weitere Besonderheit des MC68000 ist die Unterscheidung zweier Betriebsarten, die über das Supervisorflag S umgeschaltet werden können. Ist das Flag gesetzt, befindet sich der Prozessor im sogenannten Supervisor-Modus, bei gelöschtem Flag im User-Modus. Durch diese strikte Trennung zwischen System- und Anwendersoftware wird die Verwendung eines Multitasking-Betriebssystems, wie das des Amiga, ermöglicht. Die Tatsache, daß einige systemkontrollierende und -steuernde Befehle nur im Supervisor-Modus ausgeführt werden können, dient als Schutzmechanismus vor Systemänderungen. Sollte dennoch versucht werden, einen dieser Befehle im User-Modus anzu-

sprechen,dann springt der Prozessor automatisch zum Vektor der Privilegverletzung. Das Resultat ist im Normalfall eine Guru-Meditation-Meldung, also ein Systemabsturz.

Als letztes ist noch das Traceflag T zu nennen. Ist dieses Flag gesetzt, verzweigt der MC68000 nach jedem Befehl zum Trace-Vektor. Dieser Vektor kann dann bespielsweise auf eine Routine gerichtet sein, die die Registerinhalte anzeigt.

Das Supervisorflag, das Traceflag und die Interruptflags liegen im Systembyte (Bit 8-15) des Statusregisters. Eine Veränderung dieser Flags ist nur im Supervisor-Modus möglich.

4.1.3 Die Befehlsnotation des MC68000

Ein Prozessor benötigt zur Ausführung eines Befehls bestimmte Informationen wie Befehlstyp, Adressierungsart, Quell- und Zieladresse. Diese Daten werden gewöhnlich im Opcode verschlüsselt. Da der MC68000 eine 16-Bit-Maschine ist, umfaßt der Opcode entsprechend 16 Bit. Die unterschiedlichen Datentypen (8-, 16- und 32-Bit) werden, falls erforderlich, zusätzlich im Opcode untergebracht. Im Gegensatz zum 6510 ist der MC68000 ein echter Zwei-Adreß-Prozessor, das heißt, es müssen bei einigen Befehlen sowohl Quell- als auch Zieladresse angegeben werden. So können Sie zum Beispiel mit dem MOVE-Befehl ein Datum direkt aus einer Speicherstelle in eine andere verschieben, ohne dabei ein Prozessorregister benutzen zu müssen.

In der Assemblerprogrammierung wird die Datenlänge der Operanden explizit durch Anhängen der Kürzel .b (Byte), .w (Wort) und .l (Langwort) an den Befehl angegeben. Bei den üblichen Assemblern kann bei Wortverarbeitung die Kennzeichnung weggelassen werden.

Die bei der Adressierung verwendete Notation ist der des 6510 ähnlich. Die indirekte Adressierung wird durch Klammern gekennzeichnet, und durch das Doppelkreuzsymbol ('#') wird die unmittelbare (immediate) Adressierung angezeigt.

`MOVE.b $1000,d0`	kopiert den Inhalt der Speicherstelle $1000 ins Datentegister D0.
`MOVE #$1000,d0`	kopiert die Zahl $1000 ins Datenregister D0.
`MOVE.l (a0),d0`	kopiert den Inhalt der Speicherstelle, auf die A0 zeigt, nach D0.

Neu hinzugekommen sind die Symbole A, I und Q. Sie werden einfach hinter den Befehl gesetzt. Das A zeigt an, daß der Zieloperand ein Adreßregister ist. Befehle, die nur die unmittelbare Adressierung (des Quelloperanden) erlauben, werden mit dem I gekennzeichnet. Das Q wird bei ADD, SUB und MOVE zur Kennzeichnung der impliziten Adressierung verwendet. So können beispielsweise kleine Zahlenwerte, die in ein Datenregister geladen werden sollen, direkt im Opcode untergebracht werden. Vorteile der impliziten Adressierung sind die kurze Ausführungszeit und die geringe Befehlslänge.

Um dem Programmierer etwas Arbeit abzunehmen, benötigen die meisten Assembler nicht die Zusätze A und I. Sie setzen in Abhängigkeit der Operanden den entsprechenden Opcode ein.

4.1.4 Die Adressierungsarten

Der MC68000 hat mit seinen 16 Adressierungsarten eine Vielzahl an Möglichkeiten, auf Daten zuzugreifen. Doch um Fehler bei der Adressierung zu vermeiden, ist es notwendig, bestimmte Regeln beim Zugriff auf den Speicher zu beachten:

- Wort- und Langwortzugriffe sind nur auf gerade Adressen möglich

- Opcodes dürfen nur auf geraden Adressen stehen, da es sich bei ihnen ebenfalls um Worte handelt

- Bytezugriffe sind auf gerade und ungerade Adressen möglich

Wenn Sie diese Regeln nicht beachten, verzweigt der Prozessor zur entsprechenden Fehlerbehandlung. Diese endet im Normalfall mit einer Guru-Meditation.

Die folgende Übersicht listet alle möglichen Adressierungsarten in den entsprechenden Kategorien auf.

Register Direkt (register direct adressing)
- Datenregister direkt
- Adreßregister direkt
- Statusregister direkt
- Conditioncode-Register direkt

Register Indirekt (register indirect adressing)
- Adreßregister indirekt
- Adreßregister indirekt mit Postinkrement
- Adreßregister indirekt mit Prädekrement
- Adreßregister indirekt mit Offset
- Adreßregister indirekt mit Offset und Index

Unmittelbar (immediate adressing)
- unmittelbar kurz (16 Bit)
- unmittelbar lang (32 Bit)

Implizit (implicit adressing)
- unmittelbar schnell

Absolut (absolute adressing)
- absolut kurz (16 Bit)
- absolut lang (32 Bit)

Programmzähler Relativ (PC relative adressing)
- relativ mit Offset
- relativ mit Offset und Index

Einige dieser Adressierungsarten werden Ihnen sicher vom C64 her bekannt sein, andere hingegen dürften völlig neu für Sie sein. Da das Verständnis sämtlicher Adressierungsarten die Voraussetzung der Programmierung ist, werden wir diese im einzelnen genau beschreiben und Anwendungshinweise geben.

Anmerkung: In den folgenden Beispielen werden meistens 2-Adreß-Befehle (MOVE-Befehl) verwendet, die eine Angabe von Quell- und Zieloperanden erfordern. Es kann daher vorkommen, daß eine noch nicht beschriebene Adressierungsart für einen der Operanden verwendet wird.

Register Direkt

Mit Hilfe dieser Adressierungsart können Sie den Inhalt eines Registers direkt verändern oder auslesen.

Im ersten Fall ist darauf zu achten, daß ein Schreibzugriff auf das Statusregister nur im Supervisor-Modus möglich ist. Falls die Flags des Anwenderbytes geändert werden müssen, greifen Sie einfach auf das Conditioncode-Register (CCR) zu. Auch beim Schreibzugriff auf ein Adreßregister gibt es eine Besonderheit: Ein übertragenes Wort (16 Bit) wird vor der Ausführung der Operation vorzeichenrichtig auf ein Langwort (32 Bit) erweitert.

Die folgenden Beispiele sollen die Unterschiede zwischen einem Schreibzugriff auf ein Daten- und ein Adreßregister verdeutlichen.

Schreibzugriffe auf Datenregister

```
MOVE.b #200,D0          Schreibt den Wert 200 in das un-
                        terste Byte von D0, der restliche
                        Inhalt bleibt unverändert.

MOVE   D0,D1 oder       Kopiert das unterste Wort aus D0
MOVE.w D0,D1            nach D1, ohne den restlichen
                        Inhalt von D1 zu verändern.

MOVE.l #$ffff0000,D0    Überträgt das Langwort $ffff0000
                        nach D0.
```

Schreibzugriffe auf Adreßregister

```
MOVE   #$0020,A0        Das Wort $0020 (pos) wird auf das
                        Langwort $00000020 erweitert und
                        dann nach A0 geschrieben
```

```
MOVE   #$8000,A1      Das Wort $8000 (neg) wird auf das
                      Langwort $ffff8000 erweitert und
                      dann nach A1 geschrieben

MOVE.l A0,A1          Der gesamte Inhalt von A0 wird
                      nach A1 kopiert
```

Diese Adressierungsart wird verwendet zum Laden, Speichern, Verschieben und Duplizieren von Daten. Ein Programm kann beschleunigt werden durch die Verwendung vieler Register als Datenspeicher, da die registerdirekte Adressierung schneller ist als bespielsweise die absolute.

Register Indirekt

Bei dieser Adressierungsart wird auf die Daten zugegriffen, deren Adressen in einem Adreßregister abgelegt sind. Es besteht zusätzlich die Möglichkeit, einen Index und einen Offset anzugeben.

Die einfachste Variante der registerindirekten Adressierung arbeitet ohne Index und Offset. Die Adresse ist damit gleich dem Inhalt des Adreßregisters.

Einige Anwendungsbeispiele für diese Adressierungsart:

Indirekter Sprung (Adr. in A0 kann vorher berechnet od. aus einer Tabelle geholt werden.)

```
A0 : $fc0002

    JMP (A0)          führt einen absoluten Sprung
                      nach $fc0002 aus.
```

Mehrmaliger Zugriff auf eine Adr. (Maustastenabfrage):

```
    LEA $bfe001,A0    lädt die Adr. $bfe001 nach A0.
l1: BTST #6,(A0)      testet Bit 6 (linke Maustaste).
    BNE.s l1          verzweigt, wenn nicht gedrückt.
l2: BTST #6,(A0)      testet Bit 6.
    BEQ.s l2          verzweigt, wenn gedrückt.
```

Eine Ergänzung dieser Adressierungsart besteht darin, das Adreßregister nach der Befehlsausführung zu erhöhen (Postinkrement) oder davor zu verringern (Prädekrement). Die Adreßveränderung richtet sich nach der Verarbeitungslänge.

Beispiel mit Postinkrement:

```
MOVE.b #120,(A0)+      Speichere den Wert 120 unter
                       der Adresse in A0 und erhöhe
                       danach A0 um 1.
```

Beispiel mit Prädekrement:

```
MOVE.l d0,-(A1)        Verringere A1 um 4 und danach
                       speichere D0 unter der Adr.
                       in A0.
```

Die indirekte Adressierung mit Postinkrement oder Prädekrement findet ihre Anwendung bei Blockverschiebungen, Tabellenzugriffen und Benutzerstapeln. Jedes Adreßregister kann als Stackpointer verwendet werden.

Die nächste Abwandlung ist die indirekte Adressierung mit einem Offset. Hierbei errechnet sich die Adresse aus dem Inhalt des Adreßregisters plus dem Offset. Dieser Offset umfaßt 16 Bit und ist vorzeichenbehaftet.

Anwendungsbeispiele:

Auswahl von Registern eines Peripheriebausteins:

```
LEA chipbasis,A0       Basisadr. des Chips nach A0
MOVE register(A0),D0   Inhalt des Registers nach D0
```

Aufruf von Amiga-Betriebssystemroutinen (wird später genauer beschrieben):

```
LEA librarybase,A6     Basisadr. d. Library nach A6
JSR routine(A6)        ruft Routine der Library auf
```

Tabellenzugriff:

```
LEA tabstart,A0        Tabellenstart nach A0
MOVE.b element(A0),D0  Tabellenelement nach D0
```

Die letzte und aufwendigste Variante der indirekten Adressierung benutzt Offset und Index. Der Offset hat eine Länge von 8 Bit. Der Index wird in einem Daten- oder Adreßregister abgelegt und kann ein Wort oder Langwort sein. Offset und Index werden vorzeichenrichtig auf 32 Bit erweitert. Zur Adreßberechnung addiert man einfach zu den Inhalten des Adreß- und Indexregisters den Offset.

Hier zwei Beispiele:

```
A0 : $24000        D0 : unbestimmt    D1 : $200

MOVE $14(A0,D1.l),D0  Kopiert den Inhalt der Adresse
                      $24214 nach D0.

Berechnung der Adresse:
        A0      :    $00024000
        D1      :   +$00000200
        Offset  :   +$00000014

        Adresse:     $00024214

A0 : $fe8000       D1 : $12310        D2 : unbestimmt

MOVE $c8(A0,D1.w),D2  Kopiert den Inhalt der Adresse
                      $ffffa2d8 nach D2.

Berechnung der Adresse:
        A0      :    $00fe8000
        D1      :   +$00012310
        Offset  :   +$ffffffc8  (negativer Offset)

        Adresse:     $ffffa2d8
```

Diese Adressierungsart wird in Umwandlungsprogrammen, beispielsweise für Zeichencodes eingesetzt, bei denen auf verschiedene, in Beziehung stehende Tabellen zugegriffen wird. Sie kann aber ebenfalls bei komplexen Blockverschiebungen oder ähnlichen Speicherzugriffen benutzt werden.

Unmittelbar

Im Gegensatz zu anderen Adressierungsarten wird bei der unmittelbaren Adressierung anstelle einer Adresse das Datum direkt hinter dem Opcode angegeben. Das Datum kann ein Byte, Wort oder Langwort sein. Man unterscheidet zwischen kurzer und langer unmittelbarer Adressierung. Diese Differenzierung bezieht sich auf die Verarbeitungslänge; Byte- und Wortverarbeitung wird als kurz bezeichnet, Langwortverarbeitung als lang. Eine unmittelbare Adressierung kann nur auf den Quelloperanden erfolgen. Gelegentlich wird die unmittelbare Adressierung auch als Konstantenadressierung bezeichnet.

Beispiele:

`MOVE.l #23,D0`	Speichert die Zahl 23 in D0 (unmittelbar lang).
`ADDI.b #12,D2`	Addiert zu D2 die Zahl 12 (unmittelbar kurz).
`CMPI #$40,D1`	Vergleicht D1 mit der Zahl $40 (unmittelbar kurz).

Die unmittelbare Adressierung erlaubt das direkte Laden, Speichern, Addieren, usw. von Konstanten. Sie kann daher dazu verwendet werden, ein Register oder eine Speicherstelle mit einem bestimmten Wert zu versehen.

Implizit

Ein Kennzeichen der impliziten Adressierung ist die Tatsache, daß alle notwendigen Informationen zu den Operatoren bereits im Opcode enthalten sind. Da im Opcode keine langen Adressen untergebracht werden können, benutzt man als Adreßzeiger meistens die internen Register. Aus diesem Grund ist auch die direkte Adressierung als implizit anzusehen. Es gibt aber auch noch eine andere Form der impliziten Adressierung, sie wird bei den Befehlen MOVEQ, ADDQ und SUBQ eingesetzt. In diesem Fall wird eine 8-Bit-(MOVEQ) oder 3-Bit-Konstante im Opcode

untergebracht. Auf diese Weise spart man das Laden einer Konstanten in ein internes Register. Generell läßt sich sagen, daß ein Befehl mit impliziter Adressierung schneller abgearbeitet wird als mit einer anderen Adressierungsart.

Beispiele für die implizite Adressierung:

`SWAP D0`	**Tauscht die Registerhälften von D0 aus.**
`MOVEQ #12,D2`	**Bringt vorzeichenbehaftete Konstante (12) nach D2.**
`ADDQ #8,A0`	**Addiert vorzeichenlose Konstante (8) zu A0.**
`RTS`	**Rücksprung aus Unterprogramm**

Bei der impliziten Adressierung hängt die Anwendung häufig allein vom Befehl ab, da viele von ihnen nur die implizite Adressierung zulassen. Die Befehle MOVEQ, ADDQ und SUBQ lassen sich sehr gut in Schleifenkonstruktionen verwenden.

Absolut

Diese Adressierungsart sollte Ihnen vom 6510 her bestens bekannt sein. Beim MC68000 werden zwei Arten unterschieden, absolut lang und absolut kurz. Die absolut lange Adressierung benötigt die Angabe einer 32-Bit-Adresse (24 Bit werden verwendet) und kann somit den gesamten Adreßbereich direkt ansprechen. Bei der kurzen Adressierung nur ein Bereich von 64 KByte ansprechbar. Durch die Verwendung einer vorzeichenbehafteten 16-Bit-Adresse teilen sich die 64 KByte in zwei 32 KByte Blöcke auf. Ein Block liegt am Anfang, der andere am Ende des vom MC68000 adressierbaren Bereichs.

Beispiele:

`MOVE.l $fc0000,D0`	**lädt den Inhalt von $fc0000 nach D0.**

`ASL $20440`	schiebt den Inhalt von $20440 um ein Bit nach links.
`MOVE.b $200,D1`	Inhalt von $200 nach D0 (absolut kurz)

Die absolute Adressierung wird für den Zugriff auf fest definierte Adressen (Prozessorvektoren, Chip-Register) verwendet. Die kurze Adressierung ist, wenn möglich, der langen vorzuziehen, da sie weniger Speicherplatz belegt und eine kürzere Ausführungszeit hat.

Programmzähler Relativ

Sie haben bis jetzt die Programmzähler-Relative Adressierung (PC-relative) nur in Form der Branch-Befehle des 6510 kennengelernt. Der MC68000 bietet zusätzlich die Möglichkeit, den PC wie bei der indirekten Adressierung als Adreßzeiger zu verwenden. Dadurch können frei verschiebbare Programme geschrieben werden, ohne erheblichen programmiertechnischen Aufwand. Nun werden Sie sich sicher fragen, welchen Zweck man damit verfolgt, ein Programm frei verschiebbar zu machen. Das liegt an der Konzeption eines Multitasking-Betriebssystems. Solche Betriebssysteme besitzen eine dynamische Speicherverwaltung, das heißt, Sie weisen jedem Programm einen freien Speicherplatz zu. Da dieser Speicherplatz von Aufruf zu Aufruf verschieden ist, müssen die Programme in jedem Speicherbereich lauffähig sein.

Wir unterscheiden zwei Varianten der PC-relativen Adressierung,und zwar mit Offset und mit Offset und Index. Die Handhabung ist die gleiche wie bei der indirekten Adressierung, so daß wir uns mit ihr nicht näher befassen werden. Das nachfolgende Beispiel soll nur den Unterschied zur absoluten Adressierung verdeutlichen.

Programm 1 (absolut):

```
wert:   dc.w 255
start:  MOVE wert,D0        Übertrage absolut nach D0.
        RTS
```

```
Programm 2 (PC-relativ):

wert:   dc.w 255
start:  MOVE wert(PC),D0    Übertrage relativ nach D0.
        RTS
```

In beiden Programmen wird der Wert 255 in das Datenregister D0 übertragen. Um das erste Programm in einem anderen Speicherbereich ablaufen zu lassen, muß es erst neu assembliert werden. Das zweite dagegen ist ohne Änderungen in jedem Bereich ablauffähig.

Mit der PC-relativen Adressierung bekommt man ein ausgezeichnetes Hilfsmittel zur Erstellung von frei verschiebbaren Programmen.

Zum Schluß wollen wir noch eine Tabelle abdrucken, die alle Adressierungsarten mit Assemblersyntax und Adreßberechnung angibt.

Adressierungsart	Syntax	Adreßberechnung
Register Direkt		
Datenregister direkt	Dn	a=Dn
Adreßregister direkt	An	a=An
Statusregister direkt	SR	a=SR
Conditioncode-Register direkt	CCR	a=CCR
Register Indirekt		
Adreßregister indirekt (ARI)	(An)	a=(An)
ARI mit Postinkrement	(An)+	a=(An);An=An+X
ARI mit Prädekrement	-(An)	An=An+X;a=(An)
ARI mit Offset	o16(An)	a=(An)+o16
ARI mit Offset und Index	o8 (An,Rn.x)	a=(An)+(Rn)+o8
Unmittelbar		
Byte	#$b	8 Bit Operand
Wort	#$w	16 Bit Operand
Langwort	#$l	32 Bit Operand
Implizit		
unmittelbar schnell	versch.	versch.

```
Absolut
absolut kurz                    $w          a=$w
absolut lang                    $l          a=$l

Programmzähler Relativ
relativ mit Offset              o16(PC)     a=(PC)+o16
relativ mit Offset und Index    08 (PC,Rn.x) a=(PC)+(Rn)+o8

Erläuterungen zur Tabelle:

n   Registernummer
x   Operandengröße (Byte, Wort, Langwort)
X   1, 2 und 4 (entsprechend der Operandengröße)
o8  vorzeichenbehafteter Offset (8 Bit)
o16 vorzeichenbehafteter Offset (16 Bit)
$b  Byte
$w  Wort
$l  Langwort
R   A oder D (Adreß- oder Datenregister)
()  Inhalt des Registers
a   Adresse des Operanden
```

4.1.5 Der Befehlssatz des MC68000

Die Effizienz eines Maschinenprogramms richtet sich in erster Linie nach der Ausführungsgeschwindigkeit der einzelnen Befehle und den verwendbaren Adressierungsarten. Nicht zuletzt führt auch ein überschaubarer Befehlssatz zu schnelleren Programmen. Dies liegt daran, daß der Programmierer alle Befehle und deren Adressierungsarten kennt und somit die Programme optimal an die Aufgabenstellung anpassen kann. Der Befehlssatz des MC68000 erfüllt diese Kriterien und ist damit im Vergleich zu anderen 16-Bit-Prozessoren (Intel 8086) leicht zu erlernen. Der C64-Besitzer hat zudem noch den Vorteil der vielen Übereinstimmungen mit dem 6510.

Wir werden den Befehlssatz in Kategorien einteilen und diese nacheinander anhand der wichtigsten Befehle beschreiben. Der Befehlssatz läßt sich in die folgenden acht Kategorien einteilen:

Datentransportbefehle

Arithmetische Befehle

BCD-Befehle

Logische Befehle

Bitmanipulationsbefehle

Verschiebe- und Rotationsbefehle

Programmsteuernde Befehle

Systemsteuernde Befehle (Privilegierte Befehle)

Datentransportbefehle

Wie der Name schon sagt, können mit diesen Befehlen Daten aus Registern und/oder Speicherstellen geladen, gespeichert übertragen oder ausgetauscht werden. Der universellste und am häufigsten verwendete Befehl für den Datentransport ist MOVE. Seine Syntax lautet:

```
MOVE.x <Quelle>,<Ziel>
```

Mit Hilfe des MOVE-Befehls können beliebige Daten aus dem Quelloperanden in den Zieloperanden transferiert werden. Dabei sind auf den Quelloperanden alle Adressierungsarten anwendbar und auf den Zieloperanden alle außer der PC-relativen und unmittelbaren nicht. Daraus ergeben sich 88 (!) verschiedene Adressierungsvarianten, die die Vielseitigkeit dieses Befehls verdeutlichen.

Arithmetische Befehle

Mit Hilfe der arithmetischen Befehle lassen sich die vier Grundrechenarten, Vergleiche und Negationen durchführen. Die Addition, Subtraktion und Negation können wahlweise das Übertrags-Flag berücksichtigen oder nicht. Sollte bei der Subtraktion ein Borgen auftreten, das heißt das Ergebnis negativ sein, so wird das Übertrags-Flag gesetzt und nicht wie beim 6510 gelöscht.

Das nun folgende Beispiel bildet die Prüfsumme über das Amiga-Betriebssystem und ist dem Boot-ROM des Amiga 1000 entnommen.

```
A0: $00fc0000   D1: $00010000   A5 enthält Rücksprungadr.

test: MOVE.l D1,D2          ;Überträgt D1 nach D2.
      SWAP D2               ;Vertauscht Registerhälften.
      CLR.l D0              ;Löscht D0.
      BRA.s mark            ;Verzweige auf mark.
loop: ADD.l (A0)+,D0        ;Add. Langwort aus A0 zu D0
                            (65536 Durchläufe=256KByte).
      BCC.s mark            ;Verzw., wenn kein Übertrag.
      ADDI.l #$00000001,D0  ;Übertrag zu D0 addieren.
mark: DBF D1,loop           ;D1=D1-1,verzw. nach loop.
      DBF D2,loop           ;D2=D2-1,verzw. nach loop.
      NOT.l D0              ;Invert. D0.
      JMP (A5)              ;Rücksprung.
```

BCD-Befehle

Im Gegensatz zum 6510 unterscheidet der MC68000 die BCD-Befehle von den arithmetischen anhand des Opcodes und nicht durch ein Dezimal-Flag. Als Datenformat ist lediglich das Byte verwendbar, das in zwei Nibble (4 Bit) mit je einer BCD-Ziffer unterteilt ist. Zahlen im BCD-Format können addiert, subtrahiert und negiert werden. Ein Beispiel ist überflüssig, da BCD-Befehle und arithmetische Befehle ähnlich gehandhabt werden. Zudem wird mit BCD-Zahlen sehr selten operiert.

Logische Befehle

Über diese Befehle können Operanden logisch miteinander verknüpft oder bitweise invertiert werden. Meistens verwendet man sie zum Testen, Setzen oder Löschen von einzelnen oder mehreren Bits. So können Sie zum Beispiel die Flags des Statusregisters nur über diese Befehle verändern.

Beispiele:

```
Setzen von Bits:
     MOVE register,D0          Lädt Register nach D0.
     ORI #6,D0                 Setzt Bit 1 und 2.
     MOVE D0,register          Speichert D0 im Register.
   oder:
     ORI #6,register           Setzt Bit 1 und 2 im Register.

Löschen des Übertrags-Flags:
     AND #$fe,CCR            Löscht Bit 0 (C-Flag) im CCR.

Testen eines Bits (linke Maustaste):
  w: ANDI.b #64,$bfe001      Bit 6 isolieren.
     BNE w                     Warte bis Taste gedrückt.
```

Bitmanipulationsbefehle

Die Bitmanipulationsbefehle haben ein ähnliches Anwendungsfeld wie die logischen Befehle. Sie können nur zum Testen, Setzen oder Löschen einzelner Bits herangezogen werden, haben aber den Vorteil der kürzeren Befehlslänge und schnelleren Ausführung. Da die Befehle selten gebraucht werden, verzichten wir an dieser Stelle ebenfalls auf ein Beispiel.

Verschiebe- und Rotationsbefehle

Die Verschiebe- und Rotationsbefehle werden dazu verwendet, die Reihenfolge der Bits in einem Operanden zu verändern. So ist bespielsweise die Verarbeitung anderer Datenformate möglich. Die Befehle entsprechen in ihrer Funktion weitgehend denen des 6510. Neu hinzugekommen ist ein Rotationsbefehl, der nur die Bits des Operanden rotiert (6510 speichert immer ein Bit im Übertrag-Flag.).

Als Anwendungsbeispiel für die Rotationsbefehle nehmen wir eine Umwandlungsroutine, die eine Hexadezimalzahl in eine Oktalzahl (Basis 8) konvertiert. Das Programm errechnet aus zwei Hex-Ziffern (hex1, hex2) drei Oktal-Ziffern (oct1, oct2, oct3). Die beiden Hex-Ziffern sind frei wählbar und können vor dem Start geändert werden.

```
start:CLR D0            ;D0 löschen.
      MOVE.b hex2,D0    ;Erste Hex-Ziffer nach D0 übertragen.
      ROR #4,D0         ;Um vier Stellen n. rechts rotieren.
      MOVE.b hex1,D0    ;Zweite Hex-Ziffer nach D0.
      ROR #2,D0         ;Um zwei Stellen n. rechts rotieren.
      MOVE.b D0,oct1    ;Verbleib. Bits unter oct1 speichern.
      ROL #3,D0         ;Nächsten 3 Bits zurückrotieren.
      ANDI.b #7,D0      ;und isolieren.
      MOVE.b D0,oct2    ;Bits unter oct2 speichern.
      ROL #3,D0         ;Letzten 3 Bits zurückrotieren,
      ANDI.b #7,D0      ;Isolieren
      MOVE.b D0,oct3    ;und unter oct3 wegspeichern.
      RTS               ;Rücksprung aus der Routine.

hex1: dc.b $07          ;hex1 und hex2 können vor
hex2: dc.b $02          ;Aufruf geändert werden.
oct1: dc.b 0            ;oct1, oct2 und oct3
oct2: dc.b 0            ;Enthalten das Ergebnis
oct3: dc.b 0            ;der Konvertierung.
```

Programmsteuernde Befehle

Unter dieser Kategorie faßt man alle Befehle zusammen, die in irgendeiner Weise den Programmablauf ändern. Man unterscheidet dabei unbedingte und bedingte Sprünge, erstere verzweigen in jedem Fall und letztere nur bei einem bestimmten Zustand der Flags im Conditioncode-Register (CCR). Die bedingten Sprünge dürften Ihnen vom 6510 in Form der Branch-Befehle (BNE, BCS usw.) bekannt sein. In der folgenden Übersicht werden die wichtigsten Sprungbefehle des MC68000 aufgelistet.

```
Bedingte Sprungbefehle
(Das "cc" steht für eine der unten angegebenen Bedingungen.)
    Bcc label           Verzweige relativ, wenn cc wahr ist.
    DBcc Register,label Dekrementiere Register und
                        verzweige relativ, wenn cc wahr ist
                        oder Register=-1.

Unbedingte Sprungbefehle
    BRA label    Verzweige relativ.
    JMP adresse  Verzweige absolut.
    JSR adresse  Verzweige absolut in ein Unterprogramm.
```

Hier nun die Bedingungen für die Befehle Bcc und DBcc:

```
cc    Bedeutung

CC    C = 0
CS    C = 1
EQ    Z = 1
MI    N = 1
NE    Z = 0
PL    N = 0
VC    V = 0
VS    V = 1

GE    N eor V = 0
GT    Z or (N eor V) = 0
HI    C or Z = 0
LE    Z or (N eor V) = 1
LS    C or Z = 1
LT    N eor V = 1

Nur für DBcc
  T     wahr, immer erfüllt
  F     falsch, nie erfüllt
```

Systemsteuernde Befehle (Privilegierte Befehle)

Die systemsteuernden Befehle werden in erster Linie vom Betriebssystem zur Überwachung und Steuerung der Tasks und Interrupts gebraucht. Sie können nur im Supervisor-Modus verwendet werden. Da einige Befehle einen Schreibzugriff auf das gesamte Statusregister erlauben, ist es möglich, das Systembyte zu verändern. Manipulationen am Systembyte können zum Systemabsturz führen. Also, Vorsicht bei dem Umgang mit diesen Befehlen!

Beispiel (Setzen der Interrupt-Maske):

```
ANDI #$f8ff,SR  Löschen der alten Interrupt-Maske.
ORI  #$0400,SR  Auf Interruptlevel 4 setzen.
```

Zum Abschluß die Befehle des MC68000 im Überblick, geordnet nach Befehlskategorien:

Datentransportbefehle

EXG	vertauscht zwei Registerinhalte
SWAP	austausch der Registerhälften
LEA	lädt Adresse in ein Adreßregister
PEA	bringt Adresse auf den Stapel
LINK	baut neuen Stackbereich auf
MOVE	überträgt Daten von einem Operanden zum anderen
MOVEM	überträgt mehrere Register
MOVEP	überträgt Daten an die Peripherie

Arithmetische Befehle

ADD	addiert zwei Operanden
CMP	vergleicht zwei Operanden
CLR	löscht einen Operanden
DIVS	führt vorzeichenbehaftete Division durch
DIVU	führt vorzeichenlose Division durch
EXT	erweitert einen Operanden vorzeichenrichtig
MULS	führt vorzeichenbehaftete Multiplikation durch
MULU	führt vorzeichenlose Multiplikation durch
NEG	negiert einen Operanden
SUB	subtrahiert einen Operanden von einem anderen
TST	prüft den Inhalt eines Operanden

BCD-Befehle

ABCD	addiert zwei BCD-Zahlen
NBCD	negiert eine BCD-Zahl
SBCD	subtrahiert eine BCD-Zahl von einer anderen

Logische Befehle

AND	logische UND-Verknüpfung zweier Operanden
EOR	logische EOR-Verknüpfung zweier Operanden
NOT	bitweises Invertieren eines Operanden
OR	logische OR-Verknüpfung zweier Operanden

Bitmanipulationsbefehle

BCHG	testet ein Bit und invertiert es
BCLR	testet ein Bit und löscht es
BSET	testet ein Bit und setzt es
BTST	testet ein Bit

Verschiebe- und Rotationsbefehle

ASL	arithmetisches Verschieben nach links
ASR	arithmetisches Verschieben nach rechts
LSL	logisches Verschieben nach links
LSR	logisches Verschieben nach rechts
ROL	rotiert nach links
ROR	rotiert nach rechts
ROXL	rotiert nach links mit Übertrag (X-Flag)
ROXR	rotiert nach rechts mit Übertrag (X-Flag)

Programmsteuernde Befehle

Bcc	relativer Sprung, wenn cc wahr ist
BRA	unbedingter relativer Sprung
BSR	relativer Unterprogrammaufruf
CHK	testet Register gegen Grenzen des Zahlenbereichs
DBcc	dekrementiert und verzweigt bedingt
JMP	unbedingter absoluter Sprung
JSR	absoluter Unterprogrammaufruf
NOP	keine Operation
RTR	Rücksprung aus Unterprg. mit CCR-Übergabe
RTS	Rücksprung aus Unterprogramm
Scc	setze Byte, wenn cc wahr ist
TRAP	verzweigt auf Trap-Vektor
TRAPV	verzweigt auf Trap-Vektor, wenn V-Flag gesetzt

Systemsteuernde Befehle (Privilegierte Befehle)

AND	nur bei Schreibzugriffen auf das
EOR	Statusregister
OR und	
MOVE	

RESET	sendet einen Reset an die Peripherie
RTE	Rücksprung aus der Ausnahmebehandlung
STOP	hält das System an

4.1.6 Aufruf der Betriebssystemroutinen

Wie auch in BASIC und C ist es in Maschinensprache möglich, Betriebssystemroutinen in das eigene Programm einzubinden. Dazu muß auch erst die entsprechende Library geöffnet werden. Die EXEC-Library ist immer geöffnet, und ihr Basisvektor ist unter der Adresse $000004 gespeichert. Zum Öffnen einer Library verwendet man den OpenLibrary-Befehl, der in der EXEC-Library zu findet ist. So kann man beispielsweise die Intuition-Library wie folgt aufrufen:

```
execbase    = $000004                ;Basisadr. der EXEC-Lib
OpenLibrary = -552                   ;Offset auf den Befehl

        MOVE.l execbase,A6           ;lade Basisadr. n. A6
        LEA.l intuitionname,A1       ;Zeiger auf Name n. A1
        MOVEQ #0,D0                  ;Versionsnummer n. D0
        JSR OpenLibrary(A6)          ;Aufruf der Routine
        MOVE.l D0,intuitionbase      ;speich. der Basisadr.
        RTS                          ;Rücksprung

intuitionname:  dc.b "intuition.library",0
intuitionbase:  dc.l 0
```

Nach dem Aufruf dieser Routine ist die Basisadresse der Intuition-Library in der 'intuitionbase' gespeichert, sie kann dort jederzeit ausgelesen werden.

Wir wissen nun wie man eine Library mit Maschinensprache öffnet und können jetzt mit Hilfe der 'intuitionbase' einzelne Routinen der Intuition-Library ansprechen.

```
OpenWorkBench = -210                 Offset einer
                                     Intuition-Routine
MOVE.l A6,-(A7)                      bringe A6 auf Stapel
MOVE.l intuitionbase,A6              intuitionbase n. A6
JSR OpenWorkBench(A6)                Aufruf der Routine
MOVE.l (A7)+,A6                      hole A6 vom Stapel
```

Zur Konvention der Routinen

Man sollte grundsätzlich das Adreßregister A6 zur Aufnahme der Basisadresse benutzen, da einige Routinen diese Adresse in A6 erwarten. Rückgabeparameter werden immer nach D0 übergeben. Die Routinen verändern die Register D0, D1, A0 und A1. Falls Sie den Inhalt noch benötigen, müssen Sie ihn vorher sichern!

Nachdem Ihr Programm die Routine ausgeführt hat, sollte die Library wieder geschlossen werden. Dies geschieht mit dem CloseLibrary-Befehl:

```
CloseLibrary  = -414              Offset der
                                  CloseLibrary-Routine
MOVE.l intuitionbase,A1           Basisadr. nach A1
JSR CloseLibrary(A6)              Aufruf der Routine
```

Hier folgen nun einige nützliche Betriebssystemroutinen mit den entsprechenden Offsets und Parameterangaben:

Routine	Offset	Übergabeparameter	Rückgabeparameter
EXEC-Lib.			
AllocMem	-198	D0:Anzahl d. Bytes	D0:Basisadresse
(Speicher belegen)		D1:Speichertyp	des beleg. Speichers
FreeMem	-210	A1:Basisadr.	
(Speicher freigeb.)		freizugeb. Speichers	
		D0:Anzahl d. Bytes	
OpenLibrary	-552	A1:Libraryname	D0:Basisadress
(Library öffnen)		D0:Versionsnummer	der Library
CloseLibrary	-414	A1:Basisadresse	
(Library schließen)		der Library	
DOS-Lib.			
Open	-30	D1: Dateiname	D0:Handle-Nummer
(Datei öffnen)		D2:Zugriffsart	
Close	-36	D1: Handle-Nummer	
(Datei schließen)			
Read	-42	D1:Handle-Nummer	
(Bytes lesen)		D2:Bufferadr.	
		D3:Bufferlänge	
Write	-48	wie bei Read	
(Bytes schreiben)			
INTUITION-Lib.			
OpenScreen	-198	A0:NewScreen-Struct	D0:Screen-
(Screen öffnen)			Struct
CloseScreen	-66	A0:Screen-Struct	
(Screen schließen)			
penWindow	-204	A0:NewWindow-Struct	D0:Window-
(Window öffnen)			Struct
CloseWindow	-72	A0:Window-Struct	
(Window schließen)			
OpenWorkBench	-210		
(Workbenchscreen öffnen)			
CloseWorkBench	-78		
(Workbenchscreen schließen)			

Das nun folgende Programm öffnet für kurze Zeit ein Window auf dem Workbenchscreen.

```
execbase     = $000004

OpenLibrary = -552              ;Offsets
OpenWindow  = -204              ;der
CloseWindow = -72               ;Betriebssystemroutinen

start: MOVE.l execbase,A6      ;execbase nach A6
       LEA intuitionname,A1     ;Library-Name nach A1
       MOVEQ #0,D0              ;Versionsnummer nach D0
       JSR OpenLibrary(A6)      ;holt Basisadr. der Library
       MOVE.l D0,A6             ;Basisadresse nach A6
       LEA window,A0            ;neuer Window-Struct-Zeiger
       JSR OpenWindow(A6)       ;öffnet Window
       MOVE.l #300000,D1        ;Verzögerungsschleife
wait:  SUBQ.l #1,D1
       BNE.s wait
       MOVE.l D0,A0             ;Window-Struct-Zeiger nach A0
       JSR CloseWindow(A6)      ;schließt Window
       RTS                      ;Rücksprung

window: dc.w 20                 ;x-Koord. der li. ob. Ecke
       dc.w 20                  ;y-Koord.
       dc.w 300                 ;Breite des Windows
       dc.w 100                 ;Höhe des Windows
       dc.b 0                   ;Hintergrundfarbe
       dc.b 1                   ;Zeichenfarbe
       dc.l 0                   ;keine bes. Darstellungsarten
       dc.l $1000               ;Darstellungsart
       dc.l 0                   ;keine Gadgets
       dc.l 0                   ;Default-Checkmark (n. verw.)
       dc.l title               ;zeigt auf Namen des Windows
       dc.l 0                   ;kein benutzerdef. Screen
       dc.l 0                   ;keine benutzerdef. Bitmap
       dc.w 0                   ;minimale Größe
       dc.w 0                   ;des Windows
       dc.w 0                   ;maximale Größe
       dc.w 0                   ;des Windows
       dc.w 1                   ;Windowart
title:          dc.b "Mein erstes Window",0   ;Windowname
intuitionname:  dc.b "intuition.library",0    ;Library-Name
```

4.1.7 Die Speicherverwaltung des Amiga

Beim C64 mußte sich der Programmierer in Maschinensprache seine eigenen Gedanken über die Speicheraufteilung machen. So hat er zum Beispiel einen Assembler im Speicher ab $8000 untergebracht und vielleicht noch einen Maschinensprache-Monitor im $c000-Bereich. Diese Programme liefen ohne erhebliche Änderungen in keinem anderen Speichenbereich.

Im Gegensatz zu einem C64-Programm ist ein Amiga-Programm prinzipiell überall im Speicher ablaufbar. Das Betriebssystem ist in der Lage, mit Hilfe einer zusammen mit dem Programm abgespeicherten Tabelle alle absoluten Adressen eines Programms beim Laden an den jeweiligen Speicherbereich anzupassen. Sie selbst bemerken von dieser Tabelle normalerweise nichts, da Ihr Assembler die Daten automatisch erzeugt. Somit muß das Programm nicht unbedingt in PC-relativem Code geschrieben sein.

Doch nun zur eigentlichen Speicherverwaltung des Amiga. Es handelt sich hierbei um eine dynamische Speicherverwaltung. Zu diesem Zweck legt der Amiga eine Liste an, in der festgehalten wird, welche Speicherbereiche belegt sind und welche noch frei sind. Diese Liste ist vergleichbar mit der Blockbelegungstabelle (BAM) einer Diskette. Der gesamte Speicher, der in dieser Liste registriert ist, kann mit Hilfe der Befehle AllocMem und FreeMem entweder belegt oder freigegeben werden.

Ein Speicherbereich, der als belegt gekennzeichnet ist, wird vom Betriebssystem aus nicht mehr verändert.Ein freier Speicher dagegen kann verändert werden. Deshalb sollten Sie unbedingt jeden Speicherbereich, den Sie für ein Programm benötigen, durch die AllocMem-Funktion belegen lassen.

Die AllocMem- und FreeMem-Funktion sind Bestandteile der EXEC-Library.

```
AllocMem      Offset: -198
              Übergabeparameter: D0 Anzahl der Bytes
                                 D1 Speichertyp
```

```
                Rückgabeparameter: D0 Beginn des als belegt
                                      gekennzeichneten
                                      Speicherblocks

FreeMem         Offset: -210
                Übergabeparameter: A1 Beginn des freizugeb.
                                      Speichers
                                   D0 Anzahl der Bytes
```

Der unter AllocMem zu findende Parameter 'Speichertyp' gibt an, ob der Speicher im Chip-Memory (unteren 512 KByte) belegt werden soll, so daß sämtliche Custom-Chips darauf zugreifen können, oder im Fast-Memory, worauf die Chips keinen Zugriff haben. Weiterhin kann über diesen Parameter angegeben werden, ob der Speicher vorher gelöscht werden soll (MEMF_CLEAR) und ob mehrere Programme diesen Speicher benutzen dürfen (MEMF_PUBLIC). Hier nun die entsprechenden Parameterwerte:

Funktion	Parameterwert
MEMF_CHIP	2
MEMF_FAST	4
MEMF_CLEAR	65536
MEMF_PUBLIC	1

Durch Kombination der Parameterwerte ist es möglich, mehrere Funktionen gleichzeitig anzuwählen, wobei MEMF_CHIP und MEMF_FAST zusammen unsinnig wäre. Wird als Parameterwert weder Chip- noch Fast-Memory spezifiziert, so wird zuerst versucht, Fast-Memory zu belegen, danach erst Chip-Memory.

Nun kann es einmal sein, daß Sie den gesamten Speicher belegen wollen, um ihn beispielweise als Soundspeicher zu verwenden. Sie wissen jedoch nicht, wieviel Speicherplatz zur Verfügung steht. Dazu können Sie einen weiteren Befehl aus der EXEC-Library verwenden, nämlich AvailMem.

```
AvailMem        Offset: -216
                Übergabeparameter: D1 Speichertyp
                                      (wie bei AllocMem)
                Rückgabeparameter: D0 Anzahl der Bytes
```

Das abschließende Beispiel zeigt Ihnen, wie Sie das gesamte zur Verfügung stehende Chip-Memory erfahren können.

```
execbase = $000004

AvailMem = -216

start:    MOVE.l execbase,A6    ;execbase nach A6
          MOVE.l #2,D1          ;Chip-Memory
          JSR AvailMem(A6)      ;sucht Anz. d. verfüg. Bytes
          MOVE.l Fo, ChipMem    ;Wert ablegen
          RTS                   ;Rücksprung
          ChipMem: DC.l 0
```

4.2 Programmierung der Custom-Chips

4.2.1 Die Aufgaben der Chips

Speziell für den Amiga wurden drei neue Chips entwickelt, die den Prozessor entlasten sollen. Alle Bausteine sind in der Lage, ihre Daten unabhängig vom Prozessor aus dem Speicher auszulesen. Dies geschieht durch direkten Speicherzugriff, genannt DMA. In der nun folgenden Beschreibung wird auf die Chips aller Amigatypen eingegangen, einschließlich der CIAs.

Agnus/Fat Agnus

Zu den Aufgaben der Agnus gehören die Kontrolle der DMA, das Verschieben beliebiger Daten, die Steuerung der Bildschirmausgabe und die Generation der RAM-Adressen. Der Fat Agnus, der zur Zeit nur im Amiga 500 eingesetzt wird, erzeugt zusätzlich noch das Taktsignal.

Der wichtigste Bestandteil der Agnus ist ein Co-Prozessor, der Copper genannt wird. Für ihn müssen richtige Programme geschrieben werden! Er beherrscht die folgenden drei Befehle:

WAIT x,y	Warte, bis der Rasterstrahl die Position x,y erreicht hat.
SKIP x,y	Überspringe den nächsten Befehl, falls der Rasterstrahl die Position x,y bereits durchlaufen hat.
MOVE d,r	Übertrage die Daten d in das Register r.

Mit Hilfe dieser drei Befehle kann man zum Beispiel während eines Bildschirmaufbaus durch den Rasterstrahl die Farben an ganz bestimmten Positionen verändern. Ein Copper-Programm kann somit eine aufwendige Rasterstrahlprogrammierung durch den 68000-Prozessor ersetzen.

Ein weiterer Bestandteil der Agnus ist der Bit-Blitter. Dieser verfügt über drei DMA-Eingänge und einen DMA-Ausgang, die unabhängig vom Prozessor Daten im Speicher manipulieren können. Die DMA-Logik kann nun die drei Eingänge logisch miteinander verknüpfen und zum Beispiel im Grafikspeicher ablegen. Das bedeutet im Klartext, daß der Blitter mit extrem hoher Geschwindigkeit Speicherblöcke verschieben und verknüpfen kann. Sie können sich sicher vorstellen, daß schnelle Grafikanimationen für den Amiga deshalb kein Problem sind. Außerdem kann der Blitter Linien zeichnen und Flächen ausfüllen mit einer Geschwindigkeit von maximal einer Millionen Punkte pro Sekunde.

Der Agnus ist auch für die DMA-Kontrolle der anderen Chips verantwortlich. So kann man z.B. gleichzeitig über DMA ein Programm einlesen, einen 4-Kanal-Sound abspielen und eine hochauflösende Grafik darstellen, ohne den Prozessor merklich abzubremsen.

Denise

Dieser Chip ist für die Verwaltung der Grafik verantwortlich. Er erlaubt die Verarbeitung vier verschiedener Bildschirmformate:

Format	Max. Farben
320*256	32
320*512	32
640*256	16
640*512	16

Die vertikale Auslösung von 512 Punkten ist nur im sogenannten Interlace-Modus möglich. Dabei wird die Grafik in zwei Halbbilder zerlegt, die dann nacheinander, leicht verschoben auf dem Monitor angezeigt werden. Aufgrund der niedrigeren Bildwechselfrequenz (25 statt 50) entsteht ein Flackern, das abhängig vom Kontrast des Bildes ist.

Die Farben können aus 4096 verschiedenen ausgewählt und in speziellen Farbregistern abgelegt werden. Der Grafikprozessor holt sich dann aus dem Grafikspeicher die Nummer des Farbregisters, liest die entsprechende Farbe aus und stellt sie auf dem Monitor dar.

Es ist auch möglich, bis zu 4096 Farben gleichzeitig darzustellen. Die Denise verwendet dazu den Hold-and-Modify-Modus (HAM-Modus). In diesem Modus kann von Punkt zu Punkt nur ein Farbanteil (Rot, Grün oder Blau) geändert werden. Die so erstellten Grafiken besitzen nahezu Fernsehqualität.

Der Amiga verfügt auch über acht Hardware-Sprites, die 16 Punkte breit sein dürfen und beliebig hoch. Ein Sprite kann maximal vier Farben haben, durch Zusammenlegen zweier Sprites erhöht sich die Farbanzahl auf 16. Es ist mit Hilfe des Coppers möglich, ein Sprite mehrmals auf dem Bildschirm darzustellen, falls ein Zwischenraum von mindestens einer Rasterzeile vorhanden ist. Es können auch Sprite-Sprite- und Sprite-Grafik-Kollisionen erkannt sowie die Prioritäten zweier sich überschneidender Sprites festgelegt werden.

Zu guter Letzt kontrolliert Denise auch noch die Joystick-/Maus-Ports.

Paula

Dieser Chip steuert in erster Linie die 4-Kanal-Sound-Erzeugung. Jeweils zwei Kanäle werden zu einem Stereo-Kanal zusammengelegt. Anders als beim SID, bei dem die Wellenformen (Rechteck, Dreieck, usw.) bereits vorgegeben waren, müssen sie beim Amiga erst definiert werden. Dadurch sind natürlich wesentlich komplexere Wellenformen erzeugbar. So ist zum Beispiel im Amiga-Betriebssystem ein Sprachsynthesizer eingebaut. Es kann auch ein Kanal über einen anderen amplituden- und/oder frequenzmoduliert werden. Die Lautstärke ist im Gegensatz zum SID für jeden Kanal getrennt wählbar.

Die Interrupt-Verwaltung obliegt der Paula. Sie ist in der Lage, bis zu 16 verschiedene Interrupts zu erkennen. Durch diese Verwaltung ist erst das Multitasking des Amiga möglich.

Schließlich enthält die Paula noch einen DMA-fähigen Diskcontroller, der zwischen zwei Aufzeichnungsverfahren (MFM, GCR) auswählen kann, und zwei Analog-Digital-Wandler für die anschließbaren Paddles.

CIAs

Die CIAs sind für die Kommunikation mit der Außenwelt vorgesehen. Mit ihnen wurden die parallele und serielle Schnittstelle, einige Disksteuersignale und die Tastatur-Schnittstelle realisiert. Außerdem werden die Timer für die Steuerung des Multitaskings verwendet.

Übrigens: Die CIAs im Amiga sind, bis auf die nicht vorhandene Echtzeituhr, völlig software- und hardwarekompatibel zu denen, die im C64 verwendet werden.

4.2.2 Direkter Speicherzugriff - Direkter Registerzugriff

Unter dem direkten Speicherzugriff (engl.: Direct Memory Access = DMA) versteht man das Laden oder Speichern von Daten unabhängig vom Prozessor. Damit der Prozessor durch den

DMA-Betrieb nicht abgebremst wird, greift die DMA-Logik nur auf den Speicher zu, wenn der Prozessor interne Operationen ausführt. Dies ist in den meisten Fällen möglich, eine Ausnahme bildet jedoch der Bit-Blitter. Da der Blitter Daten aber sehr viel schneller verschieben kann als der Prozessor, ist seine Anwendung trotzdem empfehlenswert.

Beim direkten Registerzugriff werden die Daten mit Hilfe des Prozessors direkt in die Register übertragen. Dies führt im Vergleich zum DMA-Betrieb zu deutlichen Geschwindigkeitseinbußen.

Doch nun zurück zur DMA. Um mittels DMA Daten zu übertragen, müssen der Logik bestimmte Parameter übergeben werden wie Länge der Daten in Worten, Zugriffsart (lesen/schreiben) und Anfangsadresse. Danach muß die DMA über das DMA-Kontrollregister gestartet werden. Damit der Prozessor weiß, wann die DMA-Übertragung abgeschlossen ist, wird in einem weiteren Register die Beendigung angezeigt.

Hier nun die notwendigen Register:

Register	Adresse	Bedeutung
DMACONR	$dff002	DMA-Kontrollreg.(nur lesbar)
DMACONW	$dff096	DMA-Kontrollr.(nur schreibb.)

Über das DMACONW Register wird die DMA ein- und ausgeschaltet. Das DMACONR Register zeigt das Ende der DMA an. Beide Register haben folgende Belegung:

BIT	Beschreibung
15	gibt Lösch- oder Setzmodus an (SET=1)
14	nicht erforderlich
13	nicht erforderlich
12	unbenutzt
11	unbenutzt
10	nicht erforderlich
9	erlaubt DMA-Betrieb
8	Bitplane DMA
7	Copper DMA
6	Blitter DMA
5	Sprite DMA
4	Disk DMA
3-0	Audiokanal 3-0 DMA

Der DMA-Betrieb ist nur bei gesetztem Bit 9 möglich.

Die nun folgenden Programme erzeugen einen Ton mit einer Frequenz von etwa 1000 Hertz. Das erste Programm benutzt die Audio-DMA, im zweiten Programm werden die Audioregister direkt geladen.

```
audloc  = $dff0a0                ;Anfangsadr. der Daten
audlen  = $dff0a4                ;Länge in Worten
audper  = $dff0a6                ;Ablaufgeschw.
audvol  = $dff0a8                ;Lautstärke
dmaconw = $dff096                ;DMA-Kontrollreg.

start:  MOVE.l #dreieck,audloc   ;übertrage Anfangsadr.
        MOVE #8,audlen           ;übertr. Länge d. Dat.
        MOVE #64,audvol          ;übertr. Lautstärke
        MOVE #224,audper         ;übertr. Ablaufgeschw.
        MOVE #$8201,dmaconw      ;schalte DMA ein
wait:   BTST #6,$bfe001          ;teste linke Maustaste
        BNE.s wait               ;warte bis gedrückt
        MOVE #$0001,dmaconw      ;schalte DMA aus
        RTS                      ;Rücksprung

dreieck:dc.b 0,30,60,90          ;Sounddaten
        dc.b 120,90,60,30
        dc.b 0,-30,-60,-90
        dc.b -120,-90,-60,-30
```

Anmerkung: Das Interrupt-Register kann so programmiert werden, daß immer ein Interrupt ausgelöst wird, wenn der Soundchip neue Daten benötigt.

```
audloc  = $dff0a0                ;wie im ersten Programm
audlen  = $dff0a4
audper  = $dff0a6
audvol  = $dff0a8
auddat  = $dff0aa                ;Register f. Sounddaten
intenaw = $dff09a                ;Interrupt-Enable
intreqr = $dff01e                ;Interrupt-Request(les.)
intreqw = $dff09c                ;Int-Request(schreiben)

start:  MOVE #64,audvol          ;wie im ersten Programm
        MOVE #224,audper
        MOVE #$4000,intenaw      ;verhindere Interrupts
loop0:  LEA dreieck,A0           ;Basisadr. der Daten
        MOVEQ #7,D0              ;setze Zähler für Dat.
loop1:  MOVE (A0)+,auddat        ;Datum ins Register
loop2:  ANDI #$80,intreqr        ;lese I-Request aus
        BEQ.s loop2              ;verz.,wenn Datum noch
                                 ;gebraucht wird
        MOVE #$80,intreqw        ;lösche Requestbit
        DBF D0,loop1             ;hole nächstes Datum
        BTST #6,$bfe001          ;teste linke Maustaste
        BNE.s loop0              ;Abbruch, wenn gedrückt
        MOVE #$c000,intenaw      ;Interrupts freigeben
        RTS                      ;Rücksprung

dreieck:dc.b 0,30,60,90          ;Sounddaten
        dc.b 120,90,60,30
        dc.b 0,-30,-60,-90
        dc.b -120,-90,-60,-30
```

4.2.3 Beschreibung der Grafik-Hardware

In diesem Teil soll gezeigt werden, was die Grafik-Hardware alles zu leisten vermag und mit welchen Aufwand dies verbunden ist. Wir werden kein komplettes Programm abdrucken, sondern nur das grundlegende Vorgehen beschreiben. Der Grund dafür ist die Tatsache, daß die direkte Grafikprogrammierung ohne Verwendung von Betriebssystemroutinen zu aufwendig und auch nicht wesentlich schneller wäre. Sie werden dies sicher

noch im Laufe der Ausführungen merken. Falls Sie jedoch die Absicht haben sollten, Grafik auf dem Amiga zu programmieren, so empfehlen wir Ihnen das Buch 'Supergrafik Amiga' aus dem Hause DATA BECKER.

Wie Sie bereits in Kapitel 4.2.1 erfahren haben, verfügt der Amiga über vier verschiedene Grafikauflösungen und zahlreiche Betriebsarten. Diese werden über drei Kontrollregister eingestellt. Da die beiden letzten Register nur für eine bestimmte Betriebsart (Dual-Playfield-Mode) verwendet werden, schlüsseln wir nur das erste Kontrollregister genau auf.

Adresse: $DFF100

BIT	BPLCON0
15	Hires
14	BPU2 (Bit Plane Anzahl)
13	BPU1
12	BPU0
11	HAM (Hold-and-Modify)
10	DPF (Dual-Playfield-Mode)
09	COLOR (*)
08	Genlock (*)
07	unbenutzt
06	unbenutzt
05	unbenutzt
04	unbenutzt
03	Lightpen (*)
02	Interlace
01	ext. Synchronisation (*)
00	unbenutzt

(*) : werden bei der Grafikprogrammierung nicht benötigt

Unter BPU 0-2 wird die Anzahl der für die Grafik benötigten Grafikspeicher (Bitplanes) angegeben. Wenn Sie zum Beispiel eine zweifarbige Grafik darstellen wollen, brauchen Sie für jeden einzelnen Grafikpunkt ein Bit, bei vier Farben sind es schon 2 Bits. Der Grafikprozessor holt sich aus diesen Bitplanes die Bitmuster, liest das entsprechende Farbregister aus und stellt schließlich den Punkt auf dem Bildschirm dar.

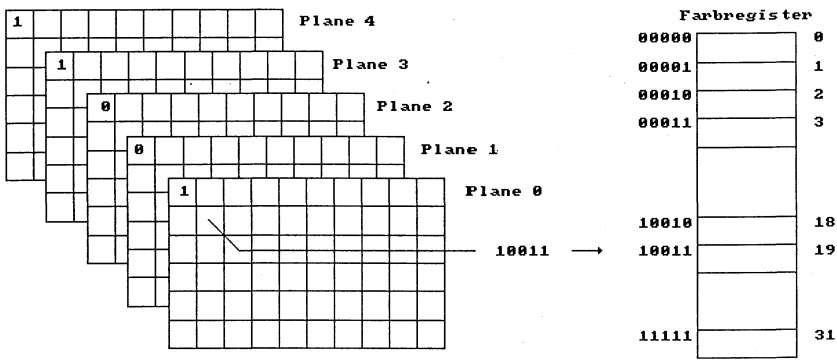

Abbildung 4.2: Zugriff auf die Bitplane-Daten

Je nach Betriebsart und Grafikauflösung werden bis zu sechs Bitplanes benötigt.

Die Bits HIRES, INTERLACE, DBF und HAM sind für die Auswahl der Betriebsarten und Grafikauflösungen zuständig. Über HIRES läßt sich die horizontale Auflösung auf 320 Punkte (niedrige Auflösung) oder 640 Punkte (hohe Auflösung) pro Zeile einstellen. Für Spiele wird meistens die niedrige Auflösung gewählt, wogegen beispielsweise Textverarbeitungsprogramme ausschließlich in der hohen Auflösung arbeiten. Das entsprechende Bit für die vertikale Auflösung ist INTERLACE. Im INTERLACE-Modus können über einen Trick 512 Zeilen dargestellt werden. Das Grafikbild wird dazu in zwei Halbbilber zerlegt, die hintereinander mit einer leichten Verschiebung dargestellt werden. Die PAL-Fernsehnorm bedient sich übriges des gleichen Tricks. DBF, eine weitere Betriebsart, ermöglicht es, zwei unabhängige Grafikbilder gleichzeitig darzustellen. Im

Kontrollregister 2 existiert ein Bit, das festlegt, welches Grafikbild im Vordergrund steht. Durch die Möglichkeit, verschieden große Grafiken darzustellen, können beide Grafiken gleichzeitig auf dem Bildschirm sichtbar sein. Dies kann man sich in Spielen zunutze machen; so kann ein Objekt ein anderes teilweise verdecken, ohne daß irgendwelche Daten im Grafikspeicher umkopiert werden müßten. Die letzte Betriebsart, der HAM-Modus, erlaubt die Verwendung von bis zu 4096 Farben in einem Bild. Wie das möglich ist? Es werden ingesamt sechs Bitplanes benötigt, zwei davon als Status-Planes, die restlichen Planes fungieren wieder als Farbspeicher. Nun kann in Abhängigkeit vom Inhalt der Status-Planes von einem Punkt zum anderen entweder einer der drei Farbanteile geändert werden (die restlichen Farbanteile bleiben dabei unverändert), oder es wird die gesamte Farbe geändert.

Nachdem wir uns nun mit den Grafikauflösungen und Betriebsarten beschäftigt haben, wollen wir zeigen, welche Schritte nötig sind, um eine Grafik darzustellen:

1. Übergabe der Grafikauflösung und der Betriebsart
2. Übergabe der zu verwendenden Farben
3. Übergabe des Bildschirmformats
4. Speicher für die Bitplanes reservieren
5. Copperprogramm erstellen (übergibt Bitplane-Zeiger)
6. Zeiger auf Copperprogramm übergeben
7. Copper-DMA einschalten

Zu den einzelnen Punkten:

Punkt 1: Die Grafikauflösung und die Betriebsart wird wie schon erwähnt im Kontrollregister 0 übergeben.

Punkt 2: Alle im Grafikbild vorkommenden Farben müssen in den Farbregistern gespeichert werden. Dabei ist immer beim ersten Register zu beginnen.

Punkt 3: Zusätzlich zum Grafikformat muß noch das Bildschirmformat angegeben werden. Es besteht nämlich die Möglichkeit, nur einen Ausschnitt aus einer Grafik darzustellen.

```
0 1 1 0 1 0 0 1 0 1 0 1 1 1 0 0 1 0
1 0 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1
1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0
0 1 1 0 1 0 0 1 0 1 0 1 1 1 0 0 1 0
1 0 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1
1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0
0 1 1 0 1 0 0 1 0 1 0 1 1 1 0 0 1 0
1 0 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1
1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0
0 1 1 0 1 0 0 1 0 1 0 1 1 1 0 0 1 0
1 0 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1
1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0
```

Abbildung 4.3: Playfield

Zu diesem Zweck existieren zwei Register, BPL1MOD und BPL2MOD, das eine für Bitplanes mit ungeraden Nummern und das andere für solche mit geraden. Die Inhalte dieser Register werden vom Grafikprozessor nach jeder Zeile automatisch zum internen Zeiger auf die Grafikdaten addiert, so daß in der nächsten Zeile der Teil der Daten, der nicht ausgegeben werden soll, übersprungen wird.

Es existieren auch noch weitere Register, mit denen sich die Grafik auf dem Bildschirm positionieren läßt und die in Grenzen eine Veränderung des Bildschirmformats erlauben.

Punkt 4: Der Speicher für die Bitplanes sollte nach Möglichkeit absolut reserviert werden, da die Bitplane-Zeiger mit Hilfe des Coppers unmittelbar übertragen werden müssen. Das Betriebssystem hält dafür eine spezielle Routine mit Namen AllocAbs bereit. Eine LORES-Bitplane umfaßt normalerweise 8000 Byte, eine HIRES-Bitplane sogar 16000.

Punkt 5: Mit Hilfe des Coppers müssen, wie schon gesagt, die Bitplane-Zeiger übergeben werden, da diese nach jedem Bildschirmaufbau wieder gelöscht werden. Die Zeiger können über den MOVE-Befehl des Coppers in die entsprechenden Register übertragen werden. Das Copper-Programm sollte mit der Datenfolge $ffff,$fffe abgeschlossen werden.

Syntax des MOVE-Befehls:

```
1. WORT
Bit 9-15 unbenutzt
Bit 1-8  Adreßbits 1-8 des Zielregisters
bit 0    immer Null

2. WORT
Bit 0-15 zu übertragendes Datenwort
```

Punkt 6: In die Register COP1LCH und COP1LCL wird die Startadresse des Copper-Programms übergeben. Der Copper springt automatisch nach jedem Bildschirmaufbau auf die in diesen Registern abgelegte Adresse.

Punkt 7: Nachdem das Copper-Programm sich im Speicher (unteren 512 KByte) befindet und die Startadresse übergeben wurde, kann die Copper-DMA über das DMA-Kontrollregister gestartet werden.

Sie sehen nun, welch ein aufwendiges Unterfangen die direkte Grafikprogrammierung ist. Die Benutzung der Betriebssystemroutinen macht die Programmierung wesentlich unkomplizierter. Wie einfach beispielsweise ein Window mit Hilfe von Intuition-Routinen erzeugt werden kann, sahen Sie in Kapitel 4.1.6.

4.2.4 Sounderzeugung mit dem Amiga

Wie schon erwähnt, ist die Paula unter anderem für die Sound-Erzeugung des Amiga verantwortlich. Im Gegensatz zum SID im C64 sind keine Wellenformen (Rechteck, Dreieck, usw.) hardwaremäßig vorgegeben, sondern müssen erst als Sounddaten im Speicher abgelegt werden. Die Sounddaten können mit einem Analog-Digital-Wandler aufgezeichnet werden. Dabei wird mit Hilfe des Wandlers in gleichen Abständen das analoge Signal (z.B. von einem Plattenspieler) abgetastet und in eine digitale 8-Bit-Information umgesetzt, die dann, im Speicher untergebracht, von der Paula wieder in ein analoges Signal zurückgeführt wird. Die Daten werden wie folgt abgetastet:

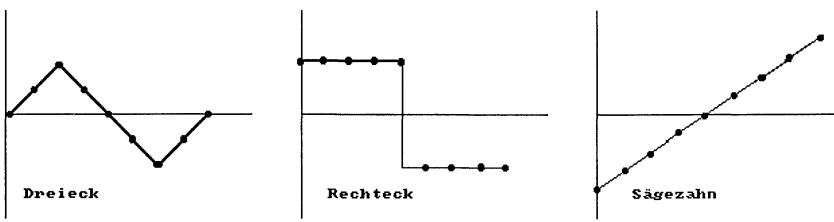

Abbildung 4.4: Sounddaten

Danach können Sie die Daten Byte für Byte in den Speicher übertragen.

Die Paula kann wahlweise die Sounddaten per DMA einlesen oder durch ein Register, das immer mit den Daten versorgt werden muß. Da die zweite Methode zuviel Rechenzeit beansprucht, befassen wir uns nur mit dem DMA-Betrieb.

Zur Sound-Erzeugung werden folgende Schritte benötigt:

1. Wähle einen der vier Audiokanäle.
2. Lege die Sounddaten im Speicher ab.
3. Übergebe Startadresse und Länge der Daten

4. Setze die Lautstärke.
5. Gebe die Ablaufgeschwindigkeit an.
6. Starte die DMA.

Da der Amiga über vier unabhängige Audiokanäle verfügt, müssen auch alle notwendigen Register in vierfacher Ausführung vorhanden sein.

Diese Register benötigt man bei DMA-Betrieb:

Name	Adresse	Bedeutung
AUDxLCH	$dff0a0+16*x	Startadr. der Sounddaten (Bit 16-18)
AUDxLCL	$dff0a2+16+x	Startadr. der Sounddaten (Bit 0-15)
AUDxLEN	$dff0a4+16+x	Länge der Sounddaten in Worten
AUDxPER	$dff0a6+16*x	Ausgabegeschwindigkeit
AUDxVOL	$dff0a8+16*x	Lautstärke (x ist die Nummer eines der vier Kanäle, begonnen bei Null.)
DMACONW	$dff096	steuert den DMA-Betrieb

Bit	Name	Bedeutung
15	setclr	bei 1: Setzt die korrespondierenden Maskenbits. Die anderen bleiben unberührt.bei 0: Löscht die korrespondierenden Maskenbits. Die anderen bleiben unberührt.
9	dmaen	DMA-Betrieb nur bei gesetzten Bit möglich
3	aud3en	erlaubt DMA für Audiokanal 3
2	aud2en	erlaubt DMA für AudioKanal 2
1	aud1en	erlaubt DMA für Audiokanal 1
0	aud0en	erlaubt DMA für AudioKanal 0

Mit diesen Informationen ist es möglich, eine kleine Sound-Demo zu schreiben. Damit Sie auch Ihre eigenen Kompositionen abspielen können, hier zwei Umrechnungstabellen:

Note	Frequenz	Period (in notper einzusetzen, s. u.)
C	523.3	428
C#	554.4	404
D	587.3	381
D#	622.3	360
E	659.3	339
F	698.5	320
F#	740.0	302
G	784.0	285
G#	830.6	269
A	880.0	254
A#	932.3	240
B	987.8	226
C	1046.5	214

Notenwert	Schleifenwert (in notlen einzus., s. u.)
Ganze Note	16
Halbe Note	8
Dreiviertel Note	6
Viertel Note	4
Dreiachtel Note	3
Achtel Note	2
Sechzehntel Note	1

Zum Abschluß nun die versprochene Sound-Demo. Die oben verwendeten Bezeichnungen notper und notlen beziehen sich auf die beiden Tabellen am Ende des Programms. Das Stück wird auf dem Audiokanal 0 wiedergegeben.

```
audloc  = $dff0a0                ;Startadr. der Sound-Daten
audlen  = $dff0a4                ;Länge der Sound-Daten
audper  = $dff0a6                ;Abspielgeschwindigkeit
audvol  = $dff0a8                ;Lautstärke
dmaconw = $dff096                ;DMA-Kontrollregister

start:    MOVE.l #dreieck,audloc ;überg. Startadr. der Daten
          MOVE #4,audlen         ;überg. Datenanz. in Worten
          MOVE #64,audvol        ;überg. Lautstärke
          LEA notper,A0          ;Startadr. von notper n. A0
          LEA notlen,A1          ;Startadr. von notlen n. A1
          MOVE (A0),audper       ;überg. erste Freq.(d. Note)
          MOVE #$8201,dmaconw    ;schalte Audio-DMA ein
          MOVE #13,D1            ;setze Zähler (14 Noten)
```

```
note:     MOVE (A0)+,audper        ;übergebe Freq. u. erhöhe A0
          CLR.l D0                 ;lösche D0
          MOVE.b (A1)+,D0          ;hole Notenlänge nach D0
          MULU #30000,D0           ;wandle in Schleifenwert
hold:     SUBQ.l #1,D0             ;erniedr. Schleifenwert um 1
          BNE.s hold               ;halte Note bis D0 gleich 0
          DBF D1,note              ;spiele nächste Note
          MOVE #$0001,dmaconw      ;schalte Audio-DMA ab
          RTS                      ;Rücksprung

dreieck:  dc.b 0,60,120,60         ;Daten der Wellenform
          dc.b 0,-60,-120,-60

notper :  dc.w 428,339,285,214,254 ;Daten entsprechen der
          dc.w 214,254,285,320,285 ;Freq. der Noten
          dc.w 339,428,381,428

notlen :  dc.b 6,3,4,4,4,2,2       ;Daten geben die
          dc.b 8,6,2,4,4,8,8       ;Notenlänge an
```

5. Die Alternative zu BASIC und Maschinensprache: C

Die Programmiersprache C ist auf dem Amiga genauso populär wie zum Beispiel BASIC oder Assembler. Das liegt sicherlich nicht zuletzt daran, daß das Betriebssystem des Amiga größtenteils in dieser Sprache geschrieben worden ist. Im nun folgenden Kapitel erfahren Sie, um was für eine Programmiersprache es sich dabei handelt.

5.1 Einführung

C ist eine Sprache, die gerade Ex-C64/C128-Anwendern einiges Umdenken abverlangt. Auf diesen Rechnern wurden und werden 99 Prozent aller Programme in BASIC oder Maschinensprache geschrieben. Dies liegt an zwei Gründen. Höhere Programmiersprachen sind auf Rechnern wie zum Beispiel dem C64 viel zu langsam, als daß man damit professionelle Programme schreiben könnte. Deshalb gibt es nahezu kein kommerziell vertriebenes Programm, daß nicht in Assembler entwickelt wird. BASIC hat sich aber trotzdem daneben halten können, weil ein BASIC-Interpreter fest in das Gerät eingebaut ist und weil BASIC wesentlich einfacher zu lernen ist als Maschinensprache.

C ist eine Compiler-Sprache. Deshalb müssen C-Programme nach ihrer Erstellung zuerst mit einem C-Compiler in Maschinensprache übersetzt werden. Auch vom C64 kennen Sie sicherlich diverse Compiler, die aber alle keine Programme erzeugen, die sich auch nur im entferntesten mit von Hand geschriebenen Maschinenprogrammen messen können. C-Programme dagegen haben eine Ablaufgeschwindigkeit, die nur um den Faktor zwei bis drei unter der von 68000-Maschinenprogrammen liegt. Damit sind sie drei bis viermal schneller als 6510- bzw. 8502-Programme, was schon deutlich macht, daß sich C selbst für zeitkritische Anwendungen eignet.

Ein anderes Problem beim C64 ist der Speicherplatz. Assembler-Programme kommen fast immer mit sehr viel weniger Bytes aus

als vergleichbare Programme in BASIC. Dies trifft zwar auch für C-Programme im Vergleich zur 68000-Maschinensprache zu, jedoch hat man ja auf dem Amiga wesentlich mehr Speicherplatz zur Verfügung als auf seinem alten Rechner.

C ist also eine Sprache, die ohne weiteres für professionelle Programme verwendet werden kann, was allein schon die Tatsache beweist, daß sie zum Entwurf des Betriebssystems benutzt wurde. Trotz der Tatsache, daß es sich um eine vollwertige höhere Programmiersprache handelt, mit verschiedenen Datentypen, Namensbenennungen oder Unterprogrammen mit Übergabeparametern, ist sie leicht in Maschinensprache zu übersetzen, was sich in der hohen Ausführungsgeschwindigkeit niederschlägt.

C hat aber auch durch ihr Konzept einige Nachteile, die zweifellos an die Nachteile von Maschinensprache erinnern. Dies betrifft insbesondere die Art, wie man Betriebssystemroutinen verwenden muß. Um beispielsweise in BASIC ein neues Fenster zu öffnen, genügt ein einfacher Befehl. In C dagegen muß man erst mit einer ganzen Menge von Daten herumjonglieren, bevor das Fenster erscheint. Das bietet andererseits aber auch den Vorteil, daß man nicht in seinen Möglichkeiten eingeengt wird. Versuchen Sie doch einmal in BASIC, in Menüs Untermenüs einzubauen, also die Menütafeln, die bei manchen Programmen einzelnen Menüpunkten zugeordnet sind und nur dann erscheinen, wenn man gerade mit dem Mauszeiger auf den Menüpunkt deutet. In C ist das nicht schwerer als ein normales Menü einzubauen, in BASIC dagegen fast ein Ding der Unmöglichkeit. Um diesen Unterschied aber richtig zu verstehen, ist es am besten, wenn Sie sich einmal ein C-Programm ansehen.

Die nun folgende Einführung in C ist nicht als vollwertiger Programmierkurs gedacht, sondern sie vermittelt Ihnen einen Vorgeschmack von dem, was Sie erwartet, wenn Sie eigene C-Programme entwerfen. Sie sollten also schon etwas in BASIC, besser sogar noch in Pascal, programmieren können, um die Erklärungen vollständig zu verstehen. Mit etwas Programmiererfahrung werden Sie aber nach der Lektüre dieses Kapitels die wichtigsten Grundzüge von C verstanden haben, so daß Sie in

der Lage sind, einfache Programme zu entwickeln. Wenn Ihnen die Zusammenfassung der Sprache hier nicht genügt, dann werfen Sie doch einmal einen Blick in das DATA-Becker-Buch C für Einsteiger. Hier finden Sie sicherlich alles, was Sie benötigen.

5.2 Das erste C-Programm

Das erste C-Programm, mit dem jeder, der die Sprache neu lernt, konfrontiert wird, sieht fast immer ungefähr so aus:

```
main()
{
   printf("Data Becker!\n");
}
```

Wie Sie dieses Programm eingeben, übersetzen lassen und starten, hängt natürlich davon ab, welchen C-Compiler Sie besitzen. Diese Informationen entnehmen Sie am besten Ihrem Compiler-Handbuch.

Wie Sie sich sicher schon denken können, druckt dieses Programm den Text "Data Becker!" auf dem Bildschirm aus. Jedes C-Programm besteht immer aus einer Reihe von Funktionen. Wenn man in C von Funktionen spricht, so meint man damit Unterprogramme, also Befehlsgruppen, die unter einem Namen zusammengefaßt werden. Jedes C-Programm enthält eine Funktion mit dem Namen main. Sie stellt das eigentliche Hauptprogramm dar.

Jede Funktionsdefinition sieht so aus: Zuerst kommt der Funktionsname, hier also main, und dahinter in runden Klammern eventuelle Übergabeparameter, auf die wir aber erst später eingehen werden. In der Zeile main() beginnt also eine Funktionsdefinition ohne irgendwelche Parameter.

Als nächstes folgt im obigen Beispielprogramm eine geschweifte Klammer {, die anzeigt, daß nun die Befehle der Funktion fol-

gen. Geschweifte Klammern werden in C grundsätzlich zum Zusammenfassen von Befehlsfolgen verwendet, und wir werden sie noch öfter benötigen.

Die einzige Anweisung in dem Beispielprogramm lautet printf(...). Dieser Befehl druckt auf dem Bildschirm Texte oder Variablen aus und entspricht somit dem Print-Befehl von BASIC. Das nach dem Printf-Befehl stehende Semikolon muß nach jedem Befehl stehen, damit der Compiler weiß, wo ein Befehl zu Ende ist. Am Schluß des Programms wird die Befehlsfolge mit der geschweiften Klammer } abgeschlossen.

Neben dem normalen Text enthält der obige Beispieltext auch noch die Zeichenkombination \n. Diese steht für das Linefeed-Zeichen (in BASIC: CHR$(10)). Dadurch wird der Ausgabe-Cursor nach der Anzeige von Data Becker an den Anfang der nächsten Zeile gesetzt. C kennt noch mehr solche mit \ eingeleitete Zeichen. \b zum Beispiel steht für Backspace, \f für Formfeed (Bildschirm löschen),\t für das Tabulator-Zeichen, \0 für das Null-Zeichen (in BASIC: CHR$(0)) und \\ für das \-Zeichen selbst.

Wenn man \n am Ende des Textes wegläßt, bleibt der Cursor hinter dem Text stehen. Das bedeutet, daß folgende Befehlsfolge äquivalent zu dem Befehl aus dem Beispiel ist:

```
printf("Data ");
printf("Becker!\n");
```

5.3 Variablen und Datentypen

Die Verwendung von Variablen soll folgendes Beispiel demonstrieren:

```
/* Bildung der Summe zweier Zahlen */
main()
{
  int wert1,wert2,summe;
```

```
    wert1=1234;
    wert2=5678;
    summe=wert1+wert2;

    printf("Die Summe von %d und %d ",wert1,wert2);
    printf("ist %d.\n",summe);
}
```

Was bei diesem Programm auffällt, ist der in /* und */ eingeklammerte Kommentar. Kommentare werden vom Compiler ignoriert. Sie können über mehrere Zeilen gehen und sollten bei keinem guten Programm fehlen.

Variablen müssen genauso wie zum Beispiel in Pascal vor ihrer Verwendung definiert werden. Mit int wert1,wert2,summe; schafft man sich drei Ganzzahl-Variablen. Solche Vereinbarungen müssen vor allen anderen Befehlen in einer Funktion stehen. Wenn man eine Variable zu definieren vergißt, beschwert sich der Compiler bei der Übersetzung.

Wie Sie sehen, erfolgen Zuweisungen genau wie in BASIC mit dem Gleichheitszeichen. Die Variable auf der linken Seite erhält den Wert des Ausdrucks auf der rechten Seite. Einfache Ausdrücke sind übrigens fast genauso aufgebaut wie in BASIC. Die Grundrechenarten werden durch die Symbole +, -, * und / vertreten. Zusätzlich gibt es den Operator %, der den Rest einer Division bestimmt.

Bei der Ausgabe der Werte wird eine Besonderheit der Printf-Anweisung deutlich. Die Variablen stehen nämlich nicht an der Stelle, an der ihre Werte beim Ausdruck erscheinen. Statt dessen enthält die Zeichenkette sogenannte Platzhalter, hier %d. Sie geben den Datentyp und das Ausdruckformat der nachfolgenden Variablen an. Bei der Ausführung werden die Platzhalter durch die Variablenwerte in der Reihenfolge ersetzt, in der die Variablen nachfolgen.

Neben dem Datentyp int kennt C auch noch folgende:

char	ein einzelnes Zeichen, 8 Bit
short	eine kurze Ganzzahl, beim Amiga: 16 Bit
long	eine lange Ganzzahl, beim Amiga: 32 Bit
float	eine Fließkommazahl in einfacher Genauigkeit
double	eine Fließkommazahl in doppelter Genauigkeit

Der Typ int entspricht entweder short oder long, je nachdem, welchen Compiler mit welchen Voreinstellungen Sie verwenden.

Vor die Typen char, int, short und long kann noch das Schlüsselwort unsigned gestellt werden. Normalerweise können Ganzzahlen sowohl positive als auch negative Werte annehmen. Durch unsigned wird der darstellbare Zahlenbereich ins Positive verschoben. Das bedeutet, daß eine Variable vom Typ short Werte von -32768 bis 32767 annehmen darf, während eine Variable vom Typ unsigned short Werte von 0 bis 65535 annehmen kann. Char-Variablen können in Rechnungen beliebig mit Ganzzahl-Variablen gemischt werden. Der Compiler benutzt dann ihren ASCII-Code. Hierfür ist wichtig, ob die Variable als char oder unsigned char vereinbart wurde. Im ersten Fall liegt der mögliche Zahlenbereich bei -128 bis 127, im zweiten Fall bei 0 bis 255.

Für die unterschiedlichen Daten-Typen werden unterschiedliche Platzhalter bei der Printf-Anweisung benötigt. Nicht alle Möglichkeiten der folgenden Liste werden von jedem Compiler unterstützt. Prüfen Sie also selbst nach, was bei Ihnen erlaubt ist und was nicht.

%d	Eine Ganzzahl wird im Dezimalsystem ausgegeben.
%o	Eine Ganzzahl wird als Oktalzahl (zur Basis 8) ausgegeben.
%x	Eine Ganzzahl wird als Hexadezimalzahl (zur Basis 16)ausgegeben.
%c	Das Zeichen einer Char-Variable wird ausgegeben.

%s Eine Zeichenkette wird ausgegeben (siehe Kap 5.7)
%f Eine Variable vom Typ "float" oder "double" wird ausgegeben.
%% Das Prozentzeichen "%" wird ausgegeben.

Weiterhin können zwischen dem %-Zeichen und dem Buchstaben Werte stehen, die verschiedene Ausgabeformatierungen ermöglichen. Einige Beispiele:

a) printf("%8d",a);

b) printf("%-12.6f",b);

c) printf("%ld",c);

In Beispiel a: Wird die Zahl grundsätzlich mit 8 Stellen ausgegeben. Ist sie kleiner, so wird mit Leerzeichen aufgefüllt.

In Beispiel b: Demonstriert die Ausgabe einer Fließkommazahl. Die Gesamtzahl der Stellen inklusive dem Dezimalpunkt beträgt 12, die Anzahl der Nachkommastellen 6. Das Minuszeichen vor der 12 besagt, daß die Zahl linksbündig ausgegeben wird.

In Beispiel c: Gibt eine Long-Zahl aus. Das l wird beispielsweise dann benötigt, wenn man in einer Long-Variablen den ASCII-Code eines Zeichens gespeichert hat und nun das zugehörige Zeichen ausdrucken will. Wie Sie merken, ist das Mischen diverser Datentypen in C oft möglich. Das nächste Programm zeigt einige Beispiele dafür:

```
main()
{
   unsigned char c;
   short s;
   c='a';
   printf("\n%c\n%d\n",c,c);
   c='A'-'a'+10;
   s=c;
   c='\n';
}
```

Wie Sie sehen, werden einzelne Zeichen in einfache Hochkommas ' gestellt (bei deutscher Tastatur: ALT + Ä), während die Zeichenkette beim Printf-Befehl mit Anführungszeichen umgeben ist. Es wird auch deutlich, daß '\n' nicht zwei, sondern nur ein einziges Zeichen meint. Der Printf-Befehl in dem Programm erklärt, warum der Datentyp der auszugebenden Variablen unbedingt noch einmal im Ausgabestring angegeben werden muß. Die Char-Variable c wird nämlich einmal als Zeichen-Variable und einmal als Ganzzahl-Variable ausgegeben.

In manchen Fällen, insbesondere beim Aufruf von Betriebssystemfunktionen, ist es erforderlich, dem Compiler klar zu machen, daß eine Zahl einen Long-Wert darstellt. Dazu stellt man ihr den Buchstaben L nach. 1234L wird vom Compiler also etwas anders behandelt als 1234. Mit der Konstruktion (Datentyp)Ausdruck erzielt man derartige Umwandlungen auch bei Variablen oder Ausdrücken. Die Konstruktion (double)n ist also durchaus sinnvoll.

5.4 Schleifen und Verzweigungen

C kennt komfortable Befehle zur Bildung von Schleifen. Hierzu drei Beispielprogramme, die alle das gleiche machen:

```
/* Programm 1 */
main()
{
   long a;
   short i;
```

```
   a=1;
   i=1;
   while (i<20) {
      a=a*2;
      printf("2 hoch%3d = %-ld\n",i,a);
      i=i+1;
   }
}


/* Programm 2 */
main()
{
   long a;
   short i;

   a=1;
   i=1;
   do {
      a=a*2;
      printf("2 hoch%3d = %-ld\n",i,a);
      i=i+1;
   } while (i<20);
}


/* Programm 3 */
main()
{
   long a;
   short i;

   a=1;
   for(i=1;i<20;i=i+1) {
      a=a*2;
      printf("2 hoch%3d = %-ld\n",i,a);
   }
}
```

Die drei Programme erzeugen den Ausdruck der Potenzen von zwei, angefangen bei zwei hoch 1 bis zwei hoch 19. Programm 1 demonstriert eine While-Schleife. Sie sieht immer so aus:

```
while (Bedingung)
  Befehlsblock
```

Solange die Bedingung erfüllt ist, wird der Befehlsblock wiederholt. Ein Befehlsblock ist entweder ein einzelner Befehl, abgeschlossen mit dem Semikolon, oder eine Gruppe von Befehlen, die von geschweiften Klammern {} umgeben sind.

Bedingungen in C lassen sich fast so schreiben wie in BASIC oder Pascal. Die Vergleichszeichen größer >, kleiner <, größer oder gleich >= und kleiner oder gleich <= sollten Ihnen daher ein Begriff sein. Gleichheit wird mit == abgetestet, um den Unterschied zur Zuweisung = deutlich zu machen. Ein Test auf Ungleichheit wird dagegen mit != durchgeführt. Das logische und wird durch den Operator && vertreten, das logische oder durch || und das logische nicht durch !. In BASIC und Pascal sind das die Befehle AND, OR und NOT.

Genaugenommen liefert ein Vergleich einen Wert, nämlich -1 für wahr und 0 für falsch. Alle Befehle, die mit Bedingungen arbeiten, berechnen einfach den Wert des Bedingungsausdrucks und testen, ob er gleich 0 ist oder nicht. Falls ja, wird die Bedingung als nicht erfüllt angesehen (falsch), anderenfalls als erfüllt (wahr). Daher kann anstelle eines Vergleichs auch eine Variable oder irgendein anderer Ausdruck als Bedingung angegeben werden, dessen Wert vom Compiler auf Null getestet wird.

Noch ein Wort zur Anordnung von Befehlen und Klammern im Text. Dem Compiler ist es zwar im Prinzip egal, ob man seine Befehle in irgendeiner Weise einrückt oder in welcher Zeile eine geschweifte Klammer steht. Sie sollten aber aus Gründen der Übersichtlichkeit immer untergeordnete Befehlsblöcke einrücken. Gewöhnen Sie sich am besten eine einheitliche Schreibweise an, die Ihnen am sinnvollsten erscheint. Dadurch wird die Lesbarkeit der Programme erheblich gesteigert, und Sie selbst erleichtern sich die Dokumentation und die Fehlersuche.

Beispielprogramm 1 testet also den Wert von i und führt den Befehlsblock aus, solange i kleiner als 20 ist. Im Befehlsblock wird die Variable a mit zwei multipliziert, und i wird um eins erhöht. Die Bedingung wird am Anfang der Schleife überprüft. Bei der While-Schleife kann es demnach vorkommen, daß der Befehlsblock überhaupt nicht ausgeführt wird.

Im zweiten Beispiel wird eine Do...While-Schleife verwendet. Sie ist folgendermaßen aufgebaut:

```
do
  Befehlsblock
while (Bedingung);
```

Diese Form der Schleife wird grundsätzlich mindestens einmal ausgeführt, da die Bedingung immer erst nach Abarbeiten des Blocks getestet wird. Ansonsten gilt aber das gleiche, was schon zur While-Schleife gesagt wurde.

Das dritte Programm schließlich demonstriert eine For-Schleife:

```
for (Initialisierung;Wiederholbedingung;Reinitialisierung)
  Befehlsblock
```

In der Initialisierung erhält eine Zählvariable einen Startwert. Dann wird die Wiederholbedingung überprüft. Hier sollte normalerweise ein Test des Zählers erfolgen. In der Reinitialisierung nimmt man dann üblicherweise das Ändern des Zählers vor. Eine For-Schleife läßt sich auch durch eine While-Schleife ersetzen:

```
Initialisierung;
while (Wiederholbedingung) {
   Befehl1;
   Befehl2;
   ...
   Reinitialisierung;
}
```

Wie Sie sehen, ist die For-Anweisung von C wesentlich flexibler als zum Beispiel die For-Anweisungen aus BASIC oder Pascal. Man sollte sie immer dann der While-Anweisung vorziehen, wenn man eine Schleife mit einem Zähler verwendet.

Gerade in Verbindung mit Schleifen lassen sich einige Vereinfachungen demonstrieren, die die Sprache C ermöglicht. Zuweisungen wie i = i+x; kommen sehr häufig vor und können statt dessen auch so formuliert werden:

```
i += x;
```

Genauso kann auch mit allen anderen Grundrechenarten verfahren werden, das heißt, es gibt entsprechende Zuweisungsoperatoren -=, *= und /=. Die For-Anweisung aus Beispielprogramm 3 läßt sich demnach auch folgendermaßen schreiben:

```
for (i=1;i<20;i+=1)
```

Das Erhöhen oder Verringern einer Variable um den Wert eins läßt sich sogar noch weiter vereinfachen. Statt i+=1 schreibt man entweder ++i oder i++, statt i-=1 läßt sich --i oder i-- verwenden. Aus der obigen For-Anweisung wird dann dieser Ausdruck:

```
for (i=1;i<20;++i)
```

Was es mit dieser Art der Zuweisung auf sich hat, und was der Unterschied zwischen ++i und i++ ist, wird im nächsten Kapitel erklärt. Zunächst aber wollen wir auf die bedingte Verzweigung, also die If-Anweisung, eingehen. Auch hierzu wieder ein Beispielprogramm.

```
main()
{
   short i;
   for (i=1;i<=20;++i)
      if (i % 2 ==0)
         printf("%d\n",i);
      else
         printf("%d   ",i);
}
```

Mit der If-Anweisung in diesem Programm wird getestet, ob die Variable i bei der Divison durch zwei den Rest Null liefert, also

ob i gerade ist. Falls ja, wird die Zahl ausgegeben und der Cursor an den Anfang der nächsten Zeile gesetzt. Anderenfalls wird die Zahl mit drei nachfolgenden Leerzeichen gedruckt.

Eine If-Anweisung hat demnach allgemein folgenden Aufbau:

```
if (Bedingung)
  Befehlsblock 1
else
  Befehlsblock 2
```

Der Befehlsblock 1 wird immer dann ausgeführt, wenn die Bedingung erfüllt ist. Wenn das nicht der Fall ist, wird statt dessen Befehlsblock 2 ausgeführt. Der Else-Zweig ist wie bei fast jeder Programmiersprache optional.

Beachten Sie übrigens, daß weder für die If-Anweisung noch für die For-Anweisung im obigen Beispiel geschweifte Klammern benötigt werden. Wie schon gesagt, kann man auf sie immer dann verzichten, wenn der zugehörige Befehlsblock nur aus einem einzigen Befehl besteht. Die If-Anweisung wird dabei zusammen mit dem Else-Teil und allen untergeordneten Befehlen als ein Befehl angesehen.

In Verbindung mit Schleifen kennt C noch zwei weitere Befehle, die aber nur in Verbindung mit der If-Anweisung Sinn haben. Da wäre zunächst der Befehl break, der innerhalb einer Schleife ausgeführt werden muß und sofort die Schleife beendet. Damit wird es möglich, die Wiederholbedingung nicht nur am Anfang oder Ende der Schleife zu testen, sondern auch in der Mitte. Dies betrifft alle drei Schleifenformen. Ein Beispiel:

```
while (1) {
  ++i;
  if (i==100) break;
  --j;
}
```

Sobald i den Wert 100 erreicht hat, wird j nicht mehr dekrementiert. Die Bedingung in der While-Schleife ist immer wahr, also kann die Schleife nur durch den Break-Befehl unterbrochen werden.

Continue funktioniert ähnlich wie break. Hier wird die Schleife sofort mit dem nächsten Durchlauf fortgesetzt. In while und Do...While-Schleifen wird mit dem Test der Bedingung fortgefahren, in For-Schleifen mit der Reinitialisierung.

Eine besondere Form der bedingten Verzweigung stellt die Switch-Anweisung dar. Das folgende Beispiel benutzt die Funktion getchar(), die deshalb zuerst erklärt werden soll. c=getchar(); holt ein Zeichen von der Tastatur und weist es der Variablen c zu. Beim ersten Aufruf von getchar werden solange alle eingetippten Zeichen in einen Zwischenpuffer übertragen, bis die Return-Taste gedrückt wird. Danach wird bei jedem weiteren Aufruf jeweils ein Zeichen aus dem Puffer an das Programm übergeben. Nun aber das Programm:

```
main ()
{
   /* Zeichen zählen */
   /* Dieses Programm erwartet eine beliebige Textzeile
      als Eingabe und zählt die darin vorkommenden Rechen-
      und Leerzeichen. */

   short rechenzeichen,leerzeichen,sonstige;
   unsigned char c;

   rechenzeichen=0;
   leerzeichen=0;
   sonstige=0;
   printf("Text eingeben:\n");
   do {
      c=getchar();
      switch(c) {
         case '+':
         case '-':
         case '*':
         case '/':
            ++rechenzeichen;
            break;
```

```
            case ' ':
               ++leerzeichen;
               break;
            case '\n':
               break;
            default:
               ++sonstige;
               break;
         }
      } while (c!='\n');

      printf("Die Eingabezeile enthält:\n");
      printf("%2d Rechenzeichen\n",rechenzeichen);
      printf("%2d Leerzeichen\n",leerzeichen);
      printf("%2d sonstige Zeichen\n",sonstige);
   }
```

Switch dient zur Auswahl unter mehreren Alternativen. Der hinter switch in Klammern angegebene Wert muß entweder ganzzahlig oder vom Typ char sein, was ja in C auf das gleiche hinausläuft. Hinter den Case-Befehlen stehen deshalb ganzzahlige bzw. Char-Konstanten, die mit dem zuvor berechneten Wert verglichen werden. Sollte Übereinstimmung festgestellt worden sein, so wird die Befehlsausführung hinter der entsprechenden Case-Anweisung fortgesetzt.

Wenn keiner der möglichen Werte angenommen wird, werden die Anweisungen hinter default ausgeführt. Dies entspricht dem Else-Teil bei einer If-Anweisung. Default ist optional und kann genau wie else weggelassen werden.

Im Beispielprogramm oben wird jeweils getestet, ob ein eingegebenes Zeichen in die Kategorie Rechenzeichen oder Leerzeichen fällt. Wenn ja, wird der jeweilige Zähler um eins erhöht. Anderenfalls wird getestet, ob das Zeichen den Wert für die Return-Taste enthält. Für diesen Fall wird überhaupt kein Zähler verändert. Im Default-Abschnitt schließlich wird der Zähler für sonstige Zeichen inkrementiert.

Der jeweils letzte Befehl einer Alternative ist fast immer break;. Dadurch wird der Switch-Block verlassen. Ohne break würden weitere Tests erfolgen, und es würde eventuell noch eine zweite

Alternative gewählt werden. Der allerletzte Break-Befehl ist zwar eigentlich unnötig, jedoch ist es guter Programmierstil, ihn trotzdem einzufügen. Dadurch kann bei späteren Erweiterungen der Fallunterscheidungsliste ein Fehler vermieden werden.

C kennt auch die Goto-Anweisung, also die unbedingte Verzweigung. Diese sollte aber nach Möglichkeit vermieden werden. Schließlich kann jedes Programm auch ohne goto geschrieben werden. Nur in sehr wenigen Fällen bringt goto eine wirkliche Vereinfachung, beispielsweise dann, wenn man aus dem Inneren von zwei geschachtelten Schleifen nach außen springen möchte, was sich nicht mit break bewerkstelligen läßt. Die Syntax von goto sieht so aus:

```
...
goto Sprungmarke;
...
Sprungmarke:
...
```

Wie wir schon zu Anfang angedeutet haben, besteht ein C-Programm aus mehreren Funktionsblöcken, von denen der Main-Block nur einer ist. Goto-Sprünge sind nur innerhalb eines Funktionsblockes erlaubt, und man kann nicht von einem Block in einen anderen überwechseln.

5.5 Zuweisungen und Ausdrücke

Zuweisungen haben wir ja schon in verschiedenen Formen kennengelernt. Da wäre zunächst die allgemeine Zuweisung wie a=b+c. Die spezielleren Formen wie d*=3 oder ++i haben wir ebenfalls schon verwendet. Eine Eigenart von C ist, daß alle Zuweisungen selbst Ausdrücke mit einem Wert sind, mit denen weitergerechnet werden kann. Der Wert einer Zuweisung ist natürlich der zugewiesene Wert. Ein Beispiel für die Möglichkeiten, die sich dadurch ergeben:

Das letzte Beispielprogramm aus Kapitel 5.4 ist im wesentlichen so aufgebaut:

```
do {
   c=getchar();
   ...
} while (c!='\n');
```

Statt einer do...while-Schleife kann man nun eine While-Schleife verwenden:

```
while ((c=getchar())!='\n') {
...
}
```

Zuerst wird die Zuweisung c=getchar() durchgeführt, dann wird der zugewiesene Wert mit '\n' verglichen. Die Schleife wird nicht mehr durchlaufen, wenn c den Wert '\n' angenommen hat. Dies unterscheidet sie auch von der ersten Schleife.

Bei den Operatoren ++ und -- muß unterschieden werden, ob sie vor oder hinter einer Variablen stehen. Angenommen, die Variable i habe den Wert drei. Nach ++i oder i++ hat sie dann den Wert vier. Der Wert des Ausdrucks ++i ist ebenfalls vier, der Wert des Ausdrucks i++ jedoch drei. Bei ++i wird zuerst inkrementiert und dann der Wert weiterverwendet, bei i++ dagegen wird der ursprüngliche Wert von i als Wert der Zuweisung angesehen. Betrachten wir als Beispiel folgende For-Anweisung:

```
for (i=1;i<=100;++i)
    printf("%d\n",i);
```

Hier werden die Zahlen von eins bis einschließlich 100 ausgegeben. Wir können nun die Reinitialisierung weglassen und sie direkt in der Wiederholbedingung unterbringen:

```
for (i=0;++i<=100;)
   printf("%d\n",i);
```

Hier wird zuerst i inkrementiert und das Ergebnis dann mit 100 verglichen. Bei Erreichen des Wertes 100 wird die Schleife das letzte Mal durchlaufen. Beachten Sie auch, daß der Startwert für

i jetzt nicht mehr 1, sondern 0 ist, weil das Inkrementieren jetzt schon vor der Ausführung des Befehlsblockes durchgeführt wird. Wollte man statt dessen den ++-Operator hinter i setzen, so muß die Schleife folgendermaßen aussehen:

```
for (i=0;i++<100;)
  printf("%d\n",i);
```

Wenn in der Wiederholbedingung festgestellt wird, daß i mit dem Wert 99 gerade noch kleiner als 100 ist, so hat i innerhalb des Befehlsblockes trotzdem schon den Wert 100.

Da uns nichts daran hindert, beliebige Ausdrücke in die Reinitialisierung zu schreiben, können wir bei diesem Beispiel auch den Printf-Befehl dort unterbringen:

```
for (i=0;i++<100;printf("%d\n",i));
```

Damit ist der Befehlsblock leer geworden, und der ausführende Teil liegt vollständig innerhalb der runden Klammern. Wie Sie sicherlich selbst feststellen, kann man in C komplexe Ausdrücke sehr knapp formulieren. Dies ist einer der Gründe, warum C-Code sehr effizient in Maschinencode übersetzt werden kann. So etwas kann natürlich stark verwirren, wenn man es zu weit treibt.

Nun aber zu weiteren Möglichkeiten, arithmetische Ausdrücke zu bilden. Ausdrücke bestehen aus Konstanten, Variablen und Operatoren. In C kann man nicht nur mit dezimalen Konstanten arbeiten, sondern auch mit oktalen und hexadezimalen. Oktalen Konstanten wird eine Null vorangestellt, hexadezimalen 0x (Null-X). 0x43 hat also den dezimalen Wert 67. Die oktale Entsprechung lautet 0103.

Klammersetzung und Grundrechenarten arbeiten genauso, wie Sie es schon von BASIC oder Pascal her gewohnt sind. Es gibt aber noch eine Reihe weiterer Operatoren, insbesondere solche, die sich auf die binäre Darstellung der Zahlen beziehen.

&, | Bitweise Und- bzw. Oder-Verknüpfung zweier Werte. Diese Operatoren sollte man nicht mit den logischen Operatoren && und || verwechseln. Die logischen Operatoren unterscheiden nur zwischen null und nicht null und bilden daraus entweder den Wert 0 für falsch oder 1 für wahr. Der Ausdruck 56 & 43 hat zum Beispiel den Wert 40, 56 | 43 den Wert 59. Bitweise sähe das nämlich so aus:

```
111000 & 101011 101000
111000 | 101011 111011
```

^ Bitweise Exklusiv-Oder-Verknüpfung zweier Werte. Auch hier wieder das Beispiel: 56 ^ 43 ergibt 19, da die Verknüpfung im Dualsystem folgendermaßen lauten würde:

```
111000 ^ 101011 010011
```

~ Invertieren der Bits eines Wertes. ~56 hat den Wert 7, da dual ~111000 gleich 000111 ist.

<<, >> a<<b liefert den um b Bits nach links verschobenen Wert von a. a<<1 ist äquivalent zu a*2. >> schiebt dementsprechend die Bits nach rechts.

Zu fast allen Operatoren existieren entsprechende Zuweisungsoperatoren:

```
+=  -=  *=  /=  %=  &=  |=  ^=  <<=  >>=
```

Beispielsweise hat a>>=b+c die gleiche Bedeutung wie a=a>>(b+c).

Eine interessante Möglichkeit, Ausdrücke zu formulieren, ist die bedingte Bewertung. Sie hat allgemein folgenden Aufbau:

```
Bedingung ? Ausdruck1: Ausdruck2
```

Wenn die Bedingung wahr ist, hat der gesamte Ausdruck den Wert von Ausdruck1, anderenfalls den von Ausdruck2. Um zum Beispiel mit dem Maximum zweier Werte a1 und a2 weiterzurechnen, läßt sich folgender Ausdruck formulieren:

```
maximum=(a1>a2)?a1:a2;
```

Die Klammern kann man meistens auch weglassen, sollte es aber aus Gründen der Übersichtlichkeit und Eindeutigkeit nicht tun.

5.6 Datenfelder und Zeiger

Datenfelder, auch Arrays genannt, sind zusammenhängende Gruppen von Variablen gleichen Typs, die sich jeweils nur durch einen Index unterscheiden. Sie werden folgendermaßen vereinbart:

```
Datentyp Feldname [n];
```

Durch diese Anweisung erhält man ein Datenfeld mit n Variablen. Durch short a[100] beispielsweise wird ein Feld mit 100 Short-Variablen definiert. Die einzelnen Variablen lassen sich dann über a[0] bis a[99] ansprechen. Anders als in BASIC existiert das Datum a[100] nicht. Ob die Indexgrenzen eingehalten werden, testet das laufende Programm normalerweise aus Geschwindigkeitsgründen nicht. Deshalb sollte man hier sehr vorsichtig sein. Außerdem muß man sich beim Anlegen großer Felder vor dem Programmstart mit dem CLI-Befehl stack genügend Speicherplatz auf dem Systemstapel reservieren, da C-Programme ihre Variablen dort ablegen. Beispielsweise sollten Sie für ein Datenfeld mit 10000 Bytes den Befehl stack 15000 (5000 Bytes extra zur Sicherheit) aufrufen.

Wenn man in BASIC alle Elemente eines Datenfeldes mit Werten vorbesetzen will, so legt man normalerweise die Werte in DATA-Zeilen ab, um sie dann am Programmanfang mit einer Schleife einzulesen. In C kann man einem Datenfeld direkt bei seiner Definition Werte zuweisen. Man bezeichnet diesen Vorgang als Initialisierung. Hierzu ein Beispiel:

```
int hoch2 [10]={1,2,4,8,16,32,64,128,256,512};
```

Mehrdimensionale Felder erhält man durch Anhängen weiterer Indizes:

```
Datentyp Feldname [n] [m] ...;
```

Von besonderer Bedeutung sind Felder des Datentyps char. Sie werden meistens zum Speichern von Zeichenketten (Strings) verwendet. Hier ein Beispielprogramm zum Einlesen einer Zeile von der Tastatur:

```
main ()
{
   short i;
   char zeile[81],c;

   printf("Text eingeben:\n");
   for (i=0;(i<80)&&(c=getchar())!='\n';zeile[i++]=c);
   zeile[i]='\0';
   printf("Kontrolle:\n");
   printf("%s\n",zeile);
}
```

Die For-Anweisung bedarf einer kurzen Erläuterung, da sie sehr komplex aufgebaut ist. i erhält den Startwert null. Die Schleife wird solange durchlaufen, wie i noch kleiner als 80 und das von der Tastatur eingegebene Zeichen ungleich Return ist. Das geholte Zeichen wird dann in dem Feld zeile untergebracht, und der Index i wird um eins erhöht.

Nach Beendigung der Schleife wird an die Zeichenkette noch das Null-Zeichen '\0' angehängt. Zeichenketten haben in C immer dieses Ende-Kennzeichen.

Der letzte Printf-Befehl demonstriert, wie man damit auch Zeichenketten ausgeben kann. %s ist der Platzhalter für einen String, und zeile ist eben der Name der Zeichenkette.

Genaugenommen ist zeile eine Konstante, die die Anfangsadresse des Datenfeldes enthält, also die Speicherstelle, an der das erste Zeichen des Feldes liegt. Da Datenfelder zusammen-

hängend im Speicher abgelegt werden, ist zeile+1 die Adresse des zweiten Zeichens im Feld, zeile+2 die des dritten Zeichens und so weiter.

Zum Rechnen mit Adressen muß man wissen, daß zeile+1 auch dann die Adresse des zweiten Feldelementes bezeichnet, wenn zeile kein Char-Feld ist und die Elemente aus mehr als einem Byte bestehen. Der Compiler rechnet automatisch die richtige Adresse unter Einbeziehung der Elementgröße aus.

Man kann die Adresse einer beliebigen Variablen ermitteln, indem man vor die Variable den Operator & schreibt. &i ist die Adresse der Variablen i. &zeile[0] ist demnach die Adresse des ersten Zeichens im Feld zeile und hat die gleiche Bedeutung wie die Konstante zeile.

Andersherum kann man den Wert einer Speicherstelle durch den Operator * ansprechen. *(zeile+2) ist der Inhalt der Adresse zeile+2, also das gleiche wie zeile[2]. *(&i) ist daher gleichbedeutend mit i.

Um mit Adressen richtig zu arbeiten, benötigt man Variablen, die deren Werte aufnehmen können. Diese werden als Zeiger bezeichnet, da sie auf eine Speicherstelle zeigen. Folgende Vereinbarung definiert eine Zeigervariable:

```
Datentyp *Variablenname;
```

Der Datentyp ist der Typ einer Variablen, auf die die Zeigervariable zeigen soll. Sie kann nicht auf Objekte verschiedenen Typs zeigen. Das Datenformat der Zeigervariablen selbst ist davon völlig unabhängig. Da die Zeigervariable nur eine Speicheradresse aufnehmen soll, belegt sie grundsätzlich vier Bytes. Ein Zeiger auf eine Char-Variable wird demnach so vereinbart:

```
char *zeiger;
```

Wenn c eine Char-Variable ist, sind nun folgende Operationen erlaubt:

```
zeiger=&c;
*zeiger='a';
```

Durch die letzte Anweisung wird der Inhalt der Speicherstelle verändert, auf die zeiger zeigt. Dies wäre aber in diesem Beispiel die Adresse der Variablen c. Deshalb hat die Zuweisung die gleiche Wirkung wie c='a';.

Zeiger kann aber auch als Zeiger auf ein Datenfeld verwendet werden:

```
zeiger=zeile; /* oder: zeiger=&zeile[0]; */
++zeiger; /* zeiger zeigt jetzt auf zeile[1] */
printf("%c\n",*zeiger);
```

Viele Operationen, die man über Zeiger abwickeln kann, lassen sich auch direkt über Indizes erledigen. Dies ist aber nicht immer der Fall, wie wir schon am Beispiel der Printf-Anweisung gesehen haben. Anwendungen für Zeiger werden wir aber nach dem nächsten Kapitel sicherlich genügend haben.

Eine Sache ist noch wichtig in bezug auf Zeichenketten-Konstanten. Zeichenketten-Konstanten sind Anreihungen von Zeichen, die mit Anführungsstrichen umgeben werden. Der Compiler ersetzt diese im Programmtext durch einen Zeiger auf die Zeichenkette. Die zugehörigen Zeichen werden, ergänzt um das \0-Zeichen, hinter dem Programmcode abgelegt. Dadurch ist folgende Anweisung möglich:

```
char *text;
text="Hallo";
```

Beachten Sie dabei, daß durch diese Zuweisung nur ein Zeiger zugewiesen wird. C hat keine Befehle, um Zeichenketten als Ganzes zu verwenden. Folgendes ist daher falsch:

```
char text2[80];
text2=text; /* falsch */
```

Die gewünschte Zuweisung muß statt dessen so verwaltet werden:

```
for(i=0;(text2[i]=*(text+i))!='\0' && i<79;i++);
```

Es gibt einen Fall, bei dem Zeichenketten-Konstanten anders verwaltet werden, nämlich bei der Initialisierung. Die Definition:

```
char text[20]="Hallo";
```

ist durchaus korrekt und steht als Kürzel für:

```
char text[20]={'H','a','l','l','o','\0'};
```

Bei der Formulierung:

```
char text[]="Hallo";
```

ordnet der Compiler dem Datenfeld text automatisch eine Länge von sechs zu, also fünf Zeichen und das Zeichen '\0'.

5.7 Funktionen

Mit Funktionen bezeichnet man in C alle möglichen Typen von Unterprogrammen, egal, ob man an sie Werte übergibt oder ob sie Werte zurückliefern. Beginnen wir auch hier wieder mit einem Beispielprogramm:

```
short max(a,b)
short a,b;
{
   if (a>b)
      return(a);
   else
      return(b);
}

main ()
{
   short c,d;
```

```
    c=10;
    d=11;
    printf("%d\n",max(c,d));

    c=12;
    printf("%d\n",max(c,d));
}
```

Dieses Programm beinhaltet eine Funktion mit dem Namen max, die das Maximum zweier ihr übergebener Werte bestimmt. Den prinzipiellen Aufbau einer Funktion kennen wir ja schon von main her. Der erste Unterschied, den wir bei max feststellen, ist die Angabe eines Datentyps vor dem Funktionsnamen. short max(...) sagt aus, daß max eine Funktion ist, die einen Wert vom Typ short zurückliefert.

Wir hätten das Schlüsselwort short aber auch weglassen dürfen. In C braucht man den Datentyp einer Funktion nicht anzugeben, wenn ihr Ergebnis ganzzahlig bzw. vom Typ char ist, oder wenn sie keinen Wert zurückliefert. Man kann und sollte es aber tun, allein schon der Übersichtlichkeit halber, und weil vielleicht nicht jeder Compiler ohne diese Angabe auskommt. Funktionen ohne Wertrückgabe erhalten dann den Datentyp void. Überall, wo wir bisher main geschrieben haben, hätten wir auch void main benutzen können, da main ja keinen Wert zurückliefert.

Wenn eine Funktion kein ganzzahliges Ergebnis liefert, so muß der Datentyp nicht nur bei der Definition der Funktion angegeben werden, sondern auch noch innerhalb aller aufrufenden Funktionen. Will die Main-Funktion zum Beispiel eine Funktion vom Typ double verwenden, so muß bei den Variablendefinitionen auch die Vereinbarung double Funktionsname() erfolgen.

Hinter dem Funktionsnamen stehen in Klammern mehrere Variablen. Diese dienen zur Übergabe von Parametern vom aufrufenden Programm an die Funktion. Welche Datentypen diese Variablen besitzen, wird danach, noch vor Beginn des Befehlsblockes, festgelegt. Wenn man im Hauptprogramm hinterher die Funktion max mit max(c,d) aufruft, werden die Inhalte der Va-

riablen c und d in die Variablen a und b übertragen. Würde max dann a oder b irgendwelche Werte zuweisen, würden c und d nicht verändert.

Die Anweisung Return beendet die Funktion. Dabei kann hinter Return in Klammern ein Rückgabewert angegeben werden, der natürlich mit dem Datentyp der Funktion übereinstimmen muß. Im Beispiel oben wird entweder a oder b zurückgeliefert, je nachdem, welcher der beiden Werte größer ist. Natürlich braucht die Funktion nicht mit diesem Befehl abgeschlossen zu werden, wenn kein Wert zurückgegeben werden soll.

Die Reihenfolge, in der Funktionen im Programm stehen, ist unerheblich. Man kann also eine Funktion schon benutzen, bevor sie im Text definiert wird. Man kann nicht, wie in Pascal, Funktionsdefinitionen schachteln.

Alle Variablen, die innerhalb einer Funktion definiert werden, sind lokal. Das bedeutet, sie können nicht innerhalb anderer Funktionen verändert oder benutzt werden. Davon sind auch die Variablen betroffen, die zur Übergabe von Parametern verwendet werden. Wenn zwei verschiedene Funktionen gleiche Variablennamen vereinbaren, so handelt es sich trotzdem um verschiedene Variablen. Beispielsweise kann im obigen Beispielprogramm von der Funktion main aus nicht auf die Variablen a und b zugegriffen werden, genauso, wie von max aus die Variablen c und d nicht zur Verfügung stehen.

Will man dagegen mit globalen Variablen arbeiten, also Variablen, die von jeder Funktion aus erreichbar sind, so müssen diese außerhalb von allen Funktionsdefinitionen vereinbart werden. Das könnte dann zum Beispiel so aussehen:

```
int a;
...
funktion1 (...)
{
...
a=b+c;
...
}
```

```
main ()
{
...
d=a;
...
}
```

So definierte Variablen werden auch als externe Variablen bezeichnet, während die innerhalb von Funktionen definierten interne Variablen genannt werden.

Der Übersetzungsvorgang eines Programms läuft normalerweise in mehreren Schritten ab. Einer dieser Schritte ist das sogenannte Linken. Dabei werden bestimmte, schon vorher übersetzte Programmteile mit dem aktuellen Programm zusammengebunden. Um Variablen und Funktionen, die innerhalb eines anderen Programmes definiert werden, auch von seinem eigenen Programm aus nutzen zu können, muß man sie noch einmal mit dem Schlüsselwort extern vereinbaren. Durch die externe Definition:

```
int a;
char text[100];
```

werden zwei Variablen geschaffen, und für sie wird Speicherplatz bereitgestellt. Dagegen sagt die Deklaration:

```
extern int a;
extern char text[];
```

dem Compiler, daß die Definition der Variablen a bzw. des Feldes text entweder noch später im Text oder in einer anderen Datei erfolgt.

Normalerweise behalten interne Variablen ihre Werte von einem Funktionsaufruf zum nächsten nicht. Durch die Angabe des Schlüsselwortes static vor der Deklaration behalten die Variablen ihre Werte. Trotzdem bleiben es lokale Variablen, die nicht von anderen Funktionen benutzt werden können. static kann aber

auch bei externen Variablen verwendet werden. Dadurch wird verhindert, daß Variablen, die in einer Quelldatei definiert werden, von einer anderen mit extern benutzt werden können.

Interne Variablen, die nicht durch static gekennzeichnet werden, nennt man auch Auto-Variablen, da sie bei jedem Funktionsaufruf automatisch neu erschaffen werden. Eine besondere Form dieser Variablen sind die sogenannten Register-Variablen. Die Vereinbarung:

```
register Datentyp Variable;
```

weist den Compiler an, die definierte Variable nach Möglichkeit innerhalb eines Prozessorregisters unterzubringen. Dieses sollte man für oft benutzte Variablen wie zum Beispiel Schleifenindizes verwenden. Beim Amiga können nur ganzzahlige Variablen als register vereinbart werden, also beispielsweise auch Zeiger. Die Adresse einer solchen Variablen läßt sich natürlich auch nicht ermitteln. Außerdem wird register ignoriert, wenn der Prozessor keine Register mehr frei hat.

Wen es interessiert: C erlaubt Rekursionen, also das Aufrufen einer Funktion durch sich selbst. Wir wollen hier darauf aber nicht weiter eingehen, da Rekursionen nur selten benötigt werden und relativ schwer für jemanden zu durchschauen sind, der sie noch nicht kennt.

Will man eine Funktion schreiben, die Variablen des Hauptprogramms verändern kann, muß man zu einem Trick greifen. Wir wollen das anhand einer Funktion demonstrieren, die eine bestimmte Variable um zwei erhöhen soll. Folgende Vorgehensweise funktioniert erst einmal nicht:

```
add2 (a)  /* falsch */
short a;
{
  a+=2;
}
```

```
main ()
{
  short wert;
  wert=0;
  add2(wert);
}
```

Hier wird zuerst der Wert der Variablen wert nach a übertragen, wonach dann a um zwei erhöht wird. wert bleibt dabei unverändert.

Um wert ändern zu können, muß man statt des Variableninhalts die Variablenadresse an die Funktion übergeben. Diese muß dann berücksichtigen, daß sie es mit einer Adresse zu tun hat.

```
add2 (a)  /* richtig */
short *a;
{
  *a+=2;
}

main ()
{
  short wert;
  wert=0;
  add2(&wert);
}
```

Genauso muß man verfahren, wenn man Datenfelder an Unterprogramme übergeben will, selbst dann, wenn die Felder nicht verändert werden sollen. Als Parameter sind nämlich nur einzelne Werte erlaubt. Angenommen, einer Funktion solle eine Zeichenkette übergeben werden. Der entsprechende Funktionsbeginn sähe dann so aus:

```
funktion (string)
char *string;
...
```

Statt char *string ist in diesem Fall auch char string[] erlaubt, was genau das gleiche bedeutet. In der Funktion lassen sich dann

die String-Elemente direkt über die Indizes ansprechen, also mit string[0], string[1] usw., oder natürlich über die Adressen: *string, *(string+1) usw.

Umgekehrt kann man natürlich auch keine Datenfelder durch Funktionen zurückliefern, sondern nur Zeiger darauf. Beachten Sie dabei unbedingt, daß auch hier der Datentyp der Funktion angegeben werden muß, und zwar sowohl bei der Funktionsdefinition als auch in der aufrufenden Funktion:

```
char *funktion(....)
{
...
}

main ()
{
char *funktion();
...
}
```

5.8 Der Präprozessor

Ein Teil des Compilers ist der sogenannte Präprozessor. Er verwaltet, welche Teile des Programms übersetzt werden, ermöglicht das Verbinden verschiedener Quelltexte und kann auch sogenannte Makros in den Text einfügen. Im folgenden werden wir kurz die beiden wichtigsten Befehle des Präprozessors erläutern, nämlich #define und #include.

Es ist guter Programmierstil, allen wichtigen Konstanten am Anfang eines Programms Namen zu geben und dann im Programm nur noch mit den Namen zu arbeiten. Dies macht die Programme oft verständlicher. Andererseits braucht man später bei Änderungen dieser Werte nur die Namensvereinbarung zu verändern und nicht das gesamte Programm. Dazu ein Beispiel:

```
#define MAX 100

main ()
{
   char text[MAX];
   int i;
   for (i=0;i<MAX;text[i++]=' ');
...
```

Der Befehl #define legt fest, daß der Name MAX überall im Programmtext bei der Übersetzung durch die Zahl 100 ersetzt werden soll. Die Namen solcher Konstanten schreibt man üblicherweise groß, auch wenn der Compiler dies nicht zwingend verlangt. In C wird streng zwischen Groß- und Kleinschreibung unterschieden, Max ist also nicht das gleiche wie MAX.

MAX wird auch als Makro bezeichnet. Es ist keine Variable, die den Wert 100 beinhaltet. Einerseits kann man nämlich keine Werte an MAX zuweisen, andererseits kann man mit #define einen beliebigen Text festlegen, durch den ein Name wie MAX ersetzt werden soll. Zum Beispiel ist folgendes erlaubt:

```
#define HOLEN a=feld[i++]
```

Überall im Text wird dann der Name HOLEN durch a=feld[i++] ersetzt. Nach der #define-Anweisung steht kein Semikolon. Das Ende der Definition wird durch das Ende der Zeile festgelegt. Auf diese Weise können auch Leerzeichen und Semikolons im Ersatztext vorkommen.

Mit #define festgelegte Namen können auch parameterisiert sein. Das sieht zum Beispiel so aus:

```
#define abs(a) ((a)>=0? (a):-(a))
```

Hierdurch schafft man quasi eine Funktion, die den Absolutbetrag einer Zahl liefert. Gegenüber einer normalen Funktionsdefinition ist sie unabhängig von irgendwelchen Datentypen. Ein Aufruf erfolgt folgendermaßen:

```
betrag=3*abs(b+c);
```

Der Präprozessor macht daraus folgenden Ausdruck:

```
betrag=3*((b+c)>=0? (b+c):-(b+c));
```

Bei Makros mit Parametern muß man auf die Klammerung sorgfältig achten. Wenn beim letzten Beispiel die äußerste Klammer fehlen würde, würde der Teil 3* nicht auf den gesamten Ausdruck, sondern nur auf (b+c) bezogen werden. Der Präprozessor führt keine Berechnung der Ausdrücke durch, sondern nur einen Textersatz.

Der andere wichtige Präprozessor-Befehl ist #include. Die Zeile:

```
#include "Dateiname"
```

bindet eine Quell-Datei an der Stelle mit in die Übersetzung ein, an der der Befehl steht. Dadurch kann man sehr lange Quelltexte in mehrere Teile aufspalten, die dann zusammen in einem Durchgang übersetzt werden. Der Übersetzer behandelt die verbundenen Dateien dabei so, als würde es sich um eine einzige Datei handeln.

Beim Amiga stellt jeder Compiler dem Programmierer alle Definitionen und Deklarationen des Betriebssystems zur Verfügung. Mit dem #include-Befehl kann man sie in seine Programme einbinden. Der Aufruf sieht am besten so aus:

```
#include <Dateiname>
```

Der Präprozessor sucht dann die einzubindende Datei in einem speziell dafür vorgesehenen Ordner. Wichtige Systemdateien sind zum Beipiel exec/types.h. Hier werden unter anderem die Makros BYTE, UBYTE, WORD, UWORD, LONG und ULONG vereinbart, die für die Datentypen char, short und long, einmal mit Vorzeichen und einmal unsigned stehen.

Sie können die Include-Dateien des Betriebssystems mit dem normalen ed-Editor laden und sich ansehen. Die Dateien sollten

sich auf Ihrer C-Compiler-Diskette, wahrscheinlich in einem Ordner mit dem Namen include, befinden. Weitere Informationen dazu finden Sie in den DATA BECKER Büchern C für Einsteiger und Amiga Systemprogrammierung.

5.9 Strukturen und Varianten

In einem Array lassen sich nur Daten gleichen Typs unterbringen. Will man dagegen zusammengehörige Daten unterschiedlichen Typs verwalten, so muß man auf Strukturen (engl. structs) zurückgreifen. Ein Standard-Beispiel für eine Struktur ist die Verwaltung von Adressen. Wir wollen eine Struktur schaffen, die Namen, Vornamen, Straße und Hausnummer aufnehmen kann:

```
struct {
   char  name[20];
   char  vorname[20];
   char  strasse[30];
   short hausnummer;
} a,b,c;
```

Dieser Anweisungsblock definiert drei Variablen a, b und c, die alle vollständige Adresse aufnehmen können. Die Definition

```
struct {...} a,b,c;
```

entspricht der Form nach einer Definition wie:

```
int a,b,c;
```

Für jede dieser Variablen wird also ein Speicherplatz von 72 Byte reserviert, zwei mal 20 Bytes für Namen und Vornamen, 30 Byte für die Straße und zwei Bytes für die Hausnummer. Um diese sogenannten Komponenten der Struktur ansprechen zu können, gibt man den Variablennamen an, danach einen Punkt und dahinter den Komponentennamen:

```
a.hausnummer=1;
d=b.name[0];
```

Oft benötigt man eine Strukturdefinition von Variablen gleichen Strukturtyps an mehreren Stellen im Programm. Dazu kann man einer Struktur einen Namen geben und diesen anstelle der Struktur verwenden. Am Beispiel der Adressenstruktur sähe das so aus:

```
struct adresse {
   char  name [20];
   char  vorname [20];
   char  strasse [30];
   short hausnummer;
};

struct adresse a,b,c;
```

Eine ganze Adreßkartei ließe sich durch folgende Definition schaffen:

```
struct adresse kartei[100];
```

Eine Strukturvariable, die global oder als static vereinbart wurde, kann direkt bei ihrer Definition Anfangswerte zugewiesen bekommen:

```
struct adresse a = {
   "Becker",
   "Data",
   "Merowinger Str.",
   30};
```

Strukturen lassen sich natürlich, genau wie Arrays, nicht einfach aneinander zuweisen, sondern man muß alle Elemente einzeln kopieren. Zur Übergabe von Strukturen an Funktionen dienen wiederum Zeiger. Eine Zeigervariable für einen Struktur kann so definiert werden:

```
struct adresse *zeiger;
```

Da man im Zusammenhang mit Strukturen Zeiger relativ häufig benötigt, gibt es hier eine mögliche Vereinfachung. Die Zuweisung:

```
zeiger=&a;
```

weist die Speicheradresse der Struktur a der Variablen zeiger zu. Jetzt könnte man auf die Elemente so zugreifen:

```
(*zeiger).name[0]='A';

Äquivalent dazu ist die Formulierung:

zeigername[0]='A';
```

Strukturen dienen, wie wir gesehen haben, dazu, verschiedene Datengruppen in mehreren Variablen unter einem Namen zusammenzufügen. Varianten dagegen sind einzelne Variablen, die verschiedene Datentypen annehmen können. Eine Variantendefinition ist genauso aufgebaut wie eine Strukturdefinition, nur daß statt des Schlüsselwortes struct das Wort union verwendet wird. Ein Beispiel:

```
union monat {
  int  nummer;
  char name[10];
} a;
```

Diese Definition vereinbart den Namen monat für die Variante und schafft gleichzeitig eine Variable a, die jetzt entweder den Monatsnamen oder die Monatsnummer aufnehmen kann. a ist groß genug, um die größte der angegebenen Komponenten zu speichern. Man sollte natürlich in einer weiteren Variablen festhalten, welchen Datentyp man gerade verwendet.

Monat läßt sich jetzt fast genauso ansprechen, als hätte man es mit einer Struktur zu tun. Über monat.nummer und monat.name[i] greift man demnach auf die Komponenten zurück. Man kann nur nicht gleichzeitig den Monatsnamen und die Monatsnummer behalten, da für beide der gleiche Speicherplatz verwendet wird.

5.10 Nutzung der Bibliotheksfunktionen

C ist eine Sprache, die keinerlei Ein- und Ausgabebefehle aufweist. Dies klingt sehr verwunderlich, denn schließlich haben wir schon mehrmals mit den Funktionen printf und getchar gearbeitet. Nun, diese Befehle kennt C eigentlich gar nicht. Wie wir aber schon in Kapitel 5.7 angemerkt haben, kann man Funktionen benutzen, die erst durch das Linken zum Programm hinzukommen. Dazu gehören unter anderem diese beiden Funktionen. Printf und getchar sind aber soweit standardisiert, daß man mit ihnen arbeiten kann, als ob es sich um C-eigene Befehle handeln würde.

Sobald man die ersten, etwas umfangreicheren Programme auf dem Amiga entwerfen will, sollte man auch etwas mehr Aufwand in die Ein- und Ausgabe stecken. Dazu kann man alle Angebote nutzen, die einem das Betriebssystem bietet, also Windows, Gadgets, Screens, Menüs usw. Da das Betriebssystem selbst in C geschrieben wurde, haben alle Systemfunktionen festgelegte Namen, die Ihr C-Compiler kennt. Man kann diese Routinen fast genauso benutzen wie Standard-Funktionen. Man muß dabei nur beachten, daß alle Routinen in Libraries (Bibliotheken) organisiert sind.

Eine Library besteht aus zwei grundlegenden Teilen. Unterhalb einer Basisadresse liegt eine Sprungtabelle, die die Einsprünge aller Funktionen einer Library enthält. Der Aufbau der Sprungtabelle braucht uns nicht zu interessieren, da, wie schon gesagt, der C-Compiler die Funktionsnamen kennt und weiß, welche Einsprünge dazu gehören. Oberhalb der Basisadresse befindet sich eine Library-Struktur, die alle möglichen wichtigen Werte in bezug auf die entsprechende Library enthält. In der Tat handelt es sich dabei um eine C-struct, und die Basisadresse wird in einer Variablen gespeichert, die vom Typ Zeiger auf Library-Struktur ist.

Um die Sprungtabelle und die Struktur einer Library im Speicher zu erzeugen und um die Basisadresse zu erfahren, wird der Befehl OpenLibrary benötigt. Dieser Funktion übergibt man den Namen und die Versionsnummer der Library, auf die man zu-

greifen möchte, und man erhält dann die Basisadresse zurück. Diese speichert man in der dafür vorgesehenen Zeigervariablen. Der Aufbau der Library-Struktur muß natürlich vorher richtig definiert worden sein, damit man diese Variable überhaupt vereinbaren kann. Man braucht das aber nicht selbst durchzuführen, sondern es genügt, die entsprechende Include-Datei einzubinden. Dazu ein Beispiel. Wir wollen die Library intuition.library öffnen. Dazu sind im Prinzip folgende Schritte notwendig:

#include <intuition/intuition.h>

/* Hierdurch werden alle möglichen Konstanten- und Struct-Definitionen in den Text eingefügt, unter anderem die struct IntuitionBase, die den Datentyp für die Zeigervariable darstellt, die die Basisadresse aufnehmen soll. */

#include <functions.h>

/* Alle externen Funktionen, die nicht vom Datentyp int oder char sind, müssen vorher deklariert werden. Diese Include-Datei enthält alle notwendigen Vereinbarungen. */

*struct IntuitionBase *IntuitionBase;*

/* An dieser Stelle wird der Zeiger für die Basisadresse definiert. Er hat genau den gleichen Namen wie die struct selbst. Der Name ist fest vorgegeben und darf nicht anders lauten, da er von den eingebundenen Betriebssystemfunktionen benutzt wird. */

```
main ()
{
   if (!(IntuitionBase=(struct IntuitionBase *)
      OpenLibrary("intuition.library",0L))) {
      printf("Kann Library nicht öffnen!\n");
      exit(0);
   }
```

/* Hier zeigt sich wieder, welche Möglichkeiten C zum Entwurf unverständlicher Funktionsaufrufe bietet. Erklären wir deshalb genau, was hier abläuft. Zuerst wird die Routine OpenLibrary

aufgerufen. Als Parameter folgen der Libraryname und die Versionsnummer der Library als Long-Zahl. Letztere kann immer den Wert null haben. Der zurückgelieferte Wert wird in den Datentyp des Intuitionbase-Zeigers umgewandelt und an den Zeiger zugewiesen. Wenn OpenLibrary den Wert null liefert, konnte die Library nicht geöffnet werden. Dieser Fall wird mit der If-Anweisung abgetestet. Der nachfolgende Befehl exit(0) beendet das Programm. */

OpenLibrary ist übrigens eine Funktion aus einer Library, nämlich der sogenannten exec.library. Diese braucht aber nie geöffnet zu werden, was natürlich auch schlecht möglich wäre (warum wohl?).

Welche Betriebssystemfunktionen es gibt und wie die Library-Strukturen aufgebaut sind, kann natürlich in diesem Buch nicht beschrieben werden. Mit diesem Stoff ließe sich nämlich mehr als nur ein Buch füllen. Wenn wir Ihr Interesse geweckt haben, dann werfen Sie doch einmal einen Blick in eines der DATA BECKER Bücher C für Einsteiger, oder Amiga Intern. Auch das Buch Amiga Maschinensprache enthält einige wertvolle Beschreibungen von Betriebssystemroutinen.

6. Wie schnell ist der Amiga wirklich?

Einer der Gründe, warum Sie sich Ihren Amiga gekauft haben, war doch sicher die wesentlich höhere Geschwindigkeit, die dieser Rechner gegenüber dem C64 aufweist. In diesem Kapitel wollen wir einmal unter die Lupe nehmen, um wieviel schneller der Amiga wirklich ist. Dazu haben wir einen Geschwindigkeitsvergleich der Sprachen AmigaBASIC, C, 68000-Assembler, C64-BASIC und 6510-Maschinensprache vorgenommen.

6.1 Planung der Benchmark-Tests

Benchmark-Tests sind Programme, die die Geschwindigkeit bestimmter Funktionen einer Programmiersprache austesten. Um verschiedene Programmiersprachen wie die, die wir untersuchen, miteinander zu vergleichen, muß man darauf achten, was überhaupt vergleichbar ist. Wir haben uns dabei für folgendes Vorgehen entschieden:

Alle Tests laufen in einer Schleife ab, die 30000mal durchlaufen wird. Zuerst wäre da natürlich die Leerschleife selbst. Danach folgen Konstantenzuweisungen und die Grundrechenarten mit allen möglichen Datentypen, sofern die jeweilige Programmiersprache diese zuläßt. Den Abschluß bilden je ein Programm zum Verschieben eines 30000-Byte-Speicherblockes und ein Programm zur Berechnung der Primzahlen unterhalb von 10000 nach dem Sieb des Erastosthenes.

Wo es nötig war, haben wir die Programme in einer Schleife mehrmals ablaufen lassen und anschließend die ermittelte Zeit durch die Anzahl der Durchläufe geteilt.

Fließkommaberechnungen durch Maschinenprogramme haben wir weggelassen, da diese sowieso in Betriebssystemaufrufen enden müßten und wir hier ja weniger die Qualität des Fließkomma-Algorithmus sondern mehr die des Prozessors herausfinden wollten. Außerdem sind wir mehr von der praktischen Seite ausgegangen. Ein Maschinenspracheprogrammierer wird ja

viel eher Fließ- oder Festkommawerte in einer Tabelle ablegen und so Berechnungen überflüssig machen als zum Beispiel ein C-Programmierer. Genau das gleiche gilt für die Multiplikation und Division von Zahlen mit dem Prozessor 6510.

Das BASIC des C64 hat keine wirklichen Integer-Rechenroutinen. Daher läßt sich hier auch kein direkter Vergleich ziehen. Das Fließkommaformat ist zwar nicht genau das gleiche wie beim Amiga, es ähnelt aber der 32-Bit-Darstellung, die sowohl von der Sprache AmigaBASIC als auch von der Sprache C unterstützt werden. Deshalb haben wir alle Rechentests des BASIC V2 den Fließkommarechentests der anderen Programmiersprachen gegenübergestellt.

6.2 Die Benchmark-Programme

Beginnen wir mit den AmigaBASIC-Programmen. Als Schleifenvariablen wurden grundsätzlich Ganzzahlvariablen verwendet. Bei den Grundrechenarten haben wir die Subtraktion weggelassen, da sie nahezu die gleichen Werte wie die Addition liefert.

Benchmark 1: Leerschleife

```
t=TIMER
FOR i%=1 TO 30000:NEXT
PRINT TIMER-t
```

Benchmark 2a: Ganzzahl-Konstantenzuweisung

```
t=TIMER
FOR i%=1 TO 30000
a%=1234
NEXT
PRINT TIMER-t
```

Für die Benchmarktests 2b, 2c und 2d ersetzen Sie a% einfach durch a&, a! und a#.

Benchmark 3a: Ganzzahl-Addition

```
t=TIMER
b%=1234:c%=5678
FOR i%=1 TO 30000
  a%=b%+c%
NEXT
PRINT TIMER-t
```

Für die Benchmarktests 3b, 3c und 3d ersetzen Sie bei den Variablen a%, b% und c% das %-Zeichen durch &, ! und #.

Benchmark 4a: Ganzzahl-Multiplikation

```
t=TIMER
b%=12:c%=34
FOR i%=1 TO 30000
  a%=b%*c%
NEXT
PRINT TIMER-t
```

Für die Benchmarktest 4b, 4c und 4d gilt das gleiche wie für die Tests 3b bis 3d.

Die Tests 5a bis 5d sehen genauso aus wie die Tests 4a bis 4d, nur daß statt des Multiplikationszeichens * ein Divisionszeichen / verwendet wird.

Benchmark 6: Speicherblock verschieben

```
CLEAR ,35000&
a$=SPACE$(30001)
a&=SADD(a$)
b&=a&+29999
t=TIMER
FOR i&=a& TO b&
  POKE i&,PEEK(i&+1)
NEXT
PRINT TIMER-t
```

Benchmark 7: Sieb des Erastosthenes

```
max%=10000
DIM prim%(max%)
ende%=SQR(max%)
t=TIMER
FOR i%=2 TO ende%
  IF prim%(i%)=0 THEN
   FOR j%=i%+i% TO max% STEP i%
    prim%(j%)=1
   NEXT
  END IF
NEXT
t1=TIMER-t
PRINT
PRINT "Rechenzeit :";t1
```

Die äußere Schleife arbeitet nur bis SQR(max%), da beim Erreichen dieses Wertes schon alle Nicht-Primzahlen aus der Liste gestrichen wurden. Am Schluß wird im Feld prim% jede Primzahl durch eine Null, jede Nicht-Primzahl durch eine Eins repräsentiert. Die Bildschirmausgabe der berechneten Zahlen ist nicht erforderlich, da wir uns ja nur für die reine Rechenzeit interessieren.

Die BASIC-Programme für den C64 sehen fast genauso aus wie die für den Amiga und werden daher hier nicht mehr aufgelistet. Die Unterschiede liegen darin, daß überall nur Fließkommavariablen verwendet werden, weil das beim C64 schneller geht. Daher fallen auch die Versionen a, b und d der Tests zwei bis fünf weg. Außerdem gelten die schon in Kapitel 3.4.1 erläuterten Unterschiede in der Zeitmessung mit der Systemvariablen TI.

Im folgenden nun die entsprechenden C-Programme. Die Werte der Benchmarks hängen natürlich vom verwendeten Compiler ab, sie vermitteln aber ein Bild von der Größenordnung, in der man sich bewegt.

Benchmark 1: Leerschleife

```
#define MAX 30000
#define TIMES 100

void
main ()
{
   register short i,j;
   short a;

   printf("Es geht los!\n");
   for (i=0;++i<=TIMES;)
      for (j=0;++j<=MAX;);
   printf("Fertig!\n");
}
```

Da sich in C die Laufzeit nicht so einfach wie in BASIC per Programm bestimmen läßt, haben wir die Schleife 100mal ablaufen lassen und die von Hand gemessene Zeit durch 100 geteilt. Auf diese Weise erhält man sehr genaue Ergebnisse. Die Zeit, die die äußere Schleife benötigt, fällt dabei nicht ins Gewicht und kann deshalb außer acht gelassen werden.

Benchmark 2a: Ganzzahl-Konstantenzuweisung

```
#define MAX 30000
#define TIMES 100

void
main ()
{
   register short i,j;

   short a;
   /* Für die Benchmarks 2b bis 2d verwenden Sie hier
      long a, float a und double a statt short a */

   printf("Es geht los!\n");
   for (i=0;++i<=TIMES;)
      for (j=0;++j<=MAX;)
         a=1234;
   printf("Fertig!\n");
}
```

Benchmark 3a: Ganzzahl-Addition

```
#define MAX 30000
#define TIMES 100

void
main ()
{
   register short i,j;

   short a,b,c;
   /* Bei den anderen Benchmark-Tests
   verwenden Sie statt 'short' die Datentypen
   'long', 'float' und 'double'. */

   b=1234;
   c=5678;
   printf("Es geht los!\n");
   for (i=0;++i<=TIMES;)
      for (j=0;++j<=MAX;)
         a=b+c;
   printf("Fertig!\n");
}
```

Die anderen Benchmarks von 2b bis 5d sind genauso aufgebaut, nur entsprechend mit anderen Datentypen und anderen Rechenoperationen.

Benchmark 6: Speicherblock verschieben

```
#define MAX 30000
#define TIMES 100

void
main ()
{
   register short i,j;
   char block [MAX+1];
```

```
   printf("Es geht los!\n");
   for (i=0;++i<=TIMES;)
      for (j=0;++j<=MAX;)
         block[j-1]=block[j];
   printf("Fertig!\n");
}
```

Für diesen Benchmark-Test sollten Sie einen Stapelspeicher von mindestens 35000 Bytes reservieren. Dies geschieht vom CLI aus mit der Anweisung stack 35000. In C lassen sich Speicherblock-Zugriffe direkt über Arrays durchführen. Deshalb ist das verschieben von Blöcken hier nicht mehr als die Zuweisung von Array-Elementen.

Benchmark 7: Sieb des Erastosthenes

```
#define MAX 10000
#define ENDE 100 /* 100=SQR(10000) */
#define TIMES 100

main()
{
   register short i,j,z;
   short prim [MAX];
   for (i=1;++i<MAX;prim[i]=0);

   printf("\nEs geht los!\n");

   for (z=1;z<=TIMES;z++)
      for (i=1;++i<=ENDE;)
         if (!prim[i])
            for (j=i+i;j<MAX;j+=i)
               prim[j]=1;

   printf("Fertig!\n\n");

}
```

Das Feld prim[] braucht nur am Anfang auf null gesetzt zu werden, da der Algorithmus genau die gleiche Rechenleistung erfüllen muß, wenn ein korrekt berechnetes Feld vorliegt.

Nun noch die entsprechenden Routinen in 68000-Assembler. Schleifenvariablen wurden von uns in Prozessor-Registern untergebracht, die Zuweisungs- und Berechnungsvariablen dagegen in Speicheradressen, da man in der Praxis fast immer genauso arbeitet. Diverse Interrupts wurden nicht abgeschaltet, da man diese Möglichkeit bei den Hochsprachen auch nicht hat und die meisten umfangreicheren Maschinenprogramme ebenfalls keinen Gebrauch davon machen. Die Versionen c und d der Tests zwei bis fünf fallen weg, da der Prozessor keine eigenen Fließkommabefehle kennt.

Benchmark 1: Leerschleife

```
start:  move.l d2,-(sp)        ;d2 auf Stack retten
        move #99,d2            ;hundertfacher Ablauf
l1:     move #29999,d1         ;30000 Durchläufe
l2:     dbf d1,l2              ;dekrement, verzweige, wenn > -1
        dbf d2,l1              ;dekrement, verzweige, wenn > -1
        move.l (sp)+,d2        ;d2 zurückholen
        rts                    ;Rücksprung
```

Benchmark 2a: Ganzzahl-Konstantenzuweisung

```
start:  move.l d2,-(sp)
        move #99,d2
l1:     move #29999,d1
l2:     move.w #1234,vara      ;Konstante in Speicher schreiben
        dbf d1,l2
        dbf d2,l1
        move.l (sp)+,d2
        rts
vara:   dc.l 0                 ;freier Speicher
```

In Maschinensprache entspricht der Schreibzugriff auf eine Speicherstelle einer Konstantenzuweisung in einer Hochsprache. Man kann den Ablauf in bestimmten Fällen noch beschleunigen, indem man die Variablenadresse vor Schleifenbeginn in ein Adreßregister lädt. Wir wollten aber den allgemeinen Fall betrachten. Für Benchmarktest 2b ersetzen Sie den move.w-Befehl hinter l2: durch einen move.l-Befehl.

Benchmark 3a: Ganzzahl-Addition

```
start:  move.l d2,-(sp)
        move #99,d2
l1:     move #29999,d1
l2:     move.w vara,d0          ;Variablenwert holen
        add.w varb,d0           ;zweiten Wert addieren
        move.w d0,varc          ;Summe wegspeichern
        dbf d1,l2
        dbf d2,l1
        move.l (sp)+,d2
        rts
vara:   dc.w 1234
varb:   dc.w 5678
varc:   dc.w 0
```

Auch für Benchmark 3b gilt: Ersetzen Sie bei den drei Befehlen in der innersten Schleife das .w durch ein .l, und verfahren Sie außerdem genauso mit allen dc-Befehlen.

Benchmark 4a: Ganzzahl-Multiplikation

```
start:  move.l d2,-(sp)
        move #99,d2
l1:     move #29999,d1
l2:     move.w vara,d0          ;Variablenwert holen
        muls varb,d0            ;mit zweitem Wert multiplizieren
        move.l d0,varc          ;Produkt wegspeichern
        dbf d1,l2
        dbf d2,l1
        move.l (sp)+,d2
        rts
vara:   dc.w 1234
varb:   dc.w 5678
varc:   dc.l 0
```

Die Multiplikation ist nur in diesem Datenformat möglich. Es gibt also keine Versionen b, c und d.

Benchmark 5a: Ganzzahl-Division

```
start:  move.l d2,-(sp)
        move #99,d2
l1:     move #29999,d1
l2:     move.l varb,d0          ;Variablenwert holen
        divs vara,d0            ;durch zweiten Wert dividieren
        move.w d0,varc          ;Quotient wegspeichern
        dbf d1,l2
        dbf d2,l1
        move.l (sp)+,d2
        rts
vara:   dc.w 1234
varb:   dc.l 5678
varc:   dc.w 0
```

Hier gilt das gleiche wie bei der Multiplikation.

Benchmark 6: Speicherblock verschieben

```
start:  move #99,d1
l1:     move #29999,d0
        lea block,a0            ;Adreßzeiger Zielbereich
        lea block+1,a1          ;Adreßzeiger Quellbereich
l2:     move.b(a1)+,(a0)+       ;Byte verschieben
        dbf d0,l2
        dbf d1,l1
        rts
block:  blk 30001
```

Der Block wird byteweise verschoben, da man sonst Probleme mit den ungeraden Adressen bekommt. Wo es aber möglich ist, kann man Speicherblöcke als Langworte verschieben, was den Ablauf deutlich beschleunigt.

Benchmark 7: Sieb des Erastosthenes

```
max     =10000
ende    =100                ;Wurzel von 10000
times   =100
start:  move.l d2,-(sp)     ;d2 retten
        lea prim,a0         ;Anfang des Daten-Speichers
        move #max/4,d0      ;Zähler für Langworte
l1:     clr.l (a0)+         ;ein Wort löschen
        dbf d0,l1           ;dekrement, verzweige, wenn > -1
        lea prim,a0         ;Anfang des Daten-Speichers
        moveq#times-1,d2    ;Wiederhol-Zähler setzen
l5:     moveq#2,d0          ;Primzahl-Zähler
l2:     tst.b (a0,d0.w)     ;Zahl prim?
        bne.s l3            ;verzweige, wenn nein
        move d0,d1
        add d0,d1           ;Startwert: Primzahl mal zwei
l4:     move.b#1,(a0,d1.w)  ;Nicht-Primzahlen auf eins setzen
        add d0,d1           ;Zeiger erhöhen
        cmpi #max,d1        ;kompletter Bereich erfaßt?
        ble.s l4            ;verzweige, wenn nein
l3:     addq #1,d0          ;Zähler erhöhen
        cmpi #ende,d0       ;schon alle Zahlen bearbeitet?
        bne.s l2            ;verzweige, wenn nein
        dbf d2,l5           ;nächster Schleifendurchgang
        move.l (sp)+,d2     ;d2 zurückholen
        rts                 ;Rücksprung
prim:   blk max+4           ;Speicherblock für Primzahlen
```

Den Abschluß bilden die Maschinenspracheroutinen für den C64. Diese unterliegen natürlich allen Einschränkungen, die auch die 68000-Programme haben. Zusätzlich fehlen die Multiplikations- und Divisions-Benchmarks, da diese nicht mit einfachen Prozessoranweisungen machbar sind. Alle Tests verwenden Zeropage-Adressen, um die Ablaufgeschwindigkeit zu erhöhen.

Benchmark 1: Leerschleife

```
c000 lda #$64                ;100
c002 sta $59                 ;in Zähler schreiben
c004 ldx #$30                ;LOW-Byte von 30000
c006 ldy #$75                ;HIGH-Byte von 30000
c008 txa                     ;Test, ob LOW-Byte null
c009 bne $c00e               ;verzweige, wenn nein
c00b dey                     ;HIGH-Byte verringern
c00c bmi $c012               ;verzweige, wenn Ende
c00e dex                     ;LOW-Byte verringern
c00f jmp $c009               ;zum Schleifenanfang
c012 dec $59                 ;Zähler für hundertfachen Ablauf
verringern
c014 bne $c004               ;verzweige, wenn ungleich null
c016 rts                     ;Rücksprung
```

Benchmark 2a: Ganzzahl-Konstantenzuweisung

```
c000 lda #$64
c002 sta $57
c004 ldx #$30
c006 ldy #$75
c008 lda #$12                ;LOW-Byte für Zuweisung
c00a sta $58                 ;wegspeichern
c00c lda #$34                ;HIGH-Byte für Zuweisung
c00e sta $59                 ;wegspeichern
c010 txa
c011 bne $c016
c013 dey
c014 bmi $c01a
c016 dex
c017 jmp $c008
c01a dec $57
c01c bne $c004
c01e rts
```

Für die Zuweisung von 32-Bit-Werten müssen an der entsprechenden Stelle zwei Lade- und Speicherbefehle hinzugefügt werden.

Benchmark 3a: Ganzzahl-Addition

```
c000 lda #$0a                ;10
c002 sta $57                 ;in Zähler schreiben
c004 ldx #$30
c006 ldy #$75
c008 clc                     ;Carry für Addition löschen
c009 lda $58                 ;LOW-Byte der ersten Variablen holen
c00b adc $5c                 ;zu LOW-Byte der zweiten addieren
c00d sta $60                 ;wegspeichern
c00f lda $59                 ;erstes HIGH-Byte holen
c011 adc $5d                 ;zweites HIGH-Byte addieren
c013 sta $61                 ;wegspeichern
c015 txa
c016 bne $c01b
c018 dey
c019 bmi $c01f
c01b dex
c01c jmp $c008
c01f dec $57
c021 bne $c004
c023 rts
```

Auch hier ist die entsprechende Änderung zur Verarbeitung von 32-Bit-Zahlen klar. Es müssen nur dementsprechend mehr Lade-, Speicher- und Additionsbefehle in die innerste Schleife eingebaut werden.

Benchmark 6: Speicherblock verschieben

```
c000 lda #$0a                ;Zähler für zehnfache Wiederholung
c002 sta $59                 ;setzen
c004 ldy #$00                ;LOW-Byte des
c006 sty $57                 ;Zeigers setzen
c008 lda #$20                ;HIGH-Byte des
c00a sta $58                 ;Zeigers setzen
c00c iny                     ;Zeiger plus eins
c00d lda ($57),y             ;Byte holen
c00f dey                     ;Zeiger minus eins
c010 sta ($57),y             ;Byte wegspeichern
c012 iny                     ;Zeiger LOW-Byte inkrementieren
c013 bne $c017               ;verzweige, wenn kein Übertrag
c015 inc $58                 ;HIGH-Byte inkrementieren
c017 cpy #$30                ;schon 30000 Bytes verschoben? Test LOW
```

```
c019 bne $c00c          ;verzweige, wenn nein
c01b lda $58            ;HIGH-Byte holen
c01d cmp #$95           ;schon 30000 Bytes verschoben? Test HIGH
c01f bne $c00c          ;verzweige, wenn nein
c021 dec $59            ;Zähler für zehnfachen Durchlauf
                         verringern
c023 bne $c004          ;verzweige, wenn ungleich null
c025 rts                ;Rücksprung
```

Benchmark 7: Sieb des Erastosthenes

```
c000 jsr $c047          ;Speicherbereich auf null setzen
c003 lda #$0a           ;Zähler für zehnfachen Durchlauf
c005 sta $5b            ;setzen
c007 lda #$02           ;Beginn bei Zahl zwei
c009 sta $5d            ;LOW-Byte des Adreßzeigers setzen
c00b sta $57            ;aktuelle Zahl
c00d lda #$00           ;LOW-Byte des Zählers der
c00f sta $59            ;inneren Schleife setzen
c011 lda #$20           ;HIGH-Byte des
c013 sta $5e            ;Adreßzeigers setzen
c015 ldx #$00           ;Index auf null setzen
c017 lda ($5d,x)        ;Zahl prim?
c019 bne $c038          ;verzweige, wenn nein
c01b lda $57            ;Zahl holen
c01d asl                ;mal zwei
c01e tay                ;als LOW-Byte des Zählers verwenden
c01f lda #$20           ;HIGH-Byte des Schleifenzählers
c021 sta $5a            ;setzen
c023 lda #$01           ;Zahl als nicht-prim
c025 sta ($59),y        ;kennzeichnen
c027 clc                ;Carry für Addition löschen
c028 tya                ;Zeiger nach Akku holen
c029 adc $57            ;Primzahl addieren
c02b tay                ;Zeiger zurückschreiben
c02c lda $5a            ;HIGH-Byte des Zeigers holen
c02e adc #$00           ;Übertrag addieren
c030 sta $5a            ;zurückschreiben
c032 cpy #$11           ;schon 10000
c034 sbc #$47           ;überschritten?
c036 bcc $c023          ;verzweige, wenn nein
c038 inc $57            ;nächste Zahl bearbeiten
c03a inc $5d            ;Adreßzeiger erhöhen
c03c lda $57            ;Zahl holen
```

```
c03e cmp #$65          ;schon 100 überschritten?
c040 bne $c017         ;verzweige, wenn nein
c042 dec $5b           ;Zähler für zehnfachen Durchlauf
                        verringern
c044 bne $c007         ;verzweige, wenn ungleich null
c046 rts               ;Rücksprung

c047 ldy #$00          ;Adreßzeiger LOW-Byte
c049 sty $5d           ;setzen
c04b lda #$20          ;Adreßzeiger HIGH-Byte
c04d sta $5e           ;setzen
c04f ldx #$27          ;Anzahl der Pages merken
c051 tya               ;Akku auf null

c052 sta ($5d),y       ;Byte löschen
c054 iny               ;Zeiger LOW-Byte erhöhen
c055 bne $c052         ;verzweige, wenn noch nicht Page-Ende
c057 inc $5e           ;Zeiger HIGH-Byte erhöhen
c059 dex               ;Page-Zähler verringern
c05a bne $c052         ;verzweige, wenn nicht null
c05c rts               ;Rücksprung
```

6.3 Die Ergebnisse

Nachdem wir eine solch umfangreiche Testreihe durchgeführt haben, wollen wir am Schluß auch eine Auswertung der Testergebnisse durchführen. Hier also zunächst in tabellarischer Form die gemessenen Zeiten in Sekunden:

Test Nr.	Am.-BASIC	C	68000	C64-BASIC	6510
1	4.4	0.16	0.05	36.2	0.26
2a	14.8	0.23	0.14	---	0.66
3a	22.7	0.39	0.27	---	1.00
4a	22.9	0.51	0.43	---	---
5a	44.7	2.51	0.90	---	---
2b	17.2	0.27	0.18	---	1.00
3b	23.0	0.39	0.34	---	1.60
4b	24.0	1.60	---	---	---
5b	59.6	2.10	---	---	---
2c	22.9	0.27	---	197.8	---
3c	26.0	1.87	---	115.6	---
4c	26.4	3.00	---	157.9	---
5c	27.6	3.90	---	193.9	---
2d	25.0	0.27	---	---	---
3d	29.5	1.87	---	---	---
4d	33.7	3.00	---	---	---
5d	36.7	3.90	---	---	---
6	43.9	0.35	0.11	191.0	0.80
7	16.7	0.22	0.12	103.7	0.70

Test 1: Leerschleife
Test 2: Zuweisungen
Test 3: Additionen
Test 4: Multiplikationen
Test 5: Divisionen,
Test 6: Speicherblock verschieben,
Test 7: Primzahlen

Zusatz a: kurze Ganzzahlen
Zusatz b: lange Ganzzahlen
Zusatz c: einfach genaue Fließkommazahlen
Zusatz d: doppelt genaue Fließkommazahlen

Aus den Testergebnissen läßt sich einiges über die Leistungsfähigkeit der verschiedenen Rechner und Programmiersprachen sagen. Beginnen wir mit dem Vergleich der BASIC-Interpreter. Hier zeigt sich die deutliche Überlegenheit des AmigaBASIC. Im Schnitt ist AmigaBASIC vier- bis achtmal schneller als BASIC V2, was sicherlich zum Teil auch auf die Möglichkeit der Verwendung von Ganzzahl-Variablen zurückzuführen ist. Dadurch laufen nämlich gerade Schleifen sehr viel effizienter ab.

Alleine die Programmiersprache C ist schon zwei- bis viermal schneller als 6510-Maschinensprache. Daher ist natürlich verständlich, daß reiner 68000-Maschinencode einen noch stärkeren Geschindigkeitszuwachs mit sich bringt. Dieser bringt gegenüber C nochmal einen Faktor von zwei bis drei. Dies spricht aber nicht nur für die Qualität des 68000-Prozessors, sondern auch für die der Programmiersprache C und des dabei verwendeten Compilers. C ist eine Sprache, die sich besonders gut in Maschinensprache übersetzen läßt, was auch erklärt, warum die Entwickler des Amiga das Betriebssystem in dieser Sprache geschrieben haben.

7. Datenübertragung vom C64 zum Amiga

Haben auch Sie schon festgestellt, daß Ihnen eine Menge Arbeit bevorsteht, bevor Sie mit Ihrem neuen Rechner Ihre Adressen schneller sortieren, Ihre Tabellen effektvoller kalkulieren und Ihre Texte besser erstellen können. Schließlich muß die 50 KByte lange Datei, für die der Arbeitsspeicher Ihres alten C64 nicht mehr ausreicht, und die Sie daher auf Ihrem Amiga weiterverarbeiten wollen, erst einmal in mühsamer Kleinarbeit abgetippt werden. Oder wie sieht es mit Grafik aus? Die Grafik des Amiga stellt zwar so ziemlich alles in den Schatten, was Sie bisher vom 64er kannten, aber so schlecht waren die Bilder des alten Rechners auch nicht. Wäre es für Sie nicht von Vorteil, wenn Sie einige der alten Bilder auch auf dem Amiga zur Verfügung hätten? Die Lösung für Ihr Problem lautet: Datenübertragung. In diesem Kapitel zeigen wir Ihnen, wie Sie ein Verbindungskabel für die Rechner Amiga und C64 bauen. Außerdem liefern wir Ihnen die nötige Übertragungssoftware.

7.1 Bau und Betrieb eines Verbindungskabels

Unser Ziel war es, eine möglichst einfache und effektive Möglichkeit zur Datenübertragung vom C64 zum Amiga zu schaffen. Am sinnvollsten erschien uns die Übertragung über den Userport des C64 zum Parallelport des Amiga. Der Anschluß ist so einfach, daß er auch von Elektronik-Laien durchgeführt werden kann. Sie sollten allerdings wissen, wie man mit einem Lötkolben umgeht. Wenn Sie sich auf diesem Gebiet unsicher fühlen, dann fragen Sie doch einen Freund oder Bekannten, ob er nicht das Kabel für Sie zusammenbaut. Wer will, kann sich auch das fertige Kabel zum Preis von 24.90 DM inkl. Mehrwertsteuer, zuzüglich Versandkostenpauschale bei folgender Firma bestellen:

Firma Milas
Inh. Milas Rajcic
Alte Linner Str. 38
4150 Krefeld 1
Tel. 02151/22420
Kennwort: Aufsteigerbuch

Bitte mit Angabe der Amiga-Version (500, 1000, 2000).

Beim Selbstbau liegen die Materialkosten bei etwa 15 DM. Folgende Bauteile werden benötigt:

Beim Amiga 500/2000: 1 Userport-Stecker, mit Gehäuse, 1 Meter bis 1,5 Meter Flachbandkabel, 11-adrig, 1 25-poliger Sub-D-Stecker (männlich), mit Gehäuse

Beim Amiga 1000: wie oben, nur statt des Sub-D-Steckers: 1 25-polige Sub-D-Buchse (weiblich), mit Gehäuse

Das Anschlußschema ist beim Amiga 500/2000 und beim Amiga 1000 der Numerierung nach gleich. Beachten Sie aber, daß die Kombination Buchse-Stecker vertauscht ist und daher der Amiga-1000-Port spiegelverkehrt zu dem des Amiga 500/2000 ist.

Die Pin-Numerierung des Userport-Steckers ist nicht immer einheitlich. Wenn Sie sich aber genau an die Abbildung halten, kann nichts schiefgehen.

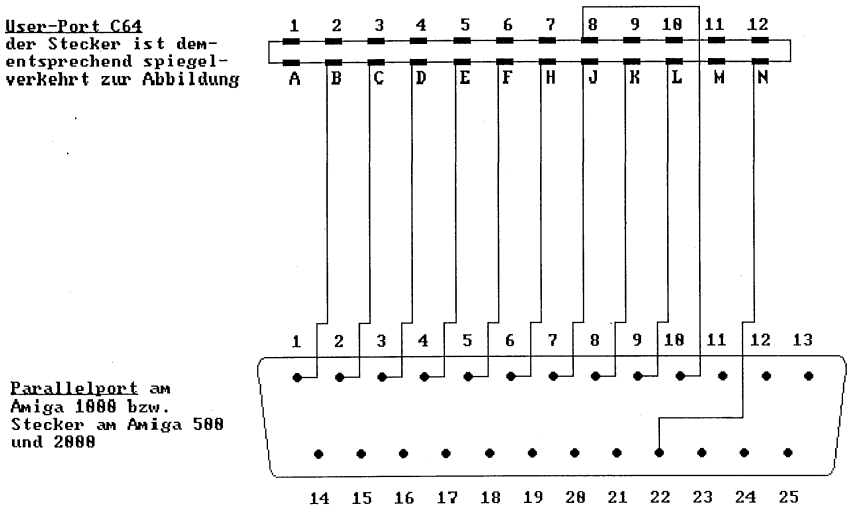

Abbildung 7.1:Anschlußschema

Verbinden Sie einfach die entsprechenden Pins der Stecker mit dem Flachbandkabel. Sie sollten auf dem Gehäuse des Userport-Steckers kenntlich machen, welche Seite nach oben gehört, damit es hier nicht zu Verwechslungen kommt. Diese könnten sich nämlich tödlich auf einige Chips auswirken.

Die folgende Erklärung der Funktionsweise des Kabels können Sie getrost überspringen, wenn Sie keine Lust haben, eigene Programme für das Übertragungskabel zu schreiben.

Die CIAs 6526 und 8520 von C64 und Amiga sind nahezu identisch aufgebaut. Dementsprechend besitzen beide je zwei 8-Bit-Datenports mit zugehörigen Datenrichtungsregistern und zwei Leitungen mit den Namen PC und Flag. Jeweils ein Datenport und die beiden Leitungen sind am Userport des C64 und am Parallelport des Amiga verfügbar. Gemäß obigem Anschlußschema werden die Datenports miteinander verbunden, außerdem PC vom Amiga mit Flag vom C64 und umgekehrt.

Die Datenübertragung geht nun folgendermaßen vor sich: Der C64 schaltet sein Datenrichtungsregister auf Ausgang, während

der Amiga seines auf Eingang schaltet. Wenn der C64 nun ein Datenbyte in das Port-Register schreibt, wird für einen Taktzyklus die Leitung PC auf null gesetzt. Die CIA im Amiga registriert das über die Flag-Leitung, wodurch das Bit 4 im Interrupt-Kontrollregister (ICR) gesetzt wird. Wenn der Amiga dieses Bit abfragt, kann er erkennen, daß nun Daten am Datenport anliegen. Der Lesezugriff auf das ICR löscht automatisch das Bit 4 wieder. Wenn der Amiga das Datenregister ausliest, passiert nun das Umgekehrte wie vorher. Die Leitung PC am Amiga wird für einen Taktzyklus null, was der C64 über seine Flag-Leitung und sein ICR registriert. Der C64 weiß so, daß der Amiga das Byte empfangen hat, und kann nun weitere Daten senden.

Hier nun die Adressen der CIA-Register beider Rechner:

Name	C64	Amiga
Datenport	$DD01	$BFE101
Datenrichtungsregister	$DD03	$BFE301
ICR	$DD0D	$BFED01

Der Kern der Übertragungsroutine beim C64 sieht so aus:

```
        STA $DD01   Datenbyte in Port-Register schreiben
        LDA #$10    Maske für Bit 4 laden
WAIT    BIT $DD0D   ICR testen
        BEQ WAIT    warten, bis Bit 4 gesetzt
```

Beim Amiga benötigt man eine solche Routine nicht, da das Betriebssystem den Parallelport automatisch richtig verwaltet. Das bedeutet, man kann einfach ein Daten-File mit dem Namen par: für die Eingabe öffnen und dann Daten aus diesem File lesen.

7.2 Welche Arten von Datenübertragung sind sinnvoll?

Wenn man eine Datenübertragung vom C64 zum Amiga betreiben will, muß man sich natürlich fragen, was das Aneinanderschließen der beiden Rechner einem bringt.

Beginnen wir mit einfachen Textdateien. Sicherlich ist es interessant für Sie, Texte, die Sie auf Ihrem C64 hatten, auf Ihrem Amiga weiterverwenden zu können. Insbesondere die Vorteile, die Ihnen die 80-Zeichen-Darstellung, der größere Speicher und der schnellere Prozessor bieten, sind nicht zu unterschätzen. Texte können aber nicht einfach Byte für Byte übertragen werden, sie müssen einer Umwandlung unterzogen werden. Der wichtigste Unterschied zwischen dem ASCII-Code von C64 und Amiga ist die Vertauschung von Klein- und Großbuchstaben. Wir haben für Sie ein Programm entwickelt, das diese Umwandlung vornimmt. Es läßt sich leicht an die Formate verschiedener Textverarbeitungen anpassen.

Ein weiteres Problem ist das Übertragen von Daten, die Sie mit einem Dateiververwaltungsprogramm erzeugt haben. Wenn Sie beispielsweise feststellen, daß Ihre Adreßkartei, die Sie mit Datamat oder Superbase erstellt haben, nicht mehr auf eine 5¼-Zoll-Diskette paßt, dann wäre es doch ganz praktisch, wenn man diese Daten nun auf eine 3½-Zoll-Diskette überspielen könnte, um sie dann mit Datamat Amiga oder Superbase Amiga weiterzuverarbeiten. Wir beschreiben Ihnen selbstverständlich, wie Sie mit unserem Übertragungsprogramm auch dieses Problem lösen.

Vielleicht besitzen Sie auch einfache BASIC-Programme ohne viele POKE-Befehle und Steuerzeichen. Unser Übertragungsprogramm ermöglicht Ihnen die leichte Umsetzung dieser Programme, sofern Sie sie auf Ihrem neuen Rechner nicht missen möchten.

Eine Sache, die ebenfalls für viele Leute von Interesse sein dürfte, ist das Übertragen von Grafik vom C64 zum Amiga. Der Amiga kann schließlich ohne weiteres die C64-Grafik simulie-

ren. Da wir eine Auswahl treffen mußten, welche Arten von C64-Grafiken wir konvertieren wollten, haben wir uns für folgende drei Grafikformate entschieden:

1. Multicolor-Grafiken im Koala-Format
2. Multicolor-Grafiken ohne Farbinformation
3. Hires-Bilder ohne Farbinformation

Diese drei Formate sind unserer Meinung nach die auf dem C64 am häufigsten verwendeten.

Wie Sie sehen, eröffnet ein solches Kabel ungeahnte Möglichkeiten. In Kapitel 8 beispielsweise werden wir darauf eingehen, wie Sie Ihren C64 als Drucker-Interface verwenden können. Wenn Sie noch eigene Ideen haben, was Sie mit dem Kabel anfangen können, dann zögern Sie nicht, diese in die Tat umzusetzen. Die in diesem Buch abgedruckten, vollständig dokumentierten Beispielprogramme sollten Ihnen dabei eine ausreichende Hilfe sein.

7.3 Das Übertragungsprogramm

Das Übertragungsprogramm besteht aus mehreren Teilen. Zuerst wäre da der C64-Teil, der das Übertragen von beliebigen Dateien ermöglicht. Dieser Teil kann unter anderem BASIC-Programme lesen, umsetzen und in ein für den Amiga sinnvolles Format bringen. Der zweite Teil der Software ist das Empfangsprogramm für den Amiga, das die Daten entgegennimmt und als Daten-Files auf der eingelegten Diskette abspeichert. Der dritte Teil ist ebenfalls ein Amiga-Programm, das es ermöglicht, C64-Grafiken in Amiga-Grafiken umzuwandeln und diese im IFF-Format abzulegen, so daß sie mit den meisten Malprogrammen eingelesen und verarbeitet werden können.

Nun aber zum eigentlichen Programm. Zuerst der C64-Teil, den Sie als normales BASIC-Programm eingeben können. REM-Zeilen und Anmerkungen können Sie getrost weglassen (bis auf den

copyright-Vermerk, den Sie fairerweise ruhig abtippen sollten). Es ist besser, vor der Eingabe mit SHIFT-COMMODORE den Klein-/Großschriftzeichensatz zu wählen.

```
10 rem ------------------------------
20 rem uebertragungsprogramm c64amiga
30 rem ------------------------------
40 rem entwickelt von michael strauch
50 rem ------------------------------
60 rem (c) 1987 data becker
70 rem ------------------------------
80 poke56579,255:poke56577,0:rem datenport initialisieren
85 poke828,1:rem vorgabe linefeed senden
90 dim n$(144),t(144):rem datenfeld für das directory
95 gosub3000:rem maschinenprogramm einlesen
100 gosub 2000:rem menue aufbauen
110 print:print:print
120 print" Bitte legen Sie eine Diskette"
130 print" mit zu uebertragenden Dateien ein"
140 print" und druecken Sie eine Taste!"
200 poke198,0:wait198,1:geta$
210 print chr$(147)
215 rem directory als eingabekanal setzen
220 open 1,8,0,"$":poke781,1:sys65478
230 print " Bitte waehlen Sie aus:":print
240 n=0:get#1,a$,a$
250 gosub1000:print:e=0:rem disknamen anzeigen
255 rem je einen filenamen anzeigen
260 fori=0to144:gosub1000:if e then print:goto400
265 ift=0thenprint:goto330
270 poke211,31:print "j/n? ";
280 poke198,0:wait198,1:geta$
290 ifa$="n"thenprint"n":goto330
300 ifa$<>"j"then280
310 print"j":n$(n)=n$:t(n)=t
320 n=n+1
330 next
400 close1
410 ifn=0then540
420 printchr$(147)
422 rem der reihe nach alle dateien uebertragen
425 fori=0ton-1
430 print "Datei "chr$(34)n$(i)chr$(34);
440 poke211,25:print "wird gelesen"
450 ift(i)=1thena$=",p,r":goto470
```

```
460 a$=",s,r"
465 rem datei eroeffnen
470 open1,8,2,n$(i)+a$:poke2,1
475 poke56579,255:du=peek(56589):rem handshake loeschen
480 forj=1tolen(n$(i)):rem alle zeichen des filenamens
485 rem zeichen senden
490 poke780,asc(mid$(n$(i),j,1)):sys 49152
500 next
505 poke2,4:poke 780,0:sys49152:rem null senden
510 poke2,c:sys 49155:rem datei senden
520 close1
530 next
540 print:print" Fertig!"
550 poke198,0:wait198,1:geta$
560 goto100
998 rem --------------------------
999 rem eine directoryzeile anzeigen
1000 poke781,1:sys65478:geta$,a$
1005 geta$,b$
1010 printasc(a$+chr$(0))+256*asc(b$+chr$(0))" ";
1020 geta$:printa$;:ifa$=chr$(32)then1020
1025 ifa$<>chr$(34)then1210
1030 n$=""
1040 geta$:printa$;
1050 ifa$<>chr$(34)thenn$=n$+a$:goto1040
1060 geta$:printa$;:ifa$=chr$(32)then1060
1070 ifa$="p"thent=1:goto1100
1080 ifa$="s"thent=2:goto1100
1090 t=0
1100 geta$:printa$;:ifa$<>""then1100
1200 sys65484:return
1210 geta$:printa$;:ifa$<>""then1210
1220 e=1:goto1200
1998 rem ---------------------------------------
1999 rem menue aufbauen und eingabe entgegennehmen
2000 printchr$(147)chr$(14)chr$(8)
2010 print"      Datenuebertragung C64Amiga"
2020 print:print
2030 print" Bitte waehlen Sie aus:"
2035 print
2036 print" 0. Konvertierung nach Tabelle"
2040 print" 1. ASCII-Dateien"
2050 print" 2. Textomat-Plus-Dateien"
2060 print" 3. BASIC-Programme"
2070 print" 4. Dateien ohne Konvertierung"
2080 print" 5. Linefeed senden?  ";
```

```
2090 ifpeek(828)=1thenprint"ja  ":goto2110
2100 print"nein"
2110 print" 6. Programmende"
2120 poke198,0:wait198,1:geta$
2130 ifa$<"0"ora$>"6"then2120
2140 ifa$="6"thenprintchr$(147);:poke56577,0:end
2142 ifa$="5"thenpoke828,1-peek(828):goto2000
2150 c=val(a$):ifc<>0then return
2160 print:tb$=""
2170 input "Tabellenname ";tb$:if tb$="" then return
2180 sys57812 tb$,8,1
2190 poke780,0:sys65493:if(191andst)<>0then2000
2200 return
2998 rem ------------------------
2999 rem maschinenprogramm einlesen
3000 ifpeek(49152)=76andpeek(49800)=194then return
3003 fori=1to751step15:forj=0to14:reada$:b$=right$(a$,1)
3005 a=asc(a$)-48:ifa>9thena=a-7
3010 b=asc(b$)-48:ifb>9thenb=b-7
3020 a=a*16+b:c=(c+a)and255:poke49151+i+j,a
3025 next:reada:ifc=athenc=0:next:return
3030 print"fehler in zeile:";peek(63)+peek(64)*256:stop
3100 data4c,2a,c2,a9,ff,8d,03,dd,ad,0d,dd,a9,00,85,90, 162
3101 data8d,3d,03,a2,01,20,c6,ff,a5,02,c9,03,f0,49,a9, 170
3102 data00,85,fd,20,f8,c1,aa,a5,02,d0,09,bd,00,cf,c9, 218
3103 dataff,f0,0e,d0,08,c9,02,d0,04,e0,a2,f0,04,8a,20, 148
3104 data1b,c2,a5,90,d0,06,a5,fd,c9,fe,d0,d9,a5,fd,20, 188
3105 data8b,c2,a2,00,bd,ef,c2,20,2a,c2,e8,e4,fd,d0,f5, 247
3106 dataa5,90,f0,bf,a9,00,20,8b,c2,4c,cc,ff,20,cf,ff, 255
3107 data20,cf,ff,a9,00,85,fe,85,fd,85,0f,20,cf,ff,aa, 200
3108 data20,cf,ff,a8,d0,06,8a,d0,03,4c,f0,c1,20,cf,ff, 180
3109 dataaa,20,cf,ff,85,62,86,63,a2,90,38,20,49,bc,20, 23
3110 datadf,bd,a2,00,bd,00,01,f0,06,20,1b,c2,e8,d0,f5, 156
3111 data86,22,86,23,20,cf,ff,d0,03,4c,bc,c1,85,fb,30, 139
3112 data5e,c9,25,f0,0c,c9,30,90,45,c9,3a,90,04,c9,3f, 181
3113 data90,3d,a9,01,85,22,a5,23,f0,09,a9,20,20,1b,c2, 165
3114 dataa9,00,85,23,a5,fb,20,1b,c2,a4,fd,b9,ed,c2,c9, 192
3115 data54,d0,11,b9,ee,c2,c9,49,d0,07,a9,01,85,fe,4c, 0
3116 dataa9,c0,b9,ed,c2,c9,53,d0,f6,b9,ee,c2,c9,54,f0, 41
3117 dataeb,d0,ed,c9,22,d0,06,a5,0f,49,ff,85,0f,a9,00, 162
3118 data85,22,4c,d2,c0,a5,0f,d0,ae,a5,fb,c9,ff,f0,68, 119
3119 datac9,9c,d0,03,4c,a4,c1,c9,aa,90,15,c9,af,90,08, 17
3120 datac9,b1,90,0d,c9,b4,b0,09,a9,00,85,23,85,22,4c, 145
3121 data4c,c1,a5,22,f0,05,a9,20,20,1b,c2,a9,01,85,23, 225
3122 data85,22,a5,fb,38,e9,7f,aa,a0,ff,ca,f0,08,c8,b9, 115
3123 data9e,a0,10,fa,30,f5,c8,b9,9e,a0,30,06,20,1b,c2, 95
```

```
3124 data4c,5f,c1,29,7f,20,1b,c2,a2,0c,a5,fb,dd,cc,c2, 202
3125 dataf0,05,ca,10,f8,30,04,a9,01,85,fe,4c,a9,c0,a5, 130
3126 data22,f0,05,a9,20,20,1b,c2,a2,00,bd,b4,c2,f0,06, 168
3127 data20,1b,c2,e8,d0,f5,a9,01,85,22,85,23,4c,a9,c0, 88
3128 dataa5,22,f0,05,a9,20,20,1b,c2,a2,00,bd,be,c2,f0, 81
3129 datae7,20,1b,c2,e8,d0,f5,f0,df,a5,0f,f0,05,a9,22, 212
3130 data20,1b,c2,a5,fe,f0,0d,a2,00,bd,c4,c2,f0,06,20, 152
3131 data1b,c2,e8,d0,f5,a9,0d,20,1b,c2,a5,fd,20,8b,c2, 76
3132 dataa2,00,bd,ef,c2,20,2a,c2,e8,e4,fd,d0,f5,4c,6c, 98
3133 datac0,a9,00,20,8b,c2,4c,cc,ff,a5,02,c9,02,b0,12, 33
3134 dataad,3c,03,f0,0d,ad,3d,03,f0,08,a9,00,8d,3d,03, 68
3135 dataa9,0a,60,20,cf,ff,c9,0d,d0,03,8d,3d,03,60,8c, 99
3136 data0e,03,a4,fd,99,ef,c2,c8,84,fd,ac,0e,03,60,85, 231
3137 datafc,a5,02,d0,0e,8e,0d,03,a6,fc,bd,00,cf,ae,0d, 8
3138 data03,4c,8b,c2,c9,04,f0,47,c9,02,f0,50,a5,fc,c9, 21
3139 data0d,f0,3f,c9,0a,f0,3b,c9,20,90,27,c9,41,90,33, 167
3140 datac9,5b,b0,05,69,20,4c,8b,c2,c9,61,90,26,c9,7b, 31
3141 datab0,05,e9,1f,4c,8b,c2,c9,c1,b0,0e,c9,a0,d0,05, 220
3142 dataa9,20,4c,8b,c2,a9,bf,4c,8b,c2,c9,e5,b0,f7,e9, 161
3143 data7f,4c,8b,c2,a5,fc,8d,01,dd,a9,10,2c,0d,dd,f0, 227
3144 datafb,60,a5,fc,8e,0d,03,a2,0a,dd,d9,c2,f0,09,ca, 129
3145 data10,f8,ae,0d,03,4c,46,c2,bd,e4,c2,ae,0d,03,4c, 135
3146 data8b,c2,33,2e,31,34,31,35,39,32,37,00,43,4c,45, 239
3147 data41,52,00,3a,52,45,4d,3f,3f,3f,00,9f,a1,cb,93, 12
3148 data94,95,92,9d,b8,9e,b7,97,c2,a5,a1,a6,a9,a8,aa, 165
3149 dataa7,a0,5f,23,0d,e4,f6,fc,c4,d6,dc,df,16,0a,a7, 200
3150 data0a,00,3b,3a,40,5d,5b,ba,3c,3e,59,5a,d9,da,2b, 60
```

Wenn Sie Dateien von speziellen Programmen wie Textomat oder Datamat übertragen wollen, benötigen Sie entsprechende Konvertierungstabellen, in denen verzeichnet ist, welches Zeichen gegen welches andere Zeichen ausgetauscht werden soll. Diese Tabellen liegen im Bereich $CF00 bis $CFFF im Speicher des C64. Bei $CF00 steht das Zeichen, das gegen CHR$(0) ausgetauscht werden soll, bei $CF01 der Ersatz für CHR$(1) usw. Dabei hat das Zeichen CHR$(255) (Hexadezimal: $FF) eine besondere Bedeutung: Es zeigt an, daß das entsprechende Zeichen weggelassen werden soll. Hier als Beispiel ein Programm, das Ihnen die Konvertierungstabelle für den deutschen Zeichensatz von Textomat 64 (nicht Textomat Plus) und Datamat 64 erstellt:

```
10 open1,8,15,"s:textomattab":close1
20 open1,8,2,"textomattab,p,w"
30 print#1,chr$(0);chr$(207);
100 fori=1to256step16:forj=0to15:reada$:b$=right$(a$,1)
105 a=asc(a$)-48:ifa>9thena=a-7
110 b=asc(b$)-48:ifb>9thenb=b-7
120 a=a*16+b:c=(c+a)and255:print#1,chr$(a);:next:reada
125 ifc=athenc=0:next:close1:end
130 print"fehler in zeile:";peek(63)+peek(64)*256:close1 :stop
300 dataff,ff,ff,ff,ff,ff,ff,ff,ff,ff,0a,ff,ff,0d,ff,ff, 9
301 dataff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff, 240
302 data20,21,22,a7,24,25,26,2f,28,29,2a,df,2c,2d,2e,2d, 182
303 data30,31,32,33,34,35,36,37,38,39,f6,e4,3b,3d,3a,3f, 216
304 datadc,61,62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6f, 244
305 data70,71,72,73,74,75,76,77,78,7a,79,d6,5a,c4,5e,0a, 99
306 dataff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff, 240
307 dataff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff, 240
308 dataff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff, 240
309 dataff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff, 240
310 data16,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff, 7
311 dataff,ff,ff,ff,ff,ff,ff,ff,ff,ff,fc,ff,ff,ff,ff,ff, 237
312 data2b,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f, 99
313 data50,51,52,53,54,55,56,57,58,5a,59,ff,ff,ff,ff,ff, 162
314 dataff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff, 240
315 dataff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff,ff, 240
```

Wenn Sie die Tabelle benutzen wollen, müssen Sie natürlich dieses Programm abtippen und starten, bevor Sie das eigentliche Übertragungsprogramm laden.

Auch beim Amiga erschien es uns am sinnvollsten, das Programm zum Datenempfang in ein BASIC-Programm zu packen, da nicht jeder einen Monitor oder Assembler besitzt. Um das folgende Programm abzutippen, müssen Sie also zuerst Ihren BASIC-Interpreter laden. Nach dem Programmstart sucht das Programm auf der Diskette im eingebauten Laufwerk nach den Dateien "dos.bmap" und "exec.bmap". Sie sollten daher vorher Ihre Amiga-Extras-Diskette eingelegt haben. Wenn Sie die beiden ".bmap"-Dateien jedoch vorher auf die Diskette, auf der Sie die empfangenen Dateien abspeichern wollen, überspielt haben, sollten Sie die beiden Library-Befehle in Zeile 9 und 12 ändern. Dadurch ersparen Sie sich das mehrfache Wechseln der Disketten.

Zum Abtippen ist noch zu sagen, daß die Zeilennummern nicht benötigt werden. Wir haben sie nur zur leichteren Orientierung eingebunden. Lassen Sie sie also ruhig weg. Sie sparen sich dadurch Tipparbeit und Speicherplatz.

```
1  REM "Empfangsroutine Amiga"
2  REM "entworfen von Michael Strauch"
3  REM "(C) 1987 Data Becker"
4
5  INPUT"Laufwerkskennung beim Speichern: ",disk$
6  PRINT
7  PRINT "Suche nach dos.bmap ..."
8  CHDIR "df0:"
9  LIBRARY "basicdemos/dos.library"
10  PRINT "Suche nach exec.bmap ..."
11  LIBRARY  "basicdemos/exec.library"
12  PRINT "Alles gefunden!"
13
14  REM "benötigte Betriebssystem-Funktionen deklarieren"
15  DECLARE FUNCTION xOpen&() LIBRARY
16  DECLARE FUNCTION xRead&() LIBRARY
17
18  REM "Kanal zum Parallelport eröffnen"
19  ModeOldfile&=1005&
20  file&=xOpen&(SADD("par:"+CHR$(0)),ModeOldfile&)
21
22  REM "eventuelle irrelevante Daten vom Port holen"
23  a$=CHR$(1)
24  WHILE a$<>CHR$(0)
25  GOSUB getpar
26  WEND
27
28  REM "Nullbytes lesen, bis erstes Datenbyte gefunden"
29  WHILE a$=CHR$(0)
30  GOSUB getpar
31  WEND
32
33  REM "Hauptschleife: jeweils eine Datei empfangen"
34  mainloop:
35  f$=""
36  WHILE a$<>CHR$(0)
37  IF a$="/" OR a$=":" THEN a$="_"
38  f$=f$+a$
39  GOSUB getpar
40  WEND
```

```
41
42  REM "Datei eröffnen"
43  CHDIR disk$
44  OPEN "o",#1,f$
45  PRINT "File ";f$;" wird eröffnet"
46
47  REM "Daten empfangen und in Datei schreiben"
48  loop:
49  GOSUB getpar
50  n&=ASC(a$)
51  b$=SPACE$(n&)
52  IF a<>1 THEN abbruch
53  IF n&=0 THEN weiter
54  a=xRead&(file&,SADD(b$),n&)
55  IF a<>n& THEN abbruch
56  PRINT#1,b$;
57  GOTO loop
58
59  REM "Datei schließen und nächstes File bearbeiten"
60  weiter:
61  CLOSE#1
62  GOSUB getpar
63  GOTO mainloop
64
65  REM "Programmende"
66  abbruch:
67  CLOSE #1
68  CALL xClose(file&)
69  END
70
71  REM "Zeichen holen"
72  getpar:
73  IF INKEY$="q" THEN RETURN abbruch
74  a=xRead&(file&,SADD(a$),1)
75  IF a<>1 OR INKEY$="q" THEN RETURN abbruch
76  RETURN
```

Den dritten Teil, ebenfalls ein BASIC-Listing für den Amiga, brauchen Sie nur abzutippen, wenn Sie auch Grafiken übertragen wollen. Auch hier können Sie sich die Zeilennummern schenken.

```
1  REM "Grafik-Konverter C64Amiga"
2  REM "Entwickelt von Michael Strauch"
3  REM "(C)1987 Data Becker"
4
5  REM "Disk-Verzeichnisse festlegen"
6  libdir$="df0:basicdemos"
7  loaddir$="df0:"
8  savedir$="df0:"
9  CLS
10  PRINT "Disk-Verzeichnisse:"
11  PRINT "'.bmap'-Dateien aus Verzeichnis ";libdir$
12  PRINT "C64-Bilder      aus Verzeichnis ";loaddir$
13  PRINT "Amiga-Bilder   nach Verzeichnis ";savedir$
14  PRINT
15  PRINT "Drücken Sie bitte eine Taste, falls alles stimmt."
16  PRINT "(Sonst Programm abbrechen und ändern.)"
17  WHILE INKEY$="":WEND
18
19  DIM bplane&(4),coltab64(15,2)
20
21  REM "Funktionen aus dos.library"
22  DECLARE FUNCTION xOpen&()   LIBRARY
23  DECLARE FUNCTION xRead&()   LIBRARY
24  DECLARE FUNCTION xWrite&()  LIBRARY
25  DECLARE FUNCTION Seek&()    LIBRARY
26  REM "xClose liefert keinen Wert zurück"
27
28  REM "Funktionen aus exec.library"
29  DECLARE FUNCTION AllocMem&() LIBRARY
30  DECLARE FUNCTION OldOpenLibrary&() LIBRARY
31  REM "FreeMem, CloseLibrary liefern keinen Wert zurück"
32
33  PRINT:PRINT "Suche nach .bmap-Dateien ... ";
34  CHDIR libdir$
35  LIBRARY "dos.library"
36  LIBRARY "exec.library"
37  PRINT "Gefunden."
38
39  REM "Maschinenprogramm einlesen"
40  ClearPublic=65537&
41  makepicSize&=11*41
42  makepic&=AllocMem&(makepicSize&,ClearPublic&)
43  RESTORE makepicdat
44  FOR i%=0 TO 40
45  s%=0
46  FOR j%=0 TO 10
```

```
47  READ a%
48  POKE makepic&+i%*11+j%,a%
49  s%=s%+a%
50  NEXT
51  READ chk%
52  IF s%<>chk% THEN
53  PRINT "Fehler in der Data-Zeile ";i%+1
54  FreeMem(makepic&,makepicSize&)
55  END
56  END IF
57  NEXT
58
59  REM "Farben festlegen"
60  RESTORE farbdat
61  FOR i=0 TO 3
62  READ r,g,b:PALETTE i,r,g,b
63  NEXT
64  farbdat: DATA 0,0,0, .9,.9,.8, .4,.3,.9, .9,.5,0
65  CLS
66
67  REM "Menü aufbauen"
68  schleife:
69  LINE (14,10)-(616,38),3,bf
70  LINE (14,10)-(616,38),1,b
71  LOCATE 3,11:COLOR 2,3
72  PRINT "Grafikwandler C64AMIGA entwickelt von Michael Strauch"
73  LOCATE 4,30
74  PRINT "(C) 1987 Data Becker"
75
76  LINE (160,50)-STEP(300,85),2,bf
77  LINE (160,50)-STEP(300,85),1,b
78  LOCATE 8,27
79  COLOR 1,0:PRINT "F1";:COLOR 1,2:PRINT" Koalabild"
80  LOCATE 9,27
81  COLOR 1,0:PRINT "F2";:COLOR 1,2:PRINT" 4-Farb-Bild 160*200"
82  LOCATE 10,27
83  COLOR 1,0:PRINT "F3";:COLOR 1,2:PRINT" 2-Farb-Bild 320*200"
84  LOCATE 13,25
85  PRINT"Optionen für erzeugtes Bild:"
86  LOCATE 15,25
87  COLOR 1,0:PRINT "F4";:COLOR 1,2:PRINT"Auflösung vertikal :"
88  LOCATE 16,25
89  COLOR 1,0:PRINT "F5";:COLOR 1,2:PRINT"Auflösung horizontal:"
90
91  hires%=0:interlace%=0
92
```

```
93  REM "Gewählte Auflösung anzeigen"
94  optionen:
95  LOCATE 15,50
96  IF hires%=0 THEN
97  COLOR 1,3:PRINT"320";:COLOR 1,2:PRINT" 640"
98  ELSE
99  COLOR 1,2:PRINT"320 ";:COLOR 1,3:PRINT"640"
100  END IF
101  LOCATE 16,50
102  IF interlace%=0 THEN
103  COLOR 1,3:PRINT"200";:COLOR 1,2:PRINT" 400"
104  ELSE
105  COLOR 1,2:PRINT"200 ";:COLOR 1,3:PRINT"400"
106  END IF
107
108  REM "Tastatur abfragen"
109  keywait:
110  t$=""
111  WHILE t$=""
112  t$=INKEY$
113  WEND
114  t%=ASC(t$)
115  IF t%=27 THEN
116  CALL FreeMem(makepic&,makepicSize&)
117  COLOR 1,0:CLS:END
118  END IF
119  IF t%<129 OR t%>133 THEN keywait
120  IF t%=132 THEN hires%=1-hires%:GOTO optionen
121  IF t%=133 THEN interlace%=1-interlace%:GOTO optionen
122  t%=t%-129
123
124  REM "Gewählten Menüpunkt markieren"
125  AREA (232,(t%+7)*8-1):AREA STEP(152,0)
126  AREA STEP (0,8):AREA STEP(-152,0)
127  COLOR 0:AREAFILL 1
128
129  REM "Eingabe des File-Namens"
130  LINE (138,140)-STEP(352,40),1,bf
131  LINE (138,140)-STEP(352,40),2,b
132  LOCATE 19,20:COLOR 3,1
133  PRINT "Bitte geben Sie den Namen des Bildes ein:"
134  LOCATE 20,20:COLOR 2,1
135  IF t%=0 THEN PRINT CHR$(191);
136  LINE INPUT bildname$
137  REM "Koala-File-Namen gesondert behandeln"
138  IF t%=0 THEN
```

```
139  bildname$=CHR$(191)+bildname$
140  IF LEN(bildname$)>15 THEN abbruch1
141  bildname$=bildname$+SPACE$(15-LEN(bildname$))
142  END IF
143
144  REM "File-Namen beim Speichern eingeben"
145  LOCATE 21,20:COLOR 3,1
146  PRINT "Name beim Speichern? RETURN: nur ansehen"
147  LOCATE 22,20:COLOR 2,1
148  LINE INPUT ILBMname$
149  LOCATE 23,25:COLOR 1,0
150
151  REM "Zu ladendes Bild suchen"
152  CHDIR loaddir$
153  bildname$=bildname$+CHR$(0)
154  fhandle&=xOpen&(SADD(bildname$),1005)
155  IF fhandle&=0 THEN abbruch1
156
157  REM "Wenn nötig, Farbspeicher reservieren"
158  IF t%=0 THEN  '"Koalabild "
159  farbstart&=AllocMem&(2001,ClearPublic&)
160  IF farbstart&=0 THEN
161  PRINT "Kein freier Speicher"
162  xClose(fhandle&)
163  WHILE INKEY$="":WEND
164  GOTO abbruch1
165  END IF
166
167  oldpos&=Seek&(fhandle&,8002,-1)
168  IF oldpos&=-1 THEN
169  zukurz: PRINT "Daten-File zu kurz !"
170  xClose(fhandle&)
171  WHILE INKEY$="":WEND
172  GOTO abbruch2
173  END IF
174
175  rlength&=xRead&(fhandle&,farbstart&,2001)
176  IF rlength&<2001 THEN zukurz
177  oldpos&=Seek&(fhandle&,2,-1)
178  IF oldpos&=-1 THEN abbruch2
179
180  ELSE   '"4-Farb- oder 2-Farb-Bild"
181  farbstart&=0
182  oldpos&=Seek&(fhandle&,8002,-1)
183  IF oldpos&=-1 THEN zukurz
184  oldpos&=Seek&(fhandle&,2,-1)
```

```
185  END IF
186
187  REM "Screen für Bild aufbauen"
188  breite%=320*(hires%+1)
189  hoehe%=200*(interlace%+1)
190  tiefe%=2^(2-t%)
191  modus%=2*interlace%+hires%+1
192  SCREEN 1,breite%,hoehe%,tiefe%,modus%
193  WINDOW 2,"C64-Grafik",(0,0)-(145,10),7,1
194  RESTORE coldat64
195  FOR i%=0 TO 15
196  FOR j%=0 TO 2
197  READ rgb%:coltab64(i%,j%)=rgb%/16
198  NEXT j%
199  PALETTE i%,coltab64(i%,0),coltab64(i%,1),coltab64(i%,2)
200  NEXT i%
201
202  REM "Umwandlungsroutine aufrufen"
203  GOSUB getscraddrs
204  dosbase&=OldOpenLibrary&(SADD("dos.library"+CHR$(0)))
205  CALL makepic&(dosbase&,fhandle&,t%,farbstart&,hires%, interlace%,
     VARPTR(bplane&(0)))
206  CloseLibrary(dosbase&)
207  xClose(fhandle&)
208  SOUND 2000,2:SOUND 1500,2
209  WHILE INKEY$="":WEND
210
211  REM "Wenn gewünscht, abspeichern"
212  IF ILBMname$<>"" THEN
213  CHDIR savedir$:GOSUB SaveILBM
214  IF saveError$<>"" THEN
215  WINDOW 1
216  PRINT saveError$
217  WHILE INKEY$="":WEND
218  END IF
219  END IF
220
221  abbruch2: IF t%=0 THEN CALL FreeMem(farbstart&,2001)
222  abbruch1: SCREEN CLOSE 1
223
224  LINE (138,140)-STEP(352,40),0,bf
225  GOTO schleife
226
227  SaveILBM:
228  REM "Abspeichern des aktuellen Screens. "
229  REM "Entnommen dem Programm 'SaveILBM'  "
```

```
230  REM "  von Carolyn Scheppner. Dieses    "
231  REM "  Programm befindet sich auf der   "
232  REM "  Amiga-Extras-Diskette. Es wurde  "
233  REM "  leicht modifiziert, da kein      "
234  REM "  Farbzyklus gebraucht wird.       "
235
236  ILBMname$ = ILBMname$ + CHR$(0)
237  fhandle& = xOpen&(SADD(ILBMname$),1006)
238  IF fhandle& = 0 THEN
239  saveError$ = "Ausgabedatei nicht erzeugbar."
240  GOTO Scleanup
241  END IF
242
243  REM - Pufferspeicherplatz reservieren
244  mybufsize& = 120
245  mybuf& = AllocMem&(mybufsize&,ClearPublic&)
246  IF mybuf& = 0 THEN
247  saveError$ = "Pufferspeicher nicht verfügbar."
248  GOTO Scleanup
249  END IF
250
251  cbuf& = mybuf&
252
253  REM - Adressen der Screen-Structures ermitteln
254  GOSUB getscraddrs
255
256  zero& = 0
257  pad%  = 0
258  aspect% = &HA0B
259
260  REM - Chunk-Längen berechnen
261  BMHDsize& = 20
262  CMAPsize& = (2^scrDepth%) * 3
263  CAMGsize& = 4
264  BODYsize& = (scrWidth%/8) * scrHeight% * scrDepth%
265  REM - FORMsize& = Chunk-Längen + 8 Bytes je Chunk-Header + "ILBM"
266  FORMsize& = BMHDsize&+CMAPsize&+CAMGsize&+BODYsize&+36
267
268  REM - FORM-Header schreiben
269  tt$ = "FORM"
270  wLen& = xWrite&(fhandle&,SADD(tt$),4)
271  wLen& = xWrite&(fhandle&,VARPTR(FORMsize&),4)
272  tt$ = "ILBM"
273  wLen& = xWrite&(fhandle&,SADD(tt$),4)
274
275  IF wLen& <= 0 THEN
```

```
276  saveError$ = "Schreibfehler beim FORM-Header."
277  GOTO Scleanup
278  END IF
279
280  REM - BMHD-Chunk schreiben
281  tt$ = "BMHD"
282  wLen& = xWrite&(fhandle&,SADD(tt$),4)
283  wLen& = xWrite&(fhandle&,VARPTR(BMHDsize&),4)
284  wLen& = xWrite&(fhandle&,VARPTR(scrWidth%),2)
285  wLen& = xWrite&(fhandle&,VARPTR(scrHeight%),2)
286  wLen& = xWrite&(fhandle&,VARPTR(zero&),4)
287  temp% = (256 * scrDepth%)
288  wLen& = xWrite&(fhandle&,VARPTR(temp%),2)
289  wLen& = xWrite&(fhandle&,VARPTR(zero&),4)
290  wLen& = xWrite&(fhandle&,VARPTR(aspect%),2)
291  wLen& = xWrite&(fhandle&,VARPTR(scrWidth%),2)
292  wLen& = xWrite&(fhandle&,VARPTR(scrHeight%),2)
293
294  IF wLen& <= 0 THEN
295  saveError$ = "Schreibfehler beim BMHD-Chunk."
296  GOTO Scleanup
297  END IF
298
299  REM - CMAP-Chunk schreiben
300  tt$ = "CMAP"
301  wLen& = xWrite&(fhandle&,SADD(tt$),4)
302  wLen& = xWrite&(fhandle&,VARPTR(CMAPsize&),4)
303
304  REM - IFF-Farbpalette aufbauen
305  FOR kk = 0 TO nColors% - 1
306  regTemp% = PEEKW(colorTab& + (2*kk))
307  POKE(cbuf&+(kk*3)),(regTemp% AND &HF00) / 16
308  POKE(cbuf&+(kk*3)+1),(regTemp% AND &HF0)
309  POKE(cbuf&+(kk*3)+2),(regTemp% AND &HF) * 16
310  NEXT
311
312  wLen& = xWrite&(fhandle&,cbuf&,CMAPsize&)
313
314  IF wLen& <= 0 THEN
315  saveError$ = "Schreibfehler beim CMAP-Chunk."
316  GOTO Scleanup
317  END IF
318
319  REM - CAMG-Chunk schreiben
320  tt$ = "CAMG"
321  wLen& = xWrite&(fhandle&,SADD(tt$),4)
```

```
322  wLen& = xWrite&(fhandle&,VARPTR(CAMGsize&),4)
323  vpModes& = PEEKW(sViewPort& + 32)
324  wLen& = xWrite&(fhandle&,VARPTR(vpModes&),4)
325
326  IF wLen& <= 0 THEN
327  saveError$ = "Schreibfehler beim CAMG-Chunk"
328  GOTO Scleanup
329  END IF
330
331  REM - BODY-Chunk schreiben (eigentliche Pixel-Daten)
332  tt$ = "BODY"
333  wLen& = xWrite&(fhandle&,SADD(tt$),4)
334  wLen& = xWrite&(fhandle&,VARPTR(BODYsize&),4)
335
336  scrRowBytes% = scrWidth% / 8
337  FOR rr = 0 TO scrHeight% -1
338  FOR pp = 0 TO scrDepth% -1
339  scrRow& = bplane&(pp)+(rr*scrRowBytes%)
340  wLen& = xWrite&(fhandle&,scrRow&,scrRowBytes%)
341  IF wLen& <= 0 THEN
342  saveError$ = "Schreibfehler beim BODY-Chunk."
343  GOTO Scleanup
344  END IF
345  NEXT
346  NEXT
347
348  saveError$ = ""
349
350  Scleanup:
351  IF fhandle& <> 0 THEN CALL xClose&(fhandle&)
352  IF mybuf& <> 0 THEN CALL FreeMem&(mybuf&,mybufsize&)
353  RETURN
354
355  getscraddrs:
356  REM - Adressen der Screen-Structures ermitteln
357  sWindow&   = WINDOW(7)
358  sScreen&   = PEEKL(sWindow& + 46)
359  sViewPort& = sScreen& + 44
360  sRastPort& = sScreen& + 84
361  sColorMap& = PEEKL(sViewPort& + 4)
362  colorTab&  = PEEKL(sColorMap& + 4)
363  sBitMap&   = PEEKL(sRastPort& + 4)
364
365  REM - Screen-Parameter ermitteln
366  scrWidth%  = PEEKW(sScreen& + 12)
367  scrHeight% = PEEKW(sScreen& + 14)
```

```
scrDepth%  = PEEK(sBitMap& + 5)
nColors%   = 2^scrDepth%

REM - Adressen der Bitplanes ermitteln
FOR kk = 0 TO scrDepth% - 1
bplane&(kk) = PEEKL(sBitMap&+8+(kk*4))
NEXT
Return

REM "Daten für C64-Farben"
coldat64:
DATA 0,0,0, 15,15,15, 8,2,3, 9,15,15
DATA 15,9,15, 6,11,6, 2,3,10, 12,12,5
DATA 13,8,4, 6,4,0, 13,6,8, 4,4,4
DATA 8,8,8, 11,15,11, 7,8,15, 11,11,11

REM "Maschinenprogramm"
makepicdat:
DATA  72,231, 60, 34, 36,111,  0, 40, 65,250,  1, 900
DATA 140, 16,170,  7,208, 34, 47,  0, 44, 65,250, 981
DATA   1,148, 16,129, 32, 60,  0,  0,  0, 40,227, 653
DATA 160, 65,250,  1,130, 32,128, 65,250,  1,129, 1211
DATA  16,175,  0, 51, 44,111,  0, 28, 32, 47,  0, 504
DATA  36, 65,250,  1,118, 16,128, 12,  0,  0,  2, 628
DATA 102,  4,112,  1, 96,  2,112,  2,227, 32, 65, 755
DATA 250,  1,102, 16,128, 66, 65, 96,  6,  0, 60, 790
DATA   0, 16,226, 81, 81,200,255,248, 65,250,  1, 1423
DATA  80, 48,129,116,  0,122, 24,120, 39, 16, 58, 752
DATA   1, 66,102,  0,  0, 30, 16, 26, 18,  0,232, 491
DATA   9, 65,250,  1, 27, 16,193,  2,  0,  0, 15, 578
DATA  16,192, 16, 42,  3,231,  2,  0,  0, 15, 16, 533
DATA 128,118,  7, 72,231,112,  0, 34, 47,  0, 44, 793
DATA  65,250,  1, 21, 36,  8,118,  1, 78,174,255, 1007
DATA 214, 76,223,  0, 14, 12,128,  0,  0,  0,  1, 668
DATA 102, 52, 97,  0,  0, 56, 32, 58,  0,238, 18, 653
DATA  58,  0,239,227,160,212,128, 81,203,255,206, 1769
DATA 231,128,148,128,  6,130,  0,  0,  0,  1,114, 886
DATA   0, 18, 58,  0,214,212,129, 81,204,255,146, 1317
DATA 212,128,148,186,  0,198, 81,205,255,134, 76, 1623
DATA 223, 68, 60, 78,117, 72,231, 31, 32, 16, 58, 986
DATA   0,190, 65,250,  0,164, 66,152, 66,144, 54, 1151
DATA  58,  0,180, 28, 58,  0,178, 12,  0,  0,  2, 516
DATA 102,  4,120,  7, 96,  2,120,  3, 67,250,  0, 771
DATA 159, 26, 17, 66, 65,227,  5,227, 81, 12,  0, 885
DATA   0,  2,103,  4,227,  5,227, 81, 18,133, 74, 874
DATA   0,102,  8, 18, 59, 16,104,122,  3, 96,  4, 532
```

```
414 DATA 122,  2,154,128, 62,  5, 65,250,  0, 94,226, 1108
415 DATA  9,100,  2,135, 80, 84, 72, 81,205,255,246, 1269
416 DATA 236,107, 81,204,255,198, 18, 58,  0, 89, 32, 1278
417 DATA 111,  0, 80, 67,250,  0, 64, 58,  7, 36, 88, 761
418 DATA  16, 58,  0, 72,103,  6, 53,153, 40,  0, 96, 597
419 DATA  10, 21,153, 40,  0,211,252,  0,  0,  0,  1, 688
420 DATA  81,205,255,230,  4,  1,  0,  1,102,  6,212, 1097
421 DATA 186,  0, 36, 96,206, 16, 58,  0, 35,103,  4, 740
422 DATA 148,186,  0, 24, 76,223,  4,248, 78,117,  0, 1104
423 DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
424 DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
425 DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
426 DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
```

In den folgenden Kapiteln beschreiben wir Ihnen, wie Sie mit der Übertragungssoftware arbeiten. Innerhalb dieses Kapitels erschien es uns aber noch angebracht, Ihnen die Maschinenprogramme in dokumentierter Form zur Verfügung zu stellen. Diese brauchen Sie natürlich nicht noch einmal abzutippen.

Zuerst die Senderoutine für den C64:

Maschinenprogramm C64

verwendete Speicheradressen

$0002	Übertragungsmodus
$000F	Flag für Anführungsstriche
$0022	Flag für "Leerzeichen vor BASIC-Befehl einfügen"
$0023	Flag für "Leerzeichen nach BASIC-Befehl einfügen"
$00FB/$00FC	Zwischenspeicher für gelesenes Zeichen
$00FD	Bufferzeiger
$00FE	Flag für ":REM??? einfügen"
$033C	Flag für "LF einfügen"
$033D	Flag für "Return vorausgegangen"
$C307-$C406	Buffer für zu sendende Zeichen
$CF00-$CFFF	Konvertierungstabelle

```
C000 JMP $C22A      Einsprung für "Zeichen senden"
C003 LDA #$FF
C005 STA $DD03      Datenport auf Ausgang stellen
C008 LDA $DD0D      Bits im ICR löschen
```

```
C00B LDA #$00
C00D STA $90          Status löschen
C00F STA $033D        Flag für "CR gelesen" löschen
C012 LDX #$01         File #1
C014 JSR $FFC6        als Eingabe-File setzen
C017 LDA $02          Übertragungsmodus testen
C019 CMP #$03         BASIC-Programm?
C01B BEQ $C066        verzweige, wenn ja
C01D LDA #$00
C01F STA $FD          Bufferzeiger auf null setzen
C021 JSR $C1F8        Zeichen aus Datei holen
C024 TAX              in Index-Register für Tabellenzugriff
C025 LDA $02          Übertragungsmodus testen
C027 BNE $C032        verzweige, wenn nicht Modus 0 (Tabelle)
C029 LDA $CF00,X      konvertiertes Zeichen holen
C02C CMP #$FF         Zeichen auslassen?

C02E BEQ $C03E        verzweige, wenn ja
C030 BNE $C03A        sonst Zeichen in Buffer schreiben
C032 CMP #$02         Übertragungsmodus = Textomat Plus?
C034 BNE $C03A        verzweige, wenn nein
C036 CPX #$A2         Trennungsstrich?
C038 BEQ $C03E        Zeichen auslassen, wenn ja
C03A TXA              Zeichen holen
C03B JSR $C21B        und in Buffer schreiben
C03E LDA $90          Dateiende erreicht
C040 BNE $C048        Buffer senden, wenn ja
C042 LDA $FD          Bufferzeiger testen
C044 CMP #$FE         schon 254 Zeichen im Buffer?
C046 BNE $C021        verzweige, wenn nein
C048 LDA $FD          Bufferzeiger holen
C04A JSR $C28B        und senden
C04D LDX #$00         Schleifenindex auf null
C04F LDA $C2EF,X      Zeichen aus Buffer holen
C052 JSR $C22A        Zeichen konvertieren und senden
C055 INX              Schleifenindex erhöhen
C056 CPX $FD          schon kompletten Buffer gesendet?
C058 BNE $C04F        zum Schleifenanfang, wenn nein
C05A LDA $90          Deteiende testen
C05C BEQ $C01D        verzweige, wenn noch nicht erreicht
C05E LDA #$00         Null als Endekennzeichen
C060 JSR $C28B        senden
C063 JMP $FFCC        CLRCH: Ein-/Ausgabe zurücksetzen
```

BASIC-Programm senden

```
C066 JSR $FFCF      Anfangsadresse
C069 JSR $FFCF      überspringen
C06C LDA #$00       Variablen auf null setzen:
C06E STA $FE        REM???-Flag
C070 STA $FD        Bufferzeiger
C072 STA $0F        Anführungszeichen-Flag
C074 JSR $FFCF      Linkadresse LOW holen
C077 TAX            und merken
C078 JSR $FFCF      Linkadresse HIGH holen
C07B TAY            und testen
C07C BNE $C084      verzweige, wenn noch nicht Programmende
C07E TXA            Linkadresse LOW testen
C07F BNE $C084      verzweige, wenn noch nicht Programmende
C081 JMP $C1F0      verzweige, wenn Programmende
C084 JSR $FFCF      Zeilennummer LOW holen
C087 TAX            und merken
C088 JSR $FFCF      Zeilennummer HIGH holen
C08B STA $62        Zeilennummer für
C08D STX $63        Umwandlung speichern
C08F LDX #$90       Exponent für Umwandlung
C091 SEC            Übertrag-Flag setzen
C092 JSR $BC49      Umwandlung IntegerFließkomma
C095 JSR $BDDF      Umwandlung FließkommaASCII
C098 LDX #$00       Stringzeiger auf null setzen
C09A LDA $0100,X    ein Zeichen der Zeilennummer holen
C09D BEQ $C0A5      verzweige, wenn String zu Ende
C09F JSR $C21B      Zeichen senden
C0A2 INX            Stringzeiger erhöhen
C0A3 BNE $C09A      zum Schleifenanfang
C0A5 STX $22        Flags für "Leerzeichen vor und nach
C0A7 STX $23        BASIC-Befehl einfügen" löschen
C0A9 JSR $FFCF      BASIN: Zeichen einlesen
C0AC BNE $C0B1      weiter, wenn noch nicht Zeilenende
C0AE JMP $C1BC      verzweige, wenn Zeilenende
C0B1 STA $FB        Zeichen zwischenspeichern
C0B3 BMI $C113      verzweige, wenn BASIC-Token
C0B5 CMP #$25       "%"-Zeichen?
C0B7 BEQ $C0C5      wenn ja, verzweige zu "trennen"
C0B9 CMP #$30       Zeichennummer kleiner als 48 ("0")?
C0BB BCC $C102      wenn ja, verzweige zu "nicht trennen"
C0BD CMP #$3A       Zeichennummer größer als 57 ("9")?
C0BF BCC $C0C5      wenn ja, verzweige zu "trennen"
C0C1 CMP #$3F       Zeichennummer kleiner als 63 ("?")?
C0C3 BCC $C102      wenn ja, verzweige zu "nicht trennen"
```

```
C0C5 LDA #$01        "trennen": Flag für "BASIC-Befehl trennen"
C0C7 STA $22         setzen
C0C9 LDA $23         Test auf vorhergehenden BASIC-Befehl
C0CB BEQ $C0D6       verzweige, wenn nein
C0CD LDA #$20        Leerzeichen zum Trennen
C0CF JSR $C21B       in Buffer schreiben
C0D2 LDA #$00        Flag für "vorhergehender BASIC-Befehl"
C0D4 STA $23         löschen
C0D6 LDA $FB         Zeichen holen
C0D8 JSR $C21B       Zeichen in Buffer schreiben
C0DB LDY $FD         Bufferzeiger holen
C0DD LDA $C2ED,Y     vorhergehendes Zeichen holen
C0E0 CMP #$54        ="T"? (von "TI")
C0E2 BNE $C0F5       verzweige, wenn nein
C0E4 LDA $C2EE,Y     aktuelles Zeichen holen
C0E7 CMP #$49        ="I"?
C0E9 BNE $C0F2       verzweige, wenn nein
C0EB LDA #$01        Flag für ":REM??? einfügen"
C0ED STA $FE         setzen
C0EF JMP $C0A9       nächstes Zeichen einlesen
C0F2 LDA $C2ED,Y     vorhergehendes Zeichen holen
C0F5 CMP #$53        ="S"? (von "ST")
C0F7 BNE $C0EF       wenn nein, nächstes Zeichen einlesen
C0F9 LDA $C2EE,Y     aktuelles Zeichen holen
C0FC CMP #$54        ="T"?
C0FE BEQ $C0EB       wenn ja, REM???-Flag setzen
C100 BNE $C0EF       wenn nein, nächstes Zeichen einlesen
C102 CMP #$22        "nicht trennen": Anführungszeichen?
C104 BNE $C10C       verzweige, wenn nein
C106 LDA $0F         Anführungszeichen-Flag holen,
C108 EOR #$FF        invertieren
C10A STA $0F         und zurückschreiben
C10C LDA #$00        Flag für "Trennen vor BASIC-Befehl"
C10E STA $22         löschen
C110 JMP $C0D2       weiter bei "Zeichen in Buffer schreiben"
C113 LDA $0F         Anführungszeichen gesetzt?
C115 BNE $C0C5       wenn ja, Zeichen nicht als Token behandeln
C117 LDA $FB         Zeichen holen
C119 CMP #$FF        PI?
C11B BEQ $C185       verzweige, wenn ja
C11D CMP #$9C        CLR?
C11F BNE $C124       verzweige, wenn nein
C121 JMP $C1A4       verzweige, wenn ja
C124 CMP #$AA        Token "+", "-", "*", "/", "^"?
C126 BCC $C13D       verzweige, wenn nein
C128 CMP #$AF;
```

```
C12A BCC $C134      verzweige, wenn ja
C12C CMP #$B1       Token ">", "=", "<"?
C12E BCC $C13D      verzweige, wenn nein
C130 CMP #$B4;
C132 BCS $C13D      verzweige, wenn nein
C134 LDA #$00       Trenn-Flags löschen:
C136 STA $23        Flag für "nach Befehl trennen"
C138 STA $22        Flag für "vor Befehl trennen"
C13A JMP $C14C      weiter bei "Befehl in Buffer schreiben"
C13D LDA $22        Trenn-Flag holen
C13F BEQ $C146      verzweige, wenn null
C141 LDA #$20       Leerzeichen zum Trennen
C143 JSR $C21B      in Buffer schreiben
C146 LDA #$01       Trenn-Flags setzen:
C148 STA $23        Flag für "nach Befehl trennen"
C14A STA $22        Flag für "vor Befehl trennen"
C14C LDA $FB        Token holen
C14E SEC            Übertrag für Subtraktion setzen
C14F SBC #$7F       Wert in Bereich von null bis 127 holen
C151 TAX            und ins X-Register schieben
C152 LDY #$FF       Y ist Zeiger auf Befehlsworttabelle
C154 DEX            Tokenzähler verringern
C155 BEQ $C15F      verzweige, wenn Befehlswort erreicht
C157 INY            Zeiger erhöhen,
C158 LDA $A09E,Y    solange Wert aus Befehlsworttabelle
C15B BPL $C157      positiv ist
C15D BMI $C154      verzweige, wenn Wortende
C15F INY            Y auf Beginn des Befehlswortes setzen
C160 LDA $A09E,Y    ein Zeichen des Wortes holen
C163 BMI $C16B      Abbruch, wenn Wortende erreicht
C165 JSR $C21B      Zeichen in Buffer schreiben
C168 JMP $C15F      nächstes Zeichen holen
C16B AND #$7F       letztes Zeichen: Bit 7 löschen
C16D JSR $C21B      Zeichen in Buffer schreiben
C170 LDX #$0C       Zeiger auf zu markierende Tokens
C172 LDA $FB        Token holen,
C174 CMP $C2CC,X    mit Wert aus Tabelle vergleichen
C177 BEQ $C17E      verzweige, wenn gleich
C179 DEX            Zeiger verringern
C17A BPL $C174      verzweige, wenn noch nicht Tabellenende
C17C BMI $C182      verzweige, wenn nur erlaubte Tokens
C17E LDA #$01       Flag für REM???
C180 STA $FE        setzen
C182 JMP $C0A9      nächstes Zeichen lesen
```

Bearbeitung von PI

```
C185 LDA $22          Trenn-Flag testen
C187 BEQ $C18E        verzweige, wenn null
C189 LDA #$20         Leerzeichen zum Trennen
C18B JSR $C21B        in Buffer schreiben
C18E LDX #$00         Zeiger auf "3.1415927" setzen
C190 LDA $C2B4,X      ein Zeichen holen
C193 BEQ $C19B        verzweige, wenn Zahl zu Ende
C195 JSR $C21B        in Buffer schreiben
C198 INX              Zeiger erhöhen
C199 BNE $C190        nächstes Zeichen
C19B LDA #$01         Flags für
C19D STA $22          "vor" und "nach Befehl trennen"
C19F STA $23          setzen
C1A1 JMP $C0A9        nächstes Zeichen lesen
```

Bearbeitung von CLR

```
C1A4 LDA $22          Trenn-Flag testen
C1A6 BEQ $C1AD        verzweige, wenn null
C1A8 LDA #$20         Leerzeichen zum Trennen
C1AA JSR $C21B        in Buffer schreiben
C1AD LDX #$00         Zeiger auf "CLEAR" setzen
C1AF LDA $C2BE,X      ein Zeichen holen
C1B2 BEQ $C19B        verzweige, wenn Text zu Ende
C1B4 JSR $C21B        in Buffer schreiben
C1B7 INX              Zeiger erhöhen
C1B8 BNE $C1AF        nächstes Zeichen
C1BA BEQ $C19B        weiter wie bei PI
```

Zeilenende bearbeiten

```
C1BC LDA $0F          Anführungszeichen-Flag gesetzt?
C1BE BEQ $C1C5        verzweige, wenn nein
C1C0 LDA #$22         Anführungszeichen
C1C2 JSR $C21B        in Buffer schreiben
C1C5 LDA $FE          REM???-Flag testen
C1C7 BEQ $C1D6        verzweige, wenn nicht gesetzt
C1C9 LDX #$00         Zeiger auf ":REM???" setzen
C1CB LDA $C2C4,X      Zeichen holen
C1CE BEQ $C1D6        verzweige, wenn Text zu Ende
C1D0 JSR $C21B        in Buffer schreiben
C1D3 INX              Zeiger erhöhen
```

```
C1D4 BNE $C1CB          nächstes Zeichen
C1D6 LDA #$0D           Return am Zeilenende
C1D8 JSR $C21B          in Buffer schreiben
C1DB LDA $FD            Bufferzeiger holen
C1DD JSR $C28B          Byte senden
C1E0 LDX #$00           Bufferindex auf null setzen
C1E2 LDA $C2EF,X        Zeichen aus Buffer holen
C1E5 JSR $C22A          Zeichen mit Konvertierung senden
C1E8 INX                Index erhöhen
C1E9 CPX $FD            schon kompletten Buffer gesendet?
C1EB BNE $C1E2          verzweige, wenn nein
C1ED JMP $C06C          nächste BASIC-Zeile abarbeiten
```

Dateiende

```
C1F0 LDA #$00           Null als Endekennzeichen
C1F2 JSR $C28B          senden
C1F5 JMP $FFCC          Eingabegerät wieder Tastatur
```

Zeichen lesen und eventuell Linefeed einfügen

```
C1F8 LDA $02            Übertragungsmodus testen
C1FA CMP #$02           ASCII-Dateien?
C1FC BCS $C210          verzweige, wenn nein
C1FE LDA $033C          Flag für "LF einfügen" testen
C201 BEQ $C210          verzweige, wenn null
C203 LDA $033D          Flag für "Return vorausgegangen" testen
C206 BEQ $C210          verzweige, wenn null
C208 LDA #$00           Flag wieder
C20A STA $033D          löschen
C20D LDA #$0A           LF laden
C20F RTS                Rücksprung
C210 JSR $FFCF          BASIN: Zeichen aus Datei lesen
C213 CMP #$0D           Return?
C215 BNE $C21A          verzweige, wenn nein
C217 STA $033D          Flag für "Return vorausgegangen" setzen
C21A RTS                Rücksprung
```

Zeichen in Buffer schreiben

```
C21B STY $030E          Y-Register retten
C21E LDY $FD            Bufferzeiger laden
C220 STA $C2EF,Y        Zeichen in Buffer schreiben
```

```
C223 INY          Bufferzeiger erhöhen
C224 STY $FD      und zurückschreiben
C226 LDY $030E    Y-Register zurückholen
C229 RTS          Rücksprung
```

Zeichen konvertieren und senden

```
C22A STA $FC      Zeichen zwischenspeichern
C22C LDA $02      Konvertierungsmodus testen
C22E BNE $C23E    verzweige, wenn nicht nach Tabelle
C230 STX $030D    X-Register retten
C233 LDX $FC      Zeichen als Index auf Tabelle holen
C235 LDA $CF00,X  konvertiertes Zeichen holen
C238 LDX $030D    X-Register zurückholen
C23B JMP $C28B    Zeichen senden
C23E CMP #$04     keine Konvertierung?
C240 BEQ $C289    wenn ja, Zeichen senden
C242 CMP #$02     Textomat-Plus-Dateien?
C244 BEQ $C296    verzweige, wenn ja
C246 LDA $FC      Zeichen holen
C248 CMP #$0D     Return?
C24A BEQ $C28B    wenn ja, senden
C24C CMP #$0A     Linefeed?
C24E BEQ $C28B    wenn ja, senden
C250 CMP #$20     Zeichencode kleiner als 32?
C252 BCC $C27B    wenn ja, nicht senden
C254 CMP #$41     Zeichencode kleiner als 65?
C256 BCC $C28B    wenn ja, senden
C258 CMP #$5B     Zeichencode kleiner 91?
C25A BCS $C261    verzweige, wenn nein
C25C ADC #$20     Klein-Groß-Umwandlung
C25E JMP $C28B    Zeichen senden
C261 CMP #$61     Zeichencode kleiner als 97?
C263 BCC $C28B    wenn ja, senden
C265 CMP #$7B     Zeichencode kleiner als 123?
C267 BCS $C26E    verzweige, wenn nein
C269 SBC #$1F     Klein-Groß-Umwandlung
C26B JMP $C28B    Zeichen senden
C26E CMP #$C1     Zeichencode kleiner als 193?
C270 BCS $C280    verzweige, wenn nein
C272 CMP #$A0     Shift-SPACE?
C274 BNE $C27B    wenn nein, Zeichen nicht senden
C276 LDA #$20     Shift-SPACE durch Leerzeichen ersetzen
C278 JMP $C28B    Zeichen senden
C27B LDA #$BF     umgekehrtes Fragezeichen als Ersatz
```

```
C27D JMP $C28B        für alle unbekannten Zeichen senden
C280 CMP #$E5         Zeichencode kleiner 229?
C282 BCS $C27B        verzweige, wenn nein
C284 SBC #$7F         Klein-Groß-Umwandlung
C286 JMP $C28B        Zeichen senden
C289 LDA $FC          Zeichen holen
C28B STA $DD01        Zeichen auf Datenport legen
C28E LDA #$10         Maske für Bit 4 laden
C290 BIT $DD0D        Bit 4 im IFR testen
C293 BEQ $C290        verzweige, wenn Bit gelöscht
C295 RTS              Rücksprung
```

Textomat-Plus-Routine

```
C296 LDA $FC          Zeichen holen
C298 STX $030D        X-Register retten
C29B LDX #$0A         Zähler für Vertauschungstabelle setzen
C29D CMP $C2D9,X      Test auf Umlaut oder ähnliches Zeichen
C2A0 BEQ $C2AB        verzweige, wenn Test positiv
C2A2 DEX              Zähler verringern
C2A3 BPL $C29D        verzweige, wenn noch nicht Tabellenende
C2A5 LDX $030D        X-Register zurückholen
C2A8 JMP $C246        weiter wie bei Standard-ASCII-Dateien
C2AB LDA $C2E4,X      Umlaut-Ersatz holen
C2AE LDX $030D        X-Register zurückholen
C2B1 JMP $C28B        Zeichen senden
```

```
C2B4 33 2E 31 34 31 35 39 32  diverse Texte und
C2BC 37 00 43 4C 45 41 52 00  Tabellen:
C2C4 3A 52 45 4D 3F 3F 3F 00  Text für PI, CLR, REM???
C2CC 9F A1 CB 93 94 95 92 9D  Tokens für REM???
C2D4 B8 9E B7 97 C2 A5 A1 A6  Umlaute
C2DC A9 A8 AA A7 A0 5F 23 0D
C2E4 E4 F6 FC C4 D6 DC DF 16
C2EC 0A A7 0A 34 30 2C 35 44
```

Und schließlich die Grafik-Konvertierroutine für den Amiga. Die im Source-Listing verwendete Notation wird zwar nicht von allen Assemblern verstanden, ist aber soweit eindeutig, und Sie sollten damit keinerlei Verständnisschwierigkeiten haben.

```
sdosbase        =28                         ;Offsets für Parameter vom
                                            ;Stapel
sfhandle        =32
smodus          =36
sfarbstart      =40
shires          =44
sinterlace      =48
sbptab          =52

Read            =-42                        ;Routine aus dos.library


start:      movem.l d2/d3/d4/d5/a2/a6,-(sp) ;Register auf Stapel retten

            move.l sfarbstart(sp),a2        ;Adresse der Farbtabelle
            lea Farbe0(pc),a0               ;Zeiger auf Farbe 0 setzen
            move.b 2000(a2),(a0)            ;Farbwert holen

            move.l shires(sp),d1            ;Flag für HIRES holen
            lea hires(pc),a0
            move.b d1,(a0)                  ;wegspeichern

            move.l #40,d0                   ;Bytes pro Zeile auf 40
                                            ;setzen
            asl.l d1,d0                     ;verdoppeln, wenn HIRES
            lea linelength(pc),a0
            move.l d0,(a0)                  ;wegspeichern

            lea interlace(pc),a0
            move.b sinterlace+3(sp),(a0)    ;Interlace-Flag vom Stapel
                                            ;holen

            move.l sdosbase(sp),a6          ;DOS-Basisadresse vom Stapel
                                            ;holen

            move.l smodus(sp),d0            ;Bildart vom Stapel holen
            lea modus(pc),a0
            move.b d0,(a0)                  ;wegspeichern

            cmp.b #2,d0                     ;C64-Bild 320 * 200?
            bne.s l5                        ;verzweige, wenn nein
            moveq #1,d0                     ;ein Amiga-Punkt pro C64-
                                            ;Pixel
            bra.s l6
l5:         moveq #2,d0                     ;zwei Punkte pro Pixel
l6:         asl.b d1,d0                     ;verdoppeln, wenn HIRES
```

```
            lea pperpix(pc),a0
            move.b d0,(a0)              ;Punkte pro Pixel
                                        ;wegspeichern

            clr.w d1                    ;Speicher für Punktmaske
                                        ;löschen
            bra.s l7
l8:         or.b #$10,sr                ;x-Flag setzen
            roxr.w #1,d1                ;in d1 schieben
l7:         dbf d0,l8                   ;verzweige, wenn Maske noch
                                        ;nicht fertig
            lea pmaske(pc),a0
            move.w d1,(a0)              ;Maske speichern

            moveq #0 ,d2                ;Offset für jede Bitplane
            moveq #24,d5                ;Zähler: 25 Reihen a acht
                                        ;Byte
loop1:      moveq #39,d4                ;Zähler: 40 Spalten mit je
                                        ;acht Punkten
loop2:      move.b modus(pc),d0         ;Koala-Bild?
            bne l1                      ;verzweige, wenn nein
            move.b (a2)+,d0             ;Byte aus Farbtabelle holen
            move.b d0,d1                ;zwischenspeichern
            lsr.b #4,d1                 ;obere Bytehälfte holen
            lea Farbe1(pc),a0
            move.b d1,(a0)+             ;Farbe 1 wegspeichern
            and.b #15,d0                ;untere Bytehälfte holen
            move.b d0,(a0)+             ;Farbe 2 wegspeichern
            move.b 999(a2),d0           ;Byte aus zweiter Farb-
                                        ;tabelle holen
            and.b #15,d0                ;untere Bytehälfte
                                        ;isolieren
            move.b d0,(a0) ;Farbe 3     ;Farbe 3 wegspeichern

l1:         moveq #7,d3                 ;Zähler: 8 Bytes pro
                                        ;Cursorfeld
loop3:      movem.l d1/d2/d3,-(sp)      ;Datenregister retten
            move.l sfhandle+12(sp),d1   ;File-Handle holen
            lea byte(pc),a0             ;Bufferadresse für zu
                                        ;lesendes Byte
            move.l a0,d2  ;buffer       ;holen
            moveq #1,d3   ;length       ;Anzahl der Bytes: 1
            jsr Read(a6)                ;Byte lesen
            movem.l (sp)+,d1/d2/d3      ;Datenregister zurückholen
            cmp.l#1,d0                  ;Datum richtig gelesen?
            bne.s abbruch               ;verzweige, wenn nein
```

```
            bsr umwandlung                  ;Byte umwandeln und auf
                                            ;Bildschirm bringen
            move.l linelength(pc),d0        ;Zeilenlänge in Bytes holen
            move.b interlace(pc),d1         ;Interlace-Flag holen
            asl.l d1,d0                     ;D0 verdoppeln, wenn
                                            ;Interlace
            add.l d0,d2                     ;Bitplaneoffset ändern
            dbf d3,loop3                    ;verzweige, wenn noch nicht
                                            ;8 Bytes gelesen
            asl.l #3,d0                     ;D0=D0*8
            sub.l d0,d2  ;bpoffset          ;Bitplaneoffset ändern
            add.l #1,d2  ;bpoffset          ;um ein Byte vorrücken
            moveq #0,d1                     ;D1 für Addition löschen
            move.b hires(pc),d1             ;HIRES-Flag löschen
            add.l d1,d2                     ;evtl. um noch ein Byte
                                            ;vorrücken
            dbf d4,loop2                    ;verzweige, wenn noch nicht
                                            ;40 Spalten
            add.l d0,d2  ;bpoffset          ;Bitplaneoffset ändern
            sub.l linelength(pc),d2         ;Bitplaneoffset ändern
            dbf d5,loop1                    ;verzweige, wenn noch nicht
                                            ;25 Zeilen
abbruch:    movem.l (sp)+,d2/d3/d4/d5/a2/a6 ;Register zurückholen
            rts                             ;Rücksprung

;gelesenes Byte umwandeln und in Bildschirm schreiben

umwandlung: movem.l d3/d4/d5/d6/d7/a2,-(sp) ;Register retten
            move.b modus(pc),d0             ;Wert für Tests holen
            lea pword0(pc),a0               ;zu ändernde Worte in Plane
            clr.l (a0)+                     ;löschen
            clr.l (a0)                      ;löschen
            move.w pmaske(pc),d3            ;Pixel-Maske holen
            move.b pperpix(pc),d6           ;shift-Wert für Maske
            cmp.b #2,d0                     ;Modus 2 (320*200)?
            bne.s l14                       ;verzweige,wenn nein
            moveq #7,d4                     ;8 Pixel pro Punkt
            bra.s l15
l14:        moveq #3,d4                     ;4 Pixel pro Punkt
l15:        lea byte(pc),a1                 ;Zeiger auf gelesenes Byte
loop4:      move.b (a1),d5                  ;gelesenes Byte holen
            clr.w d1                        ;Farbindex löschen
            asl.b #1,d5                     ;ein Bit nach X-Flag
                                            ;schieben
```

```
                roxl.w #1,d1                ;Bit nach Farbindex
                                            ;schieben
                cmp.b #2,d0                 ;Modus 2?
                beq.s l16                   ;wenn ja, nur ein Bit
                asl.b #1,d5                 ;sonst noch ein Bit holen
                roxl.w #1,d1                ;und in Farbindex schieben
l16:            move.b d5,(a1)              ;verschobenes Byte zurück-
                                            ;schreiben
                tst.b d0                    ;Modus 0 (Koala)?
                bne.s l3                    ;verzweige, wenn nein
                move.b Farbe0(pc,d1.w),d1   ;entsprechende Farbe holen
                moveq #3,d5                 ;Zähler: vier Bitplanes
                bra.s l4
l3:             moveq #2,d5                 ;Zähler für Bitplanes
                sub.l d0,d5                 ;0 oder 1, je nach Modus
l4:             move.w d5,d7                ;Bitplanezähler zwischen-
                                            ;speichern
                lea pword0(pc),a0           ;Zeiger auf zu ändernde
                                            ;Worte in Bitplanes
loop5:          lsr.b #1,d1                 ;Bit aus Farbe holen
                bcc.s l9                    ;verzweige, wenn Bit nicht
                                            ;gesetzt
                or.w d3,(a0)                ;Pixel setzen
l9:             addq #2,a0                  ;nächstes Planewort
                dbf d5,loop5                ;nächste Plane bearbeiten
                lsr.w d6,d3                 ;Pixel-Maske verschieben
                dbf d4,loop4                ;nächstes gelesenes Pixel

;Planeworte in Planes schreiben

                move.b interlace(pc),d1     ;Zähler für doppelte Auf-
                                            ;lösung
loop6:          move.l sbptab+28(sp),a0     ;Zeiger auf Planeadressen
                lea pword0(pc),a1           ;Zeiger auf Planeworte
                move.w d7,d5                ;Bitplanezähler holen
loop7:          move.l (a0)+,a2             ;Bitplaneadresse holen
                move.b hires(pc),d0         ;Test, ob Wort oder Byte
                beq.s l10                   ;verzweige, wenn nur Byte
                move.w (a1)+,(a2,d2.l)      ;Wort in Plane schreiben
                bra.s l11
l10:            move.b (a1)+,(a2,d2.l)      ;Byte in Plane schreiben
                add.l #1,a1                 ;Zeiger auf nächstes Plane-
                                            ;wort setzen
l11:            dbf d5,loop7                ;nächste Bitplane
                sub.b #1,d1                 ;Interlace?
```

```
            bne.s l12                          ;verzweige, wenn nein
            add.l linelength(pc),d2            ;darunterliegendes Byte
            bra.s loop6                        ;in Plane eingetragen
l12:        move.b interlace(pc),d0            ;Interlace?
            beq.s l13                          ;verzweige, wenn nein
            sub.l linelength(pc),d2            ;wenn ja, Bitplaneoffset
                                               ;anpassen
l13:        movem.l (sp)+,d3/d4/d5/d6/d7/a 2 ;Register zurückholen
            rts                                ;Rücksprung

Farbe0:     dc.b 0                             ;Farbspeicher
Farbe1:     dc.b 0
Farbe2:     dc.b 0
Farbe3:     dc.b 0

pword0:     dc.w 0                             ;Zwischenspeicher für
pword1:     dc.w 0                             ;berechnete Punkte
pword2:     dc.w 0
pword3:     dc.w 0

fhandle:    dc.l 0                             ;File-Handle des zu
                                               ;lesenden Files
linelength: dc.l 0                             ;Zeilenlänge in Bytes
hires:      dc.b 0                             ;Flag für horizontale Auf-
                                               ;lösung
interlace:  dc.b 0                             ;Flag für vertikale Auf-
                                               ;lösung
bptab:      dc.l 0                             ;Zeiger auf Bitplane-
                                               ;Adressentabelle
modus:      dc.b 0                             ;Koala, 4-Farb, 2-Farb
byte:       dc.b 0                             ;gelesenes Byte
pmaske:     dc.w 0                             ;Pixelmaske für or-Ver-
                                               ;knüpfung
pperpix:    dc.b 0                             ;Punkte pro Pixel - 1
```

7.4 Bedienung und Arbeitsweise des Übertragungsprogramms

Um sicherzugehen, daß die Datenübertragung reibungslos abläuft, sollten Sie folgende Reihenfolge beim Starten der Sende- und der Empfangsroutinen einhalten:

Verbinden Sie zuerst die beiden Rechner mit dem Übertragungskabel. Machen Sie das unbedingt im ausgeschalteten Zustand! Es kann sonst zu einem Kurzschluß kommen, der beiden Rechnern ein jähes Ende bereiten würde. Schalten Sie dann beide Rechner ein, und laden Sie die entsprechenden Programme. Vorsicht beim C64 mit Floppy-Beschleunigern, die den Userport benutzen. Sie können die Datenübertragung selbst dann stören, wenn sie sonst problemlos mit herausgezogenem Userportkabel arbeiten. Am besten ist es, wenn Sie Ihren Floppybeschleuniger vollständig abschalten.

Beim Amiga laden Sie zuerst die Workbench, beim Amiga 1000 vorher noch die Kickstart-Diskette. Dann starten Sie den AmigaBASIC-Interpreter und laden das Empfangsprogramm. Wenn dieses geschehen ist, legen Sie wieder die Amiga-Extras-Diskette ein, außer, wenn Sie, wie schon erwähnt, die Dateien dos.bmap und exec.bmap auf Ihre Programmdiskette überspielt haben.

Nach dem Starten des Programms will es zuerst die Kennung für die Diskettenstation von Ihnen erfahren, auf der die Daten abgelegt werden sollen. Geben Sie hier beispielsweise df0: oder df1:, eventuell auch mit einem nachfolgenden Ordnernamen an. Diese Kennung wird allen übertragenen Dateien beim Abspeichern vorangesetzt.

Beim allerersten Starten des Programms nach einem Reset verlangt das Betriebssystem nach der Workbench-Diskette, um den Parallelport-Treiber nachzuladen. Wenn dies geschehen ist, geht das Programm in Warteposition. Sie sollten jetzt Ihre Abspeicherdiskette in das vorher angegebene Laufwerk einlegen.

Achtung: Um beim ersten Programmstart sicherzugehen, daß es zu keinen Empfangsfehlern kommt, sollten Sie erst jetzt das Programm auf dem C64 anlaufen lassen. Warten Sie etwa 40 Sekunden, bis das Maschinenprogramm eingelesen wurde und sich das Menü meldet. Jetzt können Sie zwischen sieben verschiedenen Menüpunkten auswählen.

Menüpunkt null dient zur Datenübertragung entsprechend einer Konvertierungstabelle. Sie müssen nach Wählen dieses Punktes zuerst noch den Namen der Tabelle angeben. Sie wird dann automatisch nachgeladen. Die Übertragung mit einer Tabelle benötigen Sie immer dann, wenn Sie Textdateien übertragen wollen, die sich nicht in die Kategorien Textomat Plus oder Standard ASCII einordnen lassen. Sie brauchen natürlich für jedes Textformat eine entsprechende Konvertierungstabelle. Wir haben Ihnen eine Tabelle für Textomat 64 und Datamat 64 schon mitgeliefert. Wenn Sie dagegen andere Textformate benutzen wollen, kommen Sie nicht darum herum, sich eigene Konvertierungstabellen zu erstellen. Der Aufbau einer Tabelle wurde schon im vorhergehenden Kapitel 7.3 beschrieben.

Menüpunkt eins erlaubt das Übertragen von Standard-ASCII-Dateien. Damit ist es beispielsweise möglich, Daten aus Dateiverwaltungen an den Amiga zu senden. Sie können auch Text-Files, die mit dem normalen Zeichensatz arbeiten, mit diesem Menüpunkt übertragen. Die vorgenommene Konvertierung betrifft die Vertauschung von Groß- und Kleinbuchstaben und die Grafik- und Steuerzeichen des C64. Diese dem Amiga unbekannten Zeichen werden durch das umgekehrte Fragezeichen ersetzt und lassen sich so leicht finden und ersetzen. Die Option unter Menüpunkt fünf ist besonders bei dieser Übertragungsart von Wichtigkeit. Sie erlaubt das Einfügen eines Linefeed-Zeichens ("CHR$(10)") nach jedem Return-Zeichen ("CHR$(13)"). Dies ist bei manchen Textprogrammen wie Textomat Amiga oder Beckertext Amiga notwendig, wenn normale ASCII-Dateien für diese Textprogramme aufbereitet werden.

Menüpunkt zwei konvertiert Textomat-Plus-Dateien, so daß sie problemlos von Textomat Amiga oder Beckertext Amiga gelesen werden können. Dabei werden auch Umlaute richtig umgesetzt. Die Trennungsstriche werden aus dem Text entfernt, da sie von den Textprogrammen auf dem Amiga weder verwertet werden können noch verwertet werden brauchen. Das inverse S, das Steuerzeichensequenzen einleitet, wird durch ein umgekehrtes Fragezeichen ersetzt. Theoretisch könnte man diese Dateien auch mit einer Tabelle entsprechend Menüpunkt null übertragen. Da dieser Menüpunkt aber erst später hinzu kam und wir zu diesem Zeitpunkt schon die Textomat-Plus-Umsetzung fertiggestellt hatten, sahen wir keine Veranlassung, sie wieder zu entfernen.

Der dritte Menüpunkt liest ein BASIC-Programm ein und nimmt diverse Ergänzungen vor, so daß man es leicht für den Amiga umschreiben kann. Zuerst einmal werden alle BASIC-Befehle, wenn nötig, durch Leerzeichen von Variablen und Zahlen abgetrennt. Alle Zeilen, die BASIC-Befehle enthalten, die im AmigaBASIC nicht enthalten sind oder deren Syntax sich von der des AmigaBASIC unterscheiden, werden durch ein :REM??? ergänzt. Das Pi-Zeichen wird durch die Zahl 3.1415927 ersetzt, der Befehl CLR durch den Befehl CLEAR. Steuer- und Grafikzeichen werden wie bei den anderen Übertragungsarten auch durch umgekehrte Fragezeichen ersetzt. Manchmal kommt es dazu, daß ein :REM??? zuviel ergänzt wird. Dies kommt dadurch zustande, daß das Programm nicht zwischen den Systemvariablen ST/TI und in Kommentaren bzw. Print-Befehlen verwendeten ST/TI-Kombinationen unterscheiden kann. Diese Eigenart sollte aber nicht weiter stören.

Mit dem vierten Menüpunkt schließlich können Sie Daten ohne jegliche Konvertierung senden. Dies wird beispielsweise bei Bildern benötigt. Sie können so natürlich auch alle anderen Arten von Daten mit diesem Übertragungsmodus senden. Für die Verarbeitung werden Sie sich dann aber meistens eigene Programme schreiben müssen.

Menüpunkt fünf wurde schon oben im Zusammenhang mit Menüpunkt eins erklärt. Er wirkt in Verbindung mit den Menüpunkten null und eins.

Der sechste Menüpunkt dient zum Programmabbruch. Gehen Sie dabei so vor: Drücken Sie zuerst einmal die Taste Q des Amiga. Sobald Sie nun den Menüpunkt sechs anwählen, wird das Programm bei beiden Rechnern beendet. Sollten Sie das Betätigen der Taste Q versehentlich vergessen, so versuchen Sie nicht, das Amiga-Programm durch Ctlr-C oder Amiga-Punkt zu unterbrechen. In diesem Fall würde nämlich der Datenkanal zum Parallelport nicht geschlossen, was bei einem erneuten Programmstart zu einem Absturz führt. Unternehmen Sie in diesem Fall besser folgendes: Tippen Sie zuerst Q beim Amiga, auch wenn das 64er-Programm schon beendet wurde, und geben Sie POKE 56577,0 beim C64 ein. Jetzt sollte der Amiga sich zurückmelden.

Beim Auswählen eines der Menüpunkte null bis vier werden Sie dazu aufgefordert, eine Diskette ins Laufwerk einzulegen. Dann zeigt das Programm der Reihe nach alle Dateien an und fragt Sie, ob Sie die entsprechende Datei übertragen möchten. Dem Programm ist es übrigens egal, ob es sich um SEQ- oder PRG-Dateien handelt. Nur USR- oder REL-Dateien können Sie nicht übertragen. Nachdem die Auswahl beendet ist, beginnt die Übertragung. Der C64 zeigt jeweils an, welche Datei er sendet, und der Amiga bestätigt dies, indem er anzeigt, daß er diese Datei auf Diskette schreibt. Der Dateiname beim Amiga ist dabei derselbe wie beim C64.

Der Vorteil dieser Art der Übertragung liegt in der Möglichkeit, beliebig lange Dateien zu senden. Auf diese Weise können Sie sogar Dateien, die Sie beispielsweise mit einem C128 erstellt haben und die zu lang sind, um sie komplett in den Speicher eines C64 zu laden, übertragen. Außerdem dauert es nicht länger, eine Datei während des Ladens gleichzeitig zu übertragen, als wenn man zuerst die Datei einlädt und dann sendet.

7.5 Textdateien und Dateien aus Dateiverwaltungen

Zum Senden von Textdateien muß man unterscheiden, was für Texte man übertragen will. Alle Texte, die mit dem Standardzeichensatz erstellt wurden und die entsprechend der auf dem C64 normalerweise verwendeten ASCII-Code-Zuordnung abgespeichert wurden, lassen sich mit dem Menüpunkt eins des ÜbertragungsProgramms direkt senden. Darunter fallen zum Beispiel BASIC-Programme, die mit den Befehlen CMD und LIST in eine ASCII-Datei gelistet wurden. Dies sollte man nicht mit der Funktion von Menüpunkt drei verwechseln. Er übernimmt direkt ein mit SAVE abgespeichertes BASIC-Programm und paßt es weitgehend an den Amiga an. Menüpunkt eins reicht auch für viele amerikanische Textprogramme aus, die keine Sonderzeichen wie Umlaute verwalten.

Deutsche Textprogramme wie Textomat Plus oder Textomat arbeiten jedoch mit Umlauten oder anderen Spezialzeichen. Daher ist es erforderlich, hierfür spezielle Konvertierungstabellen zu verwenden. Für Textomat Plus können Sie natürlich auf eine Tabelle verzichten und statt dessen Menüpunkt zwei aufrufen. Wenn Sie dagegen zum Beispiel mit der Textomat-Tabelle senden wollen, erzeugen Sie diese zuerst mit dem entsprechenden Programm und laden sie dann mit Menüpunkt null unter Angabe des Namens TEXTOMATTAB nach.

Der erweiterte ASCII-Code des Amiga ist genormt und enthält alle möglichen Sonderzeichen verschiedener Sprachen, so daß nahezu jedes Textprogramm die beim Amiga empfangenen Dateien lesen kann. Daher braucht hier nicht nochmal eine Konvertierung durchgeführt zu werden. Es kann auch nur selten schaden, die Option Linefeed senden abzuschalten, solange man mit reinen Textdateien arbeitet. Anders dagegen sieht es aus, wenn man Dateien aus Dateiverwaltungen übertragen will. Wir demonstrieren das am Beispiel der Programme Datamat und Superbase, die es inzwischen sowohl auf dem C64 als auch auf dem Amiga gibt.

Der erste Schritt zum Übertragen der Daten ist jeweils das Erstellen einer Textverarbeitungs- bzw. Exportdatei. Darin ste-

hen hintereinander alle vorhandenen Daten, jeweils voneinander getrennt durch ein Trennzeichen wie CHR$(13) (Return). Nicht enthalten, sind Informationen über den Aufbau der Dateimaske.

Beim Programm Datamat wählen Sie zuerst den Menüpunkt Datei auswerten und dann Floppy. Die Frage nach der Anzahl der Zeichen können Sie getrost z. B. mit 80 beantworten. Wenn Sie noch keine Auswertungsdatei erstellt haben, erhalten Sie nun Gelegenheit dazu. Danach fragt Sie das Programm nach dem Namen, unter dem die Textverarbeitungsdatei abgespeichert werden soll. Wenn Sie beabsichtigen, diese Datei später von Datamat Amiga einzulesen, sollten Sie dem Namen die Endung .DIF anhängen. Schließlich wird die Datei erzeugt, wobei nach jedem Datenfeld und jedem Datensatz ein Return-Zeichen eingefügt wird.

Superbase macht es einem sogar noch einfacher. Wählen Sie einfach den Menüpunkt Maintain und danach Export. Wenn Sie jetzt den Dateinamen eingeben, wird eine Exportdatei erstellt, die prinzipiell genauso aufgebaut ist wie die Textverarbeitungsdatei von Datamat. Zwar lassen sich bei Superbase auch noch unterschiedliche Trennzeichen für Datenfelder und Datensätze vereinbaren, dies ist aber normalerweise unnötig. Auch hier gilt: Für Datamat Amiga wird die Endung .DIF beim File-Namen benötigt.

Senden Sie nun die entsprechende Datei mit dem Übertragungsprogramm zum Amiga. Schalten Sie aber die Option Linefeed senden ab. Sofern Sie Ihre Datei mit Datamat erzeugt und dabei den deutschen Zeichensatz verwendet haben, sollten Sie mit der Textomat-Konvertierungstabelle arbeiten. In allen anderen Fällen genügt der Menüpunkt eins für Standard-ASCII-Texte.

Um jetzt die Daten wieder einzulesen, müssen Sie zunächst eine Dateimaske erstellen, egal, ob Sie mit Superbase Amiga oder Datamat Amiga arbeiten. Die Felder der Maske müssen mindestens genauso groß sein wie in Ihrer C64-Datei und sollten auch in der Reihenfolge übereinstimmen, damit alle Daten problemlos erfaßt werden können.

Bei Datamat Amiga wählen Sie zunächst im Menü Optionen aus dem Untermenü Arbeitsumgebung ändern den Menüpunkt Ein-/Ausgabe-Parameter. Hier stellen Sie dann sowohl das Feldtrennzeichen als auch das Datensatztrennzeichen auf 0D. Als nächstes müssen Sie, wie im Handbuch beschrieben, eine Feldauswahl treffen, bei der entschieden wird, welche Datenfelder in welcher Reihenfolge eingelesen werden sollen. Jetzt können Sie endlich mit dem Einlesen beginnen. Dazu rufen Sie aus dem Menü Ein- Ausgabe den Menüpunkt Textdatei einlesen auf. Der Amiga bietet Ihnen dann alle Dateien mit der Endung .DIF an, von denen Sie die gewünschte anklicken. Am einfachsten geht das, wenn Sie die Dateimaske auf der gleichen Diskette angelegt haben, auf der sich auch die Exportdatei befindet.

Auch bei Superbase müssen zuerst Feld- und Datensatztrennzeichen festgelegt werden. Unter Setzen:Option stellen Sie den Feldtrenner auf 13 und den Satztrenner auf 0. Mit Prozeß:Import kann dann das Einlesen erfolgen. Geben Sie vor dem Namen der Datei auch noch die Bezeichnung des Diskettenlaufwerks, also DF0: oder DF1: an. Superbase ist allerdings keine deutsche Dateiverwaltung und hat daher Schwierigkeiten mit Umlauten und Sonderzeichen. Sie können daher keine Datamat-Dateien mit deutschem Zeichensatz übernehmen.

7.6 Übertragung von Bildern

Für die Übertragung von Bildern benötigen Sie neben den Übertragungsprogrammen auch noch das Grafikwandlerprogramm. Dieses ermöglicht es Ihnen, Multicolorgrafiken im Koala-Format und Bilder der Auflösungen 160*200 und 320*200 ohne weitere Farbinformationen in das auf dem Amiga gebräuchliche IFF-Format umzurechnen.

Zuerst übertragen Sie wie gewohnt mit dem Übertragungsprogramm alle Bilder zum Amiga, die für Sie von Interesse sind. Benutzen Sie dabei den Menüpunkt 4, also das Übertragen ohne Konvertierung. Wenn nun alle Bilder beim Amiga angekommen sind, laden Sie als nächstes das Grafikwandlerprogramm. Genau wie das Datenempfangsprogramm benötigt der Grafikwandler

die Dateien dos.bmap und exec.bmap aus dem Ordner Basicdemos von Ihrer Amiga-Extras-Diskette. In den ersten drei Zeilen des Programms wird festgelegt, aus welchem Ordner es die .bmap-Dateien holt, aus welchem Ordner die Grafikdateien gelesen werden und in welchen Ordner die IFF-Dateien geschrieben werden. Sie können diese Konventionen selbstverständlich abändern. Das Programm fragt Sie zur Sicherheit noch einmal, ob alles seine Richtigkeit hat.

Nach dem Programmstart legen Sie zuerst die Diskette mit den .bmap-Dateien ein und betätigen dann eine Taste. Warten Sie einen kurzen Augenblick, bis die Dateien nachgeladen wurden und die Maschinenunterroutine aus den DATA-Zeilen in den Speicher geschrieben wurde. Sobald sich das Menü meldet, können Sie die Diskette einlegen, auf der die Grafiken abgespeichert sind.

Wählen Sie zuerst mit den Funktionstasten F4 und F5, auf welche Auflösungsstufe des Amiga Sie Ihre Bilder konvertiert haben möchten. Zur Wahl stehen 320 oder 640 Punkte in der Breite und 200 oder 400 Punkte in der Höhe. Als nächstes geben Sie an, welche Art von Grafik Sie konvertieren möchten. Dazu betätigen Sie eine der Funktionstasten F1 bis F3.

Bei der nun folgenden Eingabe des Namens der zu ladenden Datei gibt es zwei Dinge zu beachten. Zuerst einmal: Die Namenseingabe wurde mit einem einfachen LINE INPUT-Befehl programmiert und ist daher nicht gesichert. Seien Sie also vorsichtig, damit Sie nicht die Grenzen der Eingabemaske überschreiten. Als zweites: Koala-Bilder haben einen sehr speziellen Dateinamen, der als erstes Zeichen ein Grafikzeichen enthält. Dieses Grafikzeichen wird bei der Konvertierung zu einem umgekehrten Fragezeichen, das Sie aber nicht auf der Tastatur suchen müssen, da es vom Grafikwandler schon vorgegeben wird. Er fügt auch die nötigen Leerzeichen am Ende des File-Namens an, die der Name des Koala-Bildes zwar hat, die sich beim Amiga aber nur sehr schwer erkennen lassen. Falls Sie es doch einmal benötigen: Das umgekehrte Fragezeichen erhalten

Sie sowohl bei der deutschen als auch bei der Tastaturbelegung aus dem Kickstart-ROM durch gleichzeitiges Betätigen von Shift, ALT und M.

Als nächstes erwartet das Programm von Ihnen die Eingabe des Namens, unter dem das entsprechende Bild wieder abgespeichert werden soll. Wenn Sie keinen Namen angeben, wird das Bild zwar angezeigt, aber nicht abgespeichert.

Nachdem nun alle Vorbereitungen getroffen sind, beginnt das Programm mit der Konvertierung. Die entsprechende Datei wird geladen und das daraus generierte Bild angezeigt. Stören Sie sich nicht an dem kleinen Fenster, das zu Anfang der Konvertierung angezeigt und dann sofort überschrieben wird: Es dient nur zum Auffinden der Bildschirmspeicheradressen. Sollte die angegebene Datei nicht gefunden werden können, wird das Hauptmenü neu aufgebaut, und Sie können noch einmal mit der Eingabe aller Parameter beginnen. Wenn das entsprechende Bild komplett umgewandelt ist, ertönt ein akustisches Signal. Für den Fall, daß Sie einen Namen zum Abspeichern des Bildes angegeben haben, sollten Sie nun die Diskette einlegen, auf der das Bild abgelegt werden soll. Sie können natürlich auch die Diskette verwenden, die sich schon im Laufwerk befindet.

Nach einem Tastendruck beginnt das Programm mit dem Abspeichern des Bildes im IFF-Format. Sie können es nun problemlos mit einem Malprogramm wie Deluxe Paint oder Graphicraft laden. Sobald das Speichern beendet ist, oder wenn die Grafik nur angeschaut werden sollte, wird nun wieder das Hauptmenü angezeigt. Sie können jetzt wahlweise ein weiteres Bild umwandeln oder das Programm durch ESC abbrechen.

Nun noch eine Erklärung dazu, wie die Bilder konvertiert werden. Beim Koala-Format werden die Amiga-Grafiken grundsätzlich mit vier Bitplanes dargestellt, da diese Bilder aus bis zu 16 Farben bestehen können. Wir haben dabei versucht, die verwendeten Farben nach Möglichkeit denen des C64 anzugleichen. Ob sie auch bei Ihnen korrekt angezeigt werden, hängt natürlich

davon ab, wie Sie Ihren Monitor bzw. Fernseher eingestellt haben. Grafiken des Formats 160*200 bestehen dementsprechend aus zwei Bitplanes, wobei als Farben die ersten vier C64-Farben verwendet werden. Meistens ist es daher bei solchen Grafiken notwendig, die Farben mit einem Malprogramm nachträglich noch richtig anzupassen. Das gleiche gilt auch für die 320*200-Grafiken. Sie bestehen natürlich nur aus zwei Farben und einer Bitplane.

8. Alte Hardware am neuen Gerät

Haben auch Sie sich beim Kauf Ihres Amiga gefragt, was Sie nun mit Ihrem C64 und der dazugehörigen Ausstattung machen sollen? Muß man wirklich 800 bis 900 DM für den neuen Monitor auf den Ladentisch blättern, oder tut es nicht auch der alte Monitor oder sogar der Fernseher? Und was macht man mit seinem Drucker MPS801, für den Centronics ein absolutes Fremdwort darstellt? Antworten auf alle diese Fragen liefern die nun folgenden Kapitel.

8.1 Monitor

An einen Amiga läßt sich natürlich jeder RGB-Monitor anschließen. Einen solchen Monitor werden Sie aber in den seltensten Fällen an Ihrem C64 betrieben haben. Es genügt aber, wenn Sie einen Monitor mit CVBS-(Video-)Eingang besitzen. Auch ein Fernseher mit Video-Eingang in Verbindung mit einem Videorecorder läßt sich problemlos an den Amiga anschließen. Für den Betrieb des Amiga an einem Fernseher ohne Video-Eingang benötigt man zusätzlich das von Commodore angebotene Fernseh-Modul. Dabei sind allerdings einige Nachteile in Kauf zu nehmen.

Zuerst wäre da der Video-Ausgang des Amiga 500/2000. Hier kommt nämlich nur ein Schwarzweiß-Signal an, während beim Amiga 1000 ein vollständiges Farbsignal zur Verfügung steht. Auf dem deutschen Markt wurden jedoch auch einige Amiga 1000 verkauft, die nicht der PAL-Norm, sondern nur der amerikanischen NTSC-Norm entsprechen. Das Modul von Commodore löst allerdings alle diese Probleme, da es jeden Amiga mit einem Videoausgang und einem Antennenausgang versieht.

Der zweite Nachteil, den der Anschluß eines Nicht-RGB-Monitors an den Amiga bringt, ist die schlechtere Bildqualität. Zwar läßt sich die 80-Zeichen-Darstellung selbst auf einem Fernseher mittlerer Preisklasse (im Gegensatz zum C64) noch

halbwegs erkennen, aber als besonders überragend kann man sie auch nicht bezeichnen. Gerade, wenn man keinen Video-Eingang, sondern nur einen Antenneneingang an seinem Fernseher zur Verfügung hat, leidet die Bildqualität doch sehr deutlich. Selbst ein Monitor wie der 1702, also der Standard-Monitor zum C64, bringt kein Bild zustande, das sich mit dem eines 1081-Monitors messen könnte. Wollen Sie Ihren Amiga in erster Linie zum Spielen und nur gelegentlich zum Verarbeiten von Texten oder Schreiben von Programmen einsetzen, so reicht an sich ein Fernseher schon aus. Für den professionellen Einsatz ist aber in jedem Fall der Kauf eines RGB-Monitors anzuraten.

8.2 Drucker

Ein Drucker ist ein Gerät, das man nur ungern neu kaufen möchte, wenn man sich einen Amiga anschafft. Ob Sie Ihren alten Drucker problemlos weiterverwenden können, hängt natürlich davon ab, was für einen Drucker Sie besitzen und wie Sie ihn an Ihren C64 angeschlossen haben.

Einen Drucker mit RS232- oder Centronics-Schnittstelle können Sie natürlich direkt weiterverwenden. Dies betrifft zum Beispiel die Epson-Drucker. Wenn Sie die Wahl haben: Entscheiden Sie sich für die parallele Datenübertragung. Fast alle Programme gehen von dieser Voreinstellung aus, und Sie haben keinerlei Probleme mit Baudraten, Start- und Stopbits oder ähnlichen Dingen. Statt des Interfaces, daß Sie wahrscheinlich für Ihren C64 benötigten, brauchen Sie jetzt nur noch ein einfaches Flachbandkabel, das die entsprechenden Leitungen der Schnittstellen miteinander verbindet. Dieses Kabel ist beim Amiga 500/2000 identisch mit dem für IBM-Rechner angebotenen.

Wenn Sie einen Amiga 1000 besitzen oder falls Sie Geld sparen wollen, so sollten Sie sich das entsprechende Kabel selbst löten. Benötigt wird beim Amiga 1000 eine 25-polige Sub-D-Buchse, beim Amiga 500/2000 ein 25-poliger Sub-D-Stecker. Außerdem brauchen Sie einen 36-poligen Centronics-Stecker und Flachbandkabel mit 12 Leitungen. Verbinden Sie einfach die Pins 1 bis 10 des Centronics-Steckers mit den Pins 1 bis 10 des Sub-D-

Stecker bzw. der Sub-D-Buchse genau in der Reihenfolge. Außerdem muß die Masseleitung von Pin 22 am Amiga mit Pin 28 beim Centronics-Stecker verbunden sein. Wenn Sie wollen, können Sie auch noch die Reset-Leitung durchführen, so daß der Drucker bei jedem Reset des Amiga ebenfalls einen Reset erhält. Diese liegt beim Centronics-Stecker am Pin 31, beim Amiga 500/2000 an Pin 16 und beim Amiga 1000 an Pin 25. Seien Sie aber vorsichtig, und verwechseln Sie keinesfalls irgendwelche Leitungen. Verwenden Sie unbedingt Steckergehäuse, und wechseln Sie niemals einen der Stecker, während der Amiga oder der Drucker eingeschaltet sind. Ein Kurzschluß könnte sowohl beim Amiga als auch beim Drucker diverse Chips zerstören.

Als Besitzer eines Epson oder befehlskompatiblen Druckers sind bei keinem Programm irgendwelche Voreinstellungen nötig. Bei anderen Druckern verwenden Sie das Voreinstellungsprogramm Preferences von der Workbench-Diskette. Wie Sie damit Voreinstellungen auch auf Disketten vornehmen, die Preferences nicht enthalten, steht in Kapitel 1.4.1.

Was machen Sie aber, wenn Sie einen Drucker besitzen, der als einzige Schnittstelle den IEC-Bus-Anschluß des C64 zur Verfügung hat? Oder was, wenn Sie das Geld für ein Centronics-Interface sparen wollen? Nun, was würden Sie von einem universellen Drucker-Interface halten, das grundsätzlich jedem C64-kompatiblen Drucker einen einwandfreien Betrieb am Amiga ermöglicht? Als Beigabe erhalten Sie dann sogar einen Druckerpuffer von 32 KByte. Sie haben es sicherlich schon erraten: Die Idee ist, den alten C64 selbst als frei programmierbares Interface zu verwenden. Dazu benötigen Sie nicht mehr als das Datenkabel aus Kapitel 7 und folgendes Programm für den C64:

```
80 gosub900
90 poke56,64:clr
100 sk=0:rem sekundaeradresse
120 open1,4,sk:close1:ifst<>0then120
130 open1,4,sk:cmd1
140 sys49152
150 close1
```

```
160 goto100
900 fori=1to171step15:forj=0to14:reada$:b$=right$(a$,1)
905 a=asc(a$)-48:ifa>9thena=a-7
910 b=asc(b$)-48:ifb>9thenb=b-7
920 a=a*16+b:c=(c+a)and255:poke49151+i+j,a:next:reada:ifc=a
    thenc=0:next:goto940
930 print"fehler in zeile:";peek(63)+peek(64)*256:stop
940 print"ok":return
1000 dataa9,36,85,01,a0,00,a9,40,84,fb,85,fc,84,fd,85, 244
1001 datafe,8c,03,dd,78,a9,10,2c,0d,dd,d0,1d,ad,01,dd, 41
1002 data4c,14,c0,a9,40,85,02,a2,00,a9,10,2c,0d,dd,d0, 209
1003 data0a,ca,d0,f6,c6,02,d0,f2,4c,40,c0,ad,01,dd,20, 27
1004 data62,c0,d0,e1,20,88,c0,f0,0c,20,93,c0,58,20,d2, 244
1005 dataff,78,a5,90,d0,09,a9,10,2c,0d,dd,f0,e8,d0,de, 218
1006 data84,90,a9,37,85,01,58,60,91,fb,e6,fb,d0,0f,e6, 100
1007 datafc,a5,fc,8d,00,04,c9,c0,d0,04,a9,40,85,fc,a5, 154
1008 datafb,18,69,01,aa,a5,fc,69,00,e4,fd,d0,02,c5,fe, 167
1009 data60,a5,fb,c5,fd,d0,04,a5,fc,c5,fe,60,b1,fd,48, 80
1010 datae6,fd,d0,0f,e6,fe,a5,fe,8d,01,04,c9,c0,d0,04, 56
1011 dataa9,40,85,fe,68,60,00,ff,ff,00,00,ff,ff,00,00, 48
```

Das darin enthaltene Maschinenprogramm sieht so aus:

```
c000 lda #$36       BASIC-Interpreter
c002 sta $01        abschalten
c004 ldy #$00       Index auf null setzen
c006 lda #$40       HIGH-Byte der Startdresse des Buffers
c008 sty $fb        Buffer-Zeiger LOW setzen
c00a sta $fc        Buffer-Zeiger HIGH setzen
c00c sty $fd        Buffer-Anfang LOW setzen
c00e sta $fe        Buffer-Anfang HIGH setzen
c010 sty $dd03      Datenrichtungsregister auf Eingang stellen
c013 sei            Interrupts sperren
c014 lda #$10       Maske für Bit 4 laden
c016 bit $dd0d      liegt schon Byte an?
c019 bne $c038      verzweige, wenn ja
c01b lda $dd01      Signal auf Port-Control-Leitung ausgeben
c01e jmp $c014      zum Schleifenbeginn
c021 lda #$40       HIGH-Byte des Time-Out-Zählers
c023 sta $02        setzen
c025 ldx #$00       LOW-Byte des Zählers setzen
c027 lda #$10       Maske für Bit 4 laden
c029 bit $dd0d      liegt Byte an?
c02c bne $c038      verzweige, wenn ja
c02e dex            Time-Out-Zähler LOW-Byte verringern
```

```
c02f bne $c027      verzweige, wenn ungleich null
c031 dec $02        Time-Out-Zähler HIGH-Byte verringern
c033 bne $c027      verzweige, wenn ungleich null
c035 jmp $c040      Schleifenende: Bufferinhalt senden
c038 lda $dd01      Byte vom Port holen
c03b jsr $c062      und in Buffer schreiben
c03e bne $c021      Schleifenanfang, wenn Buffer noch nicht voll
c040 jsr $c088      Buffer leer?
c043 beq $c051      verzweige, wenn ja
c045 jsr $c093      Byte aus Buffer holen
c048 cli            Interrupts zulassen
c049 jsr $ffd2      Byte zum Drucker senden
c04c sei            Interrupts sperren
c04d lda $90        Status testen
c04f bne $c05a      verzweige, wenn Fehler
c051 lda #$10       Maske für Bit 4 laden
c053 bit $dd0d      liegt neues Byte an?
c056 beq $c040      wenn nein, nächstes Byte aus Buffer senden
c058 bne $c038      wenn ja, Bytes in Buffer schreiben

c05a sty $90        Fehlerabbruch: Status löschen
c05c lda #$37       BASIC-Interpreter
c05e sta $01        einschalten
c060 cli            Interrupts zulassen
c061 rts            Rücksprung

c062 sta ($fb),y    Byte in Buffer schreiben
c064 inc $fb        Bufferzeiger LOW erhöhen
c066 bne $c077      verzweige, wenn kein Übertrag
c068 inc $fc        Bufferzeiger HIGH erhöhen
c06a lda $fc        Bufferzeiger HIGH holen
c06c sta $0400      zur Kontrolle in Bildschirm schreiben
c06f cmp #$c0       obere Speichergrenze erreicht?
c071 bne $c077      verzweige, wenn nein
c073 lda #$40       Bufferzeiger auf Speicherbeginn
c075 sta $fc        setzen
c077 lda $fb        Bufferzeiger LOW holen
c079 clc            Carry für Addition löschen
c07a adc #$01       Eins addieren
c07c tax            Wert merken
c07d lda $fc        Bufferzeiger HIGH holen
c07f adc #$00       Übertrag addieren
c081 cpx $fd        Buffer voll? Test LOW-Byte
c083 bne $c087      verzweige, wenn nein
c085 cmp $fe        Buffer voll? Test HIGH-Byte
c087 rts            Rücksprung
```

```
c088 lda $fb        Test, ob Buffer leer: LOW-Byte Bufferzeiger
c08a cmp $fd        mit LOW-Byte Bufferanfang vergleichen
c08c bne $c092      verzweige, wenn ungleich
c08e lda $fc        HIGH-Byte Bufferzeiger
c090 cmp $fe        mit HIGH-Byte Bufferanfang vergleichen
c092 rts            Rücksprung

c093 lda ($fd),y    Byte aus Buffer holen
c095 pha            merken
c096 inc $fd        Bufferanfang LOW erhöhen
c098 bne $c0a9      verzweige, wenn kein Übertrag
c09a inc $fe        Bufferanfang HIGH erhöhen
c09c lda $fe        Bufferanfang HIGH holen
c09e sta $0401      zur Kontrolle in Bildschirm schreiben
c0a1 cmp #$c0       Speicherobergrenze erreicht?
c0a3 bne $c0a9      verzweige, wenn nein
c0a5 lda #$40       Bufferzeiger auf Speicherbeginn
c0a7 sta $fe        setzen
c0a9 pla            Byte aus Buffer zurückholen
c0aa rts            Rücksprung
```

Um das Programm in Betrieb zu nehmen, stecken Sie zuerst das Datenkabel in beide Rechner und schalten danach erst die Geräte und den Drucker ein. Laden Sie dann das zuvor eingetippte C64-Programm, und starten Sie es. Sobald das Maschinenprogramm aus den DATA-Zeilen in den Speicher gelesen wurde, erscheint beim C64 die Meldung OK. Jetzt ist das Drucker-Interface bereit, seine Arbeit aufzunehmen.

Die ganze Sache hat natürlich einen Haken, den wir hier nicht verschweigen wollen. Der Amiga hat nämlich keinen Druckertreiber für Drucker wie beispielsweise den MPS 801.

Mit einem solchen Gerät können Sie demnach nicht einfach über das device prt: arbeiten, sondern Sie müssen statt dessen auf par: zurückgreifen. Das bedeutet, daß Sie fast nur selbstgeschriebene Programme mit Ihrem Drucker verwenden können, oder Programme, die eine umfangreiche Druckeranpassung ermöglichen.

Es gibt aber genügend Drucker, die nicht mit einem Centronics-Anschluß ausgestattet sind, die aber trotzdem weitgehend zu einem der vom Amiga unterstützten Druckertreiber kompatibel sind oder sich kompatibel schalten lassen. Kontrollieren Sie also, ob Ihr Drucker im Handbuch zum Beispiel als Epson- oder IBM-kompatibel ausgewiesen wird. Wenn Sie selbst ein wenig Erfahrung mit Maschinensprache besitzen, können Sie auch statt dessen eine Epson-Drucker-Emulation auf dem C64 programmieren, die die Epson-spezifischen Codes, soweit das möglich ist, in Commodore-Codes umwandelt. Dadurch würde dann selbst ein alter MPS-Drucker zum Amiga passen.

Das Druckprogramm öffnet den Druckerkanal von BASIC aus, was den Vorteil bietet, daß Sie hier noch eine Umschaltung auf einen bestimmten Drucker-Modus vornehmen können. Wenn Sie das Programm regelmäßig benutzen wollen, empfehlen wir Ihnen, es auf ein EPROM zu brennen, wodurch Sie sowohl die Floppy als auch einen Monitor für den C64 einsparen.

Vielleicht kommen Ihnen ja auch eigene Ideen, was Sie in Ihr intelligentes Drucker-Interface noch einbauen können. Die Möglichkeiten sind wirklich unbegrenzt.

8.3 Floppy

Mit dem Kauf Ihres Amiga haben Sie natürlich in jedem Fall (mindestens) ein Diskettenlaufwerk erworben, nämlich das eingebaute. Da kommt einem doch leicht die Idee, sich die Kosten für ein Zweitlaufwerk zu sparen, indem man versucht, sein altes Laufwerk wiederzuverwenden. Hierzu sollte man aber Nutzen und Aufwand miteinander vergleichen.

Theoretisch wäre es ohne weiteres möglich, eine 1541/70/71 genauso an den parallen Port des Amiga anzuschließen, wie es beim C64 und C128 gemacht wird. Da diese Laufwerke ihr Betriebssystem eingebaut haben, brauchte man "bloß" die IEC-Bus-Verwaltung des C64 auf den Amiga zu übertragen und sich damit eigene Treibersoftware erstellen. Was hätte man aber davon? Eine 170- bzw. 340-KByte-Floppy, die völlig inkompatibel

mit dem üblichen Diskettenformat ist, läßt sich wohl kaum als Ersatz für ein 880 KByte - 3½-Zoll-Laufwerk verwenden, von der unerträglichen Langsamkeit des Laufwerks ganz zu schweigen.

Der einzige Vorteil, den man daraus zöge, wäre der, daß man alte Programme und Daten vom Amiga aus lesen könnte. Aber dies läßt sich ja auch mit sehr viel weniger Aufwand bewerkstelligen, wie wir in Kapitel 7 gesehen haben. Ein Datenkabel ist viel einfacher zu bauen und benötigt auch nur ein Minimum an Betriebssoftware. Die Zeichen- und Grafik-Konvertierung, das einzig wirkliche Umfangreiche an der Software, benötigt man natürlich auch beim direkt angeschlossenen Laufwerk.

Sie sind also weitaus besser beraten, Ihr altes Laufwerk zu verkaufen und sich dafür ein neues 3½-Zoll-Laufwerk zu holen. Wenn Sie sich nicht gerade "übers Ohr hauen" lassen, machen Sie bei einem solchen Handel nahezu überhaupt keinen Verlust.

Glossar

AGNUS

Einer der Spezialchips des Amiga trägt den Namen AGNUS. Neben der Verwaltung des ->DMA enthält AGNUS auch die beides Einheiten ->Blitter und ->Copper.

Anklicken

Ein Symbol mit der Maus anklicken heißt, den Mauszeiger auf dieses Symbol zu setzen und dann die linke Maustaste kurz zu betätigen.

ASCII-Code

Der ASCII-Code ist eine Norm dafür, in welcher Reihenfolge die auf einem Computer verfügbaren Zeichen durchnumeriert werden. Mit dem ASCII-Code eines Zeichens meint man die Nummer dieses Zeichens. Eine ASCII-Code-Tabelle befindet sich im Anhang A ihres BASIC-Handbuchs.

Backspace

Mit Backspace wird die Taste oben rechts mit dem nach links zeigenden Pfeil bezeichnet. Sie entspricht in ihrer Funktion der DEL-Taste des C64 und löscht das Zeichen links vom Cursor.

BAM

(Block Availability Map, deutsch: Blockbelegungstabelle) Auf einer Diskette muß natürlich ein Verzeichnis existieren, das zeigt, welche ->Sektoren schon belegt sind und welche noch beschrieben werden dürfen. Dieses Verzeichnis stellt die BAM dar. Wenn eine Datei nicht ordnungsgemäß geschlossen wurde, kann es vorkommen, daß die BAM falsche Daten enthält. Der Amiga erkennt das automatisch und ruft dann das Programm Disk-Validator auf, um die BAM zu reorganisieren.

BCD

(Binary Code Digit) BCD ist eine besondere Form der Zahlendarstellung, die sich am Dezimalsystem orientiert. Hierbei werden je vier Bits nicht für eine Ziffer von 0 bis 15 (Hexadezimal: 0 bis F), sondern für eine Ziffer von 0 bis 9 verwendet. Dadurch passen dann in ein Byte nicht 256 Werte, sondern nur 100.

Betriebssystem

Jeder Computer benötigt zum Betrieb ein Programm, das man Betriebssystem nennt. Das Betriebssystem kümmert sich um eine ganze Menge von Aufgaben, angefangen bei dem, was passiert, wenn man den Rechner einschaltet, über die Ansteuerung der Diskettenlaufwerke bis hin zur Verwaltung von Text und Grafik. Das Betriebssystem des Amiga trägt den Namen Kickstart. Beim Amiga 500 und 2000 steht das Betriebssystem sofort nach dem Einschalten zur Verfügung, während es beim Amiga 1000 noch von der Diskette geladen werden muß. Dies wird durch eine kurze Boot-Routine erledigt, die durch direkte Ansteuerung der Diskettenstation auch ohne Betriebssystemroutinen Daten von der Diskette laden kann. Für den Amiga 1000 stehen zwei Betriebssysteme zur Verfügung, nämlich die alte Kickstart 1.1, die noch viele Fehler und Schwächen hat, und die Kickstart 1.2, die auch fast unverändert im Amiga 500/2000 steckt.

Bit-Plane

(deutsch: Bit-Ebene) Der Grafikbildschirm des Amiga besteht aus einer großen Anzahl von einzelnen Punkten. Eine Bit-Plane ist nun ein Speicherbereich, bei dem jedes Bit des Speichers einem Punkt auf dem Bildschirm zugeordnet ist. Normalerweise verwendet man mehrere Bit-Planes gleichzeitig, so daß insgesamt jedem Punkt mehrere Bits entsprechen. Die Farbe eines Punktes hängt nun davon ab, wie die zu ihm gehörenden Bits gesetzt sind. Zum Beispiel hat jeder Punkt bei einer aus drei Bit-Planes bestehenden Grafik auch drei Bits. Da man mit drei Bits bekanntermaßen acht verschiedene Werte darstellen kann, ist die Anzahl der maximal gleichzeitig verwendbaren Farben somit auf acht beschränkt.

Blitter

Einer der Teile des Spezialchips ->AGNUS nennt sich Blitter. Er nimmt dem Prozessor Arbeiten ab, wie z.B. Speicherbereiche zu verschieben oder Linien zu zeichnen.

Booten

Der sogenannte Bootstrap ist das automatische Laden eines Programmes nach dem Einschalten des Rechners oder nach einem Reset. Beim Amiga 1000 muß man zwischen zwei Boot-Vorgängen unterscheiden. Zuerst muß immer das Betriebssystem Kickstart gebootet werden. Der zweite Boot-Vorgang dient zum Starten eines Programmes oder der Workbench und entspricht dem Bootvorgang beim Amiga 500/2000.

Buffer

(auch: Puffer) Zwischenspeicher für Daten. Wenn bei irgendeiner Art von Datentransfer Daten schneller anfallen, als sie bearbeitet werden können, so richtet man einen Pufferspeicher ein, der solange die Daten aufnimmt.

Chip-RAM

->Fast-RAM

CLI

(Command Line Interface) Das CLI ist ein Teil des Amiga-DOS, der Befehlseingaben über die Tastatur entgegennimmt und diese ausführt. Das CLI ist zwar bei weitem nicht so komfortabel wie die Workbench, aber in vieler Hinsicht leistungsfähiger.

Command Line Interface

->CLI

Compiler

Ein Programm, das in einer höheren Programmiersprache geschrieben ist, kann nicht direkt vom Computer ausgeführt werden. Der Rechner ist nämlich nur dazu in der Lage, relativ simple Befehle, sogenannte Maschinenbefehle, abzuarbeiten. Direkt in Maschinensprache zu programmieren, ist aber sehr umständlich und auch schwer erlernbar. Daher geht man den Weg, daß man seine Programme zwar in einer Hochsprache wie C oder BASIC schreibt, es jedoch dem Computer überläßt, die Befehle in Maschinenbefehle zu übersetzen. Compiler sind Programme, die einen Programmtext vollständig in Maschinensprache übersetzen. Das so erzeugte Maschinenprogramm ist danach ohne den Compiler ablauffähig. Interpreter dagegen holen sich während des Programmablaufs je einen Befehl des Programmes und führen dafür eine ganze Reihe von Maschinenbefehlen aus. Interpreter haben den Vorteil, daß man ein Programm sofort nach der Eingabe starten kann. Sie sind aber meistens langsamer als compilierte Programme, da Befehle innerhalb von Schleifen jedesmal neu erkannt und übersetzt werden müssen, wohingegen der Übersetzungsvorgang bei compilierten Programmen nur ein einziges Mal, nämlich vor der Programmausführung erfolgen muß. BASIC ist eine typische Interpreter-Sprache, für die aber auch Compiler existieren. C-Interpreter gibt es (zumindest auf dem Amiga) keine, da die Sprache C sich durch leichte Compilierbarkeit auszeichnet.

Copper

Dieser Teil des Chips ->AGNUS kümmert sich um den Grafikaufbau. Er sorgt zum Beispiel für die Verwaltung von Screens mit unterschiedlichen Auflösungen und Farben.

Zylinder

Eine Spur auf einer Diskettenseite nennt man ->Track. Von einem Zylinder spricht man dann, wenn man die beiden zusammengehörigen Tracks auf den beiden Diskettenseiten meint. Somit hat eine Diskette beim Amiga 160 Tracks, aber nur 80 Zylinder.

Datei

(engl. File) Eine Datei ist eine beliebige Ansammlung von zusammengehörigen Daten. Auf einer Diskette abgelegt erhält jede Datei einen einzelnen Namen. Dateien können sowohl Programme als auch irgendwelche anderen Daten enthalten.

Directory

Das Inhaltsverzeichnis einer Diskette bezeichnet man mit dem Fachausdruck Directory. Ein Directory kann entweder Dateinamen oder Namen von Unter-Directories enthalten. Unter-Directories dienen zum Zusammenfassen von Dateinamen unter einem Oberbegriff.

DENISE

Der Chip DENISE erzeugt die Grafik und die Sprites des Amiga.

Device-Handler

(deutsch: Gerätetreiber) Jeder Computer enthält eine Anzahl von Ein-/Ausgabegeräten wie zum Beispiel Tastatur, Bildschirm oder Diskettenlaufwerke. Um diese Geräte anzusteuern, werden bestimmte ->Routinen benötigt, die teils schon im Rechner vorhanden, teils von einer Diskette nachgeladen werden können. Man nennt sie Device-Handler. Da man beispielsweise an seinem Parallelport verschiedene Geräte betreiben kann, ist es nur sinnvoll, Gerätetreiber auch nachladen zu können. Dadurch bleibt ein System flexibel und ausbaufähig.

Doppelklick

Ein zweimaliges ->Anklicken eines Symbols in kurzem Zeitabstand mit der Maus nennt man Doppelklick.

DOS

Abkürzung für Disk Operating System (deutsch: Diskettenbetriebssystem). Gemeint ist damit der Teil des ->Betriebssystems, der sich um die Verwaltung der Ein- und Ausgabe, in erster Linie bei den Diskettenlaufwerken, kümmert. Das beginnt beim Lesen und Schreiben von Bytes bis hin zur File-Verwaltung. Auch die ->RAM-Disk wird von Amiga-DOS kontrolliert, genauso wie die Druckerausgabe oder das Ausführen von Befehlsdateien. Die direkte Schnittstelle zwischen Amiga-DOS und dem Anwender ist normalerweise das ->CLI.

Drawer

->Ordner

Editor

Ein Editor ist ein Programm zum Erstellen von Programmtexten. Man spricht daher auch vom Editieren, seltener vom Edieren eines Programmes. In Amiga-BASIC hat der Editor auch die Funktion, das Austesten der Programme zu ermöglichen. Er arbeitet daher stark mit dem Interpreter zusammen. Anders dagegen die Sprache C. Hier kann man seine Programmtexte mit einem x-beliebigen Editor schreiben, zum Beispiel mit dem ed-Editor des ->CLI.

Einzelschrittmodus

Um Programme auf Fehler hin zu untersuchen, ist es oft sehr hilfreich, wenn man jeden Befehl einzeln abarbeiten lassen kann. Man sieht dann genau, was das Programm macht und kann auch immer alle Variableninhalte kontrollieren. Amiga-BASIC unterstützt ein solches Vorgehen durch seinen Einzelschrittmodus. Aber auch beim Austesten von Maschinenprogrammen kann eine Einzelschrittabarbeitung von Interesse sein.

EXEC

Einer der Teile des ->Betriebssystems trägt den Namen EXEC. Er ist in erster Linie für die Verwaltung des Speichers und des ->Multitasking zuständig und übernimmt somit die Kontrolle über alle laufenden Programme und die anderen Betriebssystemteile.

Fast-RAM

Die Amigas 500 und 1000 werden normalerweise mit 512 KByte ->RAM ausgeliefert. Alle Spezialchips wie zum Beispiel der ->Blitter können nur diesen unteren Speicherbereich adressieren, weshalb man hier von Chip-RAM spricht. Wenn nun sowohl der Prozessor als auch einer der Chips gleichzeitig versuchen, auf das Chip-RAM zuzugreifen, muß der Prozessor warten, bis der Chip fertig ist. Wenn man eine Speicher-Erweiterung in den Amiga einbaut, wie sie beispielsweise schon im Amiga 2000 vorhanden ist, braucht der Prozessor bei Zugriffen auf den neuen Speicher nicht mehr gebremst zu werden, da es hier zu keinen Kollisionen mit den anderen Chips kommen kann. Programme laufen also mit einer Speicher-Erweiterung schneller ab, weshalb man den zusätzlichen Speicher als Fast-RAM (deutsch: schnellen Speicher) bezeichnet.

Fenster

->Window

Festplatte

->Harddisk

File

->Datei

Flag

(deutsch: Flagge) Eine Speicherstelle, die nur die Werte Null und Eins annimmt und so beispielsweise eine Ja/Nein-Information speichert. Normalerweise genügt für ein Flag schon ein Bit, und man könnte in einem Byte bis zu acht Flags unterbringen. Meistens verwendet man aber für jedes Flag ein eigenes Byte, da sich das einfacher handhaben läßt.

Funktion

Der Begriff Funktion meint Unterschiedliches, je nachdem, auf welche Programmiersprache man sich bezieht. In BASIC sind Funktionen alle diejenigen Anweisungen, die einen Wert zurückliefern. Zum Beispiel ist PEEK(...) eine Funktion, PRINT dagegen nicht. Außerdem gibt es dort selbstdefinierte Funktionen mit dem Befehl DEF FN. Der Funktionsbegriff in C geht dagegen viel weiter. Hier nennt man jedes Unterprogramm Funktion, egal, ob es einen Wert zurückliefert oder nicht. Darunter fallen auch die Befehle, die als externe Unterprogramme implementiert sind, beispielsweise printf.

Gadget

Alle anklickbaren Symbole mit Ausnahme der ->Icons heißen Gadgets. Es gibt zum Beispiel String-Gadgets zur Eingabe von Text, Proportional-Gadgets (Rollbalken) oder Vorder-/Hintergrund-Gadgets zum Umschalten von Fenstern.

Guru Meditation

Wie Sie vielleicht schon aus eigener Erfahrung wissen, gibt es unterschiedliche Formen des Systemabsturzes. In den meisten Fällen stürzt der Amiga aber nicht so ab, als daß er nicht noch die Ursache des Zusammenbruchs anzeigen könnte. Dies geschieht dann mit einer Einblendung oberhalb des aktuellen Screens, in der unter anderem die Anzeige Guru Meditation Error steht.

Harddisk

(deutsch: Festplatte) Eine Harddisk ist ein Speichermedium ähnlich einer Diskettenstation, auf dem aber 10, 20 oder mehr Megabytes an Daten untergebracht werden können. Harddisks sind zudem schneller als Diskettenlaufwerke, haben allerdings den Nachteil ihres hohen Preises. Die Aufzeichnungsplatte läßt sich auch nicht ohne weiteres auswechseln, so daß man mit dem vorgegebenen Speicher auskommen muß. Normalerweise bringt man auf der Festplatte alle oft benötigten Programme unter und hat dann keinen Ärger mehr mit dem Heraussuchen von Disketten oder langen Ladezeiten.

Hardware

Die Bestandteile eines Computers und das Zubehör unterteilt man in die Kategorien Hardware und Software. Hardware ist alles das, was durch elektrische Schaltungen und Chips fest vorgegeben ist. Software dagegen bezeichnet alles, was sich durch Programmierung ändern läßt, also Programme und Daten. Zum Beispiel ist ->RAM-Speicher Hardware, während die Programme, die man ins RAM lädt, Software sind. Auch das ->Betriebssystem eines Computers ist Software, selbst dann, wenn es wie beim Amiga 500/2000 unveränderbar in ROM-Chips steckt.

Hexadezimalsystem

Das Hexadezimalsystem ist eine andere Form der Zahlendarstellung, die nicht wie gewohnt mit 10 Ziffern arbeitet, sondern mit 16. Man benutzt dazu die üblichen Ziffern und zusätzlich die Buchstaben a bis f. Die Ziffern in einer Zahl haben in dem uns vertrauten Dezimalsystem verschiedene Stellenwerte. Zum Beispiel hat die Zwei in der Zahl 234 die Bedeutung von 2 mal 100, die Drei die Bedeutung von 3 mal 10 und die Vier die Bedeutung von 4 mal 1. Im Hexadezimalsystem lauten die Stellenwerte von rechts nach links: 1, 16, 256 (=16*16), 4096 (=16*16*16) usw. Hexadezimalzahlen stellt man normalerweise die Zeichen $ oder 0x voran, um sie von Dezimalzahlen zu unterscheiden. Sie sollten sich unbedingt mit dem Hexadezimalsystem vertraut machen, wenn Sie in Maschinensprache pro-

grammieren wollen, da hier fast nur mit diesem System gearbeitet wird. Das Hexadezimalsystem bietet gegenüber dem Dezimalsystem den Vorteil, daß man es bei Speicheradressen viel öfter mit glatten Zahlen zu tun hat, also Zahlen, die mit Nullen enden. Manchmal wird statt des Hexadezimalsystems das Oktalsystem vorgezogen. Hier hat man dann nur 8 Ziffern zur Verfügung, wodurch man keine neuen Ziffern einführen muß.

Icon

(deutsch: Piktogramm) Disketten, Dateien und ->Ordner werden von der ->Workbench als kleine Bilder angezeigt, die man Icons nennt. Icons lassen sich anklicken oder verschieben. Mit dem Programm Icon-Editor können Sie sich auch eigene Icons malen.

IFF

(Interchange Format Files) Unter der Bezeichnung IFF faßt man mehrere standardisierte Dateiformate für Bilder, Texte, digitalisierte Sounds etc. zusammen. Alle Programme, die sich an diesen Standard halten, können problemlos Daten untereinander austauschen. Eines der am häufigsten gebrauchten IFF-Formate ist das ILBM-(InterLeaved BitMap)-Format für Grafikbilder.

Interlace

(deutsch: Zeilensprungverfahren) Der Amiga kann normalerweise 256 Punktzeilen untereinander auf dem Bildschirm darstellen. Dabei wird das Bild 50mal in der Sekunde zeilenweise aufgebaut. Um nun die Auflösung zu verdoppeln, verwendet man den sogenannten Interlace-Modus. Hier werden in der ersten 50stel-Sekunde die Zeilen 1, 3, 5 usw. dargestellt und in der zweiten 50stel-Sekunde die Zeilen 2, 4, 6 usw. Man erhält dadurch bis zu 512 Zeilen, was sich durch ein mehr oder weniger starkes Flackern bemerkbar macht.

Interpreter

->Compiler

Intuition

Genau wie EXEC oder DOS handelt es sich bei Intuition um einen Betriebssystemteil. Intuition verwaltet ->Screens, ->Windows, ->Icons und was sonst noch zur Benutzeroberfläche gehört.

Jokerzeichen

Jokerzeichen sind Zeichen, die als Ersatz für beliebige andere Zeichen stehen können. Das Fragezeichen ist beispielsweise für viele ->CLI-Befehle ein Jokerzeichen. Zum Beispiel löscht der Befehl delete?.test alle Dateien, deren Name aus einem Buchstaben und der Endung .test besteht.

kByte

1024 Bytes nennt man ein Kilobyte oder auch KByte.

Label

(deutsch: Markierung) In C, BASIC und Maschinensprache markiert man Positionen von Programmteilen oder Daten, zu denen man springen bzw. die man verwenden will, mit meist sinnvollen Namen, die man Label nennt. Dadurch benötigt man keine Zeilennummer oder Speicheradresse, sondern kann statt dessen mit dem Namen arbeiten.

Library

(deutsch: Bibliothek) Die für den Programmierer verfügbaren Routinen des ->Betriebssystems des Amiga sind jeweils in zusammengehörige Gruppen unterteilt, die sich Libraries nennen. Jede Library läßt sich über eine Sprungtabelle erreichen, in der die Startdadressen aller Programmroutinen abgelegt sind. Die am meisten gebrauchten Libraries haben die Namen ->EXEC, ->DOS, ->Intuition und Graphics.

Menü

Ein Menü ist eine Auswahlliste zum Aufruf bestimmter Programmfunktionen. Ein Menü besteht immer aus einer Reihe von Menüpunkten, von denen sich einer durch Drücken einer Taste oder Anklicken mit der Maus auswählen läßt. Eine besondere Form von Menüs stellen die sogenannten Pull-Down-Menüs dar. ->Screens haben oft eine Menüleiste. Durch Gedrückthalten der rechten Maustaste erscheinen in der Menüleiste mehrere Menütitel. Wenn der Mauszeiger auf einen Titel geführt wird, klappt automatisch ein Menü herunter, in dem man nun einen Menüpunkt auswählen kann.

Multitasking

Das quasi gleichzeitige Ablaufen mehrerer Programme. Der Prozessor im Amiga kann natürlich keine zwei Befehle auf einmal ausführen. Daher wechselt er von Programm zu Programm und arbeitet jedesmal eine Zeitlang die Anweisungen eines Programmes ab. Dem Anwender erscheint es so, als ob die Programme parallel abliefen.

Offset

Tabellenzugriffe werden in Maschinensprache oft folgendermaßen abgewickelt: Man nimmt eine Basisadresse, meistens den Tabellenanfang oder das Tabellenende, her und erreicht nun die einzelnen Tabellenelemente durch Addition oder Subtraktion eines Wertes von der Basisadresse. Den Wert, den man addiert oder subtrahiert, bezeichnet man als Offset.

Oktalsystem

->Hexadezimalsystem

Ordner

(auch Subdirectory, Drawer) Ein Ordner ist eine Datei, die sich genau wie ein ->Directory verhält. Das bedeutet, ein Ordner kann weitere Dateien und Ordner enthalten. Ordner dienen zum logischen Gruppieren von Dateien unter einem Namen.

Overlay

(deutsch: Überlagerung) Wenn ein Programm so lang ist, daß es nicht auf einmal in den Rechnerspeicher paßt, greift man auf die Overlay-Technik zurück. Man sucht dazu zwei Programmteile, die nicht unbedingt gleichzeitig im Speicher vorhanden sein müssen, und benutzt für beide den gleichen Speicherbereich. Jedesmal, wenn der Programmteil aufgerufen wird, der gerade nicht zur Verfügung steht, wird er von der Diskette nachgeladen. Man sollte bei Verwendung der Overlay-Techink darauf achten, daß nicht zu oft Diskettenzugriffe erforderlich sind.

Parameter

Parameter nennt man Werte, die an einen Befehl oder eine Funktion übergeben werden. Mit Parametern bezeichnet man allerdings oft auch Voreinstellungswerte in jeglicher Hinsicht.

PAULA

Dieser Chip des Amiga ist für die Tonerzeugung und die Ansteuerung der Diskettenlaufwerke verantwortlich. Außerdem kontrolliert er die gesamte Interrupt-Verwaltung.

Piktogramm

->Icon

Pixel

Der kleinste darstellbare Punkt auf einem Bildschirm bei einer gegebenen Grafikauflösung.

Puffer

->Buffer

Pull-Down-Menü

->Menü

Preferences

(deutsch: Voreinstellungen) Preferences ist ein Programm auf der Workbench-Diskette, das zum Ändern vieler Einstellungen dient. Von den Bildschirmfarben bis zur Druckereinstellung lassen sich damit jegliche Parameter vorgeben und auch auf der Diskette abspeichern, so daß alles auch nach dem Ausschalten des Rechners und Neubooten der Workbench erhalten bleibt.

Prozessor

Der Prozessor ist die eigentliche Zentraleinheit jedes Computers. Er ist der einzige Chip, der komplexe Befehle verstehen und ausführen kann. Seine Geschwindigkeit bestimmt die Geschwindigkeit aller ablaufenden Programme. Dies stimmt aber beim Amiga nur zum Teil, da die Hilfschips Blitter und Copper dem Prozessor viel Arbeit abnehmen können. Der Prozessor des Amiga trägt die Bezeichnung MC68000, der des C64 heißt 6510 und der im C128 8502.

Rasterstrahl

Ein Fernseh- oder Monitorbild wird 50mal in der Sekunde durch einen Elektronenstrahl aufgebaut. Dieser erzeugt das Bild Zeile für Zeile von oben nach unten. Das menschliche Auge ist natürlich zu träge, um eine solch schnelle Bewegung noch mitverfolgen zu können. Da der Fernsehschirm in ein Punktraster aufgeteilt ist, spricht man von einem Rasterstrahl.

RAM

(Random Access Memory). RAM ist Rechner-Speicher, in den sich Daten vom Prozessor aus schnell schreiben und aus dem sie sich genauso schnell wieder lesen lassen. Die Abkürzung RAM steht im Gegensatz zu ROM (Read Only Memory). Speicher dieser Art enthält fest vorgegebene Werte, die von keinem Programm geändert werden können und auch nach dem Ausschalten nicht verlorengehen. Eine interessante RAM-ROM-Kombination befindet sich im Amiga 1000. Der RAM-Bereich, in dem das

Betriebssystem liegt, wird nach dem Einladen so geschaltet, als ob es sich dabei um ROM handeln würde. Dadurch kann selbst ein abgestürztes Programm das Betriebssystem im Speicher weder verändern noch zerstören. Nach dem Ausschalten geht dieser Bereich jedoch verloren.

RAM-Disk

Die RAM-Disk ist ein Teil des Speichers, der genau wie eine Diskettenstation behandelt wird. Man kann demnach Dateien in die RAM-Disk kopieren und von dort aus weiterverarbeiten. Zugriffe auf die RAM-Disk erfolgen normalerweise wesentlich schneller als auf eine echte Diskettenstation. Die RAM-Disk hat aber den Nachteil, daß ihr Inhalt nach dem Ausschalten des Rechners oder nach einem Reset verlorengeht.

Register

Mit Register bezeichnet man einen Teil des ->Prozessors, der den Inhalt einer Speicherzelle aufnehmen und verarbeiten kann. Beim 68000-Prozessor unterscheidet man zwischen Daten- und Adreßregistern. Die Adreßregister dienen als Zeiger auf Speicheradressen. Ein besonderes Register ist das Statusregister. Die einzelnen Bits in ihm zeigen an, wie eine vorhergehende Operation abgelaufen ist, ob zum Beispiel das Ergebnis des letzten Befehls den Wert Null hatte.

relative Datei

Bei ->Dateien unterscheidet man zwischen sequentiellen und relativen Dateien. In sequentiellen Dateien werden alle Daten hintereinander in der Reihenfolge abgelegt, in der man sie benötigt. Man kann die Datei dann nur vom Anfang her beginnend lesen und weitere Daten nur am Ende anhängen. Relative Dateien dagegen werden in Datensätze unterteilt, so daß man über die Angabe der Nummer eines Datensatzes direkt die Stelle auf der Diskette findet, an der sich die gewünschten Daten befinden. Auf diese Weise können auch Daten mitten in der Datei abgeändert werden. Amiga-DOS legt alle Dateien so an, daß sie sowohl relativ als auch sequentiell bearbeitet werden können.

ROM

->RAM

Routine

Kurzer Programmteil oder kurzes Hilfsprogramm. Anstelle von Unterprogrammen spricht man auch oft von Unterroutinen.

Schnittstelle

Jede Verbindung eines Computers mit seiner Außenwelt heißt Schnittstelle. Zum Beispiel ist die Tastatur die Schnittstelle zwischen Mensch und Computer. Meistens meint der Begriff Schnittstelle aber die Anschlußmöglichkeiten für Peripheriegeräte. Manchmal werden aber auch Programme als Schnittstelle bezeichnet, beispielsweise das ->CLI als Schnittstelle zum Amiga-DOS.

Screen

(deutsch: Bildschirm) Screens sehen aus und verhalten sich so wie übereinandergelegte Bildschirme. Jeder Screen kann eine andere Auflösung und andere Farben beinhalten. Im Gegensatz zu Fenstern können Screens nur übereinander liegen, nicht jedoch nebeneinander. ->Windows sind jeweils einem Screen zugeordnet und an dessen Grafikmodus gebunden. Screens können wie der Workbench-Screen eine obere Leiste besitzen, anhand derer man sie verschieben kann. Wenn man einen Screen mit den Vorder-/Hintergrundsymbolen nach vorne bringt, erscheinen natürlich alle seine Fenster vor allen anderen Screens. Auch Screens dienen zur Unterstützung des ->Multitasking, da sie es ermöglichen, daß mehrere Programme gleichzeitig mit unterschiedlichen Grafikauflösungen und Farben ablaufen können.

Sektor

->Track

sequentielle Datei

->relative Datei

Software

->Hardware

Supervisor-Modus

Ein spezieller Modus des 68000-Prozessors. Er ist normalerweise dem Betriebssystem vorbehalten. Bestimmte Steuerfunktionen sind nur im Supervisor-Modus möglich, u.a. das Umschalten der verschiedenen Interrupt-Ebenen.

Stack

(deutsch: Stapel) Ein Stack ist ein Speicherbereich mit einem Zeiger darauf, dem sogenannten Stackpointer. Auf dem Stack lassen sich Daten ablegen, und man kann die Daten wieder zurückholen. Dabei wird immer das Datum zuerst zurückgeholt, das zuletzt abgelegt wurde. Der Stack verhält sich also quasi wie ein echter Stapel, bei dem das, was zuoberst liegt, auch zuerst wieder weggenommen werden muß. Eine solche Struktur wird vorzugsweise für das Ablegen von Rücksprungadressen verwendet.

Statusregister

->Register

String

Zeichenketten bezeichnet man auch mit dem englischen Fachbegriff String. Es handelt sich dabei um eine Reihe aufeinanderfolgender beliebiger Zeichen, wie zum Beispiel eine Textzeile.

Subdirectory

->Ordner

Syntax

Jede Sprache, sei es nun eine Programmier- oder eine richtige Sprache, hat einen Satz von Regeln, die bestimmen, wie die Teile der Sprache angeordnet werden dürfen. In einer Programmiersprache muß beispielsweise zu jeder öffnenden Klammer auch eine schließende Klammer gehören. Die Gesamtheit aller dieser Regeln nennt man Syntax. Die Syntax entscheidet nicht darüber, ob ein Ausdruck sinnvoll ist oder nicht. Zum Beispiel ist die BASIC-Anweisung a=3/0 zwar syntaktisch richtig, aber logisch falsch.

Token

Alle uns bekannten BASIC-Interpreter speichern die Befehlsworte nicht vollständig mit allen Buchstaben ab, sondern benutzen statt dessen ein oder zwei Bytes als Abkürzung. Deshalb muß man beispielsweise bei Amiga-BASIC unterscheiden, ob man ein Programm normal abspeichert oder im ASCII-Format. Die für die Befehle intern verwendeten Codes heißen Token.

Tracemodus

In Amiga-BASIC unterscheidet man zwischen einem Trace- und einem ->Einzelschrittmodus. Im Trace-Modus kann man sehen, welcher Befehl gerade abgearbeitet wird. Jedoch stoppt das Programm nicht nach der Ausführung jedes einzelnen Befehls. Wenn man in bezug auf Maschinensprache von einem Trace-Modus spricht, meint man meistens Einzelschrittabarbeitung.

Track

Eine Diskette ist in mehrere Spuren (engl. Tracks) unterteilt. Jeder Track enthält wiederum eine bestimmte Anzahl von Sektoren, die aus einer festgelegten Anzahl von Bytes aufgebaut sind. Beim Amiga befinden sich auf jeder Diskettenseite 80 Tracks à 11 Sektoren mit je 512 Byte. (->Cylinder)

Userport

(beim C64/C128) Die 24-polige ->Schnittstelle auf der Rückseite Ihres Gerätes nennt sich Userport. Dieser Port wird zum Beispiel für eine RS232-Schnittstelle benötigt.

Window

(deutsch: Fenster) Windows sind rechteckige Bereiche auf dem Bildschirm, die zur Ausgabe von Bildern und Texten dienen. Ein Programm könnte natürlich alle Ausgaben ohne Verwendung von Fenstern direkt auf den Bildschirm bringen. Dies würde aber zu Problemen mit dem ->Multitasking führen,nämlich dann, wenn zwei Programme gleichzeitig versuchen würden, Bildschirmausgaben zu tätigen. Geschieht die Ausgabe dagegen in Fenstern, dann können problemlos auch Überlagerungen von Ausgaben gehandhabt werden. Dem Anwender wird so ermöglicht, selbst zu bestimmen, welcher Teil des Bildschirms für welche Ausgabe gerade benutzt werden soll.

Workbench

(deutsch: Werkbank, Arbeitstisch) Die Workbench ist ein Programm, das es ermöglicht, alle wichtigen Diskettenfunktionen über Symbole, ->Windows oder ->Pull-Down-Menüs mit Hilfe der Maus abzuwickeln. Dadurch wird die Bedienung des Rechners äußerst benutzerfreundlich. Die Leistungsgrenzen der Workbench werden allerdings da erreicht, wo Dinge nicht mehr sichtbar und daher von der Workbench aus auch nicht mehr veränderbar sind. Nach einem Reset verlangt der Amiga grundsätzlich nach einer Workbench-Diskette. Sie müssen nun aber nicht unbedingt die Workbench einlegen, sondern können auch irgendeine andere Diskette mit einem selbstladenden Programm ins Laufwerk schieben.

Stichwortverzeichnis

#define 206
#include 206, 208
-Windows 56
.bmap-Dateien 112
.fd-Dateien 112
.info-Dateien 62
.key 60
4-Kanal-Sound-Erzeugung 163

Ablaufgeschwindigkeit 177
Absolut 143
Adreßberechnung 145
Adreßregister 132, 139
Adresse einer beliebigen Variablen 198
Adressierungsarten 136
Adreßzeiger 142
Agnus 160
Akkumulator 132
AmigaBASIC 67
AmigaDOS 23, 51
Amplitudenmoduliert 163
Analog-/Digital-Wandler 163, 172
Analoges Signal 172
Änderung 57
Änderung des Ausgabegeräts 58
Anklicken 14
Anweisung 180
Anwenderbyte 133
AREA 100
AREAFILL 100
Arithmetische Ausdrücke 194
Arithmetische Befehle 147
Arrays 196
ASCII-Code 182
Ask 61
Assembler 135, 136
Assemblersyntax 145

Assign 30, 35, 57
Audio-DMA 165
Aufruf von Betriebssystemfunktionen 184
Ausgabeformatierungen 183
Auswahltaste 14
Auto-Variablen 204

Backspace 180
BAM 52
BASIC 67
BASIC-Interpreter 67
BASIC-Programm 271
Basisadresse 212
Batchdateien 60
BCD 130
BCD-Befehle 148
BCD-Ziffer 148
Bedingte Sprünge 150
Bedingungen 186
Befehle des MC68000 im Überblick 151
Befehlsblock 186
Befehlsdateien 58
Befehlsnotation 135
Befehlssatz 76, 146
Befehlssatz in Kategorien 146
Benchmark 215
Benchmark-Programme 216
Benutzung der Betriebssystemroutinen 111
Bibliotheken 111
Bibliotheksfunktionen 212
Bildschirmformate 161
Binärdatei 124
Bit 130
Bit-Blitter 161
Bitmanipulationsbefehle 149
Bitplane-Zeiger 171
Bitplanes 168, 171
Block-Availability-Map 52
Boot-Sektor 51
Boot-Vorgang 29

Branch-Befehle 144, 150
Break-Befehl 190
Break-Flag 134
Byte 130, 208

C64 als Interface 281
Calculator 65
CALL 82, 114, 123
Carry-Flag 133
Case-Befehlen 191
Cd 27
Centronics-Schnittstelle 54
CHAIN 80, 86
CHDIR 88
Chip-Memory 159
CIAs 163, 235
CIRCLE 99
CLEAR 68, 77
CLI 20, 23, 64
CLI-Arbeitsdiskette 28
Clock 62
CLOSE 76, 91
Close-Gadget 16
CLR 77
CLS 84
CMD 77
COLOR-Befehl 96
COM1 83
Compiler 198, 199
Compiler-Sprache 177
CON 55
Conditioncode-Register 133, 138
Continue 190
Copper 160, 171
Copper-DMA 171
Copy 33
CSRLIN 84
Custom-Chips 129
CVD 94

CVI 94
CVL 94
CVS 94

Datamat 273
Datawandler 124
DATE$ 82
Dateien 89
Dateimaske 274
Dateiverwaltung 87
Daten aus Dateiverwaltungen 270
Daten ohne jegliche Konvertierung senden 271
Datenfeld 196, 206, 274
Datenformate 130
Datenlänge der Operanden 135
Datenregister 132
Datensatz 274
Datentransportbefehle 147
Datentyp 74, 182
Datenübertragung 269
Datenübertragung vom C64 zum Amiga 233
DECLARE FUNCTION 113
DEF FN 77
Default 191
Delete 31
Denise 161
Devices 54
Devs 64
Dezimal-Flag 134
Dezimale Konstanten 194
Dir 24
Directory 15, 51
Direkter Registerzugriff 164
Direkter Speicherzugriff 163
Direktzugriffsdateien 89
Disk-Validator 52, 53, 64
Diskcopy 64
Disketten 20
Diskettenlaufwerke 54
Diskettenstruktur 51

DMA-fähiger Diskcontroller 163
DMA-Kontrollregister 164
DMA-Logik 164
DMA-Übertragung 164
Doppelklick 14
DOS 51
DOS-Befehle 24
DOS-Systemroutinen 115
Double 201
Double Density 21
Drawer 26
Drucker 280
Druckerausgabe 55
Druckertreiber 64
Duplicate 19
Dynamische Speicherverwaltung 158

Ed 29, 64
Editor 69
Ein-/Ausgabe 83
Einrücken 71
Einrücken von Programmzeilen 69
Einzelschrittabarbeitung 72
Else 60, 78
Else-Zweig 189
Empty 65
Endcli 37
Endif 60
EOF 81, 91
Execute 59, 116, 119
Exists 60
Exklusiv-Oder-Verknüpfung 195
Expansion 65
Exportdatei 273
Extendflag X 134
Externe Variablen 203

Farbregister 96, 168
Fast-Memory 159
Fat Agnus 160

Fenster 15
FIELD 93
Files 88
Filetypen 52
Flag 132, 35
Fließkommavariablen 74
Floppy 285
Fonts 65
For-Anweisung 187, 197
Format 33
Formatiervorgang 19
Formfeed 180
FRE 77
Frei verschiebbare Programme 144
Frequenzmoduliert 163
Funktionen 179, 200

Gadgets 16
Ganzzahl 74
Geräte 54
GET 78, 92
GET# 78
Getchar 190
Globale Variablen 202
GOSUB 78
GOTO 78
Goto-Anweisung 192
Goto-Sprünge 192
Grafik 94
Grafikauflösung 168
Grafikspeicher 168
GraphicDump 47, 63
Grundrechenarten 181, 194

Harddisk 54, 65
Hardware-Sprites 162
Header 52
Hexadezimale Konstanten 194
Höhere Programmiersprachen 177
Hold-and-Modify-Modus 162

Icon 14, 60
Icon-Editor 45
IconEd 64
IconMerge 47
Icons 45
If Error 60
If Fail 60
If Warm 60
If-Anweisung 188
IF...THEN 78
IFF-Format 275
Implizit 142
Index 139, 196
Indirekte Adressierung 144
Indizes 199
Info 32
Initialisierung 196, 200
Initialize 19
Inkey$ 78
INPUT 79
INPUT$ 85
Install 51
Integer 74
Interlace-Modus 95
Interrupt-Flag 134, 135
Interrupt-Kontrollregister (ICR) 236
Interrupt-Verwaltung 163
Invertieren der Bits 195

Jokerzeichen 34
Jokerzeichen beim List-Befehl 35
Joystickabfrage 109

Keymaps 64
Kickstart 13
KILL 89
Klammersetzung 194
Koala-Format 277
Kommentar 181

Konstantenadressierung 142
Kontrollregister 167
Konvertierungstabelle 270
Kopieren kompletter Disketten 19
Kopieren von Dateien 18
Kopieren von einer Diskette 18
KYBD 83

Lab 61
Labels 73
Langwort 130
Lautstärke 163
Libraries 65, 111, 113, 212
Library-Struktur 212
LINE 97
LINE INPUT 79
Linefeed-Zeichen 180
Linken 203
Linksbündig 183
List 32, 53, 71
LLIST 84
LOAD 79
LoadSeg 122
LOC 91
LOCATE 84
LOF 92
Logische Befehle 148
Logische Operatoren 195
Lokal 202
LONG 208
Löschen von Dateien 18
LPOS(n) 84
LPRINT 84
LPRINT USING 84
LPT1 83
LSET 93

Main 179
Makro 207
Maschinenprogramm 121

Maus 14
Mauszeiger 14
Max 202
Mehrdimensionale Felder 197
MENU 104
MENU-Funktion 106
MERGE 80
Microsoft BASIC V2 67
MKD$ 94
MKI$ 93
MKL$ 93
MKS$ 93
Monitor 279
MountList 64

NAME 89
Negativ-Flag N 134
Newcli 37
Nibble 148
NIL 57
NMI 134
NoFastMem 47, 64
Notepad 65
Notizblock 44
Nützliche Betriebssystemroutinen 155

Oder-Verknüpfung 195
Offset 139
Oktale Konstanten 194
ON MENU GOSUB 108
ON MENU(0) GOTO 108
OPEN 76, 90
OpenLibrary 212
Operator 186, 193
Ordner 15, 26

PAINT 99
PALETTE 96
Parallel.device 64
Parallele Schnittstelle 163

Parallelport 54, 233
Parallelport-Treiber 269
Path 30
PATTERN 100
Paula 163, 172
PC 235
PEEK 80
PEEKL 80
PEEKW 80
Peripheriegerät 54
PI 81
Piktogramm 14
Platzhalter 181, 182
POKE 81
Port-Handler 64
POS 84
Postinkrement 140
Prädekrement 140
Präprozessor 206
Preferences 39, 62
PRESET 99
PRINT USING 84
Printer.device 64
Printers 64
Printf 180
Privilegierte Befehle 151
Privilegverletzung 135
Programmablauf 150
Programmierkurs 178
Programmierstil 206
Programmspeicher 68
Programmsteuernde Befehle 150
Programmzähler 132
Programmzähler Relativ 144
Protect 33
Prozessorregisters 204
PSET 99
Pull-Down-Menüs 14, 103
PUT 92

Quit 61

RAM-Disk 18, 26, 54
RAM-Erweiterung 48
RAM-Handler 64
RAW-Windows 56
ReadMe 32
Rechnen mit Adressen 198
Regeln beim Zugriff 136
Register Direkt 138
Register Indirekt 139
Register-Variablen 204
Rekursionen 204
Relative 52, 89
Rename 33
Reset 19
RESTORE 81
Return 81, 202
Rollbalken 16
RSET 93
Rückgabewert 202
Run 37, 58

SADD 115
SAVE 79
Say 64
Schleifen 184
Schreibschutz 21
Schubladen 26
SCREEN 95, 100
SCRN 83
Seek 116, 119
Sektoren 51
Senden von Textdateien 273
Sequentiell 52, 89
Serial.device 64
Serielle RS232-Schnittstelle 55
Serielle Schnittstelle 163
SetClock 63
SID 163

Single Density 21
Skip 61
SLEEP 83
Snapshot 45
Sound 109
Sound-Demo 173
Sounddaten 172
Speicherplatz 177
Sprungmarken 73
Sprungtabelle 212
Spuren 51
ST 81
Stackpointer 132, 140
Startup Sequence 29, 59, 64
Static 203
Statusregister 133
STOP 69, 73
STR$ 93
Strings 74
Strukturen 209
Superbase 273
Supervisor-Modus 134
Supervisorflag 135
Supervisorflag S 134
Switch 191
Switch-Anweisung 190
SYS 82
System Configuration 39, 42
Systembyte 133
Systemdateien 62
Systemsteuernde Befehle 151

Tabulator-Zeichen 180
Taschenrechner 47
Tastatur 64
Text-Labels 71
Textdateien und Dateien aus Dateiverwaltungen 273
Textomat 273
Textomat Plus 273
Textomat-Plus-Dateien 270

Textverarbeitungsdatei 273
TI 82
TI$ 82
TIME$ 82
Timer 82, 163
Trace-Modus 72
Traceflag 135
Traceflag T 135
Trackbuffer 22
Tracks 51
Type 32

Überlauf-Flag V 133
Übersichtlichkeit 186
Übertragen von Standard-ASCII-Dateien 270
Übertragung von Bildern 275
Übertragungskabel 269
Übertragungsprogramm 238
UBYTE 208
Uhr 44, 62
ULONG 208
Umbenennen einer Datei 17
Unbedingte Sprünge 150
Und-Verknüpfung 195
Union 211
UnLoadSeg 122
Unmittelbar 142
Unter-Directories 26
User-Modus 134
Userport 233
USR 82
UWORD 208

V=STICK(n) 110
V=STRIG(n) 111
VAL 93
Variablen 73, 181
Varianten 209, 211
VARPTR 127
Verbindungskabel 233

Vergleichszeichen 186
VERIFY 79
Verschiebe- und Rotationsbefehle 149
Verschiebeleiste 16
Verschobene Werte 195
Verwaltung der Grafik 161
Verzweigungsbedingungen 151
Void 201
Vordergrund-/Hintergrund-Gadgets 17

WAIT 83
WAVE 110
Wellenformen 172
Wert einer Speicherstelle 198
While-Schleife 185
WIDTH 84
WINDOW 15, 101
WORD 208
Workbench 14, 38, 62
Wort 130
WRITE 84
WRITE# 92

XClose 116, 117
XOpen 116
XRead 116, 118
XWrite 116, 118

Zeichenketten 74, 197
Zeiger 198
Zeigervariable für einen Struktur 210
Zeilennummer 69
Zeroflag Z 133
Zuweisungen 181, 192
Zuweisungsoperatoren 188, 195
Zwei-Adreß-Prozessor 135
Zweites Diskettenlaufwerk 48
Zylinder 51

DAS STEHT DRIN:

Ist der Amiga nicht Ihr erster Computer? Haben Sie vorher einen C64 oder C128 besessen oder besitzen ihn immer noch? Dann finden Sie in diesem Buch genau das, was Sie unbedingt brauchen: eine schnelle und gründliche Einführung in DOS, BASIC, C und Maschinensprache, die auf Ihrem bisherigen Wissen aufbaut. Damit aber nicht genug: Wir zeigen Ihnen, wie Sie bestehende Dateien jeglicher Art vom alten an den neuen Rechner senden und wie Sie vorhandene Peripherie weiterverwenden können. Die dafür notwendige Software finden Sie selbstverständlich als fertiges Listing zum Abtippen in diesem Buch.

Aus dem Inhalt:

- Der schnelle Einstieg (Workbench, CLI)
- Amiga-DOS (Diskettenstruktur, File-Typen, Gerätenamen, Befehlsdateien, Systemdateien)
- Vom BASIC V2 zum Amiga BASIC (was ist gleich geblieben, was hat sich geändert, Dateiverwaltung, Grafikbefehle, Benutzung der Bibliotheken, Einbinden von Maschinenprogrammen)
- Programmierung in 68000-Maschinensprache, Programmierung der Customchips
- Einführung in die Programmiersprache C
- Effizienzvergleich der verschiedenen Programmiersprachen: Wie schnell ist der Amiga wirklich?
- Datenübertragung vom C64 zum Amiga einschließlich Konvertierung (Textdateien, Dateien aus Dateiverwaltungen, BASIC-Programme, Grafiken)
- C64 als Drucker-Interface

UND GESCHRIEBEN HABEN DIESES BUCH:

Michael Strauch und Alexander Stellmach sind versierte Programmierexperten sowohl auf dem C64 als auch auf dem Amiga. Die zwei angehenden Informatikstudenten haben in dieses Buch neben ihrem umfangreichen Know-how besonders die Erfahrungen eingebracht, die sie in der Anfangszeit mit dem Amiga gemacht haben.

ISB N 3-89011-134-4 DM +039.00

DM 39,–
ÖS 304,–
sFr 37,–

DATA BECKER

9 783890 111346 03900