

Liesert

Das
endgültige
Atari ST
Handbuch

Das Buch, das keine Fragen offen läßt:
Von der Einführung zum TOS, über die Hardware
bis zur Programmierung – Praxis-Know-how von
A bis Z zum Atari ST.

DATA BECKER

H. J. Liesert

Das
endgültige
Atari ST
Handbuch

DATA BECKER

Copyright © 1991 by DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf 1

1. Auflage 1991

Umschlaggestaltung Werner Leinhos

**Textverarbeitung
und Gestaltung** Udo Bretschneider

Text verarbeitet mit Word 5.0, Microsoft

Belichtung Merlin Belichtungsstudio, Essen

**Druck und
buchbinderische Verarbeitung** Graphischer Großbetrieb Pößneck, Pößneck

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 ver-
arbeitet, vervielfältigt oder verbreitet werden.

ISBN 3-89011-383-4

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind für Amateur- und Lehrzwecke bestimmt.

Alle technischen Angaben und Programme in diesem Buch wurden vom Autor 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.

Wir weisen darauf hin, daß die im Buch verwendeten Soft- und Hardwarebezeichnungen und Markennamen der jeweiligen Firmen im allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Vorwort

Dieses Buch soll Ihnen als umfassendes Nachschlagewerk zu Ihrem Atari ST dienen. Das können Sie bereits am Umfang des Bandes ansehen. Wie Sie sicher schon vermutet haben, ist das nicht allein "auf meinem Mist gewachsen". Kräftig geholfen haben:

Uwe Braun
Rolf Brückmann
Stefan Dittrich
Lothar Englisch
Klaus Gerits
Martin Pauly
Axel Plenge
Frank Schepers
Axel Schramm
Jürgen Schulz
Rainer Severin

und last not least

Thomas Vervost

Ihnen allen gebührt mein herzlichster Dank.

Münster, im Juni 1991

Hans Joachim Liesert

Inhaltsverzeichnis

Teil A	15
1. Einführung	15
2. Das Desktop	19
2.1 Der Einschaltvorgang	19
2.2 Die Icons	24
2.2.1 Icons anklicken	26
2.2.2 Icon-Gruppen	34
2.2.3 Icons aus anderen Fenstern	35
2.2.4 Icons verschieben	36
2.3 Fenster	42
2.4 Drop-Down-Menüs	47
2.5 Das Menü des Desktop	48
2.5.1 Desk	48
2.5.2 Datei	51
2.5.3 Index	56
2.5.4 Extras	57
2.6 Der Umgang mit der Objektauswahl	63
2.7 Der Umgang mit Dialogboxen	67
3. Massenspeicherung	71
3.1 Das hierarchische File-System	71
3.1.1 Dateien	71
3.1.2 Ordner	77
3.1.3 Datei- und Pfadnamen	83
3.1.4 Extensions	86
3.2 Die Diskette	89
3.3 Die RAM-Disk	99
3.4 Die Festplatte	102
3.4.1 Installation einer Festplatte	102
3.4.2 Die Arbeit mit der Festplatte	107
3.5 Fehler beim Zugriff auf Datenstationen	113
3.6 Größenangaben	118
3.7 Datensicherung - ein ernstes Thema	121
4. Drucker	131
4.1 Die Hardware	131
4.2 Voreinstellungen	140
4.3 Der Drucker im Einsatz	143

4.4	Drucker und Anwendungsprogramme	148
4.5	Der Atari-Laserdrucker SLM804/SLM605	155
5.	Der Bildschirm	179
6.	Tastatur und Maus	185
7.	Schnittstellen	193
7.1	Die Centronics-Schnittstelle	197
7.2	Die RS-232-Schnittstelle	200
7.3	MIDI	208
7.4	Die Anschlüsse für Hard- und Floppy-Disk	210
7.5	Der Modulschacht	211
7.6	Die Joystickanschlüsse am STE	211
8.	Datenaustausch mit anderen Rechnern?	213
9.	Software	217
9.1	Textverarbeitung und Editor	218
9.2	Dateiverwaltungen und Datenbanken	225
9.3	Grafiksoftware	229
9.4	Fremde Betriebssysteme	236
9.4.1	Der PC-DITTO	237
9.4.2	Der Aladin-Macintosh-Emulator	249
9.4.3	MINIX	255
9.5	Viren	257
9.6	Kommando-Interpreter	262
9.7	Programmiersprachen	264
9.7.1	Die verschiedenen Sprachen	265
9.7.2	BASIC	266
9.7.3	C	270
9.7.4	Assembler	271
10.	Pflege und Wartung	273
10.1	Reinigung	274
10.2	Die Betriebssystem-ROMs	278
10.3	Rechner-Tuning	283
10.4	Kleine Reparaturen selbst gemacht	283
11.	Hardware-Unterschiede STE und Mega ST	285

Teil B	287
12. Die eingebaute Software	287
13. Programme unter GEM und TOS	289
13.1 TOS-Programme	289
13.2 GEM-Programme	295
13.3 Accessories	309
13.4 Objekte	311
13.4.1 Die TEDINFO-Struktur	315
13.4.2 Die ICONBLK-Struktur	316
13.4.3 Die BITBLK-Struktur	317
13.4.4 Die APPLBLK-Struktur	317
13.5 Windows und Ereignisse	318
13.6 Verwaltung der Dialogboxen	335
13.7 Programmierung der Pull-Down-Menüs	338
13.8 Objekte ohne Resource-Dateien	339
13.9 Ein Beispiel-Accessory	340
13.10 GEM und BASIC	346
13.11 Ein eigenes Desktop	347
13.12 Automatisch startende Programme	350
13.12.1 Aktuelle Uhrzeit resetfest	351
13.12.2 Automatische Druckeranpassung statt Accessory	356
13.12.3 GDOS und Accessory installieren	358
13.12.4 Ein Autostarter für GEM-Applikationen	360
13.13 GFA-BASIC und Assembler	363
14. Massenspeicher	371
14.1 Das Diskettenformat	371
14.2 Der Boot-Sektor	373
14.2.1 Ein Formatierungsprogramm	376
14.2.2 Der BIOS-Parameter-Block BPB	384
14.3 Das Inhaltsverzeichnis	390
14.4 Die FAT	393
14.5 Programmaufbau	395
14.5.1 Der Programm-Header	396
14.5.2 Die Relocation-Tabelle	397
14.6 Festplattenformat	399
14.7 Der Disk-Controller	400
14.7.1 Organisation	400
14.7.2 Kommando-Beschreibung	410
14.7.3 Status-Interpretation	442
14.8 Der Harddisk-Controller	447

14.8.1	Befehlsstruktur	450
14.8.2	Liste der Befehle	457
14.8.3	HDC-Tools	462
14.8.4	Partitions-Analysator	466
14.9	Die RAM-Disk	473
14.9.1	Ein komfortables RAM-Disk-Programm	475
14.10	Automatisches RAM-Disk-Kopierprogramm	485
14.11	Harddisk-Backup-Programm	491
15.	Grundlegende Grafikprogrammierung in BASIC und mit VDI ..	499
15.1	Kleine Grafik-Einführung	499
15.1.1	640x400 - hochaufgelöste Grafik (HGR)	499
15.1.2	640x200-mittlere Grafikauflösung (MGR)	501
15.1.3	320x200-niedrig aufgelöste Grafik (NGR):	502
15.1.4	Die Farben	503
15.2	Der Punkt als Urgrafik	505
15.2.1	PLOT oder POLYMARKER?	506
15.2.2	Farben bringen Farbe hinein	523
15.2.3	Die Zeichenmodi	538
15.3	Die Linie - Grundlage aller Grafik	552
15.3.1	Linien, Linienzüge, Polygone	552
15.3.2	Linie ist nicht gleich Linie	560
15.3.3	Das, was drin ist - Flächen ausfüllen	566
15.4	Rechtecke im Rampenlicht	577
15.5	Beliebige Flächen ausfüllen	586
15.6	Kreise und andere runde Sachen	593
15.7	Textdarstellung - gar nicht so einfach	607
15.7.1	Einfachste Textausgabe	608
15.7.2	Die Fonts und Schriftarten	620
15.7.3	Komfortabel: Textformatierung	633
15.8	Bildschirmfenster und Grafikausgabe	652
15.8.1	Ein Fenster zur Grafik - einfache Fenstertechnik	652
15.8.2	Kopieren und Überlagern	655
15.9	Sprites auch auf dem Atari ST	671
16.	Mehr zum Bildschirm	685
16.1	Das Video-RAM des Atari ST	685
16.2	Umschaltung von 50 auf 60 Hz für Farbmonitore	690
16.3	Bildschirmschoner	691
17.	Soundprogrammierung auf dem Atari ST	695
17.1	Der Sound-Befehl in GFA-BASIC	695
17.2	Mehr Töne über die Periodendauer	696
17.3	Neue Möglichkeiten mit dem Wave-Befehl	697

17.4	Sound in C und Assembler	699
17.5	Musik im Interrupt	701
Teil C		705
18. Die Komponenten des ST		705
19. Die ICs des Atari ST		707
19.1	Der Prozessor 68000	707
19.1.1	Die Register des 68000	708
19.1.2	Die Exceptions des 68000	711
19.2	Die Atari-Custom Chips	711
19.2.1	Die Register der Atari-Custom-Chips	716
19.3	Der Floppy-Disk-Controller WD 1772	722
19.3.1	Die Anschlüsse des 1772	723
19.3.2	Die Register des WD 1772	725
19.3.3	Die Programmierung des FDC	727
19.4	Der MFP 68901	730
19.4.1	Die Anschlüsse des 68901	731
19.4.2	Die Register des MFP	734
19.5	Die ACIAs 6850	747
19.5.1	Die Anschlüsse des 6850	748
19.5.2	Die Register des 6850	749
19.6	Der Sound Generator YM-2149	755
19.6.1	Die Anschlüsse des Sound Chip	756
19.6.2	Die Funktion der einzelnen Register des 2149	758
19.7	Der Atari-Blitter	762
19.7.1	Die Register des Blitters	764
19.8	Die Echtzeituhr im Mega-ST	777
19.9	Die Belegung des I/O-Bereichs des Atari ST	779
20. Die Schnittstellen		781
20.1	Die Tastatur	781
20.1.1	Die Maus	785
20.1.2	Keyboard-Kommandos	788
20.2	Die Video-Buchse	798
20.3	Der Centronics-Anschluß	802
20.4	Die RS232-Schnittstelle	803
20.5	Die Midi-Anschlüsse	805
20.6	Der Modulschacht	806
20.6.1	ROM-Cartridges	807
20.7	Die Floppy Disk-Schnittstelle	809
20.8	Die DMA-Schnittstelle	811
20.9	Der interne Erweiterungs-Bus der Mega-ST-Serie	817

Teil D	821
21. Die Funktionen des GEMDOS	821
22. Die BIOS-Funktionen des Atari ST	871
23. Die XBIOS-Funktionen des Atari ST	883
24. Der VT52-Emulator des Atari ST	919
25. Die Systemvariablen des Atari ST	923
26. Die LINE-A-Funktionen	935
27. GEM-VDI	961
28. Glossar	1049
Stichwortverzeichnis	1061

Teil A

1. Einführung

Wie jedes anständige Buch hat auch dieses ein Inhaltsverzeichnis. Und wenn Sie das bereits gelesen haben, ist Ihnen vielleicht aufgefallen, daß dieses Buch in vier Teile gegliedert ist. Ich habe mir gedacht, daß Sie in vier Fällen zu diesem Werk greifen werden:

1. Sie haben eine Frage zur Bedienung des ST. Das gilt übrigens auch für die sogenannten "Profis" unter Ihnen. Die erkennt man nämlich daran, daß auch sie einmal nicht wissen, wie man eine bestimmte Funktion des Desktop anspricht oder wie der Drucker konfiguriert werden kann. Sollten Sie aus einem solchen oder ähnlichen Grund zu diesem Buch greifen, ist Teil A für Sie gedacht. Hier finden Sie alles, was man bei der täglichen Arbeit mit dem ST wissen muß. Apropos Arbeit:
2. Wenn Sie Arbeit mit Programmieren verbinden (für manche Mitmenschen - den Autor dieser Zeilen eingeschlossen - ist das mindestens der zweit-schönste Beruf der Welt), dann ist wahrscheinlich Teil B für Sie interessant. Hier ist vor allem das "Wie" beschrieben, also Fragen wie "Wie programmiert man die GEM-Fenster?" oder "Wie programmiert man Accessories?"
3. Sie können auch zu den hartgesottenen Zeitgenossen gehören, die ganz genau wissen, an welchem Ende ein Lötkolben heiß wird. Dann juckt es Ihnen natürlich schon in den Fingern, etwas mehr mit der Hardware zu machen. Bitte sehr: Teil C sollte Ihnen weiterhelfen.
4. Der vierte Fall ist recht einfach, aber dafür um so häufiger: Sie möchten etwas nachschlagen. Seien es die Parameter eines Betriebssystemaufrufs oder die GEM-Funktionen, ein heißer Tip für Sie lautet: Sehen Sie in Teil D nach.

Das war also die Bedienungsanleitung für dieses Buch. Die Bedienungsanleitung für den ST folgt sogleich...

Den ST aufbauen

Bevor wir mit dem ST arbeiten können, muß er aufgebaut werden. Das hört sich schlimmer an als es ist, denn im Normalfall brauchen Sie dazu nicht einmal einen Schraubenzieher.

Übrigens: In diesem Kapitel beschreibe ich nur den Aufbau des Atari ST. Wenn Sie noch Zusatzgeräte wie einen Drucker oder eine Harddisk anschließen möchten, sehen Sie bitte in das entsprechende Kapitel dieses Buches.

Bevor Sie loslegen können, stellen Sie bitte erst fest, welchen Rechner Sie eigentlich vor sich haben. Der ST ist nämlich in verschiedenen Versionen gebaut worden (und wird es natürlich noch), so daß der Aufbau je nach Modell etwas differieren kann. Fangen wir mit den ältesten an. Am Anfang standen die Rechner 520 ST, 520 ST+ und 260 ST. Diese hatten noch kein Diskettenlaufwerk eingebaut, das dafür in einem eigenen Gehäuse residierte. Der Aufbau vollzieht sich folgendermaßen:

Zunächst sollten Sie den eigentlichen Rechner auf Ihren Schreibtisch stellen und dann natürlich auch den Monitor, das Diskettenlaufwerk (oder beide, wenn Sie zwei besitzen) und die Maus. Dann wird das Monitorkabel am ST angebracht. Es handelt sich dabei um einen 13-poligen DIN-Stecker, der nur in eine Buchse an der Rückseite des Gehäuses paßt. Achten Sie darauf, daß der Stecker richtig eingesteckt ist. Auch das Kabel des ersten Diskettenlaufwerks ist ziemlich unverwechselbar mit seinem dicken DIN-Stecker. Es wird ebenfalls an der Rückseite des ST eingesteckt; die andere Seite findet am Laufwerk in der Buchse "IN" Platz. Sollten Sie ein zweites Floppy-Laufwerk besitzen, wird das dazugehörige Kabel in die "OUT"-Buchse des ersten Laufwerks und das andere Ende in die "IN"-Buchse des zweiten gesteckt. Auch wenn es den Anschein hat: Mehr als zwei Diskettenlaufwerke können Sie an den ST nicht anschließen.

Jetzt kommt die Maus an die Reihe. Das Kabel muß an der rechten Seite des 520-ST-Gehäuses eingesteckt werden. Dort gibt es zwei identisch aussehende Stecker, von denen Sie den linken wählen sollten. Beim Mega ST wird die Maus in die Tastatur eingesteckt, während sie beim 1040 ST in den Gehäuseboden gesteckt wird.

Jetzt kommen die Netzteile an die Reihe. Das Netzteil für den ST wird an der Rückseite neben dem Ein-/Aus-Schalter eingesteckt, danach erst die Verbindung zum 220-Volt-Netz hergestellt. Achtung: noch nicht einschalten! Das Floppy-Netzteil wird in der gleichen Weise angeschlossen. Bleibt nur noch das Netzkabel für den Monitor: einstecken - fertig. Beim 1040er und beim Mega ST müssen Sie lediglich das Netzkabel an der Rückseite anbringen. Jetzt dürfen Sie die Geräte in der Reihenfolge Monitor - Laufwerk(e) - Rechner einschalten.

Beim Mega ST müssen Sie vorher lediglich noch das Tastaturkabel an der linken Seite des Rechners einstöpseln.

Noch ein Wort, bevor Sie mit der Lektüre dieses Buches beginnen. An einigen Stellen werden Ihnen im Text Sinnbilder begegnen, die folgende Bedeutungen haben:



Der Bleistift weist Sie auf Stellen hin, bei denen Sie nützliche Tips zum Umgang mit Ihrem ST erhalten.



Wenn bei einem Vorgang Gefahren oder Fallen auftreten, werden Sie durch dieses Dreieck darauf hingewiesen.



Besonders wissenswerte Informationen, etwa solche mit Hintergrundwissen, sind durch ein Quadrat gekennzeichnet. Achten Sie auf diese Stellen besonders!



Der Pfeil weist Sie auf mögliche Fehler und Fehlerquellen hin.



Zur Behebung von Fehlern haben wir das Häkchen-Symbol eingesetzt. Wo immer es auftaucht, finden Sie die Lösung zu Problemen und Fehlerquellen.



Wo immer eine besondere Bedienung vonnöten ist, finden Sie einen Kreis. Er zeigt Ihnen, wie Sie etwas richtig oder besser machen können.



Besonders wichtige Stellen sind mit diesem Finger markiert. Achten Sie auf diese Stellen besonders!

Das war's. Und jetzt hinein ins Vergnügen.

2. Das Desktop

Desktop heißt nichts anderes als Schreibtischplatte - und so sollten Sie das gewohnte Bild auf dem Monitor auch sehen: eben als Schreibtischplatte. Doch bevor wir dahin kommen und mit unserem Schreibtisch vernünftig arbeiten können, sind noch einige Vorarbeiten nötig. Wie Sie den Bleistift Ihres ST spitzen können, zeigt Ihnen dieses Kapitel.

2.1 Der Einschaltvorgang

Was soll daran beschreibenswert sein? Ich drücke den Knopf, und der ST läuft. Mitnichten! Es gibt da einige Regeln, die Sie auf jeden Fall beachten sollten.

Wichtig ist zum Beispiel die Reihenfolge, in der die Geräte eingeschaltet werden. Zuerst die Festplatte, falls vorhanden. In den nächsten 15 Sekunden dürfen Sie ruhig schon einmal nach Ihrer System-Diskette suchen, den Monitor sowie das Diskettenlaufwerk einschalten und die System-Diskette einlegen. Aber schalten Sie in diesen 15 Sekunden noch nicht den ST ein. Erst danach heißt es "Start frei" für den ST. Kurz nach dem Einschalten sollte das rote Lämpchen an der Diskettenstation (und vorher eventuell auch das an der Festplatte) aufleuchten. Das Lämpchen signalisiert, daß der ST dabei ist, Daten von der Diskette in den Speicher zu laden. Nach einer weiteren Wartezeit erscheint das Desktop auf dem Bildschirm, und Sie können anfangen zu arbeiten.

Auch dabei kann schon etwas schief gehen - nicht jeder steht jeden Tag gutgelaunt auf, auch unser ST nicht.



Nach dem Hochfahren des Systems erscheinen keine Icons auf dem Desktop, das ganze Bild ist bis auf die Menüleiste leer. Oder nach dem Einschaltvorgang erscheinen zwar Icons, aber es fehlen einige (z.B. die für die Harddisk).



Sie haben Ihr Diskettenlaufwerk nicht eingeschaltet oder die Stromversorgung ist nicht in Ordnung (z.B. Stecker nicht richtig eingerastet, Netzteil defekt). Damit man mit Diskettenlaufwerken arbeiten kann, muß der ST sie nach dem Einschalten erkennen können. Dies tut er, indem er einfach versucht, sie anzusprechen, was bei ausgeschalteten Laufwerken natürlich nicht klappt. Schalten Sie also alle Ihre Laufwerke ein. Dann schalten Sie den ST einmal kurz aus und dann wieder an. Sollte der

Fehler nicht behoben sein, ist Ihr Laufwerk wahrscheinlich defekt, bei Festplatten könnten aber auch noch andere Fehler in Betracht kommen (siehe Kapitel 3).



Nach dem Einschalten des Stroms beginnen die Laufwerke zwar zu laufen, aber es erscheint auch nach längerer Wartezeit kein Desktop. Statt dessen flimmert der Bildschirm hin und wieder, worauf der Einschaltvorgang scheinbar von neuem beginnt, oder es werden kleine Bomben auf dem Bildschirm angezeigt.



Bevor Sie die Fehlerbeseitigung beginnen können, müssen Sie ein Desktop zur Verfügung haben. Dazu brauchen Sie eine funktionierende Systemdiskette, und gerade auf dieser scheint ein Fehler zu sein. Also zuerst: Nehmen Sie eine formatierte, aber leere (oder eine garantiert fehlerfreie andere Systemdiskette), und starten Sie den ST noch einmal von vorn. Dann müßte nach kurzer Zeit das Desktop (ohne die Accessories) erscheinen.

Der beschriebene Fehler kann zwei Ursachen haben. Zum einen könnte es sein, daß eines der Accessories zerstört wurde (zum Beispiel durch Diskettenfehler). Ebenso besteht die Möglichkeit, daß Sie eine Datei (z.B. aus der Textverarbeitung) mit der Endung ACC benannt haben. Dann wird sie nach dem Einschalten für ein Accessory gehalten, und der ST versucht, Ihre Datei als Programm auszuführen. Die andere Fehlermöglichkeit liegt im AUTO-Ordner versteckt. Hier dürfen nur solche Programme stehen, die das GEM nicht benutzen. Sie erkennen das daran, daß in diesen Programmen weder die Pull-Down-Menüs noch Dialog- oder Fileselect-Boxen auftauchen. Wenn Sie sich nicht sicher sind, was nun die Ursache für den aufgetretenen Fehler ist, gehen Sie bitte so vor:

Benennen Sie bitte alle Dateien mit der Endung ACC um in "irgendwas.AC". Falls der Fehler nach einem erneuten Einschalten mit der fehlerhaften Diskette noch nicht verschwunden ist, liegt es wahrscheinlich am AUTO-Ordner. Sonst gehen Sie bitte hin und benennen das erste Accessory wieder in "irgendwas.ACC" um. Dann starten Sie nochmals. Falls jetzt der Fehler wieder auftritt, war das gerade umbenannte Accessory daran schuld. Entfernen Sie es von der Diskette (Löschen per Papierkorb!), und ersetzen Sie es ggf. durch eine "frische" Version, die Sie von der Originaldiskette kopieren. Auch hier nochmal der vielgehörte Rat: Arbeiten Sie niemals mit Originaldisketten, wenn es sich vermeiden läßt. Originale gehören in den Tresor oder eine sichere Ecke des Diskettenkastens! Den oben beschriebenen Vorgang wiederholen Sie für jedes einzelne Accessor, bis Sie alle durchprobiert haben.

War keines der Accessories für den Fehler verantwortlich, müssen Sie den AUTO-Ordner untersuchen. Potentielle Übeltäter sind alle Programme mit der Endung .PRG. Hier gehen Sie genau wie bei den Accessories vor. Zunächst werden alle Extensions um einen Buchstaben gekürzt. Dabei wechselt der Typ des Icons (aus dem Block mit Perforation wird ein Papierstapel mit Eselsohr), was Sie nicht stören sollte. Dann gehen Sie wieder an jedes einzelne Programm heran und stellen die alte Endung wieder her (PR -> PRG). Nach jeder Änderung müssen Sie den Rechner erneut hochfahren und überprüfen, ob der Fehler immer noch auftritt. Dieses Verfahren ist umständlich und langwierig, aber nicht zu vermeiden. Ist der Übeltäter einmal gefunden, fliegt er sofort aus dem Ordner. Auch wenn es sich dabei um ein sonst funktionierendes Programm handelt, kann es doch sein, daß es im AUTO-Ordner einen Fehler verursacht.

Letzte Möglichkeit: Die Diskette ist beschädigt. Die Diskette sollte dann in Richtung eines (echten) Papierkorbs wandern. Falls Sie die Dateien auf der Diskette noch benötigen, kopieren Sie sich diese zuvor auf eine frisch formatierte. Das Kopieren sollte aber nicht dadurch ausgelöst werden, daß Sie zwei Disketten-Icons übereinanderlegen. Vielmehr müssen Sie die Dateisymbole aus dem Fenster auf das Symbol der neuen Diskette schieben!

War diese gesamte Fehlersuche erfolglos, so liegt mit großer Wahrscheinlichkeit ein Fehler in der Hardware vor; der Rechner muß zur Reparatur (geben Sie dabei die Systemdiskette mit). Falls Bomben auf dem Schirm erscheinen, zählen Sie diese bitte, und nennen Sie die Zahl der Werkstatt. Letztere kann aus der Anzahl gewisse Rückschlüsse auf die Ursache ziehen.



Nach dem Erscheinen des Desktop fehlen die Accessories.



Hier gibt es zwei Möglichkeiten. Entweder lag während des gesamten Einschaltvorganges keine Diskette im Laufwerk. Von einer nicht existenten Diskette kann der ST natürlich keine Dateien laden, erst recht keine Accessories. Oder die eingelegte Diskette enthielt keine Accessories.



Nach dem Einschalten erscheint nicht das Desktop, sondern ein Bild mit der Mitteilung, man solle eine Systemdiskette einlegen und dann mit der Maus auf den OK-Knopf in der Bildmitte klicken.

Desktop erscheint nicht



Sie haben einen Uralt-Veteranen-ST vor sich, der das Betriebssystem noch nicht im ROM eingebaut hat. Lesen Sie bitte das Kapitel über die TOS-ROMs.



Es erscheint eine Meldung, die angibt, daß eine Resource- oder RSC-Datei fehlt.



Eines der Accessories benötigt ein solche RSC-Datei (siehe Kapitel über Massenspeicherung). Sie haben wahrscheinlich vergessen, diese mit dem Accessory zusammen zu kopieren. Holen Sie dies bitte nach.

Und jetzt beherrscht noch eine Frage den ST-Besitzer: Was steckt eigentlich hinter all dem Aufwand mit dem Hochfahren? Vorweg ein Wort an die Festplattenbesitzer. Das frühe Einschalten der Harddisk ist nötig, um dem Motor genug Zeit zum Beschleunigen zu lassen. Anders als bei Diskettenlaufwerken dreht sich die fest eingebaute Platte immer, solange der Strom eingeschaltet ist. Diese Platte ist sehr viel schwerer als eine Diskette, so daß das Beschleunigen und Abbremsen immer einen recht langen Zeitraum in Anspruch nimmt. Sobald der ST eingeschaltet wird, muß er aber schon auf die Daten der Festplatte (für die Experten: den Boot-Sektor) zugreifen können. Deshalb braucht dieses Gerät eben einen kleinen Vorsprung. Der Lohn der Mühe: Im Vergleich zur Diskette kann die Harddisk fast immer sagen "Ich bin schon da!".

Wenn dann der ST endlich aus seinem Schummer geweckt wird, tut er das, was alle kleinen Computer tun: Er versucht, ein Programm auszuführen. Dieses Programm ist die sogenannte Boot-Sequenz. (Obwohl es nichts mit dem Buch von L.G. Buchheim zu tun hat, sprechen einige Freaks in den USA von "Das Boot"). "Boot" kommt von Bootstrap-Loader (zu deutsch: Urlader - die wörtliche Übersetzung macht einfach keinen Sinn). Dabei handelt es sich um den Teil des Betriebssystems TOS, das die für den Betrieb des ST nötigen Programme und Daten zusammensucht. Das versucht der ST zunächst auf der Festplatte. Dort findet er im allgemeinen nichts, also geht er zur Diskette über. Falls er dort ein Betriebssystem findet (das muß nicht das eingebaute sein), so wird dies geladen und gestartet. Andernfalls startet der ST das in den ROMs eingebaute Betriebssystem, oder - wenn Ihr ST noch keine Betriebssystem-ROMs hat - er versagt seinen Dienst. Das Betriebssystem erkennt der Rechner daran, daß im allerersten Sektor der Diskette (zum Begriff Sektor siehe Kapitel 3) ein ausführbares "Programmchen" liegt. Dieses Mini-Programm sorgt dann dafür, daß das eigentliche Betriebssystem korrekt geladen wird - im Normalfall zumindest. Da diese Boot-Programme automatisch ausgeführt werden, sind einige unangenehme Zeitgenossen auf die Idee gekommen, hierin die allseits gefürchteten Viren unterzubringen. Keine Bange, auch dagegen ist ein Kraut gewachsen.

Damit wären wir an dem Zeitpunkt angelangt, wo das TOS (bzw. ein anderes Betriebssystem) angefangen hat zu arbeiten. Die Boot-Sequenz ist aber noch nicht abgeschlossen. Jetzt kommen nämlich die sogenannten Treiber an die

Reihe. Wie fast jeder Computer weiß auch der ST nicht von vornherein, welche Geräte vorhanden sind und wie er diese ansprechen kann. Dafür gibt es spezielle Programme, die quasi Erweiterungen des Betriebssystems darstellen. Wenn der Prozessor dann z.B. auf eine Festplatte zugreifen will, so braucht er nur das Treiberprogramm aufzurufen, das dann für die nötigen Aktionen sorgt. Diese Treiber holt sich der ST während des Bootens von der Diskette. Allerdings gibt es davon einige Ausnahmen. So ist zum Beispiel der Treiber für die Bildschirmausgaben fest in den ST eingebaut - logisch, denn den Bildschirm und die Tastatur hat das System immer "dabei". Auch ein Druckertreiber für Epson-Drucker ist schon vorhanden. Wenn Sie aber Drucker am ST betreiben möchten, die nicht zu den Epson-Geräten kompatibel sind, so müssen Sie ein solches Treiberprogramm (das der Hersteller oder Fachhändler Ihnen besorgen kann) auf Ihre Systemdiskette kopieren.

Einen ganz ausgefuchsten Trick haben sich die Entwickler für die Festplatte einfallen lassen. Im ROM des ST ist nämlich eine Art Mini-Treiber vorhanden. Wenn die Festplatte entsprechend vorbereitet ist, kann dieser Mini-Treiber den eigentlichen Festplattentreiber direkt von der Festplatte laden. Damit ist es dann möglich, das ganze System direkt von der Harddisk aus zu starten, ohne eine Systemdiskette zu benutzen (mehr dazu in Kapitel 3).

Beim Laden der Treiberprogramme muß der ST in der Lage sein, diese von den anderen "normalen" Programmen zu unterscheiden. Dafür gibt es eine besondere Einrichtung der Boot-Sequenz: den AUTO-Ordner. Das ist ein ganz normaler Ordner auf einer Diskette, der sich von anderen nur durch den Namen unterscheidet. Er heißt schlicht und einfach "AUTO". Beim Booten sucht der ST auf der Systemdiskette nach einem Ordner dieses Namens und startet dann der Reihe nach alle darin vorhandenen Programme, sofern diese die Endung PRG besitzen. Das müssen nicht unbedingt Treiberprogramme sein; Sie können auch ein Programm zum Stellen der Uhr da hineinkopieren. Dann werden Sie bei jedem Neustart des ST aufgefordert, die Uhrzeit und das Datum einzugeben. Allerdings hat diese Methode einen kleinen Haken. Nicht jedes Programm kann so gestartet werden. Beim Ansprechen des AUTO-Ordners ist zwar das Betriebssystem TOS aktiv, aber das GEM wurde noch nicht gestartet. Alle Programme, die irgendwie auf Dialogboxen, Pull-Down-Menüs oder Fenster zugreifen, laufen daher ins Leere, wenn sie auf diese Art und Weise aufgerufen werden. Also Vorsicht! Wollen Sie GEM-Programme automatisch starten, können Sie dies ab TOS 1.04 mittels der Funktion "Anwendung anmelden" erreichen (siehe dort). In den älteren Versionen ist dies nicht möglich.

Zurück zum Boot-Vorgang. Zum Glück ist die GEM-lose Zeit jetzt zuende, denn im nächsten Schritt wird dieses Programm im ST aktiviert. Festplattenbe-

sitzer haben übrigens jetzt schon einen Geschwindigkeitsvorteil, denn da der Treiber bereits installiert ist, können alle noch folgenden Aktionen von der Harddisk aus erfolgen. Das sogenannte Desktop erscheint auf dem Bildschirm, ebenso die bekannten Icons. Was da sonst noch erscheint (und wo), das entnimmt das GEM der Datei DESKTOP.INF. Sie können diese Datei leicht selbst erzeugen, indem Sie einfach den Menüpunkt "Arbeit sichern" anklicken. Dann merkt sich das GEM alle bisher gemachten Einstellungen wie z.B. Fenstergröße, Icon-Positionen usw. in eben dieser Datei DESKTOP.INF. Beim nächsten Booten liest das GEM diese Daten wieder ein und kann den Bildschirm daher genauso wieder herrichten, wie er beim letzten Abspeichern ausgesehen hat.

Bevor dies allerdings geschieht hat die Boot-Sequenz noch nach den ersten sechs Dateien mit der Endung ACC gesucht. Dabei handelt es sich bekanntlich um die Accessories, die in der linken Menüspalte aufgerufen werden können. Diese Programme werden nun geladen (bis zu sechs - mehr verkraftet das Menü nicht) und warten ab jetzt auf ihren Aufruf. Geschafft! Das System ist bereit für Ihre Aktivitäten.

Nur eines sei noch erwähnt: Die Systemdiskette schlechthin gibt es nicht. Sie können dazu im Normalfall jede beliebige, formatierte Diskette nehmen. Je nachdem, welche Programme und Accessories auf dieser Diskette vorhanden sind, wird der Boot-Vorgang von Fall zu Fall unterschiedlich ablaufen. Wenn Sie also Ihr System mit einer Diskette starten, auf der kein Treiber (zum Begriff "Treiber" siehe Kap. 3) für eine evtl. vorhandene Harddisks ist, so kann der ST die Festplatte auch nicht benutzen. Je nachdem, welche Tätigkeiten Sie mit Ihrem ST ausführen wollen, kann es auch sinnvoll sein, mehrere solcher Systemdisketten anzulegen, auf denen dann die für den jeweiligen Zweck geeigneten Treiber und Accessories gespeichert sind.

2.2 Die Icons

Die kleinen Bilder, mit denen das Desktop geschmückt ist, sind verantwortlich dafür, daß Ihr ST so leicht zu bedienen ist. Sie heißen Icons. Für unterschiedliche Zwecke gibt es auch unterschiedliche Icons. Hier sehen wir einmal alle im Bild:

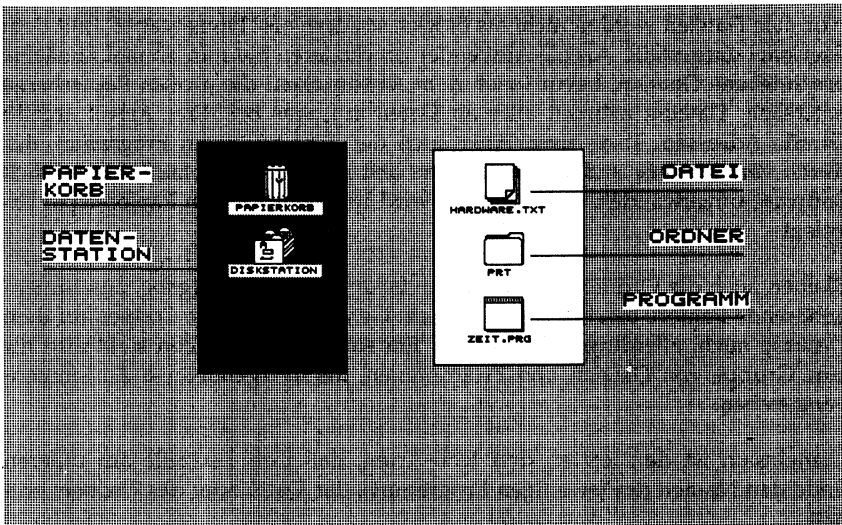


Abb. 2.1: Alle Standard-Icons des ST

Die gezeigten Icons werden in dieser Form vor allem vom Desktop verwendet. Andere Programme können eigene Icons benutzen, die nicht unbedingt das gleiche Aussehen und die gleiche Bedienung aufweisen. Beginnen wir ganz links. Der Aktenschränk steht für eine Datenstation. Das können die Diskettenlaufwerke, eine Festplatte, eine RAM-Disk oder ein ROM-Modul sein. Wie bei richtigen Aktenschränken haben auch die genannten Geräte die Aufgabe, große Mengen von Daten zu sammeln und aufzubewahren. Die Zeile unter dem eigentlichen Bild sollte im Normalfall angeben, um welche Art Datenstation es sich handelt. Sie können diesen Text auch selbst ändern (siehe unter "Floppy anmelden"). Im Icon selbst finden Sie außerdem den Kennbuchstaben, unter dem eine Station ansprechbar ist (z.B. "A" für das erste Diskettenlaufwerk).

Die Abfalltonne heißt im üblichen ST-Sprachgebrauch Papierkorb. Ihre Aufgabe ist das Löschen von Dateien. Mehr dazu weiter unten in diesem Kapitel.

Als nächstes kommen die Icons, die im Fenster erscheinen. Die Fenster des Desktop zeigen Inhaltsverzeichnisse von Datenstationen an. Demzufolge stehen die verschiedenen Icons für die möglichen Arten von Dateien. Die Blocks mit der Perforation am oberen Ende stellen Programme dar, die Sie starten können und die dann irgendeine Funktion erledigen. Die Idee dahinter: Wer auf einem normalen Schreibtisch etwas zu berechnen hat, tut dies meist auf einem solchen Block.

Normale Dateien werden nicht als Blocks, sondern als "Papierstapel" (mit dem Eselsohr) dargestellt. Als ST-Anwender produzieren Sie nun einmal zunächst beschriebene Dateien, bevor (nach dem Ausdrucken) die großen Papierberge entstehen. Dateien nehmen alles an Daten auf, was bei Ihrer Arbeit anfällt: Grafik, Adressen, Texte usw. Aber auch der Text zu einem Programm, der durch verschiedene andere Programme erst noch in ein ablauffähiges Stück Software umgewandelt werden muß, wird in einem solchen Papierstapel abgelegt.

Bleiben noch die Ordner. Ordner haben bekanntlich die Aufgabe, zusammengehörende Papiere beieinander zu halten. Genau dies tun die Ordner auf dem Desktop auch, allerdings mit Dateien. Ein solches Ordner-Icon steht also für eine Gruppe von Dateien und Programmen, die in irgendeinem Zusammenhang stehen.

Gemeinsam ist den letzten drei Icon-Arten, daß ihre Textzeile am unteren Ende den Namen des jeweiligen Programms, der Datei oder des Ordners enthält.

2.2.1 Icons anklicken

Wenn man dem ST irgendwelche Aktionen entlocken möchte, tritt meist die Maus auf die Bildfläche. Eine besonders gelungene Kombination ergeben Maus und Icons. Grundsätzlich gilt dabei folgendes: Ein Druck auf die linke Maustaste wählt ein bestimmtes Objekt aus, ein Doppelklick auf die linke Taste "öffnet" ein Objekt, ein Druck auf die rechte Taste sorgt dafür, daß sich der Druck der linken Taste auch auf ein nicht aktives Fenster beziehen kann.

Wenn Sie einen einfachen Klick auf dem Icon einer Datenstation ausführen, wird dieses invertiert. Damit zeigt uns das Desktop an, daß es die meisten der jetzt noch folgenden Aktionen auf dieses Icon bezieht. Wenn Sie also jetzt den Menüpunkt "öffne" anwählen, wird ein Fenster auf dem Desktop erscheinen, das das Inhaltsverzeichnis der eben angeklickten Datenstation anzeigt. Das gleiche können Sie aber auch erreichen, wenn Sie einen Doppelklick auf dem Icon ausführen.

Einen Doppelklick auf ein Objekt ausführen heißt, daß Sie zwei Klicks sehr schnell hintereinander durchführen. Die Geschwindigkeit, mit der das geschehen muß, können Sie selbst einstellen (siehe unter "Kontrollfeld"). Falls Sie also mit dem Doppelklick zunächst keinen Erfolg haben, könnte das daran liegen, daß dieser zu schnell eingestellt ist. Ansonsten können Sie sich mit dem

Menüpunkt "öffne" behelfen, denn damit wird genau das gleiche bewirkt. Nicht vergessen, daß der Doppelklick etwas Übung erfordert, also nicht verzagen.

Öffnen können Sie grundsätzlich jedes Icon auf dem Desktop. Programm starten Je nach Art des angeklickten Icons ist aber die Reaktion sehr unterschiedlich. Datenstationen werden geöffnet, um ein Fenster mit deren Inhaltsverzeichnis zu erhalten. Das Öffnen von Programm-Icons hat zur Folge, daß das Programm geladen wird und dann startet. Nehmen wir an, Sie möchten ein Programm mit dem Namen PROG.PRG starten, dann führen Sie einfach einen Doppelklick oder das "öffne" des Menüs auf diesem Icon aus. Sobald Sie den Mausknopf wieder loslassen, verschwindet das Desktop, und in der obersten Zeile steht der Name des Programms (in unserem Beispiel PROG.PRG) zum Zeichen dafür, daß gerade geladen wird. Nach einer mehr oder minder kurzen Wartezeit ist dieser Vorgang beendet, und das Programm beginnt mit seiner Arbeit.

So sehr Sie sich auch anstrengen werden, auch beim Klicken auf die Objekte kann es Pannen geben:



Nach einem Doppelklick auf ein Programm erscheint eine Warnmeldung, daß die Resource- oder RSC-Dateien nicht gefunden werden konnten. Das Programm wird nicht gestartet.



Zu den meisten Dateien mit der Endung PRG gehört eine weitere Datei mit der Endung RSC, in der Informationen über die Menüs und Dialogboxen enthalten sind. Wahrscheinlich haben Sie diese Datei beim Kopieren vergessen. Sehen Sie bitte in der Anleitung des Programmes nach, wo Sie diese Datei finden, und kopieren Sie diese auf Ihre Diskette.



Nach einem Doppelklick auf ein Programm erscheint die Meldung "Die gesuchte Anwendung kann nicht geöffnet werden!", das Programm startet nicht.



Sie haben versucht, ein nicht existentes Programm zu starten. Wahrscheinlich hatten Sie gerade eine andere Diskette eingelegt und das Fenster zeigte noch das Inhaltsverzeichnis der alten Scheibe an. Abhilfe in solchen Fällen: Nach jedem Diskettenwechsel die **ESC**-Taste drücken. Das bewirkt ein Aktualisieren des Fensters.



Nach dem Doppelklick erscheint eine Meldung, daß die Daten auf der Diskette defekt oder das Verbindungskabel nicht in Ordnung seien.

✓ Prüfen Sie zunächst, ob das Kabel zum jeweiligen Laufwerk wirklich in Ordnung ist (Laufwerk eingeschaltet?). Falls ja, könnte die Diskette oder Festplatte defekt sein. Sehen Sie dazu bitte im Kapitel 3 nach.

Öffnen Sie eine normale Datei (Icon-Typ Papierstapel mit Eselsohr), so erscheint folgende Auswahlbox auf dem Bildschirm:

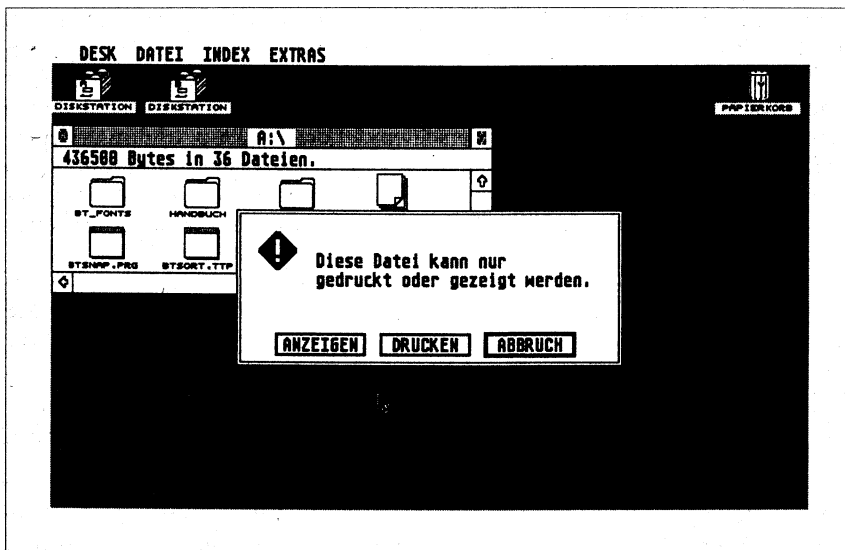


Abb. 2.2: Auswahlbox bei Doppelklick auf Dateien

Die angewählte Datei ist kein Programm und kein Ordner, enthält also (mit gewissen Ausnahmen) Daten - z.B. Texte, Adressen usw. Die Auswahlbox bietet Ihnen nun drei Möglichkeiten an: "Abbruch" klicken Sie an, wenn Sie irrtümlich die Datei geöffnet haben und zurück zum Desktop möchten. "Anzeigen" bedeutet, daß der ST die angewählte Datei auf dem Bildschirm ausgibt. Wenn Ihre Datei Texte enthält, können Sie sich so ein Bild von deren Inhalt machen. In anderen Fällen finden Sie vielleicht nur ein wildes Gewirr von Zeichen auf dem Bildschirm. Nicht-verzweifeln, das bedeutet nur, daß das Programm, mit dem diese Datei angelegt wurde, die Daten speziell codiert hat, um sie später besser und/oder schneller verarbeiten zu können.

Die Ausgabe der Datei sollte jeweils nach einer Bildschirmseite anhalten. Gleichzeitig erscheint in der untersten Zeile der Hinweis "--MEHR--". An dieser Stelle haben Sie wiederum drei Möglichkeiten. Durch einen Druck auf die **Leertaste** wird die nächste Seite angezeigt, durch **Return** - oder **Enter** -

Taste nur die nächste Zeile. Außerdem können Sie die Ausgabe abbrechen, indem Sie die **Ctrl**-Taste gedrückt halten und gleichzeitig ein "c" tippen (Kurzschreibweise unter uns Profis: **Ctrl**-**C**). Andere Möglichkeit für den Abbruch: q drücken.

Sollten Sie schon das Ende der Datei erreicht haben, so erscheint der Hinweis "-Ende der Datei-". Mit einem Druck auf die **Leer**-, **Return**- oder **Enter**-Taste kommen Sie dann wieder auf das Desktop zurück.

Wenn Sie im Besitz von TOS 1.04 sind, haben Sie noch eine weitere Möglichkeit. Durch einen Druck auf **a** können Sie den Text halbseitenweise durchlaufen lassen. Zusätzlich kann die Anzeige auch noch mit UNDO abgebrochen werden.

Zurück zur Auswahlbox. Die letzte Alternative dort lautet "Drucken". Nach der Wahl dieses Punktes versucht der ST wiederum, die Datei anzuzeigen - diesmal allerdings auf dem Drucker. Das funktioniert natürlich nur, wenn der Drucker angeschlossen, eingeschaltet und beim Desktop richtig angemeldet ist (siehe unter "Drucker-Anpassung"). Da Papier außerdem geduldiger als ein Bildschirm ist, braucht der ST auch keine Fragen wie "--MEHR--" zu stellen. Der Ausdruck läuft in einem Zug durch.

In TOS 1.04 haben Sie die Möglichkeit, das Drucken durch **q** oder UNDO ab-zubrechen.

Wie das nun einmal so ist, auch beim Anzeigen von Dateien gibt es Fehler.



Beim Anzeigen auf dem Bildschirm laufen mehr Zeilen durch als auf eine Bildseite passen. Der Hinweis "--MEHR--" wird viel zu spät angezeigt.



Hier handelt es sich um einen Programmierfehler im Desktop. Sobald Zeilen angezeigt werden sollen, die länger als 80 Zeichen sind (Bildschirmbreite), zählt der ST dies wie eine Zeile, obwohl auf dem Bildschirm unter Umständen 2, 3 oder mehr Zeilen belegt werden. Abhilfe ist mit dem folgenden Programm möglich:

Version für GFA-BASIC:

```
' BROWSER: Programm zum Anzeigen eines Files
' File-Namen ermitteln
Fileselect "A:\*.*,",F$
If (F$<>"") And (Exist(F$)) Then
  Open "I",#1,F$
  Dateilaenge%=Lof(#1)
```



```

Print "File-Name ";F$, "Dateilänge = ";Dateilaenge%
Print
Print "Space = nächste Seite, Return = nächste Zeile, Esc = Stop";
Print
Print "[Taste]"
Print "-----"
Void Inp(2)
' Zunächst Bildschirm füllen
I%=1
While (I%<25) And (Not Eof(#1))
  Gosub Zeile
  Inc I%
Wend
' Schleife zur weiteren Anzeige der Datei
While (Not Eof(#1)) And (A%<>27)
  Print "## Mehr? ##"
  A%=Inp(2)
  If A%=32 Then
    ' nächste Seite anzeigen
    I%=1
    While (I%<24) And (Not Eof(#1))
      Gosub Zeile
      Inc I%
    Wend
  Endif
  If A%=13 Then
    ' nächste Zeile anzeigen
    Gosub Zeile
  Endif
Wend
If Eof(#1) Then
  ' Wenn Dateiende erreicht, Programm abbrechen
  Print "Ende der Datei [Taste]";
  Void Inp(2)
Endif
Endif
Close #1
' Prozedur zum Anzeigen einer Zeile (Fließtext)
Procedure Zeile
  Local Z%,J%
  J%=1
  Z%=0
  While (J%<=80) And (Z%<>13) And Not Eof(#1)
    ' Zeichen aus Datei lesen
    Inc J%
    Z%=Inp(#1)
    Out 5,Z%
  Wend
  If Z%=13 Then
    Out 5,Inp(#1)
  Endif
  If J%<80 Then
    Print
  
```

```
Endif
Return
Version für ST-BASIC/Omikron-BASIC
0 ' BROWSER: Programm zum Anzeigen eines Files
1 CLS
2 ' File-Namen ermitteln
3 Pfad$="A:\*.*)"
4 MOUSEON
5 FILESELECT (Pfad$,F$,Ok%L)
6 MOUSEOFF
7 CLS
8 P%L= INSTR( MIRROR$(Pfad$),"\")
9 F$= LEFT$(Pfad$, LEN(Pfad$)-P%L+1)+F$
10 IF Ok%L=1 THEN
11 OPEN "I",1,F$
12 Dateilaenge%= LOF(1)
13 PRINT "File-Name ";F$,"Dateilänge = ";Dateilaenge%
14 PRINT
15 PRINT "Space = nächste Seite, Return = nächste Zeile, Esc = Stop";
16 PRINT
17 PRINT "[Taste]"
18 PRINT "-----"
19 A$= INPUT$(1)
20 ' Zunächst Bildschirm füllen
21 I%=1
22 WHILE (I%<25) AND ( NOT EOF(1))
23 Zeile
24 I%=I%+1
25 WEND
26 ' Schleife zur weiteren Anzeige der Datei
27 WHILE ( NOT EOF(1)) AND (A$<> CHR$(27))
28 PRINT "## Mehr? ##"
29 A$= INPUT$(1)
30 IF A$=" " THEN
31 ' nächste Seite anzeigen
32 I%=1
33 WHILE (I%<24) AND ( NOT EOF(1))
34 Zeile
35 I%=I%+1
36 WEND
37 ENDIF
38 IF ASC(A$)=13 THEN
39 ' nächste Zeile anzeigen
40 Zeile
41 ENDIF
42 WEND
43 IF EOF(1) THEN
44 ' Wenn Dateilende erreicht, Programm abbrechen
45 PRINT "Ende der Datei [Taste]";
46 A$= INPUT$(1)
47 ENDIF
48 ENDIF
49 CLOSE 1
```

```

50 END
51 ' Prozedur zum Anzeigen einer Zeile (Fließtext)
52 DEF PROC Zeile
53 LOCAL Z$,J%
54 J%=1
55 Z$=""
56 WHILE ((J%<80) AND (Z$<> CHR$(13)) AND ( NOT ( EOF(1))))
57   ' Zeichen aus Datei lesen
58   J%=J%+1
59   Z$= INPUT$(1,1)
60   PRINT CHR$(1);Z$;
61 WEND
62 IF ASC(Z$)=13 THEN
63   PRINT CHR$(1); INPUT$(1,1);
64 ENDIF
65 IF J%<79 THEN
66   PRINT
67 ENDIF
68 RETURN

```

Nachdem Sie das Programm gestartet haben, erscheint eine normale Objektauswahl auf dem Bildschirm. Hier können Sie die Datei auswählen, die danach angezeigt werden soll. Wenn Sie Abbruch anklicken, erscheint wieder das Desktop. Ansonsten zeigt das Programm zunächst noch einmal den Dateinamen und zusätzlich die Länge dieser Datei an. Hinzu kommt eine kleine Bedienungsanleitung. Erst auf beliebigen Tastendruck hin beginnt dann die Anzeige des Dateiinhalts. Wie bei der Desktop-Funktion auch haben Sie die Möglichkeit, die Ausgabe durch **Leertaste** (Space) um eine Seite oder durch **Return** bzw. **Enter** um eine Zeile weiterfahren zu lassen. Ein Abbruch ist jederzeit durch **Esc** möglich. Im Gegensatz zur Desktop-Funktion zeigt das obige Programm auch die sogenannten Steuerzeichen mit an. Das sind jene Zeichen, mit denen z.B. in Texten Attribute wie "Fettdruck" oder ähnliches gekennzeichnet werden. Auf dem Bildschirm erscheinen dafür Sonderzeichen wie z.B. Glocken, Pfeile o.ä. Das Desktop dagegen unterdrückt solche Zeichen nach Möglichkeit.



Die Anzeige einer Datei scheint sich im Kreis zu bewegen, ein Ende wird nicht erreicht.



In einem solchen Fall hat sich das Desktop in der internen Organisation der Diskette verheddert, was vor allem bei Dateien vorkommt, die keine lesbaren Daten, sondern Programmcode enthalten (z.B. Accessories). Sollte irgendwann danach noch der "--MEHR--"-Hinweis angezeigt werden, haben Sie Glück gehabt. **Ctrl-C** oder **q** bringt Sie wieder zurück ins Desktop. Erscheint dieser Hinweis nicht, bleibt Ihnen nichts anderes, als den Reset-Knopf an der Rückseite des ST gedrückt zu halten, dann eine Systemdiskette einzulegen und mit dem Loslassen der Reset-Taste den

Rechner komplett neu zu starten. Im Normalfall sind dann Daten, die auf einer RAM-Disk gespeichert waren, verloren (Ausnahme: resetfeste RAM-Disks).



Sie haben versehentlich die Alternative "Drucken" angewählt, obwohl kein Drucker angeschlossen oder eingeschaltet war.



Warten Sie mindestens dreißig Sekunden lang. Falls das Desktop sich dann wieder meldet, hat der ST dieses Problem selbst erkannt und die Ausgabe abgebrochen. Ansonsten hilft nur der Griff zum Reset-Taster.



Nach Auswahl von "Drucken" tut sich am Drucker nichts, obwohl dieser angeschlossen und eingeschaltet ist.



Es könnte sein, daß Ihr Drucker auf "Off-Line" geschaltet ist, d.h. er ist zur Zeit nicht in der Lage, Daten vom Rechner zu übernehmen. Die meisten Drucker besitzen auf dem Bedienfeld eine Taste, die mit "On-Line" oder "Off-Line" bezeichnet ist. Drücken Sie diese einmal. Tut sich dann immer noch nichts, besteht die Möglichkeit, daß der Drucker falsch beim Desktop angemeldet wurde (siehe "Drucker-Anpassung") oder schlimmstenfalls, daß der Drucker bzw. das Verbindungskabel defekt ist.



Das Ausdrucken von Dateien klappt zwar, aber es erscheinen andere Zeichen auf dem Papier als Sie das vom Bildschirm her gewohnt sind.



Hier scheint es sich um einen typischen Fall von fehlerhafter Druckeranpassung zu handeln. Schlagen Sie bitte im Druckerkapitel nach.



Nach einem Doppelklick auf eine Datei erscheint nicht die Auswahlbox, sondern es wird ein Programm gestartet.



Die Datei war über "Anwendung anmelden" einem Programm zugeordnet. Sie kann daher nicht mehr angezeigt werden, sondern lädt immer automatisch das Programm.

Was passiert, wenn ich Ordner öffne? Das Öffnen eines Ordners geht ziemlich realitätsnah ab, denn mit dieser Aktion sieht man buchstäblich in einen Ordner hinein. Für den ST bedeutet das nichts anderes, als daß er das Inhaltsverzeichnis des Ordners anzeigt. Falls in diesem Ordner-Verzeichnis wieder ein Ordner steht, können Sie auch diesen öffnen und gelangen in dessen Inhaltsverzeichnis usw. Der Weg zurück ist auch sehr einfach - klicken Sie einfach die Schließbox in der linken oberen Ecke an, und die Anzeige wechselt in den letzten Ordner zurück.

2.2.2 Icon-Gruppen

Sie können auf dem Desktop und in den meisten Programmen nicht nur ein Icon anwählen, sondern vielfach auch mehrere gleichzeitig. Die einfachste Möglichkeit dazu ist, alle nacheinander anzuklicken. Wenn Sie dies versuchen, werden Sie schnell merken, daß das jeweils zuletzt invertierte Bild wieder auf normale Darstellung umspringt, wenn Sie ein neues anklicken. Das ändert sich, sobald Sie eine der **[Shift]**-Tasten gedrückt halten. Üben wir das einmal Schritt für Schritt: Falls Sie auf dem Bildschirm noch nicht ein Inhaltsverzeichnis mit mindestens drei Dateien sehen, holen Sie sich bitte eines heran. Jetzt klicken Sie bitte eine beliebige Datei einmal an. Diese wird sofort invertiert.

Nun drücken Sie eine der beiden **[Shift]**-Tasten und halten sie so fest. Klicken Sie jetzt eine weitere Datei an, so wird auch diese invertiert, ohne daß sich die Darstellung der anderen ändern würde. Das Spielchen läßt sich mit weiteren Dateien weitertreiben, bis schließlich alle Icons schwarz gefärbt sind.

Wenn Sie in dieser Situation neben eines der Icons klicken, werden allerdings alle Dateien wieder abgewählt. Sie sollten also beim Positionieren ein wenig aufpassen.

Diese **[Shift]**-Tasten-Methode ist sehr günstig, wenn Sie nur wenige Dateien auszuwählen haben. Anders sieht das aus, wenn Sie ein ganzes Inhaltsverzeichnis mit 30 oder mehr Icons anwählen möchten. Für solche Situationen gibt es einen schnelleren Weg. Bewegen Sie bitte den Mauszeiger in die linke obere Ecke des ersten Datei-Icons im Fenster, und gehen Sie dann noch etwa einen halben Zentimeter nach links, so daß der Zeiger gerade noch im Fenster, aber nicht mehr auf dem Icon steht.

Wenn Sie jetzt die linke Maustaste gedrückt halten (bitte nicht loslassen), erscheint ein kleines Rechteck auf dem Bildschirm, das mitwächst, wenn Sie den Mauszeiger nach rechts unten bewegen. Man nennt dieses Rechteck wegen dieser Eigenschaft auch oft Gummiband-Rechteck. Probieren Sie das ruhig aus, und positionieren Sie die Maus ungefähr so, daß mehrere Icons innerhalb dieses Rechtecks liegen. Jetzt können Sie endlich die Maustaste loslassen. Wenn Sie genau darauf achten, werden Sie feststellen, daß jetzt alle die Icons invertiert sind, die ganz oder teilweise innerhalb des Rechtecks lagen.

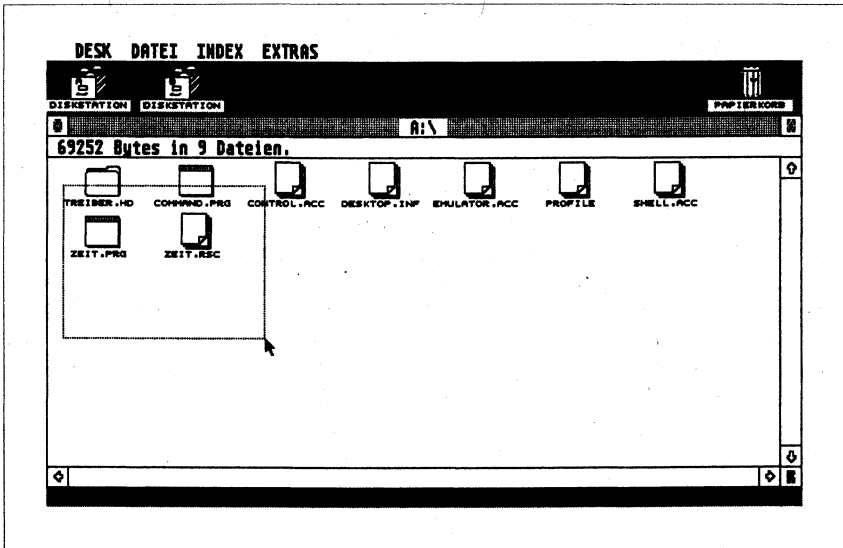


Abb. 2.3: Das Gummiband

Die Gummiband-Methode läßt sich mit der **[Shift]**-Tasten-Methode kombinieren, aber nur eingeschränkt. So können Sie zuerst mit dem Gummiband eine gewisse Anzahl Dateien "einfangen" und danach mit der **[Shift]**-Taste noch einige dazu holen. Umgekehrt funktioniert das leider nicht: Das Benutzen eines Gummibandes löscht jede frühere Auswahl!



Haben Sie sich einmal mit der Auswahl einer Datei geirrt, ist das auch kein Problem. Sie brauchen nicht noch einmal mit der ganzen Prozedur von vorne zu beginnen. Halten Sie die **[Shift]**-Taste gedrückt, und klicken Sie noch einmal auf die bewußte Datei. Damit wird Sie abgewählt und erscheint wieder schwarz auf weißem Grund. Alle anderen Icons bleiben davon unbeeinflusst (aber nicht vergessen: **[Shift]** drücken!).

2.2.3 Icons aus anderen Fenstern

Alle bisherigen Beschreibungen bezogen sich auf den Fall, daß nur ein Fenster auf dem Desktop liegt. Dann ist es auch klar, daß die Icons (soweit es sich nicht um Datenstationen oder den Papierkorb handelt) nur aus diesem einen Fenster stammen können. Was aber, wenn Sie mehrere Fenster geöffnet haben. Dann ist immer nur eines aktiv, während die anderen zwar sichtbar sind, aber

nur die Icons aus dem aktiven Fenster können Sie direkt anklicken. Erst gibt nun zwei Wege, eine Datei aus einem nicht aktiven Fenster zu aktivieren. Der einfachste ist wohl, das gewünschte Fenster zu aktivieren (durch Mausklick) und dann erst die Datei auszuwählen. Das ist aber nicht immer wünschenswert, etwa, wenn man mit dem Laufwerk A arbeiten, aber vorher kurz eine Datei aus einem anderen Laufwerk anzeigen lassen möchte. In diesem Fall hilft die rechte Maustaste. Wird sie niedergedrückt, so ist damit die Beschränkung auf ein Fenster aufgehoben, und man kann alle sichtbaren Icons benutzen. Ein Beispiel: In einem nicht aktiven Fenster liegen eine Datei, ein Ordner und ein Programm. Wenn Sie nun die rechte Maustaste gedrückt halten, können Sie mit der linken Maustaste jedes der drei Icons anklicken (worauf diese invertiert werden) oder durch Doppelklick die Datei anzeigen lassen, den Ordner öffnen oder das Programm starten. In der gleichen Weise können Sie das Gummiband und die Shift-Tasten-Methode aus dem letzten Abschnitt anwenden (wichtig: die ganze Zeit über muß die rechte Taste niedergehalten werden). Dieses Verfahren funktioniert sogar beim Kopieren von einem Laufwerk zum anderen!

Die rechte Maustaste können Sie nicht bei Datenstationen und dem Papierkorb einsetzen (wozu auch?).

2.2.4 Icons verschieben

Das Verschieben von Icons kann zwei Zwecken dienen. Icons, die nicht in einem Fenster liegen, verschiebt man, um sie an andere, übersichtlichere Plätze auf dem Bildschirm zu bringen oder um ganze Disketten zu kopieren. Icons aus Fenstern, die für irgendeine Datei stehen, verschiebt man, um sie zu kopieren. Das Vorgehen ist in beiden Fällen das gleiche.

Unter TOS 1.04 gibt es noch eine weitere Möglichkeit. Dort kann man Dateien regelrecht verschieben, d.h. es wird nicht kopiert, sondern die Datei wandert an einen anderen Ort innerhalb der Datenstation.

Um ein Icon zu bewegen, muß es zunächst invertiert, also angewählt werden. Dann schieben Sie den Mauszeiger auf das gewählte Icon und halten die linke Maustaste gedrückt. Solange Sie die Taste unten halten, erscheint an der Position des Mauszeigers ein kleiner Rahmen, der das Icon symbolisieren soll. Dieser Rahmen wandert nun mit dem Mauszeiger dorthin, wo Sie das Icon zukünftig positioniert sehen möchten, bzw. wo Sie die Datei hinkopiert haben möchten. Sobald Sie die Maustaste loslassen, erfolgt die eigentliche Reaktion des Desktop. Damit werden wir uns später beschäftigen.

Das Verschieben von Icon-Gruppen funktioniert ganz genau wie bei einem einzelnen Bild. Zunächst müssen Sie die Gruppe auswählen, dann können Sie verschieben. Dazu bewegen Sie den Mauszeiger auf ein beliebiges invertiertes Icon und drücken - wie gehabt - die linke Maustaste. Dann können Sie verschieben. Diesmal wandert aber nicht nur ein Rahmen, sondern ein ganzes Rudel dieser Markierungen mit dem Mauszeiger über den Bildschirm. Das Ziel der Wanderung wird einzig und allein durch den Mauspfel bestimmt. Stören Sie sich also nicht daran, wenn einige der Icons am Bildrand hängen bleiben, bevor Sie Ihr eigentliches Ziel erreicht haben.

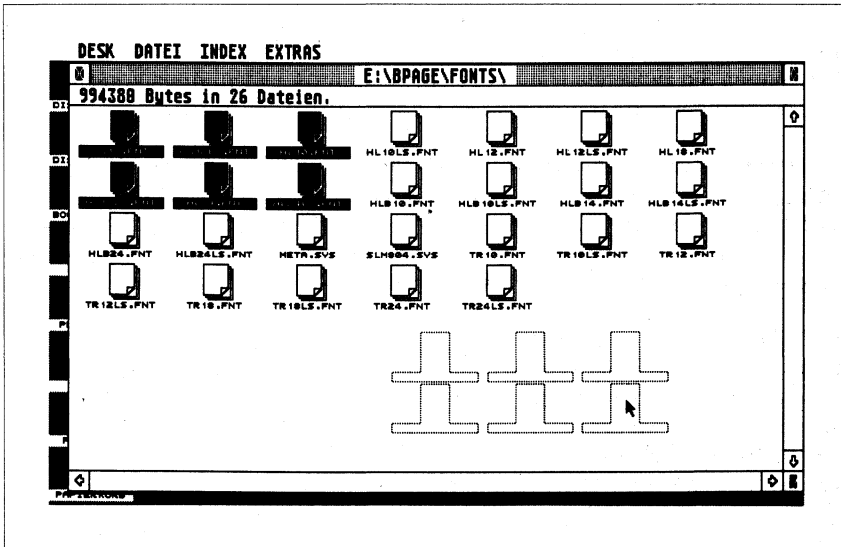


Abb. 2.4: Verschieben einer Gruppe von Dateien

Übrigens: Der Papierkorb kann "nur" verschoben werden, um ihn an einen bequemerem Platz zu bringen. Es ist nicht möglich, ihn auf ein Fenster oder ein anderes Icon zu legen.

Verschieben der Optik wegen kann man auch alle Datenstation-Icons, aber es gibt hier noch einen weiteren Zweck. Legt man zwei dieser Laufwerk-Icons aufeinander, so gibt man dem ST damit den Befehl, den Inhalt der oberen Datenstation auf die untere zu kopieren. Legt man ein Laufwerk-Icon in ein Fenster, so bedeutet dies, daß der ST alle Dateien von der oberen Datenstation auf die untere kopiert. Das hört sich ziemlich ähnlich an, es gibt aber feine und wichtige Unterschiede. Im ersten Fall wird die Datei 1:1 kopiert, d.h. Sie haben hinterher ein identisches Duplikat mit genau der gleichen Anordnung der

Dateien. Das klappt deshalb auch nur mit Disketten, die das gleiche Format besitzen (mehr zu diesem Thema im Kapitel "Massenspeicher"). Im zweiten Fall nimmt der ST jede Datei der ersten Station und überträgt sie auf die zweite. Evtl. noch vorhandene Dateien auf dem Zielgerät werden dadurch nicht berührt, während sie im ersten Fall zerstört würden (deshalb erscheint auch beim Ausführen dieser Funktion eine entsprechende Warnung).



Wenn Sie unter TOS 1.04 die Funktion Disk-Kopie (1:1) aktivieren, erscheint eine erweiterte Dialogbox, die auch das Formatieren von Disketten zuläßt. Dies dient der Bequemlichkeit, denn viele Anwender mußten beim Kopieren von Disketten feststellen, daß die Zieldiskette nicht formatiert war und mußten deshalb die Kopierfunktion abbrechen. Mit dem neuen TOS 1.04 können Sie zwischendrin auf Formatieren umschalten. Aber Vorsicht! Mit versehentlichem Formatieren können Sie viele Daten zerstören.

Wenn Sie Dateien löschen möchten, müssen Sie diese nur auf den Papierkorb legen. (Das Wort kopieren wäre wohl unpassend.). So einfach ist das. Damit es aber nicht zu einfach ist, gibt es noch eine Sicherheitsabfrage, d.h. nach dem Verschieben in den Papierkorb fragt der ST ganz artig noch mal nach: "Soll ich wirklich?". Nur wenn Sie auf OK klicken, traut der kleine Bursche sich auch.

In ähnlicher Weise kann man einzelne Dateien kopieren. Dazu muß man nur die zu kopierende Datei bzw. Gruppe von Dateien auf das Fenster legen, das Sie als Ziel des Kopiervorgangs auserkoren haben. Sie können die Dateien auch auf einen Ordner legen, dann werden sie in eben diesen Ordner hineinkopiert. Weil das alles etwas kompliziert ist, gibt die untenstehende Tabelle an, welches Verschieben welche Reaktionen auslöst.

Ziel ist...	Verschoben wird ...	Dateien bzw. Gruppe
Datenstation	1:1 Duplikat, falls iden. Format	Kopieren einzelner Dateien
Papierkorb	Nicht möglich	Dateien werden gelöscht!
Ordner	Dateien werden in den Ordner kopiert	Dateien werden in den Ordner kopiert
einzelne Datei	Nicht möglich	Nicht möglich
Desktop	Standortänderung	Nicht möglich

Ein Beispiel: Sie möchten eine Datei TEXT.TXT von einer RAM-Disk auf die Diskette in Laufwerk A kopieren. Dazu müssen Sie so vorgehen: Öffnen Sie zunächst die RAM-Disk, um ein Inhaltsverzeichnis zu erhalten. Dann klicken Sie die gewünschte Datei einmal an (sie ist jetzt invertiert). Um zu kopieren,

verschieben Sie das Icon auf das des Laufwerks A. Nach einer kurzen Zeit erscheint dann eine Box auf dem Bildschirm, die Ihnen mitteilt, daß 0 Ordner und 1 Datei kopiert werden sollen. Sie haben jetzt die Chance, diesen Vorgang noch abzubrechen, indem Sie auf "ABBRUCH" klicken. Klicken Sie "OK" an, so beginnt der ST mit der Übertragung. In der Box können Sie dabei immer sehen, wieviele Dateien noch zu übertragen sind.

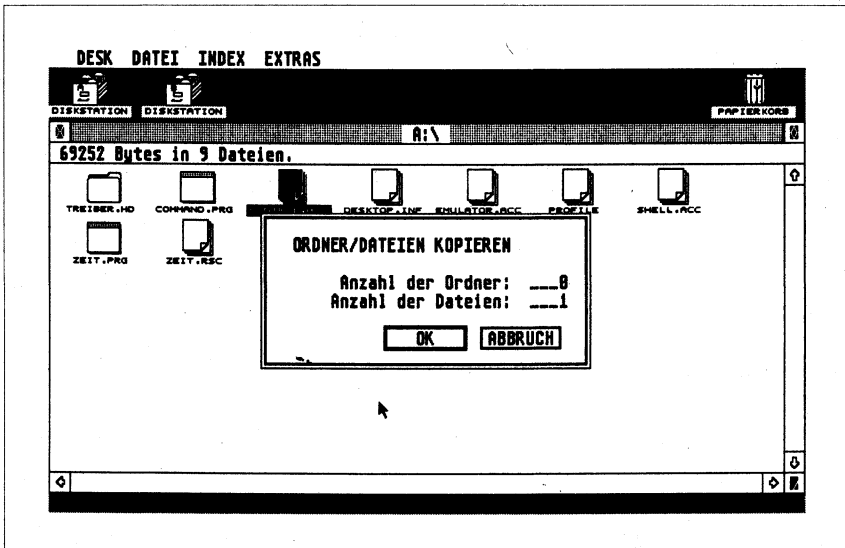


Abb. 2.5: Die Kopierbestätigung

Wenn Sie nach Beendigung des Kopiervorganges das Inhaltsverzeichnis von Laufwerk A ansehen, werden Sie dort die Datei TEXT.TXT wiederfinden.

TOS 1.04 bietet wieder einmal mehr Komfort. Einen so ausgelösten Kopiervorgang können Sie jederzeit durch einen Druck auf UNDO unterbrechen. Es erscheint dann eine kleine Dialogbox, in der Sie durch Mausclick den Abbruch bestätigen oder den Kopiervorgang fortfahren lassen können. Außerdem zeigt TOS 1.04 während des Dateitransfers immer an, welche Datei gerade bearbeitet wird.

Beachten Sie bitte, daß die alten TOS-Versionen (vor 1.04) beim Kopieren der neuen Datei das momentan eingestellte Datum geben. Die neuen Versionen behalten das Datum der Originaldatei bei.

Spinnen wir unser obiges Beispiel noch ein wenig weiter. Nachdem die Datei glücklich auf unserer Diskette gelandet ist, können wir sie aus der RAM-Disk entfernen, sprich löschen. Dazu brauchen Sie nur das Icon aus dem Fenster der RAM-Disk auf den Papierkorb zu schieben. Wieder meldet sich der ST mit einer kleinen Box, in der er die Anzahl der zu löschenden Dateien und Ordner angibt. Wie zuvor könnten Sie auch hier wieder die Notbremse ziehen und "ABBRUCH" anwählen.

Nehmen wir weiter an, auf unserer Diskette in Laufwerk A existiert ein Ordner mit Namen "TEXTE", in den wir eine Datei aus dem Hauptverzeichnis hineinkopieren wollen. Wie kriegen wir sie dahin? Recht einfach. Zunächst müssen wir die Datei in den Ordner hineinkopieren. Dazu schieben Sie das entsprechende Icon der Datei auf den gewünschten Ordner. Wie Sie sehen, funktioniert das Kopieren nicht nur von Diskette zu Diskette, sondern auch von einem Verzeichnis zu einem anderen auf der gleichen Scheibe. Ist dieser Kopiervorgang abgeschlossen, besitzen wir zwei Versionen unserer Datei TEXT.TXT. Eine steht im Hauptverzeichnis, die andere im Ordner. Falls Sie die erste nicht mehr brauchen, können Sie sie einfach löschen. Fertig!

Auch beim Kopieren, Löschen und Verschieben gibt es Pannen - aber nicht Verzagen, hier kommt Abhilfe:



Beim Erstellen einer 1:1-Kopie erscheint eine Meldung, daß dies nicht möglich sei, weil die Disketten unterschiedliche Formate besäßen.



Eine 1:1-Kopie können Sie in dieser Situation auf keinen Fall erstellen. Sie können sich aber damit behelfen, daß Sie alle Dateien einer Diskette auf die andere kopieren. Dieser Fehler tritt zum Beispiel auch dann auf, wenn Sie 1:1-Duplikate zwischen einer einseitig formatierten und einer zweiseitigen Diskette machen möchten.



Es erscheint eine Meldung "Diese Diskette hat zuwenig Speicherplatz!".



Dieser Fehler tritt auf, wenn die Zieldiskette voll ist. Kopieren Sie die noch nicht übertragenen Dateien auf eine andere Diskette, oder schaffen Sie Platz, indem Sie nicht mehr benötigte Dateien löschen.



Es erscheint eine Meldung "Daten auf Diskette... defekt".



In diesem Fall scheint die Diskette selbst defekt zu sein. Schlagen Sie bitte im Kapitel "Massenspeicher" nach.

➡ Es erscheint eine Meldung "Objekte aus Fenstern nicht auf dem Desktop ablegen", nachdem Sie versucht haben, Dateien auf ein anderes Laufwerk oder den Papierkorb zu schieben.

✓ Sie haben etwas daneben gezielt! Beim Loslassen des Mausknopfes war der Mauszeiger nicht auf dem gewünschten Ziel, sondern leicht daneben positioniert. Versuchen Sie es noch einmal. Sie können das Treffen eines Ziels daran erkennen, daß das betreffende Icon invertiert wird!

➡ Beim Kopieren erscheint eine Meldung, daß eine Datei gleichen Namens schon auf der Zielstation existiere.

✓ Wenn der ST zwei Dateien gleichen Namens auf der gleichen Diskette anlegen soll, kann dies zu Verwechslungen führen. Deshalb weigert sich der ST in solchen Fällen, den Kopiervorgang auszuführen. Sie haben hier die Möglichkeit, einen neuen Namen für die Kopie anzugeben; wird der Name nicht geändert, wird die bereits vorhandene Datei überschrieben. Durch Klicken auf "Abbruch" können Sie den Kopiervorgang abbrechen. Unter TOS 1.04 haben Sie außerdem die Möglichkeit, die störende Datei zu überspringen, d.h. es werden evtl. noch zu kopierende andere Dateien weiter bearbeitet.

➡ Beim Löschen zeigt der ST eine Meldung an, daß die Datei den Status "nur lesen" habe.

✓ Sie versuchen, eine Datei zu löschen, die ausdrücklich gegen solche Manipulationen geschützt ist. Wollen Sie trotzdem löschen, müssen Sie das Merkmal in der Infoanzeige zu dieser Datei in "lesen/schreiben" ändern.

➡ Beim Kopieren erscheint eine Meldung, daß ein Laufwerk mit dieser Kennung unbekannt sei.

✓ Sie haben eine Floppy falsch angemeldet oder angemeldet, obwohl dieses Gerät gar nicht existiert. Der Kopiervorgang kann daher nicht durchgeführt werden.

Man sollte meinen, nach soviel Schieben und Bewegen sind alle Möglichkeiten erschöpft. Mitnichten. Wie haben fast das Wichtigste vergessen, das "Dateien verschieben". Verwechseln Sie das bitte nicht mit dem Verschieben von Icons. Dateien zu verschieben bedeutet, sie von Ihrem alten Platz zu entfernen und an einen neuen zu bringen (z.B. in einen anderen Ordner). Diese Funktion kennt nur TOS 1.04, beim alten TOS müssen Sie sich mit einer Notlösung begnügen. Beim neuen TOS können Sie eine Verschiebung auslösen, indem Sie

die zu den Dateien gehörenden Icons auf das gewünschte Ziel schieben (wie beim Kopieren), gleichzeitig aber die **Ctrl**-Taste gedrückt halten. Dann werden die Dateien nacheinander in das Zielverzeichnis verschoben. Ein solcher Vorgang ist sehr viel schneller als ein Kopieren.

Die Notlösung für Besitzer des alten TOS besteht darin, die Dateien zunächst an den gewünschten Ort zu kopieren und dann die alten Versionen zu löschen.

2.3 Fenster

Vieles von dem, was Sie mit dem ST anstellen können, spielt sich in Fenstern ab. Da ist es kein Wunder, daß hier besonders viele Möglichkeiten versteckt sind. Werfen wir erst einmal einen Blick auf ein typisches Fenster.

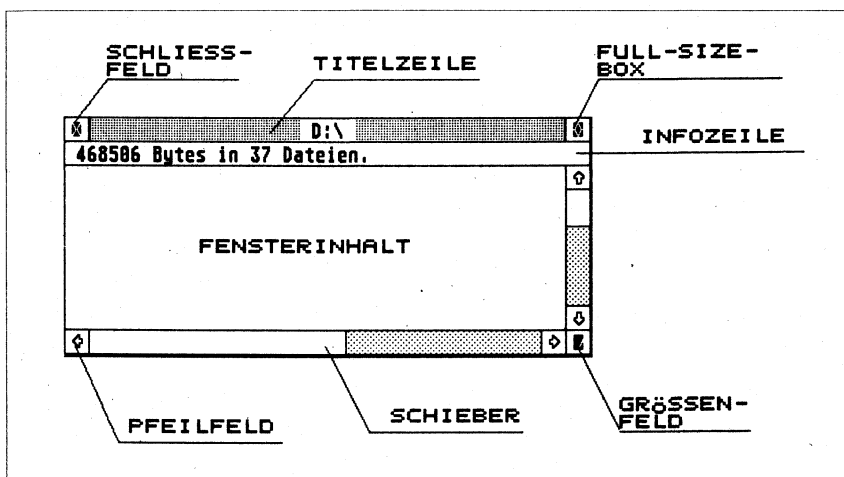


Abb. 2.6: Ein Fenster mit allen Schikanen

Fenster sind das typische Anzeigemedium Ihres ST. Wann immer Daten auf besonders komfortable Art angezeigt werden sollen, treten Fenster auf den Plan.

Das kleine Quadrat mit der durchgestrichenen Scheibe in der linken oberen Ecke ist das sogenannte Schließfeld. Klickt man hier mit dem Mauszeiger, so wird auf dem Desktop das Fenster geschlossen, falls darin das Hauptinhaltsverzeichnis einer Datenstation angezeigt wurde.

Stand im Fenster das Inhaltsverzeichnis eines Ordners, so gelangen Sie durch den Klick auf das Schließfeld wieder in den vorherigen Pfad zurück. In Programmen kann dies auch andere Bedeutungen haben - z.B. Beenden des Programms.

Rechts neben dem Schließfeld befindet sich die sogenannte Titelzeile. Sie heißt so, weil eben hier (wenn vorhanden) der Name des Fensters angezeigt wird. Desktopfenster geben hier an, welches Inhaltsverzeichnis innerhalb des Fensters dargestellt wird. Die Titelzeile hat aber noch eine andere Aufgabe. Mit ihrer Hilfe läßt sich das ganze Fenster bewegen. Das funktioniert genauso wie bei Icons:

1. Fenster durch einfachen Mausklick aktivieren. Aktive Fenster (davon gibt es immer nur eines) erkennt man daran, daß die Titelzeile grau gefärbt ist. Außerdem "liegen" sie immer ganz oben. Bei nicht aktiven Fenstern ist die Titelzeile weiß. Wenn Sie ein Fenster bewegen möchten, müssen Sie es unbedingt aktivieren, da der Trick mit der rechten Maustaste nur auf Icons wirkt!
2. Mauszeiger in die Titelzeile bewegen und linke Maustaste gedrückt halten. Das Fenster (bzw. ein Rahmen, der dieses symbolisieren soll) wird Ihren Mausbewegungen nun so lange folgen, bis Sie die Taste wieder loslassen. Direkt nach dem Loslassen wird dann das Fenster an der neuen Position gezeichnet. Sie können das Fenster überallhin schieben, auch über andere Fenster oder Icons. Wenn gerade Platzmangel auf dem Bildschirm herrscht, können Sie es auch nach rechts oder unten herauschieben. Es braucht nur eine winzig kleine Ecke der Titelzeile sichtbar zu bleiben, damit Sie das Fenster später wieder zurückholen können.

Rechts neben der Titelzeile finden Sie die "Full-Size"-Box, wie sie im Englischen genannt wird. Der Name deutet es an: Ein Mausklick auf dieses Quadrat bewirkt, daß das Fenster die Größe des gesamten Bildschirms annimmt. Ein weiterer Klick macht dies wieder rückgängig. Allerdings ist dabei eines zu beachten. Wenn Sie nach dem Vergrößern auf Bildschirmgröße das Fenster verkleinern, vergißt das Desktop die alten Ausmaße. Klicken Sie dann noch einmal auf die Full-Size-Box, so nimmt das Fenster wieder den gesamten Bildschirm ein. Das gleiche gilt, wenn Sie zwischenzeitlich ein GEM-Programm gestartet hatten.

Unter der Titelzeile finden Sie die sogenannte Infozeile. Sie ist dafür gedacht, zusätzliche Informationen über den Fensterinhalt zu geben, z.B. die Anzahl belegter Bytes in einem Ordner. In manchen Programmen wird sie nicht eingesetzt; sie besitzt auch keine weiteren Funktionen.

Unter der Infozeile folgt nun endlich der Fensterinhalt. Je nach Programm kann der sehr verschieden ausfallen. Auf dem Desktop werden hier die Icons der Dateien dargestellt, während Ihre Textverarbeitung hier den eingegebenen Text hinschreibt und ein Malprogramm vielleicht eine Zeichenfläche zur Verfügung stellt.



Im Falle des Desktop gilt noch eine kleine Besonderheit. Wenn Sie auf die **[Esc]**-Taste (links oben auf der Tastatur) drücken, versucht der ST, das Inhaltsverzeichnis neu von der Diskette zu lesen. Das hat folgenden Sinn: Nach dem Wechsel einer Diskette in einem Laufwerk steht im Fenster noch das Inhaltsverzeichnis der zuvor verwendeten Diskette, obwohl diese gar nicht mehr im Laufwerk liegt. Drücken Sie in dieser Situation **[Esc]**, so zwingen Sie den ST damit, das neue Inhaltsverzeichnis anzuzeigen.

Der Fensterinhalt wird rechts und unten durch die sogenannten Schieberfelder begrenzt. Deren Funktionen sind recht vielfältig, aber für beide Schieberfelder gleich. Kurz gefaßt kann man sagen, daß diese Felder dazu dienen, den Fensterinhalt in alle vier Himmelsrichtungen zu bewegen, falls das Fenster nicht groß genug sein sollte, um den gesamten Inhalt anzuzeigen. Damit wären wir schon bei der ersten Aufgabe der Schieber. Ihre Größe korrespondiert immer mit der Größe des Fensterinhalts. Wenn also einer der weißen Schieber nur die Hälfte des Schieberfeldes einnimmt, dann heißt das, daß noch einmal so viele Informationen, wie zur Zeit angezeigt sind, unseren Blicken verborgen bleiben. Wenn Sie jetzt den Schieber bewegen (wie ein Icon oder Fenster), dann wandert der Fensterinhalt aus dem Bild heraus, und der Rest wird sichtbar. Bildlich kann man sich das so vorstellen, daß Sie einen Rahmen über einem Blatt verschieben und dieses Blatt nur dadurch sichtbar wird. Um alle Informationen zu Gesicht zu bekommen, müssen Sie den Rahmen bewegen. Für die Bewegungen haben Sie mehrere Möglichkeiten. Durch einen Mausklick in eines der Pfeilfelder verschiebt sich der Rahmen um eine Zeile oder Spalte in der gewünschten Richtung. Wie hoch bzw. breit diese Verschiebung ist, hängt wieder einmal von der speziellen Situation ab. Eine Icon-Zeile ist nun einmal höher als eine Textzeile.

Sie können auch, wie schon beschrieben, direkt eine Position anfahren, indem Sie den Schieber entsprechend bewegen. Klicken Sie dagegen neben den Schieber (also in die grau unterlegten Abschnitte des Schieberfeldes), so bewegt sich der gedachte Rahmen um eine ganze Fensterseite. Die Größe dieser Fensterseite ist auch hier wieder vom Fensterinhalt abhängig.

Sie haben sich sicher schon gefragt, wie Sie die Größe eines Fensters verändern können. Das machen Sie mit dem Größenfeld. Damit ist das kleine schwarze Rechteck mit dem weißen Schrägstrich in der unteren rechten Ecke gemeint. Es

dient dazu, die Größe des Fensters festzulegen. Das Vorgehen dabei ist wieder einmal dem Verschieben von Icons sehr ähnlich. Also linke Maustaste in diesem Feld gedrückt halten, und Mauspfel in Richtung der gewünschten Größe bewegen. Der vom Fensterschieben her bereits bekannte Rahmen erscheint wieder. Diesmal wächst oder schrumpft er jedoch, je nach Mausbewegung. Sobald Sie die Maustaste loslassen, wird das Fenster in der neuen Größe gezeichnet. Gleichzeitig werden Sie bemerken, daß die Schieber ihre Größe geändert haben. Falls Sie also einmal ein Fenster verkleinert haben, werden Sie durch die grau unterlegten Schieberabschnitte daran erinnert, daß nicht alle Informationen sichtbar sind.



Wenn Sie sich in der Bedienung von Fenstern noch nicht zu Hause fühlen, können Sie das Programm SAMPLE.PRG starten, daß Ihnen auf der von Atari mitgelieferten Diskette zur Verfügung steht. Der einzige Sinn dieses Programms ist es, ein Fenster auf dem Bildschirm zu erzeugen, mit dem Sie nach Herzenslust herumspielen und trainieren können, bis Sie auf das Schließfeld klicken. Dann nämlich endet das Programm, und Sie gelangen wieder zurück zum Desktop.

Wer die obigen Beschreibungen liest, könnte schnell auf den Gedanken kommen, Fenster seien die absoluten Einzelgänger. Dem ist nicht so. Sie können im Desktop bis zu vier dieser nützlichen Einrichtungen auf den Bildschirm zaubern. Probieren Sie es. Führen Sie viermal hintereinander einen Doppelklick auf das Icon der Diskettenstation A aus, und es werden vier Fenster mit demselben Inhalt erscheinen. Wie Sie sehen, ist immer nur eines aktiv, und dieses aktive Fenster liegt (optisch betrachtet) stets ganz oben auf dem Stapel. Dahinter steht die Vorstellung, daß ein Fenster ein Blatt Papier auf einem Schreibtisch repräsentiert. Legen Sie mehrere Blätter auf Ihren Schreibtisch, können Sie auch nur diejenigen bearbeiten und sehen, die ganz oben liegen.

Um ein verdecktes Fenster nach oben zu holen (aktivieren), müssen Sie wenigstens eine kleine Ecke davon sehen können. Klicken Sie dort hinein, und das Desktop holt Ihnen das Fenster nach vorne. Dabei ist es egal, wo Sie das Fenster anklicken. Bei nicht aktiven Fenstern können Sie überall hin klicken, ohne etwas anderes als die Aktivierung zu bewirken.

Versuchen Sie, mehr als vier Fenster zu öffnen, so erscheint eine Warnmeldung auf dem Bildschirm. Sie müssen dann zunächst eines der anderen Fenster schließen, bevor Sie ein neues öffnen können.

Und wozu ist die Arbeit mit mehreren Fenstern gut? Nehmen wir an, Sie möchten Dateien aus einem Ordner in einen anderen Ordner der gleichen Diskette kopieren. Dann brauchen Sie zwei Fenster. Zunächst holen Sie sich ein

Inhaltsverzeichnis des ersten Ordners, dann klicken Sie noch einmal auf das Icon der gleichen Diskette, und Sie erhalten ein Fenster mit dem Hauptverzeichnis. Darin suchen Sie nun den Zielordner. Anschließend können Sie die zu kopierenden Icons aus dem Ordnerfenster in das andere schieben (entweder mit dem Rechte-Maus-Tasten-Trick aus Kapitel 2.2 oder nach Aktivieren des ersten Fensters). Das funktioniert natürlich auch zwischen verschiedenen Datenstationen.

Pannenhilfe:



Sie wollen ein neues Fenster öffnen. Dabei gibt der ST die Fehlermeldung aus, daß nur vier Fenster möglich seien. Sie zählen auf dem Bildschirm aber weniger als vier Fenster, müßten also theoretisch noch ein weiteres öffnen können.



Sie haben irgendwann vorher ein Fenster aus dem Bild herausgeschoben und dort vergessen. Von diesem Fenster müßte noch eine winzig kleine Ecke sichtbar sein, auch wenn diese Ecke vielleicht nur noch die Größe eines Punktes hat. Versuchen Sie diese am unteren Bildrand zu finden. Haben die das geschafft, können Sie dieses vierte Fenster schließen und wie gewünscht ein anderes öffnen.



Sie wissen, daß unter einem Fenster ein anderes vollständig verborgen ist. Sie möchten aber gerade dieses verborgene Fenster aktivieren.



Verschieben oder verkleinern Sie das obere Fenster so weit, daß das darunter liegende wieder sichtbar wird. Dann reicht ein Klick, um dieses zu aktivieren.

Oben habe ich bereits erwähnt, daß in Programmen zwar auch Fenster mit dem gleichen Aussehen wie auf dem Desktop vorkommen können, diese aber nicht unbedingt das gleiche Verhalten zeigen müssen. Das liegt an einer speziellen Eigenart des ST. Er bietet allen Programmen an, die Fenster zu zeichnen und zu überwachen. Entdeckt er dabei zum Beispiel einen Klick auf das Schließfeld, so faßt er dies nicht als Befehl zum Schließen des Fensters auf, sondern meldet dieses Ereignis lediglich an das Programm weiter. Das Programm entscheidet dann, was zu tun ist. Wenn der Programmierer für einen solchen Fall vorgesehen hat, etwas anderes zu tun, dann kann es durchaus vorkommen, daß z.B. durch das Schließfeld ein weiteres Fenster geöffnet wird. Falls Sie also mit irgendwelchen Fenstern arbeiten, deren Reaktionen Ihnen spanisch vorkommen, schlagen Sie in der jeweiligen Programmanleitung nach.

Die Beschränkung auf vier Fenster hat ebenfalls technische Ursachen. Der ST muß nämlich für die Verwaltung eines jeden Fensters eine gewisse Menge an Speicher bereit halten. Ließe man unendlich viele Fenster zu, wäre der Speicher selbst eines ST sehr schnell erschöpft.

2.4 Drop-Down-Menüs

In der obersten Zeile des Bildschirms finden Sie die sogenannte Menüleiste. Berühren Sie eine der Überschriften mit dem Mauszeiger, klappt ein Menü herunter. (Daher der englische Name, der soviel wie "Herunterfallende Auswahlmöglichkeit" bedeutet.). Haben Sie ein solches Herunterklappen einmal versehentlich ausgelöst, klicken Sie einfach irgendwo neben das Menü. Es verschwindet sofort, das Klicken hat keinen Einfluß auf den Rest des Programms. Innerhalb des Menüs können Sie unter verschiedenen Funktionen wählen, die je nach Programm verschieden sind. (Die Funktionen des Desktop werden weiter unten erläutert.). Dazu brauchen Sie nur den Mauszeiger auf die gewünschte Zeile zu bewegen (sie wird dabei invertiert) und mit der linken Maustaste zu klicken. Was dann passiert, hängt von der jeweiligen Funktion ab.

Innerhalb der Menüs sind noch einige Besonderheiten erwähnenswert. Menüs, die wie Schalter wirken (z.B. Ton an/aus) werden oft mit Haken dargestellt. Sobald die betreffende Funktion eingeschaltet ist, erscheint der Haken im Menü, beim Abschalten verschwindet er wieder.

Menüzeilen, die in heller Schrift dargestellt sind, können nicht angewählt werden. Viele Programme benutzen dies, um Funktionen abzuschalten, die momentan unsinnig wären (z.B. Speichern eines nicht vorhandenen Textes).



In vielen Menüs findet man außerdem noch am rechten Rand einzelne Buchstaben. Damit wird auf evtl. vorhandene Kurzwahlen verwiesen. Wenn Sie mit einem Programm sehr vertraut sind, kann es oft schneller sein, eine Funktion über eine bestimmte Tastenkombination zu aktivieren, als über das Menü. Die dafür nötigen Tastendrucke werden dann am rechten Rand vermerkt.

2.5 Das Menü des Desktop

Natürlich verfügt auch die Startbasis unserer Arbeit mit dem ST über ein solches Menü. Dessen Funktionen sind nicht immer leicht zu durchschauen, weshalb sie hier noch einmal der Reihe nach beschrieben werden sollen.

2.5.1 Desk

Egal in welchem Programm Sie sich befinden, das am weitesten links stehende Menü hat (unabhängig vom Titel) in jedem Fall die gleiche Aufgabe. Zuerst finden Sie eine Copyright-Meldung (meist Desktop-Info oder INFO genannt), darunter folgen die verschiedenen Accessories.



Mit der Funktion "Desktop Info" können Sie feststellen, welche Betriebssystemversion Sie besitzen. "TOS 1.04" ist z.B. die neue Version vom Frühjahr 1989. Ganz allgemein gilt: Wenn die Jahreszahl 1989 in dieser Copyrightmeldung auftaucht, arbeiten Sie mit einer der neuesten TOS-Versionen. Beachten Sie dann bitte alle Hinweise zum TOS 1.04.

Accessories sind Programme, die Sie auch dann noch zur Verfügung haben, wenn Ihr Rechner noch mit anderen Aufgaben beschäftigt ist. Wenn Sie also während der Arbeit mit dem Textverarbeitungssystem feststellen, daß Sie vergessen haben, den Drucker auf Einzelblattbetrieb anzupassen, dann brauchen Sie nicht das Textsystem zu verlassen, um diese Änderung vom Desktop aus vorzunehmen. Statt dessen können Sie im Textsystem das Accessory Drucker-Anpassung im links stehenden Menü aufrufen. Mit dem ST werden zwei dieser Accessory-Programme ausgeliefert; diese erzeugen (ganz untypisch) jeweils zwei Menüpunkte. CONTROL.ACC ist für die Zeilen "Kontrollfeld" und "Drucker Anpassung" zuständig. Das Accessory EMULATOR.ACC erzeugt die Menüpunkte "VT52 Emulator" und "RS 232 Einstellung". Diese werden im Kapitel über die Schnittstellen genauer erläutert, ebenso finden Sie die Erklärung der "Drucker Anpassung" im Kapitel "Drucker".



Sollten Sie Besitzer eines MEGA-ST sein, so müssen Sie hier ein wenig aufpassen (das gleiche gilt auch für normale STs mit eingebauter Echtzeituhr). Durch einen Programmierfehler verstellen die alten Versionen des CONTROL.ACC bei jedem Start des ST die so sorgsam gestellte Uhr. Dies passiert bei neueren Versionen nicht mehr. Sollten Sie Zweifel in dieser Hinsicht haben, fragen Sie bitte Ihren Fachhändler.

Eine vielgestellte Frage ist "Was kann ich mit den Accessories machen?". Das hängt im wesentlichen davon ab, welche Accessories Sie besitzen. Sie sind nicht allein auf die von Atari mitgelieferten Accessories angewiesen. Sie können weitere dazukaufen und auf Ihre Systemdiskette kopieren. So gibt es beispielsweise schon Accessories, die Notizblöcke oder gar komplette Adreßverwaltungen zur Verfügung stellen.

Hier soll uns nur das Kontrollfeld interessieren, weil wir hier die Einstellungen machen können, die wir für die Arbeit mit dem Desktop brauchen. Klicken Sie das Wort "Kontrollfeld" an. Dann erscheint ungefähr folgendes Bild.

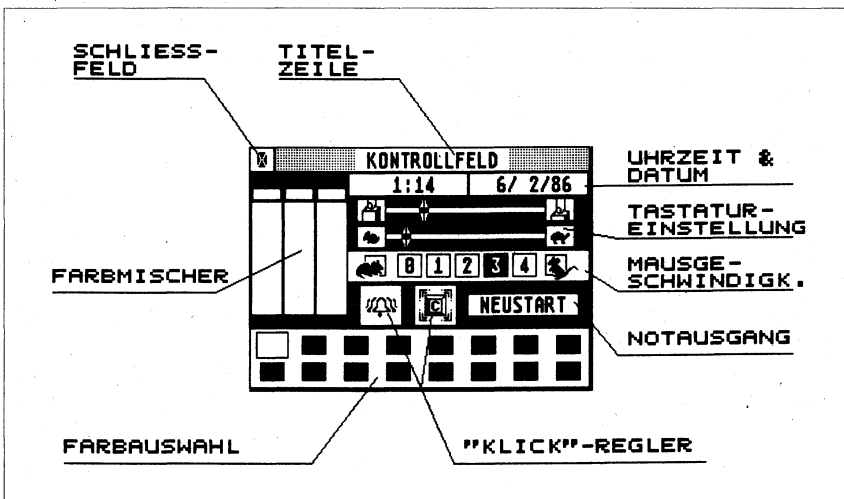


Abb. 2.7: Das Kontrollfeld

Wie Sie sicher schon bemerkt haben, wird das Kontrollfeld in einem Fenster dargestellt. Sie können es daher auch verschieben und schließen (was hier das Beenden der Einstellung zur Folge hat). Während das Kontrollfeld auf dem Bildschirm steht, können Sie nicht mit den anderen Fenstern oder dem Programm, von dem aus Sie das Accessory aufrufen, arbeiten. (Hinweis: Die Rechner der STE-Reihe werden mit einem verbesserten Kontrollfeld ausgeliefert, das zur Drucklegung dieses Buches noch nicht zur Verfügung stand. Es können daher geringe Differenzen auftreten.)

Direkt unter der Titelzeile werden das Datum und die Uhrzeit angezeigt, auf die die eingebaute Uhr gerade eingestellt ist. Im Normalfall (außer bei MegaSTs) dürften diese Werte nicht stimmen. Um sie zu ändern, klicken Sie bitte

einmal in das entsprechende Feld. Es erscheint ein kleiner senkrechter Strich, der Cursor. Sie können jetzt das betreffende Feld editieren. Mit den Pfeiltasten können Sie den Cursor nach links oder rechts bewegen, **[Esc]** löscht das Eingabefeld, **[Delete]** und **[Backspace]** löschen jeweils ein Zeichen. Geben Sie nun die Zeit bzw. das Datum neu ein (ggf. durch Nullen aufgefüllt wie in 0931 für 9 Uhr 31 oder 190791 für den 19.7.1991), oder ändern Sie das bestehende durch Überschreiben. Wenn Sie fertig sind, drücken Sie bitte **[Return]** oder **[Enter]**, und der ST übernimmt den neuen Wert.



Wenn Sie gerade die Uhrzeit eingeben, empfinden Sie es sicher als sehr lästig, für den Wechsel zum Datum zuerst **[Return]** zu drücken und dann das Datumsfeld anzuklicken. Es geht auch einfacher. Wenn Sie im Uhrzeitfeld die **[Tab]**-Taste drücken, wandert der Cursor in das Datumsfeld, und Sie können dort den gewünschten Wert eingeben, auch wenn das Feld nicht wie üblich invertiert wird. Nach dem Druck auf **[Return]** werden beide Werte übernommen.

Unter den Feldern für Zeit und Datum befinden sich zwei Gebilde, die wie die Schieberegler eines Mischpults aussehen. Mit diesen beiden Schiebern können Sie die Tastatur beeinflussen. Bedient werden beide wie ein Fensterschieber: anklicken, Maustaste festhalten und dann in die gewünschte Richtung schieben. Der untere Schieber regelt, wieviele Zeichen in der Sekunde von der Repeat-Funktion ausgegeben werden. Der Hase steht dabei für eine schnellere, die Schildkröte für eine langsamere Wiederholung. Falls Sie die Repeat-Funktion nicht kennen: Drücken Sie in einer Textverarbeitung einmal eine Buchstabetaste, und halten Sie diese fest. Nach einer kurzen Wartezeit wird dieser Tastendruck vom Rechner automatisch solange wiederholt, wie Sie die Taste festhalten.

Der obere der beiden Regler dient dazu, die Verzögerung einzustellen, mit der die Wiederholung einsetzt. Ein Verschieben nach links bedeutet wiederum eine Beschleunigung, die andere Richtung eine Verlangsamung.

In der nächsten Zeile können Sie dann die Geschwindigkeit festlegen, mit der ein Doppelklick ausgeführt werden muß, um als solcher erkannt zu werden. Echte Maus-Tasten-Artisten können dies so schnell ausführen, daß Sie den Wert 0 (die schnellste Stufe) wählen. Für den Normalanwender dürfte 2 die richtige Geschwindigkeit sein. Falls Sie damit nicht zurecht kommen, können Sie mit den Werten 3 oder gar 4 die Erkennungszeit noch weiter verlängern.

Die Glocke steht für die akustischen Warnmeldungen, die der ST dann und wann ausgibt, falls der Anwender einen Fehler gemacht hat. In einem solchen Fall läßt er ein Piepen hören (vorausgesetzt, die Lautstärke Ihres Monitors ist

nicht auf 0 gestellt). Durch Anklicken der Glocke können Sie diese Warnsignale je nach persönlichem Geschmack ein- bzw. ausschalten. Das gleiche gilt für den sogenannten Tastaturklick, der durch den Knopf mit dem "C" symbolisiert wird. Ist er eingeschaltet, so quittiert der ST jeden Druck auf eine Taste mit einem kleinen Klickgeräusch. Für beide Geräusche (Atari nennt das "Audiophile Rückmeldung") gilt, daß sie ausgeschaltet sind, wenn das betreffende Symbol hell dargestellt wird.

Sollten Sie nach so vielen Reglern einmal den Kopf verloren haben und die letzte Einstellung zurückholen wollen, ist auch das kein Problem. Klicken Sie einfach auf Neustart, und der ST ist wieder so eingestellt, wie er es auch beim Aufruf des Kontrollfeldes war.

Auf der linken Seite finden Sie wiederum drei Schieberegler, die diesmal senkrecht angeordnet sind. Vorausgesetzt, Sie besitzen einen Farbmonitor, können Sie damit zu jeder Farbe einstellen, wie sich diese aus den drei Grundfarben Rot, Grün und Blau zusammensetzt. Dazu gehen Sie so vor: Wählen Sie bitte zuerst die Farbe aus, die Sie neu mischen möchten, indem Sie das betreffende Feld am unteren Rand des Kontrollfeldes anklicken. Dann stellen Sie mit den drei senkrechten Reglern die gewünschte Mischung ein. Jeder Farbanteil kann in 8 Stufen eingestellt werden. Beim STE, der die Farben feiner mischen kann, stehen sogar 16 Stufen zur Verfügung.



Besitzer eines Schwarzweiß-Monitors SM-124 oder SM-125 werden mit dieser Einstellungsmöglichkeit nicht viel anfangen können. Falls Sie aber einmal eine umgekehrte Farbdarstellung (weiß auf schwarz) benötigen, schieben Sie bitte die Regler für die Farbe Weiß alle nach ganz unten. Damit haben Sie schwarz und weiß vertauscht. Diesen Effekt können Sie wieder umkehren, indem Sie die Regler wieder alle nach ganz oben schieben.

2.5.2 Datei

Unter "Datei" finden Sie alles, was mit der Organisation und Nutzung von Disketten zu tun hat. Beachten Sie bitte, daß einige Menüpunkte nur dann anwählbar sind, wenn gewisse Voraussetzungen erfüllt sind, z.B. können Sie eine Diskette nur dann formatieren, wenn Sie vorher eine Datenstation aktiviert haben, mit der formatiert werden soll.

Der Menüpunkt "öffne" erfüllt den gleichen Zweck wie ein Doppelklick auf ein Icon. Bei Datenstationen wird ein Fenster aufgemacht, bei Ordnern wird deren

Inhaltsverzeichnis angezeigt, Dateien werden ausgedruckt bzw. angezeigt, und Programme werden gestartet.

"zeige Info..." ist wohl der wichtigste Punkt in diesem Menü. Zu einem vorher ausgewählten Objekt werden mit dieser Funktion wichtige Daten ermittelt. War das ausgewählte Objekt ein Datenstation-Icon, so werden die Stationskennung, der Diskettenname und verschiedene Werte wie freier und belegter Platz angezeigt. Näheres hierzu im Kapitel "Massenspeicherung". Wichtig ist: Die Angaben über Datenstationen, insbesondere der Name, kann nicht geändert werden! Ein Umbenennen der Diskette ist also nicht möglich.

Bezieht sich die Information auf einen Ordner, so sind die angezeigten Daten ähnlich. Auch hier gilt, daß keine Änderungen möglich sind, es sei denn, Sie arbeiten mit dem neuen TOS 1.04. Wie man trotzdem auch mit dem alten Betriebssystem einen Ordner umbenennen kann, zeigt Ihnen wiederum das Kapitel "Massenspeicherung".

Kann man die Informationen auch ändern? Was Disketten angeht: Nein! Etwas anders sieht es mit den Informationen über Dateien aus. Hier haben Sie zwei Möglichkeiten der Manipulation. Zum einen können Sie den Namen einer Datei ändern, zum anderen deren Merkmal. Ordner können - wie schon erwähnt - nur mit dem neuen TOS geändert werden.

Sie können mit dem Info eine Datei umbenennen. Das machen Sie mit dem Namen, der in der obersten Zeile der Info angezeigt ist. Mit dem Erscheinen der Information ist am Ende des Dateinamens auch der Cursor aufgetaucht. Wie Sie schon richtig erraten haben, können Sie mit diesem Cursor den Dateinamen beliebig verändern. Bewegen Sie den Strich einfach auf die gewohnte Weise mit den Cursortasten (**Esc** löscht die ganze Zeile), und schreiben Sie einen neuen Dateinamen hinein. Ein ganz einfaches Beispiel: Sie möchten ein Accessory (z.B. EMULATOR.ACC) zeitweilig abmelden, ohne es ganz von der Diskette zu löschen. Dazu brauchen Sie nur das Icon mit dem Namen "EMULATOR.ACC" einmal anzuklicken und sich die Info dazu anzeigen zu lassen. Mit der **Backspace**-Taste löschen Sie das letzte Zeichen des Namens (die Endung sollte jetzt .AC lauten). Das reicht schon. Durch einen Klick auf OK oder einen Druck auf die **Return**-Taste wird diese Änderung auf die Diskette übernommen. Falls Sie das nicht möchten, können Sie durch einen Mausklick auf ABRUCH den Notausgang wählen.

Die zweite Änderungsmöglichkeit besteht beim sogenannten Merkmal. Klicken Sie bitte eine der beiden Boxen an, die Ihren Wünschen entspricht. "lesen/schreiben" heißt, daß Sie mit der Datei beliebig umgehen dürfen; sowohl das Ansehen also auch das Ändern der Daten sind erlaubt (dies ist der Normalfall). Handelt es sich um wichtige Daten, dann möchten Sie vielleicht

vermeiden, daß diese versehentlich gelöscht oder geändert werden. In diesem Fall klicken Sie bitte "nur lesen" an. Der ST vermerkt dann auf der Diskette, daß diese Datei nicht geändert und nicht gelöscht werden darf.



Wenn Sie wichtige Dateien vor versehentlichem Löschen schützen möchten, können Sie einfach das Merkmal auf "nur lesen" setzen. Das bietet sich aber nur bei solchen Dateien an, die nicht mehr verändert werden müssen. Sehr gut geeignet sind die meisten Programme. Wenn Sie also Ihre teure Dateiverwaltung schützen möchten, verbieten Sie nach der Installation das Schreiben auf diese Datei.

Pannenhilfe:



Nach der Änderung eines Dateinamens erscheint ein Hinweis, daß dieser Dateiname schon existiere.



Sie haben versucht, den gleichen Namen zwei Dateien im gleichen Verzeichnis zu geben. Dies läßt der ST nicht zu, um Verwechslungen zu vermeiden. Wählen Sie bitte einen andern Namen für die Datei.

"neuer Ordner" erzeugt, wie schon der Name sagt, einen neuen Ordner in dem Verzeichnis, das gerade vom obersten Fenster angezeigt wird. Sie können also sowohl in einem Hauptverzeichnis als auch in einem Ordner neue Ordner erzeugen. Ein Beispiel: Im obersten Fenster wird das Inhaltsverzeichnis des Ordners PROGRAMME von Diskette A: angezeigt (die Fensterüberschrift lautet dann A:\PROGRAMME\). Innerhalb dieses Ordners soll nun ein neuer angelegt werden, z.B. mit dem Namen ALTE.PRG, um dort nicht mehr gebrauchte Programme abzulegen. Wählen Sie "neuer Ordner" an. Daraufhin erscheint eine kleine Dialogbox, in die Sie den Namen ALTE.PRG eintippen können. Nach dem Drücken von **Return** oder **Enter** wird der neue Ordner erzeugt und erscheint im Fenster. Soll der neue Ordner dagegen als Nachbar von PROGRAMME erscheinen, so müssen Sie erst wieder das ursprüngliche Inhaltsverzeichnis anzeigen lassen, in dem dieser steht, um dort "neuer Ordner" anzuwählen.

Pannenhilfe:



Beim Erzeugen eines Ordners erscheint die Meldung, daß ein Ordner gleichen Namens bereits existiere.



Sie können keine zwei gleichnamigen Ordner innerhalb des gleichen Verzeichnisses erzeugen. Wählen Sie bitte einen anderen Namen.

Der Menüpunkt "schließe" hat die gleiche Funktion wie das Schließfeld eines Fensters. Das gerade angezeigte Verzeichnis wird verlassen. Wurde das Hauptverzeichnis einer Diskette angezeigt, wird das Fenster ganz geschlossen.

"schließe Fenster" geht dagegen noch weiter als das Schließfeld. Egal, welches Verzeichnis gerade angezeigt war, diese Funktion läßt das Fenster auf jeden Fall vom Bildschirm verschwinden.

Durch "formatiere..." wird die Diskette im aktivierten Laufwerk formatiert. Warum Disketten formatiert werden müssen und was das eigentlich ist, erfahren Sie aus dem Kapitel über Massenspeicher. Wichtig ist: Formatieren löscht alle Daten auf der Diskette, weshalb auch eine entsprechende Warnmeldung angezeigt wird. Nur wenn Sie wirklich sicher sind, daß Sie die Diskette im angegebenen Laufwerk formatieren wollen, klicken Sie bitte OK, sonst bringt ABBRUCH Sie wieder zurück zum Desktop. Falls Sie also formatieren möchten, erscheint eine weitere Box auf dem Fenster, in die Sie den Namen der Diskette eingeben können. Außerdem müssen Sie wählen, ob Sie die Diskette einseitig oder zweiseitig formatieren möchten. (Wenn Sie ein Diskettenlaufwerk vom Typ SF354 besitzen, müssen Sie einseitig wählen.). Mit einem Klick auf FORMAT starten Sie das eigentliche Formatieren (während dessen der ST mit einem Balken das Fortschreiten seiner Arbeit signalisiert), durch AUSGANG kommen Sie zurück zum Desktop. Das Formatieren dauert einige Zeit, werden Sie bitte nicht ungeduldig. Nach dessen Ende erhalten Sie noch eine Mitteilung über die Speicherkapazität, die für einseitige Disketten 357376 und für zweiseitige 726016 Bytes betragen sollte. Wird weniger angezeigt, so bedeutet dies, daß der Rechner fehlerhafte Stellen (z.B. Kratzer) gefunden hat und diese automatisch von der weiteren Verwendung ausschließt. Sie können eine solche Diskette zwar weiterverwenden, sollte sie jedoch mit Vorsicht genießen, da durchaus noch mehr Fehler vorhanden sein könnten. Nach der Kapazitätsanzeige haben Sie die Wahl, eine weitere Diskette zu formatieren oder per AUSGANG zum Desktop zurückzukehren.



Im neuen TOS 1.04 ist die Formatierfunktion leicht geändert worden. Sie liegt zusammen mit der Diskettenkopie-Funktion in einer Dialogbox, so daß jederzeit der Wechsel zwischen diesen beiden Möglichkeiten zulässig ist. Je nachdem, ob der Knopf "Formatiere" oder "Kopiere" invertiert ist, wird die eine oder andere Funktion ausgeführt. Die Bedienung ist aber weitgehend gleich geblieben.

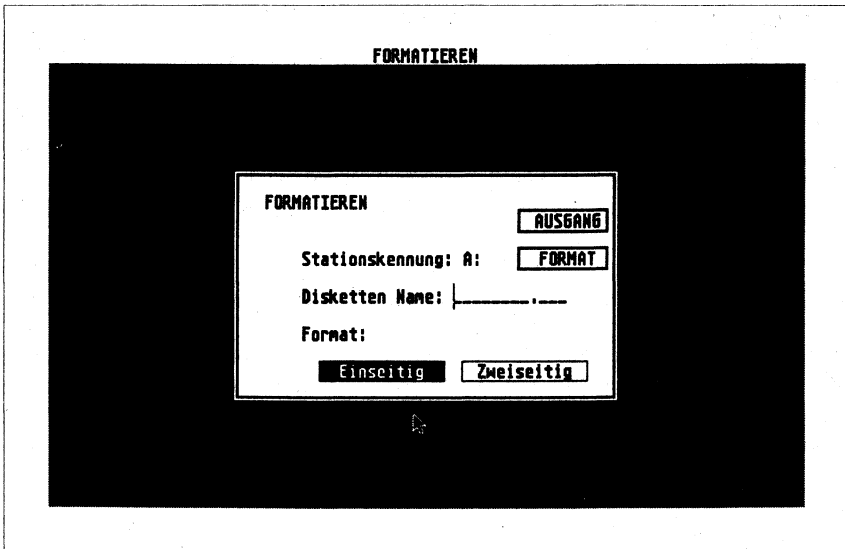


Abb. 2.8.1: Eingaben für die Formatierung

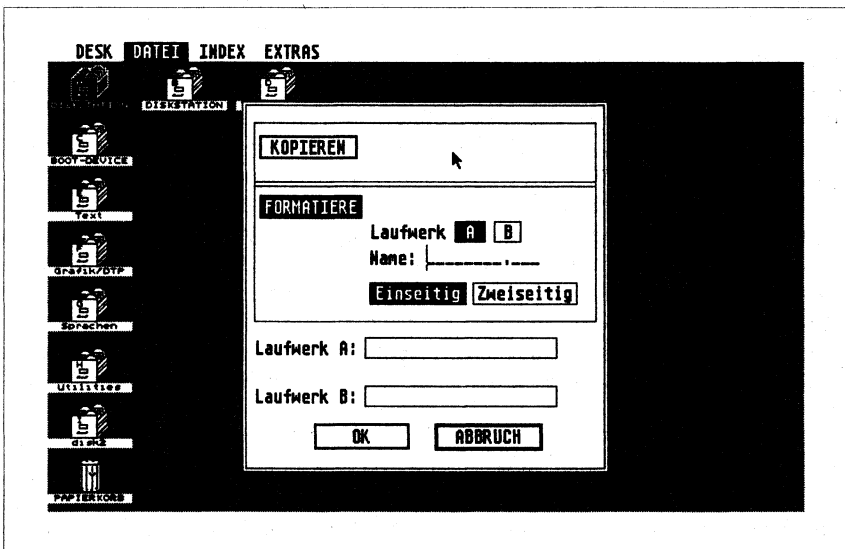


Abb. 2.8.2: Formatieren mit dem neuen TOS 1.04

Pannenhilfe:



Es erscheint eine Warnmeldung, daß die Diskette schreibgeschützt sei.



Der kleine Schreibschutzschieber in der einen Ecke der Diskette muß so stehen, daß das Loch verdeckt ist. Bevor Sie mit dem Formatieren weitermachen, sehen Sie aber lieber erst einmal nach, ob auf der Diskette vielleicht noch wichtige Daten stehen, die Sie durch den Schreibschutz sichern wollten!



Es erscheint eine Meldung, daß die Diskette defekt sei.



Pech gehabt. Diese Diskette ist so stark zerstört, daß der ST diese Fehler nicht mehr ausbügeln kann. Werfen Sie sie weg. Wenn Sie technisch interessiert sind, nehmen Sie die Diskette vorher einmal auseinander, um sich das Innenleben anzusehen.

2.5.3 Index

Das Menü "Index" hat sicher den unverständlichsten Titel. Hier können Sie bestimmen, in welcher Art und Weise der ST die Inhaltsverzeichnisse anzeigt. Die für Sie gewohnte Weise ist sicher die Anzeigeform "als Bilder", in der für jede Datei und jeden Ordner ein Icon steht. Als Alternative bietet sich "als Text" an. Diese Darstellungsform werden vor allem jene bevorzugen, die an die Arbeit mit MS-DOS- oder Großrechnern gewöhnt sind. Hier wird statt der Icons für jedes Objekt eine Schriftzeile gesetzt, in der nicht nur die Namen, sondern auch Dateigröße und Datum sowie Uhrzeit des Erzeugens angegeben sind. Bei Ordnern erscheint die Größe nicht, dafür ist ein Sonderzeichen vorangestellt. Der Umgang mit dieser Darstellungsart ist unter Umständen etwas schwieriger, da Sie Programme nur noch an Ihrer Endung (PRG, TOS, TTP) erkennen können und nicht mehr an der Iconform. Ansonsten gilt aber: Mit den Textzeilen können Sie all das tun, was auch mit Icons geht, also Verschieben, Anklicken, Doppelklicken usw. Dabei sollten Sie allerdings immer auch auf den Namen zielen, der Rest der Zeile fühlt sich durch die Maus nicht angesprochen.



Wundern Sie sich nicht, wenn die Erstellungsdaten sich bei Ihren Dateien stark ähneln. Wenn die Uhr nicht gestellt wird, läuft Sie mit dem Einschalten mit der Voreinstellung 6.2.1986, 0.00 Uhr los. Dieses Datum ist der Freigabetag Ihrer TOS-Version. Wenn Sie mit einem neueren TOS arbeiten, ist das Datum entsprechend jünger (z.B. 19.6.89 für das TOS 1.6 in den STE-Rechnern).

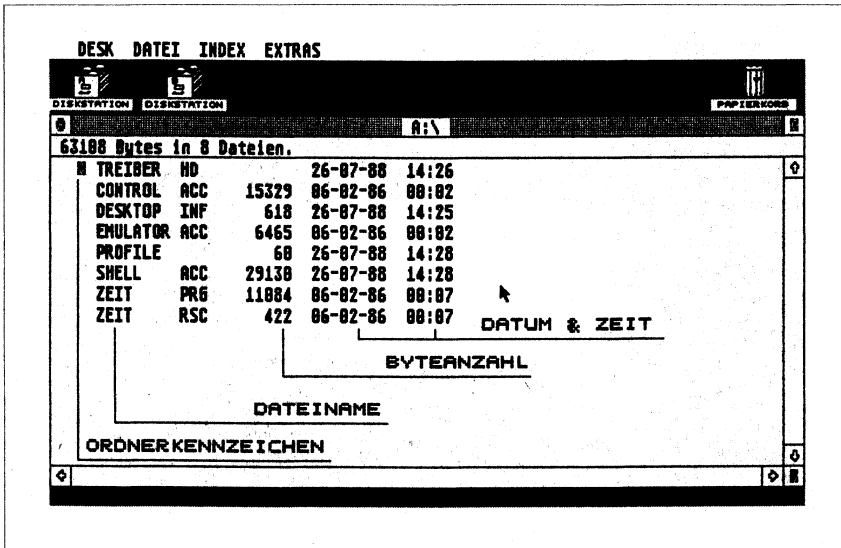


Abb. 2.9: Anzeige als Text

Mit den anderen Punkten im Menü INDEX können Sie bestimmen, nach welcher Kriterien der ST die Inhaltsverzeichnisse ordnet. "ordne Art" heißt, daß er die Icons oder Textzeilen nach den Endungen der Dateinamen ordnet (z.B. ZZZZZZZZ.AAA vor AAAAAAAAA.AAZ). Die anderen Möglichkeiten bedürfen wohl keiner Erläuterung.

2.5.4 Extras

Die Funktion "Floppy anmelden..." können Sie nur anwählen, wenn ein Datenstation-Icon angeklickt ist. Sie dient mehreren Zwecken, die jedoch alle damit zu tun haben, die Datenstation-Icons irgendwie zu manipulieren.

Die sicherlich einfachste Anwendung von "Floppy anmelden" ist das Ändern der Icon-Texte. Dazu klicken Sie die Station an, deren Text geändert werden soll, und wählen dann "Floppy anmelden..."; es erscheint eine Dialogbox, die so aussieht:

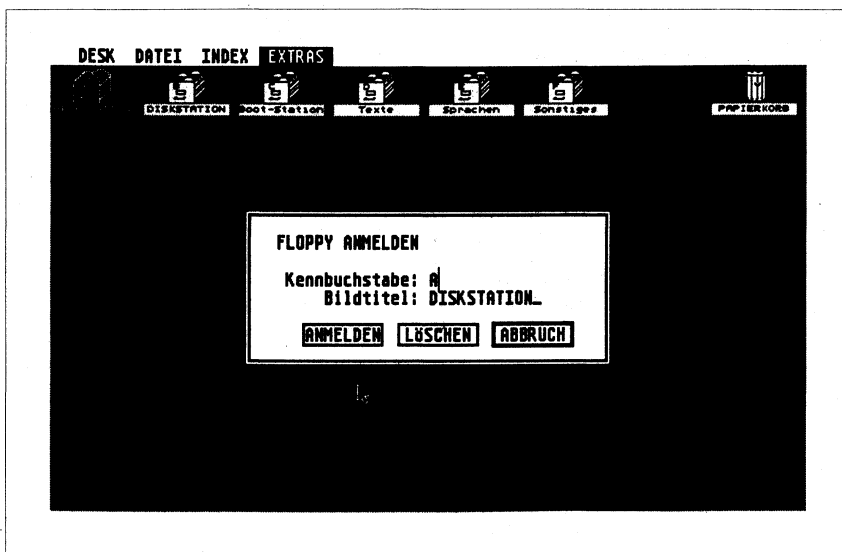


Abb. 2.10: Floppy anmelden

In der Box erscheinen die Kennung und der Titel des Laufwerks. Diesen Titel können Sie ändern, den Kennbuchstaben sollten Sie unverändert lassen. Durch Druck auf ANMELDEN übernimmt der ST dann Ihre Änderungen. ABBRUCH stellt wie üblich den Notausgang dar.

Die Option LÖSCHEN ist dagegen sehr gefährlich, weil damit das angewählte Laufwerk vom Desktop entfernt wird. Es steht dann ab sofort nicht mehr zur Verfügung, obwohl es vielleicht physisch noch vorhanden ist. Zwar kann man dies in den meisten Fällen wieder rückgängig machen, doch Sie können auch in einer Falle landen.

▼ Wenn Sie alle Laufwerke abmelden, haben Sie keine Möglichkeit mehr, Fenster zu öffnen. Weil Sie auch kein Laufwerk mehr invertieren können (es ist ja keines mehr da), ist Ihnen auch das Wiederanmelden verwehrt. Dann bleibt Ihnen nur noch das Neustarten des Rechners. Auf keinen Fall sollten Sie in dieser Situation die Funktion "Arbeit sichern" anwählen, denn diese würde den laufwerklosen Zustand untermauern.

Das Gegenstück zum Löschen ist das Anmelden neuer Laufwerke (z.B. einer RAM-Disk). Dazu invertieren Sie bitte ein beliebiges Stations-Icon und wählen dann "Floppy anmelden...". Bewegen Sie den Cursor in die Zeile mit dem

Kennbuchstaben, und tragen Sie dort die Kennung des neuen Laufwerks ein. (Diese wurde Ihnen entweder vom RAM-Disk-Programm mitgeteilt oder steht in der Anleitung des neuen Gerätes bzw. der RAM-Disk.). Wenn Sie möchten, können Sie nun noch den Titel ändern und dann das neue Laufwerk endgültig mit ANMELDEN dem ST bekannt machen.

"Anwendung anmelden" hat mit dem letzten Menüpunkt nur gewisse Namensähnlichkeiten. Es geht um eine sehr komfortable Funktion, mit der uns der Rechner einige Arbeit abnehmen kann. Stellen Sie sich vor, Sie möchten eine Adreßdatei mit einem Dateiverwaltungsprogramm erweitern. Dazu müssen Sie natürlich zuerst das Programm starten und dann in diesem die Adreßdatei laden und können dann erst loslegen. Einfacher wäre es, wenn Sie nur einen Doppelklick auf die Adreßdatei machen müßten, und automatisch würde das Programm gestartet und die Datei geladen. Genau dies ermöglicht die Funktion "Anwendung anmelden". Voraussetzung ist, daß sich a) Programm und Datei auf der gleichen Diskette befinden (Ausnahme: alle neueren TOS-Versionen ab 1.04) und b) alle Dateien, die von einem Programm erstellt werden, die gleiche Endung besitzen (das ist der Normalfall). Um bei unserem Beispiel zu bleiben nehmen wir an, das Dateiverwaltungsprogramm heiße DAT_VER.PRГ und lege Dateien mit der Endung .DAT an. Dann gehen Sie so vor:

1. Programm anwählen durch Mausklick (nicht starten).
2. "Anwendung anmelden" wählen. Es erscheint eine Dialogbox.
3. Unter Dateityp tragen Sie bitte die Endung der Dateien ein (in diesem Fall DAT).
4. Unter Anwendungsart müssen Sie bestimmen, von welchem Typ das Programm ist. Trägt es die Endung PRG, so müssen Sie "GEM" anklicken, bei TOS klicken Sie die gleichnamige Option, und "TOS erlaubt Parameter" wählen Sie, wenn Ihr Programm die Endung TTP trägt. Mit Klick auf OK übernimmt der ST Ihre Angaben.



Wenn Sie nach dem Anmelden noch "Arbeit sichern" verwenden, ist diese Einstellung auch nach dem nächsten Einschalten des Rechners wieder vorhanden, sonst geht Sie mit dem Ausschalten verloren. Ein Nachteil des Anmeldens sei nicht verschwiegen. Da der Doppelklick für das Starten eines Programmes gebraucht wird, kann man sich die Datei nicht mehr anzeigen lassen. Insbesondere bei Daten, die nicht aus einer Textverarbeitung stammen, verblaßt dieses Argument jedoch, weil hier der Ausdruck oder die Anzeige ohnehin nur mit Hilfe des Programms möglich ist.

Unter TOS 1.04 ist noch eine weitere Option hinzugekommen. Hier kann noch der sogenannte "Boot Status" festgelegt werden. Bei "Normal" ändert sich nichts - das Programm ist und bleibt ein ganz normales Programm. Wählen Sie dagegen "Auto", so wird das Programm jedesmal nach dem Einschalten und Laden der Accessories automatisch gestartet. Das ist unabhängig von der Zuordnung einer Extension.

Pannenhilfe:



Nach "Anwendung anmelden" bewirkt ein Doppelklick auf eine Datei zwar das Laden eines Programms, es wird jedoch das falsche Programm geladen.



Diese Anwendung war schon anderweitig angemeldet, und die neue Anmeldung hat die alte nicht überschrieben. Starten Sie den Rechner neu, und machen Sie einen Doppelklick auf die gleiche Datei wie zuvor. Wird diese nicht angezeigt, sondern ein Programm gestartet, so ist die alte Anmeldung auf der Diskette gespeichert. Sie müssen dann die Datei DESKTOP.INF löschen (dabei gehen auch andere Voreinstellungen wie die Position der Icons und Fenster verloren), neu starten und können dann erneut anmelden.

Falls es in Ihrem EXTRAS-Menü noch eine Funktion "Blitter" gibt, sind Sie einer der wenigen Besitzer eines ST mit dem sogenannten Blitter-Chip. Das ist ein Schaltkreis, der die Bildschirmausgabe beschleunigt. Atari baut diesen Baustein zur Zeit in alle Mega-STs und die STE-Versionen ein. Wenn Ihr Gerät noch keinen Blitter besitzt, können Sie ihn beim Händler nachrüsten lassen. Verbunden mit diesem Blitter ist auch eine eigene Version des Betriebssystems TOS, die automatisch erkennt, ob ein solcher Chip eingebaut ist. Nur wenn dies der Fall ist, erscheint auch die Funktion Blitter im EXTRAS-Menü. Sie können dann den Blitter ein- bzw. ausschalten, der Haken zeigt einen eingeschalteten Blitter an. Manche Programme arbeiten mit dem Blitter nicht korrekt zusammen, etwa Spiele, weil dann die Bildschirmausgabe zu schnell wird. Dann können Sie hier den Blitter abschalten und haben einen ganz gewöhnlichen ST vor sich.

"Voreinstellung" dient dazu, Art und Umfang der Sicherheitsabfragen festzulegen. Wie Sie sicher schon bemerkt haben, fragt der ST bei allen Lösch- und Kopiervorgängen erst noch einmal nach, bevor er sie ausführt. Diese Abfragen können Sie mit der Funktion "Voreinstellung" ausschalten - was ich nicht empfehle. Das versehentliche Verschieben eines Icons passiert auch erfahrenen Anwendern recht häufig. Wenn dann sofort ohne Nachfrage gelöscht wird, kann es Datenverluste geben.

Zusätzlich können Besitzer eines Farbmonitors hier festlegen, welche Auflösungsstufe verwendet werden soll. Achtung! Eine Änderung der Auflösungsstufe zieht auf jeden Fall einen Reset (d.h. Neustart) des Rechners nach sich.

"Arbeit sichern" ist im Gegensatz dazu ein sehr nützliches Hilfsmittel. Nach einem Klick auf diesen Menüpunkt wird auf der Diskette eine Datei namens DESKTOP.INF angelegt bzw. geändert. In dieser Datei stehen alle Einstellungen, die Sie bisher auf dem Desktop vorgenommen haben, auch jene aus den Accessories (also Kontrollfeld, Druckeranpassung, RS.232-Einstellung). Sogar die Positionen der Icons merkt sich der Rechner. Beim nächsten Starten des ST mit dieser Diskette werden Sie dann den Bildschirm wieder so vorfinden, wie Sie ihn beim letzten Mal verlassen haben. Haben Sie von der Festplatte gebootet, wird die DESKTOP.INF automatisch auf dem Boot-Laufwerk abgelegt.



Durch einen Doppelklick können Sie sich auch die Datei DESKTOP.INF anzeigen lassen. Dabei werden Sie einige Informationen erhalten. Ihre Anzeige sieht ungefähr so aus:

```
#a000000
#b001000
#c7770007000700070055200505552220770557075057705502112004
#d
#E 1B 03
#W 00 00 00 04 50 14 08 A:\*. *@
#W 00 00 0E 08 2A 0B 00 @
#W 00 00 0E 09 2A 0B 00 @
#W 00 00 0F 0A 2A 0B 00 @
#M 03 00 00 FF D RAM-DISK@ '1@
#M 00 00 00 FF A DISKSTATION@ @
#M 01 00 00 FF B DISKSTATION@ @
#T 07 00 02 FF PAPIERKORB@ @
#F FF 04 @ *. *@
#D FF 01 @ *. *@
#G 03 FF *.PRG@ @
#F 03 04 *.TOS@ @
#P 03 04 *.TTP@ @
#G 03 04 GFABASIC.PRG@ *.BAS@
```



Interessant ist hier vor allem die unterste Zeile. Sie besagt, daß die Dateien mit der Endung BAS für das Programm GFABASIC.PRG angemeldet sind. Weiter oben können Sie sehen, daß ein zusätzliches Laufwerk D mit dem Titel RAM-DISK angemeldet wurde. Wenn Sie sich also einmal nicht sicher sind, welche Laufwerke und Anwendungen Sie angemeldet haben, können Sie sich so Gewißheit verschaffen.

Sie können die DESKTOP.INF auch mit einem Texteditor manipulieren! Allerdings müssen Sie dabei ein Detail beachten. Das letzte Byte der Datei muß ein

Ctrl-Z (ASCII 26) sein, das Sie nicht mit allen Editoren eingeben können. Sie können es durch folgende Befehlsfolge in GFA-BASIC nachträglich anhängen:

```
OPEN "A", #1, "DESKTOP.INF"
OUT 1, 26
CLOSE #1
```

Im Omikron-BASIC dient dazu folgendes Programm:

```
1 OPEN "A", #1, "DESKTOP.INF"
2 PRINT#1, CHR$(26)
3 CLOSE 1
```

Manipuliert werden können vor allem die Zeilen, in denen File-Namen oder Suchmuster wie "*" gespeichert sind. Wenn Sie etwa in der ersten Zeile, die mit "#W" beginnt, die Sequenz "A:*.*" in "A:*.PRG" ändern, werden im Fenster nur noch solche Dateien angezeigt, deren Extension PRG ist.

Zeilen, die mit "#M" beginnen, beziehen sich auf angemeldete Laufwerke. Änderungen an diesen Zeilen sind leichter mit der Funktion "Floppy anmelden" des Desktop möglich. Ähnliches gilt für die anderen Zeilen, die vor allem Fensterpositionen u.ä. enthalten.

Doch zurück zum Menübaum. "Hardcopy" dient dazu, den gegenwärtigen Bildschirminhalt auf dem Drucker auszugeben. Sie können diese Funktion auch durch gleichzeitigen Druck auf **Alternate** und **Help** auslösen.



Weniger bekannt dürfte sein, daß die gleiche Tastenkombination die Hardcopy auch wieder abbrechen kann. Allerdings scheint das nicht immer zu klappen. Nachdem Sie zum zweiten Mal **Alternate**-**Help** gedrückt haben, muß der Drucker nicht sofort mit der Ausgabe aufhören. Der zweite Tastendruck bedeutet nur, daß der ST seine Übertragung an den Drucker stoppt. Wenn dieser aber noch Daten in seinem Speicher hat, die er schon übernommen, aber noch nicht ausgedruckt hat, kann es sein, daß der Druck noch einige Sekunden weitergeht. Wie lange, das hängt von der Größe des Druckerspeichers ab. Bei Puffergrößen über 32 K kann es sein, daß der ST die Übertragung schon in Sekundenbruchteilen beendet hat, ehe Sie auch nur die Gelegenheit hatten, die Hardcopy zu beenden. Dann ist unter Umständen schon das gesamte Bild im Drucker und wird komplett ausgedruckt.

Pannenhilfe:



Sie haben eine Hardcopy ausgelöst, obwohl kein Drucker angeschlossen oder dieser ausgeschaltet ist. Der Rechner scheint "abgestürzt" zu sein.

✓ Warten Sie mindestens 30 Sekunden lang. Diese Zeitspanne braucht der ST, um selbst das Fehlen des Druckers zu bemerken. In einem solchen Fall bricht der Rechner die Hardcopy selbst ab.



Die Hardcopy wird zwar gedruckt, ist aber fehlerhaft.



Ihre Druckeranpassung ist nicht in Ordnung. Sehen Sie bitte im Kapitel über Drucker nach, wie das geändert werden kann.

2.6 Der Umgang mit der Objektauswahl

Die Objektauswahl dient dazu, für das Speichern oder Laden von Dateien den Pfadnamen festzulegen. Dies kann sowohl über die Tastatur als auch per Mausklick geschehen. Die Objektauswahl wird auch File-Select-Box genannt.

Die Objektauswahl erscheint, wenn Sie in diversen Programmen (z.B. Textverarbeitungen) eine Datei von Diskette oder Harddisk laden bzw. darauf speichern möchten und dazu den entsprechenden Menüpunkt gewählt haben. Auf Ihrem Bildschirm erscheint dann in etwa folgendes Bild.

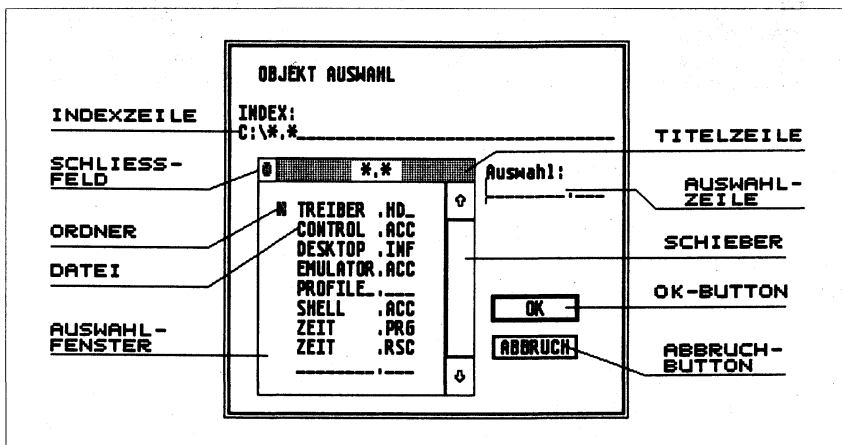


Abb. 2.11.1: Die sogenannte File-Select-Box

Ihre Objektauswahl sieht anders aus. Woran liegt das?

Es gibt zwei mögliche Gründe dafür. Wenn Sie mit TOS 1.04 arbeiten, dann haben Sie schon die Lösung. Hier wurde die Objektauswahl völlig umgekre-

pelt. Äußerlich erkennen Sie das an 16 zusätzlichen Knöpfen, mit denen Sie andere Laufwerke anwählen können.

In einigen Programmen (z.B. Tempus, Turbo C) haben sich die Programmierer auch die Mühe gemacht und eine eigene Objektauswahl programmiert. Die Bedienung ist aber in den meisten Fällen ganz analog zum Original.

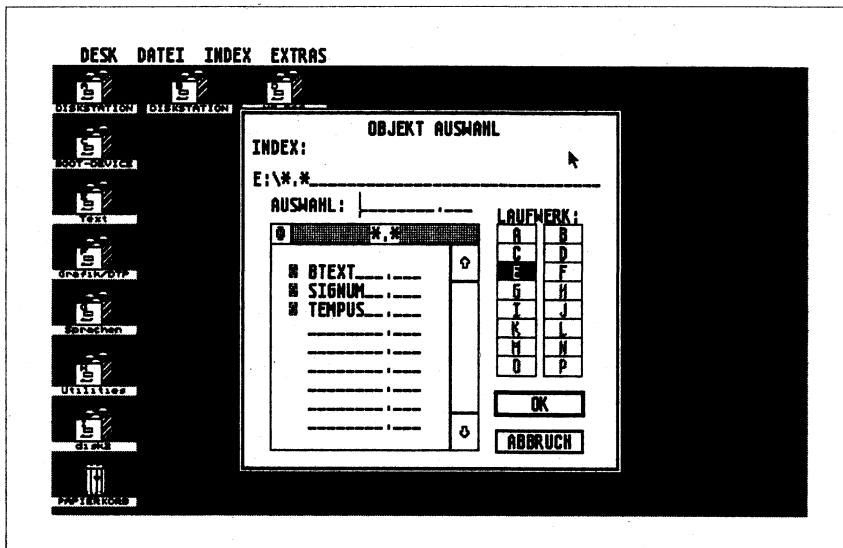


Abb. 2.11.2: Die neue Objektauswahl unter TOS 1.04

Unter der Überschrift "Index" finden Sie den sogenannten Pfad angezeigt. Der Pfad beschreibt, in welchem Ordner eine Datei zu finden ist, während der eigentliche Dateiname den Ort innerhalb dieses Ordners angibt.

In der Objektauswahl wird der komplette Pfadname zweigeteilt. Während der Pfad in der oberen Zeile steht, wird der eigentliche Dateiname in der unteren Textzeile unter "Auswahl" angegeben.

Die beiden Eingabezeilen haben zwei Funktionen. Sie zeigen den bisher ausgewählten Pfad bzw. Namen und dienen auch der Eingabe neuer Daten. Beim Erscheinen der Objektauswahl sehen Sie den Cursor (in diesem Fall ein senkrechter Strich) in der Namenszeile. Sie können diese Zeile wie auch sonstige Eingabezeilen beschreiben. **[Insert]** fügt ein Leerzeichen an der Cursorposition ein, **[Delete]** löscht das Zeichen an, **[Backspace]** dasjenige vor der Cursorposition. Den Cursor können Sie mit den Cursortasten nach links oder

rechts bewegen. **[Esc]** löscht die gesamte Eingabezeile. Außerdem hat der Punkt noch eine besondere Bedeutung. Wenn Sie diese Taste drücken, springt der Cursor automatisch an den Anfang der Dateinamen-Erweiterung, der sogenannten Extension, im obigen Beispiel wäre dies TXT. In der Eingabezeile sind einige Tasten gesperrt, weil deren Verwendung zu ungültigen Dateinamen führen würde.

Übrigens ist es wichtig, daß Sie für Pfade den sogenannten "Backslash" benutzen. Das ist der Schrägstrich, der von links oben nach rechts unten verläuft. Sie erreichen ihn auf der Tastatur durch das gleichzeitige Drücken der Tasten **[Shift]**, **[Alternate]** und **[ü]**.

Befindet sich der Cursor in der Namenszeile und Sie drücken **[↑]**, dann springt dieser an das Ende des Pfades. Im umgekehrten Fall führt **[↓]** in die Namenszeile.

Unter der Pfadanzeige finden Sie ein kleines Fenster, das nicht bewegt werden kann. In der Überschriftzeile steht, welcher Dateityp (Extension) ausgewählt wurde. In unserem Beispiel wäre das also *.TXT. Darunter werden alle Dateien angezeigt, die

- a) in dem Ordner stehen, der durch den Pfad angegeben wird und die
- b) eine Extension haben, die zur Fensterüberschrift paßt.

Ein Anklicken der Titelzeile hat die gleiche Wirkung wie der Druck auf die **[Esc]**-Taste in einem Fenster des Desktop: Das angezeigte Inhaltsverzeichnis wird aktualisiert, d.h. es wird dem angepaßt, was auf der gerade eingelegten Diskette gespeichert ist. Wenn Sie also nach dem Erscheinen der Objektauswahl die Diskette gewechselt haben, können Sie mit einem Mausklick in dieses Feld das neue Inhaltsverzeichnis anzeigen lassen. Falls Sie aber die INDEX-Zeile vorher geändert haben, geht hierbei die Extension verloren, sofern Sie mit dem alten TOS arbeiten!

Falls nicht alle passenden Dateinamen in das kleine Fenster passen, können Sie wie im Desktop die Anzeige mit dem Schieber und den kleinen Pfeilen an der rechten Seite auf- und abbrollen lassen.

Ein einfacher Klick auf einen der angezeigten Namen führt dazu, daß dieser im Fenster invertiert dargestellt wird. Gleichzeitig wird er in die Namenszeile übertragen. Führen Sie einen Doppelklick auf einen Namen aus, so wird der Name endgültig gewählt, d.h. die Objektauswahl wird beendet und der angeklickte Name an Ihr Programm übermittelt. Das gleiche können Sie errei-

chen, wenn Sie den Namen nur einmal anklicken oder eintippen und dann entweder auf die OK-Box klicken oder die Return-Taste drücken.

Klicken Sie auf die Abbruch-Box, so hat dies im Normalfall zur Folge, daß die Diskettenoperation abgebrochen wird. Haben Sie also irrtümlich einen Menüpunkt wie LADEN o.ä. aufgerufen, so können Sie dies durch "Abbruch" wieder rückgängig machen.

Falls das angezeigte Verzeichnis selbst wieder Ordner enthält, sind diese durch ein vorgestelltes Sonderzeichen markiert (siehe Abbildung 2.11.2). Ein einfacher Klick auf einen solchen Ordnernamen reicht aus, und die Anzeige wechselt in dieses Unterverzeichnis. Sie können aus diesem Verzeichnis wieder herauskommen und in das übergeordnete gelangen, indem Sie das kleine Schließfeld in der linken oberen Ecke anklicken.

Eine andere Möglichkeit zum Wechsel des Ordners ist, dessen Pfad in die INDEX-Zeile zu schreiben und dann den Schieber (beim alten TOS) oder die Titelzeile des Fensters (beim neuen TOS 1.04) anzuklicken.

Um die Verzeichnisse eines anderen Laufwerks anzeigen zu lassen, müssen Sie beim alten TOS den Pfad unter INDEX ändern. Danach klicken Sie bitte die Fensterüberschrift an. Nehmen wir an, Sie möchten Daten von der RAM-Disk (Laufwerkskennung D) laden. Nach dem Anwählen der LADE-Funktion zeigt die Objektauswahl aber den Pfad

A:\ORDNER*.*

an. Wenn Sie jetzt in die Indexzeile gehen und dort den neuen Pfad

D:*.*

eingeben, so erhalten Sie nach einem Klick auf die Fensterüberschrift das Hauptverzeichnis der RAM-Disk angezeigt. Einen Haken hat dieses Verfahren allerdings. Bei solchen Operationen geht immer die Extension verloren, d.h. auch wenn Sie D:*.*TXT eingetippt hätten, wären nach dem Klick auf die Überschrift alle Dateien des Verzeichnisses angezeigt worden - auch jene ohne Endung TXT.

Beim neuen TOS 1.04 ist das etwas bequemer. Zum Wechsel des Laufwerks reicht ein Klick auf den zugehörigen Knopf an der rechten Seite der Objektauswahl. Dabei geht nicht einmal die Extension verloren.

Wenn Sie Dateien mit einer anderen Extension als der, die in der INDEX-Zeile angezeigt wird, erreichen wollen, tun Sie folgendes: Fahren Sie mit dem Cur-

sor in die Indexzeile, und ändern Sie die Extension nach Ihren Wünschen. Danach klicken Sie bitte den Schieber (beim alten TOS) oder die Titelzeile (beim neuen TOS 1.04) an. Dadurch wird die neue Extension angemeldet, und die Anzeige weist nur noch Dateien mit dieser Endung aus.



Wenn Sie beim alten TOS nach dem Ändern der INDEX-Zeile die Titelzeile anklicken, geht auf jeden Fall die Extension verloren (d.h. sie wird auf *.* geändert).

Wenn Sie Namen eingeben wollen, die überhaupt noch nicht auf der Diskette existieren, wählen Sie zunächst den Ordner aus, in dem die Datei stehen soll. Dann tippen Sie den gewünschten Namen in die untere Eingabezeile.



In letzter Zeit war bei einigen Software-Herstellern die Tendenz zu spüren, die vom Betriebssystem gelieferte Objektauswahl durch eine eigene Auswahlbox zu ersetzen, die dann mehr Komfort bringen sollte. Das mag im Hinblick auf den Fehler mit der Dateieindung vielleicht berechtigt sein. Es führt aber auch dazu, daß die Auswahl eines File-Namens immer uneinheitlicher wird. Leider ergeben sich bei der Verwendung dieser hausgemachten Objektauswahlen so viele nicht vorhersehbare Möglichkeiten, daß es unmöglich ist, diese hier darzustellen. Es sollte Ihnen aber jetzt leicht fallen, die Erläuterungen des jeweiligen Herstellers zu diesem Thema zu verstehen (siehe die jeweilige Gebrauchsanweisung zu dem benutzten Programm).

2.7 Der Umgang mit Dialogboxen

Als Dialogboxen werden alle jene "Fenster" bezeichnet, die der Kommunikation zwischen Rechner und Anwender dienen. Dialogboxen haben sehr unterschiedliches Aussehen, die Gestaltung hängt immer vom jeweiligen Programm ab, in dem diese Boxen auftauchen. Sie haben auch schon Bekanntschaft mit dieser Einrichtung gemacht, so stellt zum Beispiel der Kasten, der beim Formatieren von Disketten erscheint, eine Dialogbox dar. Streng genommen sind auch Objektauswahlfelder solche Dialogboxen. Ich möchte mich daher auf die Erläuterung der einzelnen Elemente beschränken, die für gewöhnlich in solchen Dialogen auftauchen.

Die einfachste Form einer Dialogbox ist ein Text, der von einem oder mehreren Knöpfen (der Fachausdruck lautet Buttons) begleitet wird. Der Text soll meist auf einen Fehler hinweisen oder mehrere Alternativen erläutern, von denen Sie dann mittels der Buttons eine auswählen müssen.

Das Auswählen geschieht durch einfaches Anklicken des jeweiligen Buttons. Sehr häufig ist auch einer der Knöpfe dicker umrandet als die anderen. Man nennt ihn dann auch "Default-Button" (engl.: Voreinstellungsknopf). Drücken Sie in einer solchen Situation der **Return** - oder **Enter** -Taste, so hat das den gleichen Effekt wie ein Mausklick in diesen Default-Button.

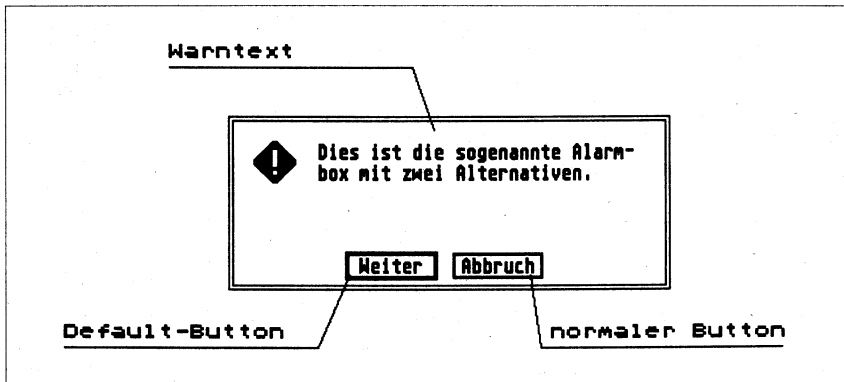


Abb. 2.12: Eine einfache Dialogbox (auch Alarmbox genannt)

Dialogboxen hätten Ihren Namen nicht verdient, wenn dort nicht auch Daten eingegeben werden könnten. Dies geschieht über Eingabezeilen, wie Sie sie schon von Objektauswahlen und anderen Funktionen her kennen. Je nach Programm können hier verschiedene Typen von Daten eingegeben werden: Zahlen, Texte, einzelne Buchstaben usw.

Je nach dem, was der Programmierer vorgesehen hat, sind einige Tasten gesperrt. Wenn Sie also einen Dateinamen eintippen sollen, zeigt die **Leertaste** keine Wirkung, weil Leerzeichen in Dateinamen nicht vorkommen dürfen. Umgekehrt werden bei einer Zahleneingabe die Buchstabentasten gesperrt sein. Für den Umgang mit diesen Eingabezeilen gelten jedoch immer die gleichen Regeln:

Esc löscht die gesamte Eingabezeile, der Cursor wird nach ganz links gesetzt.

Die Cursortasten links und rechts bewegen den Cursor innerhalb der Zeile, soweit dies möglich ist.

Die Cursortasten auf und ab lassen der Cursor in die nächste bzw. vorhergehende Eingabezeile wechseln (falls vorhanden).

Mit **Insert** fügen Sie rechts der Cursorposition ein Zeichen ein. Passen dadurch nicht mehr alle Zeichen in diese Zeile, geht das ganz rechts stehende verloren.

Backspace und **Delete** löschen das Zeichen links vom bzw. am Cursor. Der Rest der Zeile rückt nach links nach.

Außerdem können Sie den Cursor jederzeit mit der Maus bewegen, d.h. positionieren Sie den Mausfeil auf die Zeile, in der der Cursor erscheinen soll. Ein Druck auf die linke Maustaste sorgt dann dafür, daß der Cursor in diese Zeile gelangt.

In manchen Fällen ist die Eingabezeile durch irgendwelche Zeichen unterbrochen (siehe auch Abb. 2.13.). Falls der Programmierer es vorgesehen hat, bewirkt ein Druck auf eines dieser Trennzeichen, daß der Cursor zur nächsten Position vorrückt (wie Sie das auch von der Objektauswahl her kennen).

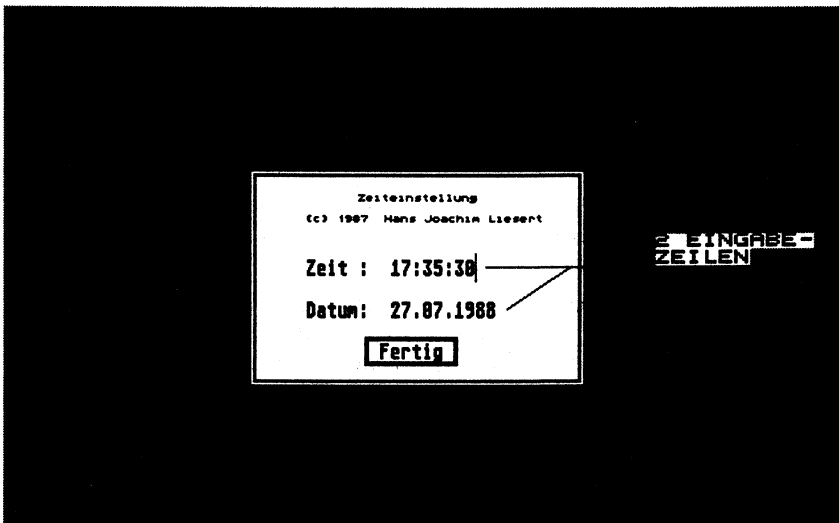


Abb. 2.13: Eine einfache Dialogbox mit unterbrochenen Eingabezeilen

Ebenso können in Dialogboxen noch die sogenannten "Radio-Buttons" vorkommen. Das Radio ist hier wörtlich zu nehmen. Diese Knöpfe arbeiten nämlich wie die Tasten eines guten alten Röhren-Radios. Drückt man eine hinein, springen alle anderen wieder heraus. Genauso kann man bei Radio-Buttons immer nur eine der angebotenen Alternativen anwählen. Der Unterschied zu den bisher bekannten Buttons besteht darin, daß der Dialog durch ein Klicken

in einen Radio-Button nicht abgebrochen wird. Sie können also mehrfach probieren und sich die nötigen Einstellungen überlegen.

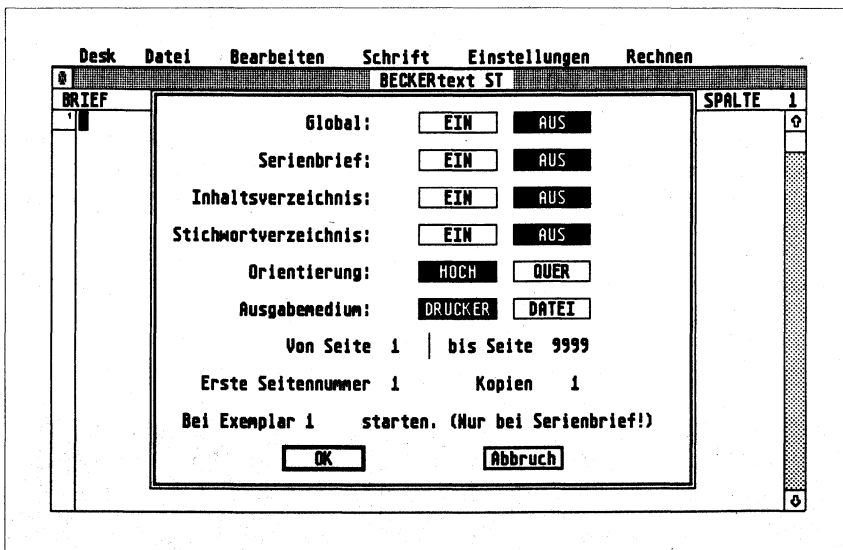


Abb. 2.14: Dialogbox mit Radio-Buttons (Die oberen Buttons sind vom Typ Radio, darunter folgt Text mit eingestreuten Eingabezeilen und ganz unten die normalen Buttons, die den Dialog beenden.)

Über die beschriebenen Möglichkeiten hinaus ist es jedem Programmierer selbst überlassen, welche Elemente er in seine Dialogboxen einbaut. Daher sollte Sie der seltene Fall nicht wundern, daß auch Schieber oder sogar komplette Fenster in einer Dialogbox auftauchen. Dieser werden normalerweise genau wie auf dem Desktop bedient.

3. Massenspeicherung

Unter Massenspeichern versteht man in der EDV solche Geräte, die in der Lage sind, große Datenmengen dauerhaft (d.h. auch nach dem Abschalten des Stroms) zu speichern und wieder auszugeben. Das Gegenstück dazu ist der Arbeitsspeicher Ihres Computers, der schnell, aber dafür klein und teuer ist. In ihm werden nur die Daten abgelegt, mit denen der Rechner gerade arbeitet. Arbeitsspeicher verlieren ihren Inhalt normalerweise, wenn der Strom abgeschaltet wird.

Bei der Arbeit mit dem ST werden die Massenspeicher durch die Datenstationen (Icons mit Aktenschränken) symbolisiert. Für den Umgang damit gibt es besondere Regeln. Das fängt zum Beispiel damit an, daß jeder Eintrag in den Massenspeicher einen Namen haben muß, um ihn später wieder identifizieren zu können.

Die weiter unten besprochene RAM-Disk ist sozusagen ein "Wanderer zwischen den Welten", denn sie simuliert im Arbeitsspeicher einen Massenspeicher.

3.1 Das hierarchische File-System

Dateien werden vom ST nicht einfach auf die Diskette oder Festplatte geschrieben. Sie werden in eine bestimmte Hierarchie von Ordnern gesetzt, die ganz unterschiedlich aussehen kann. Wer richtig mit dieser Hierarchie umgeht, findet seine Daten sehr viel schneller wieder.

3.1.1 Dateien

Der Begriff Datei (im Engl.: File) wird im Fachjargon recht uneindeutig verwendet. Wir wollen uns darauf einigen, darunter jede Ansammlung von Daten auf einer Datenstation zu verstehen, die mit einem Namen versehen ist. Demzufolge sind auch Programme Dateien - ausführbare Dateien, deren Inhalt Programmcode ist. Man nennt solche Files deshalb auch Programmdateien. Sie können sie wie jedes andere Daten-File auch behandeln, mit dem einzigen Unterschied, daß ein Öffnen zum Starten des darin gespeicherten Programms führt.

Dateien werden auf zwei verschiedenen Wegen angelegt. Entweder entstehen sie durch Kopieren einer anderen Datei oder sie werden von Programmen er-

zeugt. Beispiel: Sie haben mit einem Grafikprogramm ein Bild erstellt. Beim Speichern dieses Bildes erzeugt das Programm eine neue Datei und legt darin die Bilddaten ab.

Um Dateien umzubenennen, klicken Sie die betreffende Datei mit der linken Maustaste an, und rufen Sie den Menüpunkt "Info anzeigen..." auf. In der darauffolgenden Dialogbox können Sie dann den Namen ändern.

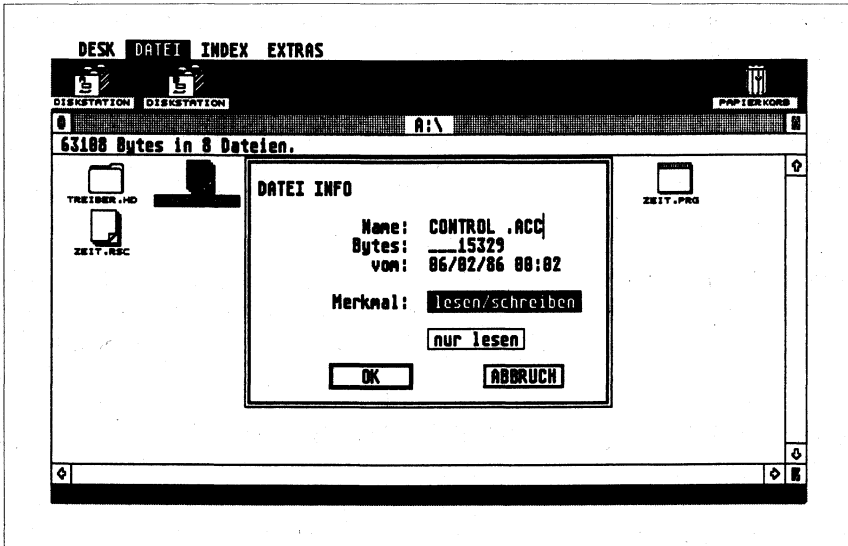


Abb. 3.1: Hier kann der Dateiname geändert werden

Eine vielgestellte Frage ist: "Kann ich die Uhrzeit und das Datum einer Datei auch ändern?", Das Desktop läßt die Änderung dieser Daten nicht zu. Abhilfe schafft auch hier ein kleines Programm:

Version für GFA-BASIC:

```
' Programm zum Ändern von Uhrzeit und Datum eines Files
' File-Namen ermitteln
Fileselect "E:\*.*", "", F$
If (F$ <> "") And (Exist(F$)) Then
  Print "File "; F$
  ' C-Format für File-Namen herstellen
  F$ = F$ + Chr$(0)
  ' File eröffnen
  A% = Gemdos(&H3D, L:Varptr(F$), 0)
```

```

If A%<0 Then
    Print "GEMDOS-Fehlermeldung ";A%
    Stop
Else
    ' A% enthält das Handle der geöffneten Datei (wichtig!!!)
    H%=A%
Endif
' Puffervariable für Zeit/Datum bereitstellen
Buffer%=0
' Datum und Zeit aus der Datei holen
Void Gemdos(&H57,L:Varptr(Buffer%),H%,0)
' Datum und Zeit dekodieren
Zeit%=(Buffer% And (2^31-1))-(Buffer% And -2^31) Div 65536 Datum%=(Buffer%
And (2^31-1)) Mod 65536
Tag%=Datum% Mod 32
Datum%=Datum% Div 32
Monat%=Datum% Mod 16
Jahr%=1980+(Datum% Div 16)
Sekunde%=(Zeit% Mod 32)*2
Zeit%=Zeit% Div 32
Minute%=Zeit% Mod 64
Stunde%=Zeit% Div 64
' Daten zur Ausgabe aufbereiten
Minute%=Left$("00",3-Len(Str$(Minute%)))+Str$(Minute%)
Sekunde%=Left$("00",3-Len(Str$(Sekunde%)))+Str$(Sekunde%)
Print Tag%;". ";Monat%;". ";Jahr%
Print Using "##",Stunde%;
Print Minute%;Sekunde%
Print "Wollen Sie Änderungen vornehmen?"
A%=Inp(2)
If A%=Asc("j") Then
    ' Datum und Zeit ändern
    Input "Tag ";Tag%
    Input "Monat ";Monat%
    Input "Jahr ";Jahr%
    Input "Sekunde ";Sekunde%
    Input "Minute ";Minute%
    Input "Stunde ";Stunde%
    ' Daten kodieren und schreiben
    Zeit%=(Sekunde% Div 2)+Minute%*32+Stunde%*2048
    Datum%=Tag%+Monat%*32+(Jahr%-1980)*512
    Buffer%=(Zeit% And 32768)*-65536+(Zeit% And 32767)*65536+Datum%
    Void Gemdos(&H57,L:Varptr(Buffer%),H%,1)
Endif
' File wieder schließen
A%=Gemdos(&H3E,H%)
If A%<0 Then
    Print "GEMDOS-Fehlermeldung ";A%
    Stop
Endif
Endif

```

Hier die Version für Omikron- bzw. ST-BASIC:

```

0 ' DATUMSET - Programm zum Ändern von Uhrzeit und Datum eines Files
1 ' Version für Omikron- bzw. ST-BASIC
2 MOUSEON
3 ' File-Namen ermitteln
4 Pfad$="E:\*.*"
5 Nam$=""
6 FILESELECT (Pfad$,Nam$,Ok%)
7 IF (Ok%=1) THEN
8   F$= LEFT$(Pfad$, LEN(Pfad$)- INSTR( MIRROR$(Pfad$)+"\\","\")+1)+Nam$
9   CLS
10  PRINT "File ";F$
11  ' C-Format für File-Namen herstellen
12  F$=F$+ CHR$(0)
13  F$L= LPEEK( VARPTR(F$))+ LPEEK( SEGPTR +28)
14  ' File eröffnen
15  GEMDOS (A%,61, HIGH(F$L), LOW(F$L),0)
16  IF A%<0 THEN
17    PRINT "GEMDOS-Fehlermeldung ";A%
18    STOP
19  ELSE
20    ' A% enthält das Handle der geöffneten Datei (wichtig!!!)21   H%=A%
22  ENDIF
23  ' Puffervariable für Zeit/Datum bereitstellen
24  Buffer$L=0
25  ' Datum und Zeit aus der Datei holen
26  GEMDOS (A%,87, HIGH( VARPTR(Buffer$L)), LOW( VARPTR(Buffer$L)),H%,0)
27  ' Datum und Zeit dekodieren
28  Zeit$L=(Buffer$L AND (2^31-1))-(Buffer$L AND -2^31))\6553629
Datum$L=(Buffer$L AND (2^31-1)) MOD 65536
30  Tag$L=Datum$L MOD 32
31  Datum$L=Datum\L\32
32  Monat$L=Datum\L MOD 16
33  Jahr$L=1980+(Datum\L\16)
34  Sekunde$L=(Zeit\L MOD 32)*2
35  Zeit$L=Zeit\L\32
36  Minute$L=Zeit\L MOD 64
37  Stunde$L=Zeit\L\64
38  ' Daten zur Ausgabe aufbereiten
39  Minute$= RIGHT$( STR$(Minute$L), LEN( STR$(Minute$L))-1)
40  Sekunde$= RIGHT$( STR$(Sekunde$L), LEN( STR$(Sekunde$L))-1)41  Minute$=
LEFT$(":"00",3- LEN(Minute$))+Minute$
42  Sekunde$= LEFT$(":"00",3- LEN(Sekunde$))+Sekunde$
43  PRINT "Datum: ";Tag$L;".";Monat$L;".";Jahr$L
44  PRINT "Uhrzeit: ";Stunde$L;Minute$;Sekunde$
45  PRINT "Wollen Sie Änderungen vornehmen?"
46  A$= INPUT$(1)
47  IF A$="j" THEN
48    ' Datum und Zeit ändern
49    INPUT "Tag ";Tag$L
50    INPUT "Monat ";Monat$L
51    INPUT "Jahr ";Jahr$L

```

```

52 INPUT "Sekunde ";Sekunde%L
53 INPUT "Minute ";Minute%L
54 INPUT "Stunde ";Stunde%L
55 ' Daten kodieren und schreiben
56 Zeit%L=(Sekunde%L\2)+Minute%L*32+Stunde%L*2048
57 Datum%L=Tag%L+Monat%L*32+(Jahr%L-1980)*512
58 Buffer%L=(Zeit%L AND 32768)*-65536+(Zeit%L AND 32767)*65536+Datum%L
59 GEMDOS (A%,87, HIGH( VARPTR(Buffer%L)), LOW( VARPTR(Buffer%L)),H%,1)
60 ENDIF
61 ' File wieder schließen
62 GEMDOS (A%,62,H%)
63 IF A%<0 THEN
64 PRINT "GEMDOS-Fehlermeldung ";A%
65 STOP
66 ENDIF
67 ENDIF

```



Wenn Sie DATUMSET starten, erscheint zunächst eine Objektauswahl auf dem Bildschirm. Wählen Sie aus, zu welcher Datei die Daten geändert werden sollen. Falls Sie hierbei ABBRUCH oder nicht existente Dateien wählen, bricht das Programm ab. Andernfalls werden Datum und Uhrzeit von der Diskette geholt und angezeigt. Daraufhin fragt das Programm, ob Sie diese Werte ändern möchten. Falls ja, tippen Sie bitte ein ☐. Sie können dann der Reihe nach alle Werte eingeben. Beachten Sie bitte, daß das Jahr aus technischen Gründen größer als 1980 sein muß. Geben Sie falsche Werte ein, so kann dies unter Umständen zu Fehlermeldungen führen. Nachdem Sie die letzte Eingabe mit oder abgeschlossen haben, werden die neuen Werte auf die Diskette geschrieben.

Sie können Dateien auch kopieren, indem Sie die zugehörigen Icons auf dem Desktop verschieben, z.B. auf eine andere Diskette oder in einen anderen Ordner (siehe auch Kap. 2). Ab TOS 1.04 können Sie diesen Vorgang zusätzlich jederzeit durch einen Druck auf die -Taste abbrechen.



Besitzt man nur ein Laufwerk, so kann das Kopieren von einer Diskette zur anderen allerdings zur Tortur werden, denn dann müssen Sie pro Datei mindestens viermal die Diskette wechseln. In einem solchen Fall kann ich nur die Anschaffung einer RAM-Disk empfehlen. Mit deren Hilfe entfallen die Diskettenwechsel (die übrigens auch der Mechanik schaden), falls man so vorgeht:

- 1) Originaldiskette einlegen.
- 2) Alle zu übertragenden Dateien auf die RAM-Disk kopieren.
- 3) Diskette wechseln: Zieldiskette einlegen.
- 4) Inhalt der RAM-Disk auf die Zieldiskette kopieren.

Falls die RAM-Disk nicht groß genug sein sollte, um alle Dateien auf einmal aufzunehmen, teilen Sie den Kopiervorgang in mehrere solcher Schritte auf.

Wenn Sie eine Datei in einem anderen Ordner ansiedeln wollen (man nennt das übrigens Verschieben), können Sie das mit dem neuen TOS 1.04 sehr einfach tun. Sie müssen dazu nur das Icon (oder ggf. mehrere) zum gewünschten Verzeichnis schieben und dabei die **Ctrl**-Taste gedrückt halten. In der danach erscheinenden Dialogbox finden Sie dann auch den Hinweis, daß die Dateien verschoben werden. Auch das Verschieben läßt sich durch UNDO abbrechen.

Beim alten TOS bleibt Ihnen nichts anderes übrig, als die Dateien zunächst in den gewünschten Zielordner zu kopieren und danach die Originale zu löschen.

Dateien löscht man, indem man sie in den Papierkorb auf dem Desktop schiebt. Das kann man auch mit ganzen Gruppen von Dateien machen. Wollen Sie alle Dateien einer Diskette löschen, können Sie diese auch formatieren, das geht in der Regel schneller.

Wenn Sie Ihren Computerkünsten (so wie der Verfasser dieser Zeilen) selbst nicht so recht trauen, können Sie eine Datei vor versehentlichem Löschen schützen. Dazu müssen Sie wieder die Funktion "Info anzeigen" aufrufen. Wenn Sie dort den Button für "nur lesen" anklicken, sind alle Schreibzugriffe auf diese Datei gesperrt. Das bedeutet, daß der ST jede Änderung (und damit auch das Löschen) verhindert.

Und wenn es dann doch einmal passiert sein sollte, fragen Sie sich sicher, ob man noch etwas retten kann. Ja und nein, lautet die Antwort des Experten. Ja, weil beim Löschen per Papierkorb die Datei nicht wirklich von der Diskette gelöscht wird, sondern nur die Angaben, auf welchen Oberflächenteilen sie gespeichert ist. Theoretisch ist es möglich, diese Angaben von Hand wieder zusammenzubasteln. Dazu müßten Sie allerdings im schlimmsten Fall über 700 bzw. 1400 Teilstücke der Diskette mit einem speziellen Programm begutachten und darin Ihre Daten wiedererkennen. Sie sehen also, dies ist ein nicht sehr empfehlenswertes Unterfangen. Falls doch einmal Not am Mann ist, sollten Sie die Diskette sofort aus dem Laufwerk nehmen und zu Ihrem Fachhändler bringen. Dieser kann Ihnen, wenn er schon nicht hilft, wenigstens das genannte Spezialprogramm (einen Diskettenmonitor) verkaufen.



Die Anordnung der Dateien auf der Diskette wird durch die sogenannte "File Allocation Table" (zu deutsch etwa "Datei-Anordnungstabelle", kurz FAT) gesteuert. Hier findet sich für jede Datei ein Verweis,

der die dazugehörigen Cluster angibt. Die FAT wird vom ST immer an der gleichen Stelle angelegt, so daß sie selbst auch ohne solche Positionsangaben gefunden werden kann.

Die FAT spielt auch eine wichtige Rolle, wenn man Dateien von der Diskette löscht. Dabei werden nicht, wie Sie jetzt vielleicht vermuten, die zugehörigen Sektoren alle überschrieben. Das würde unverhältnismäßig lange dauern. Im Gegenteil, die Daten-Sektoren bleiben unberührt, es wird nur der Verweis verändert. Beim Löschen einer Datei wird nur der Name aus dem Inhaltsverzeichnis gestrichen und alle Verweise in der FAT auf 0 gesetzt, so daß diese Cluster an andere Dateien vergeben werden können. Gelänge es, alle diese Verweise wieder herzustellen, so könnte die Datei restauriert werden. Leider hat eine zweiseitige Diskette schon 720 Cluster, und es ist schwer zu sagen, in welchem die nächsten Stücke der gesuchten Datei stehen. Das Restaurieren einer gelöschten Datei ist also mit der Suche nach der Nadel im Heuhaufen zu vergleichen.

Zum Öffnen einer Datei brauchen Sie nur einen Doppelklick auszuführen. Bei Daten-Files wird dieses dann angezeigt (bzw. bei angemeldeten Anwendungen ein Programm geladen, daß dieses übernehmen kann), Programm-Files werden ausgeführt.



Oft ist auch von sequentiellen und relativen Dateien die Rede. Damit wird eine spezielle Organisationsform bezeichnet, die vom Desktop aus nicht sichtbar wird. Sequentielle Dateien werden von Programmen so aufgebaut, daß die Daten immer nur der Reihe nach gelesen und bearbeitet werden können. Relative Dateien (auch Random-Access-Dateien) sind so aufgebaut, daß man bei der Bearbeitung einzelne Daten mitten aus ihnen herauspicken kann. Beide Verfahren haben Vor- und Nachteile. Aufgrund spezieller Möglichkeiten ist der ST sogar in der Lage, in gewissen Grenzen die Organisationsform zu übersehen und sequentielle und relative Dateien gleich zu behandeln.

3.1.2 Ordner

Eine Datenstation des ST hat nicht nur das Hauptverzeichnis, das beim Doppelklick auf das Icon im Fenster erscheint. Der Anwender kann eine beliebige Anzahl von Unterverzeichnissen einrichten, in die wiederum neue Unterverzeichnisse eingesetzt werden können usw. Diese Unterverzeichnisse heißen Ordner und werden durch spezielle Icons auf dem Desktop symbolisiert. In einem Fenster wird jeweils nur ein Verzeichnis angezeigt. Welches, das sagt uns die Titelzeile.

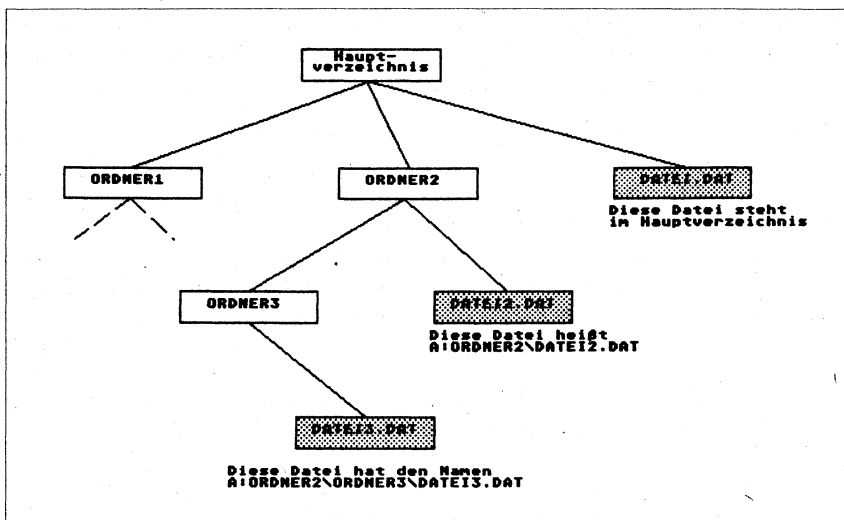


Abb. 3.2: Ein Verzeichnisbaum

Im Laufe der Zeit kann eine Diskette, besonders aber eine Festplatte, sehr viele Dateien aufnehmen (bei Festplatten durchaus 1000 und mehr Files). Würden diese alle in einem Verzeichnis gespeichert, wäre ein hübsches Chaos die Folge. Der Ausweg: Man gruppiert die Dateien und bringt jede dieser Gruppen in einem Ordner unter. Beispiel: Sie arbeiten mit einer Textverarbeitung und einem Grafikprogramm. Da Sie sehr viele Texte schreiben und dazu auch häufig Illustrationen benötigen, ist das Verzeichnis schon recht unübersichtlich geworden. Richten Sie nun zwei neue Ordner ein. Der erste erhält den Namen TEXTE, der zweite wird GRAFIKEN genannt. Klar, was jetzt passiert. In den TEXTE-Ordner legen Sie alle Ihre Textfiles, in den anderen Ordner alle Bilder.

Spinnen wir den Gedanken ein wenig weiter. Sie bearbeiten mehrere Projekte mit jeweils vielen einzelnen Texten. Solange diese Texte im gleichen Verzeichnis stehen, ist auch das sehr unübersichtlich. Erzeugen Sie innerhalb des Ordners TEXTE zwei neue Ordner PROJEKT1 und PROJEKT2. In diese werden dann jeweils die Texte geschoben, die zu dem entsprechenden Projekt gehören. Schon wieder ein wenig Ordnung mehr. Wie Sie sich so etwas bildlich vorstellen müssen, zeigt Abb. 3.2.

Wenn Sie eine solche Vorgehensweise konsequent durchhalten, werden Ihre Disketten immer übersichtlich bleiben. Das Spielchen mit den Ordnern im Ordner können Sie übrigens so lange weitertreiben, wie es Ihnen Spaß macht.

Der ST läßt beliebig komplexe Schachtelungen von Ordnern zu (solange noch Platz auf der Diskette ist).

Natürlich können Sie auch feststellen, wieviele Ordner auf einer Diskette stehen. Die pure Anzahl erfahren Sie durch "Info anzeigen" für die Diskette. Man sieht einem Ordner aber von außen nicht an, wieviele und vor allem welche Unterordner noch darin stecken (und wieviele Unterverzeichnisse die Unterverzeichnisse enthalten und wieviele diese wieder...). Ausnahme: Im neuen TOS wird in der INFO-Anzeige die korrekte Anzahl angegeben. Besitzer von MS-DOS-Computern (den sogenannten Kompatiblen) kennen da ein hilfreiches Programm names TREE, das die gesamte Diskette untersucht und zu jedem gefundenen Ordner ausgibt, wieviele und welche Unterverzeichnisse darin stecken. Festplattenbesitzer schwören auf dieses Tool, denn bei mitunter 100 oder mehr Ordnern verliert man schon einmal den Überblick. Neidisch geworden? Muß nicht sein, denn hier kommt ein TREE-Programm für den ST.

Die Bedienung: Nach dem Start erhalten Sie zunächst eine kleine Anleitung auf dem Bildschirm ausgedruckt. Da es dem Programm egal ist, wo es mit der Suche nach Ordnern beginnt, können Sie hier den Namen des Ordners oder Verzeichnisses eingeben, dessen Unterverzeichnisse angezeigt werden sollen. Für eine Festplatte müssen Sie natürlich die Laufwerksangabe in C: ändern. Falls Sie gar nichts eintippen, so nimmt das Programm automatisch das Hauptverzeichnis von Laufwerk A als Startpunkt an. Nach einem Tastendruck wird dann der Bildschirm gelöscht und die Ordner der Reihe nach angezeigt. Die Anzeige erfolgt dabei so, daß ein Verzeichnis umso weiter eingerückt wird, je tiefer es in der Hierarchie steht.

Hier das Listing für GFA-BASIC

```
' TREE - Programm zum Anzeigen der Directory-Struktur
'
' Alten DTA-Buffer sichern
D%=Gemdos(&H2F)
' Frage nach Laufwerk/Ordner, in dem gesucht werden soll
Print "TREE - ein Programm zum Anzeigen von Verzeichnisbäumen"
Print
Print "Bitte geben Sie an, auf welchem Laufwerk bzw. in welchem Ordner"
Print "die Verzeichnisstruktur sichtbar gemacht werden soll. Beispiele:"
Print "Eingabe 'A:' (Default) zeigt alle Ordner auf Laufwerk A an."
Print "Eingabe 'A:\GFA' zeigt alle Ordner im Ordner GFA auf Disk A an."
Print
Input "Wo soll die Ordnersuche starten";Start$
Print
Print "Auf Tastendruck geht es los!"
Void Inp(2)
If Start$="" Then
  Start$="A:"
```

```

Endif
Cls
Start$=Upper$(Start$)
Print "Ordner in ";Start$
Print
' Alle Ordner durchsuchen
Anz%=0
Gosub Dir_suche(Start$,0)
Print
If Anz%=0 Then
    Print "Keine Unterverzeichnisse in ";Start$;" vorhanden!"
Else
    Print Start$;" enthält ";Anz%;" Unterverzeichnisse."
Endif
Print "Weiter mit Tastendruck."
Void Inp(2)
Edit
' Alte Einstellungen zurück
Void Gemdos(&H1A,L:D%)
' -----
Procedure Dir_suche(F$,Tiefe%)
' Sucht alle Ordner im angegebenen Ordner F$
' und läßt andere gefundene Ordner rekursiv durchsuchen
Local A$,F2$,Dta$,Adr%,A%
' lokalen DTA-Buffer für dieses Verzeichnis reservieren
Dta$=Space$(44)
Adr%=Varptr(Dta$)
Void Gemdos(&H1A,L:Adr%)
' Suche starten
F2$=F$+"\*.*)" +Chr$(0)
A%=Gemdos(&H4E,L:Varptr(F2$),&H10)
' ggf. anzeigen
Gosub Anzeigen
' restliche Files in diesem Ordner untersuchen
While A%=0
    ' nächsten Eintrag suchen
    A%=Gemdos(&H4F)
    ' ggf. anzeigen
    Gosub Anzeigen
Wend
Return
' -----
Procedure Anzeigen
' testet, ob eine Datei ein Ordner ist und gibt ggf. dessen Namen aus
' File-Namen aus Buffer lesen, Buffer gleichzeitig löschen
A$=""
For I=Adr%+30 To Adr%+43
    Exit If Peek(I)<=32
    A$=A$+Chr$(Peek(I))
    Poke I,32
Next I
' Ist die gefundene Datei ein Ordner ?
If (Peek(Adr%+21)=16) And (A$<>"") And (A$<>".") And (A$<>"..") Then

```

```

' nur korrekte File-Namen und Ordner anzeigen
Print Tab(4*Tiefe%);"'-> ";A$
Inc Anz%
' Ordner durchsuchen
Gosub Dir_suche(F$+"\"+A$,Tiefe%+1)
' nach Rückkehr alte DTA wieder herstellen
Void Gemdos(&H1A,L:Adr%)
Endif
Return

```

Hier die Version für ST-BASIC

```

0 ' TREE - Programm zum Anzeigen der Directory-Struktur
1 '
2 ' Frage nach Laufwerk/Ordner, in dem gesucht werden soll
3 PRINT "TREE - ein Programm zum Anzeigen von Verzeichnisbäumen"
4 PRINT
5 PRINT "Bitte geben Sie an, auf welchem Laufwerk bzw. in welchem Ordner"
6 PRINT "die Verzeichnisstruktur sichtbar gemacht werden soll. Beispiele:"
7 PRINT "Eingabe 'A:' (Default) zeigt alle Ordner auf Laufwerk A an."
8 PRINT "Eingabe 'A:\OMI' zeigt alle Ordner im Ordner OMI auf Disk A an."
9 PRINT
10 INPUT "Wo soll die Ordnersuche starten ? ";Start$
11 PRINT
12 PRINT "Auf Tastendruck geht es los!"
13 A$= INPUT$(1)
14 IF Start$="" THEN
15   Start$="A:"
16 ENDIF
17 CLS
18 Start$= UPPER$(Start$)
19 PRINT "Ordner in ";Start$
20 PRINT
21 ' Alle Ordner durchsuchen
22 Z%=0
23 Dirsuche(Start$,0)
24 PRINT
25 IF Z%=0 THEN
26   PRINT "Keine Unterverzeichnisse in ";Start$;" vorhanden!"
27 ELSE
28   PRINT Start$;" enthält ";Z%;" Unterverzeichnisse."
29 ENDIF
30 PRINT "Weiter mit Tastendruck."
31 A$= INPUT$(1)
32 END
33 ' -----
34 DEF PROC Dirsuche(F$,Tiefe%)
35 ' Sucht alle Ordner im angegebenen Ordner F$
36 ' und läßt andere gefundene Ordner rekursiv durchsuchen
37 LOCAL A$,Dta$,Adr$L,Anz%,R1$,R2$
38 ' lokale Inhaltsverzeichnisdatei öffnen
39 OPEN "F",Tiefe%+1,F$+"*. **",16

```

```

40 REPEAT
41   GET Tiefe%+1,1
42   FIELD Tiefe%+1,21 AS R1$,1 AS A$,8 AS R2$,14 AS Dta$
43   Anz%=Anz%+1
44   IF NOT EOF(Tiefe%+1) THEN
45     N$= LEFT$(Dta$, MAX( INSTR(Dta$, CHR$(0))-1,0))
46     ' Ist die gefundene Datei ein Ordner ?
47     IF ( BIT(4, ASC(A$))) AND (N$<>".") AND (N$<>"..") THEN
48       ' nur korrekte File-Namen und Ordner anzeigen
49       PRINT TAB (4*Tiefe%); "'-> ";N$
50       Z%=Z%+1
51       ' Ordner durchsuchen
52       Dirsuche(F$+"\"+N$,Tiefe%+1)
53     ENDIF
54   ENDIF
55 UNTIL EOF(Tiefe%+1)
56 CLOSE Tiefe%+1
57 RETURN

```

Neue Ordner werden über das Desktop erzeugt (Menüpunkt "neuer Ordner"). Gelöscht werden Ordner wie Dateien auch - man schiebt sie in den Papierkorb. Aber Vorsicht. Auch alle in dem Ordner vorhandenen Dateien werden dabei gelöscht!

Sie können einen Ordner aber nicht wie eine normale Datei vor dem Überschreiben schützen - das wäre auch nicht sinnvoll, da Ordner selbst keine Daten enthalten, sondern nur die in ihnen gespeicherten Dateien.

Das Umbenennen von Ordnern ist ein Problem - oder auch nicht. Im neuen TOS 1.04 ist das Problem beseitigt worden. Sie können genau wie beim Umbenennen einer Datei vorgehen. Im alten TOS verhalten sich die Ordner anders: Sie wehren sich mit Händen und Füßen dagegen, umbenannt zu werden. Zum Glück gibt es einen einfachen Weg, dies zu umgehen. Will man einen Ordner umbenennen, so kann man so vorgehen:

- 1) Neuen Ordner mit dem gewünschten Namen auf der Diskette erzeugen.
- 2) Alle Dateien aus dem alten in den neuen Ordner kopieren. (Die Dateien existieren jetzt in beiden Ordnern.).
- 3) Alten Ordner löschen.

Probleme können auftauchen, wenn der zu kopierende Ordner, welcher umbenannt werden soll, mehr Bytes enthält als noch auf der Diskette oder der Festplatte frei sind. Diesen zusätzlichen Platz brauchen wir zum Kopieren. Sollte Ihr ST bei einer solchen Umbenennungsaktion die Fehlermeldung "Diskette voll" ausgeben, müssen Sie etwas anders vorgehen:

- 1) RAM-Disk mit mindestens der Größe des Ordnerinhalts einrichten.
- 2) Neuen Ordner mit dem gewünschten Namen auf der Diskette erzeugen.
- 3) Alle Dateien aus dem alten Ordner in die RAM-Disk kopieren.
- 4) Alten Ordner löschen.
- 5) Dateien aus der RAM-Disk in den neuen Ordner kopieren.

Dieses etwas umständlich anmutende Verfahren benutzt die RAM-Disk als Zwischenlager und vermeidet so die Überlastung der Diskette. Allerdings birgt es gewisse Gefahren in sich. Nachdem Sie Schritt 4) ausgeführt haben, existieren Ihre Daten nur noch auf der RAM-Disk. Falls nun der Strom ausfällt, bevor Sie die Dateien auf die Diskette zurückkopieren konnten, sind diese unwiederbringlich verloren. Allerdings ist das Risiko eines Stromausfalls recht gering, es sei denn, Ihre kleinen Kinder spielen zu Ihren Füßen mit den Netzteilen... Wenn Sie ganz sicher gehen wollen, können Sie statt einer RAM-Disk eine richtige Diskette als Zwischenlager benutzen. Das dauert aber sehr viel länger und ist insbesondere bei nur einem Laufwerk extrem umständlich.

In der einschlägigen Fachpresse haben Sie vielleicht schon vom 40-Ordner-Problem gehört. Bedingt durch einen Fehler im Betriebssystem kann es vorkommen, daß bei der Verwendung von mehr als 40 bzw. 100 Ordnern (je nach Version) die interne Speicherverwaltung des Atari durcheinandergerät. Das macht sich durch völlig unverständliche Fehlermeldungen oder andere seltsame Erscheinungen bemerkbar. Glücklicherweise tritt dieses Problem in der Regel nur bei Festplatten auf. Als Abhilfe liefert Atari zu jeder Festplatte ein Programm namens FOLDR100.PRG mit, das im AUTO-Ordner bei jedem Booten gestartet werden sollte. Es richtet Platz für zusätzliche 100 Ordner ein, womit die Probleme vorläufig beseitigt sein dürften. Wollen Sie mehr als 140 Ordner benutzen, können Sie den neugeschaffenen Platz erweitern, wenn Sie den Programmnamen ändern (Menüpunkt "Info"). Die Zahl hinter dem Wort FOLDR gibt an, für wieviele zusätzliche Ordner Speicher bereitgestellt wird. Wählen Sie also als neuen Namen FOLDR200.PRG, so wird für 200 zusätzliche Ordner reserviert.

3.1.3 Datei- und Pfadnamen

Wie selbstverständlich sind wir bisher davon ausgegangen, daß Dateien und Ordner Namen haben. Aber da steckt ein wenig mehr dahinter. Es ist zu unterscheiden zwischen dem eigentlichen File- oder Dateinamen und dem Pfad. Beide zusammen ergeben den Pfadnamen. Allerdings sind diese Bezeichnungen nicht einheitlich; vielfach werden die Begriffe "Pfadname" und "Dateiname" synonym gebraucht. Der Dateiname besteht aus 1 bis 8 Zeichen,

hinzu kommt eine sogenannte Extension (oder auch Endung, Dateityp) von maximal drei Zeichen, die durch einen Punkt angehängt wird. Beispiele für Dateinamen sind:

```
XXXXXXXX.YYY  
TESTTEXT.TXT  
PROGRAMM  
A
```

Innerhalb des Namens und der Extension dürfen Sie fast alle Zeichen benutzen. Verboten sind Leerzeichen und alle unten aufgeführten Zeichen:

\ { } [] () < > = * & , . ; : - + ! ? / ~ | " % @

Einige dieser Zeichen haben außerdem noch andere Bedeutung. Außerdem werden Kleinbuchstaben automatisch in Großbuchstaben umgewandelt.



Als Ersatz für das Leerzeichen können Sie den Unterstrich verwenden (Aufruf durch **Shift** + **-**).

Obwohl Dateinamen zum Auffinden einer Datei innerhalb eines Verzeichnisses dienen und deshalb dort nur einmal vorkommen dürfen, kann der gleiche Namen in verschiedenen Ordnern durchaus mehrfach auftauchen. Zur Unterscheidung dieser Fälle dient der Pfad, der dem Betriebssystem angibt, in welchem Ordner es nach der gewünschten Datei suchen soll. Der Pfadname beginnt mit dem Kennbuchstaben des angesprochenen Laufwerks, gefolgt von einem Doppelpunkt und einem Backslash. Liegt die gesuchte Datei im Hauptverzeichnis einer Diskette, so ist der Pfad damit komplett. Ein Beispiel:

A:\TEST.PRГ

bezeichnet die Datei TEST.PRГ im Hauptverzeichnis der Diskette in Laufwerk A. Dagegen greifen Sie mit dem Namen

A:TEST.PRГ

auf die Datei TEST.PRГ im aktuellen Verzeichnis (das kann auch ein Ordner sein) zu. Wenn Sie mit der Objektauswahl arbeiten, haben Sie diese Probleme nicht. Hier werden immer Pfade angegeben, die im Hauptverzeichnis beginnen. Einige Anwendungen verlangen aber eine Dateinameneingabe, die nicht über Objektauswahlen läuft. Dann sollten Sie diese Regeln beachten. Insbesondere sollten Sie sich merken, in welchem Ordner Sie zuletzt gearbeitet haben, denn dies ist der aktuelle Ordner.

Liegt eine Datei in einem Ordner, so muß dessen Name an den Pfad angehängt werden. Sind mehrere Ordner ineinander verschachtelt, so müssen alle zum Auffinden nötigen Ordnernamen angegeben werden. Ein Beispiel:

A:\ORDNER.1\ORDNER\TEST.PRG

besagt, daß die Datei TEST.PRG im Ordner "ORDNER" zu finden ist, der seinerseits im Ordner mit der Bezeichnung "ORDNER.1" steht. Etwas anderes besagt dieser Pfad, wenn Sie den ersten Backslash weglassen. Dann "beginnt" der Pfad im aktuellen Verzeichnis und führt von dort aus weiter. Ein Beispiel: Wir befinden uns im Ordner ORDNER.1 und geben dann den Pfadnamen A:ORDNER\TEST.PRG ein. In diesem Fall wird das Programm-File im gleichen Ordner gesucht wie im vorhergehenden Beispiel. Befinden wir uns aber im Hauptverzeichnis der Diskette, dann hätte der Pfad A:ORDNER\TEST.PRG zur Folge, daß TEST.PRG in dem Ordner ORDNER gesucht wird, der im Hauptverzeichnis steht.

Ordnernamen werden genau wie Dateinamen aus bis zu acht Zeichen und nach Wunsch mit bis zu drei Zeichen Extension gebildet. Im Pfad werden die Ordner- und Dateinamen dann durch den sogenannten Backslash getrennt. (Das ist der Schrägstrich, der von links oben nach rechts unten verläuft.). Man erreicht den Backslash auf der Tastatur durch die Kombination Alternate + Shift + ü.

Nur die in der oben angegebenen Weise gebildeten Pfadnamen sind korrekt, jedoch ist das Betriebssystem so fehlertolerant, daß einige Teile mitunter weggelassen werden können. Gibt man zum Beispiel gar keinen Pfad an, so nimmt der Rechner statt dessen das zuletzt benutzte Verzeichnis an. Läßt man die Laufwerkskennung weg, sucht das Betriebssystem auf dem zuletzt benutzten Laufwerk nach dem Dateinamen.

Durch die Benutzung sogenannter Joker kann man mit einem Pfadnamen auch mehrere Dateien gleichzeitig bezeichnen. Dabei steht der * für eine bis zu 8 Zeichen lange Folge beliebiger Zeichen und das ? für ein beliebiges Zeichen. Beispiele:

- | | |
|----------|---|
| ABC*.TXT | bezeichnet alle Dateien, die mit ABC beginnen und die die Extension TXT besitzen. |
| *ABC.TXT | steht für alle Dateien, die mit ABC enden und die die Extension TXT besitzen. |
| ** | erreicht alle Files im gewählten Verzeichnis. |

- ABC.* bezeichnet alle Dateien, die als Namen ABC plus einer beliebigen Extension tragen.
- ABC?.TXT steht für alle Dateien mit der Extension TXT, deren Name mit ABC beginnt und von einem beliebigen Zeichen gefolgt wird.
- ABC?EFGH.TXT bezeichnet alle Text-Files, die mit ABC beginnen, auf das ein beliebiger Buchstabe und schließlich EFGH.TXT folgt.

Selbstverständlich lassen sich die beiden Joker-Zeichen auch kombiniert einsetzen. Die Benutzung dieser Joker ist nicht immer möglich. Ein Beispiel für die sinnvolle Nutzung ist die Objektauswahl, wo die obere Eingabezeile einen solchen Pfadnamen beinhaltet, mit dessen Hilfe die anzeigenswerten Dateien im darunterstehenden Fenster aussortiert werden. Jokerzeichen dürfen aber niemals im Pfad selbst auftauchen, sondern dürfen immer nur im Dateinamen stehen (also ganz am Ende des Pfadnamens). Es ist also nicht möglich, mittels der Jokerzeichen mehrere Ordner gleichzeitig anzuwählen.

3.1.4 Extensions

Die Endung (oder Extension) des File-Namens dient dazu, einzelne Dateitypen voneinander zu unterscheiden. Prinzipiell dürfen Sie jede beliebige zulässige Extension vergeben, einige werden allerdings schon vom Betriebssystem für eigene Zwecke benutzt.

- PRG wird für Programmdateien benutzt, die GEM-Programme enthalten. PRG-Dateien werden grundsätzlich als ausführbar betrachtet, auch wenn darin vielleicht gar nichts gespeichert ist. Der Aufruf solcher unechten Programme führt mit fast hundertprozentiger Sicherheit zum Absturz des Rechners.
- TOS steht für Programmdateien, die nicht auf die Fähigkeiten des GEM zurückgreifen. In solchen Programmen gibt es normalerweise keine Menüleisten und Dialogboxen.
- TTP steht für TOS-Programme, die vor dem Start noch gewisse Angaben von Ihnen brauchen. Dazu erscheint direkt nach dem Doppelklick auf dem Bildschirm eine Dialogbox, in der eine Eingabezeile für Ihre An-

gaben ausgefüllt werden muß. Was Sie dort hinschreiben müssen, hängt vom jeweiligen Programm ab. Sehen Sie bitte in der Anleitung der Software nach.

- RSC** ist die Abkürzung für Resource. Solche Dateien gehören zu den PRG-Programmen. Sie enthalten Daten für die Gestaltung von Menüleisten und Dialogboxen. Sie dürfen diese Dateien nicht löschen. Sollten Sie ein PRG-Programm kopieren wollen, achten Sie bitte darauf, daß die zugehörige RSC-Datei mitkopiert wird. Manche Programme haben mehrere dieser RSC-Dateien, einige kommen ohne aus. Im Zweifel gibt das Handbuch des Programms hierüber Auskunft.
- BAK** wird für Sicherheitskopien verwendet. Diese Sicherheitskopien werden von vielen Programmen automatisch erzeugt, wenn Sie alte Daten ändern. In einem solchen Fall wird die Extension des alten Files in BAK geändert und die neue Version unter dem bisherigen Namen gespeichert. Auf diese Weise haben Sie immer den letzten Stand Ihrer Arbeit als Sicherheit mitgespeichert. Sollte die neue Version defekt sein, oder haben Sie einen Fehler gemacht, können Sie die Sicherheitskopie wieder umbenennen (z.B. von xxxxxxxx.BAK in xxxxxxxx.TXT) und erneut benutzen. Sollte Ihre Diskette einmal zu voll sein, um neue Daten aufzunehmen, können Sie sich in vielen Fällen helfen, wenn Sie nicht mehr benötigte BAK-Dateien löschen.
- APP** bezeichnet spezielle Arten von GEM-Programmen, die von anderen Programmen nachgeladen werden. In den meisten Fällen können diese Programme auch ohne Schwierigkeiten in PRG-Versionen umbenannt werden und dann selbständig laufen.
- \$\$\$** ist eine Extension, die Sie als Anwender normalerweise nicht vergeben können. Einige Programme benutzen sie, um temporäre Dateien zu erzeugen, die nach Benutzung wieder gelöscht werden. Das Auftauchen solcher Dateien auf Ihren Disketten ist in den meisten Fällen ein sicheres Zeichen für einen Fehler, z.B. war der Schreibschutz aktiv, was bewirkt, daß das Programm seine Dateien nicht wieder löschen konnte.
- TXT** wird von den meisten Textverarbeitungen benutzt, um Text-Files zu kennzeichnen.

- BAS** steht für ein BASIC-Programm. Leider benutzen fast alle BASIC-Interpreter die gleiche Extension, so daß Sie nicht feststellen können, für welche Sprache dieses Programm geschrieben wurde.
- DOC** ist ebenfalls eine Extension für Dateien aus Textverarbeitungen.
- INF** wird für Informations-Files benutzt, die Angaben über bestimmte Zustände eines Programms oder des Desktop (DESKTOP.INF) enthalten.
- ACC** ist die Extension, mit der Accessories markiert werden. Nach dem Einschalten sucht der ST nach den ersten 6 Dateien, die diese Extension tragen und versucht sie als Accessories zu installieren. Sie sollten diese Extension niemals für Ihre eigenen Daten vergeben, um Abstürze zu vermeiden. Es kann vorkommen, daß auch zu Accessories ein Resource-File (Extension RSC) gehört.
- SYS** wird bei Dateien verwendet, die wichtige Systemdaten oder Treiber für das GEM enthalten. Beispiele sind ASSIGN.SYS (Zuordnungstabelle für Zeichensätze) und META.SYS (virtueller Gerätetreiber).
- GEM** steht für sogenannte Meta-Files. Dahinter verbirgt sich ein universelles Datenformat für Grafiken, das den Austausch von Bildern zwischen verschiedenen Rechnern und Programmen ermöglichen soll.
- OPT** ist die Kurzform von OPTION. Solche Dateien dienen ähnlichen Zwecken wie die INF-Files.
- FNT** wird für Zeichensatzdateien benutzt.
- C** wird normalerweise an Quelltexte für C-Programme vergeben.
- H** findet für Hilfsdateien in C-Compilern Verwendung.
- O** steht für Objekt-Files, das sind compilierte, aber nicht gelinkte Dateien.
- PAS** dient als Kennzeichen für PASCAL-Programmtexte.
- S** ist die Extension für Assembler-Quelltexte.

PIC wird für pixel-orientierte Grafikdateien benutzt.

NEO wird für Farbgrafiken eingesetzt.

PI3 ebenfalls für Grafikdateien (ebenso PI1 und PI2).

3.2 Die Diskette

Diskette ist nicht gleich Diskette. Zunächst einmal ist die Größe wichtig. Sie wird am Durchmesser der eingebauten Scheibe bemessen, dieser beträgt 3,5 Zoll. Es gibt auch andere Diskettengrößen. Die bekanntesten sind wohl die 5,25-Zoll-Disketten, die früher sehr verbreitet waren und nun langsam aber sicher von der 3,5-Zoll-Diskette verdrängt werden. Außerdem drucken die Hersteller noch diverse Kapazitätsangaben auf die Disketten, da die Speichermöglichkeiten nicht durch die Größe der Scheibe, sondern die Güte der Magnetschicht bestimmt werden.

1SSD oder SSSD

steht für "Single Sided, Single Density". Diese Qualität wird für 3,5-Zoll-Disketten eigentlich nicht angeboten, da sie keine ausreichende Kapazität ermöglicht.

1SDD oder SSDD

bedeutet "Single Sided, Double Density". Mit anderen Worten: Die Diskette kann einseitig genutzt werden und bietet die doppelte Schreibdichte. Dies ist die richtige Diskettensorte, wenn Sie ein einseitiges Laufwerk SF354 besitzen. Sie können solche Disketten zwar auch mit anderen Laufwerken benutzen, dann besteht aber die Gefahr des Datenverlustes, weil der Hersteller nur für eine Seite die erforderliche Sicherheit garantiert.

2SDD oder DSDD

ist die Abkürzung für "Double Sided, Double Density". Diese Diskette hat ebenfalls doppelte Schreibdichte, kann aber zwei Seiten nutzen. Sie sollten diese kaufen, falls Sie einen Mega-ST, einen 1040 ST oder ein Laufwerk SF314 besitzen. Diese Geräte beschreiben beide Seiten der Magnetscheibe.

2SHD oder DSHD

Dies sind Disketten des Typs "High Density", also mit hoher Schreibdichte. Beim Kauf dieser Disketten werfen Sie Geld zum Fenster hinaus, denn der ST kann die hohe Qualität dieser Scheiben gar nicht ausnutzen.



Auch wenn eine Diskette von außen nicht so aussieht, handelt es sich dabei um ein rundes Gebilde, das von einem Motor recht schnell gedreht wird. Zum Vergleich: Eine Schallplatte dreht sich mit 33,3 oder 45 Umdrehungen pro Minute, die runden Magnetscheiben in der Diskettenhülle dagegen mit 300 U/min. Festplatten sind noch schneller (meistens 3600 U/min). Diese hohen Geschwindigkeiten sind der Grund dafür, daß sowohl Festplatten als auch normale Diskettenlaufwerke recht lange Beschleunigungs- und Bremszeiten haben. Vielleicht ist Ihnen schon aufgefallen, daß Ihr Diskettenlaufwerk noch ein bißchen nachläuft, während sich der Computer schon alle Daten geholt hat und längst mit etwas anderem beschäftigt ist. Dabei handelt es sich um einen Trick, der Zeit sparen soll. Wie gesagt braucht der Motor einige Zeit, um eine konstante Geschwindigkeit zu erreichen. Daher schaltet sich der Motor nach der Übertragung der letzten Daten nicht sofort ab. Falls jetzt erneut auf die Diskette zugegriffen werden muß, hat die Scheibe noch die korrekte Geschwindigkeit und muß nicht wieder beschleunigt werden. Festplatten haben dieses Prinzip noch ausgeweitet, sie laufen die gesamte Zeit über mit voller Geschwindigkeit.

Wie kommen nun die Daten auf die Scheibe? Sie kennen sicher das Aufzeichnungsprinzip eines Tonbandes: Töne (Daten) werden als magnetische Spuren auf eine Eisenoxid- oder Chromdioxid-Schicht gebracht. Stellen Sie sich nun eine Schallplatte vor, die mit einer solchen Schicht bedeckt ist. Statt der Nadel führt man nun einen Tonkopf über die Platte. Das wäre schon fast das gesamte Funktionsprinzip von Disketten und Festplatten. Die Unterschiede liegen im Detail. Disketten werden (vor allem in Amerika) auch Floppy Disks genannt. Floppy heißt so viel wie "weich" oder "biegsam". Das bezieht sich auf die Scheibe, die im Innern des Gehäuses liegt. Sie besteht aus weichem, biegsamen Kunststoff und hat einen Durchmesser von 3,5 Zoll (siehe Abb. 3.3). Bestimmt kennen Sie auch die größeren Exemplare mit 5,25 Zoll Durchmesser, bei denen auch die Hülle biegsam ist. Von dieser Flexibilität haben die Disketten ihren Namen erhalten. Allerdings ist das alles andere als ein Vorteil. Flexible Disketten lassen sich schön knicken - und genau das ist das schlimmste, was man mit einer solchen Scheibe machen kann. Im Normalfall sind dann alle Daten verloren, weil der Lesevorgang durch den Knick (wie durch einen Kratzer auf einer Schallplatte) behindert oder gar unmöglich gemacht wird. Deshalb ist man bei den neueren 3,5-Zoll-Disketten dazu übergegangen, ein hartes Gehäuse drumherum zu bauen. Diese Hülle verträgt es auch einmal, in die Hosentasche gesteckt zu werden.

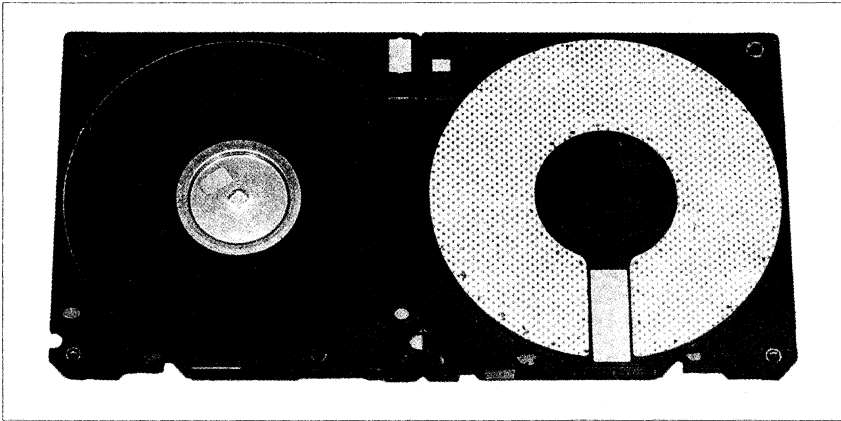


Abb. 3.3: Das Innere einer 3,5-Zoll-Diskette

(Bitte machen Sie das nicht nach, es sei denn, Sie brauchen die Diskette garantiert nie mehr!)



Der den meisten Disketten beiliegende Aufkleber hat die unangenehme Eigenschaft, nicht immer dauerhaft zu kleben. Ein kleiner Trick schafft hier meist Abhilfe. Streichen Sie die reservierte Fläche mit einem Papierklebestift ein. Darauf reiben Sie den widerspenstigen Aufkleber mit einem weichen Tuch fest (bitte nicht mit Gewalt). So befestigte Beschriftungen haben sich als recht dauerhaft erwiesen.



Beschriften Sie die Aufkleber mit wasserfesten Faserschreibern. Bedingt durch die Konstruktion der Disketten wird die Beschriftung beim Einlegen der Diskette besonders häufig mit den Fingern berührt. Die natürliche Hautfeuchtigkeit sorgt dann bei normaler Beschriftung schnell für ein Verwischen der Beschriftung.

Jede Diskette hat einen kleinen Schieber in einer Ecke. Das ist der sogenannte Schreibschutzschieber. Ist er so eingestellt, daß das Loch verdeckt ist, so kann die Diskette beschrieben und gelesen werden. Ist der Schieber in der anderen Position (Sie können dann durch das Loch hindurchsehen), so verweigert der ST jedes Schreiben auf dieser Diskette. Damit haben Sie einen gewissen Schutz vor dem Verändern oder Löschen Ihrer Daten.

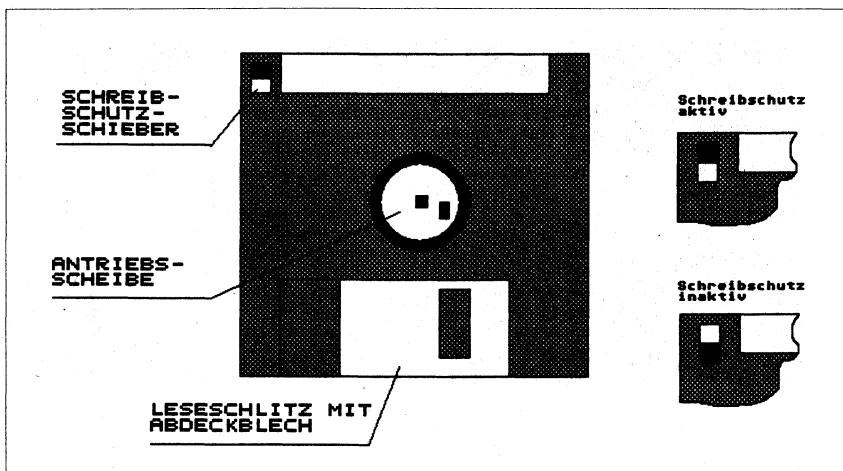


Abb. 3.4: Der Schreibschutz

Beim Formatieren etwa oder in vielen Software-Beschreibungen wird zwischen dem sogenannten ein- und zweiseitigen Format unterschieden. Damit hat es folgende Bewandnis. Das Diskettenlaufwerk SF354 besitzt nur einen Schreib-/Lesekopf und kann deshalb nur eine Seite der Magnetscheibe bearbeiten. Alle anderen ST-Laufwerke verfügen über zwei dieser Schreib-/Leseköpfe und arbeiten daher auf zwei Seiten. Das bringt natürlich auch eine Verdopplung der Speicherkapazität mit sich. Dafür sind SF354-Laufwerke ein wenig billiger (trotzdem werden sie kaum noch gekauft).

Leider kann man die beiden Formate nur bedingt austauschen. Das einseitige Laufwerk kann nur einseitige Disketten bearbeiten. Umgekehrt können zweiseitige Laufwerke aber einseitige Disketten lesen und beschreiben. Sie können mit einem zweiseitigen Laufwerk sogar einseitige Disketten formatieren, so daß Sie mit Besitzern von SF354-Laufwerken Daten austauschen können.

Zweiseitige Disketten melden nach dem Formatieren einen freien Bereich von 726016 Bytes, das sind 709 KByte. Die Differenz zu den immer wieder genannten 720 KByte erklärt sich dadurch, daß das Inhaltsverzeichnis einer jeden Diskette auch Platz benötigt.

Einseitige Disketten haben Platz für 357376 Bytes, was 349 KByte entspricht.

Es kann aber auch sein, daß die Diskette nach dem Formatieren weniger Platz hat als üblicherweise vorgesehen. Mitunter kommt es vor, daß einzelne Teile

einer Diskettenoberfläche defekt sind, z.B. durch winzige Kratzer. Diese eignen sich dann nicht mehr für die Datenspeicherung. Der ST erkennt dies beim Formatieren und schließt diese defekten Oberflächenstücke von der Benutzung aus. Sie können solche Disketten unter Kapazitätsverlust weiterverwenden.

Wer nicht nur mit dem Atari arbeitet, fragt sich sicher, ob man die Disketten auch in anderen Rechnern verwenden kann. In den meisten Fällen wird das scheitern. So benutzt der Amiga ein ganz anderes Diskettenformat als der ST, und PCs, die zwar auch mit 3,5-Zoll-Disketten arbeiten, können mit den ST-Disketten nicht immer etwas anfangen. Im Normalfall sollte aber der ST mit jeder Diskette umgehen können, die von einem PC formatiert worden ist, so daß Sie solche Scheiben zum Datenaustausch benutzen können.

Wenn Sie das neue TOS 1.04 verwenden, können Sie Ihre Disketten auch von den meisten PC's mit 3,5-Zoll-Laufwerk bearbeiten lassen.

Pannenhilfe:



Beim Beschreiben oder Lesen einer 3,5-Zoll-Diskette, die von einem PC formatiert wurde, gibt der ST Fehlermeldungen aus wie "Daten auf Disk. A: defekt ...".



Wahrscheinlich ist eines der beiden Laufwerke (entweder das des ST oder des PC) defekt, so daß beim Formatieren leichte Abweichungen entstehen, die der jeweils andere Rechner nicht mehr auffangen kann. Hier hilft nur, die beiden Geräte überprüfen zu lassen und ggf. zu reparieren. In ganz seltenen Fällen kann es auch vorkommen, daß der ST das MS-DOS-Format nicht richtig erkennt. In diesem Fall sollten Sie es mit einer anderen Diskette, die unter einer anderen MS-DOS-Version formatiert wurde, erneut versuchen.

Jede Diskette muß vor der ersten Benutzung formatiert werden, d.h. es werden magnetische Markierungen auf der Diskette angelegt, anhand derer der Rechner später die Daten wiederfinden kann. Da fast jeder Rechnerhersteller ein eigenes Format für diese Markierungen hat, kann der Formatiervorgang nicht vom Hersteller der Diskette übernommen werden. Sie können das erledigen, indem Sie den Menüpunkt "formatiere..." auf dem Desktop anwählen. In letzter Zeit gibt es auch fertig formatierte Disketten zu kaufen. Damit möchte der Hersteller Ihnen Arbeit abnehmen. Sofern diese Disketten für MS-DOS formatiert sind, müßte der ST problemlos mit ihnen zurechtkommen. Hier geht probieren über studieren.



Wenn Sie eine Diskette formatieren, die bereits Daten enthält, so gehen diese unwiederbringlich verloren!



Sie können die obige Falle auch ausnutzen, wenn Sie ganz sicher sind, daß Sie eine Diskette löschen wollen. Dann formatieren Sie sie einfach.



Immer wieder werden auch Programme angeboten, die beim Formatieren auf den Disketten ein leicht verändertes Format aufbringen und so die Kapazität steigern sollen. Das funktioniert zwar, hat aber einen gefährlichen Haken. Eine normale Diskette hat 80 Spuren, auf denen die Daten untergebracht werden. Die meisten Laufwerke sind aber in der Lage, mehr als 80 (bis zu 83 Spuren) zu bearbeiten. Die Kapazitätssteigerung wird nun dadurch erreicht, daß eben diese zusätzlichen Spuren mitformatiert werden. Wenn Ihr Laufwerk das schafft, haben Sie Glück gehabt, ansonsten kann es auch sein, daß der Schreib-/Lese-Kopf ab Spur 82 gegen eine mechanische Sperre läuft und blockiert wird. Dies ist für die Mechanik äußerst ungesund. Schon nach wenigen Versuchen kann auf diese Weise ein Laufwerk dejustiert werden (d.h. die Köpfe geraten aus der optimalen Bahn), was sich durch Lese- und Schreibfehler bemerkbar machen kann. Eine neue Justierung beim Händler ist unvermeidbar. Deshalb der Rat: Formatieren Sie immer nur vom Desktop aus oder mit Programmen, die nicht mehr als 80 Spuren nutzen.



Warum müssen Disketten vor der ersten Benutzung formatiert werden? Vielleicht haben Sie sich darüber auch schon gewundert, weil Sie doch Ihre Tonband-Cassetten direkt benutzen können. Was passiert hier?

Um den Sinn der Formatierung zu verstehen, muß man wissen, wie die Daten auf der Platte angeordnet werden. Eine Schallplatte hat genau zwei spiralförmige Rillen. Auf einer Diskette oder Festplatte werden die Daten aber nicht in Spiralen, sondern in konzentrischen Ringen untergebracht.

Jede dieser Spuren ist dann noch einmal in Sektoren aufgeteilt, die jeweils 512 Bytes an Daten enthalten. Dahinter steht folgendes Problem: Wenn Sie auf einer Schallplatte eine bestimmte Stelle suchen, sind Sie auf die gute alte Try-and-error-Methode angewiesen, es sei denn, Sie können die gewünschten Stelle an der Form der Rillen erkennen. Dieses System ist nun überhaupt nicht für Computer geeignet, denn er kennt die gewünschten Daten ja im Voraus nicht. Teilt man dagegen die Daten in Spuren und Sektoren auf, so kann man sie adressieren, z.B. durch Spur 5, Sektor 7.

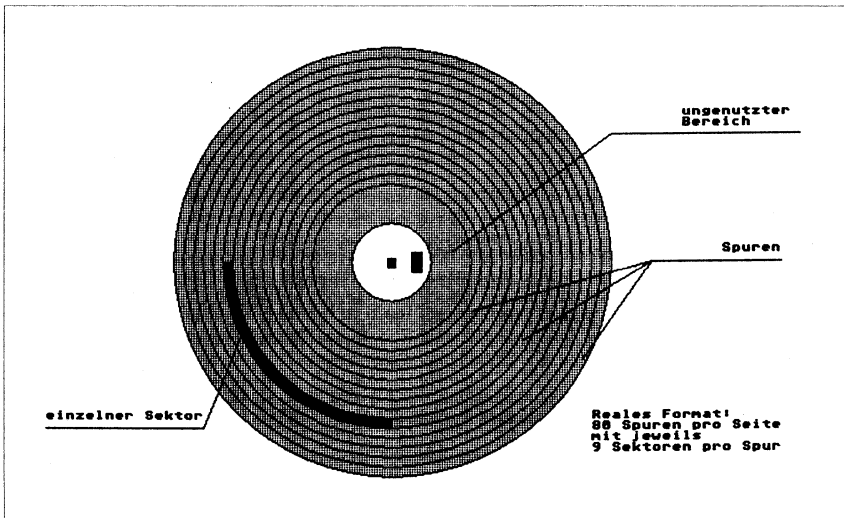


Abb. 3.5: Aufteilung einer Diskette

Solche Spuren und Sektoren werden nun nicht bei der Herstellung auf die Diskette gebracht, weil viele Computer unterschiedliche Einteilungen vornehmen. Daher müssen Sie die Diskette vor der ersten Benutzung formatieren. Dabei geschieht nichts anderes, als daß der ST 80 solcher Spuren und darin jeweils 9 (leere) Sektoren pro Spur anlegt. Dazu ein kleines Rechenbeispiel:

$$80 \text{ Spuren} * 9 \text{ Sektoren} * 512 \text{ Bytes} = 360 \text{ KByte}$$

Das ist - oh Wunder - genau die Kapazität einer einseitigen Diskette. Für Diskettenlaufwerke, die beide Seiten einer Diskette beschreiben können, verdoppelt sich natürlich diese Zahl.

Übrigens müssen auch Festplatten vor der ersten Benutzung formatiert werden, weil auch sie die Daten in solchen Sektoren anlegen. Wegen der größeren Genauigkeit können diese aber mehr Spuren und auch mehr Sektoren unterbringen.

Wie Dateien kann man auch ganze Disketten kopieren. Vorausgesetzt, daß es sich um zwei Disketten mit identischem Format handelt, können Sie das Icon der einen Diskette auf das der anderen schieben. Dann werden die Disketten 1:1 kopiert, d.h. hinterher haben beide Scheiben absolut identischen Inhalt (näheres siehe auch unter "Desktop"). Haben Sie zwei Disketten mit unter-

schiedlichen Formaten, so verweigert der ST die 1:1-Kopie, und Sie müssen alle Dateien "von Hand" auf die andere Diskette schieben.

Näheres über eine Diskette erfahren Sie, wenn Sie sich die Info zu einer Diskette anzeigen lassen. Darin erfahren Sie nicht nur den Namen und die Anzahl der Ordner und Dateien. Es werden auch noch zwei Zahlenwerte angegeben: die belegten und die noch freien Bytes. Wenn Sie die Kapazität einer Diskette ermitteln wollen, addieren Sie diese beiden Werte. Ergibt sich ein Wert von ca. 700000 Bytes, so haben Sie eine zweiseitige Diskette vor sich, bei ca. 350000 Bytes handelt es sich um eine einseitige. Aus technischen Gründen (siehe Kapitel "Größenangaben auf Massenspeichern") wird hier aber nur selten die wirkliche Kapazität von 726016 bzw. 357376 Bytes herauskommen.

Auch Disketten haben, wie Sie an der Info-Meldung sehen, einen Namen. Das Desktop läßt eine Änderung des Diskettennamens (der Fachausdruck lautet Volume-Label) jedoch nicht zu. Dennoch kann man in einigen Fällen mit einem kleinen Programm diesen Namen ändern. Er besteht - wie ein normaler Dateiname auch - aus 8 Zeichen plus Extension.

Hier das Programm für GFA-BASIC:

```
' DISKNAME - Programm zum Umbenennen einer Diskette
' Version für GFA-BASIC 2.0
'
Print "DISKNAME"
Print "Dieses Programm ermöglicht das Ändern des Namens der"
Print "Diskette in Laufwerk A. Legen Sie bitte die Diskette ein."
Print "Fertig -> Taste drücken"
Print
Void Inp(2)
' Puffer für Sektor bereitstellen
Buffer$=Space$(513)
Adr%=Varptr(Buffer$)
' ggf. auf gerade Adresse setzen
If (Adr% Mod 2)=1 Then
  Inc Adr%
Endif
' ersten Directory-Sektor lesen
A%=Bios(7,0)
A%=Bios(4,0,L:Adr%,1,11,0)
If A%=0 Then
  If Peek(Adr%+11)=8 Then
    ' erste 11 Zeichen = Diskettenname
    N$=Left$(Buffer$,11)
    Print "Alter Name: ";N$
    ' neuen Namen abfragen
    Input "Neuer Name: ",A$
    ' Namen aufbereiten
```

```
If Len(A$)<11 Then
    A$=A$+Space$(11)
Endif
A$=Left$(Upper$(A$),11)
' neuen Namen in den Sektor schreiben
Mid$(Buffer$,1,11)=A$
A%=Bios(4,1,L:Adr%,1,11,0)
Else
    Print "Diese Diskette hat keinen Namen!"
Endif
Else
    Print "Fehler"
Endif
```

Hier die Version für Omikron- bzw. ST-BASIC:

```
0 ' DISKNAME - Programm zum Umbenennen einer Diskette
1 ' Version für Omikron- bzw. ST-BASIC
2 '
3 PRINT "DISKNAME"
4 PRINT "Dieses Programm ermöglicht das Ändern des Namens der"
5 PRINT "Diskette in Laufwerk A. Legen Sie bitte die Diskette ein."
6 PRINT "Fertig -> Taste drücken"
7 PRINT
8 A$= INPUT$(1)
9 ' Puffer für Sektor bereitstellen
10 Buffer$= SPACE$(513)
11 Adr$L= LPEEK( VARPTR(Buffer$))+ LPEEK( SEGPTR +28)
12 ' ggf. auf gerade Adresse setzen
13 IF (Adr$L MOD 2)=1 THEN
14     Adr$L=Adr$L+1
15 ENDIF
16 ' ersten Directory-Sektor lesen
17 BIOS (A%,7,0)
18 BIOS (A%,4,0, HIGH(Adr$L), LOW(Adr$L),1,11,0)
19 IF A%=0 THEN
20     IF PEEK(Adr$L+11)=8 THEN
21         ' erste 11 Zeichen = Diskettenname
22         N$= LEFT$(Buffer$,11)
23         PRINT "Alter Name: ";N$
24         ' neuen Namen abfragen
25         INPUT "Neuer Name: ";A$
26         ' Namen aufbereiten
27         IF LEN(A$)<11 THEN
28             A$=A$+ SPACE$(11)
29         ENDIF
30         A$= LEFT$( UPPER$(A$),11)
31         ' neuen Namen in den Sektor schreiben
32         MID$(Buffer$,1,11)=A$
33         BIOS (A%,4,1, HIGH(Adr$L), LOW(Adr$L),1,11,0)
34     ELSE
35         PRINT "Diese Diskette hat keinen Namen!"
```

```
36 ENDIF
37 ELSE
38 PRINT "Fehler"
39 ENDIF
```

 Nach dem Start des Programms werden Sie aufgefordert, die Diskette einzulegen, die umbenannt werden soll. Wenn Sie das erledigt haben, drücken Sie bitte eine beliebige Taste. Das Programm untersucht dann die Diskette. Falls es einen alten Diskettennamen findet, wird dieser angezeigt, und Sie können einen neuen eingeben. Aus Vereinfachungsgründen können Sie dabei einfach eine bis zu 11 Zeichen lange Zeichenkette eingeben, in der keine Sonderzeichen vorkommen sollten (insbesondere keine Punkte und Backslashes). Die Eingabe wird mit  oder  beendet, danach wird der neue Name auf die Diskette gebracht.

Manchmal ist auf der Diskette kein Name eingetragen. Da eine Änderung in solchen Fällen Ihre Dateien zerstören könnte, gibt das Programm nur eine Meldung aus und stoppt dann. Eine Änderung ist dann nicht möglich. Wichtig: Bestimmte Disketten, die mit einem Sonderformat versehen wurden, können nicht umbenannt werden. Sie erkennen solche Disketten in der Regel daran, daß sie eine andere Kapazität als üblich besitzen.

Vielleicht haben Sie sich auch schon gewundert, warum auf ihrem Desktop zwei Datenstation-Icons (A und B) erscheinen, obwohl Sie nur ein Laufwerk besitzen. Ganz einfach. Beim Laufwerk B handelt es sich in diesem Fall um das sogenannte virtuelle Laufwerk, d.h. der Rechner simuliert nur das Vorhandensein eines zweiten Laufwerks. Das ermöglicht es Ihnen, mit zwei Disketten zu arbeiten. Wie funktioniert das? Führen Sie bitte einen Doppelklick auf das Icon der Station B aus. Bevor das gewohnte Fenster erscheint, verlangt der ST von Ihnen, eine Diskette für Laufwerk B einzulegen. Machen Sie das bitte mit einer beliebigen anderen Diskette als der gerade eingelegten. Nach dem OK-Klick erscheint das Inhaltsverzeichnis auf dem Desktop. Wie Sie sehen, haben Sie die Verzeichnisse zweier Disketten gleichzeitig auf dem Bildschirm. Wenn Sie nun wieder auf eine Datei von der Diskette A zugreifen wollen, verlangt der ST zuerst von Ihnen, diese Scheibe einzulegen. Der Rechner merkt sich also, welche Diskette gerade eingelegt ist.

Nehmen wir an, Sie möchten eine Datei von einer Diskette zur anderen kopieren. Legen Sie die Quelldiskette in das Laufwerk ein. Diese Diskette stellt ab sofort das Laufwerk A dar. Schieben Sie nun das Icon der Datei auf die andere Datenstation (B). Der Rechner wird Sie nach kurzer Zeit auffordern, die Diskette für Laufwerk B einzulegen. Damit ist die Zieldiskette gemeint, die ab jetzt für das Laufwerk B steht. Immer wenn der ST beim Vorhandensein zweier Floppy-Stationen nun auf ein anderes Laufwerk zugreifen müßte, for-

dert er von Ihnen die entsprechende Diskette an. Dieses Spielchen wiederholt sich so lange, bis alle Daten übertragen sind.

3.3 Die RAM-Disk

Eine RAM-Disk ist nichts weiter als ein Programm, das im Speicher des Computers eine Diskettenstation simuliert. Eine RAM-Disk kann mit einigen Einschränkungen genau wie ein normales Diskettenlaufwerk betrieben werden.

Ein RAM-Disk-Programm wird mit dem ST (im Gegensatz zu einigen anderen Computern) nicht ausgeliefert. Sie müssen sich daher ein solches Programm besorgen. Ihr Händler hält sicher eine ganze Reihe bereit, aber es geht unter Umständen auch ohne große Kosten. Mittlerweile existieren viele Public-Domain-RAM-Disks, die Sie für eine geringe Gebühr oder sogar kostenlos erwerben können.

Die Vorteile sind schon bestechend: Eine RAM-Disk ist wahnsinnig schnell, unter Umständen zehnmal schneller als eine normale Diskettenstation. Außerdem erleichtert Sie den Umgang mit Dateien, denn man kann Sie beim Kopieren von Dateien als Zwischenlager verwenden und erspart sich so lästige Diskettenwechsel. Falls Programme sehr häufig auf immer die gleichen Dateien zugreifen müssen (z.B. Compiler auf die Libraries, Grafikprogramme auf Symbolbibliotheken usw.), sind diese in einer RAM-Disk meist sehr gut untergebracht. Diese Zugriffe, die sonst sehr aufhalten können, sind dann in Nullkomma-nichts durchgeführt. Der große Nachteil einer RAM-Disk ist natürlich, daß sie Speicher frißt. Insbesondere, wenn Ihr ST "nur" 512 KByte Speicher hat, kann sich die RAM-Disk schon störend bemerkbar machen. Nicht vergessen werden sollte, daß die Datensicherheit bei weitem nicht so groß ist, wie bei normalen Disketten. Sobald Sie den Strom abschalten, sind die Daten weg.

Die RAM-Disk ist ein Muß für alle, die nur ein Diskettenlaufwerk besitzen. Die gesamten Dateioperationen können immens beschleunigt werden. Besitzer von zwei Diskettenlaufwerken sind mit einer RAM-Disk (am besten eine bei Bedarf "zuschaltbare") auch nicht schlecht bedient. Stolze Besitzer einer Festplatte können im allgemeinen auf simulierte Laufwerke verzichten. Die Harddisk ist in den meisten Fällen nur unwesentlich langsamer als eine RAM-Disk, so daß die meisten Vorteile wegfallen und die Nachteile überwiegen.

Die Installation einer RAM-Disk hängt vom jeweiligen Programm ab. RAM-Disk-Accessories machen naturgemäß keine Probleme, sie werden einfach auf die Systemdiskette kopiert und stehen dann nach dem Einschalten des Rechners zur Verfügung. Die andere Möglichkeit: Die RAM-Disk ist als TOS- oder

TTP-Programm gespeichert. In diesen Fällen können Sie sie einfach in den AUTO-Ordner Ihrer Systemdiskette kopieren; die RAM-Disk wird dann nach dem Einschalten des Rechners ebenfalls automatisch gestartet. Je nach Programm müssen Sie dann noch angeben, wie groß das neue (simulierte) Laufwerk werden soll. Einige RAM-Disks erlauben sogar die nachträgliche Änderung der Größe, so daß Sie sie immer ganz individuell anpassen können. Gute Programme melden das simulierte Laufwerk (falls nötig) auch gleich beim Desktop an, aber das ist nicht immer so. Wenn Sie also nach dem Start kein zusätzliches Icon auf dem Desktop finden, müssen Sie den Menüpunkt "Floppy anmelden" benutzen. Den Kennbuchstaben für das neue Laufwerk erhalten Sie in den meisten Fällen vom RAM-Disk-Programm oder Sie konnten ihn selbst festlegen.



Wenn Sie nach dem Anmelden der RAM-Disk den Menüpunkt "Arbeit sichern" anwählen (während Ihre Systemdiskette eingelegt ist), so brauchen Sie das Anmelden beim nächsten Start des Rechners nicht mehr vorzunehmen. Dies geschieht dann automatisch durch das Laden der Datei DESKTOP.INF.



Wenn Sie die RAM-Disk nicht jedesmal beim Einschalten des Rechners starten (z.B. per AUTO-Ordner oder als Accessory), dann sollten Sie den obigen Tip nicht befolgen, da damit das neue Laufwerk auch dann angemeldet wird, wenn gar kein Programm dahintersteht.

In vielen Fällen sprechen die Hersteller von reset-fester oder residenter RAM-Disk. Beide Begriffe meinen das gleiche: Der Inhalt des simulierten Laufwerks übersteht die Betätigung des Reset-Tasters an der Rückseite des ST unbeschadet, sofern dabei die Systemdiskette wieder eingelegt wird (das Programm bzw. das Accessory muß neu geladen werden). Nach dem Booten steht dann die RAM-Disk mit dem alten Inhalt zur Verfügung. Wohlgemerkt, dies bezieht sich nur auf den Reset. Wird der Rechner durch Aus- und Einschalten neu gestartet, hat selbst die beste RAM-Disk keine Chance mehr. Durch die Unterbrechung der Spannung ist der Speicher komplett gelöscht worden.



Wie arbeitet ein RAM-Disk-Programm? Eigentlich recht einfach. Es "klinkt" sich ins Betriebssystem ein, d.h. alle Zugriffe auf eine Datenstation laufen zunächst durch das RAM-Disk-Programm. Dieses filtert die Zugriffe. Sobald darin die Kennung der RAM-Disk auftaucht, werden die Daten nicht mehr von der Diskette geholt bzw. dorthin gebracht. Statt dessen werden sie in den Speicher umgeleitet. Sogenannte residente RAM-Disks organisieren den Speicherbereich für die Daten so, daß er bei einem Reset des ST nicht zerstört wird.

Die Arbeit mit einer RAM-Disk vollzieht sich ganz genauso, wie Sie es auch mit einer normalen Datenstation tun würden. Sie können die Icons verschieben, löschen und auch sonst alles tun, was Sie auf anderen Laufwerken machen. Das Schöne daran ist aber: Alles geht sehr viel schneller. So benötigen Sie zum Löschen aller Einträge einer RAM-Disk selten mehr als 3 Sekunden. Eines können Sie nicht tun: 1:1-Duplikate anfertigen, indem Sie das RAM-Disk-Icon auf eine andere Datenstation schieben. Außerdem ist es nicht möglich, eine RAM-Disk zu formatieren. Das Desktop weigert sich schlicht, den entsprechenden Menüpunkt für andere Laufwerke als A und B zugänglich zu machen.



Häufig kommt es vor, daß man Dateien unter anderem Namen auf eine andere Diskette kopieren möchte. Das kann man mit einer RAM-Disk besonders effizient tun:

- 1) Alle Dateien auf eine genügend große RAM-Disk kopieren.
- 2) Dateien in der RAM-Disk umbenennen. (Das geht wahnsinnig schnell.)
- 3) Geänderte Dateien auf die Zieldiskette kopieren.

Pannenhilfe:



Bei der Arbeit mit einem Anwendungsprogramm haben Sie versucht, Dateien auf der RAM-Disk zu speichern. Das ging nicht, statt dessen erhielten Sie eine Fehlermeldung wie "Pfadname existiert nicht" oder ähnliches.



Normalerweise tauchen solche Fehler auf, wenn man versucht, auf einen nicht existenten Ordner zuzugreifen. Bei RAM-Disks kann es allerdings auch vorkommen, daß diese Fehlermeldung erscheint, wenn das betreffende Laufwerk zwar beim Desktop angemeldet ist (und deshalb ein Icon erscheint), hinter diesem Icon aber kein Simulationsprogramm steht. Klarheit kann hier ein Doppelklick auf das Laufwerks-Icon bringen. Erscheint statt des erwarteten Fensters die Meldung "Ein Laufwerk mit dieser Kennung ist unbekannt", so wurde das RAM-Disk-Programm aus irgendwelchen Gründen nicht gestartet.



Sie haben kurz zuvor die Reset-Taste betätigt. In solchen Fällen kann es vorkommen, daß die Zuordnung von Icons zu Laufwerken durcheinandergerät. Die RAM-Disk, die üblicherweise z.B. als Laufwerk C benutzt wurde, findet sich dann unter der Bezeichnung D wieder.



Melden Sie Icons für andere Laufwerkskennungen, die bisher nicht benutzt wurden, an, und testen Sie, ob sich dahinter die verlorengelaubte RAM-Disk verbirgt.

3.4 Die Festplatte

Eine Festplatte (oder Harddisk) funktioniert vom Prinzip her wie eine Diskette, es gibt jedoch wichtige konstruktive Unterschiede. Der Datenträger in einer Festplatte besteht aus einem festen Material, z.B. Metall. Außerdem sind hier oft mehrere Platten übereinander auf einer Motorachse montiert. Ihren Namen haben diese Geräte daher, daß die Platten fest in das Gehäuse eingebaut sind und daher nicht ausgewechselt werden können. Diese feste Montage ermöglicht eine sehr viel genauere Führung der Tonköpfe, wodurch die Platte sehr eng beschrieben werden kann. Als Ergebnis kommt dabei bei gleicher Plattengröße im Vergleich zur Diskette eine zig-fache Kapazität heraus. Außerdem können Festplattenlaufwerke wegen der hohen Umdrehungsgeschwindigkeit (3600 U/min) Daten rasend schnell schreiben und lesen. Um diese Geschwindigkeit auch bei der Übertragung zum Rechner zu gewährleisten, wird hier das DMA-Prinzip angewendet.

Für den ST gibt es mittlerweile mehrere Festplatten-Modelle (SH204, SH205, Megafile 30 usw.). Die technischen Unterschiede sind eigentlich minimal. Sie können davon ausgehen, daß die SH205 eigentlich nur eine hübschere Variante der älteren SH204 ist. Die Megafile-Modelle sind schneller und moderner, bei der Anwendung ergeben sich jedoch keine Änderungen. Übrigens gilt das auch für das Wechsellplattenlaufwerk, das von Atari angeboten wird. Betrachten Sie es einfach als eine Festplatte mit auswechselbarem Datenträger.

3.4.1 Installation einer Festplatte



Die nachfolgend vorgestellten Abläufe sind in genau dieser Form nur für die Atari-Festplatten SH204 und SH205 gültig. Andere Harddisks (auch von Fremdanbietern) werden im Prinzip ähnlich behandelt, doch ergeben sich wegen der verschiedenen Treiberprogramme Änderungen, die hier leider nicht berücksichtigt werden können. Im Zweifelsfall ziehen Sie bitte die Anleitung Ihrer Festplatte und ggf. Ihren Händler zu Rate.

Der erste Schritt ist natürlich das Anschließen. Zunächst einmal: Anschlüsse nur bei ausgeschalteten Geräten vornehmen. Die Festplatte wird an den sogenannten DMA-Port des ST angeschlossen, der mit "Hard Disk" bezeichnet ist.

Sollte dieser Stecker schon durch ein anderes Gerät (z.B. Atari-Laserdrucker) belegt sein, so sehen Sie bitte im dazugehörigen Handbuch nach, wo Sie das Verbindungskabel statt dessen anbringen können. Wichtig ist auch, daß Sie die dabei vorgeschriebene Reihenfolge einhalten.



Wie jedes Laufwerk braucht auch eine Festplatte einen Controller. Für die Diskettenlaufwerke ist er schon in den ST eingebaut, bei Festplatten ist er dagegen in das Festplattengehäuse eingebaut. Dieser Controller ist dafür verantwortlich, daß nur spezielle ST-Harddisks angeschlossen werden können. Festplatten für PCs (die oft erheblich billiger sind), benutzen andere Controller, die der ST nicht bedienen kann. Hin und wieder findet man allerdings Anpassungsschaltungen, die dieses Problem lösen.

Vielleicht ist Ihnen schon aufgefallen, daß das Verbindungskabel zwischen ST und Hard-Disk sehr kurz ist. Das hat seine Gründe. Zum einen gilt, daß eine Datenübertragung umso störanfälliger ist, je höher die Geschwindigkeit ist. Die Datenverbindung zur Festplatte arbeitet mit Übertragungsraten von bis zu 8 Millionen Bits pro Sekunde. Hinzu kommt, daß ein Kabel immer wie eine Antenne wirkt, die Störimpulse geradezu aufsaugt. Um daher eine geringe Störanfälligkeit zu garantieren, haben die Entwickler der Festplatten ein sehr kurzes Kabel vorgesehen. Sie sollten nicht der Versuchung erliegen und dieses Kabel verlängern. Das kann zwar funktionieren, muß es aber nicht.



Die Gehäuse der meisten Festplatten sind so robust, daß man sie ohne weiteres auch als Monitorständer benutzen kann. Auf diese Weise stört das kurze Verbindungskabel nicht so sehr.

Wenn Sie das Gerät angeschlossen haben, können Sie mit den letzten Vorbereitungen für den Betrieb beginnen. Wenn Sie die Festplatte bereits früher schon einmal benutzt haben, brauchen Sie natürlich nichts weiter zu tun. Ist es aber das erste Mal, daß Sie dieses Gerät an Ihren Rechner anschließen, so müssen Sie die Harddisk zunächst formatieren. Zunächst machen Sie bitte von der mitgelieferten Diskette eine Kopie. Das müssen Sie (noch) bei ausgeschalteter Festplatte tun.



Es ist nicht unbedingt nötig, die Treiberdiskette 1:1 zu kopieren. Da die Dateien sehr klein sind, ist das normale Dateien-Kopieren über eine RAM-Disk sehr viel schneller.

Im AUTO-Ordner befindet sich die Datei AHDI.PRГ. Dabei handelt es sich um den eigentlichen Treiber, der für die richtige Behandlung der Festplatte zuständig ist. Sie können auf keinen Fall mit der Festplatte arbeiten, ohne die-

sen Treiber vorher gestartet zu haben. Solange der Treiber im AUTO-Ordner der Systemdiskette liegt, wird er automatisch gestartet - Sie brauchen sich also nicht darum zu kümmern. HDX.PRG ist ein Programm zum Formatieren und Partitionieren; der Festplatte, aber auch für andere Funktionen. Schließlich gibt es SHIP.PRG, das für ein Parken der Schreib-/Leseköpfe sorgt, falls das Gerät transportiert werden soll.

Die eigentliche Formatierung der Harddisk ist recht einfach - dafür ist HDX.PRG zuständig. Starten Sie dieses Programm bitte von der Systemdiskette (nicht von der Harddisk!), und wählen Sie den Menüpunkt "Format" an. Zunächst erscheint eine englischsprachige Warnung, die besagt, daß alle Daten auf der Festplatte verloren gehen, wenn Sie formatieren. Durch "Cancel" können Sie diesen Vorgang abbrechen, mit "OK" geht es weiter. Dann müssen Sie das Gerät auswählen, welches formatiert werden soll. Wenn Sie nur eine Harddisk angeschlossen haben (das ist wohl der Normalfall), können Sie nur UNIT 0 auswählen. Klicken Sie dieses Icon bitte einmal an, danach OK. Falls Sie mehrere Festplatten besitzen, wird Ihr Händler Ihnen gesagt haben, welche Nummer welche Platte hat (falls nicht, fragen Sie bitte nach). Dann möchte das Programm noch wissen, welchen Typ von Laufwerk Sie besitzen. Wählen Sie bitte SH204 durch Anklicken und Druck auf die OK-Box aus. (SH104 ist ein kleineres Modell, das auf dem deutschen Markt nicht ausgeliefert worden ist.).

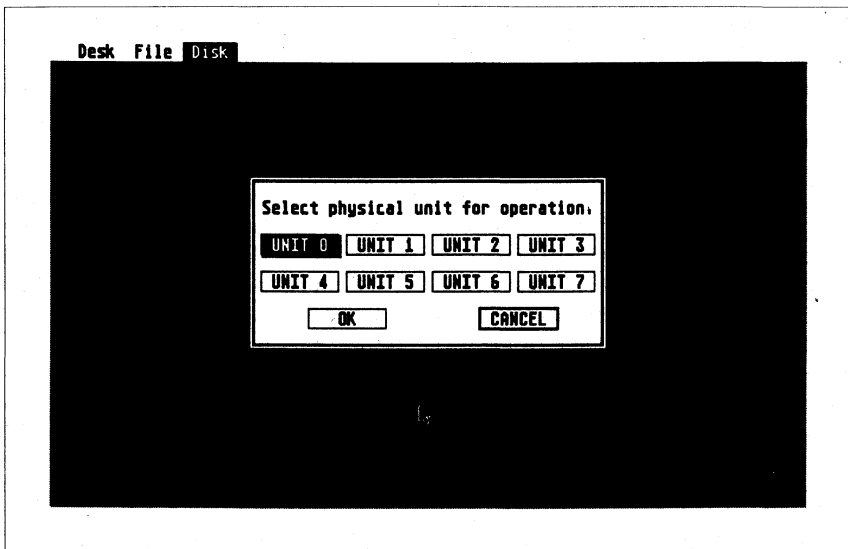


Abb. 3.6: Das Formatierungsmenü

Sind Sie einmal soweit vorgedrungen, erhalten Sie eine erneute Sicherheitsmeldung. Das Programm möchte wissen, ob sie WIRKLICH die Festplatte formatieren möchten. "Yes" startet diesen Vorgang, mit "No!" geht's zurück zum Menü. Während des Formatierens, das etwa ein bis zwei Minuten in Anspruch nimmt, erscheint keinerlei Information auf dem Bildschirm. Lediglich die Maus nimmt die bekannte Bienenform an. Nach einer Weile erscheint dann wieder der normale Mauspfel, die Formatierung ist beendet.

Sie können die Festplatte jetzt noch nicht benutzen. Erst müssen Sie partitionieren, d.h. die 20,3 MByte (oder mehr bei anderen Modellen), die Ihnen zur Verfügung stehen, in bis zu 4 Partitionen aufteilen. Eine einzelne Partition kann bis zu 16 MByte umfassen. Ausnahme: Seit einiger Zeit wird eine neue Version des HDX (4.02) ausgeliefert, die Partitionen mit einer Größe von bis zu 2 Gigabyte (das sind 2048 Megabyte) zuläßt. Außerdem sind mehr als 4 Partitionen möglich.

Eine Partition ist eine Art virtuelles Laufwerk, die das Treiberprogramm auf der Festplatte einrichtet. Jede Partition wird vom ST wie ein eigenständiges Laufwerk behandelt, auf der (fast) alle üblichen Aktionen ablaufen können. Physisch werden aber alle Partitionen auf einem Gerät zusammengefaßt, um dessen große Kapazität nutzen zu können.



Die Aufteilung ist nötig, weil der ST nicht mehr als 16 MByte in einem Laufwerk verwalten kann. Der Grund dafür: Für die FAT ist bei der Festlegung der Laufwerkspezifikation nur ein gewisser Platz festgelegt worden. Dieser Platz reicht nun einmal maximal für 16 MByte. Zu der Zeit, als dies festgelegt wurde, waren Festplatten mit Kapazitäten von 10 MByte das höchste der Gefühle. Sie sehen, in welch kurzer Zeit die Entwicklung unseren vergleichsweise modernen ST überholt hat.

Für die Einrichtung der Partitionen ist wiederum HDX.PRg zuständig. Wählen Sie bitte den Menüpunkt "Partition" an. Es erscheinen wieder die üblichen Sicherheitsabfragen, die Sie bereits vom Formatieren her kennen. Wichtig ist, daß auch das Partitionieren alle Daten auf der Festplatte löscht. Sie können zwar die Partitionierung aufrufen, ohne neu zu formatieren (etwa um die Aufteilung Ihren geänderten Wünschen anzupassen), aber auch dabei müssen Sie den Datenverlust in Kauf nehmen. Das HDX-Programm fragt Sie dann, wie die Laufwerke eingeteilt werden sollen. Die genaue Aufteilung bleibt Ihnen überlassen; einige Vorschläge dazu können Sie im Programm durch einen Klick auf "Menu" erhalten. Wenn Sie in einer der beiden Dialogboxen OK anklicken, wird die bisher gewählte Partition übernommen (nicht ohne daß vorher noch einmal eine Sicherheitsabfrage erscheint); "Cancel" bringt Sie wie üblich zum Menü zurück. Je nachdem, wieviele Partitionen Sie einrichten, wer-

den diese vom ST später als Laufwerke C, D, E und F behandelt (bei der Benutzung mehrerer Festplattengeräte auch weitere Buchstaben).

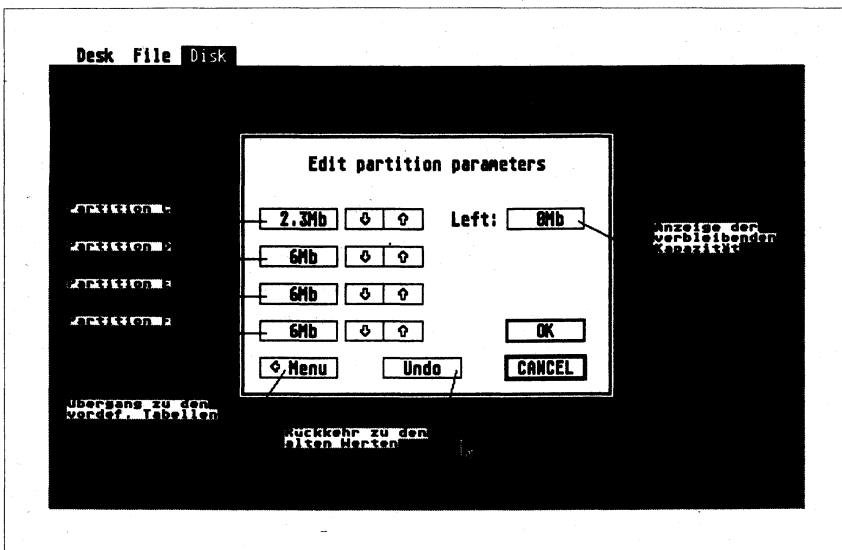


Abb. 3.7: Das Partitionierungsmenü

Nun ist nur noch eines zu tun. Für jede der eingerichteten Partitionen sollten Sie einmal den Menüpunkt "Markbad" aufrufen. Damit werden alle schadhaften Sektoren der Harddisk ausgemustert, um später einen fehlerfreien Betrieb zu gewährleisten.



Es gibt keine Festplatte ohne Fehler in der Magnetschicht. Diese Fehler sind herstellungsbedingt und werden soweit möglich schon im Werk erkannt. Aus diesem Grund werden Festplatten auch immer mit einer leicht höheren Kapazität hergestellt als der Hersteller eigentlich garantiert. Fallen einzelne Teile aus, so können die eigentlich überschüssigen Sektoren den Kapazitätsverlust wieder ausgleichen.

Was bleibt nun noch zu tun? Zunächst verlassen Sie bitte HDX.PRG. Dabei wird ein Reset ausgeführt, lassen Sie also Ihre Systemdiskette mit dem AHDI.PRG im Laufwerk A. Nach einiger Zeit erscheint das Desktop, auf dem jetzt ein weiteres Laufwerk-Icon mit der Bezeichnung "Hard-Disk" erscheint. Jetzt bleiben nur noch Kleinigkeiten zu tun:

- 1) Für jede eingerichtete Partition melden Sie bitte ein Laufwerk mit dem zugehörigen Buchstaben (D-F) an. Die Partition C wurde automatisch durch AHDI angemeldet.
- 2) Aktivieren Sie das Fenster für das Hauptverzeichnis von Laufwerk C, und klicken Sie dann bitte "Arbeit sichern" an. In der Datei DESKTOP.INF sind jetzt die eben angemeldeten Laufwerke gespeichert. Damit ist dafür gesorgt, daß Sie die Partitionen beim nächsten Start Ihres Rechners nicht neu anmelden müssen.
- 3) Kopieren Sie alle Accessories, die Sie benutzen möchten, auf Laufwerk C. Ab jetzt werden Accessories (nicht aber der AUTO-Ordner) bei jedem Booten von der Harddisk geladen. Damit verkürzt sich die Ladezeit ganz erheblich.
- 4) Das System steht zu Ihrer Verfügung. Ab jetzt können Sie damit arbeiten, wie es Ihnen beliebt.



Teilen Sie Ihre Programme und Dateien nach Themen geordnet auf die einzelnen Partitionen auf - z.B. Textverarbeitung, Programmiersprachen usw.

Zunächst ändert sich nicht viel. Oben habe ich bereits erwähnt, daß ab sofort die Accessories von der Harddisk geladen werden. Stehen Sie dort nicht, weicht der ST allerdings selbständig wieder auf die alte Systemdiskette aus. Außer der höheren Kapazität und der enormen Geschwindigkeit werden Sie allerdings nicht viel von der Festplatte merken. Auf Ihrer Systemdiskette muß aber ab jetzt im AUTO-Ordner der Festplattentreiber AHDI vorhanden sein, sonst können Sie das Gerät nicht nutzen.



Seit einiger Zeit gibt es Public-Domain-Programme, die es erlauben, das gesamte System komplett von der Harddisk hochzufahren. Mit diesen Programmen wird die Festplatte so eingerichtet, daß AHDI von der Festplatte und nicht mehr von der Systemdiskette gebootet werden kann. Seit der SH205 liefert Atari auch selbst ein solches Programm auf der Treiberdiskette mit aus. Es muß nur gestartet werden, danach ist der Treiber auf der Festplatte installiert.

3.4.2 Die Arbeit mit der Festplatte

Prinzipiell unterscheidet sich die Arbeit mit der Harddisk nicht von der mit anderen Datenstationen. Sie können die Dateien wie auch bei einer Diskette

kopieren, öffnen und löschen. Naturgemäß ist es nicht möglich, 1:1-Kopien einer Festplatten-Partition zu machen. Und natürlich weigert sich das Desktop auch, eine Festplatte zu formatieren. Das können Sie ausschließlich mit dem mitgelieferten Spezialprogramm.



Wenn Sie einmal alle Dateien einer Festplatten-Partition löschen wollen, brauchen Sie dies nicht unbedingt per Papierkorb tun. Etwas einfacher ist die Funktion ZERO im HDX.PRG, das den gleichen Job etwas schneller ausführt.

Es ist natürlich möglich, auf einer Festplatte alle Dateien im Hauptverzeichnis unterzubringen. Aber das wird schnell sehr unübersichtlich. Deshalb gibt es für die Arbeit mit der Festplatte einige goldene Regeln:

- 1) Nur solche Dateien auf die Festplatte bringen, die wirklich häufig gebraucht werden. Soll meinen: Wenn Sie die Arbeit an einem Projekt abgeschlossen haben und die dazugehörigen Dateien für längere Zeit nicht mehr benötigen, lagern Sie diese auf Disketten aus. So wird der doch recht teure Speicherplatz auf der Festplatte nicht unnötig belegt. Sollten Sie dann später diese Dateien wieder häufiger benötigen, kopieren Sie sie einfach wieder auf die Festplatte.
- 2) Fassen Sie logisch zusammengehörige Dateien in Ordnern zusammen. Auf diese Weise werden die Inhaltsverzeichnisse übersichtlicher, weil z.B. die Textdateien alle im Verzeichnis TEXTE stehen, während die Grafiken im Ordner GRAFIKEN abgelegt werden.
- 3) Löschen Sie von Zeit zu Zeit nicht mehr benötigte BAK-Dateien. Sie werden sich wundern, wieviel Platz dadurch wieder frei wird.



Es gibt einige Programme, die in bestimmten Verzeichnissen gespeichert sein müssen, um die gewünschte Wirkung zu zeigen. Bestes Beispiel dafür sind die Accessories, die nur dann beim Booten geladen werden, wenn Sie im Hauptverzeichnis von Partition C stehen. Der AUTO-Ordner zeigt dagegen nur dann Wirkung, wenn er im Haupt-Verzeichnis der Systemdiskette angelegt wurde.

Wenn Sie häufig Dateien löschen und wieder neue Dateien erzeugen, kann es vorkommen, daß die Festplatte vergleichsweise langsam wird. Ursache: Beim Erzeugen neuer Dateien versucht die Festplatte zuerst, die Plätze gelöschter Dateien neu zu belegen. Das führt dazu, daß einzelne Files recht weit über die Plattenoberfläche verstreut werden, so daß der Zugriff darauf sehr langsam wird. Es gibt zwei Möglichkeiten, dies zu ändern. Die erste ist sicher die bequemere. Sie besorgen sich ein spezielles Reorganisationsprogramm, das alle

Dateien auf der Festplatte untersucht und bei Bedarf möglichst zusammenhängend speichert. Die zweite Möglichkeit ist sehr arbeitsintensiv. Speichern Sie möglichst alle Dateien auf Disketten ab, dabei können Sie nach Partitionen getrennt vorgehen. Dann löschen Sie bitte alle gesicherten Dateien auf der Festplatte. Wenn Sie den ehemaligen Inhalt nun wieder zurückspeichern, werden die Dateien zusammenhängend gespeichert; der Zugriff wird wieder schneller.

Beim Einschalten des Rechners (bitte erst 15 Sekunden nach dem Einschalten der Festplatte) muß sich in Laufwerk A die Systemdiskette mit dem Harddisk-Treiber-Programm befinden, es sei denn, Sie haben den Treiber auf der Festplatte installiert. Ohne dieses ist keine korrekte Arbeit möglich.



Das Festplatten-Treiber-Programm können Sie sich quasi wie eine Erweiterung des Betriebssystems vorstellen. Der ST weiß nach dem Einschalten zunächst nicht, welche Geräte vorhanden sind und wie diese angesprochen werden müssen. Wäre dies bereits im Betriebssystem fest verankert, so könnten immer nur bestimmte, festgelegte Geräte an den ST angeschlossen werden. So ist aber die Möglichkeit gegeben, neue Entwicklungen zu berücksichtigen. Das Treiberprogramm sorgt nun dafür, daß alle Zugriffe auf die Festplatte in der für dieses spezielle Gerät notwendigen Weise geschehen. Fehlt der Treiber oder wird er beschädigt, so ist eine Benutzung der Festplatte unmöglich. Für den Fall der Fälle sollten Sie daher die Originaldiskette, die der Festplatte beigelegt war, gut aufheben, um evtl. defekte Programme durch das Original ersetzen zu können.

Was muß ich tun, wenn ich einmal ohne Festplatte arbeiten möchte? Das ist sehr einfach. Schalten Sie die Festplatte gar nicht erst ein, und booten Sie den Rechner mit einer Systemdiskette, in dessen AUTO-Ordner der AHDI nicht vorhanden ist. Tun Sie dies, obwohl die Festplatte eingeschaltet ist, so hat der ST diese zwar als physikalisches Gerät erkannt (es erscheint dann auch ein Icon für Laufwerk C), doch logisch kann er sie nicht bedienen. Hinzu kommt das Risiko, daß einige Viren auch eine solche "teilweise" vorhandene Festplatte infizieren können. Also immer ganz ausschalten!

Pannenhilfe:



Nach Beendigung des Einschaltvorganges werden die Icons für die Festplatte nicht auf dem Desktop dargestellt.

✓ Melden Sie ein neues Laufwerk mit dem Kennbuchstaben C und der Bezeichnung "Hard-Disk" an. Starten Sie dann das Programm AHDI.PRG im Auto-Ordner Ihrer Systemdiskette. Nach der Beendigung dieses Programms gibt es zwei Möglichkeiten:

- a) Das Icon für Laufwerk C ist immer noch auf dem Desktop. Sehr gut! In diesem Fall war die Festplatte einfach nicht korrekt beim Desktop angemeldet. Löschen Sie bitte (falls vorhanden) die Datei DESKTOP.INF von Laufwerk C. Danach können Sie die Partitionen D, E und F noch anmelden, falls Sie diese beim Formatieren vorgesehen hatten. Nachdem dies geschehen ist, klicken Sie bitte den Menüpunkt "Arbeit sichern" im Desktop an. Damit werden die eben angemeldeten Laufwerke in der Datei DEKSTOP.INF gespeichert und stehen somit beim nächsten Start zur Verfügung.
- b) Die andere Möglichkeit: Das Icon für Laufwerk C ist nach der Beendigung von AHDI.PRG wieder verschwunden. Starten Sie bitte HDX.PRG (von der mitgelieferten Diskette), und sehen Sie nach, welche Werte für die einzelnen Partitionen eingetragen sind. Stehen dort keine oder irrealen Werte, dann haben Sie leider Pech gehabt: Ihre Daten sind weg. Sie müssen die Hard-Disk neu formatieren (siehe den letzten Abschnitt). Hat auch das keinen Erfolg, müssen Sie den Sektor 0 des Laufwerks C komplett mit Nullen überschreiben (ein Programm dazu ist unten abgedruckt) und danach die Festplatte erneut formatieren. Ist auch dies erfolglos, liegt ein Hardware-Fehler vor; das Gerät muß in die Werkstatt.

Waren die Werte für die Partition dagegen in Ordnung, so können Sie das HDX mit QUIT verlassen. Starten Sie bitte den Rechner neu. Tritt der Fehler dann immer noch auf, bringen Sie das Gerät bitte mit einer möglichst genauen Fehlerbeschreibung zur Werkstatt.



Nach dem Einschalten erscheint das Desktop nicht. Außerdem werden auf dem Bildschirm kleine Bomben abgebildet oder das Bild flimmert ab und zu und der Boot-Vorgang scheint erneut zu beginnen.



Eines der Accessories scheint defekt zu sein. Um diesen Fehler zu beheben, müssen Sie zunächst an ein funktionierendes Desktop herankommen. Das geht so:

- 1) Alle Geräte ausschalten, Verbindungskabel zur Hard-Disk lösen.

- 2) Rechner, Diskettenlaufwerk und Monitor wieder einschalten. Tritt der Fehler hierbei nochmal auf, so liegt er im AUTO-Ordner Ihrer Systemdiskette begründet. Gehen Sie dann bitte weiter wie im Kapitel 2 ("Der Einschaltvorgang") vor.
- 3) Falls das Desktop erscheint, melden Sie bitte Laufwerk C wieder an. Verbinden Sie die Hard-Disk wieder mit dem ST, und schalten Sie die Hard-Disk ein.
- 4) Nach ca. 15 Sekunden starten Sie bitte das AHDI.PRГ aus dem AUTO-Ordner der System-Diskette.
- 5) Löschen Sie bitte im Hauptverzeichnis von Laufwerk C die Datei DESKTOP.INF.
- 6) Sie können jetzt damit beginnen, alle Accessories wie in Kapitel 2 einzeln zu überprüfen.

Sektor 0 mit Nullen überschreiben

Wenden Sie dieses Programm bitte nur in den oben beschriebenen, absoluten Notfällen an. Wenn Sie dieses Programm bei einer funktionierenden Festplatte anwenden, sind die darauf befindlichen Daten verloren, und Sie müssen neu formatieren!

Hier die Version für GFA-BASIC:

```
' BOOTNULL
' Dieses Programm überschreibt den Sektor 0 von Laufwerk C mit Nullen.
' Achtung! Auf keinen Fall bei Festplatten ohne Fehlersymptome oder
' auf einer RAM-Disk einsetzen!
'
Print "Dieses Programm zerstört Daten auf Laufwerk C. Sind Sie sicher,"
Print "daß Sie es dennoch einsetzen wollen? (j/n)"
Repeat
  T$=Inkey$
Until T$<>""
If T$="j" Then
  ' Platz für Boot-Sektor einrichten
  A$=String$(512,0)
  Adr%=Varptr(A$)
  ' neuen Boot-Sektor auf Diskette schreiben
  R%=Bios(4,3,L:Adr%,1,0,2)
Endif
```

Hier die Version für Omikron- bzw. ST-BASIC

```

1 ' BOOTNULL
2 ' Dieses Programm überschreibt den Sektor 0 von Laufwerk C mit Nullen.
3 ' Achtung! Auf keinen Fall bei Festplatten ohne Fehlersymptome oder
4 ' auf einer RAM-Disk einsetzen!
5 '
6 PRINT "Dieses Programm zerstört Daten auf Laufwerk C. Sind Sie sicher,"
7 PRINT "daß Sie es dennoch einsetzen wollen? (j/n)"
8 REPEAT
9   T$= INKEY$
10  UNTIL T$<>" "
11  IF T$="j" THEN
12    ' Platz für Boot-Sektor einrichten
13    A$= STRING$(512,0)
14    Adr$L= LPEEK( VARPTR(A$))+ LPEEK( SEGPTR +28)
15    ' neuen Boot-Sektor auf Diskette schreiben
16    BIOS (R$,4,3, HIGH(Adr$L), LOW(Adr$L),1,0,2)
17  ENDIF

```

Beim Transport einer Festplatte ist Vorsicht geboten. Damit hier nichts passieren kann, gibt es auf jeder Harddisk eine Spur, die nicht mit Daten belegt wird, die sogenannte "Park-Spur" oder auch Landezone. Auf der Diskette, die Ihrer Festplatte beigelegt war, ist ein Programm namens SHIP.PRГ enthalten, das die Köpfe Ihrer Festplatte auf diese Spur fahren kann und dort quasi bis zur nächsten Benutzung arrettiert. Wenn Sie also Ihre Festplatte bewegen wollen, sollten Sie zunächst dieses Programm aufrufen und sofort nach dessen Beendigung das Gerät ausschalten. Dann können Sie unbesorgt auf Reisen gehen. Aber Vorsicht: Nach dem Parken darf die Festplatte auf keinen Fall mehr benutzt werden, weil sonst die Köpfe wieder aus der Park-Spur herausfahren.

Wenn Sie dann die Harddisk transportieren wollen, sollten Sie dies trotz Kopf-Parkens immer nur in der Originalverpackung tun. Diese bietet zusätzlichen Schutz gegen Stöße, so daß Ihren Daten nichts mehr passieren kann.



Sie können das Programm zum Parken der Schreib-/Lese-Köpfe (SHIP.PRГ) nicht von der Festplatte aus starten! Das Programm selbst führt zwar seine Aufgabe korrekt aus, danach versucht aber das Desktop wieder, das Inhaltsverzeichnis dieses Laufwerks zu lesen. Dazu müssen die Köpfe zwangsläufig auf die Spur mit dem Inhaltsverzeichnis wandern. Und genau da sollten sie ja nicht hin. Also gehen Sie bitte so vor:

- 1) Nehmen Sie eine Diskette, auf der das Parkprogramm gespeichert ist, und legen Sie diese bitte in Laufwerk A.
- 2) Schließen Sie bitte bis auf das Fenster für Laufwerk A alle anderen Fenster (vor allem die für die Festplatte).

- 3) Starten Sie das Parkprogramm.
- 4) Sobald das Parkprogramm beendet ist, schalten Sie die gesamte Anlage ab. Starten Sie auf keinen Fall irgendwelche anderen Programme!
- 5) Jetzt können Sie die Festplatte abmontieren und transportieren.

Was ist zu tun, wenn es trotz aller Vorsicht doch einmal zu einem Schaden an der Festplatte gekommen sein sollte? Leider können Sie nicht so ohne weiteres feststellen, ob Ihre Harddisk einen Head-Crash hat. Bei ungewöhnlichem Verhalten (laute Geräusche o.ä.) prüfen Sie bitte, ob die Platte noch ansprechbar ist. Wenn ja, versuchen Sie bitte zunächst, die defekten Sektoren mittels der Funktion MARKBAD im HDX-Programm zu deaktivieren. Dabei gehen natürlich diejenigen Dateien verloren, die mit den defekten Sektoren zu tun hatten. Führen Sie diese Funktion für jede Partition aus. Es muß aber nicht sein, daß damit der Fehler behoben ist. In den meisten Fällen ist dann der Datenbestand endgültig verloren, die Hardware kann aber durch geschicktes Formatieren (bei dem die zerstörten Sektoren ausgespart werden) und evtl. einige (teure) Reparaturen gerettet werden. Fragen Sie bitte Ihren Fachhändler. Falls er die Reparatur selbst ausführen kann, weiß er zumindest eine Adresse, wo dies möglich ist.

Meldet sich die Platte überhaupt nicht mehr, sollten Sie das Gerät ausschalten und nicht mehr benutzen.

Wenn Sie Ihr Gerät zur Werkstatt bringen wollen und dieses noch ansprechbar ist, versuchen Sie unbedingt, die Köpfe zu parken. Es ist nicht sicher, ob dies wirklich funktioniert, aber sie könnten damit weiteren Schaden vermeiden. Und transportieren Sie die Festplatte bitte besonders vorsichtig (Originalpackung!).

3.5 Fehler beim Zugriff auf Datenstationen

Fehlermeldungen beim Zugriff auf Datenstationen sind in den meisten Fällen auf zwei Ursachen zurückzuführen. Entweder ist die Hardware defekt, oder Sie haben falsche Datei- bzw. Pfadnamen angegeben. Im ersten Fall haben Sie wenig Chancen, die Daten zu restaurieren. Meistens wurde die magnetisierbare Oberfläche einer Platte (Diskette oder Harddisk) durch Kratzer und/oder Verschmutzung beschädigt. Ein Versuch, Daten von solchen Stellen zu bearbeiten, führt auf jeden Fall zu Fehlermeldungen, in einigen Fällen sogar zur weiteren Zerstörung der Hardware. Besonders gefürchtet ist der sogenannte Head-Crash einer Festplatte. Damit ist das Aufschlagen der Schreib-/Lese-

Köpfe auf der Plattenoberfläche gemeint, die fast nie ohne Unfall (sprich Beschädigungen) ausgeht. Die andere Möglichkeit zur Zerstörung von Daten ist der Einfluß magnetischer Felder auf den Datenträger. Gerät eine Diskette in ein starkes Magnetfeld (und wird womöglich noch darin bewegt), so kann es passieren, daß die magnetischen Datensignale derart abgeschwächt werden, daß sie das Laufwerk nicht mehr erkennt. In solchen Fällen sind die Daten ebenfalls verloren, doch bleibt Ihnen wenigstens der Trost, daß Sie die Diskette nach erneuter Formatierung weiterbenutzen können.



Headcrash ist ein Wort, das die alten Hasen unter den Festplattenbesitzern erschauern läßt - nicht ohne Grund, wie Sie gleich sehen werden. Aus den vorangegangenen Abschnitten wissen wir, daß eine Festplatte mit sehr eng beieinander liegenden Spuren arbeitet, um möglichst viele Daten auf den Magnetschichten unterbringen zu können. Magnetische Signale werden aber immer schwächer, je kleiner der Magnet bzw. die magnetisierte Fläche ist, von denen sie ausgehen. Das bedeutet, daß die Signale einer Harddisk um ein Vielfaches schwächer als die einer Diskette sind. Als Folge davon muß der Tonkopf filigraner, die Positionierung genauer und überhaupt die ganze Mechanik präziser sein. Außerdem streuen Magnetfelder, je weiter sie sich durch den Raum ausbreiten. Streuen bedeutet aber, daß unter Umständen Daten auf Nachbarspuren gestört werden. Sie merken schon, Festplatten sind besonders empfindliche Geräte, die besonderer Behandlung bedürfen.

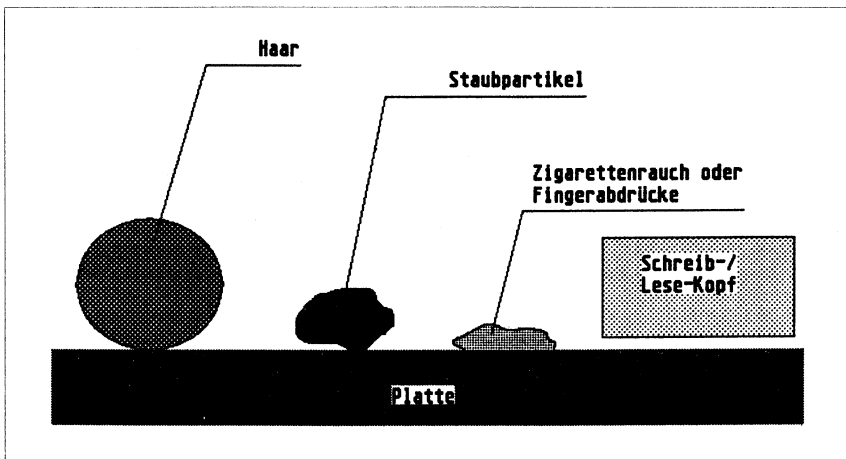


Abb. 3.8: Die Ursachen eines Headcrash

Das fängt schon mit dem Transport an. Sie dürfen eine solche Festplatte nicht einfach von Ihrem Schreibtisch nehmen und sie z.B. einer weiten Reise über

die Autobahn aussetzen. Rüttelbewegungen jeder Art können den sogenannten Headcrash verursachen. Dazu muß man wissen, daß die Schreib-/Lese-Köpfe die Platten nicht berühren, sondern in einem wahnwitzig geringen Abstand auf einem Luftpolster über sie hinwegfliegen, d.h. eigentlich dreht sich die Platte unter ihnen durch. Dieser Abstand ist so gering, daß sich z.B. der Nikotin aus Zigarettenrauch oder sogar ein Fingerabdruck als störendes Hindernis bemerkbar macht. Aus diesem Grund sind Festplatten auch immer hermetisch gekapselt und die eintretende Luft wird durch spezielle Filter gereinigt. Sie werden sich unschwer vorstellen können, daß Staub auf der Plattenoberfläche oder ein Aufschlagen der Köpfe durch Rüttelbewegungen tiefe Furchen in die hauchdünne Magnetschicht reißen kann - der sogenannte Headcrash ist da. Folge dieser Beschädigungen ist fast immer der Verlust der betroffenen Datensektoren. Das kann bei normalen Sektoren recht glimpflich ausgehen, es ist dann eben nur eine Datei zerstört. Die weitaus meiste Zeit verbringen die Schreib-/Lese-Köpfe aber über den Sektoren, die das Inhaltsverzeichnis speichern. Sind diese zerstört, so kann man meistens überhaupt nichts mehr mit der Platte anfangen, die Daten sind unwiederbringlich verloren.

Gegen Fehler kann man vor allem eines tun: Behandeln Sie Ihre Datenträger gut. Das bedeutet bei Disketten, daß diese staubfrei aufbewahrt werden (am besten in einem speziellen Diskettenkasten) und daß Sie nicht an ihnen herummanipulieren. Auf keinen Fall sollten Sie die Magnetschicht unter dem Schutzschieber berühren, denn Ihre Fingerabdrücke können bereits Datenverluste auslösen. Bei Festplatten bedeutet Vorbeugung, daß Sie das Gerät niemals transportieren, ohne die Köpfe vorher geparkt zu haben. Außerdem sollten Sie nicht versuchen, die Harddisk zu öffnen. Achten Sie außerdem darauf, daß Ihre Datenträger keinen magnetischen Feldern ausgesetzt werden. Diese gehen vor allem von Lautsprechern, Rasierapparaten und ähnlichen Geräten aus.

Wenn Ihre Daten einmal zerstört sein sollten (siehe unten), kommt es vor allem darauf an, noch zu retten, was zu retten ist. Merken Sie sich zunächst genau, in welcher Situation der Fehler aufgetreten ist. Trat der Datendefekt beim Lesen oder Schreiben einer Datei auf, so können Sie Glück im Unglück haben. Meistens ist dann nur diese eine Datei betroffen, von der vielleicht noch eine BAK-Version existiert. Versuchen Sie nun, alle anderen Dateien dieser Datenstation auf eine andere Diskette zu kopieren. Das kann bei Festplatten in eine ziemliche Arbeit ausarten, ist aber notwendig, um den totalen Datenverlust zu vermeiden, falls der Datenträger defekt ist.

Etwas gefährlicher ist es schon, wenn der Fehler beim Öffnen eines Ordners auftrat. Das bedeutet, daß das Inhaltsverzeichnis Ihrer Diskette oder Festplatte

defekt ist - einer der schlimmsten denkbaren Fälle. Versuchen Sie in diesem Fall, alle anderen Ordner zu öffnen. Sofern das möglich ist, sichern Sie die Dateien auf eine frisch formatierte Diskette. Dateien, die in nicht zu öffnenden Ordnern liegen, können Sie abschreiben.

Erst wenn Sie Ihre Dateien so in Sicherheit gebracht haben, dürfen Sie mit der Fehlersuche beginnen.

Pannenhilfe:



Nach dem Anklicken der Datenstation-Icons werden falsche oder unsinnige Dateinamen im Fenster angezeigt.



Je nachdem, auf welchem Gerät der Fehler aufgetreten ist, kann er unterschiedliche Ursachen und Wirkungen haben. Handelt es sich um ein Verzeichnis auf einer Diskette (besonders häufig sind Ordner von diesem Problem betroffen), so ist das meist harmlos. Drücken Sie bitte zunächst die **Esc**-Taste, um das Laden des Verzeichnisses erneut zu erzwingen. Bringt das keine Änderung und sind auch die Verbindungskabel in Ordnung, schließen Sie das Verzeichnis und öffnen es dann bitte erneut. Sollte der Fehler jetzt noch nicht behoben sein, ist die Diskette defekt. Sie müssen sie neu formatieren. (Nicht ohne vorher möglichst viele Dateien gerettet zu haben.). Es kann durchaus sein, daß diese Diskette dann nie wieder Probleme macht.

Es könnte auch sein, daß Sie gerade mit einer alten Betriebssystemversion versuchen, Ordner zu lesen, die mit einer neueren Version erzeugt wurden. In diesem Fall hilft nur eines: Mit einer anderen Systemdiskette oder mit einer frisch formatierten Disk neu booten. Dann müßte das Problem behoben sein.

Etwas anders liegt der Fall, wenn der genannte Fehler auf einer Harddisk auftritt. Starten Sie bitte HDX.PRG, und überprüfen Sie die Partitionierungswerte. Sind diese richtig, können Sie erneut booten. Ist der Fehler nicht verschwunden, so ist das Gerät defekt. Waren die Partitionierungswerte falsch eingetragen, so müssen Sie die Festplatte neu formatieren; Ihre Daten sind leider verloren. Zeigt auch das keinen Erfolg, müssen Sie den Sektor 0 vom Laufwerk C mit Nullen überschreiben (ein Programm dafür finden Sie in Kapitel 3.4.2.) und dann erneut formatieren. Sollte sich wiederum nichts ändern, so ist das Gerät defekt.



Ein Zugriff wird mit der Meldung "Diskette in Laufwerk x: ist schreibgeschützt..." quittiert.



Es wird versucht, eine Datei zu löschen oder zu ändern, während der Schreibschutzschieber an der Diskette nach oben gedrückt ist. Setzen Sie den Schieber so um, daß das Loch in der Ecke der Diskette geschlossen wird. Falls Sie aber nicht möchten, daß die Daten auf der Diskette verändert werden, wählen Sie bitte ABRUCH.



Beim Zugriff auf eine Datei erscheint die Meldung "Floppy x: antwortet nicht. Bitte überprüfen und eine Disk einlegen."



Sie haben mit großer Wahrscheinlichkeit vergessen, eine Diskette in das angesprochene Laufwerk einzulegen. Holen Sie dies bitte nach. Wenn Sie das erledigt haben, können Sie "Weiter" anklicken; der Zugriff müßte dann in der gewohnten Weise ablaufen. Andere Möglichkeit: Das Laufwerk war nicht eingeschaltet oder die Stromversorgung nicht richtig eingesteckt. Trifft keine der genannten Möglichkeiten zu, ist mit großer Wahrscheinlichkeit das Gerät defekt und muß repariert werden. In diesem Fall sollten Sie Abbruch anklicken, um weitere Zugriffsversuche zu verhindern.



Es erscheint die Meldung "Diese Anwendung kann das angesprochene Objekt nicht finden."



Sie versuchen, eine Datei anzusprechen, die auf dem angesprochenen Laufwerk nicht existiert. Dies kann daher rühren, daß Sie eine andere Diskette eingelegt haben als das Desktop anzeigt. Oder: Sie arbeiten mit einem Anwendungsprogramm, und dieses erwartet eine andere Diskette als die eingelegte. Sie haben nach dem Erscheinen der Fehlermeldung die Möglichkeit, die richtige Diskette zu suchen und einzulegen. Wenn das erledigt ist, drücken Sie OK, und das Programm fährt fort oder bricht die Aktion ab.



Auf dem Bildschirm erscheint eine Meldung wie "Daten auf Disk x: defekt. Überprüfen Sie bitte die Verbindung!"



Das Überprüfen der Verbindungskabel ist in der Tat eine gute Idee. Es ist leicht möglich, daß einer der Stecker aus den Buchsen gerutscht ist und nun die Daten verstümmelt im ST ankommen. Als nächstes sollten Sie sich das Inhaltsverzeichnis des Laufwerks mal mit der Option "als Text" anzeigen lassen. Sofern die angesprochene Datei die Länge 0 aufweist, ist sie schon beim Abspeichern nicht korrekt auf die Diskette geschrieben worden. Löschen Sie diese 0-Byte-Datei, und versuchen Sie bitte, diese Datei erneut auf die Diskette zu bringen (z.B. durch Kopieren). In einigen Fällen behebt dies

den Fehler. Ansonsten ist es möglich, daß der Datenträger defekt ist. Bei Disketten gehen Sie bitte wie gewohnt vor: Dateien retten, soweit möglich, und dann die Diskette formatieren. Bei Festplattenlaufwerken ist das schon schwieriger. Auf jeden Fall sollten Sie auch hier die Dateien retten. Dann rufen Sie bitte das Programm HDX.PRg auf, und starten Sie die Funktion "Markbad" für die betroffene Partition. Dabei können einige Dateien zerstört werden. Falls der Fehler danach immer noch auftritt oder die gesamte Platte wie leergefegt ist, müssen Sie das Gerät zur Reparatur bringen (Kopf parken nicht vergessen).



Die meisten Anwendungsprogramme geben eigene Fehlermeldungen aus, die nicht mit denen des Desktop übereinstimmen müssen. Oft sagt der Text aber sinngemäß das gleiche aus, so daß eine korrekte Diagnose recht einfach ist. Manche Programme geben aber auch nur Fehlernummern aus, die oft ein negatives, teilweise aber auch ein positives Vorzeichen haben können. Hier ein paar Hinweise:

- 2 Gerät nicht bereit
- 4 Prüfsumme falsch
- 6 Spur nicht gefunden
- 7 falscher Boot-Sektor
- 8 Sektor nicht gefunden
- 10 Schreibfehler
- 11 Lesefehler
- 13 Diskette schreibgeschützt
- 14 Diskette wurde gewechselt
- 15 unbekanntes Gerät
- 16 Fehler beim Überprüfen eines Sektors
- 17 andere Diskette einlegen
- 33 Datei nicht gefunden
- 34 Pfadname nicht gefunden
- 35 zuviele Dateien offen
- 36 Zugriff nicht möglich
- 46 ungültige Laufwerksbezeichnung
- 66 keine Binärprogrammdatei

3.6 Größenangaben

Die Größenangaben, die der ST in bezug auf Dateien und Disketten macht, sind mit Vorsicht zu genießen. Nicht immer geben Sie ein korrektes Abbild der Situation.

Wählen Sie zu einem Laufwerk (auch Festplatte oder RAM-Disk) den Menüpunkt "zeige Info..." im Desktop. In der untersten Zeile der Dialogbox wird dann die Anzahl freier Bytes angegeben.

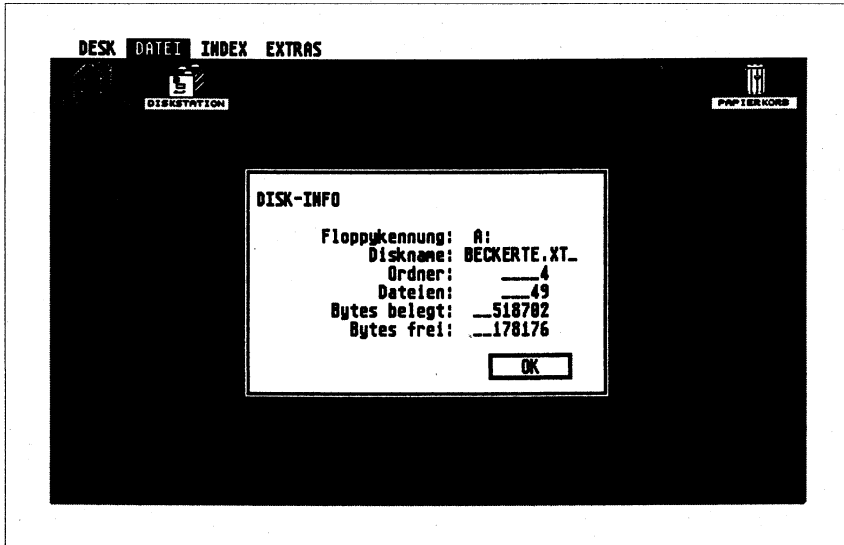


Abb. 3.9: Die Angabe freier Bytes zu einer Harddisk-Partition

Pannenhilfe:

➡ Die Info hat ergeben, daß auf der Diskette gerade noch genug Platz vorhanden ist, um ein paar Dateien hinzuzukopieren. Doch bei der Ausführung dieser Operation erscheint die Meldung, daß die Diskette voll ist.

✓ Hier liegt kein Fehler im üblichen Sinne vor. Auf der Diskette ist wirklich nicht genug Platz. Ursache: Auch für eine Datei, die nur 100 Bytes umfaßt, müssen 1024 Bytes auf der Diskette vergeben werden. Ebenso kann der Speicherplatz der Diskette auch bei größeren Dateien immer nur in Schritten von 1024 Bytes vergeben werden (also z.B. 16384 Bytes für eine Datei mit 15500 Zeichen). Wenn Sie nun mehrere Dateien auf eine Diskette kopieren, entsteht durch diese schrittweise Vergabe ein gewisser Überhang, der zusätzlich verbraucht wird. Abhilfe ist daher nicht möglich.

☐ 1024 Bytes entsprechen genau zwei Sektoren. Eine Gruppe von zwei Sektoren nennt man auch Cluster (engl.: Haufen, Ansammlung). Wie andernorts bereits erwähnt, merkt sich der ST die Lage einer Datei auf der

Diskettenoberfläche durch die Angabe von Spur und Sektor. Genauer muß es eigentlich heißen: durch die Angabe des Clusters. Faßt man jeweils zwei Sektoren zu solch einem Cluster zusammen, so braucht das Inhaltsverzeichnis weniger Platz, weil nicht so viele verschiedene Einträge unterschieden werden müssen. Im Inhaltsverzeichnis (genauer in der File-Allocation-Table, kurz FAT) steht daher für jede Datei, welche Cluster sie belegt. In der FAT kann aus verständlichen Gründen für jeden Cluster nur ein Eintrag gemacht werden, also kann er auch nur durch einen File belegt werden, selbst wenn dieser nur 100 Bytes davon beanspruchen sollte. Die restlichen Bytes dieses Clusters stehen also - obwohl unbenutzt - keiner anderen Datei zur Verfügung.

Die Angabe der belegten Bytes in der Info-Anzeige gibt an, wieviele Bytes der gesamte Inhalt der Diskette umfaßt. Dabei sind die reinen Dateilängen gemeint und nicht die wirklich belegten Bytes der Diskette. Daher kann sich bei der Addition von belegten und freien Bytes ein anderer Wert als die wirkliche Kapazität der Diskette ergeben.

Durch die Byteanzahl, die im Desktop-Fenster angegeben wird, erfahren Sie, wieviele Bytes durch alle sichtbaren Dateien in diesem Fenster belegt werden. Dateien, die in Ordnern liegen, werden dabei nicht mitgezählt, ebenso die Ordner selbst. Sind in einem Fenster also nur 5 Ordner dargestellt, so lautet die Meldung: 0 Bytes in 5 Dateien, auch wenn in den Ordnern vielleicht hunderttausende von Zeichen belegt sind.

Sie möchten sicher auch erfahren, wieviele Bytes ein Ordner wirklich belegt. Diesen Wert erhalten Sie, wenn Sie sich die Info zu diesem Ordner anzeigen lassen. Diesmal sind auch die nicht sichtbaren Dateien mitgezählt!

Größeres Vertrauen dürfen Sie der Dateilänge in Fenstern (bei Anzeige als Text) bzw. in der Info zu einer Datei schenken. Hier steht die echte Dateilänge, d.h. die Anzahl Bytes, die Sie wirklich nutzen können.



Wollen Sie wissen, wieviele Bytes durch diese Datei wirklich besetzt werden, so müssen Sie die Dateilänge auf das nächste volle Vielfache von 1024 aufrunden. Beispiele:

```
Dateilänge= 1000 -> belegte Bytes: 1024  
Dateilänge= 100000 -> belegte Bytes: 100352
```

Pannenhilfe:



Zu einer Datei wird die Länge "0 Bytes" angezeigt.

✓ Diese Meldung kann zwei Ursachen haben. Entweder ist die Datei wirklich leer und hat deshalb die Länge 0, oder die Datei wurde nicht korrekt erzeugt (z.B. durch einen Schreibfehler). In diesem Fall setzt der ST die Dateilänge von sich aus auf 0, um Zugriffe auf nicht existente oder zerstörte Daten zu vermeiden. Diese fehlerhafte Erzeugung von Dateien kann auf Beschädigungen der Diskette zurückgeführt werden. Eine andere Möglichkeit: Sie haben die Diskette aus dem Laufwerk genommen, während das rote Lämpchen brannte und der ST versuchte, die Datei zu erzeugen.

3.7 Datensicherung - ein ernstes Thema

Datensicherung (auch Backup genannt) ist besonders für Benutzer einer Festplatte wichtig. Fehler auf Festplatten haben die unangenehme Eigenschaft, daß sie immer einen sehr großen Datenbestand von 20, 40 oder gar 60 Megabyte betreffen, während ein Fehler auf einer einzelnen Diskette "nur" 720 Kilobytes lahmlegen kann. Deshalb sei Festplattenbesitzern hier eindringlich empfohlen, zumindest regelmäßig Backups zu machen. Ist ein Head-Crash (siehe oben) erst einmal aufgetreten, dann kann man oft nicht mehr viel retten. Anwender, die ohne Festplatte arbeiten, sollten sich aber nicht in Sicherheit wiegen. Stellen Sie sich vor, Sie schreiben Ihre Diplom-Arbeit mit dem ST und erleiden kurz vor Fertigstellung einen Diskettenschaden! Hier heißt es ebenfalls: Jeden Tag die Textdisketten sichern.

Im einzelnen bedeutet Backup, daß Sie von Ihren wichtigen Dateien immer eine Kopie mit dem aktuellen Stand besitzen. Dafür gibt es mehrere Wege. Der einfachste ist, regelmäßig die gefährdeten Dateien auf eine andere Diskette zu kopieren.

Wichtig ist: Wenn irgendwie möglich, arbeiten Sie bitte auf keinen Fall mit den Originaldisketten des Softwareherstellers. Diese gehören an einen sicheren Ort (Schrank, Tresor etc.). Bevor Sie ein Programm benutzen, sollten Sie davon eine sogenannte Arbeitskopie ziehen, d.h. Sie erstellen eine 1:1-Kopie von der Diskette (siehe Kapitel 2). Es ist wichtig, daß solche Disketten 1:1 kopiert werden! Manche Softwarepakete verlangen allerdings ein etwas anderes Vorgehen (z.B. zuerst Eintragen von Benutzername und Anschrift). Dies ist dann in der Anleitung beschrieben.

Sie können sich dabei nicht auf die BAK-Versionen Ihrer Dateien verlassen. Wenn eine Festplatte oder Diskette zerstört wird, dann meist so gründlich, daß auch die BAK-Dateien in Mitleidenschaft gezogen werden. Der Sinn dieser BAK-Dateien ist es, Sie vor Datenverlusten durch Fehlbedienungen zu schützen.

Ist das sogenannte Generationenprinzip besser?

Dabei handelt es sich um eine Erweiterung des BAK-Mechanismus, die Sie allerdings per Hand erledigen müssen. Bei sehr wichtigen Dateien sollten Sie von Zeit zu Zeit eine neue Generation beginnen, d.h. den aktuellen Stand der Arbeit abspeichern. Dabei geben Sie für jede neue Version einen anderen Namen an, z.B. PROJEKT1.TXT, PROJEKT2.TXT usw. Sollten Sie durch Fehlbedienungen Ihre Arbeit zerstören, haben Sie immer noch die älteren Versionen. Nachteil: Auf diese Weise sammeln sich schnell viele veraltete Daten auf der Diskette oder Festplatte, die Sie von Zeit zu Zeit löschen müssen (aber bitte nicht voreilig).

Wie oft Sie Ihre Daten sichern, hängt von vielen Faktoren ab, zum Beispiel auch vom persönlichen Arbeitsaufwand und der Datenmenge. Falls Sie jeden Tag mit dem ST arbeiten und dabei große Datenmengen produzieren, ist es ratsam, zumindest die an diesem Tag hinzugekommenen bzw. geänderten Dateien an jedem Abend zu kopieren. Das gleiche gilt für Besitzer einer Festplatte.

Zunächst einmal ist es nicht notwendig, daß Sie bei jeder Datensicherung alle Dateien kopieren. Es reicht, dies einmal im Monat (bzw. in längeren Abständen bei geringerer Beanspruchung) zu tun. In kürzeren Abständen werden dagegen nur die geänderten Dateien gesichert. Manche Anwender empfehlen auch eine Gesamt-Sicherung der Harddisk vor jedem Transport.

Nun ist das Kopieren von Dateien auch mit dem ST und seinem grandiosen GEM ein recht mühsames Unterfangen. Zum Glück kann man dies auch einem Programm überlassen. Für Festplatten gibt es schon seit einiger Zeit spezielle Programme, die das Backup besonders schnell durchführen. Außerdem haben diese die angenehme Eigenschaft, die Dateien zu komprimieren, so daß Sie mit weniger Disketten auskommen können. Für die normale Sicherung aller Dateien einer 60-MByte-Platte können Sie einen Diskettenstapel von bis zu 30 cm Höhe brauchen. Bei sehr großen Datenmengen ist dann auch die Anschaffung eines Tape-Streamers empfehlenswert. Das ist eine Art Tonband-Gerät, das in der Lage ist, große Mengen von Daten auf sehr kleinem Raum zu speichern. Allerdings sind diese Geräte recht teuer.

Anwender von Diskettenstationen werden im Stich gelassen, was die Datensicherung betrifft. Aus diesem Grunde ist unten ein Programm abgedruckt, daß von einer Diskette alle Files kopiert, die nach einem bestimmten Datum erstellt worden sind. Sinn dieser Aktion: Kopiert man alle Dateien, die nach dem letzten Backup entstanden sind, so werden automatisch immer die aktuellsten Daten gesichert. Allerdings geht das Programm davon aus, daß auf der Diskette genug Platz für alle geänderten Daten ist. Eine weitere Voraussetzung

für die Anwendung des untenstehenden Programms ist, daß Sie bei jedem Systemstart die Uhr und das Datum des ST richtig stellen. Besitzer eines Mega-ST haben es da leichter, in diesen Rechnern ist eine batteriegepufferte Uhr eingebaut, die nur einmal gestellt werden muß und dann so lange die richtige Zeit anzeigt, bis die Batterien leer sind.

Außerdem sind zwei Fälle zu unterscheiden. Falls Sie eine Harddisk oder zwei Diskettenlaufwerke besitzen, können Sie direkt von einer Station zur anderen sichern. Für Sie ist das Programm "Direkt-Backup" gedacht. Aus technischen Gründen liegen die beiden Programme nur als GFA-BASIC-Versionen vor.

Sollten Sie nur ein Laufwerk besitzen, kann das Kopieren nicht so einfach durchgeführt werden, da Sie zwischendurch die Diskette wechseln müssen. Der Ausweg aus dem Dilemma: Die zu kopierenden Dateien werden auf einer RAM-Disk zwischengelagert. Besitzen Sie also nur eine Diskettenstation, so müssen Sie das Programm "RAM-Backup" in Verbindung mit einer RAM-Disk benutzen.

```
' AUTOBACK (Version für GFA-BASIC)
' Programm für automatisches Backup einer Diskette
'
' Alten DTA-Buffer sichern
D%=Gendos(&H2F)
' Quelldiskette anfordern
Print "Bitte legen Sie die Quelldiskette in Laufwerk A ein!"
Print "Weiter mit beliebiger Taste."
Print
Void Inp(2)
' Letztes Backup-Datum erfragen
Print "Bitte das Datum des letzten Backup eingeben:"
Input "Tag   ";Tag%
Input "Monat ";Monat%
Input "Jahr  ";Jahr%
Print
' Daten kodieren
Bdatum%=Tag%+Monat%*32+(Jahr%-1980)*512
' Alle relevanten Dateien auf RAM-Disk kopieren
Chdrive 1
Gosub Dir_suche("A:", "")
' Zieldiskette anfordern
Print
Print "Bitte legen Sie die Zieldiskette in Laufwerk A ein!"
Print "Weiter mit beliebiger Taste."
Print
Void Inp(2)
' Alle Dateien von RAM-Disk nach Laufwerk A kopieren
Chdrive 4
Gosub Dir_suche("D:", "")
Print
```

```
Print "Backup beendet!"
' Alte Einstellungen zurück
Chdrive 1
Void Gemdos(&H1A,L:D%)
```

```
-----
Procedure Dir_suche(L$,F$)
' Sucht alle Dateien im angegebenen Ordner F$ auf Laufwerk L$,
' die zum letzten Backup-Datum passen und läßt andere Ordner
' rekursiv durchsuchen
Local A$,F2$,Dta$,Adr%,A%
' lokalen DTA-Buffer für dieses Verzeichnis reservieren
Dta$=Space$(44)
Adr%=Varptr(Dta$)
Void Gemdos(&H1A,L:Adr%)
' Suche starten
F2$=F$+"\\*.*"+Chr$(0)
A%=Gemdos(&H4E,L:Varptr(F2$),&H10)
' ggf. kopieren
Gosub Vergleich
' restliche Files in diesem Ordner untersuchen
Repeat
' DTA-Buffer löschen
For I=Adr%+30 To Adr%+43
Poke I,32
Next I
' nächsten Eintrag suchen
A%=Gemdos(&H4F)
' ggf. kopieren
Gosub Vergleich
' Ist die gefundene Datei ein Ordner ?
If (Peek(Adr%+21)=16) And (A%=0) And (A$<>"..") Then
' ja -> Ordner durchsuchen
Gosub Dir_suche(L$,F$+"\\ "+A$)
' nach Rückkehr alte DTA wieder herstellen
Void Gemdos(&H1A,L:Adr%)
Endif
Until A$<>0
Return
-----
```

```
Procedure A_nach_d(Fm$,Am$)
' Kopiert Datei (Fm$+"\\ "+Am$) von Laufwerk A in die RAM-Disk
Local Pointer%,Dateilaenge%,Anz%
' Wenn Ordner in RAM-Disk nicht existiert, dann erzeugen
Gosub Ordner_erb("D:",Fm$)
' Dateien öffnen und übertragen
Open "I",#1,"A:"+Fm$+"\\ "+Am$
Open "O",#2,"D:"+Fm$+"\\ "+Am$
Pointer%=0
Dateilaenge%=Lpeek(Adr%+26)
While Not Eof(#1)
Anz%=Min(512,Dateilaenge%-Pointer%)
Print #2,Input$(Anz%,#1);
Pointer%=Pointer%+512
```

```

Wend
Close #1
Close #2
Return

```

```

-----
Procedure D_nach a(Fm$,Am$)
' Kopiert Datei (Fm$+"\ "+Am$) von RAM-Disk nach Laufwerk A
' und löscht dabei ggf. alte Backups
Local Pointer%,Dateilaenge%,Anz%
' Wenn Ordner in Laufwerk A nicht existiert, dann erzeugen
Gosub Ordner_erb("A:",Fm$)
' Testen, ob schon ein altes Backup vorliegt
If Exist("A:"+Fm$+"\ "+Am$) Then
' wenn ja -> löschen
Kill "A:"+Fm$+"\ "+Am$
Endif
' Dateien öffnen und übertragen
Open "I",#1,"D:"+Fm$+"\ "+Am$
Open "O",#2,"A:"+Fm$+"\ "+Am$
Pointer%=0
Dateilaenge%=Lpeek(Adr%+26)
While Not Eof(#1)
Anz%=Min(512,Dateilaenge%-Pointer%)
Print #2,Input$(Anz%,#1);
Pointer%=Pointer%+512
Wend
Close #1
Close #2
Return

```

```

-----
Procedure Ordner_erb(Lw$,Fo$)
' testet, ob der Ordner Fo$ auf dem in Lw$ angegebenen Laufwerk
' existiert und erzeugt ihn nötigenfalls
Local Dta$,F2$,Dta%,Fc%
' neuen DTA-Buffer für diese Prozedur reservieren
Dta%=Gendos(&H2F)
Dta%=Space$(44)
Void Gendos(&H1A,L:Varptr(Dta%))
' Suche starten
F2$=Lw$+Fo$+"\ *.*"+Chr$(0)
Fc%=Gendos(&H4E,L:Varptr(F2%),&H10)
If (Fc%<0) And (Len(Fo$)>0) Then
' Ordner existiert nicht, also erzeugen
Mkdir Lw$+Fo$
Endif
' alten DTA-Buffer wiederherstellen
Void Gendos(&H1A,L:Dta%)
Return

```

```

-----
Procedure Vergleich
' testet, ob eine Datei kopiert werden muß und veranlaßt dies ggf.
' File-Namen aus Buffer lesen
A$=""

```



```

For I=Adr%+30 To Adr%+43
  Exit If Peek(I)<=32
  A$=A$+Chr$(Peek(I))
Next I
' Datum holen und vergleichen
Datum%=Dpeek(Adr%+24)
If (Peek(Adr%+21)<>16) And (A$<>"") Then
  ' nur korrekte File-Namen und keine Ordner bearbeiten
  If (Bdatum%<Datum%) And (L$="A:") Then
    ' Neue Datei Laufwerk A -> kopieren
    Print "Kopiere Datei ";F$+"\ "+A$
    Gosub A_nach_d(F$,A$)
  Endif
  If (L$="D:") Then
    ' Datei von RAM-Disk auf Laufwerk A kopieren, kein Datumstest
    Print "Sichere Datei ";F$+"\ "+A$
    Gosub D_nach_a(F$,A$)
  Endif
Endif
Return

```

○ Wenn Sie mit RAM-Backup arbeiten wollen, brauchen Sie eine RAM-Disk, die als Laufwerk D angemeldet ist und genug Platz bietet, um alle seit dem letzten Backup geänderten Dateien aufzunehmen. Vor dem Programmstart muß die RAM-Disk leer sein. Erfüllt Ihr Rechner diese Voraussetzungen, so können Sie RAM-Backup starten. Zunächst werden Sie gebeten, die sogenannte Quelldiskette einzulegen. Das ist die Diskette, von der ein Backup gemacht werden soll. Wenn Sie das erledigt haben, drücken Sie bitte eine beliebige Taste. Als nächstes möchte das Programm von Ihnen das Datum der letzten Datensicherung wissen. Noch einmal der Hinweis: Das Backup kann nur dann richtig funktionieren, wenn Sie bei jedem Booten die Uhr richtig stellen, falls Sie eine batteriegepufferte Uhr besitzen, können Sie das natürlich vergessen. Sobald Sie das Datum eingegeben haben, beginnt der Rechner damit, alle Dateien neueren Datums auf die RAM-Disk zu kopieren. Sie können das am Aufleuchten der roten Lampe am Laufwerk erkennen.

Je nach Umfang der Daten kann dieser Vorgang einige Zeit in Anspruch nehmen. Während dieser Tätigkeit zeigt der Rechner auf dem Bildschirm die Namen der Dateien an, die gerade in die RAM-Disk übertragen werden ("Kopiere ..."). Nach einiger Zeit verlangt der ST dann die sogenannte Zieldiskette, also die Scheibe, auf die die geänderten Daten gesichert werden sollen. Nach dem Einlegen drücken Sie auch hier bitte wieder eine beliebige Taste. Das Programm beginnt dann damit, alle Dateien aus der RAM-Disk auf die Zieldiskette zu sichern und zeigt das auch an ("Sichere ..."). Hier das Listing für das Sichern mit zwei Laufwerken:

```

' D D BACK
' Programm für automatisches Backup mit mindestens zwei Laufwerken
'
' Alten DTA-Buffer sichern
D%=Gemdos(&H2F)
' Disketten anfordern
Print "Bitte geben Sie das Quellaufwerk an!"
Q$=Upper$(Input$(1))
Print Q$
Print
Print "Bitte geben Sie das Ziellaufwerk an!"
Print "Sorgen Sie bitte ggf. dafür, daß die Disketten eingelegt sind."
Z$=Upper$(Input$(1))
Print Z$
Print
' Letztes Backup-Datum erfragen
Print "Bitte das Datum des letzten Backup eingeben:"
Input "Tag   ";Tag%
Input "Monat ";Monat%
Input "Jahr  ";Jahr%
Print
' Daten kodieren
Bdatum%=Tag%+Monat%*32+(Jahr%-1980)*512
' Alle relevanten Dateien kopieren
Chdrive (Asc(Q$)-Asc("A"))
Gosub Dir_suche(Q$+" ":" ")
Print
Print "Backup beendet!"
' Alte Einstellungen zurück
Chdrive 1
Void Gemdos(&H1A,L:D%)
' -----
Procedure Dir_suche(L$,F$)
' Sucht alle Dateien im angegebenen Ordner F$ auf Laufwerk L$,
' die zum letzten Backupdatum passen und läßt andere Ordner
' rekursiv durchsuchen
Local A$,F2$,Dta$,Adr%,A%
' lokalen DTA-Buffer für dieses Verzeichnis reservieren
Dta$=Space$(44)
Adr%=Varptr(Dta$)
Void Gemdos(&H1A,L:Adr%)
' Suche starten
F2$=F$+"\*.**"+Chr$(0)
A%=Gemdos(&H4E,L:Varptr(F2$),&H10)
' ggf. kopieren
Gosub Vergleich
' restliche Files in diesem Ordner untersuchen
Repeat
' DTA-Buffer löschen
For I=Adr%+30 To Adr%+43
    Poke I,32
Next I
' nächsten Eintrag suchen

```

```

A%=Gemdos(&H4F)
' ggf. kopieren
Gosub Vergleich
' Ist die gefundene Datei ein Ordner ?
If (Peek(Adr%+21)=16) And (A%=0) And (A$<>"..") Then
' ja -> Ordner durchsuchen
Gosub Dir_suche(L$,F$+"\ "+A$)
' nach Rückkehr alte DTA wieder herstellen
Void Gemdos(&H1A,L:Adr%)
Endif
Until A$<>0
Return

```

```

-----
Procedure A_nach_d(Fm$,Am$)
' Kopiert Datei (Fm$+"\ "+Am$) von Laufwerk q$ nach z$
Local Pointer%,Dateilaenge%,Anz%
' Wenn Ordner in Ziel-Disk nicht existiert, dann erzeugen
Gosub Ordner_erz(Z$+".",Fm$)
' Dateien öffnen und übertragen
Open "I",#1,Q$+"."+Fm$+"\ "+Am$
Open "O",#2,Z$+"."+Fm$+"\ "+Am$
Pointer%=0
Dateilaenge%=Lpeek(Adr%+26)
While Not Eof(#1)
  Anz%=Min(512,Dateilaenge%-Pointer%)
  Print #2,Input$(Anz%,#1);
  Pointer%=Pointer%+512
Wend
Close #1
Close #2
Return

```

```

-----
Procedure Ordner_erz(Lw$,Fo$)
' testet, ob der Ordner Fo$ auf dem in Lw$ angegebenen Laufwerk
' existiert und erzeugt ihn nötigenfalls
Local Dta$,F2$,Dta%,Fc%
' neuen DTA-Buffer für diese Prozedur reservieren
Dta%=Gemdos(&H2F)
Dta$=Space$(44)
Void Gemdos(&H1A,L:Varptr(Dta$))
' Suche starten
F2$=Lw$+Fo$+"*.*"+Chr$(0)
Fc%=Gemdos(&H4E,L:Varptr(F2$),&H10)
If (Fc%<0) And (Len(Fo$)>0) Then
' Ordner existiert nicht, also erzeugen
Mkdir Lw$+Fo$
Endif
' alten DTA-Buffer wiederherstellen
Void Gemdos(&H1A,L:Dta%)
Return

```

```

-----
Procedure Vergleich
' testet, ob eine Datei kopiert werden muß und veranlaßt dies ggf.

```

```

' File-Namen aus Buffer lesen
A$=""
For I=Adr%+30 To Adr%+43
  Exit If Peek(I)<=32
  A$=A$+Chr$(Peek(I))
Next I
' Datum holen und vergleichen
Datum%=Dpeek(Adr%+24)
If (Peek(Adr%+21)<>16) And (A$<>"") Then
  ' nur korrekte File-Namen und keine Ordner bearbeiten
  If (Bdatum%<Datum%) Then
    ' Neue Datei kopieren
    Print "Kopiere Datei ";F$+"\"+A$
    Gosub A_nach_d(F$,A$)
  Endif
Endif
Return

```

○ Direkt-Backup verlangt das Vorhandensein zweier beliebiger Laufwerke am ST. Dies können z.B. eine Harddisk und eine Diskettenstation, im Extremfall aber auch eine RAM-Disk und eine Diskettenstation sein. Nach dem Start dieses Programms werden Sie zunächst nach der Kennung (ein Buchstabe) des Quellaufwerks gefragt (also der Station, deren Daten gesichert werden sollen). Bitte tippen Sie den Buchstaben (z.B. C für die Harddisk, A für die erste Diskettenstation) ein. Groß- und Kleinschreibung spielen keine Rolle. Im nächsten Schritt müssen Sie in gleicher Weise die Kennung des Ziellaufwerks eingeben. Bitte achten Sie darauf, daß die beiden Laufwerke - wenn nötig - die richtigen Disketten beinhalten. Sind die beiden Laufwerke dem Programm bekannt, werden Sie noch nach dem Datum des letzten Backup gefragt. Sobald Sie diese Eingabe abgeschlossen haben, beginnt das Programm damit, die geänderten Files auszusortieren und auf das Ziellaufwerk zu kopieren. Jede so kopierte Datei wird auch mit Namen angezeigt. Auch dieses Programm geht davon aus, daß auf der Ziel-Diskette genug Platz für alle zu sichernden Daten ist.

Bei der regelmäßigen Benutzung der BACKUP-Programme hat sich folgendes Vorgehen als nützlich erwiesen. Reservieren Sie zu jeder wichtigen Datendiskette eine weitere Backup-Diskette. Jeden Tag (am besten Abends) starten Sie dann das Programm und sichern immer auf die gleiche Diskette. Da alte Dateien auf der Backup-Diskette, die überholt sind, automatisch durch die neuesten Versionen ersetzt werden, haben Sie auf der Sicherungsdiskette nach dem Backup immer den neuesten Stand. Wenn Sie ganz sicher gehen wollen, bewahren Sie Original und Kopie immer an getrennten Orten auf, so daß im Fall der Fälle nur eine Version zerstört würde.



Sollten Sie die Datensicherung nicht jeden Tag betreiben, hier noch ein kleiner Tip. Notieren Sie nach jedem Backup das Tagesdatum in einer Ecke des Diskettenaufklebers (z.B. mit Bleistift, damit man es wieder entfernen kann), dann können Sie es bei Bedarf sofort ablesen.

4. Drucker

Neben den Massenspeichern haben wir hier wohl das wichtigste Zubehör vor uns. Drucker gibt es in verschiedenen Ausführungen, und leider hat auch jedes Modell so seine Eigenheiten. Aus diesem Grunde ist es wohl kaum möglich, allgemeingültige Patentrezepte für die Bedienung der "Papierspucker" zu geben.

4.1 Die Hardware

Man unterscheidet diverse Typen von Druckern. Je nach der Art, wie sie die Buchstaben zu Papier bringen, spricht man von Nadel-, Tintenstrahl-, Typenrad-, Thermo- und Laserdruckern. Nadeldrucker besitzen einen Druckkopf, in dem 9 oder 24 Nadeln von Elektromagneten gegen das Farbband geschossen werden. Der Druckkopf bewegt sich quer über das Papier und ermöglicht so den senkrecht untereinander angeordneten Nadeln, ein Zeichen Spalte für Spalte aufzubauen. Weil jeder Buchstabe und jedes Symbol von solchen Druckern aus einer Matrix von Punkten zusammengesetzt wird, gehören diese Geräte zur Gruppe der Matrixdrucker. An dieser Eigenschaft können Sie die Ausgaben eines Matrixdruckers auch erkennen, obwohl die Druckqualität in den letzten Jahren derart gestiegen ist, daß man selbst mit der Lupe große Schwierigkeiten hat, die einzelnen Punkte zu unterscheiden. Der große Nachteil der Nadeldrucker: Sie sind meist sehr laut, was Ihnen schon Spitznamen wie "Brötchentoaster" oder "Kreissäge" eingetragen hat.

Tintenstrahldrucker gehören ebenfalls zu den Matrixdruckern. Hier wird ein Zeichen nicht durch Nadeln erzeugt, sondern durch den Abschuß mikroskopisch kleiner Tintentröpfchen, die durch elektrostatische Aufladung an die richtige Stelle des Papiers gelenkt werden. Vorteil: Wegen der berührungsfreien Druckmethode sind die Tintenstrahldrucker sehr leise. Nachteil: Sie können keine Durchschläge erzeugen und arbeiten nicht mit jedem Papier.

Thermodrucker arbeiten ebenfalls nach dem Matrixprinzip. Hier wird das Abbild auf dem Papier durch Hitze erzeugt. Das klappt natürlich nicht so ohne weiteres mit jedem Papier oder Farbband. Entweder werden spezielle Thermo-Farbbänder benutzt (man spricht dann auch vom Thermo-Transfer-Verfahren), von denen die (feste) Tinte auf das Papier herübergeschmolzen wird, oder man muß sehr teures hitzeempfindliches Spezialpapier benutzen. Neben dem hohen Preis hat dieses den Nachteil, daß es nach einiger Zeit stark abdunkelt. Das Thermo-Transfer-Verfahren wird vor allem für Schreibmaschinen und ähnliche Druckwerke benutzt (führend ist hier IBM), während das reine

Thermo-Druckverfahren wegen der schlechten Eigenschaften des Papiers immer mehr aus der Mode kommt.

Auch Laserdrucker gehören im weitesten Sinne zu den Matrixdruckern. Sie arbeiten im Prinzip wie ein Kopierer (mit denen sie auch starke Ähnlichkeiten haben). Auf eine Trommel wird durch Laser-Licht und elektrostatische Aufladung ein "elektrisches" Abbild einer ganzen Druckseite gebracht, die aus einzelnen Punkten zusammengesetzt ist. Durch die aufgeladenen Punkte wird Farbpulver angezogen, das dann auf das Papier abgewälzt wird. Da Laser-Licht ungeheuer fein fokussiert werden kann, sind die Punkte entsprechend winzig. Als Ergebnis erhält man das beste Druckbild, das mit Computern derzeit zu erzielen ist. Der große Nachteil: Sowohl in der Anschaffung als auch in den laufenden Kosten ist der Laserdrucker die weitaus teuerste Alternative. (Billige Geräte kosten um die 2000 DM, nach oben hin ist erst bei fünfstelligen Beträgen Schluß.). Ausnahme: Der Atari-Laserdrucker ist vergleichsweise billig, denn er arbeitet mit einem Trick. Üblicherweise enthält jeder Laserdrucker einen kompletten Rechner mit recht großem Speicher, in dem zunächst die ganze Seite aufgebaut wird, ehe sie gedruckt werden kann. Der Atari-Laserdrucker überläßt diese Arbeit dem ST und kann dadurch einen großen Teil der Elektronik einsparen.

Ein ganz anderes Prinzip verfolgen die Typenraddrucker, die kaum noch hergestellt werden. Ihnen ist sicherlich das Arbeitsprinzip einer Schreibmaschine bekannt, wo ganze Buchstaben (Typen) auf das Farbband geschlagen werden, um so einen farbigen Abdruck auf dem Papier zu erzeugen. Genau so arbeiten Typenraddrucker, bei denen eine Anzahl von Typen auf einem Rad angeordnet sind. Um ein Zeichen zu drucken, muß der Drucker das Rad zunächst in die richtige Position drehen und kann dann die Type "abfeuern". Es wird Ihnen schnell einleuchten, daß dieses Prinzip sehr langsam ist, dafür aber recht gute Druckqualität liefert. Weiterer Nachteil: Typenraddrucker können keine Grafiken ausgeben. In letzter Zeit hat es sich eingebürgert, daß elektronische Schreibmaschinen (bei denen es sich aus Computersicht lediglich um Typenraddrucker mit eingebauter Tastatur handelt) mit der Möglichkeit versehen sind, an einen Computer angeschlossen zu werden.

Eigentlich kein Drucker ist ein Plotter. Er ist vielmehr eine Zeichenmaschine, in der ein Stift über das Papier bewegt wird und so Linien ziehen kann. Solche Geräte sind teuer und bei der Ausgabe von Texten sehr langsam. Dafür arbeiten sie sehr genau und sind daher für technische Zeichnungen (Stichwort CAD) prädestiniert. Vom Anschluß her unterscheiden sich Plotter selten von den Druckern, lediglich die softwaremäßige Behandlung ist anders.

Wo Drucker angeschlossen werden, hängt vom Modell ab. Sehen Sie bitte in der Beschreibung Ihres Gerätes nach, mit welcher Schnittstelle (das Fachwort dafür lautet Interface) es ausgerüstet ist. Falls Sie die Bezeichnungen "parallel" oder "Centronics" finden, wird Ihr Drucker an den Anschluß "Printer" auf der Rückseite des ST angeschlossen. Finden Sie dagegen die Bezeichnungen "seriell", "RS.232" oder "V.24", so gehört Ihr Druckerkabel in den Stecker mit dem Aufdruck "Modem". Besitzen Sie dagegen einen Atari-Laserdrucker, so wird dieser an die Buchse "Hard-Disk" angeschlossen. Beachten Sie bitte in diesem Fall die Bedienungsanleitung!



Machen Sie es sich bitte zur Gewohnheit, alle Anschlüsse nur bei ausgeschalteten Geräten vorzunehmen. In vielen Fällen sind die Anschluß-Pins direkt mit den elektronischen Komponenten Ihrer Geräte verbunden. Statische Aufladungen oder Kurzschlüsse können die Hardware zerstören. Berühren Sie deshalb vor dem Montieren eine Wasserleitung oder ein blankes Heizungsrohr. Diese sind meist geerdet und führen evtl. vorhandene Aufladungen ab.

In ganz seltenen Fällen ist das Schnittstellenkabel direkt dem Drucker beigelegt, meistens müssen Sie es aber dazukaufen, weil es viele verschiedene Typen von Anschlüssen gibt. Sofern der Drucker eine parallele Schnittstelle besitzt, ist der Fall recht einfach. Fragen Sie Ihren Händler nach einem parallelen Druckerkabel für den ST. Sie können ein solches Kabel auch selbst herstellen; das ist aber mit einiger Löterei verbunden. Für alle, die das trotzdem versuchen wollen, ist die Pin-Belegung der wichtigsten Steckerarten im Kapitel über Schnittstellen aufgelistet. Dabei sollten Sie beachten, daß parallele Schnittstellen keine besonders großen Übertragungswege vertragen. Bei spätestens drei Metern Länge ist Schluß.

Besitzen Sie einen Drucker mit serieller Schnittstelle, so wird der Fall etwas komplizierter. Die serielle Schnittstelle ist leider nicht so genormt, wie man sich das als Anwender wünschen würde. Im Normalfall wird Ihr Händler Ihnen zu Ihrem Drucker auch das passende Kabel besorgen können. Im anderen Fall müssen Sie selbst löteten. Die Pin-Belegungen sind auch hier im Kapitel "Schnittstellen" aufgelistet.

Ein nicht ganz einfaches Problem stellt mitunter die Suche nach dem geeigneten Ort für den Drucker dar. Persönlich bevorzuge ich die Aufstellung auf dem Schreibtisch - aber das ist Geschmackssache. Wenn Sie Ihren Drucker nicht immer im Blick haben müssen, können Sie ihn ruhig auch auf einem separaten Tischchen in der Nähe des Rechners unterbringen. Allerdings sollten Sie beachten, daß unter dem Drucker genug Platz ist, um einen Papierstapel unterzubringen. Außerdem muß bei der Verwendung von sogenanntem End-

lospapier hinter dem Drucker genügend Raum für die bedruckten Blätter bleiben. Sehr wichtig: Die beiden Stapel mit den bedruckten bzw. unbedruckten Blättern müssen genau unter dem Drucker liegen, sonst besteht die Gefahr, daß das Papier schiefzieht oder sogar ein Papierstau entsteht. Halten Sie die Papierwege auch von Kabeln frei.

Grundsätzlich können Drucker mit zwei Papierarten beschickt werden: Endlospapier und Einzelblätter. Einzelblätter sind meistens "korrespondenztauglicher", weil sie keine Perforation aufweisen und außerdem genau dem DIN-A4-Format (21 cm x 29,7 cm) entsprechen. Zudem haben Einzelblätter den Vorteil, daß man sein bisheriges Briefpapier (evtl. mit eingedruckter Adresse usw.) weiterbenutzen kann. Nachteil: Man muß jedes Blatt einzeln einspannen oder einen teuren Sheet-Feeder (engl.: Einzelblattzuführung) kaufen.

Endlospapier ist das EDV-Papier überhaupt. Endlos bedeutet viele aneinanderhängende Blätter, die im Zickzack gefaltet auf einem Stapel liegen. Wie alles in Deutschland ist auch Endlospapier genormt: seine Maße betragen aus irgendwelchen unerfindlichen Gründen 24,13 cm x 30,48 cm (inklusive Lochrändern). Man spricht hier auch von 9,5 Zoll Breite und 12 Zoll Länge. Nimmt man die Lochränder ab, so hat dieses Papier die gleiche Breite wie Einzelblätter, allerdings ist es knapp einen Zentimeter länger. Für breitere Drucker (die DIN-A3-Blätter einspannen können), gibt es auch Endlospapier mit größerer Breite (12 Zoll). Endlospapier gibt es in verschiedenen Ausführungen: Mit normaler Perforation, mit Micro-Perforation (die man nach dem Reißen nicht mehr so leicht sieht) und als Billigpapier zuweilen auch ohne seitliche Perforation. Der Name "Endlospapier" ist natürlich nicht korrekt - jeder Stapel hat einmal ein Ende. Normalerweise erhalten Sie solches Papier in Kartons mit 500, 1000, 2000 oder 5000 Blatt.

Manche Hersteller bieten auch vorgedruckte Formulare oder Endlospapier mit mehreren Durchschlägen an. Sehen Sie bitte vor dem Kauf im Handbuch Ihres Druckers nach, wieviele Durchschläge dieser maximal verträgt. Außerdem im Angebot: Endlos-Aufkleber und Endlos-Briefumschläge.

Die meisten Drucker verkraften Papierstärken bis zu 0,25 mm (das ist schon Karton-Stärke). Wichtiger ist meist die Gewichtsangabe in Gramm pro Quadratmeter. Qualitäten unter 80 g/qm sind eigentlich nur für den Ausdruck von Listings geeignet; außerdem reißen diese Papiere leichter, so daß bei einfacheren Druckermodellen der Papierstau schon vorprogrammiert ist. Wollen Sie mit dem Drucker Korrespondenz schreiben, so sind Papiergewichte ab 80 g/qm empfehlenswert. Dieses Papier scheint auch nicht mehr durch.



Vielfach werden Endlospapiere heute schon aus Umweltpapier hergestellt. Wenn Sie vielleicht auch noch Bedenken haben, dieses Recycling-Papier für Ihre Korrespondenz einzusetzen, sollten Sie es zumindest für den Ausdruck Ihrer "internen" Unterlagen benutzen - unser aller Umwelt zuliebe.

Soll Endlospapier verwendet werden, so muß dieses mittels zweier Stachelwalzen transportiert werden (daher auch die Lochleiste an beiden Seiten). Diese Walzen und die dazugehörige Mechanik nennt man Traktor. In vielen Druckern ist ein solcher Traktor fest eingebaut, andere benutzen sogenannte Aufsatztraktoren, die bei Bedarf eingebaut werden können.

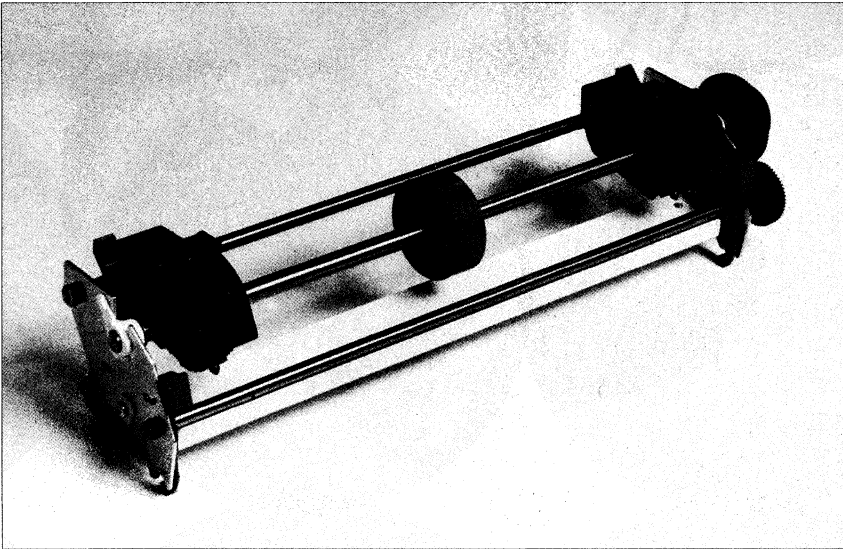


Abb. 4.1: Ein Traktor

Ein anderer, häufig gebrauchter Begriff ist "Einzelblatteinzug". Er hat zwei Bedeutungen. Zum einen ist damit das Gerät gemeint, in welches ein Stapel Einzelblätter eingelegt werden kann. Auf Knopfdruck oder Software-Befehl wird dann automatisch das nächste Blatt in den Drucker eingezogen. Solche Geräte sind recht teuer. Die andere Bedeutung kennzeichnet eine Eigenschaft des

Druckers. Auf einen Knopfdruck hin dreht der Drucker seine Walze genau so weit, daß ein an der Rückseite per Hand angelegtes Blatt bis zum Anfang eingezogen wird.

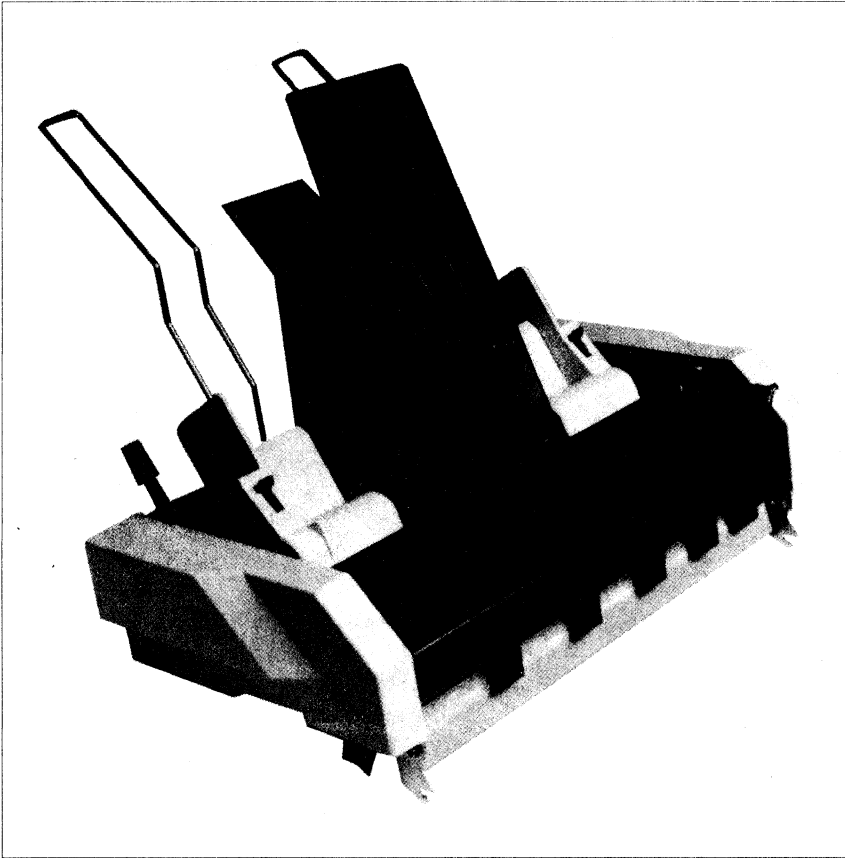


Abb. 4.2: Ein Einzelblatteinzug

Kommen wir zu den Schriftarten. NLQ ist die Abkürzung für Near Letter Quality (zu deutsch: Fast-Korrespondenz-Qualität). Damit wird die Fähigkeit eines Matrixdruckers gemeint, Buchstaben in korrespondenztauglicher Qualität zu erzeugen. Das Gegenteil von NLQ ist die sogenannte Draft-Qualität, die Sie bestimmt von Ihrer letzten Stromrechnung her kennen.

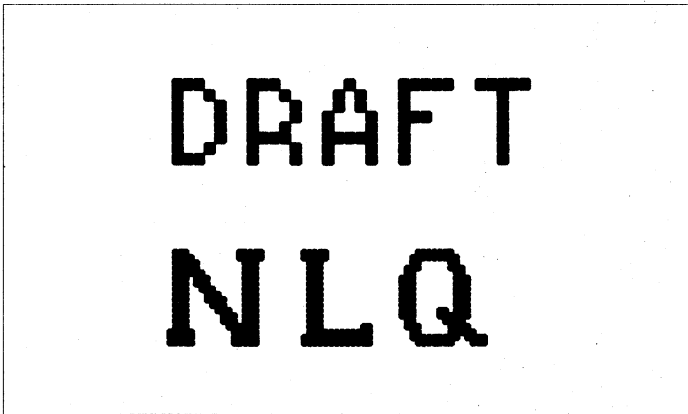


Abb. 4.3: NLQ- und sogenannte Draft-Qualität



9-Nadel-Drucker erkaufen sich die NLQ-Fähigkeit mit einem kleinen Trick: Die Zeichen werden nicht in einem Durchgang gedruckt. Statt dessen fährt der Druckkopf mehrfach über die gleiche Zeile hinweg, wobei Papier und Druckkopf jedesmal um ein winziges Stück verschoben werden. Auf diese Weise werden die Abstände zwischen den einzelnen Punkten auf dem Papier minimal. Die teureren 24-Nadeldrucker wenden diesen Trick zwar ebenfalls an, brauchen aber unter Umständen nicht so viele Durchgänge und sind deshalb schneller. Außerdem ist die normale Druckqualität bei diesen Geräte oft schon so gut, daß man auf NLQ verzichten kann.

Viele Drucker werden mit dem Attribut "Epson-kompatibel" bezeichnet. Der japanische Hersteller Epson ist seit Jahren führend auf dem Markt der Drucker. Dazu hat wohl vor allem beigetragen, daß dieser Hersteller die Befehlscodes für alle seine Drucker weitgehend einheitlich gehalten hat. In den letzten Jahren sind viele Konkurrenten dazu übergegangen, ihre Geräte ebenfalls mit dem ESC/P-Standard auszurüsten (ESC/P bedeutet Epson Standard Code for Printers). Wenn Ihr Drucker diese Befehle beherrscht, sind Sie schon so gut wie aus dem Schneider, da fast jedes Programm für diese voreingestellt ist.

Das zweite wichtige Attribut ist "IBM-kompatibel". Ungefähr zu der Zeit, als IBM seinen ersten PC auf den Markt brachte, hat der blaue Riese (so der Spitzname in der Branche) auch einen eigenen PC-Drucker vorgestellt. Wie das nun einmal so ist: IBM hat es sich nicht nehmen lassen, die eigenen Drucker mit anderen Zeichensätzen und vor allem leicht geänderten Befehlen (im Vergleich

zu ESC/P) auszurüsten. Da IBM aber auch Trendsetter ist, hat sich der IBM-Standard schnell ausgebreitet, so daß viele Drucker heute beide Modi beherrschen. Natürlich werden Sie sich jetzt fragen, ob Atari auch einen eigenen Standard hat. Jein. Atari hat sich bei seinem eigenen Drucker SMM804 an die bisher bekannten Befehlscodes gehalten. Aber der ST hat gegenüber IBM- und Epson-Standard einen leicht veränderten Zeichensatz. Der SMM804 ist neben einigen anderen Druckern als einziger in der Lage, diese Spezialzeichen des ST ohne Klimmzüge auszudrucken. Es ist aber kein Beinbruch, wenn Sie keinen dieser sogenannten ST-kompatiblen Drucker erworben haben. Bei normalen Anwendungen bieten diese kaum jemals einen Vorteil.

Bevor Sie anfangen können mit dem Drucken, müssen Sie sich fast immer mit den DIP-Schaltern befassen. DIP ist die Abkürzung für Dual-Inline-Package. Gemeint sind damit alle elektronischen Bauteile, die wie herkömmliche ICs mit zwei Reihen von Pins versehen sind. DIP-Schalter sind kleine Schalter, die in solch einem DIP-Gehäuse eingebaut sind. Mit Ihrer Hilfe können an den meisten Druckern Voreinstellungen vorgenommen werden. Wo die DIP-Schalter (Spitzname: Mäuseklaviere) an Ihrem Drucker sitzen, sagt Ihnen die Anleitung, ebenso wie die genaue Bedeutung. Wegen Ihrer mikroskopischen Größe können Sie die Schalter natürlich nicht mit den Fingern bedienen. Ein Kugelschreiber leistet da schon gute Dienste.

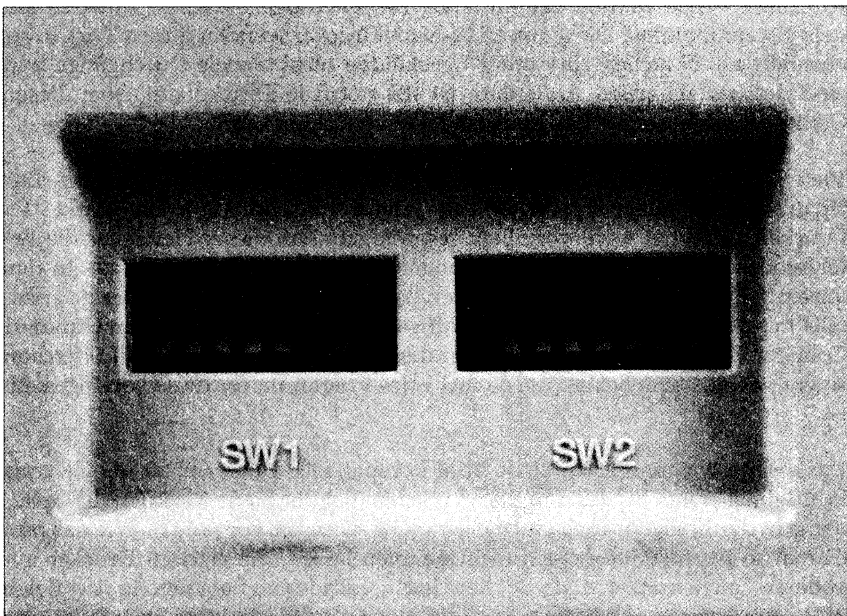


Abb. 4.4: DIP-Schalter

Was kann an den DIP-Schaltern eingestellt werden?

Die meisten Drucker legen an diesen Schaltern fest, ob sie im ESC/P- oder IBM-Modus arbeiten. Hinzu kommen dann Angaben wie Zeichensatz, Druckqualität, Normal- oder Schmalschrift, Seitenlänge usw. Im Normalfall sind die Schalter ab Werk auf einen bestimmten Zustand eingestellt, der zunächst einmal an den meisten Rechnern funktionieren sollte. Im Zweifelsfall stellen Sie die Schalter bitte so ein, daß der IBM-Modus in der Qualität Draft gewählt ist. Falls vorhanden, sollte die Seitenlänge auf 12 Zoll eingestellt sein und die Zeilenvorschub-Option CR+LF lauten. Alle anderen Optionen sollten zunächst ausgeschaltet sein.



Die Einstellungen, die Sie an den Schaltern vornehmen, werden nicht sofort wirksam. Erst beim nächsten Einschalten werden diese übernommen. Außerdem können Sie jede Änderung durch einen Softwarebefehl wieder zurücknehmen. Dies gilt aber nur bis zum nächsten Ausschalten des Druckers.

Fast alle Drucker verfügen über eine kleine Blechzunge in der Papierführung, über die das Vorhandensein von Papier überprüft werden kann. Fehlt es, so unterbricht der Sensor automatisch den Druck, um ein Bedrucken der Walze zu verhindern. In den meisten Fällen blinkt dann eine LED am Drucker, die mit "PAPER OUT" beschriftet ist. In solchen Fällen können Sie erst dann weiterdrucken, wenn Sie neues Papier eingelegt haben und dann ON-LINE drücken.

Pannenhilfe:



Bei der Benutzung besonders schmaler Papiere (z.B. Endlos-Aufkleber) signalisiert der Drucker "PAPER OUT", obwohl Papier eingespant ist.



Das Papier ist so schmal, daß es links am Sensor vorbeiläuft. Für solche Fälle kann man die Funktion dieses Schalters bei vielen Druckern per Mauseklavier oder Software-Befehl unterbinden. Im letzteren Fall müssen Sie allerdings erst ein normales Blatt einlegen, um den Sensor zu überlisten, denn dieser verhindert den Empfang von Befehlscodes, solange das Papier fehlt. Erst nachdem Sie wieder ON-LINE gedrückt haben, können Sie den Befehl senden.

4.2 Voreinstellungen

Bevor Sie Ihren Drucker benutzen können, müssen diverse Einstellungen gemacht werden. So muß der Computer wissen, welche Papierbreite benutzt wird, wo der Drucker angeschlossen ist und auf welche Kommandos er reagiert.

Drucker werden in der Regel dadurch gesteuert, daß der Computer spezielle Steuerzeichen sendet, die sich von den üblichen, druckbaren Zeichen unterscheiden. Mit diesen Steuerzeichen ist es dann möglich, dem Drucker Befehle zu erteilen wie etwa "Drucke die nächste Zeile fett". Leider sind diese Befehle nicht genormt. Zwar halten sich die meisten Hersteller inzwischen an gewisse Standards, doch gibt es immer wieder feine Unterschiede, die den Anwender zur Verzweiflung treiben können.

Eine Möglichkeit zur Voreinstellung finden Sie im Accessory "Drucker Anpassung". Dieser Menüpunkt gehört zum CONTROL.ACC. Falls Ihnen dieses Accessory nicht zur Verfügung stehen sollte, kopieren Sie es bitte von der zum Atari mitgelieferten Disk auf Ihre Systemdiskette, und booten Sie den Rechner neu. Nach dem Aufruf der "Drucker Anpassung" erscheint folgendes Menü:

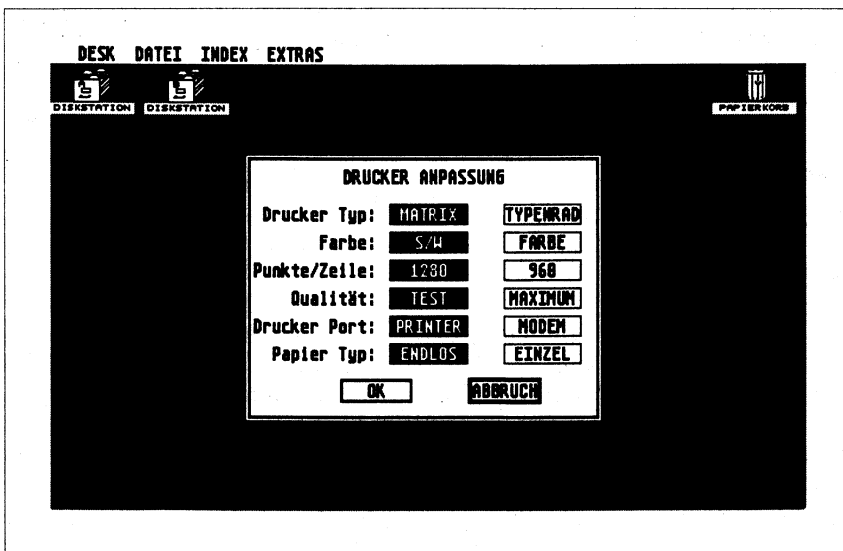


Abb. 4.5: Das Accessory "Drucker Anpassung"

Gehen wir die einzelnen Menüpunkte einmal durch:

Drucker Typ:

Hier stellen Sie den Typ Ihres Druckers ein. Nur wenn Sie wirklich einen Typenraddrucker (mit dem charakteristischen runden Typenrad) angeschlossen haben, sollten Sie diesen Punkt anwählen. Ansonsten muß Ihre Wahl "MATRIX" lauten. Haben Sie Typenrad gewählt, so läßt der ST keine Hardcopies mehr zu, weil diese Geräte nicht grafikfähig sind.

Farbe:

Besitzt Ihr Drucker ein mehrfarbiges Farbband und kann dieses auch nutzen? Wenn ja (im Zweifelsfall Anleitung befragen), so können Sie Ihre Hardcopies auch mehrfarbig ausgeben lassen. In diesem Fall bitte "FARBE" wählen.

Punkte/Zelle:

Vergessen Sie alles, was Sie über Punktdichten und sonstigen Schnickschnack gehört haben. Falls Sie einen Drucker besitzen, der DIN-A3-Blätter einspannen kann (bzw. DIN-A4 quer), dann wählen Sie bitte den Button mit der Aufschrift "1280". In allen anderen Fällen dürfte "960" die richtige Alternative sein.

Qualität:

Die Einstellung dieser Buttons ist zunächst einmal beliebig. Auf dem Desktop hat sie keine Bedeutung. Einige Anwenderprogramme holen sich jedoch hier die Information ab, in welchem Modus sie drucken sollen. Dann wählen Sie bitte "Test" für Draft-Qualität oder "Maximum" für NLQ.

Drucker Port:

Diese Einstellung ist ungeheuer wichtig. Sehen Sie bitte nach, in welchem Anschluß das Druckerkabel eingesteckt ist. Befindet es sich in der Buchse mit Namen "Printer", so wählen Sie diese Alternative; ist das Kabel im "Modem"-Anschluß, so müssen Sie auch "Modem" anklicken.

Papier Typ:

Auch diese Auswahl ist zunächst ohne Bedeutung. Einige Programme (wie z.B. Beckertext) erhalten von hier die Information, welche Papierart im Drucker eingespannt ist und reagieren entsprechend. Wählen Sie bitte immer die Alternative, die gerade den Gegebenheiten entspricht.

Wie Sie das auch schon von anderen Programmen her kennen, können Sie die Drucker-Anpassung entweder durch einen Klick in die OK-Box beenden (dann werden die Einstellungen übernommen) oder den Notausgang über Abbruch wählen.

Normalerweise sollte jeder ESC/P- oder IBM-kompatible Drucker einigermaßen lesbare Ergebnisse liefern. Vor allem die Hardcopies (mittels **Alternate**-**Help**) sollten möglich sein. Bei der Ausgabe von Texten ist es dann trotzdem möglich, daß einige Zeichen falsch oder überhaupt nicht gedruckt werden. Dies läßt sich nur durch den Einsatz zusätzlicher Software ändern.

Pannenhilfe:



Hardcopies werden zwar gedruckt, aber diesen fehlt der rechte Rand. Außerdem ist der Druck sehr schwach.



Sie haben versucht, eine DIN-A3-Hardcopy auf einem DIN-A4-Drucker auszugeben. Ändern Sie die Einstellung "Punkte/Zeile" im Accessory "Drucker Anpassung" auf 960.



Der ST druckt keine Hardcopies, obwohl **Alternate**-**Help** gedrückt bzw. "Hardcopy" im Desktop angeklickt wurde.



In der "Drucker Anpassung" haben Sie als Druckertyp Typenrad angegeben. Diese Geräte können keine Hardcopies drucken. Falls Sie sich verwählt haben, ändern Sie die Einstellung, und versuchen Sie es dann bitte noch einmal.



Nach dem Start eines Druckvorganges tut sich überhaupt nichts. Der Mauszeiger läßt sich nicht mehr bewegen.



Es könnte sein, daß Sie den Drucker nicht eingeschaltet oder nicht korrekt angeschlossen haben. Stimmen alle Einstellungen (insbesondere "Drucker Port")? Wenn ja, so drücken Sie bitte einmal die Taste ON-LINE auf dem Drucker. Tut sich dann immer noch nichts, war vielleicht das Programm, mit dem die Druckausgabe veranlaßt wurde, fehlerhaft. Starten Sie den ST neu. Versuchen Sie dann bitte, eine Hardcopy oder einen Ausdruck einer Datei machen zu lassen. Wenn sich jetzt immer noch nichts tut, obwohl die Einstellungen ansonsten korrekt waren, ist mit großer Wahrscheinlichkeit der Drucker oder das Verbindungskabel defekt.

Wenn Sie einen Drucker mit serieller Schnittstelle (Anschluß über "Modem") besitzen, müssen Sie außerdem noch festlegen, in welchem Modus die Schnittstelle arbeiten soll. Sehen Sie bitte im Handbuch Ihres Druckers nach, welche Parameter er für die serielle Schnittstelle voraussetzt bzw. welche anhand der DIP-Schalter eingestellt sind. Diese Werte übertragen Sie bitte in das Accessory "RS 232 Einstellung" (Beschreibung im Kapitel "Schnittstellen").



Bedingt durch einen Fehler im Betriebssystem der "alten" ST-Modelle (260, 520, 1040) wird die serielle Schnittstelle nicht korrekt bedient.

Daher wird ein Drucker an der seriellen Schnittstelle in den meisten Fällen nur ein einziges Zeichen ausgeben und dann stehenbleiben. Abhilfe bringen hier spezielle Treiberprogramme, die Sie evtl. von Ihrem Händler erhalten können. Außerdem beinhalten die meisten Anwendungsprogramme eigene Treiber für Drucker an der seriellen Schnittstelle.



Was sind Treiberprogramme, und was bewirken sie?

Treiberprogramme klinken sich sozusagen in das normale Betriebssystem des ST ein und sorgen dafür, daß alle Ausgaben an den Drucker im richtigen Format geschehen. Die Treiber liegen meist als Accessory oder im AUTO-Ordner vor, so daß sie automatisch gestartet werden. Manche lassen außerdem noch weitere Eingaben zu, mit denen der Drucker an die speziellen Gegebenheiten angepaßt werden kann.



Sollte sich Ihr Drucker partout weigern, mit dem ST zusammenzuarbeiten, kann dies an fehlenden Treibern liegen. Insbesondere die 24-Nadeldrucker können erst mit den dazugehörigen Treiberprogrammen voll ausgenutzt werden. Bei Laserdruckern tut sich ohne die Treiber gar nichts. Deshalb wird dem Atari-Laserdrucker SLM804/SLM605 auch eine Diskette mit diversen helfenden Programmen beigelegt.

4.3 Der Drucker im Einsatz

Bekanntlich hat fast jede Sprache ihre speziellen Zeichen - nehmen wir doch nur einmal unsere deutschen Umlaute oder die Akzentzeichen der Franzosen. Computer arbeiten normalerweise mit dem ASCII (American Standard Code for Information Interchange - zu deutsch: amerikanischer Standardcode für Informationsaustausch). Das Wort "American" in der Bezeichnung deutet es schon an: Dieser Code ist für den Gebrauch der englischen Sprache angelegt und enthält deshalb keinerlei spezifische Zeichen - vom \$ mal abgesehen. Die verschiedenen Druckerhersteller wollten ihre Drucker natürlich auch bei uns verkaufen, also mußten internationale Zeichensätze her. Bei Epson ist man

darauf verfallen, ein paar ungebräuchliche Zeichen wie "{", "~" u.ä. durch die jeweiligen länderspezifischen Symbole zu ersetzen. IBM hat dagegen eine andere Lösung gefunden. Der ursprünglich nur 127 Zeichen umfassende Satz wurde auf das Doppelte erweitert und in den entstehenden Freiraum alle gewünschten Spezialzeichen integriert. Der Vorteil dieser Methode: Alle IBM-Drucker und -Computer weltweit können mit dem gleichen Zeichensatz arbeiten, während bei der Epson-Methode immer ein Umschalten auf die jeweils gewünschte Tabelle nötig wird. Der ST benutzt im wesentlichen den IBM-2-Zeichensatz, ändert aber einige Sonderzeichen ab. Eine Auflistung der verschiedenen Zeichensätze finden Sie im Anhang.



Einige Drucker halten zwei IBM-Zeichensätze bereit, IBM-1 und IBM-2. IBM-1 ist meistens die ältere Version, die die Umlaute noch nicht enthält!

Zum Einschalten des jeweiligen Zeichensatzes gibt es im allgemeinen zwei Möglichkeiten. Die erste: Sie verstellen die DIP-Schalter entsprechend den in der Anleitung gegebenen Möglichkeiten. Normalerweise kann aber der IBM-Zeichensatz auch nur dann eingestellt werden, wenn der gesamte Drucker in den IBM-Modus umgestellt wird. Umgekehrt sind die Epson-Zeichensätze nur über den ESC/P-Modus erreichbar. Für den Anfang erreichen Sie die besten Ergebnisse, wenn Sie den IBM-Modus mit dem Zeichensatz 2 wählen. Beachten Sie bitte, daß die Änderungen an den DIP-Schaltern erst mit dem nächsten Einschalten des Druckers wirksam werden.

Die andere Möglichkeit zum Umschalten des Zeichensatzes ist das Absenden von Befehlscodes. Das geht am einfachsten über das BASIC. Die nötigen Befehlscodes hängen davon ab, in welchem Modus sich Ihr Drucker befindet (DIP-Schalter-Einstellungen). Ein Beispiel:

Ihr Drucker befindet sich im ESC/P-Modus mit dem amerikanischen Zeichensatz. Lassen Sie den Befehl

```
LPRINT CHR$(27); "R"; CHR$(n)
```

ausführen, wobei Sie für n einen Wert von 0 bis 10 einfügen (siehe Tabelle). Damit wird Ihr Drucker auf den jeweiligen Länderzeichensatz umgestellt. Falls Ihr Handbuch ein etwas anderes Format für die Befehlscodes anzeigt: ESC steht für CHR\$(27), normale Zahlen können Sie direkt übernehmen. Bei hexadezimalen Zahlen, die Sie daran erkennen können, daß Buchstaben auftauchen, ohne daß diese in Anführungszeichen stehen, müssen Sie evtl. noch umwandeln (siehe Kap. "Zahlensysteme"). Andere Möglichkeit: Übernehmen Sie den hexadezimalen Wert, und stellen Sie dann bei GFA-BASIC ein "&H" und bei ST-BASIC ein "\$"-Zeichen davor. Beispiel: 1B wird zu &H1B bzw. \$1B.

Tabelle der Zeichensatznummern:

n	Land	n	Land
0	USA	6	Italien
1	Frankreich	7	Spanien
2	Deutschland	8	Japan
3	England	9	Norwegen
4	Dänemark I	10	Dänemark II
5	Schweden		

In den meisten Fällen können Sie mit folgender Befehlssequenz den Druckermodus wechseln:

```
LPRINT CHR$(27); "~50"
```

schaltet den ESC/P-Modus ein,

```
LPRINT CHR$(27); "~51"
```

bringt den Drucker in den IBM-Modus.

Pannenhilfe:



Beim Ausdrucken von Dateien über das Desktop erscheinen statt der erwarteten Umlaute irgendwelche Sonderzeichen.



Zunächst sollten Sie sicherstellen, daß die auszudruckende Datei keinerlei Steuerzeichen enthält. Zu diesem Zweck lassen Sie bitte die Datei auf dem Bildschirm anzeigen. Tauchen dabei nur normale Buchstaben auf, so liegt der Fehler nicht an der Datei. Ihr Drucker arbeitet wahrscheinlich mit dem falschen Zeichensatz. Vergewissern Sie sich, in welchem Modus der Drucker eingestellt ist. (Aus- und Einschalten bringt das Gerät in den Modus, der durch die DIP-Schalter bestimmt wird.). Arbeitete der Drucker im IBM-Modus, so besteht die Möglichkeit, daß einige Akzent- und Umlautsymbole als sogenannte Hochbit-Steuerzeichen aufgefaßt werden. Falls vorhanden, setzen Sie bitte einen DIP-Schalter so, daß diese Option auf Akzentzeichen steht. Fruchtet das auch nichts, so müssen Sie leider mit den falschen Zeichen leben. Einziger Ausweg: Drucken Sie die Datei mit einem Textverarbeitungsprogramm korrekt aus.

Falls Ihr Drucker im ESC/P-Modus war, versuchen Sie es bitte mit dem IBM-Modus oder dem deutschen Zeichensatz.

➡ Beim Ausdrucken einer Datei mit einem Anwendungsprogramm erscheinen falsche Zeichen auf dem Papier.

✓ Hier haben wir einen typischen Fall von falscher Druckeranpassung. Weiter unten können Sie nachlesen, wie man eine solche Druckeranpassung selbst durchführt.

➡ Das Ausdrucken einer Hardcopy endet in einem wüsten Chaos.

✓ Wahrscheinlich arbeiten Sie mit einem 24-Nadel-Drucker. Solche Geräte wurden von Atari bei der Entwicklung der Hardcopy-Funktion leider nicht berücksichtigt. Besorgen Sie sich bitte ein spezielles Hardcopy-Accessory für Ihren Druckertyp. In den meisten Fällen können Sie das beim Händler kostenlos erhalten.

➡ Beim Ausdrucken (über die serielle Schnittstelle) erscheinen wirre Zeichenfolgen auf dem Papier, die anscheinend gar nichts mit der Datei zu tun haben.

✓ Dieses Problem liegt ausnahmsweise mal nicht am fehlerhaft eingestellten Zeichensatz. Es spricht alles dafür, daß die Parameter der seriellen Schnittstelle nicht korrekt eingestellt sind - siehe Kapitel "Schnittstellen".

➡ Beim Ausdrucken einer Datei über das Desktop werden einige Zeichen plötzlich fett, kursiv oder in anderen Schriftarten ausgedruckt.

✓ Die Datei enthält Steuerzeichen. Drucken Sie sie mit dem Textverarbeitungssystem aus, mit dem sie erstellt worden ist.

Papier wird nicht überall auf der Welt in der gleichen Länge hergestellt. In der amerikanischen Norm hat das normale Endlosblatt eine Länge von 11 Zoll, in Europa werden dagegen 12-Zoll-Blätter verwendet. Sie sollten deshalb die Seitenlänge an den DIP-Schaltern immer auf diesen Wert einstellen, um Komplikationen zu vermeiden. Der Drucker benutzt die Zoll-Angabe, um das Papier am Seitenende korrekt bis zum nächsten Blattanfang zu transportieren.

Pannenhilfe:

➡ Beim Ausdruck mehrerer Seiten hintereinander wird die nächste Seite immer ein Stück versetzt angefangen.

✓ Der Drucker ist nicht auf die richtige Seitenlänge eingestellt. Er nimmt 11-Zoll-Blätter an und transportiert das Papier deshalb auch nur so weit, wie es den kürzeren Blättern entspricht. Auf diese Weise wandert jeder Seitenanfang ein paar Zeilen weiter nach oben. Stellen Sie die Blattlänge auf 12 Zoll, und schalten Sie dann bitte den Drucker kurz aus und dann wieder ein.



Beim Drücken der FF-Taste oder am Seitenende legt der Drucker den Blattanfang immer in die Mitte eines neuen Papiers. Die Seitenlänge ist aber richtig eingestellt.



Der Drucker geht davon aus, daß das Papier beim Einschalten so eingelegt ist, daß sich der Druckkopf genau am Blattanfang befindet. Das Gerät ist keinesfalls in der Lage, die Perforation zu "sehen". Statt dessen zählt der Drucker für jede Seite einfach die Menge des transportierten Papiers und berechnet daraus die Länge bis zum nächsten Blattanfang. Schalten Sie den Drucker aus, und transportieren Sie bitte das Papier von Hand so, daß der Druckkopf direkt unter der Perforation steht. Dann können Sie wieder einschalten.



Transportieren Sie das Papier bitte niemals bei eingeschaltetem Drucker von Hand weiter. Solange das Gerät eingeschaltet ist, stehen die empfindlichen Schrittmotoren unter Spannung. Werden diese dann von Hand bewegt, verkürzt das die Lebensdauer und die Präzision, die gerade beim Drucken sehr wichtig ist.



Beim Ausdrucken einer Datei oder eines Listings wird über die Perforation gedruckt; es bleibt kein unterer und oberer Rand.



Das beschriebene Problem ist eigentlich der Normalfall. Nur Anwendungsprogramme versuchen von sich aus, eine korrekte Seitenformatierung zu erreichen. Sie können aber den Drucker veranlassen, die Perforation jedesmal zu überspringen, indem Sie ihm einen entsprechenden Befehl senden (siehe Drucker-Anleitung).



Einige Geräte erlauben sogar, diese Funktion per DIP-Schalter einzustellen. Ich rate davon ab, weil die meisten Anwendungsprogramme dann nicht mehr richtig formatieren können. Sie sollten den Perforation-Skip (so die englische Bezeichnung dafür) nur bei Bedarf per Befehlscode ein- und vor dem Gebrauch eines Anwendungsprogramms ggf. wieder ausschalten.

Was bedeuten CR und CR/LF?

CR ist die Abkürzung für Carriage Return (also Wagenrücklauf). Damit ist das Zurückkehren des Druckkopfs an den linken Papierrand gemeint. LF steht für Line Feed, was auf deutsch Zeilenvorschub bedeutet. Erhält ein Drucker ein CR, so führt er normalerweise nur den Wagenrücklauf aus, das Papier wird dabei nicht weiter transportiert. Ebenso verhält es sich mit dem LF; hier wird nur transportiert, der Rücklauf unterbleibt. An den meisten Druckern kann man aber einstellen, daß bei jedem CR automatisch ein LF ausgeführt wird; man nennt das auch Auto-Line-Feed. Sie sollten diese Funktion zunächst nicht einschalten, da die meisten Programme für den ST automatisch am Zeilenende CR und LF senden.

Pannenhilfe:



Alles wird in einer Zeile gedruckt, das Papier bewegt sich nicht.



Kontrollieren Sie zunächst, ob das Papier korrekt eingelegt ist und ggf. Traktor oder Walze richtig greifen. Wenn ja, so erwartet das druckende Programm ein Auto-Line-Feed. Schalten Sie diese Option bitte per DIP-Schalter oder Software ein.



Nach jeder Zeile wird eine Leerzeile gedruckt.



Hier ist es genau umgekehrt. Das Programm erwartet, daß Auto-Line-Feed abgeschaltet ist und schickt deshalb zusätzlich zu jedem CR auch ein LF. Schalten Sie bitte den Auto-Line-Feed ab.

4.4 Drucker und Anwendungsprogramme

Im Prinzip können Anwendungsprogramme mit jedem Drucker zusammenarbeiten. Es hängt davon ab, welche Anpassungsmöglichkeiten die Programmierer vorgesehen haben. In den meisten Fällen werden fertige Anpassungen für die wichtigsten Druckertypen mitgeliefert, so daß Sie nur noch den richtigen Typ angeben müssen. Manchmal fehlt dieser Service, und Sie müssen die Anpassungstabellen selbst erstellen. Das ist nicht sehr schwer.

In den meisten Fällen gibt das Programm eine ganze Liste von Anpassungen vor, die jeweils den Namen eines Druckertyps tragen. Nun kommt es darauf an, ob Sie Ihren Drucker in dieser Liste wiederfinden. Wenn ja - wunderbar. Wählen Sie diese Anpassung; damit dürften die meisten Probleme behoben sein.

Befindet sich Ihr Druckertyp nicht in der Liste, versuchen Sie, einen dazu kompatiblen zu finden. Will sagen: Arbeitet Ihr Drucker im ESC/P-Modus, so können Sie normalerweise die Anpassung für den Epson FX80 oder den Epson FX85 wählen (manchmal wird eine solche Anpassung sogar "STANDARD" genannt). Diese sind eigentlich immer vorhanden. Falls Ihr Drucker im IBM-Modus arbeitet, nehmen Sie bitte die IBM-Anpassung. Dies gilt auch, wenn Sie einen Drucker besitzen, der besonders auf den ST abgestimmt ist (z.B. SMM804 oder TAXAN KP 810 ST) und für den keine speziellen Anpassungen vorhanden sind.

Sofern das Programm dies zuläßt, können Sie eine Druckeranpassung fast immer dadurch selbst erstellen, daß Sie eine Tabelle mit den Druckerbefehls-codes anfertigen. Dieses Prinzip wird inzwischen von fast allen Programmen in ähnlicher Weise benutzt, so daß ich die Vorgehensweise am Beispiel BECKERTTEXT einmal aufzeigen möchte. Bevor Sie dies mit Ihrer eigenen Software probieren, sollten Sie aber auf jeden Fall das dazugehörige Handbuch durchlesen!

Druckeranpassungen werden fast immer in Form von Texten angefertigt, die für jede Funktion und für jedes Zeichen eine Folge von Codes enthalten. Meistens sind diese Texte in einem speziellen Ordner abgespeichert und können direkt in eine Textverarbeitung geladen werden (ggf. die Extension in der Objekt-Auswahl ändern). Nehmen Sie dazu eine Anpassung, die Ihrem Modell schon ziemlich nahe kommt, sprich mit der die Daten wenigstens teilweise korrekt ausgedruckt worden sind. Eine solche Tabelle sieht ungefähr so aus:

\0	27,'@',27,'O',27,18	* Druckertabelle für IBM-Kompat.
\1+	27,'E'	* fett ein
\1-	27,'F'	* fett aus
\2+		
\2-		
\3+	27,'4'	* kursiv ein
\3-	27,'5'	* kursiv aus
\4+	27,'-',1	* unterstreichen ein
\4-	27,'-',0	* unterstreichen aus
\5+	27,'G'	* Doppeldruck ein
\5-	27,'H'	* Doppeldruck aus
\6+	27,'S',0	* Hochsetzen ein
\6-	27,'T'	* Hochsetzen aus
\7+	27,'S',1	* Tiefsetzen ein
\7-	27,'T'	* Tiefsetzen aus
\A	18,27,'P'	* elite
\B	18,27,'M'	* pica
\C	15	* schmal
\E	14	* breit
\a	10	* 10 cpi
\b	12	* 12 cpi

\c	17	* 17 cpi
\e	5	* 5 cpi
\Z	27, 'A'	* Zeilenabstand in 72stel "
\z	27, '2'	* Nachspann Zeilenabstand
\n	13,10	* neue Zeile (CR/LF)
ä	132	
ö	148	
ü	129	
ß	225	
Ä	142	
Ö	153	
Ü	154	
	21	

Zu Anfang stehen meist die Steuerzeichen, die das Programm auch zur Textformatierung und -gestaltung nutzt. Im Falle von Beckertext sind diese Steuerzeichen durch einen vorgestellten BACKSLASH gekennzeichnet. Alles, was in einer Zeile nach dem Symbol für ein Zeichen folgt, wird später statt des Zeichens an den Drucker geschickt. Eine Ausnahme bilden die mit einem Stern abgetrennten Kommentare, sie dienen nur zu Ihrer Orientierung. "\0" steht bei Beckertext für die Schriftart 0, die wieder auf normalen Druck zurückschalten sollte. Im obigen Beispiel sind dort insgesamt drei Befehle (siehe Drucker-Handbuch) eingetragen:

27,"@" steht für die Druckerinitialisierung, d.h. das Gerät wird wieder in den Einschaltzustand zurückversetzt.

27,"O" löscht eine ggf. eingeschaltete untere Randbegrenzung.

27,18 löscht alle Tabulatoren (nur bei IBM!).

"\1+" beschreibt die Codes, die zum Einschalten der Fettschrift nötig sind. Im IBM-Modus lautet dieser 27,"E".



Nehmen wir nun an, Sie möchten, daß der Drucker automatisch alles in NLQ-Schrift druckt, so daß Sie sich das Umschalten sparen können. Das bedeutet, die "normale" Schriftart wechselt zu NLQ. Der Befehlscode für "NLQ ein" ist 27,"x1". Hängen Sie diese Codes an die bereits in der Zeile \0 eingetragenen an, und speichern Sie die Anpassung unter einem anderen Namen. Immer wenn Sie in Zukunft diese Druckeranpassung benutzen, wird automatisch alles in NLQ gedruckt.

Gehen wir weiter nach unten: In der Zeile, die mit "ä" beginnt, ist der Wert 132 eingetragen. 132 ist im IBM-Zeichensatz die Nummer des "ä". Bei jedem Druck eines "ä" sendet Beckertext also den Code 132 an den Drucker. Dies ist eigentlich der einfachste Fall. Bei ESC/P-Druckern müssen die Umlaute aus den

länderspezifischen Zeichensätzen geholt werden. Dazu werden dann meist mehrere Befehle in diese Zeile eingetragen, etwa so:

```
ä      27,'R',2,'{' ,27,'R',0
```

27,"R",2 schaltet den deutschen Zeichensatz ein (nur bei Epson-kompatiblen Druckern!), "{" ist das Zeichen des amerikanischen Satzes, das im deutschen Zeichensatz zum Umlaut umdefiniert worden ist. 27,"R",0 schaltet schließlich wieder in den amerikanischen Zeichensatz um.

Solche Anpassungen können Sie für jedes beliebige Zeichen vornehmen.

Das genaue Aussehen der Befehlscodes müssen Sie dem Handbuch zu Ihrem Drucker entnehmen. Diese Codes werden in den unterschiedlichsten Formaten angegeben, so daß Sie sie evtl. ein wenig übersetzen müssen. In den obigen Beschreibungen ist Ihnen sicher schon aufgefallen, daß die Codes entweder als Zahl oder als Zeichen (und dann in Hochkommata) auftraten. Die Zahlen stehen dabei für ein Zeichen im sogenannten ASCII-Code (siehe Anhang). Nicht alle Zeichen dieses Codes sind druckbar, viele davon haben steuernde Funktionen. Bestes Beispiel dafür ist der Code 27, der für das Zeichen ESC steht. Richtig - dieses Zeichen finden Sie auch auf der Tastatur, aber dort wird es von den meisten Programmen als Funktionstaste betrachtet, so daß Sie ESC nicht direkt eintippen können, sondern eben als Code. In den meisten Druckeranleitungen werden die Befehlscodes in ungefähr dieser Form angegeben:

```
ESC "R",CHR$(2)
```

ESC steht - wie bereits erwähnt - für den Code 27. Treten weitere solcher Bezeichnungen auf, sehen Sie bitte in der ASCII-Tabelle nach, welchen Code diese besitzen. Das "R" können Sie in die Druckeranpassung fast originalgetreu übernehmen. In einigen Fällen müssen Sie die Anführungsstriche durch Hochkommata ersetzen. Folgt schließlich noch CHR\$(2). Dabei handelt es sich eigentlich um eine BASIC-Funktion, die das Zeichen mit dem Code 2 ausgibt. Für unsere Übersetzung heißt das, daß wir einfach die eingeklammerte Zahl hinschreiben können. Insgesamt wird der obige Befehl also so übersetzt:

```
27;'R';2
```

Manche Programme verlangen die Angabe der Zahlen im Hexadezimalsystem.

Schließlich müssen Sie dem Programm nur noch beibringen, daß es die neue Tabelle benutzen soll. Wie das geht, hängt wiederum weitgehend vom Programm ab. Auf jeden Fall sollten Sie die Tabelle unter einem neuen Namen im

Verzeichnis mit den bereits vorhandenen Anpassungen abspeichern. Dabei müssen Sie in den meisten Fällen den sogenannten ASCII- oder Nicht-Word-prozessor-Modus benutzen, der dafür sorgt, daß der eben erstellte Text ohne interne Steuerzeichen abgespeichert wird. In einigen Fällen muß diese Tabelle nun noch von einem speziellen Programm bearbeitet werden, in anderen Fällen reicht es, wenn Sie das Programm verlassen. Im darauf folgenden Neustart müssen Sie dann nur noch angeben, daß diesmal eine andere Tabelle benutzt werden soll.

Vielfach ist nicht von Anfang an klar, welche Zeilen in der Tabelle geändert werden müssen. Hier heißt es probieren. Am besten ist es, wenn Sie einmal alle möglichen Zeichen in die Textverarbeitung eintippen und diese Zeichensammlung ausdrucken lassen. Am Ergebnis können Sie später ablesen, welche Zeichen richtig und welche falsch gesendet werden. Wenn Sie den Zeichensatz in Normalschrift richtig angepaßt haben, können Sie mit ein paar Testausdrucken in den verschiedenen Schriftarten wie "fett", "kursiv" usw. fortfahren. In diesem zweiten Schritt können Sie dann die Befehlscodes für die zugehörigen Schriftarten ändern, bis diese richtig angepaßt sind.

Pannenhilfe:



Beim Einschalten mehrerer Attribute gleichzeitig werden diese nicht korrekt ausgedruckt, also statt kursiver Fettschrift erscheint beispielsweise nur Fettdruck.



Hier handelt es sich nicht um eine fehlerhafte Anpassung, sondern um ein spezielles Druckermerkmal. Viele Geräte sind nämlich nicht in der Lage, alle Schriftarten beliebig zu mischen. Treten solche verbotenen Kombinationen auf, wird häufig eine der beiden Funktionen unterdrückt.

Viele Textverarbeitungsprogramme benutzen Formulare oder Layout-Tabellen, um die richtige Gestaltung einer Seite beim Druck festzulegen. Auch hier soll wieder Beckertext als Beispiel dienen. Hier sieht ein Formular wie auf der folgenden Abbildung aus.

Wie Sie sehen, werden hier elementare Werte wie zum Beispiel die Seitenlänge eingetragen. Diese Zahlen variieren von Programm zu Programm, aber auch von Blattgröße zu Blattgröße. Im schlimmsten Fall können Sie die richtigen Einstellungen nur durch Probieren ermitteln.

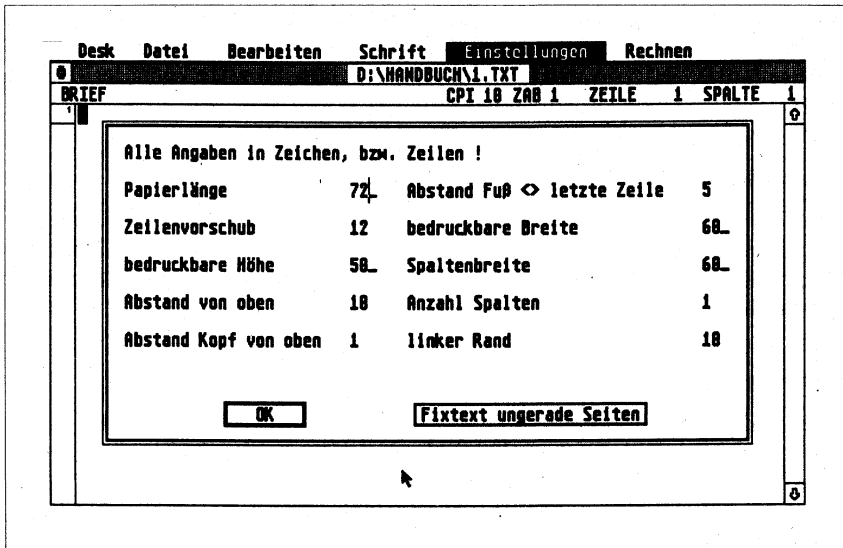


Abb. 4.6: Ein Beckertext-Formular

Papierlänge

Hier wird eingestellt, wie lang das Papier insgesamt ist. Für Endlospapier werden hier 72 Zeilen oder 12 Zoll angegeben. Bei Einzelblättern ist das etwas anders, denn diese müssen zwangsläufig etwas weiter eingespannt werden als Endlosblätter. Als Folge verkürzt sich die noch nutzbare Länge des Blattes. In der Praxis hat sich bei Einzelblättern eine Papierlänge von 66 Zeilen oder 11 Zoll als richtig erwiesen. Für andere Druckmedien (z.B. Aufkleber oder Postkarten) müssen Sie diese Länge entsprechend kürzen.

"Bedruckbare Höhe" oder "Spaltenlänge"

Dieser Wert beschreibt, wieviele der möglichen Zeilen wirklich bedruckt werden sollen. Hiervon hängt auch ab, wie groß der untere Rand wird. 50 bis 55 Zeilen lassen sich auf einem DIN-A4-Blatt unterbringen, ohne daß es eng wird.

"Abstand von oben" oder "oberer Rand"

Mit dieser Zahl geben Sie an, wieviele Zeilen vom oberen Papierrand bis zur ersten bedruckten Zeile freibleiben sollen. Die Differenz aus Papierlänge und der Summe von Spaltenlänge und oberem Rand gibt die Größe des unteren Randes an (im Beispiel: $72 - (55 + 10) = 7$ Zeilen unterer Rand). Mit diesem

Wert müssen Sie evtl. ein wenig experimentieren, bis das Druckergebnis Ihren Vorstellungen entspricht. Bei Einzelblättern muß dieser Wert außerdem sehr klein sein (0 oder 1), da sich der obere Rand schon beim Einlegen des Papiers ergibt.

"Abstand Kopf..." bzw. "Abstand Fuß..."

Viele Textverarbeitungsprogramme erlauben die Verwendung von Kopf- und Fußzeilen. Diese werden in einem bestimmten Abstand unter den Blattrand bzw. unter den Ausdruck gesetzt. Die genauen Abstände können hier festgelegt werden.

Zeilenvorschub

Hier wird - ähnlich einer Schreibmaschine - angegeben, wie weit die Zeilen auseinandergedruckt werden. Die Einheit, in der das geschieht, ist sehr verschieden. Beckertext arbeitet mit typografischen Punkten (das ist 1/72 Zoll), wobei 12 Punkte dem sogenannten einzeiligen Druck entsprechen. Andere Programme erlauben die direkte Angabe von Werten wie 1, 1,5, oder 2 für den jeweiligen Zeilenabstand.

"Bedruckbare Breite" oder "Blattbreite"

Diese Zahl legt fest, wieviele Zeichen maximal in eine Zeile passen sollen. Im deutschen Sprachraum wird bei DIN-A4-Blättern mit Zeilenbreiten von 60 bis 65 Zeichen gearbeitet. Nutzt man die gesamte Blattbreite aus, so können bis zu 80 Zeichen in eine Zeile gebracht werden.

"Spaltenbreite"

Einige Programme - darunter auch Beckertext - ermöglichen den Ausdruck mehrerer Spalten auf einer Seite. Deren Breite kann hier festgelegt werden. Entsprechend gibt der nächste Wert im oben gezeigten Formular die Anzahl dieser Spalten an.

"Linker Rand"

Hier wird eingetragen, wie groß der freie Platz zwischen erstem Zeichen in einer Zeile und dem linken Blattrand bleiben soll. Sie sollten diesen Wert so wählen, daß der rechte Rand etwa gleich groß wird. Die Breite des rechten Randes ergibt sich aus der Differenz zwischen maximal möglicher Zeilenbreite (80 bei DIN-A4) und der Summe aus linker Randbreite und bedruckbarer Breite (im Beispiel: $80 - (60 + 10) = 10$).

4.5 Der Atari-Laserdrucker SLM804/SLM605

Der SLM804/SLM605 war eine echte Sensation, als er auf dem Markt erschien. Inzwischen hat man sich an ihn gewöhnt - die erste Euphorie ist vorbei und hat kritischer Distanz Platz gemacht. Es stellte sich aber heraus, daß man die alten ST-Modelle ohne weiteres mit dem SLM804/SLM605 betreiben kann - und daß das Konzept mit der fehlenden "Intelligenz" des Druckers und ausgelagertem Controller durchaus Freunde gewinnen konnte. Bevor wir aber jetzt einige Tips geben, wie Sie dem SLM804/SLM605 Ergebnisse entlocken können, soll das Prinzip dieses Gerätes erläutert werden. Wie funktioniert eigentlich ein Laserdrucker?

Wie ein Laserdrucker funktioniert

Auch ein Laserdrucker arbeitet punktweise, ähnlich wie ein Matrixdrucker. Damit sind die Ähnlichkeiten aber auch schon vollends beschrieben. Der entscheidende Unterschied liegt in der Art und Weise, in der das Ergebnis zu Papier gebracht wird. Laserdrucker gehören zur Familie der Seitendrucker - es wird immer eine komplette Seite ausgedruckt, die vollständig im Puffer des Druckers vorhanden ist. Diesen Puffer, für eine Seite benötigt man rund ein MByte Speicherplatz, hat unser SLM804/SLM605 bekanntlich nicht, auch nicht die nötige Intelligenz, um diese Seite selbst aufzubereiten. Dazu benötigen wir, wie schon erwähnt, das RAM unseres ST.

Wie funktioniert aber nun der eigentliche Druckvorgang? Bei einem Seitendrucker macht man sich die Kräfte der Elektrostatik zunutze. Die vom Computer gelieferten Bilddaten werden von einem Controller verarbeitet und mittels Lichtstrahlen auf eine mit photosensitivem, also lichtempfindlichem, Material beschichtete Walze geschrieben. Dabei handelt es sich nicht um eine chemische Reaktion, sondern um einen elektrostatischen Effekt: Die Oberfläche der Walze wird mit hoher Spannung aufgeladen. Fällt nun das Licht auf Bereiche der aufgeladenen Oberfläche, wird diese an den betreffenden Stellen entladen. Diese entladenen Stellen werden später auf der ausgedruckten Seite weiß bleiben. Die nicht entladenen Stellen (die mit rund 1000 Volt aufgeladen sind) bilden die zu druckenden Zeichen bzw. Grafikpunkte und stellen eine inverse Darstellung der Seite dar, ein sogenanntes Image.

Als Lichtquelle dient bei einem Laserdrucker meist ein Halbleiter-Dioden-Laser, dessen Licht nicht direkt auf die lichtempfindliche Walze fällt, sondern von einem Reflektorsystem (rotierender, mehrflächiger Spiegel, dessen Bewegung mit der Rotation der Walze vom Controller synchronisiert wird) gezielt auf die Walze gelenkt wird. Die Energie dieses Lichts ist entscheidend für die Geschwindigkeit des Ausdrucks, denn je höher die Energie des einfallenden

Lichtes, desto schneller kann eine Stelle der Walze entladen werden. Teure Laserdrucker verwenden daher einen energiereicheren Edelgas-Laser, beispielsweise einen Argon-Laser. Übrigens muß die Lichtquelle eines Seitendruckers nicht unbedingt aus einem Laser bestehen - NEC z.B. bietet einen Drucker an, bei dem diese Aufgabe (nach dem gleichen Prinzip wie beim Laser) von LEDs, also Leuchtdioden wahrgenommen wird. Doch weiter in der Funktionsbeschreibung des Laserdruckers. Die Walze mit der (noch unsichtbaren) durch die Ladungen gebildeten Inversdarstellung der Seite rotiert nun an einem Behälter vorbei, der sogenannten Toner enthält, gewissermaßen die Farbe, mit der unser Drucker etwas zu Papier bringt. Zwischen dem Toner und den aufgeladenen Stellen der Walze besteht nun ein erheblicher Potentialunterschied, der dazu führt, daß die sehr feinen Staubpartikel des Toners (mit niedrigem Potential) von den mit rund 1000 Volt aufgeladenen Walzenbereichen angezogen werden und sich dort sichtbar anordnen. Auf der Walze entsteht ein Abbild der zu druckenden Seite. Der letzte Schritt besteht jetzt darin, daß die Walze den Tonerstaub auf Papier rollt, das höher aufgeladen ist als die Walzenoberfläche. Aufgeladen wird das Papier durch den sogenannten "Corona-Draht". Der Staub wandert deshalb auf das Papier. Schließlich wird dieses Papier noch erhitzt, und der Tonerstaub brennt in die Seite ein - sonst könnten wir ihn gleich wieder runterblasen! Zu guter Letzt kommt die fertige Druckseite aus dem Ausgabeschlitz des Druckers.

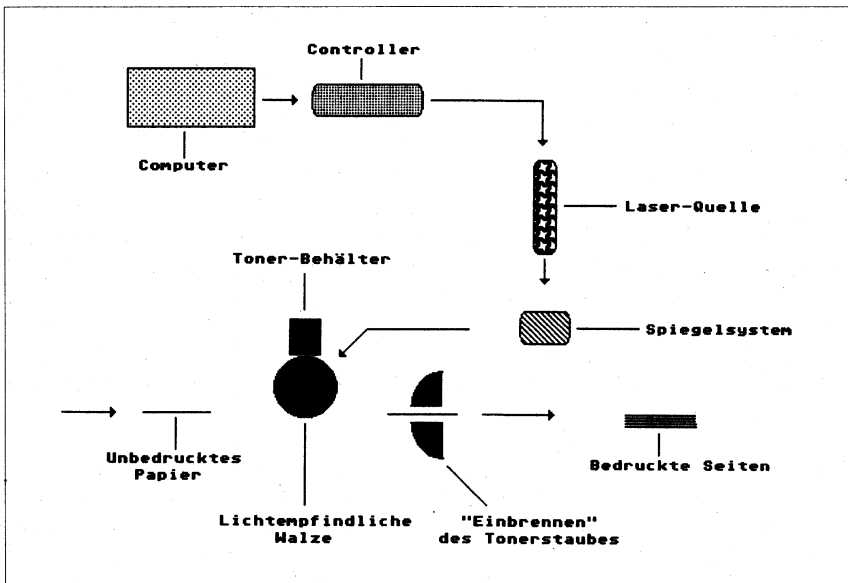


Abb. 4.7: Die Arbeitsweise eines Laserdruckers

So einfach wie diese Beschreibung ist die Sache natürlich nicht, das Prinzip dürfte aber jetzt klar sein. Zur besseren Anschauung sollten Sie sich die obenstehende Grafik ansehen, die die grundsätzlichen Abläufe in einem Laserdrucker vereinfacht darstellt.

Der "dumme" SLM804/SLM605

Ein normaler Laserdrucker besitzt einen eigenen Prozessor, oft ist es ein Motorola 68000 oder gar ein 68020. Der übernimmt die Steuerung des Druckers. Weiterhin verfügt ein solcher handelsüblicher Laserdrucker über nicht unbedeutende RAM-Kapazitäten. Üblich sind wenigstens ein bis zwei MByte Speicher. Der Atari-Laser enthält aber nicht mehr als die nackte Mechanik eines Laserdruckers. Der Controller ist als eigenes Gerät ausgelagert, und die Rechenleistung sowie das nötige RAM stellt der Mega ST zur Verfügung. Die Steuerung des Controllers bzw. des Laserdruckers erfolgt deshalb mit einem Emulationsprogramm.

Die Auflösung des Laserdruckers beträgt 300 x 300 Punkte/Inch. Das entspricht dem marktüblichen Wert. Eine komplette Seite hat daher eine Gesamtauflösung von 3386 mal 2336 Punkten. Da eine normale Grafikseite also rund ein MByte RAM beansprucht, kann man den SLM804/SLM605 mit den alten ST-Modellen ohne spezielle Vorkehrungen nur sehr bedingt benutzen. Es geht aber immerhin - wenn man sich auf Hardcopies des Bildschirms oder einfache Textseiten beschränkt. Der unten erwähnte Emulator Laserbrain erlaubt allerdings den Einsatz mit allen ST-Typen. Mit einem Mega ST bleibt jedoch für jede denkbare Applikation genügend Speicherplatz übrig. Wir haben dafür aber andere Probleme. Zum Beispiel die Ansteuerung über den DMA-Port. Der DMA-Port wird deshalb benutzt, weil gut und gerne 7,5 MByte innerhalb einer Minute zu übertragen sind - will man die standardmäßigen 8 Seiten/Minute ausdrucken. Solche Datenmengen lassen sich mit dieser hohen Geschwindigkeit nur mit DMA handhaben. Es besteht also die Notwendigkeit, alle Druckerbefehle auf die DMA-Schnittstelle umzuleiten. Atari liefert zu diesem Zweck einen Emulator mit dem SLM804/SLM605.

Leider funktioniert der Emulator nicht mit jedem Programm, es müssen eigene Druckertreiber erstellt werden. Der gelieferte Emulator kann lediglich einen Typenraddrucker des Typs Diablo emulieren - Grafik läßt sich damit nicht ausdrucken. Viele Softwarehäuser arbeiten bereits an Treibern für die DMA-Schnittstelle und den Atari-Laserdrucker. Manche hatten auch schon Erfolg. Und zwar nicht nur die Anbieter von DTP-Programmen, die mittlerweile nahezu alle einen Treiber für den SLM804/SLM605 bieten können. Im Anschluß an die Beschreibung des Zusammenbaues des SLM804/SLM605 und der mit

gelieferten Programme werden wir Ihnen noch ein hochinteressantes Emulationsprogramm für den SLM804/SLM605, das bereits erwähnte Laserbrain, näherbringen.

Laser-Puzzle - SLM804/SLM605 zusammenbauen

Der SLM804/SLM605 wird in Einzelteilen geliefert. Keine Angst, es sind nicht viele, und der Zusammenbau ist leicht. Die verschiedenen Elemente wollen wir kurz mit ihrer Funktion aufführen:

Die lichtempfindliche Walze

Auf dieser Walze entsteht das oben beschriebene Inversbild der späteren Druckseite. Die Oberfläche ist sehr empfindlich gegen mechanische Beanspruchungen. Ist die Oberfläche einmal verkratzt, ist eine neue Walze fällig, denn jeder Kratzer ist später auf dem Ausdruck zu sehen. Außerdem ist die Walze ja lichtempfindlich - man sollte sie nicht länger als nötig im Licht lassen. Die Walze wird mit der Kombination aus Tonerbehälter und Tonerpatrone zusammengeklinkt (beachten Sie die Abbildungen im Handbuch) und in den Drucker eingelegt.

Nach dem Zuklappen wird die Walze mitsamt dem Tonerbehälter an die Oberseite des Druckers befördert. Bei einem späteren Ausbauen muß ein weiterer Riegel an der rechten Seite des Druckergehäuses entriegelt werden. Nach dem eigentlichen Öffnen mit dem großen Hebel an der rechten Seite liegt das Ganze dann wieder auf dem unteren Gehäuseteil. Die Walze hat nach Aussage von Atari eine Lebensdauer von ungefähr 10000 Seiten, danach muß sie voraussichtlich gewechselt werden.

Der Tonerbehälter und die Tonerpatrone

Der Tonerbehälter wird vor dem Einbau an die oben beschriebene Walze geklinkt. In den Tonerbehälter stecken wir zuvor noch die in einem staubdichten Beutel befindliche Tonerpatrone. Vor dem Einbau wird die Patrone leicht hin- und herbewegt, damit sich das Tonerpulver besser verteilt. Passen Sie auf, daß Sie nichts von dem Toner verschütten - verschmutzte Kleidung und Hände sind kaum sauber zu kriegen.

Nach dem Einbau der Walzen-/Tonerbehälter-Kombination in den Drucker müssen Sie noch das Klebeband entfernen, mit dem die Patrone verschlossen ist. Es läßt sich zur Seite herausziehen. Jetzt ist der Drucker fast betriebsbereit.

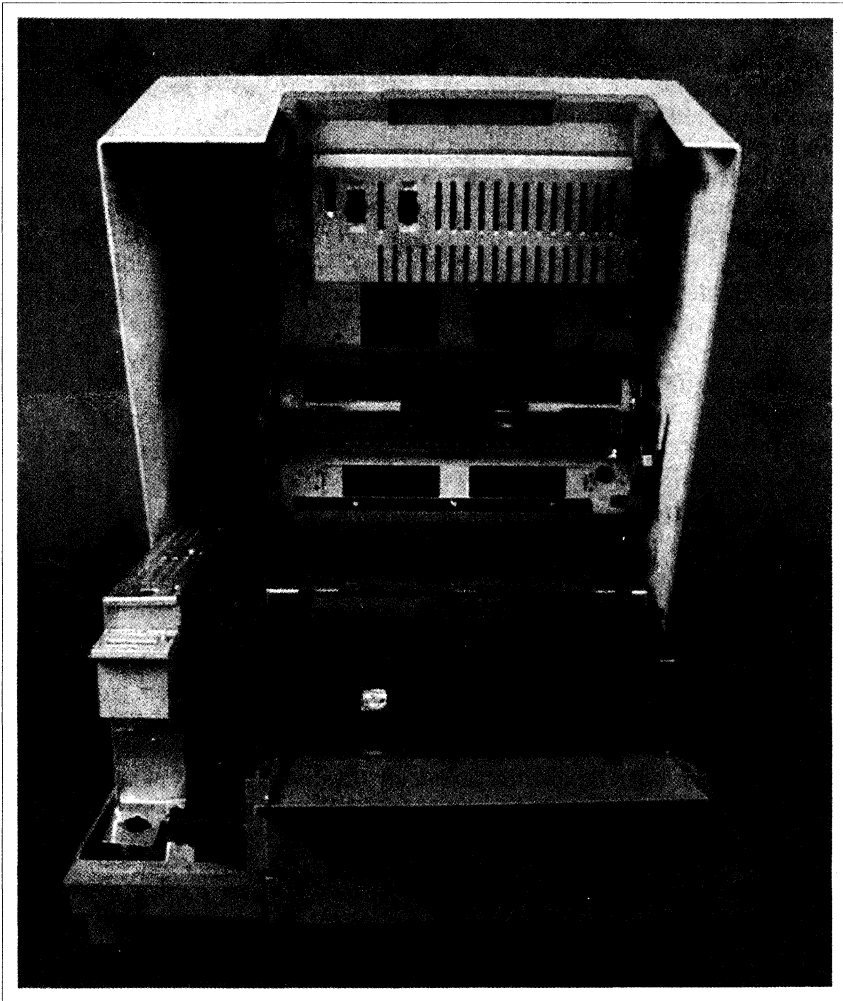


Abb. 4.8: Das Innenleben des Atari-Laserdruckers

Eine Tonerladung reicht je nach Anwendungsweise für ungefähr 3000 Seiten. Werden viele schwarze Flächen gedruckt, braucht man mehr Toner und wird deshalb schon erheblich früher zur Kasse gebeten - eine neue Patrone kostet um die DM 200,-. Auch sollte die Intensität Density mit einem Drehknopf an der Gehäuserückseite des SLM804/SLM605 auf die niedrigste Einstellung gebracht werden, die einen guten Ausdruck ergibt.

Der Controller SLMC804

Dies ist die Steuereinheit des Laserdruckers, die für die Verbindung zwischen Computer und SLM804 zuständig ist. Die Verbindung zwischen SLMC804, Drucker und Mega ST sowie einer oder mehrerer Festplatten erfolgt wie im ersten Kapitel beschrieben. Die beiden Leuchtdioden auf der Frontseite zeigen an, ob Daten an den Drucker gesendet werden (gelbe Diode) bzw. ob der Drucker und der Controller mit Strom versorgt werden (grüne Diode). Die Stromversorgung der Steuereinheit erfolgt über den Drucker. Der SLMC804 kann sechs grundsätzliche Kommandos, die im Handbuch zum SLM804/SLM605 dokumentiert sind, verstehen und an den Drucker liefern. Diese sind:

Request Sense (0x03)	Abfrage des Druckerzustandes (Status)
Print (0x0a)	Druck in Hochauflösung beginnen
Inquiry (0x12)	Kennung Laserdrucker abfragen
Mode Select (0x15)	Druckparameter einstellen
Mode Sense (0x1a)	aktive Druckparameter ermitteln
Stop Print (0x1b)	Endlosdruck stoppen

Ist der Laserdrucker komplett zusammengebaut, lohnt noch ein Blick auf die verschiedenen Möglichkeiten, Papier einzulegen. Es gibt zwei grundsätzliche Methoden: die Magazinzuführung und die manuelle. Letzterem Zweck dient der schmale Schlitz an der Frontseite des Druckers, oberhalb der großen Öffnung, in die das Papiermagazin eingesteckt wird. Hier können Sie fast alles hineinstecken, was aus Papier oder folienähnlich ist. Sogar Visitenkarten auf dünnem Karton oder Postkarten lassen sich bedrucken. Alle Blätter werden automatisch gewendet und liegen nach dem Ausdrucken in der richtigen Reihenfolge.



Verwenden Sie niemals den manuellen Einzug, wenn ein gefülltes Papiermagazin im Drucker ist - ein Papierstau wäre die Folge. Ebenso schädlich ist es, stärkeres Material wie z.B. dünnen Karton automatisch einziehen zu lassen.

Die Atari-Programme zum Laserdrucker

Dem SLM804/SLM605 liegt eine Diskette bei, ohne die Sie Ihrem Laserdrucker sicher nicht viel entlocken werden. Bevor wir den Drucker in Betrieb nehmen, müssen diese Programme installiert werden. Wir wollen uns die Funktion der Programme einmal der Reihe nach ansehen:

Diab630.PRG

Das ist das eigentliche Emulationsprogramm. Es wird ein Typenraddrucker des Typs Diablo630 emuliert, so daß mit verschiedenen Textprogrammen wie 1st Word Plus oder BECKERtext ST ein recht komfortables Arbeiten in Korrespondenzqualität möglich ist. Der Emulator erlaubt es zwar nicht, sämtliche bekannten Kommandosequenzen des Diablo-Typenraddruckers zu nutzen, die wichtigsten werden jedoch unterstützt. Hinzu kommen einige zusätzliche Möglichkeiten, die wiederum ein realer Diablo-Drucker nicht hat - beispielsweise das Drucken in Outline-Schrift. Das Programm Diab630.PRG wird normalerweise im AUTO-Ordner der Bootdisk bzw. dem entsprechenden Verzeichnis der Festplattenpartition C abgelegt. So wird der Emulator jedesmal automatisch gestartet, wenn das System gebootet wird. Das Programm beginnt dann mit dem Laden diverser Zeichensätze, im Englischen - und mittlerweile auch bei uns, nennt man solche Zeichensätze auch "Fonts", falls solche vorhanden sind und mit dem weiter unten beschriebenen Programm Setup630.PRG zuvor installiert wurden. Der Ladevorgang der Zeichensätze wird auf dem Bildschirm angezeigt.

Setup630.PRG

Dieses Programm dient der Anpassung der Emulatorfunktion an die Bedürfnisse des Anwenders bzw. der eingesetzten Textverarbeitung. Das Programm Diab630.PRG wird entsprechend verändert, so daß es beim Booten mit den geänderten Einstellungen gestartet wird. Neben der Einstellung verschiedener Druckparameter erlaubt das Programm die Installation verschiedener Zeichensätze, die mit dem Emulator verwendet werden können. Schließlich informiert Setup630.PRG noch über den aktuellen Status des SLM804/SLM605 (ziemlich überflüssig, denn diese Information gibt der Drucker schon über sein Anzeigenfeld) und erlaubt Testausdrucke, einen Reset des Druckers sowie, falls nötig, das Senden eines Formfeeds (Seitenvorschubs) an den Drucker, sofern noch Daten im Speicher sind, die aus irgendeinem Grund nicht ausgedruckt wurden.

Nach dem Starten des Programms durch Doppelklick auf Setup630.PRG erscheinen eine Leiste mit drei Rollmenüs sowie eine Menübox mit verschiedenen voreingestellten Druckparametern. Sie können diese Parameter nun verändern oder mit einem Klicken auf die Abbruchbox ohne Veränderungen fortfahren.

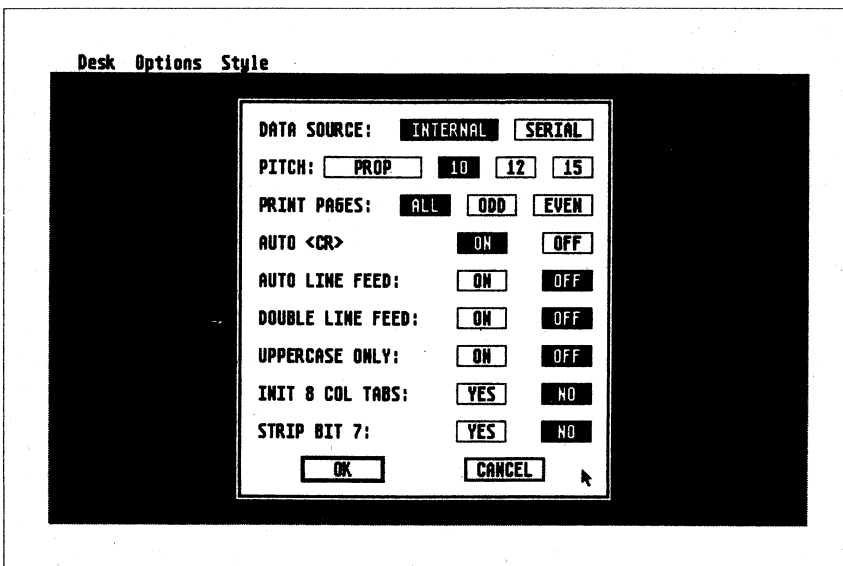


Abb. 4.9: Einstellen der Druckparameter mit Setup630

Die folgenden Einstellungen lassen sich vornehmen:

DATA SOURCE: Hier kann bestimmt werden, woher der Emulator seine zu druckenden Daten bezieht. Mit "INTERNAL" geschieht dies durch das verwendete Textprogramm, wird "SERIAL" gewählt, erfolgt die Datenzufuhr von einem über die serielle Schnittstelle angeschlossenen Gerät her. Es ist also möglich, mit einem Nicht-Atari-System (z.B. Amiga) Daten über den Laserdrucker auszugeben. Die Grundeinstellung ist allerdings "INTERNAL".

PITCH: Hiermit wird die Zeichendichte eingestellt, mit der gedruckt werden soll. Voreingestellt ist "PROP" bzw. Proportionalschrift. In diesem Falle kann von einer fixen Zeichendichte (gemessen in CPI bzw. "characters/inch") nicht gesprochen werden, da sie sich ständig je nach Breite der Zeichen ändert. Damit das verwendete Textprogramm auch intern eine korrekte Formatierung des

Textes vornimmt, muß die eingestellte Zeichendichte auch im Programm eingegeben werden.

PRINT PAGES: Die Einstellung teilt dem Drucker mit, ob sämtliche oder lediglich ungerade oder gerade nummerierte Seiten ausgedruckt werden sollen (interessant bei doppelseitigem Druck).

AUTO <CR>: Falls Ihre Textverarbeitung kein CR bzw. "carriage return" sendet, wird dies bei "ON" vom Emulator erledigt. Die Grundeinstellung ist "OFF".

AUTO LINE FEED: Nach jeder Druckzeile wird automatisch ein LF bzw. ein Zeilenvorschub an den SLM804/SLM605 gegeben. Entweder benutzt man diese Funktion, um ein fehlendes LF der Textverarbeitung zu ersetzen oder zum Drucken in zweizeiligem Abstand. Ähnliches gilt für DOUBLE LINE FEED, hier werden gleich zwei Zeilen eingefügt. Normalerweise sind diese Einstellungen "OFF".

UPPERCASE: Veranlaßt den Drucker, alle Ausgaben in Großbuchstaben vorzunehmen. Voreingestellt ist "OFF".

INIT 8 COL TABS: Es werden bei "ON" in Abständen von 8 Spalten Tabulatoren gesetzt. Default ist auch hier wieder "OFF".

STRIP BIT 7: Je nachdem, ob Ihre Textverarbeitung mit 7- oder 8-Bit-Daten arbeitet, muß dieser Parameter auf "ON" oder "OFF" stehen. Default ist "OFF", es wird mit 8 Bit gearbeitet.

Nach dem Einstellen der Druckparameter wird durch ein Klicken auf "OK" die Konfiguration erst einmal temporär gesichert. Später erfolgt dann das dauerhafte Sichern des veränderten Emulatorprogrammes mit Hilfe eines eigenen Menüpunktes (s.u.). Zuerst allerdings sollten die gewünschten Zeichensätze installiert werden.

Fonts installieren mit Setup630

Atari liefert auf der dem Laserdrucker beigelegten Diskette mit Hilfs- und Installationsprogrammen einige Fonts, die mit dem Rollmenü "Style" ausgewählt und installiert werden können. Dazu müssen sich die Fontdateien mit der Extension ".FNT" im gleichen Verzeichnis befinden wie das Setup630.PRG selbst. Die standardmäßigen Fonts auf der Diskette sind:

LISTING.FNT mit 15 CPI
TYPE10LS.FNT mit 10 CPI
ATTP10LS.FNT mit 10 CPI

Damit läßt sich schon einiges anfangen. Während der LISTING-Font mit einer Zeichendichte von 15 CPI nicht mit Schriftattributen wie kursiv oder outline versehen werden kann (der Versuch kann zu Datenverlusten führen), sieht das bei den anderen beiden Fonts schon anders aus.

Bevor wir jedoch zeigen, wie das Ausstatten von Fonts mit Attributen sowie die Installation dieser Fonts vor sich geht, noch ein Hinweis: Sollten die vorhandenen Zeichensätze dem Anwender nicht ausreichen, lassen sich jederzeit auch alle anderen eventuell vorhandenen GEM-Zeichensätze mit der Extension ".FNT" verwenden. Solche Fonts finden sich bei vielen Programmen unter GEM für den ST - so z.B. Easydraw.

Zum Installieren eines Fonts wird im Rollmenü "Options" die Option "Install Fonts" angewählt. Es erscheint die Select-Box des Bildes 52. Alle im aktuellen Verzeichnis vorhandenen Zeichensätze mit der Extension ".FNT" werden nun angezeigt. Diese stehen nun für die Installation bereit. Das bloße Vorhandensein genügt noch nicht. Der erste gewünschte Font wird nun einmal mit dem Mauszeiger angeklickt und erscheint in inverser Darstellung. Auf der rechten Seite der Box wird nun eines der leeren Felder ebenfalls angeklickt. Ein weiteres Klicken auf INSTALL läßt den ausgewählten Font am bisher leeren Platz namentlich erscheinen - der Font ist installiert. Bis zu acht verschiedene Zeichensätze lassen sich auf diese Weise installieren.

Die acht zur Verfügung stehenden Font-Einträge werden von Atari als "Slots" bezeichnet. Der Font auf dem obersten Platz der Slot-Liste wird als Default-Zeichensatz behandelt und kann, im Gegensatz zu den anderen Zeichensätzen, nicht mit REMOVE gelöscht werden. Ersetzt und verändert werden kann er aber schon. Das Aussehen eines Fonts kann übrigens mit "Examine Font" betrachtet werden. Dabei werden allerdings keine Attribute angezeigt, nur der unveränderte Zeichensatz. Mit TEMP ORDER können Fonts vorübergehend installiert werden, ohne diese dauerhaft zu speichern, mit RESTORE wird der

vor einer Veränderung bestehende Zustand wiederhergestellt - alle Veränderungen werden mit diesem Kommando rückgängig gemacht.

Zeichensatz-Attribute zuordnen

Das Aussehen der Fonts kann nun mit Hilfe des Rollmenüs "Style" verändert werden. Zu diesem Zweck wird der Fontname in der Liste der installierten Zeichensätze angeklickt. Danach kann man in "Style" zwischen "Bold" (Fett), "Lighten" (helle bzw. magere Schrift), "Italicize" (kursiv) und "Outline" (umrandet) auswählen. Es lassen sich auch Attribute kombinieren, z.B. fett und kursiv. Aktivierte Attribute werden durch ein Häkchen gekennzeichnet. Nach dem Anklicken der gewünschten Attribute werden diese mit "Install Style" dem jeweils invers dargestellten Font zugeordnet. Dabei erscheint der Fontname mit den gewünschten Attributen in der Liste.

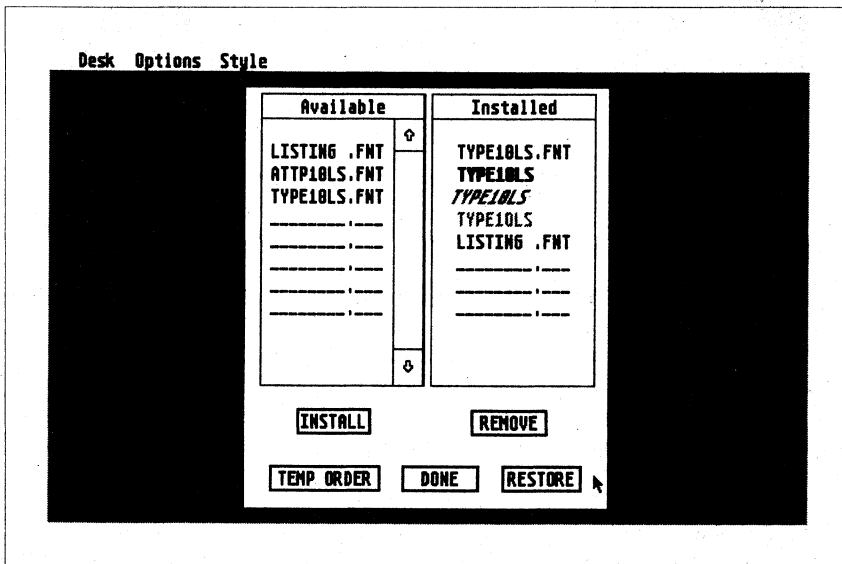


Abb. 4.10: Font-Installation mit Setup630 - in der rechten Liste befinden sich bereits vier mit unterschiedlichen Attributen installierte Versionen des Fonts TYPE10LS sowie der Font LISTING

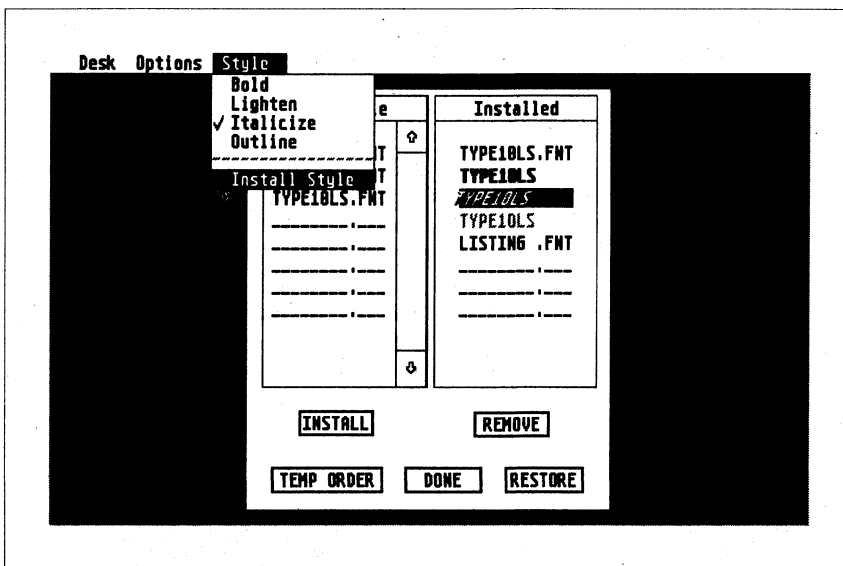


Abb. 4.11: Der Font TYPE10LS wird mit dem Attribut "kursiv" ausgestattet

Sichern von Diab630.PRG

Es können nun, wenn gewünscht, weitere Attribute zugeordnet werden. Sind alle Veränderungen vorgenommen, muß die neue Einstellung des Diab630.PRG noch gesichert werden. Hierzu klickt man auf DONE und anschließend im Rollmenü "Options" auf "Save Emulator". Die Box in Abbildung 4.12 erlaubt das Sichern des Emulators im aktuellen (current) Verzeichnis, also dort, wo sich das Setup630.PRG und die Fonts befinden. In diesem Falle muß das Programm jedesmal mit dem üblichen Doppelklick gestartet werden. Wahlweise kann man Diab630.PRG aber auch im AUTO-Ordner ablegen, dann erfolgt der Start beim Booten des Rechners. Weiterhin findet sich die Frage, ob man die "front panel changes" zu sichern gedenke. Hiermit sind eventuelle Veränderungen der Druckparameter gemeint ("Printer settings", s.o.). Sind dort keine Veränderungen erfolgt, lautet die Antwort NO. Dies ist Standard, falls lediglich Fonts geändert wurden. Nach dem Klicken auf OK wird das neue Emulator-Programm gesichert.

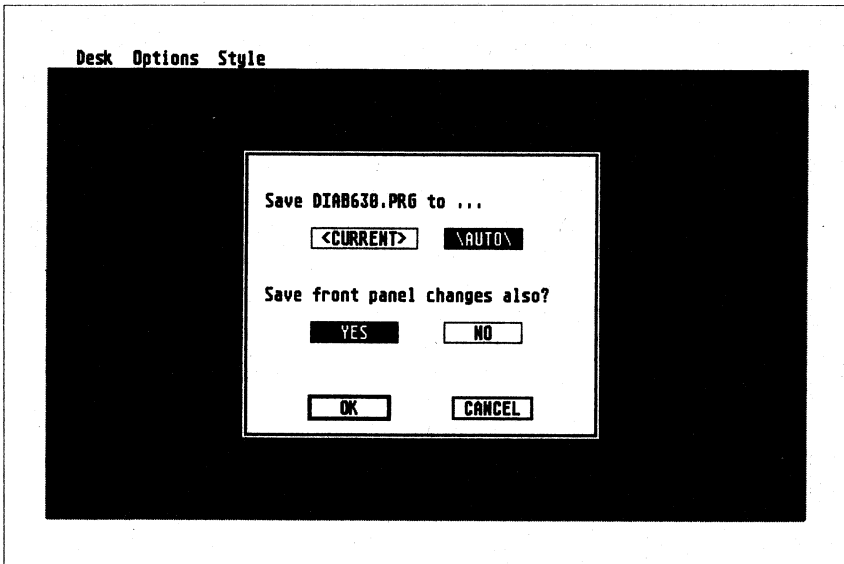


Abb. 4.12: Das Sichern des veränderten Diab630.PRG

Drucken mit Diab630

An zwei Beispielen wollen wir nun noch zeigen, wie man ein Textprogramm mit dem Diablo-Emulator einsetzen kann. Beide zeigen, wie die unterstützten Escape-Sequenzen eines Diablo-Typenraddruckers in eine Druckeranpassung eingebaut werden können. Voraussetzung für leichtes Einbinden ist, daß die verwendete Textverarbeitung Druckerdateien im ASCII-Format benutzt bzw. bereitstellt. Solche Dateien kann auch ein wenig erfahrener Computerbenutzer leicht verändern. Man findet diese Druckertreiber-Dateien z.B. bei BECKER-text ST und 1ST Word bzw. 1ST Word Plus. Die Datei-Extensionen sind ".PRT" (BECKERtext) und ".HEX" (1ST Word). Man lädt solche Files im ASCII- oder Source-Modus des jeweiligen Textprogrammes und speichert sie auch in diesem Modus ab. Die erste Spalte des Druckertreiber-Textes enthält jeweils die programminternen Steuerzeichen, dahinter folgen die Steuersequenzen für den Drucker bzw. die Emulation. Mit wenigen Ausnahmen beginnt jede Sequenz mit einem "Escape" bzw. dem entsprechenden Dezimalwert "27". Durch Kommata getrennt folgen dann die jeweiligen Steuercodes. Beispielsweise lautet die Initialisierungsanweisung für den Diablo630-Emulator/Drucker: 27,13,80. Eine Eingabe in ASCII ist ebenfalls möglich, dann lautet die Sequenz: ESC CR P. Wir wollen uns jedoch auf die dezimale Schreibweise beschränken.

Die Bedeutung der programminternen Codes geht aus den Handbüchern der entsprechenden Textverarbeitungen hervor. Eine einfache Anpassung für BECKERtext ST zeigt das Bild 55. Eine entsprechende Datei für 1ST Word Plus finden Sie in Bild 56. Die Anwahl des Zeichensatzes mit fetter und kursiver Schrift erfolgt mit speziellen Sequenzen, denn diese Schriftarten sind beim Emulator den Slots (Listenplatz in der Zeichensatz-Auswahlbox, s.O.) "2" (fett) und "3" (kursiv) fest zugeordnet. Sie sollten diese Attribute daher in diesen Slots installieren. Beim Aufruf des Attributs "fett" mit dem entsprechenden Diablo-Steuerkommando benutzt der Emulator automatisch den in "2" installierten Font, bei "kursiv" gilt dies ebenso für Slot "3". Weitere Zeichensätze werden immer mit dem gleichen Kommando ausgewählt - nur die Nummer des Slots ändert sich. Der Font im ausgewählten Slot wird dann zum Grund-Zeichensatz. Nach dem Aufruf und Einsatz des jeweiligen Zeichensatzes muß wieder auf den Font in Slot "1", den ursprünglichen Grund-Zeichensatz, zurückgeschaltet werden. Die übliche Sequenz zum Umschalten auf den Grund-Font funktioniert ja nicht mehr, weil ein anderer Zeichensatz diese Funktion jetzt ausübt.

Die folgenden Kommando-Sequenzen werden vom Diablo-Emulator für den SLM804/SLM605 verstanden und ausgeführt:

Grundfunktionen:

Leerzeichen:

ASCII: SP
Dezimal: 32

Backspace:

ASCII: S
Dezimal: 8

Carriage Return:

ASCII: CR
Dezimal: 13

Horizontaler Tabulator:

ASCII: HT
Dezimal: 9

Vertikaler Tabulator:

ASCII: VT
Dezimal: 11

Zeilenvorschub:

ASCII: LF
Dezimal: 10

Seitenvorschub:

ASCII: FF
Dezimal: 12

*Steuercodes Diablo 630/SLM804***Oberen Textrand setzen:**

ASCII: ESC T
Dezimal: 27,84

Unteren Textrand setzen

ASCII: ESC L
Dezimal: 27,76

Oberen und unteren Rand löschen:

ASCII: ESC C
Dezimal: 27,67

Linken Rand setzen:

ASCII: ESC 9
Dezimal: 27,57

Rechten Rand setzen:

ASCII: ESC 0
Dezimal: 27,48

Horizontaltabulator setzen:

ASCII: ESC 1
Dezimal: 27,49

Horizontalen Tabulator löschen:

ASCII: ESC 8
Dezimal: 27,56

Vertikalen Tabulator setzen:

ASCII: ESC -
Dezimal: 27,45

Tabulatoren gesamt löschen:

ASCII: ESC 2
Dezimal: 27,50

Zeilen je Seite

ASCII: ESC FF a (für a kann ein Wert von 1 bis 126 gewählt werden)
Dezimal: 27,12,a

Zeilendichte einstellen:

ASCII: ESC US a
Dezimal: 27,31,a

Die horizontale Bewegung des Druckkopfes wird nach der Formel $a-1$ gesetzt, wobei a einen Wert von 1 bis 126 einnehmen darf. a entspricht Einheiten von 1/120 inch.

Zeichendichte löschen:

ASCII: ESC S
Dezimal: 27,83

Absoluter Tabulator:

ASCII: ESC HT a
Dezimal: 27,9,a

Ein horizontaler Tabulator wird an der Position a gesetzt.

Automatischer Wagenrücklauf (carriage return):

ASCII: ESC ?
Dezimal: 27,63

Automatischer Wagenrücklauf aus:

ASCII: ESC !
Dezimal: 27,33

Rückwärtsdruck ein:

ASCII: ESC 6
Dezimal: 27,54

Vorwärtsdruck ein:

ASCII: ESC 5
Dezimal: 27,53

Zeilenvorschub setzen:

ASCII: ESC RS a
Dezimal: 27,30,a

Der Zeilenvorschub wird auf $a-1$ gesetzt, a entspricht Einheiten von 1/48 inch.

Absoluter Vertikaltabulator:

ASCII: ESC VT a
Dezimal: 27,11,a

Ein absoluter Tabulator wird auf die Zeile a gesetzt.

Halber Zeilenvorschub:

ASCII: ESC U

Dezimal: 27,85

Halber Zeilenvorschub zurück:

ASCII: ESC D

Dezimal: 27,68

Zeilenvorschub zurück:

ASCII: ESC LF

Dezimal: 27,10

Unterstreichen ein:

ASCII: ESC E

Dezimal: 27,69

Unterstreichen aus:

ASCII: ESC R

Dezimal: 27,82

Font in Slot 2 aktivieren (Fett):

ASCII: ESC O

Dezimal: 27,79

Font in Slot 3 aktivieren (Kursiv):

ASCII: ESC W

Dezimal: 27,87

Grundzeichensatz aktivieren (in der Regel in Slot 1):

ASCII: ESC &

Dezimal: 27,38

Dieses Kommando muß verwendet werden, um von Slot 2 oder 3 zurückzuschalten.

Font in Slot a als Grundfont installieren:

ASCII: ESC SO a

Dezimal: 27,14,a

Ein in einem der Slots 1-8 installierter Zeichensatz wird zum Grundfont. Der alte Grundfont kann nur durch erneutes Ausführen des Kommandos wieder eingestellt werden. a ist ein Dezimalwert der ASCII-Zeichen 1-8.

Rechtbündig drucken:

ASCII: ESC M

Dezimal: 27,77

Text zentrieren:

ASCII: ESC =
 Dezimal: 27,61

Druckkopf 1/120 inch zurück:

ASCII: ESC BS
 Dezimal: 27,8

Seitendruck gerade, ungerade bzw. alle:

ASCII: ESC # a
 Dezimal: 27,35,a

a kann sein: A (alle), E (gerade) oder O (ungerade Seiten)

Zurücksetzen aller Textverarbeitungseinstellungen:

ASCII: ESC X
 Dezimal: 27,88

Reset Diablo 630:

ASCII: ESC CR P
 Dezimal: 27,13,80

Es gibt noch einige wenige Grafikkommandos, die unterstützt werden. Damit läßt sich allerdings nur recht wenig anfangen, beispielsweise ein Rechteck zeichnen etc. Zum Grafikdruck ist der EPSON-FX-Emulator (siehe weiter unten) allerdings erheblich besser geeignet.

Grafik einschalten:

ASCII: ESC 3
 Dezimal: 27,51

Grafik ausschalten:

ASCII: ESC 4
 Dezimal: 27,52

Grafikzeichen drucken:

ASCII: ES " aaa SP
 Dezimal: 27,34,aaa,32

Hier ist "aaa" der dezimale ASCII-Wert eines Grafikzeichens.

Kasten zeichnen:

ASCII: ESC DC2 B www,xxxx,yyy,zzzz,ttt SP
 Dezimal: 27,18,66,www,xxxx,yyy,zzzz,ttt,32
 www linke obere Ecke, horizontale Koordinate
 xxxx linke obere Ecke, vertikale Koordinate
 yyy Breite des Kastens
 zzz Höhe des Kastens
 ttt Strichdicke

```

*****
* Druckerdatei zur Diablo-Emulation für 1ST Word Plus *
*****

SLM 884 * Druckerbezeichnung

* 5 Grundparameter: Nicht aktiv bei Diablo-Emulation
*
* 1: Druckertyp, 0=Matrix, 1=Typenrad
* 2: Zeichenbreite
* 3: Zeichenhöhe
* 4: Druckkopfposition nach Seitenvorschub
* 5: Doppeldruck
* 6: Pause zwischen Druckseiten (Einzelblattmodus)
*
0, 0, 0, 0, 0, 0 * Alle Parameter 0 bei SLM884

* Ab hier Steuersequenzen Diablo 630/SLM884 mit Emulation
*
1, 0, A          * Linefeed mit Return
* 2              * Vorwärts drucken
* 3              * Zurück drucken
* 5              * Absoluter Horizontaltabulator
6, 1B, 4F        * Fettschrift normal ein
7, 1B, 26        * Fettschrift normal aus
8, 1B, 4F        * Fettschrift NLQ ein
9, 1B, 26        * Fettschrift NLQ aus
A, 1B, 57        * Kursiv normal ein
B, 1B, 26        * Kursiv normal aus
C, 1B, 57        * Kursiv NLQ ein
D, 1B, 26        * Kursiv NLQ aus
E, 1B, 25        * Magere Schrift normal ein
F, 1B, 26        * Magere Schrift normal aus
10, 1B, 25       * Mager NLQ ein
11, 1B, 26       * Mager NLQ aus
12, 1B, 0E, 34   * Hochgestellt normal ein
13, 1B, 0E, 31   * Hochgestellt normal aus
14, 1B, 0E, 34   * Hochgestellt NLQ ein
15, 1B, 0E, 31   * Hochgestellt NLQ aus
16, 1B, 55, 1B, 0E, 34 * Tiefsetzen normal ein
17, 1B, 0E, 31, 1B, 44 * Tiefsetzen normal aus
18, 1B, 55, 1B, 0E, 34 * Tiefsetzen NLQ ein
19, 1B, 0E, 31, 1B, 44 * Tiefsetzen NLQ aus
1A, 1B, 45       * Unterstreichen normal ein
1B, 1B, 52       * Unterstreichen normal aus
1C, 1B, 45       * Unterstreichen NLQ ein
1D, 1B, 52       * Unterstreichen NLQ aus
1E, C            * Seitenvorschub
1F, 1B, 0D, 50   * Horizontale Initialisierung
21, 1B, 0D, 50   * Drucker Reset Diablo 630
0                * NULL-Byte zum Abschluß der Steuersequenzen

```

Abb. 4.13: Eine Druckerdatei für die Diablo 630-Emulation mit 1ST Word Plus

yyyy, zzzz und tttt werden in Punkten angegeben (1/300 inch), entsprechend der maximalen Auflösung des SLM 804.



Wenn Sie Fonts mit Setup630 installieren, kann Ihre Textverarbeitung das Aussehen dieser Zeichensätze natürlich nicht auf den Bildschirm bringen. Über das Aussehen informiert lediglich der Menüpunkt "Install Style" im "Style"-Rollmenü.

```

\*****
\*                               DIABLO 630 Typenrad für SLM884                               *
\*****
\0      27,13,80                  * DIABLO 630 Reset
\1+     27,79                    * fett ein
\1-     27,38                    * fett aus (zurückschalten auf Grundfont)
\2+     27,87                    * kursiv ein
\2-     27,38                    * kursiv aus
\3+     27,69                    * unterstreichen ein
\3-     27,82                    * unterstreichen aus
\4+     27,14,52                 * mager ein (Font in Slot 4 wird Grundfont)
\4-     27,14,49                 * mager aus (Grundfont zurück auf Slot 1)
\5+     27,68,27,14,53           * hochsetzen ein
\5-     27,14,49,27,85           * hochsetzen aus
\6+     27,85,27,14,53           * tiefsetzen ein
\6-     27,14,49,27,68           * tiefsetzen aus
\A      27,14,49                 * Grundfont mit 10 CPI in Slot 1
\B      27,14,54                 * Grundfont mit 12 CPI in Slot 6
\C      27,14,53                 * Listing-Font in Slot 5 ist Grundfont
\E      *
\A      10                       * 10 cpi
\b      12                       * 12 cpi
\c      15                       * 15 cpi
\e      5                        * 5 cpi Proportional
\n      13,10                   * neue zeile (CR/LF)
\JJ     27,77                   * Blocksatz
\X1.0   27,30,9                 * 1 zeilig
\X1.5   27,30,13                * 1.5 zeilig
\X2.0   27,30,17                * 2 zeilig

```

Abb. 4.14: Druckerdatei zur Diablo-Emulation mit BECKERtext ST 2.0

Setup630.RSC

Dieses Resource-File enthält wichtige GEM-Daten für das Setup.PRГ. Es muß vorhanden sein, wenn Setup630.PRГ gestartet wird.

Setup630.ACC

Hier handelt es sich um ein Desk-Accessory, das dem Accessory "Drucker Anpassung" entspricht, also Einstellungen wie Magazineinzug oder Manuell bzw. Einzelblatt ermöglicht. Es muß sich in dem Rootdirectory von Partition C der Festplatte bzw. auf der Boot-Diskette befinden, um geladen zu werden. Das Accessory erlaubt überdies den Ausdruck einer Testseite (mit Angabe der Version des Druckertreibers etc.), ein Reset des Druckers ist ebenso möglich wie eine Anzeige des Drucker-Status. Die Wirkung dieses Accessory ist schon dadurch zu erkennen, daß die Busy-Leuchtdiode des SLMC804 aufleuchtet, sobald es aktiviert ist - eine Hardcopy des Bildschirms ist während der Parameterübergabe nicht möglich.

Sdump.PRГ

Dieses Programm leitet die Hardcopy-Funktion, erzeugt durch die Tastenkombination **[Alternate]** + **[Help]**, auf den DMA-Port um und damit an den SLM804/SLM605. Man kann das Programm entweder durch den üblichen Doppelklick starten, oder man bindet es in den AUTO-Ordner ein. Dann steht die Hardcopy-Funktion nach jedem Boot-Vorgang automatisch zur Verfügung. Mit Sdump.PRГ lassen sich in Ermangelung vorhandener Treiber sehr schön die Grafiken eines Zeichenprogramms ausgeben. Dabei sind "Stufen" in den Grafiken natürlich zu sehen - der Ausdruck entspricht exakt dem Anblick auf dem Bildschirm. Mit einem geeigneten Treiber würde der SLM804/SLM605 runde "Ecken" auch wirklich rund, also ohne die "Treppchen" darstellen.

1st_PRNT.DOT und SLM804.HEX

Dies sind nichtinstallierte Druckertreiber-Dateien für die Textprogramme 1st Word (.DOT) und 1st Word Plus (.HEX). Sie können wie üblich in diesen Textverarbeitungen installiert werden. Veränderungen ergeben allerdings meist nur mangelhafte Ergebnisse. Schriftattribute sollten nur mit SETUP630 verändert werden. Steuerzeichen werden offenbar falsch oder gar nicht interpretiert.

Fonts

Auf der Atari-Diskette befinden sich noch weitere Dateien mit der Extension .FNT. Hierbei handelt es sich um die Zeichensätze für den Diablo-Emulator. Mit Install Fonts können diese installiert werden.

Test.PRG

Dieses kleine Programm funktioniert sofort nach dem Installieren und Inbetriebnehmen des Druckers. Klicken Sie einfach auf das Programm-Icon. Es veranlaßt den SLM804/SLM605 zum Ausdrucken eines Testmusters. Mehr als eine reine Funktionskontrolle ist mit diesem Programm nicht möglich.

Erste Lebenszeichen

Der Drucker ist nach dem Installieren bzw. Starten der oben genannten Programme betriebsbereit. Beachten Sie in jedem Falle die Reihenfolge beim Einschalten: Drucker einschalten und warten, bis die grüne Lampe an der Frontseite aufleuchtet. Danach Festplatte und sonstige Peripherie und ganz zuletzt den Mega ST einschalten.

Wenn Sie jetzt auf Test.PRG klicken, müßte der SLM804/SLM605 eine Testseite ausgeben. Haben Sie Sdump.PRG im AUTO-Ordner eingerichtet, können Sie jetzt eine Hardcopy des Bildschirms ausgeben lassen - betätigen Sie **Alternate** + **Help**. Sollte der Drucker das Papier "verschmieren", so verstellen Sie den Knopf Density, bis das Druckbild überzeugend ist. Und nun können Sie, wenn Sie mit 1st Word und Typenrad-Emulation nicht zufrieden sind, auf die Suche nach Programmen mit SLM804/SLM605-Treiber gehen.

Ein Epson-Emulator für den SLM 804

Der ST ist hervorragend für die Arbeit mit einem Drucker des Herstellers Epson bzw. dazu kompatiblen Geräten geeignet. Darauf weist schon die Hardcopy-Funktion hin, die für einen solchen Drucker eingerichtet ist. Die Zeichensätze entsprechen zu einem großen Teil dem sogenannten IBM-Zeichensatz - auch damit ist der Epson-Drucker vertraut. Als Urtyp aller Epson- bzw. kompatiblen Geräte gilt der FX80 bzw. dessen Nachfolger FX85 oder FX800. Andere Drucker, wie die der Epson-LQ- oder LX-Reihe, verfügen meist auch über die Funktionen dieser Geräte.

Wegen der genannten Ausrichtung auf diese Drucker bzw. deren weiter Verbreitung statteten alle Softwarehersteller ihre Produkte mit Druckertreibern für Epson-Drucker aus - meist für das Modell FX80 bzw. dessen Nachfolger. Da auch viele andere Druckerhersteller ihre Geräte "Epson-kompatibel"

machten, brauchen die Anwender keine Sorge wegen eines fehlenden Treibers zu haben. Ganz anders sieht das auf den ersten Blick bei den Besitzern eines SLM 804 aus. Doch Hilfe ist auf dem Wege! Denn alle Programme mit Epson-Treibern können auch mit dem Atari-Laserdrucker ausdrucken. Wie funktioniert das?

Atari stellt den Käufern eines SLM 804 einen sogenannten FX-Emulator zur Verfügung. Dieses Programm emuliert, ähnlich dem Diablo-Emulator der SLM 804-Diskette, einen solchen Drucker - der Atari SLM 804 kann nach dem Ausführen des Programms Befehle verstehen, die für den FX80 oder andere Epson-Drucker gedacht sind. Der Emulator wird in den AUTO-Ordner der Boot-Diskette bzw. Partition C der Festplatte kopiert und startet so automatisch beim Einschalten des Computers. Danach startet man seine Anwendung unter Einsatz eines vorhandenen Treibers für den FX oder andere Epson-Drucker. Beispiel: Bei der Textverarbeitung BECKERText ST wird nach dem Starten des Emulators das Textprogramm gestartet und anschließend als Drucker ein IBM-Drucker bzw. die Epson-Anpassung ausgewählt. Schon läßt sich alles mit dem SLM 804 zu Papier bringen. Und damit nicht genug - per Dialogbox lassen sich auch die speziellen Möglichkeiten des Laserdruckers nutzen. Beispielsweise werden Stufen "geglättet", und die Auswahl verschiedener Fonts ist ebenfalls möglich. Der SLM 804 "ist" jetzt zwar FX80, das Druckbild jedoch von erheblich höherer Qualität. Das gilt auch für die Hardcopy-Funktion über die Tasten **Alternate** + **Help**. So sind beispielsweise keine Einzelpunkte wie bei der Matrix-Technik des FX zu erkennen, Schwarztöne sind wirklich als schwarze Flächen dargestellt usw. Der FX-Emulator, der ehemals als Produkt des Softwarehauses DMC mit der Bezeichnung "Laserbrain" das Licht der Welt erblickte, dürfte den meisten Anwendern mit Atari-Laser wie gerufen kommen. Übrigens sollten Sie immer eine Anpassung für IBM-Kompatible-Drucker benutzen, wenn Sie Bilder ausdrucken wollen, da der FX-Emulator sonst immer einen kleinen Zwischenraum zwischen den einzelnen Druckzeilen erzeugt und damit das Bild auseinander zieht.

Das Beste ganz zum Schluß: Der FX-Emulator kostet nichts - oder fast nichts. Aufzuwenden sind lediglich die Energie für einen Gang zum Atari-Händler sowie eine Leerdiskette. Beim Händler kann das Programm nämlich kostenlos kopiert werden. Wieso Atari das nützliche Utensil z.Zt. nicht gleich auf die dem Laser beiliegende Diskette kopiert - diese Frage wird wohl kaum jemand beantworten können.

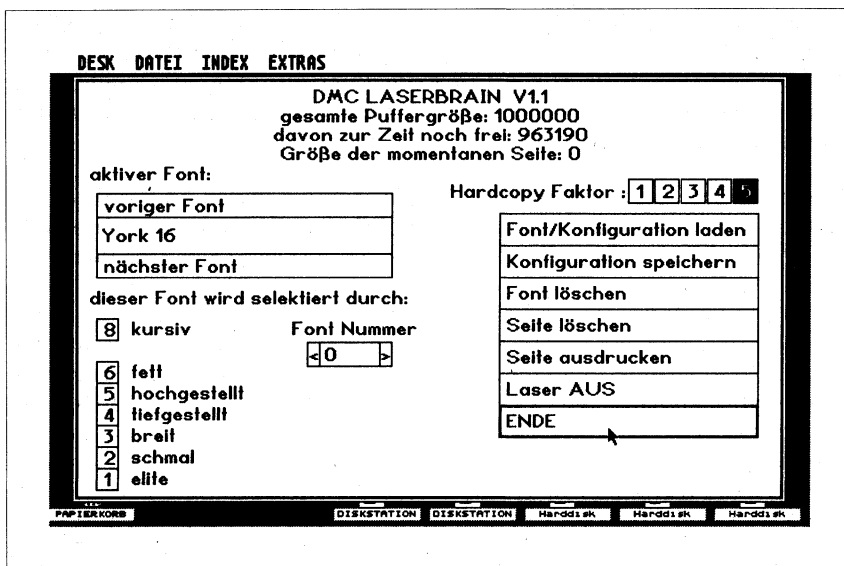


Abb. 4.15

5. Der Bildschirm

Kein anderes Teil eines Computer-Systems wirkt so direkt auf den Menschen ein wie der Bildschirm. Betrachtet man allein den gesundheitlichen Aspekt, dann wird deutlich, daß man dieser Frage mehr Aufmerksamkeit schenken sollte als dies gemeinhin üblich ist.

Zunächst können Sie an jedes ST-Modell die Atari-Monitore SM124 und SC1224 anschließen. Der SM124 ist der mittlerweile berühmte Schwarzweiß-Monitor, der bezüglich seiner Darstellungsqualität beispielhaft ist. Mit diesem Gerät können Sie nur in der höchsten Auflösungsstufe arbeiten. SC1224 ist die Typenbezeichnung des Atari-Farbmonitors, der rein äußerlich dem SM124 recht ähnlich sieht. Dieses Gerät kann die Auflösungsstufen mittel und niedrig darstellen. Die beiden Atari-Monitore können die jeweils andere Auflösung nicht darstellen und werden vom Rechner automatisch erkannt. Neben diesen Geräten können Sie mittlerweile eine Reihe speziell angepaßter Monitore von Fremd-Herstellern anschließen. So sind sogenannte Multi-Sync-Monitore auf dem Markt, die mittels spezieller Schaltungen alle drei Auflösungsstufen in gleich guter Qualität abbilden können. Wichtig: Monitore von Fremdherstellern müssen speziell angepaßt sein, um am ST zu funktionieren. Fragen Sie Ihren Fachhändler. Schließlich können Sie an einige ST-Modelle (520 ST/M etc.) auch Ihren heimischen Fernseher anschließen. Diese Alternative kann aber bei dauerndem Gebrauch auf keinen Fall empfohlen werden, da die Darstellungsqualität weit hinter den "Profi"-Monitoren zurückbleibt. Mit einem speziellen Adapterkabel können Sie auch an jeden ST einen Fernseher mit SCART-Buchse anschließen. In diesen Fällen weiß Ihr Händler Rat. Sie erkennen die SCART-Buchse an Ihrem Fernseher durch ihre ungewöhnliche Form.

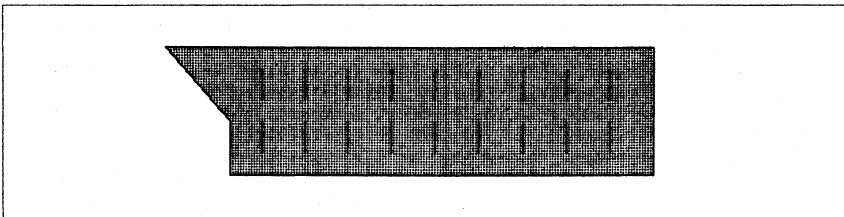


Abb. 5.1: So sieht eine SCART-Buchse aus



Wenn Sie einen Farbmonitor mit minderer Qualität besitzen, können Sie die Darstellung evtl. ein wenig aufwerten. Dazu müssen Sie im

Kontrollfeld die Farben so einstellen, daß der Hintergrund dunkel erscheint, während die Zeichen mit möglichst kontrastreichen hellen Farben dargestellt werden.



Versuchen Sie auf keinen Fall, einen nicht angepaßten Monitor an Ihren ST anzuschließen. Das führt in aller Regel zur Zerstörung der Hardware - sowohl Rechner als auch Monitor können dabei Schaden nehmen.



Monitore und Fernseher arbeiten nach dem gleichen Prinzip. Das darzustellende Bild wird in Zeilen zerlegt und diese Zeilen jeweils an den Monitor übertragen. Das ankommende Signal steuert einen Elektronenstrahl, der von einer "Elektronenkanone" in der Bildröhre erzeugt wird. Je nachdem, ob in der gerade geschriebenen Zeile ein Punkt leuchten soll oder nicht, wird der Strahl ein- bzw. ausgeschaltet. Bei eingeschaltetem Strahl treffen die abgeschossenen Elektronen auf eine Phosphorschicht unter der Sichtfläche der Bildröhre. Da Phosphor die Eigenschaft besitzt, unter Elektronenbeschuß zu leuchten, entsteht so Punkt für Punkt ein Bild auf dem Schirm. Schwarzweiß-Monitore arbeiten mit einem einzigen Strahl, während bei Farbbildschirmen drei Strahlen unterschiedliche leuchtende Phosphorschichten beschießen. Je nachdem, ob und wie stark rot, grün oder blau leuchtender Phosphor getroffen wird, entsteht eine andere Farbe. So lassen sich durch geschickte Mischung alle Farben auf dem Bildschirm erzeugen.

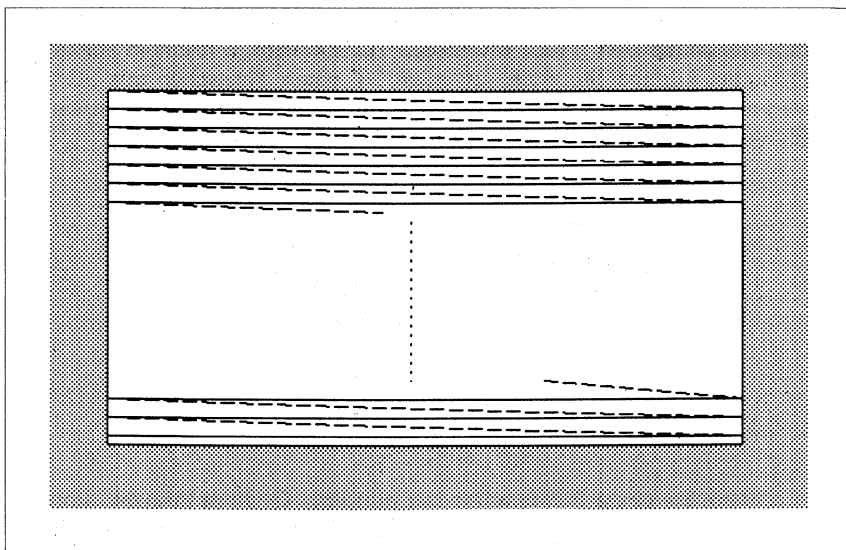


Abb. 5.2: Zeilenweiser Aufbau des Bildes

Das Bild auf dem Schirm wird nun zeilenweise aufgebaut. Immer wenn eine Seite fertig erzeugt wurde, beginnt der "Zeichenvorgang" wieder in der linken oberen Ecke. Dadurch, daß der Phosphor eine gewisse Zeit nachleuchtet, erhält unser träges Auge den Eindruck eines stehenden Bildes. Die Anzahl der Bilderzeugungen pro Sekunde wird auch Bildwiederhol- oder vertikale Ablenkfrequenz genannt und ist wesentlich für die Qualität der Darstellung. Fernseher haben eine Bildwiederholfrequenz von 25 Hz, der SM124 dagegen arbeitet mit 71 Hz. Je höher dieser Wert ist, desto flimmerfreier erscheint das Bild.

Im Unterschied dazu ist die horizontale Ablenkfrequenz zu sehen, die angibt, wie schnell eine einzelne Bildzeile erzeugt werden kann. Dieser Wert muß genau auf den jeweiligen Computer angepaßt sein, damit der Rechner den Monitor bei der Bilderzeugung nicht überholt. Deshalb kann nicht jeder Monitor an jeden Rechner angeschlossen werden.

Eine recht neue Entwicklung stellen die sogenannten Multi-Sync-Monitore dar. Diese sind mit einer speziellen Elektronik ausgestattet, die die Frequenzen ermitteln kann, mit denen der Computer arbeitet. Der Monitor stellt sich dann automatisch auf den optimalen Wert ein, so daß er mit fast jedem Rechner benutzt werden kann. Solche Geräte sind die einzigen, die nach einer gewissen Anpassung alle drei Auflösungen des ST darstellen können.

Eine vielgebrauchte Abkürzung im Zusammenhang mit Monitoren ist RGB. RGB steht für rot-grün-blau und bezeichnet Farbmonitore, bei denen die drei Elektronenstrahlen direkt vom Computer gesteuert werden können. Im Normalfall erhält ein Farbmonitor ein sogenanntes Composite-Signal, das die Steuerung aller drei Farben gemeinsam überträgt. Bei dessen Decodierung geht im Normalfall ein wenig von der Darstellungsqualität verloren. RGB-Geräte erhalten diese Signale von vornherein getrennt und brauchen diesen Verlust daher nicht in Kauf zu nehmen. Der ST bietet an seinem Monitorausgang beide Signalvarianten an. Farbfernseher arbeiten meistens mit dem Composite-Signal, so daß sie allein aus diesem Grunde schon eine geringere Darstellungsqualität bieten.

Ein weiteres Kürzel, CRT, ist die Abkürzung für Cathode Ray Tube, was nichts anderes als der englische Begriff für Bildröhre ist.

Nun heißt es, den Monitor zu plazieren. Am besten findet der Bildschirm auf einem speziellen Monitorständer Platz, mit dem Sie das Gerät bei Bedarf mit einem Handgriff in die richtige Position bringen können. Der Ständer sollte genau gegenüber Ihrer Sitzposition stehen, um unnötige Beanspruchungen des Nackens zu verhindern. Sie sollten dabei darauf achten, daß die Kabel nicht

geknickt werden und daß das Tageslicht möglichst seitlich auf den Bildschirm trifft, um Spiegelungen zu vermeiden.

Im Normalfall wird der Monitor an der gleichnamigen DIN-Buchse auf der Rückseite des ST angeschlossen. Bei der Verwendung von Fremdherstellengeräten ist es unter Umständen nötig, einen Adapter zwischenzuschalten. Bei Modellen mit eingebautem Modulator (520 ST/M etc.), die auch den Anschluß eines Fernsehers erlauben, befindet sich eine weitere Buchse, die das Antennenkabel aufnimmt.

Pannenhilfe



Bei Herumrücken der Geräte ist auch das Monitorkabel bewegt worden. Daraufhin hat der ST einen Reset ausgeführt.



Hier liegt kein Hardwarefehler vor. Wahrscheinlich ist das Monitorkabel nicht richtig in die Buchse eingesteckt (die Qualität der Buchsen und Stecker ist nicht gerade ein Ruhmesblatt für Atari) und bei der Bewegung aus dem Gerät gerutscht. Wann immer der ST feststellt, daß der Monitor-Anschluß wechselt, führt er automatisch einen Reset durch. Kontrollieren Sie also den Stecker, und sichern Sie ihn gegen weiteres Verrutschen.

Der ST verfügt über drei verschiedene Auflösungsstufen. In der Auflösungsstufe "hoch" stehen Ihnen 640 x 400 Bildpunkte in zwei Farben (schwarz und weiß) zur Verfügung. Diese Auflösung ist am besten für die professionelle Nutzung geeignet (insbesondere Textverarbeitung etc.), weil sie sich durch hohe Qualität auszeichnet. Die Auflösungsstufe "mittel" hat 640 x 200 Punkte bei 4 Farben. Wie in der höchsten Auflösung können Sie auch hier 80 Zeichen pro Zeile darstellen, allerdings erscheinen sie ein wenig in der Höhe gestreckt. Auch die Grafiken wirken gedehnt. Falls Sie einen Farbmonitor besitzen, können Sie diese Auflösung einsetzen, um Textverarbeitung u.ä. zu betreiben. In der niedrigen Auflösung können Sie nur noch 320 x 200 Punkte darstellen lassen - dies allerdings bei 16 Farben. Dies reicht nur noch für 40 Zeichen pro Zeile (in der normalen Schriftart). Grafiken wie Zeichen erscheinen nicht verzerrt, letztere aber etwas klobig. Sie sollten diesen Modus vor allem für Grafiken und Spiele benutzen.

Sofern das Umschalten zwischen den verschiedenen Stufen überhaupt möglich ist, können Sie die Auflösung im Desktop unter dem Menüpunkt "Voreinstellung" ändern. Aber dabei wird auf jeden Fall ein Reset ausgeführt!

Außer der Auflösung müssen Sie am ST eigentlich nichts einstellen. Beim ersten Mal sollten Sie allerdings am Monitor selbst die Helligkeit und den Kontrast einregeln - das geht genau wie beim Fernseher. Die Regler dazu finden Sie an der rechten Seite des Gerätes. Außerdem müssen Sie die Lautstärke auf den gewünschten Wert drehen, da alle Tonausgaben über den eingebauten Lautsprecher des Monitors laufen. Wenn Sie vergessen, den Ton richtig aufzudrehen, hören Sie unter Umständen nichts von Ihren Programmen (was allerdings auch einmal ganz angenehm sein kann).

Anwender, die zwei Monitore benutzen (einen für die Farbdarstellung und einen für den Schwarzweiß-Modus), werden eine kleine Umschaltbox begrüßen, die das lästige Auswechseln der Stecker überflüssig macht. Je nach persönlichen Gegebenheiten kann außerdem ein Entspiegelungsvorsatz sehr sinnvoll sein. (unbedingt im Laden testen - hier gibt es viel Schnickschnack.).

6. Tastatur und Maus

In der Tastatur überwacht ein eigener kleiner Prozessor eine ganze Reihe von Schaltern, nämlich die Tasten. Wird einer dieser Schalter betätigt, so merkt das der Tastaturprozessor. Gleichzeitig ermittelt er die Nummer der Taste und sendet diese an den ST. Dieser kann dann seine eigenen Schlüsse aus der übermittelten Tastennummer ziehen.



Bei genauer Betrachtung entpuppt sich die Tastatur als kleiner Computer im Computer, der vielfältige Aufgaben zu erledigen hat. Wie jeder richtige Computer besitzt der Tastaturprozessor RAM und ROM (allerdings sehr klein) sowie eine Schnittstelle zum Hauptrechner. Man hat diese Methode gewählt, um den Hauptrechner von zeitraubenden Routinearbeiten zu entlasten. So braucht der 68000 nicht ständig alle Tasten zu überwachen, sondern kann darauf warten, daß der Tastaturprozessor von ganz alleine die Betätigung einer Taste meldet. Hinzu kommen noch diverse andere Tätigkeiten, die in der Tastatur zunächst ohne Beteiligung des Hauptrechners ablaufen. So wird auch die Maus über die Tastatur gesteuert, der Tastaturprozessor meldet die Bewegungen bei Bedarf weiter. Schließlich hat auch noch die Uhr in der Tastatur ihren Sitz. Uhrzeit und Datum können hier bei Bedarf abgeholt werden.

Der Tastaturprozessor liefert auch keineswegs den ASCII-Code der gedrückten Taste beim Hauptrechner ab. Letzterer erhält wirklich nur eine interne Tastennummer und zusätzliche Informationen, ob Tasten wie **Shift**, **Ctrl** etc. gedrückt sind. Der Hauptrechner muß aus diesen Informationen den ASCII-Code des gewünschten Zeichens selbst errechnen. Hintergrund dieser Vorgehensweise: Der ST wird je nach Land mit vielen verschiedenen Tastaturen ausgeliefert. So werden Sie auf einem USA-ST wohl kaum die deutschen Umlaute finden. Um nun nicht für jede Tastatur eine spezielle Version des Prozessors einbauen zu müssen, wird statt dessen einfach das Umrechnungsprogramm im Hauptrechner gewechselt. Dieses sitzt im ROM, das sowieso auf das jeweilige Land angepaßt werden muß (für deutsche Warnmeldungen etc.).



Aus technischen Gründen hat der Tastaturprozessor eine recht unangenehme Eigenschaft. Halten Sie die Mausknöpfe während des Bootens gedrückt, so kann der Tastaturprozessor abstürzen. Sie erkennen das daran, daß zwar das Desktop wie gewohnt erscheint, aber sämtliche Ein- und Ausgaben per Maus oder Tastatur unmöglich sind - der Mauspfel rührt sich nicht von der Stelle. In solchen Fällen hilft nur noch neues Booten. Achten Sie deshalb immer darauf, daß die Maustasten beim Einschalten des Rechners nicht verklemmt oder sonstwie niedergedrückt sind.

Wie Sie sicher schon bemerkt haben, hat die Tastatur auch ein gewisses Eigenleben - so zum Beispiel die automatische Tastenwiederholung. Diese ist leider nicht abzustellen, Sie können aber über das Kontrollfeld (siehe Kap. 2) die Geschwindigkeit und die Verzögerung der Wiederholung einstellen. Die Klickgeräusche aus dem Monitorlautsprecher bei jedem Druck auf eine Taste sollen Ihnen ein genaues Gefühl für den Anschlag verschaffen. Erst wenn der Klick ertönt, hat die Taste wirklich ausgelöst. Bei Bedarf können Sie diese Option im Kontrollfeld ausschalten.

Ein hervorstechendes Merkmal des ST sind seine schrägen Funktionstasten. Für den normalen Betrieb Ihres ST sind diese nicht nötig. Ihr Sinn liegt vielmehr darin, in Anwenderprogrammen bestimmte Funktionen auszulösen (daher der Name). Welche Aktionen Sie damit aktivieren können, ist von Programm zu Programm unterschiedlich. Sehr beliebt in Textverarbeitungen ist die Möglichkeit, verschiedene Floskeltexte (z.B. "Sehr geehrte...") auf die Funktionstasten zu legen, um diese auf Tastendruck in den Text einzufügen.

Rechts unten am alphanumerischen Tastenfeld befindet sich die **Caps Lock**-Taste. CAPS ist die Abkürzung für CAPitalS, also Großbuchstaben. Sie kennen sicher von Ihrer Schreibmaschine her die feststellbare Umschalttaste für Großbuchstaben. (Nebenbei: Die Umschalttaste heißt auf Ihrem ST **Shift**.). **Caps Lock** wirkt so ähnlich. Nach einem Druck auf **Caps Lock** werden bei einem Druck auf eine Buchstabentaste nur noch Großbuchstaben erzeugt, die anderen Tasten (Zahlen, Sonderzeichen usw.) werden jedoch wie bisher behandelt. Wenn Sie also einen Text nur in Versalien (Großbuchstaben) eingeben möchten, benutzen Sie diese Funktion. Ein weiterer Druck auf **Caps Lock** bewirkt die Rückkehr zum normalen Modus.

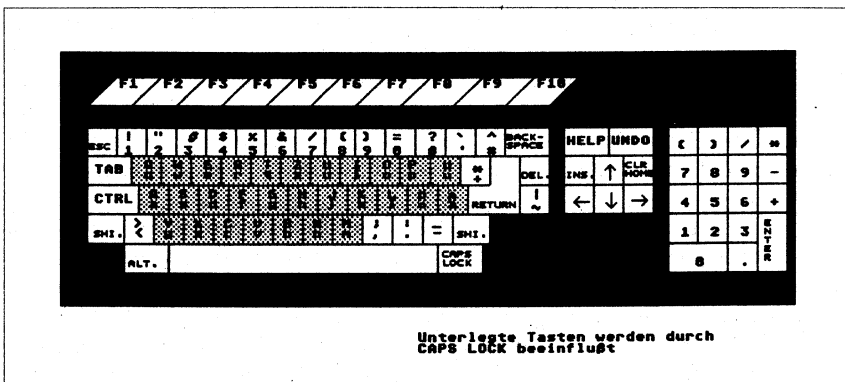


Abb. 6.1: Die Tastaturbelegung (unten: einfacher Tastendruck, oben: einfacher Tastendruck mit **Shift**)

▼ In vielen Programmen hat **Caps Lock** die unangenehme Wirkung, daß die Cursortasten nicht mehr so richtig funktionieren. Hinzu kommt, daß nicht wie auf anderen Computern eine kleine Warnleuchte auf die **Caps Lock**-Funktion aufmerksam macht. Aus diesem Grund übersieht man diese Möglichkeit sehr leicht.

Mit der **Alternate**-Taste können Sie eine Art dritte Ebene Ihrer Tastatur erreichen. Die erste Ebene stellen die normalen Buchstaben und Zeichen dar, die zweite wird über den zusätzlichen Druck auf **Shift** erreicht: Großbuchstaben und andere Sonderzeichen. Mit **Alternate** erreichen Sie jene Zeichen, die auf einigen Tasten an der rechten Seite klein eingezeichnet sind, z.B. die eckigen Klammern. Die vierte Ebene Ihrer Tastatur erreichen Sie schließlich über **Shift** und gleichzeitiges **Alternate**. Damit erreichen Sie die Zeichen, die klein rechts oben auf den Tasten markiert sind, z.B. den Backslash ("****") oder die geschweiften Klammern.

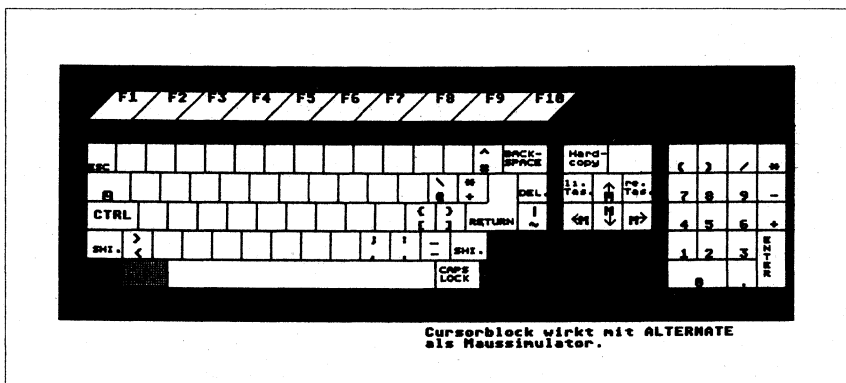


Abb. 6.2: Tastenbelegung bei gedrückter **Alternate**-Taste (unten: ohne **Shift**, oben: mit **Shift**)

Eigentlich auf jeder Computer-Tastatur ist **Ctrl** vorhanden. Der Name deutet es an: Mit der **Ctrl**-Taste können Sie den Rechner oder Programme in gewissen Grenzen kontrollieren und steuern. Vom Prinzip her arbeitet **Ctrl** wie **Shift** und **Alternate**. Sie halten die Taste fest und drücken dann eine Zeichen- oder Sonderzeichentaste. In den meisten Fällen erreichen Sie dadurch die Erzeugung eines sogenannten Control-Codes. Der bekannteste Code dieser Art ist **Ctrl-C**, die Kombination, mit der einige Programme (und auch die Dateianzeige des Desktop) abgebrochen werden können. Andere Programme reagieren ähnlich auf **Ctrl-G** oder **Ctrl-D**. In einigen wenigen Programmen kann eine Bildschirmausgabe durch **Ctrl-S** angehalten werden.

Mit **Ctrl**-**Q** geht es dann weiter. **Ctrl**-**M** bedeutet schließlich in vielen Fällen das gleiche wie ein Druck auf die **Return**-Taste.

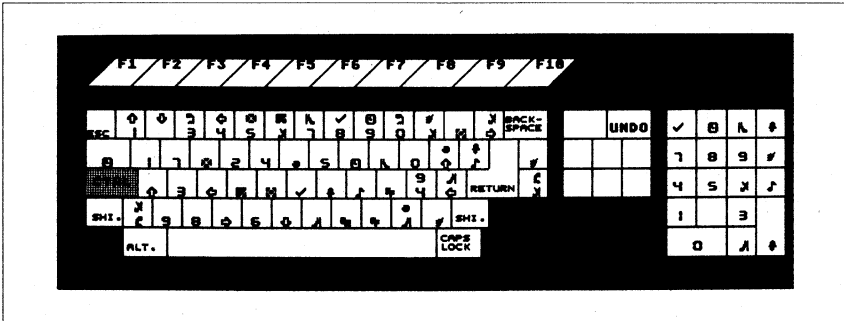


Abb. 6.3: Tastenbelegung mit **Ctrl** (unten: ohne **Shift**, oben: mit **Shift**)

Esc ist die Kurzform von ESCape, also zu deutsch "Flucht". Das ist ein wenig passender Ausdruck. In den meisten Fällen können Sie mit der **Esc**-Taste die Erzeugung einer Steuersequenz einleiten oder Befehle an ein Programm übergeben. Das Desktop reagiert auf die **Esc**-Taste auf zwei Weisen. Bei der Bearbeitung einer Eingabezeile wird diese gelöscht. Befindet sich ein Fenster auf dem Desktop, so versucht der ST, das darin angezeigte Inhaltsverzeichnis zu aktualisieren. Wichtig bei der Verwendung von **Esc** ist: Diese Taste muß nicht festgehalten werden, wie das bei **Shift** u.ä. der Fall ist.

Manchmal schwer zu unterscheiden sind **Backspace** und **Delete**. Im Normalfall bedeutet **Backspace**, daß in einer Dialogbox oder sonstigen Eingabe das Zeichen links vom Cursor gelöscht wird. Dabei wandert der Cursor um eine Stelle nach links. Im Gegensatz dazu löscht **Delete** das Zeichen, auf das der Cursor gerade zeigt. Der Cursor bleibt dann an der gleichen Stelle. Diese Zuordnung gilt aber nicht für alle Programme. Sehr häufig findet man den Fall, das **Backspace** nur den Cursor nach links bewegt und **Delete** das Zeichen links vom Cursor löscht. Die genaue Ausgestaltung ist von Programm zu Programm verschieden.

Auch die Funktion der **Help**-Taste hängt wieder vom Programm ab. Vielfach kann man durch Druck auf **Help** einen kleinen Hilfstext abrufen, der einem die Funktionen des Programms erklärt. Auf dem Desktop ergibt sich noch eine weitere Einsatzmöglichkeit. Durch Druck auf **Help**, während Sie die **Alternate**-Taste festhalten, können Sie eine Hardcopy des Bildschirms auslösen. Mit der gleichen Tastenkombination können Sie die Ausgabe auch wieder abbrechen (siehe Kapitel 2).

Undo bedeutet eigentlich nur "ungeschehen machen". In vielen Programmen hat **Undo** dann auch die Funktion, die letzte Aktion wieder rückgängig zu machen. So erlauben viele Grafikprogramme per **Undo** das Löschen der zuletzt gezeichneten Figur.

Insert heißt auf deutsch "einfügen". Je nach Programm ist die Wirkung dieser Taste verschieden. Im Desktop erreichen Sie durch die Kombination **Alternate**-**Insert** das gleiche wie durch einen Druck auf die linke Maustaste. In Eingabezeilen wird durch **Insert** ein Zeichen Platz geschaffen. Manchmal kann durch **Insert** auch ein sogenannter Einfügemodus eingeschaltet werden, d.h. alle nachfolgenden Zeichen werden an der Cursorposition eingefügt und überschreiben bereits vorhandene Eingaben nicht mehr.

Ebenfalls neben dem Cursortastenblock befindet sich **Clr/Home**. Home bedeutet seit Anbeginn der Computerzeit das Zurückkehren des Cursors in die linke obere Bildschirmcke. Tatsächlich führen die meisten Textverarbeitungsprogramme diese Funktion auch aus, wenn man diese Taste drückt. **Clr** (eigentlich **Shift**-**Home**) bewirkt fast das gleiche, allerdings wird dann zusätzlich der Bildschirm gelöscht. Diese Taste kann aber auch völlig anders belegt sein. Auf dem Desktop kann durch **Alternate**-**Home** die rechte Maustaste simuliert werden.

Bei den Pfeiltasten handelt es sich um die sogenannten Cursor-Tasten, mit denen in den meisten Programmen der Cursor in die entsprechende Richtung bewegt werden kann. Das Desktop benutzt die Cursortasten außerdem zur Simulation von Mausbewegungen (falls diese einmal fehlen sollte). **Alternate** plus Cursortaste bewegt den Mauspfel um acht Schritte in die zugehörige Richtung. Da dies für manche Fälle zu grob ist, kann man durch zusätzliches Festhalten der **Shift**-Taste die Bewegung auf einen Schritt pro Tastendruck verlangsamen.

Die meisten Spezialländerzeichen (z.B. die dänischen Umlaute, aber auch die griechischen und hebräischen Zeichen, siehe ASCII-Tabelle im Anhang) können über die Tastatur nicht erreicht werden. Einige Programme bieten aber die Möglichkeit, diese Zeichen per Mausklick aus einer Tabelle auszuwählen. Das angeklickte Zeichen wird dann in die Eingabe übernommen. In anderen Programmen kann nach Drücken einer speziellen Tastenkombination (z.B. **Ctrl**-**a**) der ASCII-Code eines gewünschten Zeichen als Zahl eingetippt werden (also z.B. **Ctrl**-**a**-**1**-**8**-**9** für das Copyright-Zeichen). Informieren Sie sich über diese Möglichkeiten bitte in der jeweiligen Gebrauchsanleitung.

Von den IBM-kompatiblen ist das längst bekannt: Mit **Ctrl**-**Alternate**-**Delete** kann ein Warmstart ausgelöst werden, d.h. der Rechner versucht neu zu booten. Dabei gehen natürlich alle Daten verloren, die nicht auf Diskette

oder Festplatte gespeichert sind. (Also Vorsicht - lösen Sie einen Warmstart nur dann aus, wenn der Rechner abgestürzt ist oder Sie neue Accessories laden wollen.). Leider gibt es diese nützliche Einrichtung nur in den neuen TOS-Versionen ab 1.04.

Im Normalfall kann der ST nur mit der Originaltastatur arbeiten. Inzwischen bieten aber einige Hersteller Adapterschaltungen an, die auch den Anschluß einer qualitativ hochwertigen PC-Tastatur erlauben.

Kommen wir nun zur Maus. Ihre Funktionsweise ist recht einfach. Wenn Sie die Rollkugel aus der Maus herausnehmen, sehen Sie zwei kleine Stahlwalzen, die offensichtlich von der Kugel bewegt werden können (ein drittes Rad dient als Andruckrolle). Diese Walzen dienen dazu, die Bewegungen der Maus in alle vier Richtungen aufzunehmen. Der Tastaturprozessor überwacht diese Walzen und meldet die Bewegungen an den Hauptrechner weiter.



Die eben beschriebenen Walzen sind auf einer Achse mit Lochscheiben montiert, die sich an jeweils zwei Lichtschranken vorbeibewegen. Jede Drehung der Walzen (und damit der Lochscheiben) erzeugt eine Reihe von Unterbrechungen des Lichtstrahls. Der Tastaturprozessor zählt die dadurch entstehenden Impulse und erhält dadurch die zurückgelegte Strecke. Außerdem muß noch die Richtung der Bewegung ermittelt werden. Dies wird mit einem Trick erreicht. Wie erwähnt existieren für jede Lochscheibe zwei Lichtschranken. Diese sind so angeordnet, daß sie niemals gleichzeitig unterbrochen werden können. Aus der Abfolge der Unterbrechungen kann der Prozessor aber die Richtung der Bewegung erkennen.

Pannenhilfe:



Bei schneller Bewegung wird der Mauszeiger nicht in der Richtung der Maus, sondern entgegengesetzt verschoben.



Sie schieben zu schnell. Der Tastaturprozessor ist zwar flink, aber nicht flink genug, um die Impulse der Maus bei so schneller Bewegung korrekt auswerten zu können. Schieben Sie die Maus langsamer, dann wird dieses Phänomen verschwinden.

Die Maus des ST ist eine speziell angepaßte Version mit einer wesentlich vereinfachten Elektronik, deren Aufgaben übernimmt ja der Prozessor in der Tastatur. Mäuse von PCs können im Normalfall nicht angeschlossen werden, da sie die Auswertung von Bewegungsrichtung und -strecke zum Teil selbst vor-

nehmen müssen und außerdem über aufwendigere Schnittstellen angeschlossen werden. Auch bei den Mäusen von Amiga und Macintosh könnten Probleme entstehen.

Falls Ihre Maus einmal ausgefallen ist, können Sie den Mauszeiger mittels einiger spezieller Tastenkombinationen bedienen. Es sind jeweils äquivalent:

Linke Maustaste	Alternate - Insert
Rechte Maustaste	Alternate - Home
Zeiger schnell links	Alternate - ←
Zeiger langsam nach links	Shift - Alternate - ←
Zeiger schnell nach rechts	Alternate - →
usw.	

Nachteil dieser Simulation: Oft ist es damit nicht möglich, eine Taste gedrückt zu halten, während man den Mauszeiger bewegen oder eine andere Taste drücken möchte.

▼ Beim Simulieren der linken Maustaste mittels Alternate - Insert geschieht es recht häufig, daß man versehentlich die Help-Taste mitdrückt und so eine Hardcopy auslöst.

Ein in der ST-Ära fast vergessenes "Eingabegerät" ist der Joystick. Prinzipiell kann man an den ST auch die einstmals so beliebten Steuerknüppel anschließen. Dabei werden alle jene Joysticks akzeptiert, die an die alten Atari- und Commodore-Geräte (Videospiele und 8-Bit-Computer wie C-64 oder Atari 600 XL) passen. Allerdings ist der Nutzen recht begrenzt. Als Mausersatz sind die Steuerknüppel nicht geeignet, benötigt werden sie aber für einige Spielprogramme. Wer einen STE besitzt, kann sogar bis zu sechs Joysticks anschließen. Dazu braucht man nur einen Adapter, damit die Joysticks auf die neuen Anschlüsse an der linken Seite passen (siehe Kapitel 7.).

Hin und wieder kann man auch Trackballs kaufen, die statt der Maus verwendet werden. Trackballs sind eigentlich nichts anderes als auf den Rücken gedrehte Mäuse. Das Gehäuse steht still auf dem Tisch, während der Anwender nur die Kugel in der Mitte bewegt. Achten Sie beim Kauf solcher Geräte darauf, daß diese speziell an den ST angepaßt sind. Atari-kompatibel alleine reicht nicht, denn es hat auch Trackballs für die alten Atari-Video-Spiele gegeben, die bestenfalls als Joystickersatz an Ihrem ST dienen können.

7. Schnittstellen

Im Gegensatz zum englischen Fachwort Interface kennzeichnet der deutsche Begriff in seiner wörtlichen Bedeutung den Sinn einer Schnittstelle schon ganz gut. Man kann sich durchaus vorstellen, daß ein Computer hier aufgeschnitten wird, um anderen Geräten Einblick bzw. Zugriffsmöglichkeiten zu geben. Im Klartext: Schnittstellen dienen der Kommunikation Ihres Computers mit anderen Geräten. Rein äußerlich können Sie eine Schnittstelle an einem Stecker auf der Rückseite erkennen.

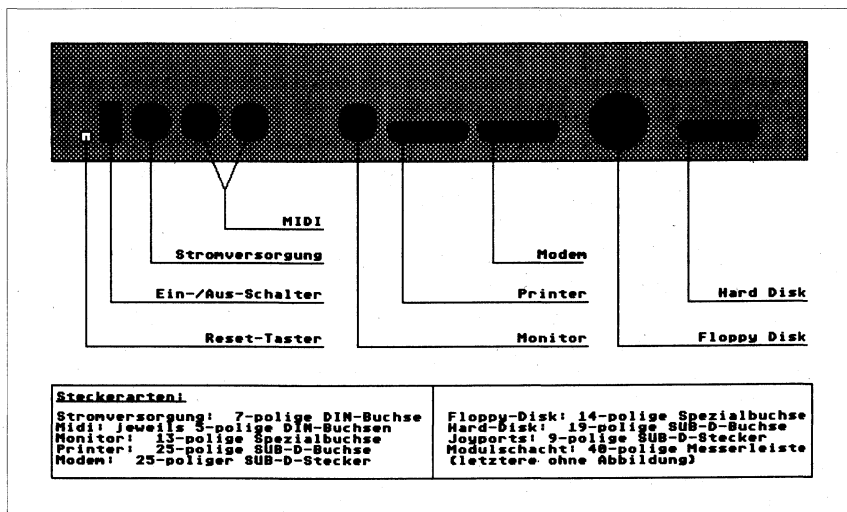


Abb. 7.1: Die Anschlüsse auf der Rückseite des 520 ST+ (bei anderen Modellen ist die Anordnung etwas geändert, die Steckerformen sind aber gleich)

Printer bezeichnet eine sogenannte Centronics-Schnittstelle, "Modem" steht für ein RS-232-Interface. Die Mega-STs besitzen im Gehäuse einen weiteren Anschluß, der die Benutzung von zusätzlicher Hardware (wie etwa Bildschirmadapter für Großbildschirme) erlaubt. An den Rechnern der STE-Reihe finden Sie zusätzlich an der linken Seite zwei Anschlüsse für Joysticks und andere Eingabegeräte (z.B. Paddles, Lichtgriffel usw.). Außerdem gibt es unter dem Netzschalter zwei Cinch-Buchsen, über die Stereo-Ton verfügbar ist. (Schließen Sie doch einmal Ihre Stereo-Anlage an.).

Ganz wichtig ist, daß Sie vor jedem Anschließen oder Abnehmen von Kabeln alle Geräte ausschalten. Die Anschlüsse sind im allgemeinen recht empfind-

lich. Nur in absoluten Ausnahmefällen (zur Fehlerbeseitigung bei Festplatten) dürfen Sie auch einmal bei eingeschaltetem Gerät mit den Steckern hantieren. Bevor Sie die Kabel berühren, sollten Sie eventuelle statische Aufladungen durch das Berühren einer Wasserleitung o.ä. abführen. Tun Sie das nicht, so riskieren Sie unter Umständen eine Beschädigung der Hardware.

Die Anschlüsse des ST sind vor allem aus Sparsamkeitsgründen recht empfindlich ausgefallen. Um die Kosten für die Herstellung niedrig zu halten, hat Atari darauf verzichtet, die meisten Schnittstellen zu puffern. Die Pins sind also im Normalfall direkt mit den zugehörigen Chips auf der Platine verbunden. Deren Ersatz kann - da es sich vielfach um Spezialteile aus Atari-eigener Produktion handelt - recht teuer werden.

Im Interesse einer sicheren mechanischen wie elektrischen Verbindung sollten Sie die Stecker festschrauben, wo immer sich die Gelegenheit dazu bietet. Bestes Negativbeispiel ist wohl der Monitorstecker, der - da ohne Schrauben - bei Bewegungen des Rechners recht leicht aus der Buchse rutschen und so einen Reset verursachen kann.

Sofern keine Kabel mitgeliefert werden, sollten Sie auf fertige Ware aus dem Handel zurückgreifen. Diese Kabel sind meistens mit einer Abschirmung versehen, die Störungen vorbeugen kann. Sollte das einmal nicht möglich sein, müssen Sie unter Umständen ein eigenes Kabel zusammenlöten. Die genauen Pinbelegungen dazu finden Sie in den entsprechenden Kapiteln zu jeder Schnittstelle bzw. im Anhang. Sie sollten aber im Interesse der Störsicherheit die Kabel nicht zu lang gestalten. Druckerkabel für die Centronics-Buchse sollten drei Meter nicht überschreiten, RS-232-Kabel dürfen etwas länger sein (unter Umständen bis zu 50 m).

Die Länge eines Kabels wird vor allem durch zwei Faktoren begrenzt: Den Signalverlust durch Widerstand und die Beeinflussung durch Störimpulse. Der Signalverlust tritt in jedem Kabel durch den elektrischen Widerstand auf und wird umso größer, je länger das Kabel ist. Wie stark dieser Verlust ist, hängt auch von den sonstigen Eigenschaften der Schnittstelle ab. So sind RS-232-Schnittstellen nicht so empfindlich wie z.B. das Centronics-Interface. Der zweite Faktor tritt vor allem bei Parallelschnittstellen auf. Durch die eng nebeneinander liegenden Datenleitungen im Kabel kann es hier zu Induktionen kommen, die die Signale empfindlich stören können. Wird ein solches Signal auch noch durch den Widerstand geschwächt, kann es mit der Übertragung ganz aus sein. Am häufigsten werden beim Atari die sogenannten DIN-Buchsen verwendet. Diese Stecker haben zwar meist sehr wenig mit der Norm zu tun, rein äußerlich haben sie jedoch das gleiche Aussehen wie die bekannten Stecker an manchen älteren Stereo-Anlagen. Das runde Gehäuse beherbergt

eine unterschiedliche Anzahl von Pins. Beispiele für solche DIN-Stecker sind die Anschlüsse für Floppy, Monitor, Midi und die Stromversorgung. Nachteil dieser sehr preisgünstigen Verbindungen: Sie besitzen keine Sicherungsschrauben und rutschen deshalb sehr leicht.

SUB-D-Verbinder finden Sie an den Anschlüssen für Hard-Disk, Drucker, Modem und Joysticks bzw. Maus. Diese sehr zuverlässige Verbindung ist im Normalfall auch mit Sicherungsschrauben ausgerüstet. Die Anzahl der Leitungen in solchen Steckern ist sehr verschieden und reicht von 5 bis 50. SUB-D-Verbinder mit Pins werden männlich, solche ohne Pins weiblich genannt. Durch die besondere Form (wie ein D - daher der Name) ist gewährleistet, daß diese Verbinder immer richtig eingesteckt werden. Messerleisten - wie die für die ROM-Module - sind ebenfalls sehr preisgünstig und werden häufig für interne Verbindungen gebraucht.

Recht häufig findet man an Druckern die sogenannten Amphenol-Stecker. Obwohl es sich so anhört, hat das nichts mit Chemie zu tun. Amphenolstecker haben ebenfalls eine D-Form, die Verbindung wird aber durch Kontakte ähnlich denen in einer Messerleiste hergestellt. Rein äußerlich können Sie diese Stecker leicht an den beiden Sicherungsbügeln erkennen, die statt Schrauben für eine sichere Verbindung sorgen.

Aber Vorsicht! Gleiche Steckerformen bedeuten nicht auch, daß die Geräte zusammenpassen. Es wäre unmöglich, für jedes Gerät eine eigene Steckerform zu schaffen. Daher gibt es oft gleiche Verbindungsstücke, obwohl die dazugehörigen Geräte elektrisch oder logisch überhaupt nicht zueinander passen.

Bestes Beispiel sind Maus und Joystick, die beide über den gleichen Anschluß versorgt werden, aber völlig unterschiedliche Daten an den Computer liefern. Eng mit dem Begriff der Schnittstelle ist die Baudrate verbunden. Damit wird die Übertragungsgeschwindigkeit bei einigen Schnittstellen angegeben.

Handshaking (wörtlich: Händeschütteln) bezeichnet die Fähigkeit der meisten Schnittstellen, die Übertragung zwischen beiden Geräten so zu steuern, daß jeder Beteiligte genug Zeit zur Entgegennahme bzw. Verarbeitung der Daten hat. Zu diesem Zweck werden die meisten Verbindungen mit Handshaking-Leitungen ausgestattet, auf denen die Geräte ihre Bereitschaft zur Übertragung signalisieren können. Der Sender muß dann immer so lange mit der Übertragung warten, bis der Empfänger sein "Fertig" schickt.

Handshaking ist die Methode, die die größtmöglichen Übertragungsgeschwindigkeiten erlaubt. Findet kein Handshaking statt, so muß der Sender so langsam arbeiten, daß der Empfänger genug Zeit hat, die Daten auch anzu-

nehmen und zu verarbeiten. Beim Handshaking werden dagegen diese Wartezeiten nur dann eingelegt, wenn der Empfänger diese wirklich benötigt.

Kommen wir zur Kardinalfrage: Was wird wo angeschlossen? Für Monitor, Diskettenlaufwerke und Festplatten kommen natürlich nur die Spezialschnittstellen in Frage, die dafür eingerichtet wurden. Beim Drucker ist das schon wieder anders. Diese können - je nach Ausstattung - entweder an die RS-232- oder die Centronics-Schnittstelle angeschlossen werden. Modems, Akustikkoppler und Netzwerke werden im Normalfall über den RS-232-Anschluß angekoppelt. Wenn Sie einen anderen Rechner mit dem ST verbinden wollen, kommt ebenfalls diese Schnittstelle in Frage.

Unter seriellen Schnittstellen versteht man solche Verbindungen, die die Bits eines Zeichens nacheinander über eine einzige Leitung schicken. Beispiele für solche serielle Schnittstellen sind die RS 232 und MIDI. Vorteil: Für die Übertragung werden weniger Leitungen gebraucht. Es ist damit sogar möglich, Daten per Telefon zu übertragen. Nachteil: Die Übertragung ist langsamer als bei paralleler Übertragung.

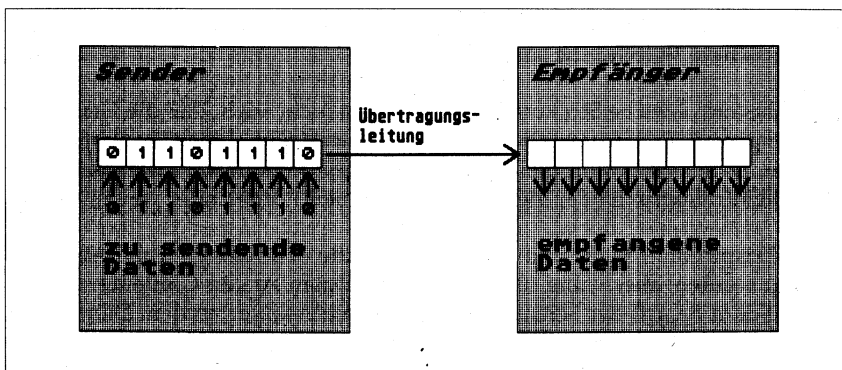


Abb. 7.2: Die Funktionsweise einer seriellen Schnittstelle

Im Gegensatz zur seriellen Schnittstelle stehen bei der parallelen acht oder mehr Datenleitungen gleichzeitig zur Verfügung, so daß die Übertragung sehr viel einfacher erfolgen kann. Ein Zeichen, das aus acht Bits besteht, wird also in einem einzigen Schritt übertragen. Bekanntester Vertreter dieser Arbeitsweise ist die Centronics-Schnittstelle, die sich hinter dem Printer-Anschluß verbirgt.

Aber auch die Verbindungen zur Floppy und zur Festplatte arbeiten nach diesem Prinzip. Der Vorteil ist die schnelle Übertragung. Nachteil: Aufwendiger

und etwas störanfälliger. Parallele Schnittstellen werden deshalb meistens für kurze, schnelle Verbindungen eingesetzt, während serielle Verbindungen oft für die Datenfernübertragung Verwendung finden.

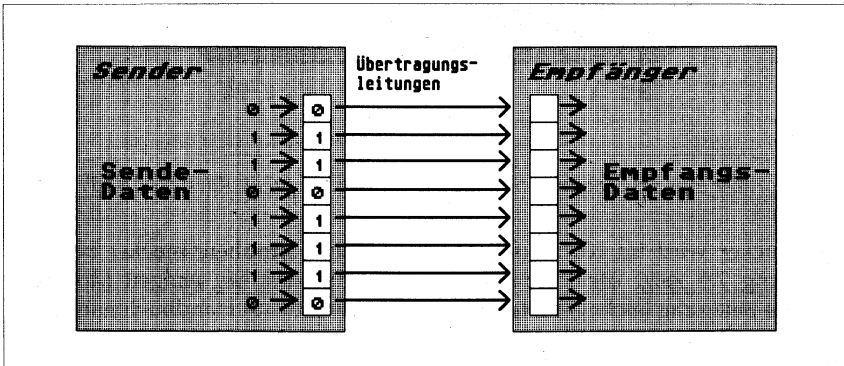


Abb. 7.3: Arbeitsweise einer parallelen Schnittstelle

7.1 Die Centronics-Schnittstelle

Die Centronics-Schnittstelle ist am ST mit Printer gekennzeichnet. Der Stecker selbst ist ein 25-poliger SUB-D-Verbinder.

Sie dient, wie der Name schon sagt, hauptsächlich dem Anschluß von Druckern. Dabei handelt es sich um eine 8-Bit-Parallel-Schnittstelle, die vom Druckerhersteller Centronics erstmals eingeführt wurde. Seitdem ist sie so oft verwandt worden, daß sie sich von ganz allein zum weltweiten Standard entwickelt hat.

Bitte entnehmen Sie der Abbildung 7.4 die Numerierung der einzelnen Pins. Die meisten der Leitungen werden zum Anschluß gar nicht benötigt. So sind beim ST die Leitungen 10 und 12 bis 17 gar nicht belegt. Im Folgenden finden Sie die Beschreibung der einzelnen Leitungen der Schnittstelle:

1 Strobe (low-aktiv), zeigt mit einem Low-Impuls die Gültigkeit der Daten auf den Datenleitungen an.

2-9 Data 1 - Data 8, 8 Datenleitungen

11 Busy, zeigt an, daß der Drucker noch keine weiteren Daten annehmen kann.

18-25 Ground, Erde

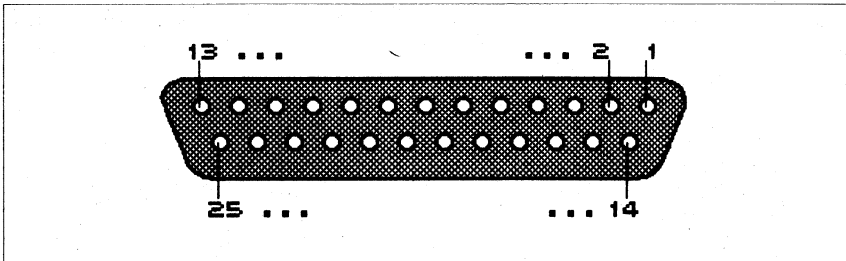


Abb. 7.4: Die Pins der Centronics-Schnittstelle (bei Aufsicht auf die Rückseite des ST)

Am Drucker wird im Normalfall der sogenannte Amphenolstecker benutzt. Abbildung 7.5 zeigt die Pin-Numerierung. (Hinweis: Bei einigen Modellen kann die Belegung leicht geändert sein; sehen Sie bitte im Handbuch nach.):

- 1-9 wie am ST
- 10 Acknlg (low-aktiv), zeigt den Empfang von Daten an
- 11 wie am ST
- 12 Paper empty, zeigt Papierende an
- 13 Selected, zeigt On-Line-Betrieb an
- 14 Autofeeds XT (low-aktiv), veranlaßt automatisches LF nach jedem CR
- 15 Nicht benutzt
- 16 Signal GND, Signalerdung
- 17 Chassis GND, Gehäuseerdung
- 18 Gleichspannung, +5V
- 19-30 GND, Masse-Leitungen
- 31 Reset (low-aktiv), dient zum Reset des Druckes
- 32 Error (low-aktiv), zeigt Fehler im Drucker an
- 33 Ext GND, externe Erdung
- 34-35 Nicht benutzt
- 36 Select in (low-aktiv), zum Anwählen des Geräts

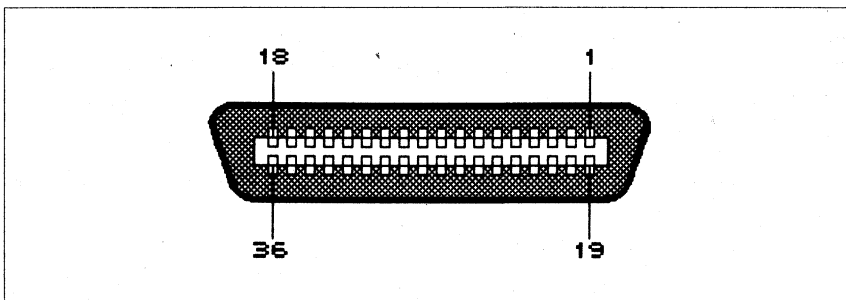


Abb. 7.5: Der Amphenolstecker (Aufsicht)

Um ein Drucker-kabel selbst zu bauen, gibt es zwei Möglichkeiten. Die einfachere ist wohl die Verwendung von sogenannten Quetschverbindern. Diese sind Stecker, in die ein Flachbandkabel eingelegt wird, das dann von einer eingequetschten Platte in der richtigen Position gehalten wird. Besorgen Sie sich bitte je einen Amphenol- und einen SUB-D-Quetschverbinder und ein 25-poliges Flachbandkabel in der gewünschten Länge. Das Flachbandkabel ist an einer Seite durch einen andersfarbigen Streifen markiert. Das ist die Leitung 1.

Legen Sie das Kabel bitte so in die beiden Stecker ein, daß Leitung 1 jeweils auf Pin 1 zu liegen kommt. Einzige Ausnahme: Leitung 18 sollte an irgendeiner Stelle unterbrochen werden. Im Amphenolstecker bleiben dabei natürlich einige Pins frei. Jetzt können Sie die beiden Stecker zusammenquetschen. Der sicherste Weg: Spannen Sie die Stecker in einen Schraubstock mit glatten Backen ein, und drehen Sie so lange zu, bis die Platten einrasten.

Vorsicht: Wenn die Platten einmal eingerastet sind, bekommt man sie kaum noch los. Die andere Methode zum Quetschen stammt zwar aus der Praxis, aber sollte nur von geübten Bastlern durchgeführt werden. Dabei werden die Stecker unter einen Bürostuhl gelegt. Dann setzen Sie sich bitte mit etwas Schwung an Ihren Arbeitsplatz...



Bei Verwendung eines Quetschverbinderkabels arbeitet der Drucker nicht korrekt oder piepst nach dem Einschalten.



Die mit den Quetschverbindern hergestellten Kabel sollten mit den meisten Druckern funktionieren. In seltenen Fällen können sich jedoch Probleme ergeben, dann müssen Sie das Kabel löten. Das Piepsen ist ein Hinweis des Druckers, daß das Kabel nicht in Ordnung ist. Kontrollieren Sie bitte, ob die Litzen wirklich auf den richtigen Pins liegen. (Ein Durchgangsprüfer oder ein einfaches Meßgerät können dabei sehr gute Dienste leisten.).

Wenn Sie schon ein Quetschverbinderkabel gebaut haben, das nicht funktioniert, und nun löten wollen, brauchen Sie nicht unbedingt zwei neue Stecker zu kaufen. Es reicht, wenn Sie nur den Amphenolstecker in der Lötversion kaufen, die Quetschverbindung am ST-Stecker kann weiterbestehen bleiben.

Die Lötmethode hat den Vorteil, daß Sie auch Rundkabel mit entsprechend vielen Litzen benutzen können. Die Leitungen sollten dann so verbunden werden:

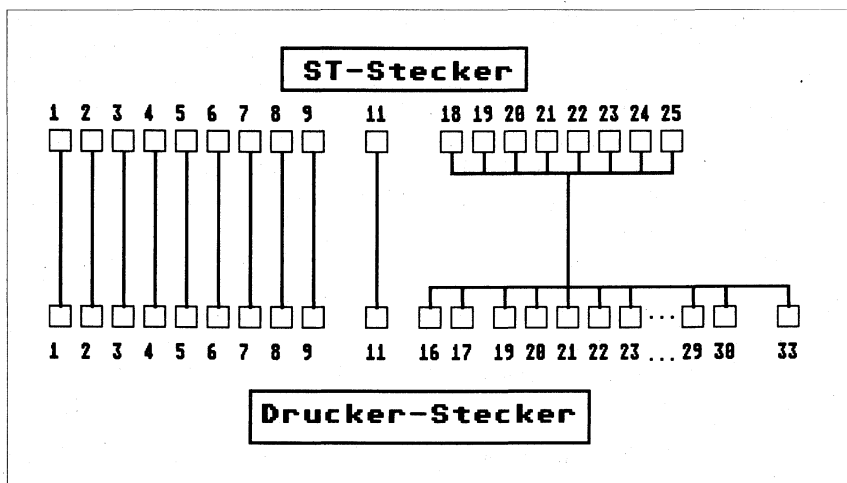


Abb. 7.6: Die Verbindungen für ein Druckerkabel



Ein an die Centronics-Schnittstelle angeschlossener Drucker reagiert nicht.



Sind die Kabel korrekt eingesteckt? Ist das Kabel korrekt gelötet? Ist der Drucker am richtigen Interface angemeldet (Accessory Drucker-Anpassung). Kontrollieren Sie diese Punkte bitte. Achten Sie außerdem darauf, daß der Drucker immer auf ON-LINE geschaltet ist.

7.2 Die RS-232-Schnittstelle

RS 232C (bzw. Recommended Standard Number 232 Revision C, so der genaue Name) ist die Bezeichnung einer Norm für serielle Schnittstellen. Die Nummer 232 ist dabei von entscheidender Wichtigkeit, weil es auch andere RS-Normen gibt. In Deutschland wird diese Norm auch oft mit V24 bezeichnet. RS 232 steht für eine Norm, die weltweit die am stärksten verbreitete ist, was serielle Schnittstellen angeht.

An der Rückseite Ihres ST finden Sie den dazugehörigen Stecker mit der Bezeichnung Modem (mit dem Telefonhörer). Diese Bezeichnung hat historische Gründe, denn die RS-232-Norm war ursprünglich für die Datenfernverarbeitung gedacht. Aufgrund der hohen Verbreitung wird die RS-232C auch oft einfach als serielle Schnittstelle bezeichnet.

Beim Begriff Modem handelt es sich, wie so oft in der Welt der Computer, um eine Abkürzung, denn dieser Begriff steht für **MOD**ulator-**DEM**odulator. Man braucht ein solches Gerät, um Daten per Telefon zu übertragen (daher der Hörer). Es reicht nicht, einfach ein Bit (einen Impuls) über die Telefonleitung zu schicken, denn dort würde es hoffnungslos untergehen. Vielmehr ist es so, daß Einsen und Nullen durch verschieden hohe Töne repräsentiert werden. Bei abgehenden Daten müssen also aus Bits Töne gemacht werden. Diesen Vorgang nennt man auch Modulieren. Bei ankommenden Daten geht das Verfahren natürlich genau umgekehrt; aus Tönen müssen Bits gemacht werden, was die Fachwelt mit Demodulieren bezeichnet.

Was ist der Unterschied zwischen Modems und Akustikkopplern, die mindestens ebenso bekannt sind? Modems sind direkt an der Telefonleitung angeschlossen und arbeiten daher mit Strömen. Im Unterschied dazu sind die sogenannten Akustikkoppler zu sehen, die aus Tönen Bits erzeugen und umgekehrt. Sie sind nicht an das Telefonnetz angeschlossen, sondern beziehen Ihre Daten aus dem Lautsprecher des Telefonhörers bzw. "sprechen" sie in das Mikrofon. Hier handelt es sich also um eine rein akustische Kopplung. Beide Gerätearten leisten im Grunde genommen ähnliche Arbeiten, nur sind die Modems aufgrund der elektrischen Kopplung an das Telefonnetz zu komfortableren Funktionen in der Lage (wenn die Post sie lassen würde). Wichtig ist im Moment nur, daß beide an der RS-232-Schnittstelle angeschlossen werden.

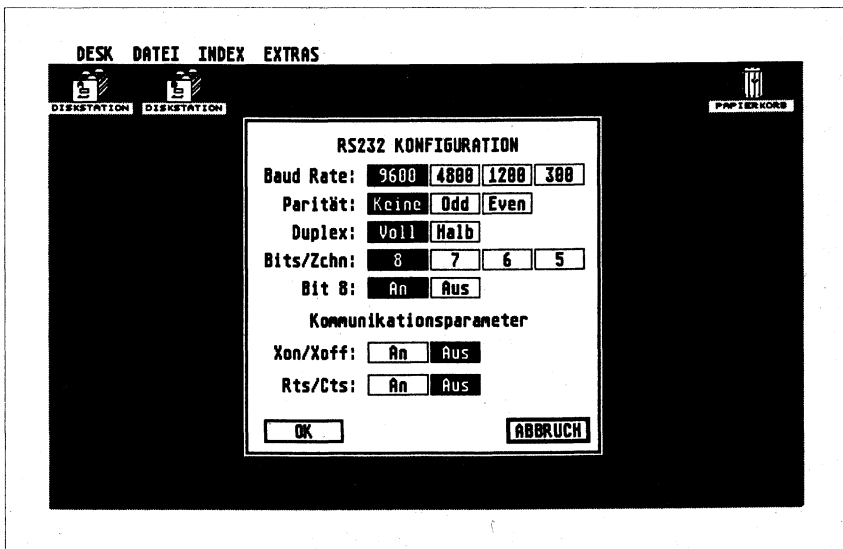


Abb. 7.7: Das Accessory zur Anpassung der RS-232-Schnittstelle

Bevor man die RS-232-Schnittstelle benutzen kann, muß man im Accessory "RS 232 Einstellung" einige Parameter einstellen. Die serielle Schnittstelle kann nämlich mit verschiedenen Geschwindigkeiten und mit mehreren Handshaking-Verfahren arbeiten. Die richtige Auswahl dieser Werte ist wichtig, weil sonst nur verstümmelte oder gar keine Übertragungen stattfinden können.

Für alle Einstellungen gilt: Die Übertragung kann nur klappen, wenn beide angeschlossenen Geräte auf die gleichen Werte eingestellt sind. Sehen Sie also bitte nach, wie das angeschlossene Gerät arbeitet.

Baudrate

Hier wird die Übertragungsgeschwindigkeit eingestellt. Da die Bits, aus denen sich ein Zeichen zusammensetzt, nacheinander über die Leitung geschickt werden, ist es nötig, den zeitlichen Abstand dieser Impulse festzulegen. Die gebräuchlichsten Übertragungsraten sind 1200 und 2400 Baud. Für die Kopplung von Rechnern und die Bedienung von Druckern empfiehlt sich zunächst eine Geschwindigkeit von 1200 Baud. Klappt die Übertragung, kann man auch auf höhere Geschwindigkeiten umsteigen, die allerdings störanfälliger sind.

Parität

In manchen Fällen wird dem übertragenen Zeichen noch ein sogenanntes Paritätsbit hinterhergeschickt, quasi die Quersumme der Bits. "Odd" heißt ungerade, "Even" steht demnach für gerade Parität. Gerade Parität bedeutet, daß bei einer ungeraden Zahl von 1-Bits ein zusätzliches 1-Bit gesendet wird, um die Anzahl gerade zu machen (und umgekehrt). Die Einstellung hier ist je nach Anwendung verschieden.

Duplex

Die RS-232-Schnittstelle kann in zwei Richtungen übertragen. Beim Duplex-Betrieb muß der Empfänger jedes Zeichen, daß empfangen wird, an den Sender zurückschicken. Auf diese Weise werden Übertragungsfehler erkannt. Im Halb-Duplex-Betrieb entfällt dieses Zurücksenden, dafür zeigt der ST bei Verwendung des VT-52-Emulators die gesendeten Zeichen auf dem Bildschirm an.

Bits/Zchn

In der Datenfernübertragung hat ein Zeichen nicht immer genau acht Bits (wie das im Computer üblich ist). Wenn weniger Bits übertragen werden, stehen auch weniger Zeichen zur Verfügung, dafür ist die Übertragung schneller. Bei Druckern und Computern sollten Sie immer acht Bits pro Zeichen wählen, um

den vollen Zeichensatz ausnutzen zu können. Bei der Datenfernübertragung müssen Sie sich informieren, welche Bitzahl Ihr Kommunikationspartner verlangt.

Die Verwendung von weniger Bits hat wieder einmal historische Gründe. So hat der ursprüngliche ASCII-Zeichensatz nur sieben Bit breite Zeichen enthalten. Beim guten alten Telex wurden nur sechs Bits benutzt. Übrigens stehen Ihnen bei der Verwendung von fünf Bits nur noch 32 verschiedene Zeichen zur Verfügung.

Bit 8

Einige ältere Drucker verwerten von den acht gesendeten Bits nur sieben, das höchstwertige (Nr. 8) wird einfach vergessen. Für solche Drucker muß das Bit 8 ausgeschaltet werden. Hierzu können leider auch keine Empfehlungen gegeben werden - "Probieren geht über Studieren".

Xon/Xoff

Diese Kürzel bezeichnen ein Software-Handshaking. Die beiden Kommunikationspartner fordern sich durch das Senden spezieller Zeichen gegenseitig auf, mit der Übertragung fortzufahren bzw. diese anzuhalten.

Rts/Cts

Rts steht für Request to Send, Cts für Clear to send. Mit dieser Auswahl können Sie das sogenannte Hardware-Handshaking einschalten, d.h. die beiden Geräte signalisieren sich über spezielle Leitungen die Empfangs bzw. Sendebereitschaft.

Die ST-Modelle 260, 520 und 1040 sind aufgrund eines Fehlers im ROM nicht in der Lage, das Handshaking korrekt durchzuführen. Eine Übertragung mit Handshaking (ob hard- oder softwaremäßig) führt fast immer zum Abstürzen des Rechners. Abhilfe: Benutzen Sie spezielle Kommunikationssoftware, die die Fehler ausbügeln kann. Falls Sie einen Drucker über die RS-232 anschließen möchten, können Sie diesen nur mit Anwendungssoftware benutzen, die spezielle Treiber für die serielle Schnittstelle enthält. Die Mega-STs sind von diesen Fehlern nicht betroffen, ebenso alle Geräte mit Blitter-TOS. Wenn eine Übertragung trotz der Fehler ohne Software stattfinden soll, müssen beide Parameter auf "Aus" geschaltet sein. Näheres siehe unter "Rechnerkopplung".

Wenn der Rechner einfach die Daten-Bits über die serielle Schnittstelle auf die Reise schicken würde, könnte niemals eine korrekte Übertragung zustande

kommen. Nehmen wir an, das erste Bit wäre eine Null - im Kabel geschieht nichts, denn Null bedeutet Strom aus. Wie soll der Empfänger feststellen, daß dort ein Null-Bit über die Leitung huscht? Deshalb muß vor jedem Datenpaket ein Start-Bit (eine 1!) geschickt werden, um dem Empfänger den Beginn der Übertragung zu signalisieren. Am Ende eines Zeichens folgen dann noch ein oder zwei Stopp-Bits, die den Abschluß markieren.

Preisfrage: Wie stelle ich die Stopp-Bits an meinem ST ein? Mit dem Accessory geht das überhaupt nicht, der Empfänger muß sich zwangsläufig nach dem ST richten. Manche Terminalprogramme erlauben jedoch das Setzen dieses Parameters. Im Normalfall arbeitet der ST mit zwei Stopp-Bits. Auch hier hilft eigentlich nur Probieren, falls die Übertragung wider Erwarten doch einmal nicht klappen sollte. Der VT-52-Emulator stellt eigentlich eine noble Zugabe von Atari dar. VT-52 ist ein sehr weit verbreitetes Terminal für Großrechner. Der ST kann dieses Terminal emulieren (d.h. nachahmen), so daß Sie Ihren Rechner theoretisch direkt an einen Großcomputer anschließen könnten. Der VT-52-Emulator dient dazu, Texteingaben an Großrechner zu schicken und Ausgaben von diesen auf dem Bildschirm anzuzeigen. Leider wird dieser Vorteil durch die Fehler in der RS 232 wieder zunichte gemacht. Derzeit wird der VT-52-Emulator durch den neueren VT-100-Emulator ersetzt.

Wie sieht nun die Steckerbelegung bei der seriellen Schnittstelle aus? Zunächst ist zu bemerken, daß der ST nur über eine Spar-Ausgabe der RS 232 verfügt. Einige Pins mit Sonderfunktionen sind hier nicht belegt. Das braucht Sie aber nicht zu beunruhigen; die wenigsten Geräte verfügen über eine voll ausgestattete serielle Schnittstelle. Der Stecker am ST ist ein 25-poliger SUB-D-Verbinder (männlich). Beachten Sie bitte, daß die Pins hier genau umgekehrt wie beim Centronics-Verbinder numeriert sind. Die Pins 6, 9 - 19, 21 und 23 - 25 sind nicht belegt.

1	GND	Gehäuseabschirmung
2	TD	Sendedaten
3	RD	Empfangsdaten
4	RTS	Request-to-Send, Hardware-Handshaking
5	CTS	Clear-to-Send, Hardware-Handshaking
7	SG	Signal-Masse
8	DCD	Data Carrier Detect, Modems zeigen hier an, daß sie eine Verbindung hergestellt haben.
20	DTR	Data Terminal Ready, hier signalisiert der Rechner, daß er bereit zum Senden ist.
22	RI	Ring Indicator, Modems melden hier das "Klingeln" des Telefons (den Wunsch eines Partners nach Aufnahme einer Verbindung).

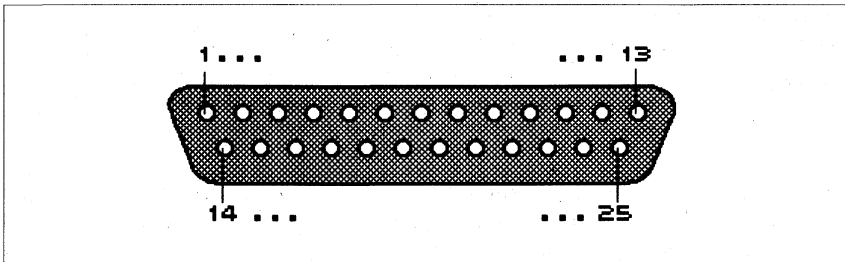


Abb. 7.8: Die Pinbelegung am RS-23-Stecker (Aufsicht von außen)

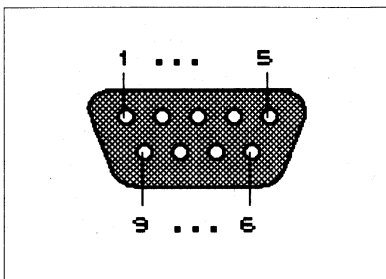


Abb. 7.9: 9-poliger RS-232-Stecker für PCs

Empfangsgeräte wie Modems o.ä. sind meistens mit einem weiblichen SUB-D-Verbinder ausgerüstet (ebenfalls 25-polig). Die Pinbelegung ist bei diesen Verbindern identisch, die Pins werden jedoch wie in Abb. 7.9 numeriert. Unter den PC-Kompatiblen hat sich die Unsitte verbreitet, die RS-232 über 9-polige SUB-D-Verbinder zu führen (wie bei einem Joystick). Dabei ist die Pinbelegung wie folgt:

1	DCD
2	RD
3	TD
4	DTR
5	Signal Ground
6	DSR (Data Set Ready, fehlt beim ST, zeigt Betriebsbereitschaft des Empfängers an)
7	RTS
8	CTS
9	RI

Bei der RS-232 unterscheidet man den Anschluß von sogenannten Datenend-einrichtungen (Computer, Drucker) und Datenübertragungseinrichtungen (Modems, Akustikkoppler). Letztere sind besonders einfach zu behandeln. Zum Anschluß werden die einzelnen Signale 1:1 miteinander verbunden, d.h. TD an TD, RD an RD usw.

Welche Pins dabei jeweils miteinander verbunden werden müssen hängt von den vorhandenen Steckerformen ab. Liegen auf beiden Seiten 25-polige Verbinder vor, so können Sie im allgemeinen Quetschverbinder benutzen.

Ein Nullmodem soll einem Computer ein nicht vorhandenes Modem an der seriellen Schnittstelle vortäuschen. Man braucht es bei der Übertragung von Daten zwischen zwei Rechnern oder zwischen Rechner und Drucker.

Wie bereits erwähnt, wurde die RS232-Schnittstelle ursprünglich nur für den Anschluß von Modems und sonstigen Datenfernübertragungseinrichtungen konzipiert. Mit der Zeit entdeckte man diese Norm aber auch als ideale Verbindungsmöglichkeit zwischen Rechnern. In diesem Falle fehlen natürlich die Modems, die sonst die Daten auf die Reise über die Telefonleitung schicken. Eine serielle Schnittstelle erwartet ankommende Daten auf der RD-Leitung und sendet auf TD.

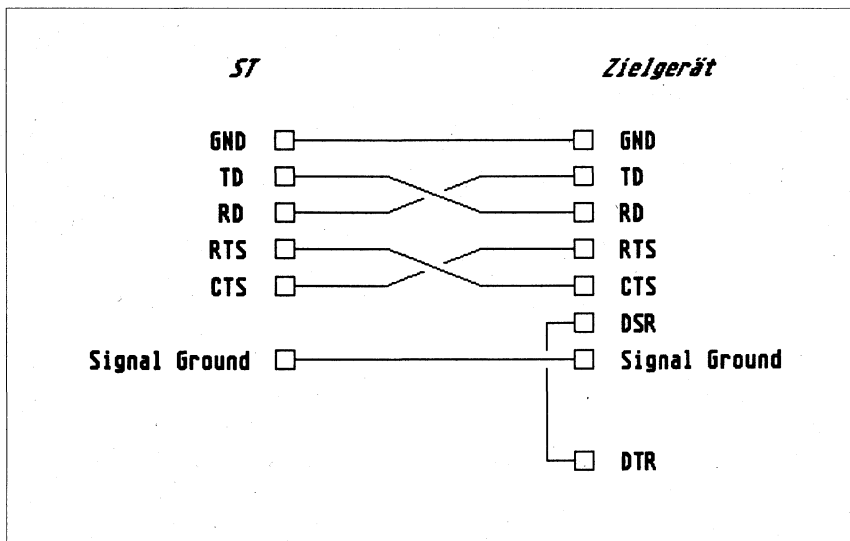


Abb. 7.10: Die Verdrahtung eines Null-Modems

Werden nun zwei Rechner mit einem 1:1-Kabel verbunden, so senden beide Geräte auf der gleichen Leitung. In einem Nullmodem werden die Daten- und Handshaking-Signale über Kreuz geschaltet, so daß jede Leitung am richtigen Pin landet. Technisch gesehen ist dahinter nur ein simples Kabel zu sehen, in dem einige Adern sich überkreuzen.

Zur Rechnerkopplung brauchen Sie also das erwähnte Null-Modem, dessen Schaltung Sie in Abb. 7.10 ablesen können. Wegen der gekreuzten Leitungen lassen sich hier keine Quetschverbinder benutzen. Vielfach können Sie solche Null-Modems auch schon kaufen.

In den meisten Fällen können Sie das oben gezeigte Null-Modem auch für den Anschluß eines Druckers verwenden. Manchmal funktioniert das jedoch nicht, dann müssen Sie etwas anders vorgehen. Abb. 7.11 zeigt die etwas geänderte Beschaltung. Bei der Verwendung dieses Druckerkabels sollten Sie jedoch beachten, daß ein Hardware-Handshaking nicht möglich ist. In diesen Fällen müssen Sie unbedingt im Accessory "RS 232 Einstellung" die Option RTS/CTS aus- und die Option Xon/Xoff einschalten.



Die meisten Drucker mit serieller Schnittstelle arbeiten am ST wegen der erwähnten Fehler im ROM nur dann korrekt, wenn Anwenderprogramme einen eigenen Treiber eingebaut haben.

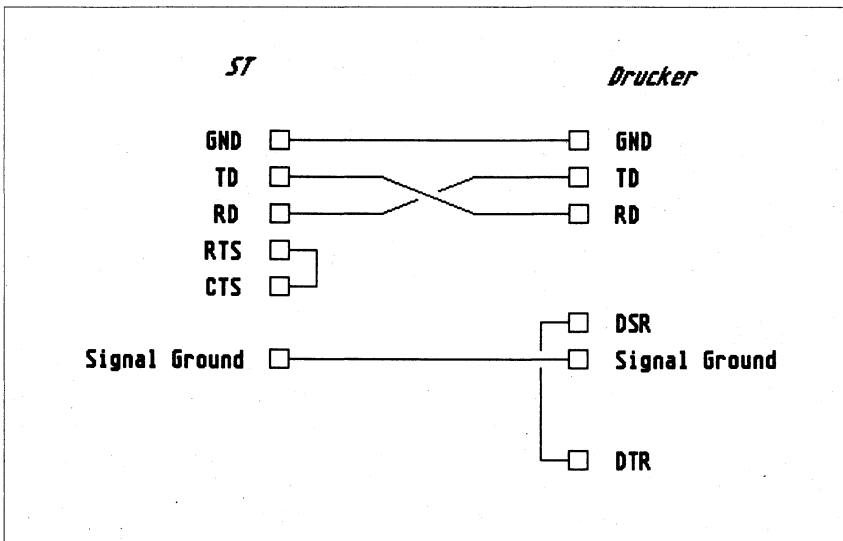


Abb. 7.11: Ein serielles Druckerkabel



Bei der Benutzung der seriellen Schnittstelle zeigt das angeschlossene Gerät ein untypisches Verhalten (z.B. lassen sich Drucker nicht auf ON-LINE schalten) oder reagiert gar nicht. Daten werden offensichtlich nicht übernommen.



Wahrscheinlich ist das Kabel nicht korrekt. Benutzen Sie vielleicht ein 1:1-Kabel für die Ansteuerung eines Druckers oder Computers? Ist das Kabel korrekt gelötet? Sind die Schnittstellenparameter korrekt eingestellt? Überprüfen Sie bitte alle diese Möglichkeiten. Schließlich kann auch noch der seltene Fall vorliegen, daß das angeschlossene Gerät mit keinem der oben vorgestellten Kabel zusammenarbeitet. In diesem Fall sollten Sie einen Fachmann konsultieren.



Am Zielgerät kommen nur wirre Zeichen an.



In diesem Fall sind wahrscheinlich Parität und/oder Baudrate der Schnittstelle nicht richtig eingestellt. Kontrollieren Sie diese Werte bitte, und sorgen Sie dafür, daß beide Geräte auf gleiche Werte eingestellt sind. Sollten Sie dazu die DIP-Schalter an einem Drucker ändern müssen, schalten Sie diesen bitte vor der nächsten Übertragung kurz aus und dann wieder ein, um die Übernahme der neuen Einstellung zu gewährleisten. Falls beide Geräte mit den gleichen Parametern arbeiteten und der Fehler trotzdem auftritt, könnte die Baudrate zu hoch gewählt sein. Wählen Sie einen geringeren Wert.



Am Zielgerät kommt nur ein Zeichen an, dann stoppt die Übertragung.



Hier handelt es sich um den Fehler in den Betriebssystem-ROMs, der eine korrekte Übertragung verhindert. Um mit dem Zielgerät richtig arbeiten zu können, brauchen Sie eine geeignete Software mit eigenen Treibern.

7.3 MIDI

MIDI ist die Abkürzung für Musical Instrument Digital Interface. Engagierte Hobby-Musiker unter Ihnen werden diesen Standard vielleicht kennen. Hier haben sich nämlich die Hersteller von digital arbeitenden Musikinstrumenten (z.B. Synthesizer oder Drum-Machines) geeinigt und eine einheitliche Anschlußnorm geschaffen. Im Klartext: Mittels MIDI-Schnittstellen ist man in der Lage, mehrere solcher Musikinstrumente miteinander zu koppeln und gemeinsam zu steuern.

Damit ist es also denkbar, daß Ihr ST als Steuerrechner für eine ganze Batterie von Instrumenten dient und z.B. komplette Melodien an ein Keyboard verschickt. Wenn Sie also das nächste Mal in ein Rockkonzert gehen und auf der Bühne einen ST stehen sehen, wissen Sie: MIDI ist am Werk.

Die MIDI-Schnittstelle zeigt sich in Form von zwei runden Buchsen, die bei 520 ST+, 520 ST/M und 260/ST direkt neben dem Anschluß für das Netzteil, beim 1040ST auf der linken Seite des Gerätes neben dem Modulport und beim Mega ST hinten zwischen Printer- und Monitoranschluß zu sehen sind und die Bezeichnungen "MIDI In" und "MIDI Out" tragen. Beim 1040 ST und dem STE befinden sich die Anschlüsse neben dem Modul-Port.

MIDI ist eine serielle Schnittstelle und basiert auf dem Prinzip der Stromschleife, d.h. alle angeschlossenen Geräte sind ringförmig miteinander verbunden. Möchte ein Gerät mit einem anderen kommunizieren, schickt es seine Mitteilung über den Ausgang MIDI Out los. Diese Nachricht kommt beim nächsten Gerät über MIDI In herein. Der Empfänger sieht sich dann die Meldung genau an und stellt fest, ob sie für ihn bestimmt ist. Falls ja, kann er entsprechend reagieren. In jedem Fall aber wird die Meldung über Midi Through an den nächsten in der Kette weitergegeben, auch wenn die Nachfolger gar nicht davon betroffen sind. Auf diese Weise läuft die Nachricht über alle angeschlossenen Geräte zurück zum Sender. Dieses umständliche Verfahren hat durchaus seinen Sinn. Falls die Meldung nicht unbeschädigt wieder beim Sender ankommt, muß irgendwo ein Übertragungsfehler vorliegen, und der Sendevorgang kann wiederholt werden. Die Übertragungsgeschwindigkeit bei MIDI beträgt 31250 Baud, das Datenformat ist ein Start-Bit, acht Daten-Bits und ein Stopp-Bit.

Zunächst einmal brauchen Sie für die Nutzung der MIDI-Schnittstelle Software, die dafür eingerichtet ist. Es gibt bereits spezielle Programme, mit deren Hilfe der ST Melodien speichern und auf angeschlossenen Keyboards abspielen kann (sog. Sequenzer). Meist kann dann auch noch eine Drum-Machine mit angeschlossen werden und und und...

MIDI ist recht schnell und braucht, da es sich hierbei um eine serielle Schnittstelle handelt, nur wenige Leitungen, die zudem noch sehr lang sein dürfen. Aus diesem Grund ist man schon mehrfach auf die Idee gekommen, diese Schnittstelle nicht nur zum Steuern von Musikinstrumenten zu benutzen. So existieren Netzwerke, die auf MIDI basieren und viele Computer miteinander verbinden.

Die Bedienung der MIDI-Schnittstelle hängt einzig und allein vom verwendeten Programm ab. Daher können hier leider keine weitergehenden Tips gegeben werden.

7.4 Die Anschlüsse für Hard- und Floppy-Disk

"Hard-Disk" steht für den sogenannten DMA-Anschluß. Ursprünglich nur für den Anschluß einer Festplatte gedacht, sind hier wichtige Weiterentwicklungen gemacht worden. Bekanntestes Beispiel ist der Atari-Laserdrucker, der hier ebenfalls mit dem ST verbunden wird. Das funktioniert, weil sich hier eigentlich eine abgespeckte SCSI-Schnittstelle verbirgt, die sehr weitgehende Fähigkeiten besitzt.

An die "Floppy-Disk"-Buchse kommt, wie Sie wissen, ein Diskettenlaufwerk. Falls in Ihrem Rechner kein Laufwerk eingebaut ist, stecken Sie hier das Kabel zum Laufwerk A ein, an das dann seinerseits Laufwerk B angeschlossen wird. Bei den Modellen 1040 STF, 520 STF und den Mega-STs, die alle eingebaute Laufwerke haben, wird hier eine zusätzliche Station angeschlossen. Wichtig ist: Mehr als zwei Disketten-Stationen kann der ST nicht bedienen, auch wenn der Anschluß einer weiteren Station möglich scheint, weil ja schließlich noch eine Buchse frei ist.

Die Festplatte hat neben der großen Kapazität noch die angenehme Eigenschaft, daß sie ungeheuer schnell arbeitet. Man sagt sogar, daß die Harddisk im Prinzip genauso schnell wie eine RAM-Disk arbeitet, lediglich durch die ab und zu notwendigen Spurwechsel der Schreib-/Lese-Köpfe wird sie etwas langsamer. Wie dem auch sei, das Zauberkürzel in diesem Zusammenhang heißt DMA. Das steht für Direct Memory Access (direkter Speicherzugriff) und kennzeichnet eine recht ausgefuchste Methode, Daten zwischen Speicher und Festplatte hin und her zu transportieren.

Im Normalfall läuft die Übertragung von Daten zwischen Speicher und Peripherie ungefähr so ab: Der Prozessor holt die Daten aus dem Speicher, gibt sie an die Schnittstelle weiter und diese an das betreffende Gerät (bzw. umgekehrt). Bei DMA-Betrieb kann das Peripheriegerät direkt in den Speicher schreiben oder daraus lesen. Der Prozessor braucht nur noch den Ort der Speicherung anzugeben und hat mit dem Rest nichts mehr zu tun. Auf diese Weise wird bei jedem übertragenen Zeichen der Umweg über den Prozessor vermieden.

DMA wird übrigens nicht nur in Verbindung mit Festplattenlaufwerken angewandt. Auch die Floppy-Disk füllt den Speicher per DMA. Hier merkt man aber von der großen Geschwindigkeit nicht so viel, weil die Diskettenstation an sich sehr viel langsamer ist. Außerdem bedingt diese Technik, daß Festplatte und Disketten nie gleichzeitig bedient werden können, da immer nur ein Gerät über den DMA-Kanal zugreifen darf.

7.5 Der Modulschacht

Was hier ein bißchen an ein Telespiel erinnert, hat einen durchaus ernsten Hintergrund. So können hier ROM-Module eingesteckt werden, die seriöse wie amüsante Software enthalten. Bestes Beispiel sind die BASIC-Interpreter von GFA und Omikron, die beide auf solchen ROMs erhältlich waren bzw. noch erhältlich sind. Außerdem findet man in der einschlägigen Fachliteratur immer wieder Bauanleitungen, die den schlichten Modulanschluß in eine universelle Schnittstelle verwandeln.

Neben Anwendungssoftware auf Modulen gibt es auch reine Datenmodule, die keine Programme enthalten. Sie werden von verschiedenen Programmen genutzt, die darauf wie auf normale Speicher zugreifen können. Schließlich gibt es noch sogenannte Diagnosemodule, über die jeder Atari-Händler verfügt. Diese enthalten spezielle Prüfprogramme, mit denen eine Fehlerdiagnose sehr vereinfacht werden kann.

Sie können ein ROM-Modul in das GEM einbinden, indem Sie es als Laufwerk mit der Kennung c anmelden. Jetzt können Sie die Programme nutzen, aber nicht verändern oder neu anlegen.

Achten Sie beim Einstecken eines Moduls darauf, daß alle Geräte ausgeschaltet sind!. Sie müssen streng darauf achten, daß Sie das Modul richtig herum in den Rechner einstecken. Sollten sich nach dem Einschalten Anomalien ergeben, schalten Sie den Rechner aus, und prüfen Sie, ob es korrekt eingesteckt wurde.

7.6 Die Joystickanschlüsse am STE

Der STE hat gegenüber den älteren ST-Versionen zwei zusätzliche Anschlüsse für Spielgeräte. Pro Buchse können zwei Joysticks betrieben werden oder zwei Paddles (Drehregler). An einen der beiden Anschlüsse paßt außerdem ein Lichtgriffel. Diese Geräte benötigen einen 15-poligen SUB-D-Stecker oder einen Adapter.

Glücklicherweise sind die Nummern der Pins aufgeprägt (Nr. 1 ist rechts oben, Nr. 15 links unten). Der erste Joystick trägt die Nummer 1 usw. "Oben 0" ist also das Signal, das ausgelöst wird, wenn Sie den Steuerknüppel 0 nach oben schieben. Hier ist die dazugehörige Pinbelegung:

1	Oben 1	6	Feuerknopf 1	11	Oben 2
2	Unten 1	7	Versorgungsspannung	12	Unten 2
3	Links 1	8	Reserviert	13	Links 2
4	Rechts 1	9	Masse	14	Rechts 2
5	Paddle Y-Richtung	10	Feuerknopf 2	15	Paddle X-Richtung

Um Ihre alten Joysticks an die neuen Buchsen anzuschließen, brauchen Sie einen Adapter. Diesen können Sie sich selbst bauen. Zunächst brauchen Sie einen 15-poligen und zwei 9-polige SUB-D-Stecker, außerdem zwei gleich lange Kabel mit mindestens sieben Adern. An die 9-poligen Stecker werden später die beiden Joysticks angeschlossen.

Joystick 2	15-poliger Stecker	Joystick 1	Pinbelegung
	1 -----	1	1 Oben
	2 -----	2	2 Unten
	3 -----	3	3 Links
	4 -----	4	4 Rechts
	5 -----		5 reserviert
	6 -----	6	6 Feuerknopf
7 -----	7 -----	7	7 Versorgungsspannung
	8 -----		8 Masse
8 -----	9 -----	8	9 reserviert
5 -----	10 -----		
1 -----	11 -----		
2 -----	12 -----		
3 -----	13 -----		
4 -----	14 -----		
	15 -----		

Diese Verbindungen müssen gemacht werden.

8. Datenaustausch mit anderen Rechnern?

Für den Austausch von Daten mit anderen Rechnertypen gibt es mehrere Möglichkeiten. Zunächst bietet sich wohl an, den Datenaustausch mittels Disketten zu probieren. Solange der Tauschpartner ebenfalls über ein 3,5-Zoll-Diskettenlaufwerk verfügt, gibt es zumindest die theoretische Möglichkeit, die eigenen Disketten auch dort einzusetzen. Die andere Möglichkeit zum Datenaustausch ist die Kopplung zweier Rechner über die serielle Schnittstelle.

Aber auch, wenn der Datenaustausch klappt, ist das Ergebnis nicht immer das gewünschte. Auf jeden Fall können Sie reine Text-Files austauschen. Wichtig ist, daß diese Dateien keine Steuerzeichen mehr enthalten, denn diese würden auf dem anderen Rechner falsch interpretiert. Benutzen Sie deshalb in Textverarbeitungen die Möglichkeit, den Text als reines ASCII-File abspeichern zu können. Dabei gehen natürlich die Formatierungen verloren, aber das ist leider nicht zu vermeiden.

Bei Dateien aus einer Dateiverwaltung oder aus Grafik-Programmen wird die Übertragung schon schwieriger. In den meisten Fällen arbeiten die Programme auf anderen Rechnern mit verschiedenen Datenformaten, so daß sich bei der Übernahme Schwierigkeiten ergeben können. Hier hilft nur Probieren weiter. In manchen Fällen ist es auch möglich, kleine Konvertierungsprogramme zu schreiben. Allerdings ist dazu eine intime Kenntnis der Datenstrukturen erforderlich.

So gut wie überhaupt nicht möglich ist der Austausch von Programmen. Zur Zeit existieren auf dem Markt vier große Rechnerfamilien, deren Programme untereinander nicht austauschbar sind. Neben den ST-Modellen gibt es die weitverbreitete Gruppe der PC-Kompatiblen, die Amigas und die Modellfamilie rund um den Apple Macintosh. Jede dieser Familien hat eigene Programmformate, die PC-Kompatiblen haben auch noch einen anderen Prozessor, so daß hier die Übertragung selbst bei den PCs aus dem Hause Atari scheitert. Amiga und Mac besitzen zwar denselben Prozessor (den 68000), verarbeiten aber ganz unterschiedliche Betriebssysteme, so daß es auch hier unmöglich ist, die Programme des einen auf dem anderen Rechner laufen zu lassen. Wichtige Ausnahmen: Für den ST gibt es sogenannte Emulatoren, die einen PC bzw. einen Macintosh auf dem ST nachahmen können. Wenn Sie einen solchen Emulator besitzen, können Sie natürlich die fremde Software übernehmen.

Wie geht der Datenaustausch mit einem Amiga vor sich? Das einfachste wird hier die Übertragung per serieller Schnittstelle sein. Dazu müssen Sie allerdings einige Tricks beachten - siehe unten. Ein anderer Weg steht durch das

Programm DOS-2-DOS offen, das es dem Amiga ermöglicht, diverse andere Diskettenformate (darunter auch das Atari-Format) zu lesen und zu beschreiben. Auf diese Weise können Sie Dateien höchst elegant per Diskette auf den Amiga übertragen. Sind die Daten dann einmal auf dem anderen Rechner angekommen, müssen diese noch ein wenig nachbearbeitet werden. Der Amiga benutzt nämlich einen leicht abgeänderten Zeichensatz, so daß insbesondere Spezialzeichen und Umlaute ein sehr merkwürdiges Aussehen zeigen werden. Abhilfe: Laden Sie die Datei in eine Textverarbeitung, und führen Sie einen Ersetzungsvorgang durch, bei dem das fremde Zeichen durch das gewünschte ersetzt wird.

Zur Übertragung auf einen Mac steht leider nur die serielle Schnittstelle zur Verfügung. Zur genauen Vorgehensweise sehen Sie bitte weiter unten nach. Außerdem stimmen die Zeichensätze nicht überein, so daß Sie einige Tauschvorgänge ausführen müssen.

Mit den PC's haben wir den einfachsten Fall. Falls der PC mit einem 3,5-Zoll-Laufwerk ausgestattet ist, steht einer Übertragung per Diskette nichts mehr im Wege. Der Atari ST kann das Diskettenformat der PC's lesen und beschreiben. Einzige Voraussetzung: Die zur Übertragung vorgesehene Diskette muß auf einem PC formatiert sein. Auf diese können Sie dann alle Dateien kopieren, die der jeweils andere Rechner erhalten soll. Falls der PC kein 3,5-Zoll-Laufwerk besitzt, müssen Sie auf die serielle Schnittstelle ausweichen. Zum Glück ist auch der Zeichensatz des PC nicht allzusehr vom ST verschieden. In den meisten Fällen wird es reichen, einfach das "ß" anzupassen. Beim ST wird dafür der ASCII-Code 158 verwendet, beim PC ist es die 225.

Pannenhilfe:



Bei der Übertragung per Diskette weigert sich einer der Rechner, die Diskette des jeweils anderen zu lesen, obwohl dies eigentlich möglich sein müßte.



Es kann sein, daß eines der beiden Laufwerke dejustiert ist, d.h. die Schreib-/Leseköpfe sind ein wenig aus der richtigen Bahn gerutscht. Lassen Sie Ihr Laufwerk beim Händler überprüfen und ggf. neu einstellen. Das ist nicht teuer und trägt auch zur allgemeinen Datensicherheit bei.

Wie geht nun der Datenaustausch per RS-232-Schnittstelle vor sich? Zunächst müssen Sie die beiden Rechner mit einem Nullmodem wie in Kapitel 7 beschrieben miteinander koppeln. Vorsicht: Bei einigen Geräten können die Stecker zwar genau wie beim ST aussehen, doch die Pinbelegung ist evtl. ganz anders. Informieren Sie sich bitte in der zugehörigen Gebrauchsanleitung.

Steht die Verbindung (und sind auf beiden Rechnern gleiche Schnittstellenparameter eingestellt), so kann die Übertragung beginnen. Dazu brauchen Sie auf dem ST ein geeignetes Terminalprogramm, also eine Software, die der Datenübermittlung dient. Achten Sie bitte darauf - sofern Sie eine ältere TOS-Version besitzen - daß das Terminalprogramm eigene Routinen zur Bedienung der Schnittstelle besitzt, sonst pfuscht Ihnen der Fehler im Betriebssystem ins Handwerk.

Falls Sie mit einer der neueren TOS-Versionen arbeiten, können Sie auch das folgende BASIC-Programm einsetzen. Es besteht aus nur sechs Zeilen und überträgt eine beliebige Datei Zeichen für Zeichen per serieller Schnittstelle an einen anderen Rechner. Am schnellsten ist die Übertragung, wenn beide Rechner das RTS/CTS-Verfahren beherrschen. Das Listing des Terminalprogramms sieht so aus:

Version für GFA-BASIC

```
Open "I",#1,"Dateiname"
While Not Eof(#1)
  Byte=Inp(#1)
  Out 1,Byte
Wend
Close #1
```

Version für Omikron- bzw. ST-BASIC

```
1 OPEN "I",1,"Dateiname"
2 OPEN "V",2
3 WHILE NOT EOF(1)
4   Byte$=INPUT$(1,1)
5   PRINT#1, Byte$
6 WEND
7 CLOSE 1
8 CLOSE 2
```

Für "Dateiname" müssen Sie natürlich den Namen der Datei einsetzen, die Sie übertragen möchten. Falls es mit diesen Programmen Schwierigkeiten geben sollte, müssen Sie auf ein richtiges Terminalprogramm umsteigen.

Auf dem Zielgerät muß ebenfalls ein Terminalprogramm eingesetzt werden, das aber diesmal auf Empfang geschaltet wird.

9. Software

Software (wörtlich: "Weiche Ware") ist das Gegenteil von Hardware. Das Wort Hardware steht im Englischen auch für "Werkzeug". Die Geräte stellen also ein Werkzeug dar, die Programme, die darauf laufen, sind die Software.

Unter Anwendungsprogrammen versteht man solche Software, die speziell für eine bestimmte Anwendung geschrieben ist. Beispiele: Textverarbeitungen, Dateiverwaltungen etc. Das Gegenteil von Anwendungen sind Programmiersprachen und Tools.

Tool heißt ebenfalls "Werkzeug". Solche Programme dienen zum Beispiel zur bequemeren Verwaltung der Dateien auf einer Diskette. Das Programm "TREE" aus Kapitel 3 ist ein typisches Tool. Andere Tools können Dateien besonders schnell kopieren oder ermöglichen einen direkten Zugriff auf jedes einzelne Byte einer Diskette (sogenannte Disk-Monitoren).

Im Normalfall ist jedes Kopieren von fremder Software ein Verstoß gegen das Urheberrechtsgesetz. Gekaufte Programme sind fremdes geistiges Eigentum und dürfen daher nur für den eigenen Gebrauch vervielfältigt werden - z.B. wenn Sie ein Programm von der Originaldiskette auf eine Arbeitskopie oder die Festplatte übertragen. Auch die berühmte Weitergabe an den besten Freund (von dem es plötzlich sehr viele gibt), ist verboten. Nur wenn die Hersteller des Programms ausdrücklich das Kopieren freigeben, ist es auch erlaubt. In manchen Fällen wird sogar das Herstellen von Sicherungskopien durch eine Kopierschutz unmöglich gemacht. Eine Sicherungskopie wäre hier aber rechtlich erlaubt - jedoch nur für den eigenen Gebrauch. Falls Sie das Original-Programm weitergeben, müssen Sie die Kopie vernichten.

Etwas anders ist das bei Public-Domain-Software, Freeware und Shareware. Hier handelt es sich um Programme, die Sie beliebig oft kopieren und verbreiten dürfen. Public-Domain heißt soviel wie "Öffentliches Gut", d.h. der Programmierer hat auf seine Urheberrechte an dieser Software verzichtet und stellt sie jedermann kostenlos zur Verfügung. Das gleiche gilt für die sogenannte Freeware. Shareware ist etwas ähnliches. Der Programmierer hat hier nicht gänzlich auf seine Rechte verzichtet. Sie dürfen Shareware beliebig kopieren und verteilen und sich das Programm nach Herzenslust anschauen. Wenn Sie die Software aber wirklich gebrauchen können, sollten Sie fairerweise den geforderten Betrag (meistens 10,- bis 20,- DM) an den Programmierer schicken. Damit haben Sie das Programm endgültig erworben. In den meisten Fällen erhalten Sie dann als kleines Dankeschön eine detaillierte Anleitung. Oft kann man sich auch als Anwender registrieren lassen und erhält so jede neue Version automatisch zugeschickt.

Der billigste Weg, um an solche freie Software heranzukommen, ist sicher das Kopieren von Freunden (dies ist ja ausdrücklich erlaubt). Public-Domain-Software wird aber auch von sogenannten Kopierdiensten vertrieben, die für die Kopie einer Public-Domain-Diskette recht saftige Gebühren verlangen, mitunter bis zu 20 DM. Wohlgemerkt: Diese Kopierdienste sind keineswegs die Urheber dieser Programme, sie kopieren sie nur. Ich empfehle Ihnen daher, sich mit möglichst vielen ST-Besitzern zusammenzutun, um diese Gebühren für den einzelnen klein zu halten. Wenn Sie die Disketten einmal haben, kann Sie niemand daran hindern, die Software an Bekannte weiterzugeben. Ähnliches gilt auch für die Kopierdienste einiger Computerzeitschriften, die aber meist sehr viel zivilere Gebühren als die kommerziellen Vertreiber fordern.

Hochklassige Software können Sie hier sicherlich nicht erwarten. Doch ist es oft erstaunlich, welchen Standard die freien Programme bereits erreicht haben. Leider hilft hier nur eine Methode: Probieren. Das kann bei kommerziellen Kopierdiensten natürlich recht teuer werden - aber auch mit "richtiger" Software kann man Pech haben.

9.1 Textverarbeitung und Editor

Ein Editor ist ein Programm, mit dem man Texte schreiben und redigieren kann - mehr nicht. Eine Textverarbeitung erlaubt darüberhinaus noch vielfältige andere Funktionen. Das beginnt bei der Formatierung des Textes (automatischer Randausgleich, Zentrieren), geht weiter bei Trennhilfen und Lexika und hört auf bei der Verwaltung von Fußnoten, der automatischen Erstellung von Inhaltsverzeichnissen und dem Einbinden von Grafiken. Editoren sind daher meist sehr viel einfacher und schneller. Sie werden vor allem zum Schreiben von Programmtexten eingesetzt. In den meisten Fällen wird zu jeder Programmiersprache ein Editor mitgeliefert, der oft auch noch an die speziellen Gegebenheiten der Sprache angepaßt ist.

Natürlich muß eine Textverarbeitung zuerst einmal Texte bearbeiten können. Diese sollten dann in beliebiger Form ausdrückbar sein - und zwar auf möglichst vielen Druckern, ohne daß Sie eine umständliche Anpassung vornehmen müssen. In Punkto Formatierung muß wenigstens der sogenannte Block- und der Flattersatz/Blocksatz; möglich sein. Unter Umständen kann auch ein mehrspaltiger Ausdruck sehr nützlich werden, wenn Sie zum Beispiel für eine Zeitung schreiben. Vielfach wird auch eine Trennhilfe oder ein Lexikon geboten, was in den deutschen Sprache allerdings auf gewisse Schwierigkeiten stößt. Gute Programme erlauben die Erstellung von Serienbriefen (mit jeweils gleichem Text, aber verschiedenen Adressen, die aus einer Dateiverwaltung direkt übernommen werden). Wenn Sie häufig Abbildungen zur Illustration Ihrer

Texte benötigen, sollte Ihre Textverarbeitung unbedingt Grafikdateien aus einem Malprogramm einbinden können. So entfällt das lästige Kleben. Für alle, die häufig "professionelle" Texte erstellen müssen (Wissenschaftler, Studenten, Autoren), ist die automatische Erstellung von Inhalts- und Stichwortverzeichnis wichtig. Hierbei müssen die entsprechenden Zeilen bzw. Wörter im Text nur markiert werden, und das Programm setzt beim Druck daraus automatisch die gewünschten Verzeichnisse zusammen. Kaufleute werden es begrüßen, wenn die Textverarbeitung auch direkt Berechnungen im Text ausführen kann - z.B. die Summe aller Werte in einer Spalte.

Es versteht sich wohl von selbst, daß die Textverarbeitung die Benutzung verschiedener Schriftarten wie fett, kursiv usw. unterstützt.

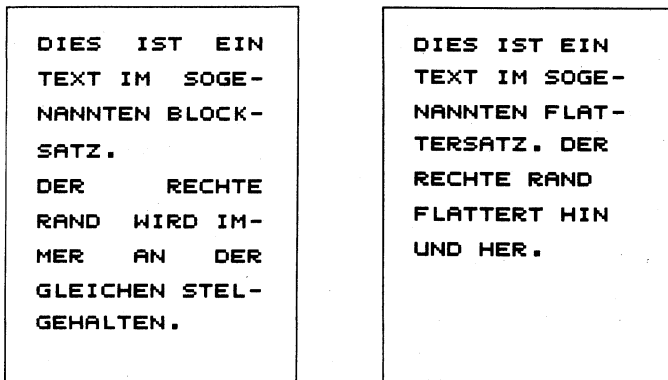


Abb. 9.1: Block- und Flattersatz

Eine Trennhilfe trennt automatisch solche Wörter, die über ein Zeilenende hinausreichen. Je nach Aufwand, den die Entwickler getrieben haben, ist diese Trennhilfe mehr oder weniger gut. Nur eines gibt es nicht: 100%-ig korrekt arbeitende Trennprogramme. Die deutsche Sprache wehrt sich bisher mit Händen und Füßen dagegen, Trennungen mechanisch durchführen zu lassen. Das liegt vor allem an den zusammengesetzten Hauptwörtern, aber auch an einigen Ausnahmen. Wenn Sie also eine Trennhilfe benutzen, müssen Sie die so erzielten Trennungen auf jeden Fall noch kontrollieren.



Die meisten Trennhilfen lassen sich durch das Einfügen eigener Bindestriche von der richtigen Schreibweise überzeugen. Wenn Ihre

Textverarbeitung also wieder einmal falsch getrennt hat, setzen Sie an der richtigen Stelle einen Bindestrich, und lassen Sie die Zeile neu formatieren. In den meisten Fällen übernimmt das Programm Ihren Vorschlag, soweit er in die Zeile paßt.

Die nächste Stufe der Rechtschreibhilfe stellen die Lexika dar. In einer meist sehr großen Datei sind viele Wörter und Silben gespeichert. Diese werden mit dem eingetippten Text verglichen. Findet das Programm eine Silbe oder ein Wort, das nicht im Lexikon gespeichert ist, so geht es von einem Rechtschreibfehler aus und moniert dieses Wort. Sie haben dann die Möglichkeit, dieses Wort zu ändern oder - falls es sich um ein neues Wort handelt - dieses in das Lexikon zu übernehmen. Auf diese Weise wächst das Lexikon mit der Zeit auf den Wortschatz an, den Sie in Ihrer täglichen Schriftsprache verwenden. Je nach Programm kann man den Text durch ein solches Lexikon nachträglich oder während der Eingabe kontrollieren lassen. Nützlicher Nebeneffekt der Lexika: Die Trennhilfe wird genauer, da die neuen Wörter im Lexikon meist silbenweise abgelegt werden.



Wenn Sie die ersten Male mit einem Lexikon arbeiten, enthält es in der Regel nur wenige Wörter. Lassen Sie in dieser Situation die Kontrolle während der Eingabe durchführen, so werden Sie nach wenigen Wörtern immer wieder unterbrochen, was sich natürlich äußerst hinderlich auf den Schreibfluß auswirkt. Lassen Sie deshalb das Lexikon zunächst nur nachträglich den Text bearbeiten, bis nur noch wenige der richtig geschriebenen Wörter moniert werden. Dann können Sie ohne große Störungen auch die mitlaufende Kontrolle benutzen.

Ein wichtiges Hilfsmittel beim Editieren sind Blöcke. Blöcke in einer Textverarbeitung oder einem Editor sind markierte Textteile, die meistens beliebig lang sein können (Buchstaben, Wörter, Zeilen, Absätze). Mit diesen Blöcken lassen sich dann eine Reihe von Operationen ausführen. So können die meisten Textverarbeitungsprogramme Blöcke verschieben, kopieren und löschen. Die Mehrheit der Programme zeigt einen Block dadurch an, daß der Text auf dem Bildschirm invertiert oder farbig unterlegt wird.

Blöcke eignen sich vorzüglich dazu, vorgefertigte Textbausteine zu kompletten Briefen zusammenzusetzen. Wenn Sie also öfter einmal wiederkehrende Formulierungen benötigen (z.B. in Rechnungen oder Geschäftsbriefen), tippen Sie diese einmal ein und speichern diese dann auf Diskette oder Festplatte - möglichst unter einem aussagekräftigen Namen. Wenn Sie dann beim Schreiben eines Briefes diesen Baustein wieder benötigen, laden Sie einfach den Textblock von der Diskette, statt ihn mühsam neu eintippen zu müssen. Andere Anwendungsgebiete von Blöcken: Innerhalb eines Textes (z.B. in Program-

men) wird ein ähnliches oder sogar identisches Stück mehrfach gebraucht. Markieren Sie es als Block. Diesen Block können Sie dann auf Tastendruck dahin kopieren (also vervielfältigen), wo er wieder gebraucht wird. Dann können Sie noch evtl. nötige Änderungen durchführen.

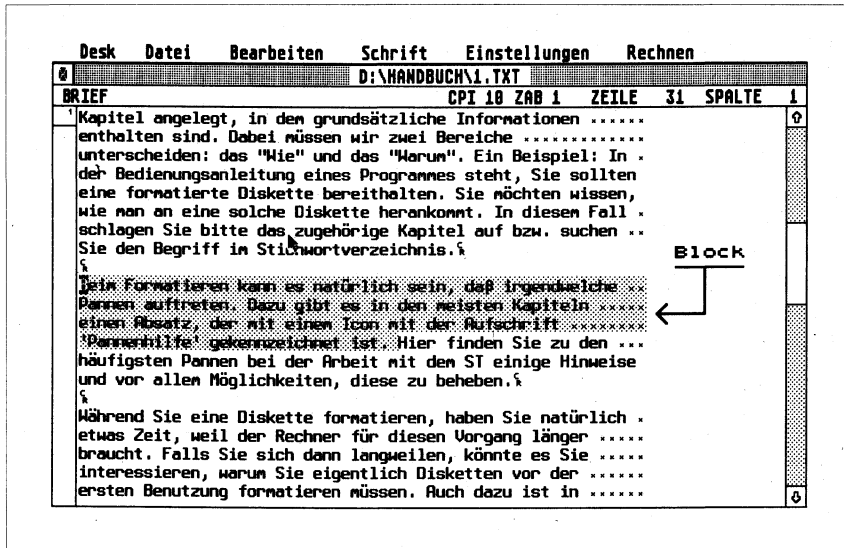


Abb. 9.2: Ein markierter Block

Auch sonst lassen sich Blöcke recht kreativ einsetzen. Stellen Sie sich vor, Sie haben einen Text erstellt, der in letzter Minute komplett neu gegliedert werden muß. Das heißt nicht etwas neutippen. Markieren Sie nacheinander alle umzustellenden Textteile als Block, und verschieben Sie sie dann an den gewünschten Ort. Der Block wird an der alten Stelle entfernt und am neuen Ort fein säuberlich eingefügt - fertig. Natürlich können Sie überflüssige Teile auch entfernen: Block markieren und löschen.



In vielen Fällen müssen Änderungen an der Formatierung über den ganzen Text vorgenommen werden, die Textverarbeitung läßt dies aber nur absatzweise zu. Normalerweise müßte man dann jeden Absatz einzeln anfahren und formatieren. Mitunter hilft aber auch folgender Trick: Markieren Sie den gesamten Text als Block. Stellen Sie nun die neuen Formate ein. Die meisten Programme beziehen diese Änderung nicht mehr auf den Absatz, in dem der Cursor steht, sondern auf den markierten Block - also in unserem Fall den ganzen Text.

Dieser Trick funktioniert natürlich auch mit anderen Funktionen als der Formatierung. Nehmen wir an, Sie haben eine Schriftart so an den Drucker angepaßt, daß Sie damit den NLQ-Modus einschalten können. Um nun einen ganzen Text in NLQ drucken zu lassen, markieren Sie wieder alles als einen Block und schalten diesen Block auf die gewünschte Schriftart um.

Mindestens ebenso nützlich wie die Blockfunktionen ist die Ersetzen-Funktion. Zunächst einmal wirkt sie trivial: Jedes Vorkommen eines Wortes wird durch ein anderes ersetzt. Doch auf den zweiten Blick entdeckt man eine Reihe von Anwendungen mehr. Wenn Sie beispielsweise einen Text mit einigen sehr häufigen Fachworten schreiben (wie wäre es mit "Allphasen-Netto-Umsatzsteuer mit Vorsteuerabzug"?), dann ist es natürlich sehr zeitsparend, wenn man statt des Wortungetüms eine Abkürzung eintippen kann. Im obigen Beispiel: Überall, wo das Fachwort auftreten soll, tippen Sie ANU ein. Ist der Text dann fertig, lassen Sie ANU einfach durch "Allphasen-Netto-Umsatzsteuer mit Vorsteuerabzug" ersetzen. (Wen es interessiert: Das ist der Fachausdruck für das derzeit in der Bundesrepublik gültige Mehrwertsteuersystem.).

Auch für das Entfernen von immer wiederkehrenden Wörtern aus einem Text eignet sich die Ersetzen-Funktion sehr gut. Beispiel: Sie möchten überall im Text den Satz "Na also." löschen. Die brutale Methode wäre, jedes Vorkommen dieses Satzes zu suchen und dann zu löschen. Mit Ersetzen geht das einfacher. Lassen Sie den Satz "Na also." durch einen leeren Text ersetzen. Auf diese Weise wird der Satz im ganzen Text gelöscht.

Die nächste wichtige Anpassung stellt die Zeichensatzkonvertierung dar. Wenn Sie einen Text von einem anderen Rechner übernommen haben, stimmen einige Zeichen (z.B. ä,ß) nicht mehr. Statt dessen erscheinen irgendwelche Grafik- oder sonstige Sonderzeichen. Abhilfe: Lassen Sie diese Sonderzeichen durch die gewünschten Zeichen ersetzen. Wird also statt des ß ein # angezeigt, lassen Sie # durch ß ersetzen.



Falls das ersetzte Zeichen auch in anderen Fällen im Text vorkommt, erscheint dort natürlich auch das Ersatzzeichen. Wurde also das Doppelkreuz nicht nur für das ß eingesetzt, sondern auch z.B. zum Unterstreichen, so erscheint statt dieser Unterstreichungen eine Reihe von ß. Die meisten Editoren und Textverarbeitungen erlauben es aber, vor jedem Ersetzungsvorgang eine Sicherheitsabfrage einzusetzen. Auf diese Weise können Sie dann jeweils überprüfen, ob die Ersetzung korrekt ist.

Pannenhilfe:



Das Zeichen ß oder Worte, die das ß enthalten, werden nicht gefunden, obwohl es nachweislich im Text vorkommt.



Wenig beachtet, aber wichtig: Der ST kennt zwei ß-Zeichen. Das eine mit dem ASCII-Code 158 existiert nur auf dem ST und wird normalerweise als ß eingesetzt. Es gibt aber noch das Beta-Zeichen mit dem ASCII-Wert 225, das genauso aussieht und zum Beispiel auf PC's zur Darstellung des scharfen "S" eingesetzt wird. In solchen Fällen sollten Sie das Beta-Zeichen durch das ß ersetzen lassen und können dann wie gewohnt fortfahren. Allerdings brauchen Sie dazu eine Textverarbeitung, die es erlaubt, auch solche Zeichen einzugeben, die sonst nicht per Tastatur erreichbar sind (siehe Kapitel "Tastatur und Maus").

Im allgemeinen benutzt jedes Textverarbeitungs-Programm eine andere Methode, um Formatierungen und Schriftarten auf der Diskette zu speichern. Wenn Sie daher eine Datei aus einem anderen Textverarbeitungssystem mit Ihrem Programm laden, erhalten Sie mit ziemlicher Sicherheit ein totales Chaos auf dem Bildschirm. Doch dagegen ist ein Kraut gewachsen. Um Texte zwischen verschiedenen Programmen auszutauschen, sollten Sie diese im ASCII-Modus abspeichern, d.h. der Text wird ganz ohne Formatierungen und Steuerzeichen auf der Diskette abgelegt. In der Datei stehen dann nur noch die reinen Texteingaben, die Sie gemacht haben. In manchen Programmen müssen Sie dazu erst den sogenannten Word-Processor-Modus ausschalten (Word-Processor ist der engl. Ausdruck für Textverarbeitung). Die Anwendung dieser Methode heißt aber auch, daß alle Einstellungen wie Schriftarten oder spezielle Formatierungen verloren gehen. Übrigens speichern alle Editoren (nicht Textverarbeitungen) im ASCII-Format ab.

Unter Fließtext wird häufig eine Textdatei verstanden, die nur den reinen Text enthält (wie im obigen Abschnitt besprochen). Dabei darf dann nur am Ende eines Absatzes ein CR-Zeichen stehen. Einige Programme (z.B. Beckertext 1.0) speichern, wenn man die ASCII-Option verwendet, den Text mit einem CR-Zeichen nach jeder Zeile ab. In diesem Fall können Sie leider keinen Fließtext erzeugen. Fließtext ist eigentlich die universellste Methode, Texte auf andere Systeme zu übertragen. Insbesondere, wenn Sie als Journalist tätig sind, sollten Sie darauf achten, daß Ihre Textverarbeitung diese Fähigkeit besitzt. Sie können dann den Text auf Diskette in die Redaktion geben.

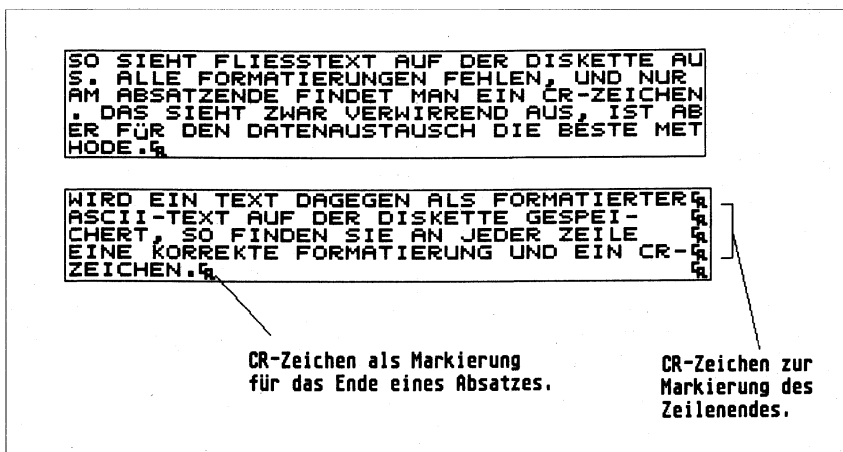


Abb. 9.3: Fließtext und formatierter ASCII-Text

Viele Programmierer möchten auch Listings von Programmen in der Textverarbeitung bearbeiten. Ob das funktioniert, kommt ganz darauf an. Die Quelltexte von Compiler-Sprachen können Sie auf jeden Fall laden und auch manipulieren. Wenn Sie die so bearbeiteten Dateien dann jedoch wieder compilieren wollen, müssen Sie diese unbedingt im ASCII-Modus speichern. Dabei ist es unerheblich, ob das Programm Fließtext speichert oder nicht. Bei BASIC-Programmen sieht das nicht ganz so einfach aus. Diese werden im allgemeinen codiert gespeichert, so daß zum Beispiel statt eines PRINT-Befehls nur eine Nummer auf der Diskette steht. (Dadurch werden die Programme schneller und brauchen weniger Platz.). Die bisher bekannten BASIC-Interpreter bieten aber die Möglichkeit, die Listings auch als ASCII-File auf der Diskette zu speichern (z.B. per SAVE "name",A). In diesem Fall werden die Befehlsworte ausgeschrieben, das ganze Programm steht also im Klartext in der Datei. Solche Files können Sie auch in die Textverarbeitung laden und dort bearbeiten. Wollen Sie derart manipulierte Dateien wieder in das BASIC laden, müssen Sie diese häufig per MERGE in den Speicher bringen - LOAD funktioniert dann nicht.

Etwas schwieriger ist die Übernahme von Dateien aus gänzlich andersartiger Software in die Textverarbeitung. Das kommt auf den Versuch an. Sie können zum Beispiel die Datei DESKTOP.INF als Text laden und ansehen. Eine Veränderung wird im Normalfall nichts bringen, aber theoretisch können Sie die Daten so quasi per Hand manipulieren. Auch die Dateien aus einer Tabellenkalkulation oder aus Grafikprogrammen werden im Editor selten sinnvolle Ergebnisse bringen.

Manche Programme erlauben es, die Druckausgabe nicht zum Drucker, sondern in eine Datei zu schicken. Beim Drucken in eine Datei handelt es sich nicht, wie auf den ersten Blick zu vermuten wäre, um eine spezielle Art des Speicherns von Texten. Die Datei wird tatsächlich Byte für Byte so erzeugt, wie sie auch an den Drucker geschickt würde - inklusive aller Steuercodes für den Drucker. Wenn Sie solch eine Datei vom Desktop aus auf dem Drucker ausgeben lassen, erhalten Sie haargenau den gleichen Ausdruck wie aus dem Textsystem. Sie sparen sich aber das mitunter langwierige Laden des Programmes. Sie sollten die Option benutzen, wenn Sie den gleichen Text häufiger ausdrucken müssen (z.B. für Aufkleber, Formulare usw.). Statt dann jedesmal die Textverarbeitung zu starten, reicht ein Doppelklick auf die Druckdatei.

Die Möglichkeit, ein Inhaltsverzeichnis oder ein Stichwortregister erstellen zu lassen, ist oft an das Ausdrucken des Textes gebunden. Wenn Sie das Inhaltsverzeichnis nachträglich erstellen möchten, z.B.: weil Sie das beim Druck vergessen haben, ist das kein Beinbruch. Hier gibt es einen kleinen Trick. Falls Ihr Drucker an der seriellen Schnittstelle angeschlossen ist, klemmen Sie das Kabel bitte ab (Geräte ausschalten). Bei Druckern an der Centronics-Schnittstelle entfällt dieser Schritt. Dann stellen Sie das Accessory "Drucker Anpassung" bitte so ein, daß der ST von einem Drucker an der seriellen Schnittstelle (Modem) ausgeht. In der "RS 232 Einstellung" setzen Sie bitte alle Handshaking-Parameter auf "Aus", die Baudrate auf 9600 und den Duplex-Modus auf "Halb". Dann lassen Sie den Text noch einmal ausdrucken - diesmal mit eingeschalteter Inhaltsverzeichnis- bzw. Stichwortregister-Option. Die Daten gehen über die serielle Schnittstelle hinaus, an der kein Gerät angeschlossen ist. Das merkt der ST aber nicht, weshalb der Druck wie üblich abläuft - nur es erscheint eben nichts auf dem Papier. Dabei kann das Textverarbeitungsprogramm das Inhaltsverzeichnis bzw. das Stichwortregister erstellen und in einer Datei ablegen.

9.2 Dateiverwaltungen und Datenbanken

Wenig bekannt ist der Unterschied zwischen Dateiverwaltung und Datenbank. Die Dateiverwaltung ist die einfachere Form. Hier kann nur eine Datei verwaltet werden, z.B. eine Adressenkartei. Einfachere Exemplare gehen von festen Formaten der Datensätze aus, weiter entwickelte Programme ermöglichen dem Anwender die Definition der Datensätze nach eigenen Bedürfnissen. Eine Kombination der Datei mit fremden Daten ist nicht möglich. Zur Standardausrüstung zählen außerdem Funktionen zum Suchen von Adressen nach bestimmten Kriterien, evtl. auch nach mehreren Kriterien gleichzeitig. Zusätzlich können meist auch noch Gruppen von Datensätzen ausgesondert werden (z.B. alle Adressen zum Namen Müller).

Im Unterschied dazu sind Datenbanken sehr viel flexibler. Hier können mehrere Dateien verwaltet und kombiniert werden. Ein Beispiel: Sie haben sowohl eine Adressendatei in Ihrer Datenbank als auch eine Geburtstagsdatei. In letzterer haben Sie sich die Geburtsdaten Ihrer Bekannten notiert. In der Adressendatei befinden sich nicht nur die Namen und Adressen Ihrer Freunde, sondern auch noch Geschäftsadressen, die nicht in der Geburtstagsdatei auftauchen. Umgekehrt mag es in der Geburtstagsdatei einige Geburtstagskinder geben, die nicht in der Adressendatei vorkommen. Einem guten Datenbanksystem können Sie nun den Auftrag erteilen, die Adressen aller Freunde auszugeben, die im Monat Juli feiern werden. Dabei kombiniert die Software automatisch die beiden Dateien und sucht nur diejenigen Adressen heraus, zu denen es auch ein Geburtsdatum gibt.

Mit einer Dateiverwaltung hätten Sie diese Aufgabe zwar auch mit vertretbarem Aufwand lösen können. Sie hätten dazu allerdings vor der Erstellung der Datei ein Feld mit dem Geburtsdatum erstellen müssen, das dann in jeder Adresse wieder aufgetaucht wäre (also auch bei den Geschäftsadressen). Es leuchtet schnell ein, daß Datenbanken sehr viel flexibler sind. Diese Flexibilität hat allerdings ihren Preis - nicht nur in geldlicher Hinsicht. Datenbanken werden überwiegend mit einer speziellen Zugriffssprache bedient, die zunächst erlernt werden muß. Ein Zugriff könnte dann so oder ähnlich aussehen:

```
SELECT Adresse FROM Adressendatei
WHERE Name IN Geburtstagsdatei
      AND Monat FROM Geburtstagsdatei = 'JULI'
```

Eine Steigerung des Datenbankenbegriffes stellen die relationalen Datenbanken dar. Hier braucht sich der Anwender nicht mehr um die Organisation der Daten zu kümmern, dies übernimmt das System. Zur Definition von Dateien genügt es allein, die sogenannten funktionalen Abhängigkeiten zwischen den Feldern anzugeben. Ein Beispiel für eine solche funktionale Abhängigkeit ist der Zusammenhang zwischen Name und Adresse. Im allgemeinen kann jede Adresse allein durch die Angabe des dazugehörigen Namens gefunden werden. (Namensgleichheiten seien hier der Einfachheit halber ausgeschlossen.). In gleicher Weise kann man aber auch an das Geburtsdatum herankommen. Relationale Datenbanken sammeln solche funktionalen Abhängigkeiten und ordnen die Daten so an, daß man sie über diese Beziehungen abrufen kann. Dabei hört sich der Begriff "funktionale Abhängigkeit" schlimmer an als es ist, denn er entspricht einfach dem landläufigen Begriff einer Tabelle. Um bei unserem obigen Beispiel zu bleiben: Wir benutzen eine Tabelle mit den beiden Spalten Name und Adresse. In jeder Zeile dieser Tabelle ist dann eine Kombination von Name und Adresse eingetragen, nämlich genau die Adresse, die zu diesem Namen gehört.

Es gibt Anwender, die mit einer Dateiverwaltung die Adressen ihres Bekanntenkreises oder Ihre Schallplattensammlung archivieren. Aber seien wir ehrlich: Im Normalfall ist die Adresse aus dem ca. 100 Leute umfassenden Bekanntenkreis im Notizbuch schneller gefunden, und wo die gesuchte Schallplatte steht, weiß man sowieso auswendig. Etwas anders sieht der Fall aus, wenn wirklich große Datenmengen zu verwalten sind, also beispielsweise die Kundenkartei eines Handwerkers oder das mittlerweile recht groß gewordene Archiv eines Journalisten. Als Faustregel läßt sich festhalten, daß eine Dateiverwaltung immer da angebracht ist, wo man mehr als 10 Minuten täglich mit dem Suchen von bestimmten, immer ähnlich aussehenden Informationen zubringt. Legen Sie besonderen Wert auf Flexibilität, dann müssen Sie sich unter Umständen sogar eine Datenbank anschaffen.



Programme, die die deutschen Umlaute nicht korrekt behandeln (also das "ä" nicht nach dem "a", sondern nach dem "z" einordnen), haben in Ihrem Diskettenkasten nichts zu suchen - denn damit handeln Sie sich unter Umständen mehr zusätzlichen Aufwand als Erleichterung ein. Testen Sie solche Software ggf. beim Händler. Dateiverwaltungen wie Datenbanken sollten außerdem in der Lage sein, diverse Formen von Auswertungen zu erstellen: Listen, Aufkleber usw. Standard ist eigentlich, daß nicht nur nach einzelnen Daten, sondern nach ganzen Bereichen (z.B. alle Adressen aus dem Postleitzahlbereich 4) gesucht werden kann und die Ergebnisse dieser Auswertungen dann in den genannten Listen dargestellt werden. In vielen Fällen kann man auch nach mehreren Kriterien gleichzeitig suchen lassen (z.B. alle Müllers im Postleitzahlbereich 4). Sehr nützlich sind auch Funktionen für den Import und Export, d.h. Übernahme bzw. Übergabe von Daten aus bzw. an andere Programme.

Ein häufig benutzter Begriff im Zusammenhang mit Dateiverwaltungen ist "Datensatz". Ein Datensatz enthält eine Reihe von Datenfeldern, in die zusammengehörige Informationen eingetragen werden können. So ist zum Beispiel eine Adresse ein Datensatz, in dem Felder wie Name, Straße, Ort usw. existieren. Das Aussehen eines Datensatzes muß festgelegt werden, bevor die ersten Daten eingegeben werden. Dabei bestimmen Sie unter anderem, wie viele Zeichen pro Datenfeld reserviert werden.

Ein Datensatz wird vom Computer in den meisten Fällen als eine Einheit angesehen, d.h. wenn Sie die Daten nach Namen ordnen lassen, werden auch die übrigen Felder in der durch die Namen vorgegebene Reihenfolge mitsortiert. Bei der Festlegung des Datensatzaussehens müssen Sie in vielen Programmen angeben, welches der Felder das Indexfeld ist. Die gesamte Datei wird nach dem Indexfeld sortiert, d.h. mit diesem wird die Reihenfolge der Datensätze bestimmt. Wählen Sie in einer Adressendatei die Namen als Indexfeld, so

werden die Adressen in der Reihenfolge der Namen gespeichert, wählen Sie das Ortsfeld, so richtet sich die Reihenfolge nach dem Ort in einer Adresse. Sie sehen also, daß ein Indexfeld eine sehr wichtige Rolle hat, denn nach Telefonnummern sortierte Daten sind im allgemeinen recht unübersichtlich. Das soll aber nicht heißen, daß Sie solcherart geordnete Daten nicht auch in eine andere Reihenfolge bringen können.

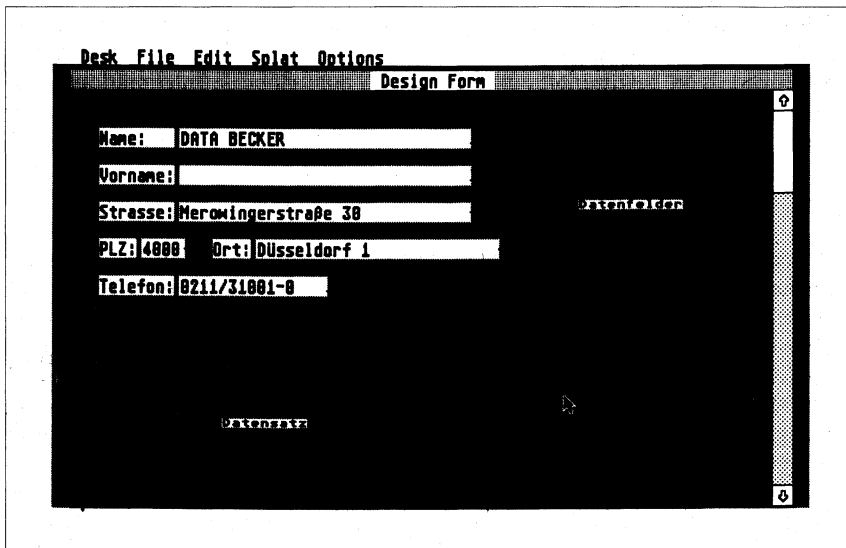


Abb. 9.4: Ein Datensatz

In vielen Fällen ist auch die Übernahme von Informationen aus einer Datenbank oder Dateiverwaltung in andere Programme möglich. So steht in den meisten Fällen nichts dagegen, wenn Sie eine Auswertung als Text in die Textverarbeitung laden und dort nachbearbeiten. Umgekehrt können Sie eine Adressenliste, die als Text vorliegt, im Normalfall leider nicht in eine Dateiverwaltung übernehmen. Viele Textverarbeitungen erlauben außerdem den Druck von Serienbriefen. Dazu wird zunächst mit der Dateiverwaltung eine Liste von Adressen erstellt, an die der Brief geschickt werden soll. Das Textverarbeitungsprogramm übernimmt diese Liste und setzt die Adressen in jeden Brief automatisch beim Druck ein.



Viele Programme erlauben auch die Ausgabe in einem sogenannten ASCII- oder DIF-Format (DIF = Data Interchange Format, zu deutsch Datenaustauschformat). Dahinter verbirgt sich nichts anderes als die Ausgabe aller Felder als Klartext, also ohne interne Steuerinformationen der Software.

Mit ein wenig Experimentieren gelingt es häufig, diese Daten mit anderen Programmen (insbesondere Textverarbeitungen o.ä.) weiterzuverarbeiten.

9.3 Grafiksoftware

Wer sich mit Grafiksoftware schon beschäftigt hat, wird schon auf die Begriffe "pixelorientiert" und "vektororientiert" gestoßen sein. Ein Pixel ist ein Bildpunkt. Pixel-orientierte Grafikprogramme arbeiten demnach so, daß man jeden einzelnen Punkt eines Bildes mit der Maus einfärben kann. Dazu muß eine sogenannte Bit-Map gespeichert werden, also eine Art Landkarte, die für jeden Punkt die Art der Färbung angibt.

Bei der Verwendung eines Monochrom-Monitors stehen dem Künstler natürlich nur zwei Farben (schwarz und weiß) zur Verfügung. Aber, wie man am Beispiel des Desktop sieht, durch geschickte Auswahl der geschwärzten Punkte kann man auch Graustufen erzeugen. Pixel-orientierte Programme haben diverse Vorteile, so gehört zum Beispiel eine sogenannte Spritzpistole zur Standardausstattung. Damit ist eine Funktion gemeint, die ziemlich zufällig Punkte in einem eng begrenzten Kreis setzt und so einen Effekt erzielt, der ungefähr dem Sprühnebel einer Spraydose entspricht. Der Nachteil: Hat man sich vermalte, müssen die falsch gesetzten Bildpunkte häufig mit einem "Radiergummi" (hier aber elektronisch) wieder gelöscht werden, was recht mühsam sein kann, wenn sich die gezeichnete Figur mit einer anderen überschneidet.

Pixel-orientierte Bilder sind recht speicherplatzintensiv, dafür kann man sie meistens direkt vom Bildschirm abfotografieren (siehe unten). Aus diesem Grunde kann man meist auch nur eine Bildschirmseite bearbeiten. Um Punkte genauer setzen zu können, verfügen die meisten Programme auch über eine sogenannte Bit-Lupe, d.h. ein vorher bestimmtes Gebiet wird mehrfach vergrößert angezeigt und kann dann nachbearbeitet werden. Bekannte Programme auf diesem Gebiet sind DOODLE, DEGAS, NEOCHROME und PROFI PAINTER.



Das Ausbügeln von Fehlern in Pixel-Grafiken ist sehr arbeitsintensiv. Viele Programme erlauben aber das Arbeiten mit mehreren Bildschirmen gleichzeitig. Man sollte daher nach etwa zehn Arbeitsschritten die bisherige Zeichnung in eine andere Bildseite kopieren und dort mit der Bearbeitung fortfahren. Macht man dann Fehler, kann man immer noch auf die letzte Seite zurückgreifen und mit der alten Version weitermachen.

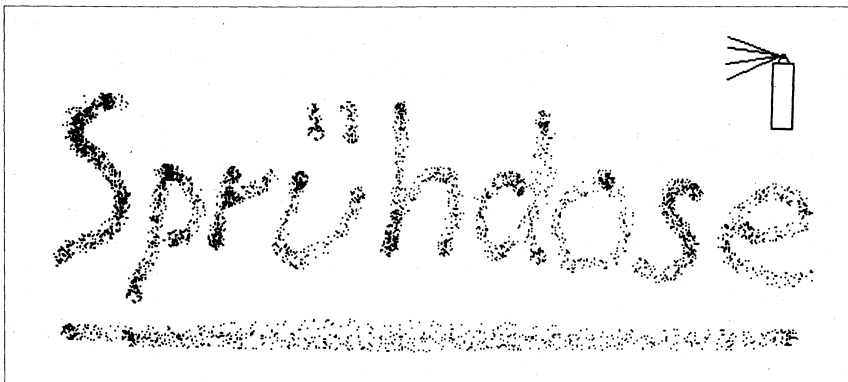


Abb. 9.5: Der Sprüh Dosen-Effekt

Eine andere Kategorie sind die vektor- oder objekt-orientierten Programme. Ein Objekt kann jedes grafische Element sein, also zum Beispiel eine Linie, ein Kreis, ein Rechteck usw. Statt einer Bit-Map speichert ein solches Programm eine Liste von Objekten, in denen die Koordinaten und das Aussehen etwa in der folgenden Art gespeichert sind: "Kreis mit Radius 100 Punkte und Mittelpunkt 200,200". Durch das Zeichnen mit der Maus werden neue Elemente in diese Liste aufgenommen. Der Clou dabei: Im Gegensatz zu pixelorientierten Programmen kann man die gezeichneten Objekte nachträglich verändern. Man klickt zum Beispiel einen Kreis an und vergrößert seinen Radius nachträglich auf 150 Punkte. Für das Programm bedeutet das lediglich das Ändern der Liste und anhand dieser das Neuzeichnen des Bildes. Genauso kann man sich auch ein lange vorher gezeichnetes Objekt herausgreifen und dieses wieder löschen oder an eine andere Stelle des Bildes verschieben. Der Speicherbedarf des Bildes ist bei objekt-orientierten Programmen sehr viel geringer; auch ermöglichen die meisten Programme ein Arbeiten mit Bildgrößen von DIN A4 oder DIN A3. Um jede Stelle des Bildes gleich gut bearbeiten zu können, kann man ein Fenster wie einen Rahmen über dieses Bild schieben und sich den jeweils benötigten Ausschnitt heranziehen. Sogar das Zoomen ist vorgesehen. Bekannteste Vertreter dieser Gattung von Grafiksoftware sind EASY-DRAW, STAD sowie alle CAD-Programme wie BeckerDesign, CAMPUS und GFA-DRAFT.

Es ist schwer zu sagen, ob nun pixel- oder objekt-orientierte Grafikprogramme besser sind. Das hängt von Ihren Wünschen ab. Wollen Sie eher malend - im darstellenden Sinne - tätig werden, dann wird ein pixel-orientiertes Programm wegen der leichter erzielbaren Effekte für Sie besser sein. Diese Software entspricht wohl am meisten dem klassischen Pinsel. Wenn Sie eher an

(technische) Zeichnungen wie etwa Struktogramme, Diagramme oder gar Konstruktionsskizzen denken, dann sollten Sie zum objekt-orientierten Programm greifen, das weitgehend den Drehbleistift nachahmt.



Wenn Sie häufig programmieren, werden Sie wahrscheinlich auch hin und wieder Struktogramme zeichnen müssen. Das ist mit Papier und Bleistift nicht sehr bequem. Wenn Sie diese Arbeit mit einem objekt-orientierten Grafikprogramm durchführen, sind Sie nach einer kurzen Einarbeitungszeit schneller und außerdem haben Sie die Möglichkeit, Fehler nachträglich auszubügeln.

Der Transfer von Grafiken ist nur bedingt möglich, da wieder einmal jedes Programm seine eigenen Speicherformate hat. Bei pixel-orientierter Software besitzen aber alle diese Formate eine gewisse Ähnlichkeit, so daß die Programme selbst bereits vielfach in der Lage sind, fremde Bilder einzulesen. Einzelne, weitergehende Übertragungsmöglichkeiten sind unten beschrieben. Der Transfer zwischen objekt-orientierten Programmen ist noch schwieriger. Sollten Sie jedoch mit EASY-DRAW oder anderen Programmen arbeiten, die ihre Bilder per Metafile als Dateien mit der Endung GEM ablegen, so können Sie diese Grafiken uneingeschränkt austauschen - sogar mit GEM-Programmen auf PCs. Der Austausch von Bildern zwischen pixel- und objekt-orientierten Programmen ist in der Regel unmöglich. Einzige Möglichkeit: Sie "fotografieren" den Bildschirminhalt eines objekt-orientierten Grafikprogrammes mit einem speziellen Programm und übertragen diese Daten dann in die pixel-orientierten Programme.

Etwas anders sieht es da mit den objekt-orientierten Grafikprogrammen aus. Das sogenannte Metafile ist eine Einrichtung des GEM, die einen gewissen Standard bei der Speicherung objekt-orientierter Grafiken bringen soll. Statt die Datei auf der Diskette oder Festplatte selbst zu erzeugen, übergibt das Programm die Daten einem sogenannten Metafile-Treiber, der daraus eine Datei in festgelegtem Format erzeugt. Sie können diesen Treiber am File-Namen META.SYS auf der Diskette erkennen.

Zunächst einmal muß META.SYS im System installiert sein, um mit Metafiles arbeiten zu können. Das wird automatisch erreicht, wenn beim Booten eine Diskette benutzt wurde, in deren AUTO-Ordner das Programm GDOS.PRГ steht. GDOS ist eine Art Betriebssystemerweiterung, das für die Behandlung von Treibern wie META usw. zuständig ist. Es gibt aber auch Programme, die GDOS nutzen, ohne auf Metafiles zurückzugreifen (Beispiel: Profi-Painter). Sie können das daran feststellen, daß in der Datei ASSIGN.SYS der Metafile nicht erwähnt wird. Wenn Sie nun eine GEM-Datei laden oder speichern wollen, muß in dem Laufwerk, von dem gebootet wurde, die Systemdiskette liegen -

unabhängig davon, auf welchem Laufwerk Sie die Bild-Datei liegen haben. Das GEM holt sich bei dieser Gelegenheit den META-Treiber von der Diskette und übergibt ihm die Daten, die dieser dann umwandelt.

▼ Einige Anwenderprogramme arbeiten auf keinen Fall, wenn das GDOS.PRG bei Start im AUTO-Ordner lag (z.B. Beckertext 1.0). Wenn Sie also nach der Benutzung eines GDOS-Programmes mit solcher Software arbeiten wollen, müssen Sie auf jeden Fall neu booten (mit einer Systemdiskette ohne GDOS).

Wenn Sie DOODLE-Bilder in DEGAS einlesen möchten, geht das mit einem kleinen Trick. DEGAS und DOODLE benutzen eigentlich bei schwarzweißen Bildern das gleiche Format, nur Degas setzt noch einige Informationen hinzu. Ohne diese Zusatzinformationen weigert sich DEGAS aber auf jeden Fall, ein Bild einzulesen. Das folgende Programm behebt das:

Hier die Version für GFA-BASIC:

```
' HEADER - konvertiert DOODLE-Bilder in das DEGAS-Format
' Konvertierungsdaten
Data 0,2,7,119,0
Data 0,0,0,0,0
Data 0,0,0,0,0
Data 0,0,0,0,0
Data 0,0,0,0,0
Data 0,0,0,0,0
Data 0,0,0,0
' File auswählen
Fileselect "*.**", "", Name$
Hidem
' Datei laden
Bload Name$, Xbios(2)
Void Inp(2)
' Header im Speicher installieren
For I=1 To 34
  Read H%
  Poke Xbios(2)-35+I, H%
Next I
' Extension ändern
Gosub Extension(Name$)
Let Name$=Left$(Name$, Extpos)+"PI3"
' neue Datei speichern
Bsave Name$, Xbios(2)-34, 32034
'

Procedure Suchpfad(F$)
  Local S, Sh, D
  ' Letzten Backslash suchen
  S=Instr(F$, "\")
  While S<>0
```

```
' Noch weitere Backslashes vorhanden?
Sh=S
S=Instr(S+1,F$,"\")
Wend
' letzten gültigen Wert zurückholen
S=Sh
If S<>0 Then
' Backslash gefunden, Suchpfad klar
Pos=S
Else
' Kein Backslash, evtl. Laufwerksbezeichnung
D=Instr(F$,".")
Pos=D
Endif
Return
Procedure Extension(F$)
Local P
' Zunächst Länge des Suchpfades = erste Suchposition berechnen
Gosub Suchpfad(F$)
' Variable POS gibt Länge des Suchpfades an
' Punkt suchen
P=Instr(Pos+1,F$,".")
If P=0 Then
' Kein Punkt vorhanden, also auch keine Extension
Extpos=Len(F$)
Else
Extpos=P
Endif
Return
```

Hier die Version für Omikron- bzw. ST-BASIC:

```
1 ' HEADER - konvertiert DOODLE-Bilder in das DEGAS-Format
2 ' Konvertierungsdaten
3 DATA 0,2,7,119,0
4 DATA 0,0,0,0,0
5 DATA 0,0,0,0,0
6 DATA 0,0,0,0,0
7 DATA 0,0,0,0,0
8 DATA 0,0,0,0,0
9 DATA 0,0,0,0
10 ' File auswählen
11 PRINT CHR$(27)+"f"
12 Pfad$="\*.*)"
13 Dn$=""
14 MOUSEON
15 FILESELECT (Pfad$,Dn$,A%)
16 MOUSEOFF
17 CLS
18 ' Datei laden
19 Suchpfad(Pfad$)
20 Pfad$= LEFT$(Pfad$,Ps%L)
```

```

21 XBIOS (Ad%L,2)
22 BLOAD Pfad$+Dn$,Ad%L
23 A$= INPUT$(1)
24 ' Header im Speicher installieren
25 FOR I%L=1 TO 34
26   READ H%
27   POKE Ad%L-35+I%L,H%
28 NEXT I%L
29 ' Extension ändern
30 Extension(Dn$)
31 Dn$=Pfad$+ LEFT$(Dn$,Extpos%L)+"PI3"
32 ' neue Datei speichern
33 BSAVE Dn$,Ad%L-34,32034
34 MOUSEOFF
35 PRINT CHR$(27)+"e"
36 END
37 '
38 DEF PROC Suchpfad(F$)
39   LOCAL S%L,Sh%L,D%L
40   ' Letzten Backslash suchen
41   S%L= LEN(F$)- INSTR( MIRROR$(F$),"\")+1
42   IF S%L<= LEN(F$) THEN
43     ' Backslash gefunden, Suchpfad klar
44     Ps%L=S%L
45   ELSE
46     ' Kein Backslash, evtl. Laufwerksbezeichnung
47     D%L= INSTR(F$,":")
48     Ps%L=D%L
49   ENDIF
50 RETURN
51 DEF PROC Extension(F$)
52   LOCAL P%L
53   ' Punkt suchen
54   P%L= INSTR(F$,".")
55   IF P%L=0 THEN
56     ' Kein Punkt vorhanden, also auch keine Extension
57     Extpos%L= LEN(F$)
58   ELSE
59     Extpos%L=P%L
60   ENDIF
61 RETURN

```



Nach dem Start erscheint eine File-Select-Box, mit der Sie das zu konvertierende Bild anwählen können. Es wird auf dem Bildschirm angezeigt. Danach wartet das Programm zunächst auf einen Tastendruck. Sobald Sie eine beliebige Taste gedrückt haben, wird das Bild konvertiert und in das gleiche Verzeichnis gespeichert, in dem auch das Original steht. Dabei wird die Extension von DOO in PI3 geändert. Beachten Sie bitte, daß für die Konvertierung nur Bilder aus dem Schwarzweiß-Modus in Frage kommen.



Das obige Konvertierungsprogramm eignet sich auch für Bilder, die mit dem Programm GRAF_PRO aus dem GFA-BASIC-Buch erstellt wurden. Außerdem sollte es bei allen Bilddateien funktionieren, deren Länge genau 32000 Bytes beträgt.



Wählen Sie eine Datei aus, die mehr als 32000 Bytes umfaßt, wird Ihr ST mit fast tödlicher Sicherheit abstürzen. Also Vorsicht!

Eine wichtige Funktion ist die Einbindung von Grafiken in eine Textverarbeitung. Wie weit das funktioniert, hängt von der Textverarbeitung ab. Einige Programme sind in der Lage, Bilddateien direkt zu verarbeiten, bei anderen muß der Bildschirm erst "fotografiert" werden, d.h. mit einem speziellen Programm (z.B. BT_SNAP.PRГ von der Beckertext-Diskette oder SNAPSHOT.ACC) wird der Bildschirminhalt in eine Datei auf der Diskette oder Festplatte übertragen. Diese Dateien können dann von der Textverarbeitung ausgedruckt werden.

Bilddateien aus BT_SNAP können Sie unter bestimmten Voraussetzungen auch in ein Grafikprogramm laden. Das geht nur, sofern Sie den ganzen Bildschirm fotografiert haben (in diesem Fall hat die Datei eine Länge von 32006 Bytes) und das Bild im Schwarzweiß-Modus aufgenommen wurde. Dann können Sie das Programm Header (siehe oben) einsetzen. Allerdings muß dabei in der GFA-BASIC-Version die Zeile

```
Bload Name$,Xbios(2)
```

in

```
Bload Name$,Xbios(2)-6
```

und in der Omikron-/ST-BASIC-Version die Zeile

```
22 BLOAD Pfad$+Dn$,Ad$L
```

in

```
22 BLOAD Pfad$+Dn$,Ad$L-6
```

geändert werden.



Die oben beschriebene Konvertierung klappt nicht bei Dateien, die mit SNAPIFF aus der Beckertext-Version 2.0 erzeugt wurden. IFF ist ein komprimiertes Grafikformat, das ursprünglich vom Amiga stammt!

9.4 Fremde Betriebssysteme

Für keinen anderen Computer als den ST gibt es so viele Möglichkeiten, ihn mit "fremden" Betriebssystemen zu betreiben. Obwohl Computer meistens nur für ein spezielles Betriebssystem vorgesehen sind (man denke nur an die vielen MS-DOS-Rechner, die quasi um das Betriebssystem herum gebaut sind), laufen viele dieser Systeme auch auf dem ST.

Das Betriebssystem bestimmt, welche Art von Programmen auf einem Rechner lauffähig sind. Nun ist es Ihnen bestimmt schon so ergangen, daß Sie beim Fachhändler oder bei einem Freund ein tolles Programm gesehen haben, das Sie auch benutzen möchten. Leider ist es aber nur für MS-DOS oder ein anderes System geeignet. Ein fremdes Betriebssystem auf Ihrem ST versetzt Sie in die Lage, diese Programme ebenfalls zu betreiben. In manchen Fällen ist das fremde Betriebssystem aber auch speziell für den ST bestimmt (z.B. Mirage). Die Hersteller solcher Systeme sind häufig der Meinung, daß das TOS für Spezialanwendungen (z.B. Maschinensteuerungen) nicht geeignet sei. Das neue Betriebssystem ist dann auf solche Aufgaben besonders vorbereitet. Schließlich gibt es noch Betriebssysteme, die einzig und allein zu Hobby- oder Lehrzwecken auf den ST übertragen wurden. MINIX ist ein Beispiel dafür.

Es gibt zwei Möglichkeiten, wie fremde Systeme auf dem ST laufen können. In einigen Fällen handelt es sich um Anpassungen fremder Systeme an den ST. Diese Lösung hat den Nachteil, daß die Programme für das fremde System nur die gewohnte Betriebssystemumgebung, nicht aber die gewünschte Hardware vorfinden. Dadurch kann es nötig werden, die Programme neu zu übersetzen (compilieren). Das scheitert in den meisten Fällen daran, daß so gut wie kein Hersteller den Quelltext seiner Programme herausgibt. Software für solche angepassten Betriebssysteme ist recht selten.

Die andere Möglichkeit ist, daß der ST so umgestaltet wird, daß er das fremde Betriebssystem akzeptiert und dieses scheinbar die gewünschte Umgebung vorfindet. Man nennt solche Lösungen daher im allgemeinen auch Emulatoren.

Emulatoren sind Programme und Hardware-Zusätze, die einen anderen Computer auf dem ST emulieren. In der EDV-Fachsprache sagt man emulieren, wenn ein Programm oder Computer einen anderen nachahmt; man spricht von Simulation, wenn ein Computer eine völlig artfremde Maschine, z.B. ein Flugzeug simuliert. Dabei wird das fremde Computermodell so vollständig und detailgetreu nachgeahmt, daß der ST in möglichst vielen Funktionen genau wie ein PC oder ein Macintosh reagiert. Für den ST gibt es mehrere Emulatoren; drei davon sind sehr bekannt: CPMZ80.TOS ahmt einen CP/M-Com-

puter mit Z80-Prozessor nach. Diesen Emulator gibt es schon seit 1985; er ist Public-Domain-Software und wird kaum noch gebraucht, da CP/M nach und nach von der Bildfläche verschwindet. PCDITTO emuliert einen PC-Kompatiblen vollständig per Software nach, der Aladin läßt den Apple Macintosh auf dem ST lebendig werden. Andere Emulatoren gibt es für die Rechner C-64 und Atari XL. Außerdem können Aladin-Besitzer einen weiteren Emulator benutzen, der (ursprünglich auf dem Mac) einen Apple II emuliert.

Es gibt im wesentlichen zwei Verfahren, nach denen ein Emulator arbeiten kann. Benutzt das emulierte Gerät den gleichen Prozessor wie der ST (was zum Beispiel beim Macintosh der Fall ist), so muß man nur die Umgebung dieses Prozessors im Speicher des ST nachahmen. So sind zum Beispiel die Organisation des Bildschirmspeichers und der Schnittstellen beim Mac ganz anders gelöst als beim ST. Der Emulator gaukelt dem Prozessor nun vor, daß eben diese veränderte Hardware vorliegt. Auf diese Weise ist er in der Lage, Programme für den fremden Computer auszuführen. Allerdings muß dazu das Betriebssystem des fremden Computers ebenfalls vorhanden sein. Das wird im allgemeinen bei einem Emulator nicht mitgeliefert.

Es kann aber auch sein, daß der simulierte Computer mit einem anderen Prozessor ausgestattet ist, der gänzlich andere Maschinenbefehle versteht als der 68000 im ST. Dann hat der Emulator noch die zusätzliche Aufgabe, den fremden Prozessor zu emulieren, also jeden fremden Maschinenbefehl in eine Reihe von 68000er-Befehlen umzusetzen. Es wird Ihnen einleuchten, daß darunter die Geschwindigkeit der Emulation leidet.

Die dritte Möglichkeit: Der Emulator ist hardwaremäßig realisiert, wie das einige Zeit lang im Gespräch war. Hier noch von einem Emulator zu sprechen, scheint mir allerdings etwas verfehlt, denn im Gerät steckt dann ein kleiner Computer für sich, der die Hardware des ST zur Ein- und Ausgabe nutzt. Seit der CeBit 1986 kursierten immer wieder Gerüchte, daß Atari einen solchen hardwaremäßigen PC-Emulator für den ST auf den Markt bringen wolle, vorgestellt wurde er bereits 1986. Aber bisher ist es bei den Gerüchten geblieben. Andere Firmen haben sich auch wiederholt an dieser Aufgabe versucht; und seit einigen Monaten gibt es nun auch endlich Hardware-PC-Emulatoren wie den ATonce, Supercharger, ATSpeed usw.

9.4.1 Der PC-DITTO

PC-DITTO ist ein softwaremäßiger PC-Emulator, d.h. auf dem ST wird ein IBM-PC nachgeahmt. Der IBM-PC ist der älteste der IBM-Personal-Computer und sollte nicht mit dem AT verwechselt werden, der mit einem anderen Pro-

zessor arbeitet. Sie können dann auf dem ST fast alle Programme für den IBM-PC laufen lassen, allerdings mit erheblichen Geschwindigkeitseinbußen.

Auf den folgenden Seiten kann auf keinen Fall eine Einführung in die Arbeit mit MS-DOS gegeben werden. Dazu ist dieses Betriebssystem viel zu komplex. Die Bedienung ist bei weitem nicht so einfach wie beim ST, so daß ich den MS-DOS-Neulingen unter Ihnen nur die Lektüre der einschlägigen Fachliteratur empfehlen kann. Ich gehe in den nächsten Abschnitten davon aus, daß Sie mit MS-DOS vertraut sind.

Auf den folgenden Seiten beschäftigen wir uns ausschließlich mit der Version 3.64 des PC-DITTO. Ältere Versionen als 3.0 arbeiten zum Teil auf gefährliche Weise anders als die neueren Ausgaben. Unter Umständen können dabei sogar Daten auf der Festplatte zerstört werden. Beachten Sie bitte die mitgelieferten Dateien und Anleitungen sehr genau!

PC-DITTO emuliert einen IBM-Computer so exakt, daß der ST, auf dem diese Emulation stattfindet, seine eigenen Disketten nicht mehr lesen kann. Das kann später unter Umständen zu Problemen führen. Außerdem müssen Sie wissen, daß mit dem PC-DITTO keinerlei Betriebssystem (MS-DOS) ausgeliefert wird. Das müssen Sie sich vorher besorgen.

Wenn Sie einen PC-kompatiblen Rechner mit 3,5-Zoll-Laufwerk besitzen, sind Sie schon aus dem Schneider. Kopieren Sie das DOS auf eine (vom PC formatierte) 3,5-Zoll-Diskette. Besitzen Sie keinen PC-Kompatiblen, so müssen Sie das DOS kaufen, dabei erhalten Sie dann auch ein Handbuch. Ich empfehle Ihnen auf jeden Fall, ein DOS mit einer möglichst großen Versionsnummer zu erwerben. (Zur Zeit der Erstellung dieses Buches war DOS 4.01 die neueste Version.). Dieses DOS muß auf jeden Fall auf einer 3,5-Zoll-Diskette im 720-KByte-Format gespeichert sein! Versionen unter 2.0 sind für die Arbeit mit PC-DITTO ungeeignet, Ihnen stehen dann unter anderem keine Ordner mehr zur Verfügung.

Kopieren Sie bitte zunächst alle Dateien von der PC-DITTO-Diskette auf eine eigene Diskette, und bewahren Sie das Original an einem sicheren Ort auf. Lesen Sie auf jeden Fall die Datei README.1ST; wenn Sie Besitzer einer Festplatte SH204 sind, bitte auch die Datei AHDFIX.DOC. Dann öffnen Sie den Ordner STPROGRA. Starten Sie nun bitte noch das ST-Programm PCD-MENU.PRG. Auf dem Bildschirm erscheinen mehrere kleine Buttons, die auf Mausklick ein Menü freigeben. Anhand der hier gemachten Einstellungen wird der PC-DITTO später konfiguriert. Gehen wir die Punkte einmal durch.

Disk

Hier wird eingestellt, welche Laufwerke der PC verwalten soll. Im Normalfall können die Einstellungen so belassen werden, wie sie bereits bestehen. Wenn Sie ein zweites Laufwerk an den ST angeschlossen haben, können Sie festlegen, von wo das DOS später gebootet wird. Steht der erste Schalter auf "FIRST", so wird das DOS von Laufwerk A geladen, und später wird das Laufwerk A des ST auch von DOS als Laufwerk A betrachtet. Möchten Sie, daß das externe Laufwerk von DOS als A: angesprochen wird, so müssen Sie hier "SECOND" anklicken. Mit dem zweiten Schalter legen Sie dann fest, welcher Art das externe Laufwerk ist. Wählen Sie 5,25 nur dann, wenn Sie wirklich eine große Floppy-Station an Ihren ST angeschlossen haben. Besitzen Sie kein zweites Laufwerk, so müssen Sie auf jeden Fall "None" anklicken.

Falls Sie eine Festplatte besitzen, können Sie diese auch unter DOS benutzen; klicken Sie den nächsten Schalter in diesem Fall bitte auf "Yes". Wenn Sie zwar eine Harddisk besitzen, diese aber nicht benutzen wollen, können Sie "No" wählen. Empfohlen werden kann die Benutzung der Festplatte unter DOS nicht, da das System aufgrund der Emulation so langsam arbeitet, daß vom Geschwindigkeitsvorteil nichts zu spüren ist. Wenn Sie allerdings Dateien bearbeiten wollen, die Sie mit GEM auf der Festplatte abgelegt haben, kommen Sie um die Benutzung der Harddisk nicht herum.

Auf der Festplatte können Sie sowohl MS-DOS- als auch TOS-Dateien in bunter Reihenfolge mischen. Das gilt nicht, wenn eine Partition für ein anderes Betriebssystem (z.B. Aladin) vorgesehen ist. Sie können solche Partitionen zwar auch unter MS-DOS lesen, empfehlenswert ist das aber auf keinen Fall. Im Normalfall sollten Sie daher den nächsten Schalter (Ignore Non-GEM-Partitions) auf "Yes" stellen. Ausnahme: Wenn Sie eine Festplatte von einem Fremdhersteller benutzen, die mehr als vier Partitions hat, müssen Sie unter Umständen den Schalter auf "No" stellen, weil die fünfte und alle folgenden Partitionen dann ignoriert würden, auch wenn diese GEM gehören.

Wenn Sie versuchen, eine Festplatten-Partition mit weniger als 4,2 Megabyte unter Ditto-Versionen kleiner 3.0 zu verwenden, werden die darauf befindlichen Daten zerstört. Also Vorsicht!

Haben Sie Einstellungen verändert, verlassen Sie das Menü bitte per "Save Changes".

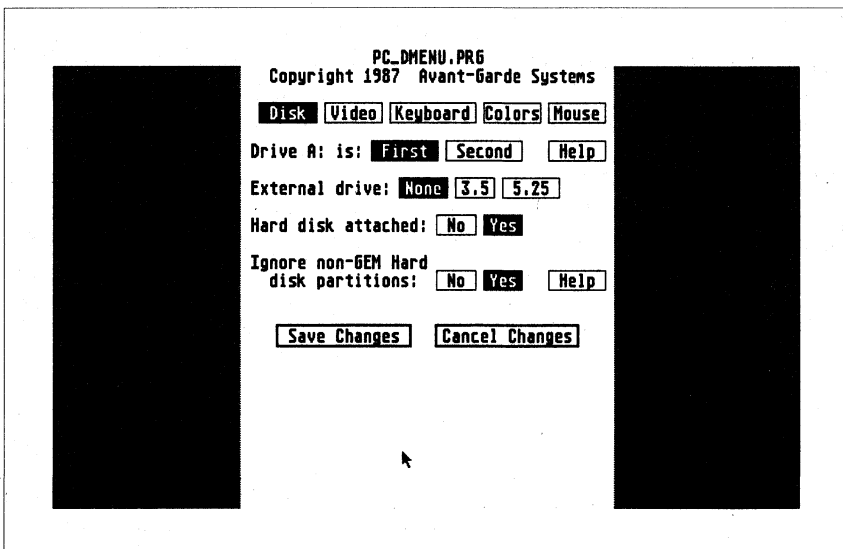


Abb. 9.6: Die Einstellungen unter Disk

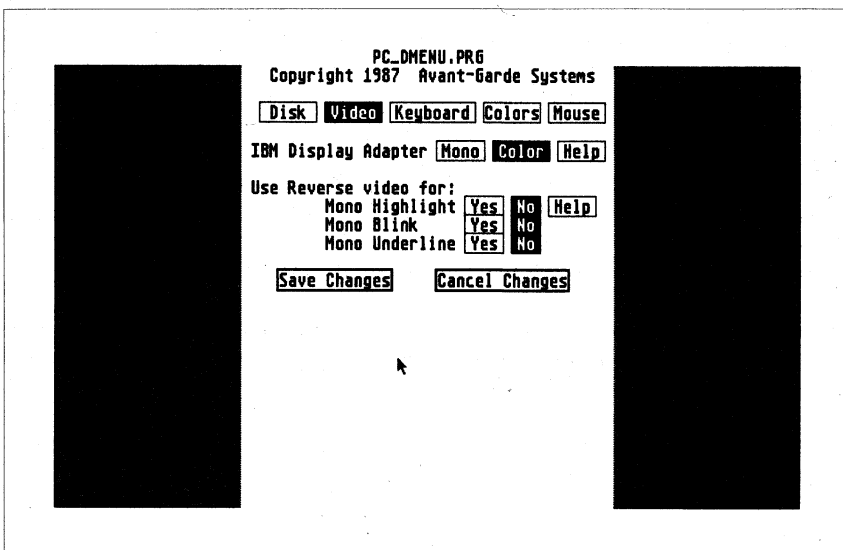


Abb. 9.7: Die Einstellungen unter Video

Video

Hier wird eingestellt, welche Art von Video-Karte emuliert wird. Der ST kann auch beim Betrieb mit dem SM124 den CGA-Modus emulieren; die Farben werden dann als Graustufen dargestellt. Wählen Sie unter "Display Adapter" die Option "Mono", so wird ein IBM-Monochrom-Adapter ohne Grafik emuliert; dies ist in den meisten Fällen nicht empfehlenswert. In der Stellung "Color" wird der CGA-Modus eingeschaltet. Sie können wählen zwischen reverser und normaler Darstellung der Zeichen.

Keyboard

Hier stellen Sie - fast wie im Kontrollfeld - ein, welche Geschwindigkeit und Verzögerung die Tastaturwiederholung haben soll. Außerdem müssen Sie angeben, mit welcher Tastenbelegung Sie arbeiten möchten. Wählen Sie hier GERMAN. Durch diese Einrichtung regelt der PC-DITTO die Tastaturbelegung ganz alleine. Das bedeutet, daß Sie unter DOS das Programm KEYB oder KEYBGR und auch den Treiber KEYBOARD.SYS nicht einsetzen dürfen. Streichen Sie diese Programme bitte aus den Dateien CONFIG.SYS und AUTOEXEC.BAT. Andernfalls würden zwei Treiber gegeneinander arbeiten und den ganzen Rechner zum Absturz oder zumindest zu sehr seltsamen Reaktionen bringen.

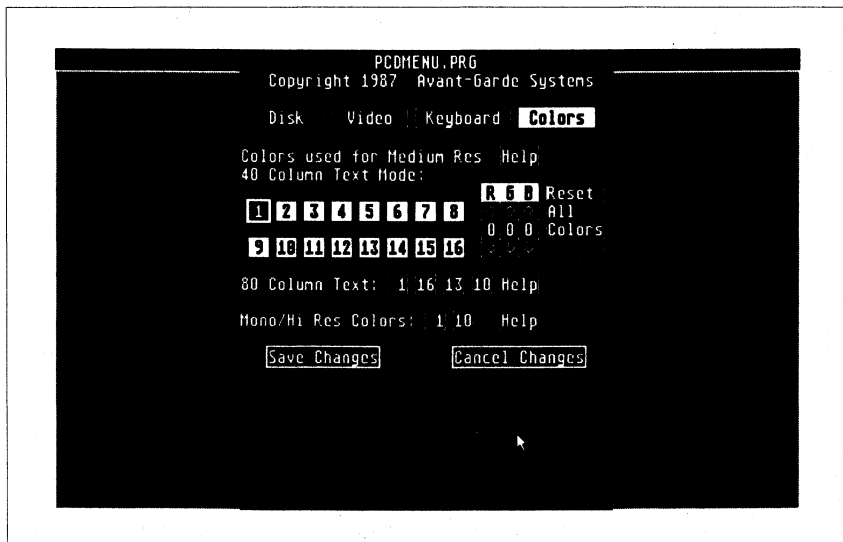


Abb. 9.8: Die Einstellungen unter Colors

Colors

Hier wird - ebenfalls analog zum Kontrollfeld - die Farbgebung in den verschiedenen Bildschirmmodi eingestellt. Im allgemeinen können Sie die Einstellungen so belassen, wie sie sind.

Wenn Sie Änderungen an den Einstellungen vorgenommen haben, verlassen Sie PCDMENU bitte über "Save Changes".

Damit ist der PC-DITTO fertig installiert.

Maus

Ab Version 3 kann DITTO auch die Maus des ST so umsetzen, daß ein PC darauf reagiert. Dazu gibt es zwei Möglichkeiten: die serielle Maus und eine Generic-Maus. "Serielle Maus" bedeutet, daß eine Microsoft-Maus, das am weitesten verbreitete Modell für PCs, emuliert wird. Diese entspricht weitgehend der vom ST gewohnten Arbeitsweise. Die Generic-Maus setzt die Mausbewegungen in Tastendrucke um. Üblicherweise werden so CursorBewegungen erzeugt. Für die linke und rechte Taste der Maus können Sie angeben, welchen Tastendruck diese entsprechen sollen. Dazu müssen Sie die gewünschten Zeichen in die beiden dafür vorgesehenen Felder eintragen. Diese Einstellung gilt allerdings nur für die Generic-Maus. Bei der seriellen Maus sind sie wirkungslos. Dafür müssen Sie für letztere Emulation zusätzlich den Treiber PC_DMOUS.COM starten. Setzen Sie dazu einfach die Zeile

PC_DMOUS

in Ihre AUTOEXEC.BAT auf der MS-DOS-Systemdiskette. Haben Sie alle Einstellungen gemacht, verlassen Sie das Programm bitte mit SAVE ALL CHANGES.

Gestartet wird der Emulator einfach per Doppelklick auf PC-DITTO.PRG. Das ist das eigentliche Emulationsprogramm, das im ST die PC-Umgebung herstellt. Nach dem Start erscheint die Aufforderung, entweder eine DOS-Diskette einzulegen (auf dieser muß das Betriebssystem MS-DOS/PC-DOS installiert sein) und **[Return]** zu drücken oder per **[Esc]** zum Desktop zurückzukehren.

Unter MS-DOS können Sie nur solche Disketten benutzen, die im MS-DOS-Format formatiert worden sind. Obwohl das ST-Format physikalisch gleich ist (80 Spuren mit je 9 Sektoren), ist es logisch anders aufgebaut. Ein MS-DOS-Rechner wird sich weigern, auch nur das Inhaltsverzeichnis einer ST-Diskette anzuzeigen. Sie können solche MS-DOS-Disketten unter PC-DITTO erzeugen.

Glücklicherweise ist aber umgekehrt der ST in der Lage, MS-DOS-Formate sowohl zu lesen als auch zu beschreiben.

Besitzen Sie das neue TOS 1.04, so entfallen diese Probleme. Ihre Disketten werden automatisch so formatiert, daß sowohl ST als auch PC sie lesen und beschreiben können.

Der Knackpunkt bei der Verwendung des richtigen Diskettenformats ist der Boot-Sektor. Bei ST-formatierten Disketten enthält er speziellen 68000er-Code, der von DOS nicht erkannt werden kann. Bei der Formatierung unter DOS wird dagegen 8086-Code aufgebracht, den der ST zum Glück entschlüsseln kann. Das Lesen des Boot-Sektors ist aber zwingend notwendig, weil hier das genaue Format einer Diskette angegeben ist. Nur wenn der Rechner diese Angaben kennt, ist er auch in der Lage, das Inhaltsverzeichnis korrekt anzuzeigen.

Etwas schwierig ist es, eine DOS-formatierte Diskette zu erhalten. Der einfachste Weg wird sein, das MS-DOS-Kommando `FORMAT` auszuführen. Das hat aber den Nachteil, daß die Diskette nur mit 40 Spuren beschrieben wird und somit nur 360 KByte Speicherkapazität besitzt. Auch der ST kann solche Disketten lesen und beschreiben. Wollen Sie DOS-Disketten mit den vollen 720 KByte formatieren, können Sie unter DOS `PC_DFMT.COM` ausführen lassen. Dieses Programm formatiert 720-KByte-DOS-Disketten.

Die Laufwerke des ST werden beim DOS nicht als 3,5-Zoll-Geräte angemeldet; dies wäre auch nur ab der Version 3.2 möglich. Der PC-DITTO kann aber bei der Installation noch nicht wissen, welche DOS-Version später geladen wird. Aus diesem Grund wird das ST-Laufwerk als 5,25-Zoll-Gerät betrachtet, was dazu führt, daß bei jedem Aufruf von `FORMAT` die Diskette mit 360 KByte (dem Standard-Format für 5,25 Zoll) formatiert wird. Diese Klippe ist leider nicht zu umgehen, außer durch die Verwendung von `PC_DFMT.COM`.

Starten Sie das Programm durch `PC_DFMT`. Gegebenenfalls müssen Sie es von Ihrer PC-DITTO-Diskette (es liegt im Ordner `PC_PROGR`) auf eine DOS-formatierte Diskette kopieren. Nach dem Start erscheint ein Menü vor Ihnen, in das Sie einige Eintragungen machen müssen, um DOS-Disketten zu formatieren. Sie können den Cursor mit den Cursor-Tasten auf und ab bewegen und jeweils die gewünschten Parameter eintippen bzw. mit ☐ / ☐ einstellen. Die Einstellung für 720-KByte-Disketten lautet: x (hier setzen Sie bitte das Laufwerk ein, auf dem formatiert werden soll), 80 Spuren, 9 Sektoren, 2 Seiten. Zusätzlich können Sie noch einen Namen für die Diskette vergeben. Die Option "Reserve Space" benötigen Sie nur in einigen wenigen Spezialfällen für sehr alte DOS-Versionen. Durch `F10` starten Sie dann den Formatierungsvorgang.

Seit Version 3.2 ist MS-DOS auf die Verwendung der sogenannten Hochkapazitätsdisketten eingerichtet. Damit sind die 720-KByte-Disketten, also das übliche 3,5-Zoll-Format gemeint. Beim Lesen und Beschreiben treten dann keinerlei Probleme auf. Lediglich das Formatieren muß aus den beschriebenen Gründen unter PC_DFMT.COM geschehen. Bei DOS-Versionen unter 3.2 ist die Sache etwas komplizierter. Hier muß ein spezieller Treiber installiert werden, der mitgeliefert wurde. Fügen Sie in die CONFIG.SYS-Datei folgende Zeile ein:

```
DEVICE=PC_DDRVR.SYS
```

Damit wird der Treiber beim nächsten Booten in das System eingebunden. Dieser Treiber ermöglicht es, die 720-KByte-Disketten unter speziellen Laufwerksbezeichnungen anzusprechen, obwohl die Disketten physikalisch im gleichen Gerät stecken. Welche Kennungen zu verwenden sind, entnehmen Sie bitte der folgenden Tabelle:

Vorhandene Geräte	Disk in A ansprechbar als	Disk in B ansprechbar als
1 Floppy	B	-
2 Floppies	C	D
1 Floppy, Harddisk	D *	-
2 Floppies, Harddisk	D *	E *



Die mit einem Stern markierten Angaben sind mit Vorsicht zu genießen. Sie gelten nur, wenn PC-DITTO von Ihrer Festplatte nur die Partition C kennt. Gibt es mehr Partitionen, verschieben sich die Laufwerksbezeichnungen entsprechend.

Wenn Sie den MS-DOS-Befehl ASSIGN A=x bzw. bei zwei Floppies ASSIGN A=x, B=y ausführen lassen, wobei Sie für x und y die Laufwerkskennungen der neuen Scheinlaufwerke eingeben, so können Sie die 720-KByte-Disketten auch unter den herkömmlichen Bezeichnungen A bzw. B ansprechen.

Natürlich können Sie auch Ihre Festplatte unter DOS benutzen. Zunächst muß diese mit PC_DMENU.PRG beim DITTO angemeldet worden sein. Ist dies der Fall, so meldet der Emulator nach dem Start, wie viele Partitionen er erkennt. Das heißt aber noch nicht, daß auch MS-DOS alle diese Partitionen erkennt. Zunächst können Sie nur Laufwerk C: benutzen. Wollen Sie auch die anderen Partitionen benutzen, müssen Sie zusätzlich den Harddisk-Treiber PC_DHD.SYS in das System einbinden. Dazu brauchen Sie nur folgende Zeile in die Datei CONFIG.SYS auf Ihrer DOS-Systemdiskette einfügen:

device=PC_DHD.SYS

Beim nächsten Booten erkennt auch MS-DOS die restlichen Partitionen.

In manchen Fällen kann es vorkommen, daß der DITTO zuviele Partitionen auf der Harddisk findet (obwohl nur die von GEM bekannten vorhanden sind) oder beim Zugriff auf die Platte undefinierbare Reaktionen des Rechners stattfinden. Dieses Phänomen ist vor allem bei der Benutzung der Atari-Festplatte SH204 zu beobachten. Abhilfe bietet das Programm AHDFIX.PRG, das Sie unter TOS einmal starten müssen. Es räumt den Boot-Sektor der Festplatte auf. Dort vorhandener "Speichermüll" kann das MS-DOS unter Umständen verwirren. AHDFIX.PRG muß nach jeder Benutzung von HDX.PRG und HINSTALL.PRG ausgeführt werden.

Sie können die Festplatte wie jede Diskette lesen und beschreiben. Jede Datei des TOS ist auch für MS-DOS zugreifbar und umgekehrt.

Anders als unter TOS können Sie die Festplatte unter MS-DOS ganz normal formatieren. Das sollten Sie auf keinen Fall tun! Für die Formatierung der Atari-Festplatten sollte nur das HDX.PRG benutzt werden.

Die Benutzung der Maus ist unter MS-DOS im Grunde nicht anders als unter TOS. Aber: Nicht jedes Programm ist auf die Verwendung einer Maus eingerichtet. Seien Sie deshalb nicht enttäuscht, wenn sich bei vielen Programmen der Mauszeiger nicht blicken läßt. Außerdem muß vor der Benutzung der Maus der zugehörige Treiber gestartet sein. Das geht recht einfach, wenn Sie in der Datei AUTOEXEC.BAT folgende Zeile anfügen:

PC_DMOUS.COM



Beim Booten zeigt der PC-DITTO eine Meldung an, daß die Diskette im Laufwerk A keine Systemdiskette bzw. defekt sei.



Es könnte sein, daß Sie eine Diskette mit ST-Format statt der Systemdiskette in das Laufwerk gesteckt haben. Auch wenn eine solche ST-formatierte Diskette das DOS enthalten sollte, kann es nicht davon gelesen werden. Falls Sie MS-DOS-Versionen älter als 3.2 verwenden, sollten Sie beachten, daß diese nicht von 720-KByte-Disketten gebootet werden können. Schließlich kann es noch sein, daß die Diskette wirklich kein DOS enthält oder fehlerhaft ist.



Beim ersten Zugriff auf eine neu eingelegte Diskette (z.B. nach B: Return) erscheint die Fehlermeldung "Allgemeiner Lesefehler".

- ✓ Entweder ist die Diskette defekt, nicht formatiert oder es handelt sich um eine ST-formatierte Diskette. Diese kann der DITTO nicht lesen.
- ➡ Nach dem Booten sind die deutschen Umlaute auf der Tastatur nicht erreichbar.
- ✓ Der PC ist nicht auf die deutsche Tastaturbelegung umgeschaltet worden. Versuchen Sie zunächst bitte, diesen Fehler durch Drücken von `[Alt]-[Ctrl]-[F2]` zu beheben. Falls das nicht hilft, müssen Sie im Programm PC_DMENÜ auf die deutsche Tastaturbelegung umschalten.
- ➡ Nach dem Starten von MS-DOS werden ohne menschliches Zutun Tastendrücke wiederholt, die jede Eingabe unmöglich machen.
- ✓ Hier stimmt etwas mit der Tastaturbehandlung nicht. PC-DITTO hat bereits eingebaute Treiber für die gängigen Standard-Tastaturen, so daß das Programm KEYB bzw. KEYBGR überflüssig wird. Streichen Sie es aus der Datei AUTOEXEC.BAT, und starten Sie das System neu. Der Fehler müßte damit beseitigt sein.

Sehr einfach ist der Datentransfer vom ST in den PC-DITTO. Egal, ob es sich dabei um Programme oder Dateien handelt, Sie brauchen diese Files nur auf eine Diskette im PC-Format zu kopieren. Von dieser kann der MS-DOS-Rechner sie dann lesen. Der umgekehrte Weg verläuft genauso: Daten, die der PC geschrieben hat, können Sie mit dem ST weiterverarbeiten. Auf diese Weise können Sie auch die mitgelieferten PC-Programme von der Original-Diskette des Emulators in den PC bringen.

Zwei Programme, die zum Lieferumfang des PC-DITTO gehören, sind SPEEDY3 und QWICKEYS. Hierbei handelt es sich in beiden Fällen um sogenannte Public-Domain-Software, die aber für den Betrieb des PC-DITTO fast unverzichtbar ist. SPEEDY3 beschleunigt die Bildschirmausgabe in einigen Fällen, was gerade beim PC-DITTO sehr nötig ist. QWICKEYS sorgt für eine schnellere Bedienung von Tastatur und Cursor-Steuerung. Sie sollten beide Programme durch einen Aufruf in der AUTOEXEC.BAT bei jedem Systemstart automatisch aufrufen lassen. Schreiben Sie dazu die beiden Zeilen

```
SPEEDY3 /H  
QWICKEYS
```

in diese Batch-Datei.

Wie Sie an der Bildschirmausgabe selbst sehr schnell merken werden, ist der PC-DITTO nicht gerade ein Sprinter. Allgemein wird ein Norton-Faktor von

ca. 0,3 angegeben - das bedeutet, daß der Original-PC von IBM (der auch schon als recht langsam gilt) etwa 3mal schneller arbeitet.

Verlassen wird der Emulator entweder durch Ausschalten des Rechners oder durch gleichzeitiges Drücken der Tasten **Ctrl**-**Alternate**-**Delete**. Es erscheint wieder die Einschaltmeldung des PC-DITTO. Mit **Return** können Sie das DOS neu laden, per **Esc** kommen Sie zurück zum Desktop.

Für Neulinge: Kleiner MS-DOS-Exkurs

Wenn Sie das erste Mal mit MS-DOS zu tun haben, werden Sie schnell merken, daß unter diesem Betriebssystem nichts so komfortabel läuft, wie Sie das vom ST her gewohnt sind. Zunächst: Alle Eingaben und Kommandos werden über Tastatur getätigt, auch das Kopieren. Angenehm für uns ist, daß die Dateinamen und das hierarchische File-System auf dem ST mit denen des MS-DOS identisch sind. Auch sonst werden Sie einige Ähnlichkeiten entdecken.

Wenn Sie MS-DOS geladen haben, erscheint in der aktuellen Zeile (das ist die, in der der Cursor steht) das sogenannte System-Prompt. Das sieht ungefähr so aus:

```
A:>_
```

Das A gibt an, daß Laufwerk A das gerade aktuelle Laufwerk ist. (Auf dem Desktop würde das bedeuten, daß Sie ein Fenster mit dem Hauptverzeichnis von Laufwerk A geöffnet haben.).

Um das Inhaltsverzeichnis dieser Diskette anzeigen zu lassen, tippen Sie bitte

```
DIR Return
```

ein. DIR ist das Kürzel für Directory, dem englischen Wort für Inhaltsverzeichnis. Daraufhin werden die Files der Diskette wie beim ST im Textmodus angezeigt. Ordner werden durch ein angehängtes **DIR** gekennzeichnet.

Um in einen Ordner zu gelangen, tippen Sie

```
CD ordnername Return
```

Um dessen Inhalt zu sehen, müssen Sie dann nur wieder DIR eingeben. Im Ordnerverzeichnis tauchen immer die beiden Ordner "." und ".." auf. Das hat mehr technische Gründe; das DOS braucht sie für die Verwaltung der Verzeichnisstruktur. "." steht dabei für das aktuelle Verzeichnis, ".." für das übergeordnete. Um aus dem Ordner wieder in das übergeordnete Verzeichnis zu kommen, können Sie den Befehl

CD ..

verwenden.

Wenn Sie auf Disketten in einem anderen Laufwerk mit der Kennung x zugreifen wollen, tippen Sie einfach

X:

Für x müssen Sie dabei natürlich die richtige Kennung des Laufwerks eintippen, also beispielsweise A für das erste Laufwerk.

Programme tragen unter DOS die Extension COM oder EXE. Dateien mit den Endungen PRG, TOS oder TTP können unter DOS nicht gestartet werden.

Um ein Programm zu starten, tippen Sie einfach dessen Namen ein, aber ohne Extension. Wenn Sie also ein Programm namens TEXT.EXE starten wollen, müssen Sie nur

TEXT

eintippen.

Zum Löschen einer Datei dient der Befehl DELETE. Um zum Beispiel die Datei TEXT.TXT zu löschen, müßten Sie folgendes eintippen:

DEL TEXT.TXT

Tippen Sie,, um einen Ordner mit Namen TEXTE zu erzeugen, diesen Befehl ein:

MD TEXTE

Unter MS-DOS können nur leere Ordner gelöscht werden. Sie müssen also vorher gegebenenfalls mit "CD ordnername" in den Ordner hineingehen und dort alle Dateien mit DEL löschen, um dann nach CD.. das Kommando

RMDIR ordnername

zum Löschen geben zu können.

9.4.2 Der Aladin-Macintosh-Emulator

Hier handelt es sich um eine Mischung aus soft- und hardwaremäßigem Emulator. Mit seiner Hilfe wird es möglich, die ROMs aus dem Apple Macintosh an einen ST anzuschließen. Durch diese ROMs und das Emulationsprogramm ist der ST dann in der Lage, das Mac-Betriebssystem (den Finder) zu laden und zu verarbeiten. Wohlgemerkt: Auch der Aladin enthält kein Betriebssystem, dieses müssen Sie sich besorgen. Der Aladin besteht aus einem Steckmodul und einigen Programmen. Die Programme haben einzig und allein die Aufgabe, die Hardware des Apple Macintosh mit den Mitteln des ST nachzubilden. Wir reden hier übrigens von der neuesten Version (3.0) des Aladin. Ältere Emulator-Versionen kennen eventuell die eine oder andere Funktion, die hier beschrieben wird, nicht. Kuriosität am Rande: Der Aladin-Emulator nutzt die Hardware des ST so gut aus, daß der emulierte Mac ca. 20 % schneller als das Original ist. Außerdem steht Ihnen eine größere Bildschirmfläche zur Verfügung. Allerdings funktioniert der Aladin-Emulator nur mit dem MonochromBildschirm SM-124.

Im Gegensatz zu MS-DOS wird der Finder nicht frei verkauft. Eine legale Kopie erhalten Sie also nur, wenn Sie bereits einen Mac besitzen und dessen Finder auf das Aladin-Format übertragen. Ähnliches gilt für die ROMs. Im freien Handel sind diese so gut wie überhaupt nicht erhältlich, also müßten Sie sie schon aus Ihrem Mac aus und in den Aladin einbauen. Die Beschaffungsaktion wird dadurch erschwert, daß Apple die Verbreitung des Aladin durch Klagen zu verhindern suchte. Erst seitdem die Rechte an Aladin auf eine niederländische Firma übergegangen sind, haben diese Machenschaften aufgehört.

In das Aladin-Steckmodul müssen Sie die Apple-Betriebssystem-ROMs einsetzen und zwar so, daß die kleinen Nasen auf der Oberfläche der Chips nach oben zeigen (die Seite mit der Steckleiste liegt rechts). Das Modul wird in den ST-Modulschacht gesteckt. Wichtig: Sie benötigen die sogenannten 64-K-ROMs. Die 128-K-ROMs (für die neueren Mac-Versionen) sind ungeeignet.

Um auf einem Mac eine Systemdiskette zu erstellen, müssen Sie im Normalfall drei Dateien auf eine neue Diskette kopieren. Es sind dies der Finder, die Druckeranpassung ImageWriter und das Systemdokument System. Sie können auch einfach die Systemdiskette kopieren, mit der Sie den Mac starten. Diese Diskette ist logischerweise eine reine Macintosh-Diskette, d.h. weder TOS noch Aladin können sie lesen. Bevor Sie sie also mit Aladin benutzen können, muß sie konvertiert werden.

Auch dieser Emulator braucht sein spezielles Diskettenformat. Leider ist dies aber so vom ST- und auch vom Macintosh-Format verschieden, daß solche

Disketten nur unter dem Aladin beschrieben werden können. Einzige Ausnahme: die Übertragungsprogramme EMPF350.TOS und EMPF400.TOS. Diese ermöglichen - zusammen mit den Gegenstücken für den Mac - die Übertragung von Daten und Programmen von einem echten Mac auf den ST. Die so erzeugten Disketten können vom Aladin bearbeitet werden, nicht aber unter GEM. Zur Übertragung brauchen Sie neben den genannten Programmen auch noch ein Kabel, das Sie sich anhand der Anleitung zusammenlöten müssen. Bevor Sie loslegen können, müssen Sie auf jeden Fall die Systemdiskette auf das Atari-Format übertragen. Die Aladin-Disketten sind alle so gestaltet, daß Sie - falls sie zum Booten benutzt werden sollten - einen Hinweis darauf geben, daß sie nicht für den ST geeignet sind. Bevor Sie also loslegen können, müssen Sie die Systemdiskette des Mac mittels der Rechnerkopplung und der genannten Programme auf den ST übertragen.

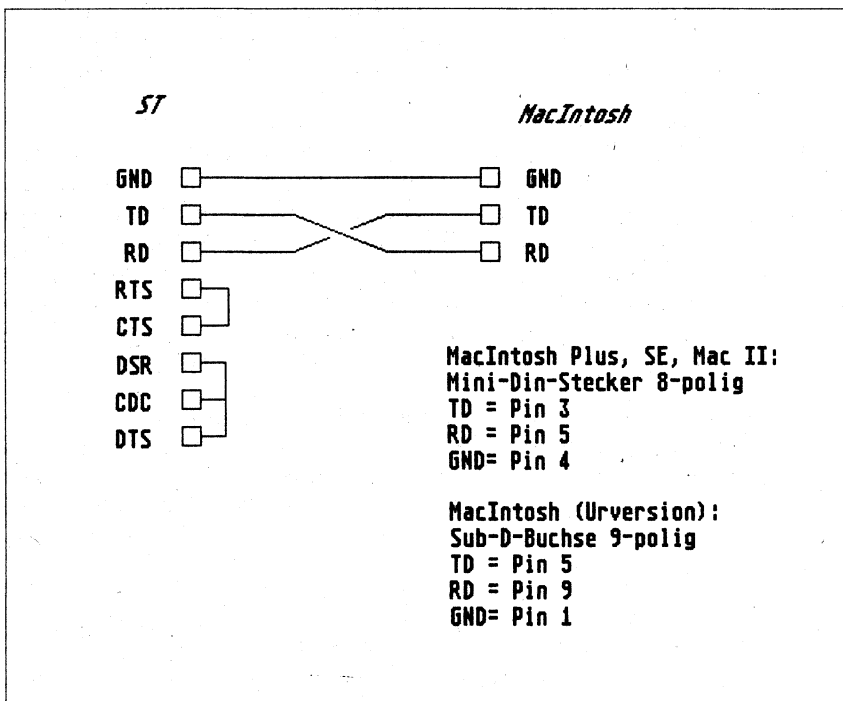


Abb. 9.9: Das Übertragungskabel zwischen MAC und ST

Zunächst müssen Sie sich das angesprochene Kabel bauen, wie es die obenstehende Abbildung zeigt. Verbinden Sie die beiden Computer. Am ST wird die

ses Kabel an der seriellen Schnittstelle ("Modem") angeschlossen. Auf der Mac-Diskette des Aladin-Emulators gibt es ein Programm namens "350-K-Diskette erstellen". Dieses brauchen Sie, wenn Ihr ST mit einem einseitigen Laufwerk ausgestattet ist. Die normalen Mac-Disketten haben eine Speicherkapazität von 400 KByte, die natürlich nicht auf eine einseitige Diskette passen. Nehmen Sie eine leere, formatierte Mac-Diskette, und starten Sie das Programm. Nachdem die Diskette bearbeitet wurde, kann sie nur noch 350 KByte Daten aufnehmen. Auf dieser Diskette können Sie dann alle Dateien und Programme kopieren, die Sie auf den ST übertragen möchten.

Wenn Sie einen ST mit doppelseitigem Laufwerk besitzen, brauchen Sie nur das Kabel. Die Daten können dann ohne weitere Vorbereitung gesendet werden.

Auf der Mac-Diskette des Aladin-Emulators gibt es zwei Programme "350-K-Diskette senden" und "400-K-Diskette senden". Auf der ST-Diskette befinden sich zwei Programme EMPF350K.TOS und EMPF400K.TOS. Diese Programme arbeiten jeweils gleich, der einzige Unterschied besteht in der Größe der Diskettenformate, die damit übertragen werden. Starten Sie bitte zuerst das Empfangsprogramm (EMPFxxx.TOS) auf dem ST, dann das Sendeprogramm auf dem Mac "xxx-K-Diskette senden". Sie werden dann auf beiden Geräten aufgefordert, jeweils Disketten einzulegen. Nach ca. fünf Minuten sind die Daten übertragen. Sie können die so erstellte Kopie der Macintosh-Diskette sofort mit Aladin verwenden.

Sollten Sie mit einem doppelseitigen Laufwerk arbeiten, hat die so erstellte Diskette nur 400 KByte Kapazität, obwohl normale Aladin-Disketten 710 KByte speichern können. Sie sollten die Dateien daher einfach auf eine mit Aladin formatierte Diskette umkopieren, um die hohe Kapazität auch ausnutzen zu können.

Auf diese Art können Sie im Prinzip jedes Programm übertragen, bis auf einige Fälle, in denen das Programm an den Aladin-Emulator angepaßt werden muß. Dazu wird ein Programm "Anpassung" mitgeliefert. Es hat die Aufgabe, an Programmen leichte Korrekturen (sog. Patches) anzubringen. Beachten Sie, daß nur solche Programme auf dem Aladin laufen, die nicht die neuen 128-K-ROMs benötigen.

Wenden Sie das Anpassungsprogramm niemals auf Original-Software an, denn diese könnte dadurch unwiderruflich zerstört werden. Lassen Sie "Anpassung" immer nur auf Kopien des Programms arbeiten.

Sehen Sie sich nun die Anpassungsdateien an, die zum Programm "Anpassung" gehören. Wenn Sie darunter Namen von Software wiedererken-

nen, die Sie einsetzen wollen, müssen Sie diese anpassen. Unter Umständen gibt es verschiedene Versionen einer Software. Welche Version Sie besitzen, können Sie mit "Anpassung" feststellen, denn es gibt dort einen speziellen Menüpunkt "Version zeigen". Die Anpassung läuft ganz automatisch ab, wenn Sie den entsprechenden Menüpunkt anwählen.

Schließlich gibt es noch die Janus-Disketten. Wie in der alten Mythologie haben auch die Janus-Disketten zwei Gesichter - eine Vorder- und eine Rückseite. Die eine Seite dieser Disketten trägt ein GEM-Format, sie kann also wie eine ganz normale einseitige Diskette unter TOS benutzt werden. Die Rückseite ist dagegen im Aladin-Format initialisiert worden. Legt man eine solche Diskette ein, während Aladin läuft, wird nur die Rückseite beachtet, die wie eine normale Aladin-Diskette behandelt wird. Das funktioniert natürlich nur mit zweiseitigen Laufwerken, auf einseitigen Geräten kann nur die TOS-Seite genutzt werden. Auch das Original der Aladin-Startdiskette ist eine solche Janus-Diskette. Achtung! Rückseite bedeutet nicht, daß Sie die Diskette verkehrt herum in das Laufwerk schieben müssen.

Sicher möchten Sie auch Sicherungskopien von den Aladin-Originalen machen. Die Startdiskette (mit der Kennzeichnung ST; es handelt sich dabei um eine Janusdiskette) können Sie ganz normal mit dem Desktop 1:1 kopieren, wenn Sie ein Disketten-Icon auf ein anderes schieben. Dabei wird allerdings nur die TOS-Seite der Janus-Diskette kopiert, die Rückseite mit dem Aladin-Format muß später beim Betrieb des Emulators auf eine andere Aladin-Diskette gesichert werden.

Die zweite Diskette aus dem Paket mit der Kennzeichnung MAC ist im Original-Mac-Format erstellt worden. Weder TOS noch Aladin können mit dieser Diskette etwas anfangen, sie muß auf einem echten Macintosh kopiert werden.

Vor dem ersten Start des Aladin sollten Sie das Konfigurationsprogramm aufrufen (mittels C nach dem Booten von der Aladin-Boot-Diskette). Die Einstellungen hier sind alle selbstdokumentierend. Sie sollten sie nach den jeweiligen Gegebenheiten Ihres ST wählen.

Falls Sie Aladin nicht auf einer Festplatte installieren, richten Sie eine so große RAM-Disk (unter Aladin Super-Disk genannt) ein, daß die Systemdateien darin Platz finden. Damit können Sie später die Arbeit mit dem Mac erheblich beschleunigen.

Jetzt kann es endlich losgehen. Legen Sie die Aladin-Startdiskette in das Laufwerk A, und schalten Sie den Rechner ein. Der ST beginnt zu booten und nach ein paar Sekunden erscheint das Titelbild auf dem Bildschirm. Auf einen Druck auf die Leertaste hin wird der eigentliche Emulator gestartet. Mit

Help beginnt die Emulation, auf dem Schirm erscheint ein Mac-Icon. Sobald Ihnen der Macintosh zublinzelt, können Sie die Systemdiskette einlegen. Der Mac (jajwohl der Mac, nicht mehr der ST) bootet jetzt sein Betriebssystem. Nach einigen Sekunden erscheint das Desktop des Macintosh (verblüffend die Ähnlichkeit, nicht?).

Wenn Sie mit einer Super-Disk arbeiten, sollten Sie zuerst die Systemdateien auf die Super-Disk kopieren. Ist dies getan, können Sie die System-Diskette auswerfen lassen und den Menüpunkt "Ausschalten" wählen. Dies veranlaßt den Mac zum neuen Booten - diesmal aber von der Super-Disk, was in Nullkomma-nichts geschehen ist. Der Vorteil dabei: Während der normalen Arbeit mit dem Mac kommt es häufiger vor, daß Teile des Finders von der Diskette nachgeladen werden müssen. Dies erfolgt bei einer RAM-Disk natürlich sehr viel schneller. Festplattenbenutzer können sich diese Prozedur sparen.

Die Übertragung von Daten zwischen ST und Mac klappt wahrscheinlich überhaupt nicht, denn die Diskettenformate sind absolut inkompatibel. Sie können also mit dem ST auf keinen Fall die Mac-Disketten beschreiben. Der einzige Weg, um Daten zu übertragen, wäre wohl die Verwendung eines Terminalprogramms. Der Mac wird mit dem ST über das selbstgebaute Kabel gekoppelt und die Daten dann wie über eine Telefonverbindung überspielt.

Statt dessen können Sie ST-Daten aber mit Aladin verarbeiten! Dazu gibt es auf der Rückseite der Startdiskette ein Programm mit Namen "GemLoad". Es dient dazu, TOS-Dateien zu lesen und auf Aladin-Disketten zu kopieren. Sie können es starten und werden dann aufgefordert, eine Diskette einzulegen. Achten Sie bitte darauf, daß der Schreibschuttschieber geschlossen sein muß, damit der Aladin die Diskette auch erkennt!

Sie können dann die Dateien auswählen, die konvertiert werden sollen. Dabei haben Sie auch die Auswahl unter mehreren Formaten, in denen die Daten (z.B. für Textverarbeitungsprogramme) abgespeichert werden sollen.

Um eine Aladin-Diskette zu erhalten, lassen Sie einfach eine neue Diskette unter Aladin (also mit der Funktion im Mac-Desktop) initialisieren. Das geht so:

1. Legen Sie die neue Diskette in Laufwerk A ein. Der Mac versucht nun, das Inhaltsverzeichnis dieser Diskette automatisch anzuzeigen. Weil die Diskette aber leer ist, mißlingt das.
2. Der Mac fragt, ob er die Diskette initialisieren soll. Falls Sie die Diskette wirklich formatieren lassen wollen, antworten Sie bitte mit ja.

3. Der Mac formatiert die Diskette im Aladin-Format.

Die Schreibschutzschieber der Aladin-Disketten müssen immer geschlossen sein, weil der Emulator sonst den Diskettenwechsel nicht erkennen kann. Ein Schreibschutz ist daher nur softwaremäßig möglich.

Seit der Version 3.0 ist auch Aladin auf die Benutzung der Festplatten eingerichtet. Mein Tip: Wenn es nur irgendwie möglich ist, installieren Sie Ihren Aladin auf der Festplatte. Es gibt keine bequemere Art der Arbeit mit dem Mac. Die Installation ist dabei recht simpel. Sie müssen allerdings eine ganze Partition komplett für den Aladin reservieren. Sie können diese Partition nicht gleichzeitig für TOS und Aladin nutzen, weil darauf ein neues Format aufgebracht wird. Löschen Sie also bitte vorher alle TOS-Dateien, die auf dieser Partition stehen. Dann müssen Sie im Aladin-Emulator das Programm "Festplatte installieren" starten. Mit diesem Programm können Sie dem Emulator mitteilen, welche Partition von welcher Festplatte Sie für das Macintosh-Betriebssystem nutzen möchten. Achtung! Sie müssen hier genau die Partition umstellen, die Sie vorher für Aladin freigemacht haben. Im Normalfall entspricht Partition 0 dem Laufwerk C, Partition 1 dem Laufwerk D usw. Wenn Sie das Programm durch "Setzen" beenden, werden die angewählten Partitionen für Aladin konvertiert. Genauer: Die Festplatte wird scheinbar wie eine normale Diskette initialisiert. Sie können dem dabei entstehenden neuen Mac-Laufwerk einen Namen geben. Ab sofort können Sie dieses Laufwerk wie jede normale Mac-Diskette behandeln. Wenn Sie die Systemdateien auf die Festplatte kopieren, können Sie das System sogar von der Festplatte booten (sehr empfehlenswert).

Sie können den Aladin nicht mit ausgeschalteten Diskettenlaufwerken betreiben, wie das unter TOS möglich ist. Beim Betrieb des Aladin müssen immer alle angeschlossenen Diskettenlaufwerke eingeschaltet sein. Umgekehrt können Sie aber ganz ohne Festplatte arbeiten, auch wenn diese als Systemlaufwerk dient. Sie müssen dann nur von einer ganz normalen Systemdiskette booten, während das Festplattenlaufwerk ausgeschaltet bleibt.

Kleine Macintosh-Kunde für Neulinge

Der Mac wird wie der ST vor allem über die Maus gesteuert. Branchenkenner sagen, daß das GEM sein Vorbild in der Benutzeroberfläche des Mac hatte, was bei genauem Hinsehen auch nicht verwundert. Vom Aussehen her werden Sie kaum Unterschiede feststellen.

Beim Mac müssen Sie kaum noch Fenster öffnen. Das geht ganz automatisch. Wenn Sie eine Diskette in das Diskettenlaufwerk einlegen, wird der Mac in

den meisten Fällen sofort versuchen, das Inhaltsverzeichnis anzuzeigen. Tut er das nicht, so können Sie das erscheinende Icon für diese Diskette doppelklicken.

Das Starten von Programmen und Öffnen von Ordnern funktioniert wie beim ST mit Doppelklick. Geschlossen wird natürlich mit der kleinen Schließbox in der linken oberen Ecke des Fensters. Auch sonst entspricht die Bedienung der Fenster weitgehend der unter GEM.

Icons können auch verschoben werden, und diesmal nicht nur zum Kopieren. Kopiert wird zwar auch wie auf dem ST - also durch Verschieben auf einen Ordner oder das Symbol einer anderen Diskette. Wird eine Datei aber innerhalb des Fensters verschoben, so wechselt Sie lediglich den Ort. Sie können also (zumindest bei der Anzeige als Icon) die Reihenfolge der Icons selbst bestimmen.

Das Wichtigste bei der Arbeit mit einem Mac: Man kann eine Diskette nicht beliebig aus dem Laufwerk nehmen. Der Original-Mac besitzt gar keinen Auswurfkopf an seinem Laufwerk. Er wirft die Disketten auf Befehl vom Desktop motorisch aus. Da der ST über diese Möglichkeit nicht verfügt, dürfen Sie den Auswurfkopf nur dann betätigen, wenn Sie vorher den Menüpunkt "Auswerfen" betätigt haben. Tun Sie das trotzdem, so wird der Mac durch Piepsen und Blinken der Laufwerks-LED auf den Fehler aufmerksam machen.

Menüs werden fast wie beim ST bedient - aber nur fast. Um ein Menü herunterklappen zu lassen, fahren Sie bitte auf dessen Überschrift und halten dann die Maustaste fest. Solange die Taste gedrückt bleibt, ist das Menü sichtbar, wenn Sie die Taste loslassen, verschwindet es. Um eine Funktion anzuwählen, fahren Sie den Pfeil bitte bei gedrückter Taste auf die gewünschte Zeile und lassen dort die Maustaste los.

9.4.3 MINIX

Irgendwo in der reichlich erscheinenden Fachliteratur wird Ihnen bestimmt schon der Begriff UNIX begegnet sein. MINIX ist ein nachprogrammiertes UNIX für den ST.

UNIX ist ein sogenanntes Mehrbenutzer-Betriebssystem, das es erlaubt, einen Computer zugleich von mehreren Anwendern benutzen zu lassen. UNIX ist schon 20 Jahre alt. Trotzdem ist es kein alter Hut, sondern nach Meinung vieler Experten das Betriebssystem der Zukunft. Ursprünglich für den Betrieb auf Großrechnern gedacht, wird es in letzter Zeit immer mehr auf Hochleistungs-

PCs übertragen. UNIX war in den ersten Jahren nach seinem Erscheinen kein kommerzielles Produkt (im Gegensatz zu heute), sondern ein reines Forschungsobjekt. Das führte dazu, daß der Hersteller (AT & T, der Telekommunikationsriege aus den USA) UNIX für ein Butterbrot an Universitäten abgab - inklusive der Quelltexte. Für die Studenten war dies ein gefundenes Fressen, konnten sie doch den trockenen Stoff aus den Vorlesungen über Betriebssysteme gleich an UNIX ausprobieren. Aus den Experimenten der Studenten entstanden zugleich viele bedeutende Erweiterungen von UNIX. Das änderte sich aber, als UNIX nach und nach auch einen kommerziellen Wert bekam. AT & T änderte seine Lizenzbedingungen, so daß die Quelltexte nicht mehr zur Verfügung standen. Gleichzeitig verloren die Studenten ihr Lehrobjekt.

Professor Andrew S. Tanenbaum von der Freien Universität Amsterdam nahm dies vor wenigen Jahren zum Anlaß, MINIX zu programmieren - einen Nachbau des Original-UNIX, der sich in allen wichtigen Punkten genau wie das Vorbild verhält, aber eben kein echtes UNIX ist. MINIX entstand zunächst für den IBM-PC/XT, im letzten Jahr wurde es auch auf den ST übertragen.

Sowohl die Quelltexte als auch das Betriebssystem selbst sind für jedermann zugänglich, so daß der universitäre Lehrbetrieb wieder ein Experimentalobjekt hat. Zu MINIX gibt es ein Buch (in Englisch), in dem alle Quelltexte abgedruckt sind: Andrew S. Tanenbaum; Operating Systems: Design and Implementation, Prentice-Hall, 1987, ISBN 0-13-637406-9.

Was den Studenten recht ist, sollte uns billig sein. Sie müssen nicht gleich das Betriebssystem ändern oder verbessern. MINIX kann Ihnen auch helfen, wenn Sie sich auf UNIX vorbereiten und zu diesem Zweck am häuslichen ST üben möchten. Eines sei aber auch vorweg gesagt: Es gibt so gut wie keine Software für MINIX käuflich zu erwerben; Sie sind also auf das Selbst-Programmieren angewiesen. Zum Umfang von MINIX gehören ein Editor und der in der UNIX-Welt unverzichtbare C-Compiler.

Außerdem gibt es eine Menge von nützlichen Kommandos, die für sich schon eine ganze Batterie von Software-Paketen überflüssig machen.

MINIX ist keine Public-Domain-Software! Das oben genannte Buch von Andrew S. Tanenbaum finden Sie auch in vielen Bibliotheken. Wenn Sie also nur die Sourcen ansehen möchten, reicht das schon. Wenn Sie MINIX auch einsetzen möchten, können Sie das Paket käuflich erwerben.

Was gibt es sonst noch zu MINIX zu sagen? Sehr viel - so viel, daß es hier keinen Platz finden kann. Wer sich näher mit diesem Betriebssystem beschäftigen will, sollte sich geeignete Fachliteratur beschaffen.

9.5 Viren

Viren sind Programme, die sich selbsttätig und unbemerkt vom Anwender auf andere Disketten kopieren oder - in Rechnernetzen - in alle angeschlossenen Computer verbreiten können. Der Name Virus deutet bereits an, daß sie unseren Rechner "krank" machen können. Nach nicht zu bestimmender Zeit werden solche Viren plötzlich aktiv und zerstören zum Beispiel Daten. Die Folgen eines Virus sind unterschiedlich. Manche Programme begnügen sich mit einer schlichten Meldung auf dem Bildschirm, garstige Exemplare formatieren mal eben ohne Vorwarnung ihre Diskette. Das Fatale an Viren ist, daß man ihr Vorhandensein im eigenen System im Normalfall erst dann bemerkt, wenn es schon zu spät ist.

Es gibt zwei Arten von Viren. Die einen sind sogenannte Boot-Viren, die über den Boot-Sektor in das System gelangen; die anderen sind sogenannte Programmviren. Um die Arbeitsweise solcher Boot-Viren verstehen zu können, muß man zunächst wissen, was beim Booten eines Atari ST abläuft. Bekanntlich versucht der ST nach dem Einschalten zunächst, den Sektor 0 der Diskette in Laufwerk A zu lesen. Man nennt diesen Sektor auch Boot-Sektor. Er enthält im Normalfall nur die Informationen über die Kapazität und den logischen Aufbau der Diskette. Außerdem hat dieser Sektor noch eine andere Aufgabe. Sollen andere Betriebssysteme als das im ROM (z.B. eine Diskettenversion) geladen werden, so hat der Boot-Sektor eine besondere Form. Neben den Formatierungsinformationen enthält er dann auch noch ein kleines Assembler-Programm (ca. 0,5 KByte), den Lader. Falls der Boot-Sektor (und das ist wichtig) als Prüfsumme den hexadezimalen Wert \$1234 aufweist, wird dieser Lader beim Systemstart in den Speicher geladen und ausgeführt. Das ist genau die Stelle, an der sich Viren in das Betriebssystem einklinken. Im Normalfall hat ein solcher Lader nur die Aufgabe, das auf der Diskette vorhandene Betriebssystem zu laden und zu starten. Viren dagegen tarnen sich eben als solche Lader und haben nach dem Start durch die Boot-Sequenz nichts anderes zu tun, als sich resident im Speicher zu installieren (quasi wie ein Accessory). Danach geben sie die Kontrolle wieder an die Boot-Sequenz zurück, und das System macht weiter als wäre nichts geschehen.

Der Virus ist nun also in unser System eingedrungen. Wie aber kommt er auf andere Disketten? Dazu muß man eine weitere Eigenheit des TOS kennen. Zu vielen wichtigen Routinen des TOS gibt es sogenannte Vektoren. Das sind Speicherzellen in den unteren 4 KByte des RAM, die jeweils auf eine Routine zeigen. Letztere werden nämlich nie direkt, sondern immer nur über diese Zeiger angesprungen. Da diese Zeiger im RAM liegen, ist es natürlich ein leichtes, ihre Werte zu verändern. Viren verbiegen nun die Vektoren von Betriebssystem-Routinen derart, daß sie auf das Virusprogramm zeigen. Statt der

gewünschten Betriebssystem-Routine wird also der Virus angesprungen, und erst dieser ruft das TOS auf. Mit dieser Technik ist es nun möglich, den Virus vor eine Betriebssystem-Funktion für den Diskettenzugriff zu schalten, z.B. die Funktion zum Lesen einer Datei. Wenn in dieser Situation versucht wird, eine Datei zu lesen, wird damit automatisch der Virus angesprungen und die im Laufwerk befindliche Diskette infiziert.

Etwas anders ist das mit Programmviren. Diese auf dem ST selteneren Viren hängen sich an bestehende Programme an - z.B. Ihre Textverarbeitung. Dabei wird in den meisten Fällen das Anwenderprogramm leicht beschädigt, was aber nur selten auffällt, bevor es zu spät ist. Solche infizierten Programme können nur noch aus dem Verkehr gezogen werden. Vorsicht! Programmviren können sich auf vielfältige Weise tarnen, so daß wir dazu an dieser Stelle nichts näheres sagen können.

Safer Copy ist angesagt. Wer oft mit fremden Disketten arbeitet, sollte aufpassen, von wem diese stammen. Auf jeden Fall sollte dabei der Schreibschutz aktiviert werden. Damit kann man alle Viren blockieren. Wichtiger ist vielleicht die Erkenntnis, daß viele der bisher bekannten Viren auf dem ST aus dem Boot-Sektor stammen. Deshalb können diese auch nur dann aktiv werden, wenn mit einer verseuchten Diskette gebootet wird. Also goldene Regel Nr. 1:

Niemals mit fremden Disketten booten!

Wenn dies unumgänglich sein sollte, gilt folgendes: Möglichst wenige Geräte einschalten (vor allem die Harddisk ausschalten, abmelden reicht nicht), und vor der Verwendung eigener Disketten den Rechner ca. zehn Sekunden lang ausschalten. Die meisten Viren überstehen einen Reset, also wirklich die Stromzufuhr abschalten! Schließlich sollten Sie alle Ihre Disketten auf Boot-Sektor-Viren überprüfen und diese ggf. entfernen. Dazu dient das unten abgedruckte Programm (in jeweils einer Version für GFA-BASIC und Omikron-bzw. ST-BASIC). Die Funktionsweise ist recht einfach. Der Boot-Sektor wird auf Ausführbarkeit untersucht, d.h. ob er die magische Prüfsumme \$1234 besitzt. Ist dies gegeben, so zeigt das Programm den Boot-Sektor auf dem Bildschirm an. Dabei sind zwei Fälle möglich:

- a. Es handelt sich um einen normalen Lader. Diese sind meistens mit deutlichen Copyright-Meldungen versehen. Sie sollten nicht gelöscht werden.
- b. Sie haben einen echten Virus gefunden. Letzte Sicherheit kann hier eigentlich nur das Disassemblieren des Boot-Sektors geben. Sie können jedoch bei ausführbaren Boot-Sektoren ohne Copyright-Meldungen fast immer von

Viren ausgehen. In diesem Fall: Betätigen Sie die j-Taste und löschen damit den Virus von der Diskette. Dazu braucht das Programm nur den Sektor mit leeren Daten zu überschreiben.



Absolute Sicherheit vor Viren wird es auch mit diesem Programm nicht geben, denn die Programmierer solcher Seuchen schlafen nicht und entwickeln ihre Programme weiter.

Version für GFA-BASIC:

```
' VIRUSCHK - Virus-Killer
' Dieses Programm entdeckt Viren im Boot-Sektor einer Diskette
' und zerstört sie auf Wunsch. Zur Orientierung
' wird ein gefundener Virus auf dem Bildschirm angezeigt.
' Taucht darin die Copyright-Meldung von Atari oder einer
' anderen Firma auf, so handelt es sich nicht um einen Virus, sondern um
' einen Bootstraploader für Treiberprogramme, Betriebssysteme o.ä.
' In diesem Fall bitte nicht löschen.
'
' Platz für Boot-Sektor einrichten
A$=Space$(512)
Adr%=Varptr(A$)
T!=False
Repeat
  ' Diskette anfordern
  Print "Bitte legen Sie die verdächtige Diskette in Laufwerk A ein."
  Print "Dann drücken Sie bitte eine beliebige Taste bzw. Esc für Ende."
  Print
  Repeat
    T$=Inkey$
  Until T$<>" "
  If T$<>Chr$(27) Then
    ' Boot-Sektor einlesen
    R%=Bios(4,2,L:Adr%,1,0,0)
    ' Auswerten
    If R%<>0 Then
      ' Fehler beim Lesen
      Alert 3," Lesefehler ! |",1,"Abbruch",R%
    Else
      ' Check-Summe bilden
      S%=0
      For I%=0 To 511 Step 2
        Add S%,Dpeek(Adr%+I%)
        S%=S% Mod 65536
      Next I%
      ' Check-Summe auswerten
      If S%=&H1234 Then
        ' ausführbarer Sektor gefunden!
        ' Sektor anzeigen'
        Print "Virusgefahr! Der Boot-Sektor hat folgendes Aussehen:"
        Print
```

```

For I%=0 To 511
  Out 5, Peek(Adr%+I%)
Next I%
Print
Print
Print "Löschen j/n ?"
Repeat
  T$=Inkey$
Until T$="j" Or T$="n"
If T$="j" Then
  ' neuen Boot-Sektor im Speicher erzeugen
  For I%=30 To 511
    Poke Adr%+I%, &HE5
  Next I%
  ' neue Check-Summe berechnen
  S%=0
  For I%=0 To 511 Step 2
    Add S%, Dpeek(Adr%+I%)
    S%=S% Mod 65536
  Next I%
  If S%=&H1234 Then
    ' ggf. Check-Summe ändern
    Dpoke Adr%+510, (Dpeek(Adr%+510)+1) Mod 65536
  Endif
  ' neuen Boot-Sektor auf Diskette schreiben
  R%=Bios(4,3,L:Adr%,1,0,0)
  If R%<>0 Then
    ' Fehler beim Schreiben
    Alert 3, " Lesefehler !  |",1,"Abbruch",R%
  Endif
Endif
Else
  Print "Diskette enthält keinen Boot-Sektor-Virus!";Chr$(13)
  Void Inp(2)
Endif
Endif
Else
  T!=True
Endif
Cls
Until T!=True

```

Version für Omikron- bzw. ST-BASIC:

```

0 ' VIRUSCHK - Virus-Killer
1 ' Dieses Programm entdeckt Viren im Boot-Sektor einer Diskette und zerstört
2 ' sie auf Wunsch. Zur Orientierung wird ein gefundener Virus auf dem Bild-
3 ' schirm angezeigt. Taucht darin die Copyrightmeldung von Atari oder einer
4 ' anderen Firma auf, so handelt es sich nicht um einen Virus, sondern um
5 ' einen Bootstraploader für Treiberprogramme, Betriebssysteme o.ä. In diesem
6 ' Fall bitte nicht löschen.
7 '

```

```
8 '
9 ' Platz für Boot-Sektor einrichten
10 A$= SPACE$(512)
11 Adr$L= LPEEK( VARPTR(A$))+ LPEEK( SEGPtr +28)
12 T%=0
13 REPEAT
14 ' Diskette anfordern
15 PRINT "Bitte legen Sie die verdächtige Diskette in Laufwerk A ein."
16 PRINT "Dann drücken Sie bitte eine beliebige Taste bzw. Esc für Ende."
17 PRINT
18 T$= INPUT$(1)
19 IF T$<> CHR$(27) THEN
20 ' Boot-Sektor einlesen
21   BIOS (R%,4,2, HIGH(Adr$L), LOW(Adr$L),1,0,0)
22 ' Auswerten
23 IF R%<>0 THEN
24 ' Fehler beim Lesen
25   FORM_ALERT (1,"[3] [ Lesefehler ! ][Abbruch]",R%)
26 ELSE
27 ' Check-Summe bilden
28   S$L=0
29   FOR I%=0 TO 511 STEP 2
30     S$L=S$L+ PEEK(Adr$L+I%)*256+ PEEK(Adr$L+I%+1)
31     S$L=S$L MOD 65536
32   NEXT I%
33 ' Check-Summe auswerten
34 IF S$L=$1234 THEN
35 ' ausführbarer Sektor gefunden!
36 ' Sektor anzeigen'
37 PRINT "Virusgefahr! Der Boot-Sektor hat folgendes Aussehen:"
38 PRINT
39 FOR I%=0 TO 511
40   PRINT CHR$(1); CHR$( PEEK(Adr$L+I%));
41 NEXT I%
42 PRINT
43 PRINT
44 PRINT "Löschen j/n ?"
45 T$= INPUT$(1)
46 IF T$="j" THEN
47 ' neuen Boot-Sektor im Speicher erzeugen
48   FOR I%=30 TO 511
49     POKE (Adr$L+I%),$E5
50   NEXT I%
51 ' neue Check-Summe berechnen
52   S$L=0
53   FOR I%=0 TO 511 STEP 2
54     S$L=S$L+ PEEK(Adr$L+I%)*256+ PEEK(Adr$L+I%+1)
55     S$L=S$L MOD 65536
56   NEXT I%
57 IF S$L=$1234 THEN
58   WPOKE Adr$L+510,( WPEEK(Adr$L+510)+1) AND $FFFF
59 ENDIF
60 ' neuen Boot-Sektor auf Diskette schreiben
```



```

61      BIOS (R%,4,3, HIGH(Adr%L), LOW(Adr%L),1,0,0)
62      IF R%<>0 THEN
63          ' Fehler beim Schreiben
64          FORM_ALERT (1,"[3][ Lesefehler ! ][Abbruch]",R%)
65      ENDIF
66  ELSE
67      PRINT "Diese Diskette enthält keinen Boot-Sektor-Virus!"; CHR$(13)
68      T$= INPUT$(1)
69  ENDIF
70  ELSE
71      T%=1
72  ENDIF
73  CLS
74  UNTIL T%=1

```

9.6 Kommando-Interpreter

In der Zeit vor der Erfindung der grafischen Benutzeroberflächen wie GEM konnte ein Computer nur über die Tastatur bedient werden. Obwohl diese Steuerung über Kommandoworte meistens umständlicher ist, gibt es immer noch einige Freaks, die auch auf dem ST lieber so arbeiten. Zu diesem Zweck gibt es Kommando-Interpreter, die eine Befehlszeile von der Tastatur entgegennehmen und diese dann ausführen. Fairerweise sei gesagt, daß der Betrieb über Kommando-Interpreter (manchmal auch Shells genannt) auch diverse Vorteile mit sich bringt. Das ist wohl der Grund, warum zum Beispiel MS-DOS nur ausnahmsweise und mit Zusatzprogrammen über eine grafische Benutzeroberfläche bedient werden kann.

Die Befehls Worte für die Interpreter orientieren sich meistens an MS-DOS oder dem großen Vorbild UNIX.

DIR oder LS	Anzeigen des Inhaltsverzeichnisses
CD ordnername	In einen Ordner hineingehen
MKDIR ordnername	Neuen Ordner erzeugen
RMDIR ordnername	Leeren Ordner löschen
DEL oder ERA name	Datei löschen
TYPE oder CAT name	Datei anzeigen
EXIT	Kommandointerpreter verlassen
TIME	Uhr stellen bzw. anzeigen
DATE	Datum stellen bzw. anzeigen
MV oder REN	Dateien umbenennen

CP oder COPY
PWD oder PATH

Dateien kopieren
Aktuellen Pfad anzeigen/setzen

Hier ist nur eine kleine Auswahl aufgeführt, den genauen Befehlssatz des Kommando-Interpreters müssen Sie Ihrer Anleitung entnehmen.

Wenn Sie einmal eine Reihe von Dateien irrtümlich in einen falschen Ordner hineinkopiert haben, können Sie sie ganz leicht in das gewünschte Verzeichnis auf dem gleichen Laufwerk verschieben, wenn Sie für jede Datei den Befehl

```
REN dateiname ordnerpfad
```

eingeben. Beispiel: Die Datei TEXT.TXT steht im aktuellen Verzeichnis, soll aber in den Ordner A:\TEXTE\PROJEKT1 verschoben werden. Dazu müssen Sie folgenden Befehl eintippen:

```
REN TEXT.TXT A:\TEXTE\PROJEKT1\
```

Der letzte Backslash ist hier sehr wichtig, weil die Datei sonst in PROJEKT1 umbenannt würde.

Mit den Batch-Dateien begegnen wir dem großen Vorteil der Kommando-Interpreter. Diese haben zumeist die Fähigkeit, sogenannte Batch-Dateien (meist mit der Extension BAT) auszuführen. Vorweg jedoch ein kleiner Hinweis: Batch-Dateien sind oft schon kleine Programme für sich, so daß ihre Beherrschung einiges an Wissen voraussetzt. Dieses kann hier leider nicht in der Kürze dargestellt werden, so daß ich auf die Anleitung der Kommando-Interpreter verweisen muß.

Batch-Dateien enthalten im Grunde genommen nur ganz normale Anweisungen an den Kommando-Interpreter, die dieser dann ausführt, als wären Sie über die Tastatur eingegeben worden. Batch-Dateien können mit jedem Texteditor erstellt werden, der ASCII-Texte speichern kann (also ohne Steuerzeichen). Solche Folgen von Anweisungen können dann nur durch Angabe des Namens aufgerufen werden. Ein Beispiel: Nehmen wir an, vor dem Start eines Programms müßte jedesmal eine Datei mit alten Fehlermeldungen gelöscht werden, die zudem noch recht versteckt in einem Ordner liegt. Sie können natürlich vor jedem Start des Programms in den Ordner hineingehen und die bewußte Datei (nennen wir sie ERRORS.ERR) löschen. Es geht aber auch einfacher. Folgende Batch-Datei löscht die Datei im Ordner DATEN in Laufwerk A und startet dann automatisch das Programm MAIN.PRG:

```
DEL A:\DATEN\ERRORS.ERR  
CD A:\  
MAIN
```

Zur Erläuterung: Die erste Zeile löscht die Error-Datei im genannten Ordner. Mit dem zweiten Kommando wird das Hauptverzeichnis von Laufwerk A zum aktuellen Verzeichnis gemacht und dann durch MAIN das Programm gestartet.

Wird diese Datei auf der Diskette unter dem Namen START.BAT gespeichert, so kann der obige Vorgang jedesmal einfach durch das Eintippen von START ausgelöst werden.

Batch-Dateien ermöglichen oft noch viel weitergehende Manipulationen, oft sind sie sogar richtige Programmiersprachen mit Verzweigungen, Schleifen und Sprüngen.

9.7 Programmiersprachen

Ein Computer kann im Normalfall nur seine spezielle Maschinensprache verstehen, die für den Menschen recht kompliziert ist. Um das Programmieren einfacher und bequemer zu machen, hat man künstliche Sprachen erfunden, die für den Menschen einfache Befehle enthalten. Ein Befehl in einer solchen höheren Programmiersprache entspricht dabei unter Umständen vielen hundert Befehlen in der Maschinensprache (daher die Bezeichnung "höher").

Ein Compiler ist ein Übersetzungsprogramm, das Programmtexte in einer höheren Sprache in die Maschinensprache übersetzt. Diese Übersetzung wird vor dem Programmlauf vorgenommen; es entsteht dabei ein komplettes Programm in Maschinensprache. Diese Zielfprogramme sind sehr schnell und können vom Computer direkt ausgeführt werden. Nachteil: Compiler sind mitunter recht umständlich zu bedienen. Typische Compiler-Sprachen sind C, PASCAL und FORTRAN.

Auch ein Interpreter ist ein Übersetzungsprogramm, das Programmtexte in die Maschinensprache übersetzt. Im Gegensatz zum Compiler wird aber die Übersetzung hier während des Programmlaufs vorgenommen, d.h. der Interpreter übersetzt jeden Befehl Stück für Stück und führt ihn dabei sofort aus. Interpreter sind immer langsamer als Compiler-Sprachen, haben aber den Vorteil, daß sie einfacher bedient werden können. Als Ergebnis der Übersetzung entsteht auch kein Maschinenprogramm, das später noch einmal ausgeführt werden könnte. Statt dessen muß die Übersetzung bei jedem Programmlauf erneut erfolgen. Für den Anfänger auf dem Gebiet der Program-

mierung empfiehlt sich auf jeden Fall ein Interpreter. Typische Interpreter-Sprachen sind BASIC, LISP, LOGO und FORTH.

Richtige Maschinensprache besteht aus Folgen von Zahlen, wobei jede Zahl einen Befehlscode oder die Adresse von Daten darstellt. Wollte man direkt in Maschinensprache programmieren, so müßte man diese Zahlencodes direkt hinschreiben, was sehr fehleranfällig ist. Ein Assembler ist ein Übersetzungsprogramm, das jedem Maschinenbefehl einen Namen gibt (z.B. ADD für einen Additionsbefehl) und Assembler-Quelltexte in solche Maschinenprogramme übersetzt. Das hört sich eigentlich nach der Aufgabe eines Compilers an, es gibt aber wichtige Unterschiede. So ist das Programmieren in Assembler im wesentlichen mit der Maschinensprache-Programmierung identisch; nur kann man statt der Zahl einen Namen hinschreiben, der für den Menschen wesentlich einfacher zu verstehen ist. Zweiter Unterschied: Höhere Programmiersprachen nehmen dem Programmierer sehr viele Aufgaben ab, die mit der Verwaltung von Speicher und Hardware zu tun haben. So muß der Assembler-Programmierer selbst darauf achten, daß sich Programme und Daten im Speicher nicht überschneiden.

9.7.1 Die verschiedenen Sprachen

Um Ihnen einen kleinen Einblick in die verschiedenen Sprachen zu geben, möchte ich jeweils ein kleines Beispiel in BASIC, C und Assembler vorführen. Es handelt sich dabei um eine ganz einfache Aufgabe, die Sie auch ohne Vorkenntnisse nachvollziehen können.

Unsere Programmieraufgabe lautet, die Worte in einem eingegebenen Satz zu zählen. Genauer: Es sollen die Leerzeichen gezählt werden, denn diese trennen ja im allgemeinen zwei Worte voneinander. Um uns nicht mit unnötigen Schwierigkeiten zu belasten, wird der Wortlaut des Satzes in jedem Programm fest installiert; wenn Sie also die Worte in einem anderen Satz zählen lassen möchten, müssen Sie das Programm ändern.

Wie kann man nun die Leerzeichen in einem Satz zählen? Wie würden Sie es machen? Richtig: Alle Buchstaben der Reihe nach ansehen und auf einer kleinen Tafel einen Strich mehr machen, wenn es sich dabei um ein Leerzeichen gehandelt hat. Genauso kann das auch unser Computer machen. Übrigens haben wir gerade einen Algorithmus formuliert. Ein Algorithmus ist nichts weiter als eine Folge von Rechenanweisungen, die die Lösung eines Problems beschreiben. Unser Problem lautet: "Worte zählen" und unser Algorithmus sieht (etwas geordneter aufgeschrieben) so aus:

Vorher: die Anzahl (der bis jetzt gefundenen Leerzeichen) ist 0.
Für jeden Buchstaben des Satzes führe folgendes aus:

Wenn der Buchstabe ein Leerzeichen ist,
dann erhöhe die Anzahl um 1.

Sie werden sehen, daß dieser Algorithmus sehr einfach in ein Programm umgesetzt werden kann. Gleichzeitig haben wir gesehen, was das Programmieren ausmacht: Ein Problem muß analysiert und in viele kleine Teilschritte zerlegt werden. Die Qualität eines Programmierers mißt sich nicht daran, wie virtuos er mit seinem Computer umgehen kann, sondern an der Fähigkeit, Probleme zu analysieren.

9.7.2 BASIC

Wir werden uns im folgenden mit zwei BASIC-Dialekten befassen: GFA- und Omikron-BASIC. Von der Programmierung her sind diese beiden Dialekte fast gleichwertig, allerdings gibt es in der einen Sprache Befehle, die in der jeweils anderen nicht vorhanden sind.

Wie gibt man nun Programme in das GFA-BASIC ein? Starten Sie den GFA-Interpreter zunächst durch einen Doppelklick auf GFABASIC.PRG. Nach dem Laden erscheint der Editor auf dem Bildschirm, der dann ungefähr so aussieht:



Abb. 9.11: Der Editor des GFA-BASIC

Sie können jetzt direkt beginnen, den Programmtext einzutippen. Mit den Cursortasten können Sie dabei wie gewohnt den Cursor bewegen. Aber: Falls Sie beim Eintippen einen Fehler gemacht haben, weigert sich der Editor, diese Zeile zu verlassen. Statt dessen erscheint in der unteren Menüleiste eine Fehlermeldung. In diesem Fall kontrollieren Sie bitte, ob Sie alles richtig eingetippt haben. Erst nachdem Sie den Fehler korrigiert haben, können Sie den Cursor aus der Zeile hinausbewegen.

Wenn Sie mit dem Eintippen fertig sind, sollten Sie das Programm zunächst abspeichern. Dazu klicken Sie bitte mit der Maus in das Feld "SAVE". Es erscheint eine Objektauswahl auf den Bildschirm, mit der Sie das Verzeichnis und den Namen für die Speicherung angeben können. Nach Klick auf OK wird dann gespeichert.

Wenn Sie abgespeicherte Programme wieder laden wollen, klicken Sie bitte auf "LOAD". Es erscheint wiederum eine Objektauswahl, mit der Sie das gewünschte Programm anwählen können.



Wenn Sie ein Programm laden, geht das noch im Speicher befindliche Programm verloren. Achten Sie deshalb darauf, alte Programme ggf. zuerst zu speichern, bevor Sie ein anderes laden.

Zum Starten des eingetippten Programms müssen Sie nur "RUN" in der obersten Menüzeile anklicken. Das Programm wird dann ausgeführt, wobei der Editor-Bildschirm verschwindet. Am Programmende (das kann durchaus sehr schnell kommen) erscheint eine kleine Hinweisbox ("Programmende"). Mit dem Klick auf OK gelangen Sie wieder in den Editor.

Zum Verlassen des GFA-BASIC müssen Sie nur QUIT in der oberen Menüzeile anklicken. Es folgt noch eine Sicherheitsabfrage, dann landen Sie wieder im Desktop.

Etwas anders ist die Vorgehensweise beim Omikron-BASIC. Der Interpreter wird durch einen Doppelklick auf OM_BASIC.PRG gestartet. Nach dem Laden drücken Sie bitte ; damit gelangen Sie in den eigentlichen Editor. Sie können jetzt damit beginnen, den Text des Programms einzutippen. Auch das geht wie gewohnt. Einziger Unterschied: Die -Taste wirkt so, wie man es sonst von her gewohnt ist.

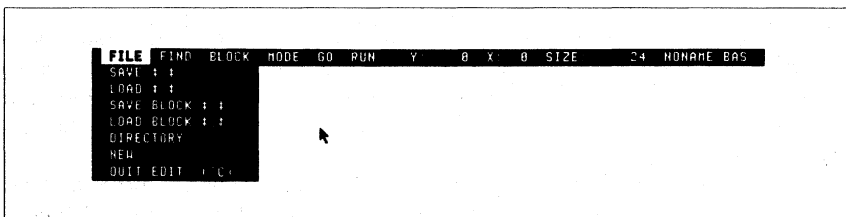


Abb. 9.12: Der Editor des neuen ST-BASIC

In der obersten Bildschirmzeile befinden sich Menüs, die wie die des Desktop funktionieren, aber nicht ganz so aussehen. Wählen Sie "SAVE *.*" im am weitesten links stehenden Menü. Es erscheint eine Objektauswahl, mit der Sie wie gewohnt Verzeichnis und Namen der neuen Datei festlegen können.

Das Laden "alter" Programme geht wie gewohnt über die Menüleiste. "LOAD *.*" anklicken, Datei auswählen, und schon wird das Programm geladen.

Um ein Programm zu starten, wählen Sie "RUN" im ganz rechts stehenden Menü. Das Programm wird sofort ausgeführt. Das Programmende wird nicht irgendwie gekennzeichnet. Nach Beendigung des Programms landen Sie nicht wieder im Editor. Diesen müssen Sie erst durch einen Druck auf Help wieder anwählen.

Um das BASIC wieder zu verlassen, tippen Sie nach der Ausführung eines Programmes (also nicht im Editor) den Befehl QUIT ein. Das BASIC möchte dann noch eine Bestätigung von Ihnen - drücken Sie j, und Sie landen wieder im Desktop.

Wie sieht nun das Programm zum Leerzeichenzählen aus? Es ist für beide BASIC-Dialekte fast gleich. Wenn Sie das untenstehende Programm unter GFA-BASIC eintippen wollen, müssen Sie nur die Zeilennummern am Anfang jeder Zeile weglassen - also z.B. "NEXT I" statt "6 NEXT I".

Hier das Listing:

```
0 Satz$="In diesem Satz gibt es 6 Leerzeichen!"
1 Anzahl=0
2 FOR I=1 TO LEN(Satz$)
3   IF MID$(Satz$,I,1)=" " THEN
4     Anzahl=Anzahl+1
5   ENDIF
6 NEXT I
7 PRINT Anzahl
```

Was bedeuten nun die einzelnen Zeilen? In Zeile 0 wird der zu untersuchende Satz festgelegt. Damit das BASIC diesen Satz irgendwie bearbeiten kann, muß er mit einem Namen versehen werden, man spricht auch davon, daß der Satz einer Variablen zugewiesen wird. In unserem Fall lautet dieser Name "Satz\$". Das Dollarzeichen am Ende kennzeichnet dabei, daß es sich um einen Text handelt, der mit diesem Namen versehen ist. Bei Zahlen entfällt dieser Zusatz, wie man in Zeile 1 sehen kann. Ab sofort betrachtet der Interpreter überall dort, wo der Name "Satz\$" auftaucht, den vorher eingetippten Satz, überall dort, wo "Anzahl" steht, arbeitet das BASIC mit dem Wert 0. Man sagt auch, die Variable Anzahl hat den Wert 0 (dieser Wert kann sich ändern), und die Variable Satz\$ hat den Wert "In diesem Satz gibt es 6 Leerzeichen!".

Erinnern Sie sich an unseren Algorithmus von oben? Dort stand "Für jeden ...". Tatsächlich kennt auch das BASIC einen solchen Befehl. Lesen wir in Zeile 2 ein wenig weiter. Dort steht FOR I= 1 TO LEN(Satz\$). LEN(Satz\$) ist eine weitere Funktion des BASIC, die angibt, wie lang ein Text in einer Variable ist. Wir könnten unsere Zeile jetzt also ungefähr so übersetzen: "Für I von 1 bis zur Länge des Textes". In der Tat wird damit der ganze Text vom ersten bis zum

letzten Buchstaben durchlaufen. Die Variable I enthält dabei jeweils die Nummer des Buchstabens, der gerade betrachtet wird, also 1, 2, 3, usw. Wir wissen aber immer noch nicht, welche Bedeutung das FOR eigentlich hat. Sehen Sie sich dazu Zeile 6 an. Da steht "NEXT I", also auf gut Deutsch "Das nächste I, bitte". Dies ist die Anweisung für das BASIC, mit dem nächsten Wert von I weiterzumachen. Alles, was zwischen dem FOR und dem NEXT steht, wird für jeden möglichen Wert von I einmal ausgeführt. Man nennt das auch Schleife, weil die Bearbeitung dieses Zwischenstücks (die Zeilen 3, 4 und 5) immer wieder von vorne beginnt, sooft das FOR dies angibt. In unserem Fall werden die Zeilen 3, 4 und 5 (der sogenannte Schleifenrumpf) zuerst für den Fall I=1 durchlaufen, dann mit I=2 usw. bis I quasi am Ende des Textes angekommen ist.

Was passiert im Schleifenrumpf? MID\$(Satz\$,I,1) ist eine Anweisung, die das BASIC dazu veranlaßt, aus dem Text in Satz\$ ab der I-ten Stelle genau ein Zeichen herauszugreifen. In unserem Fall wird also der i-te Buchstabe des Textes betrachtet. Dieser wird mit dem Leerzeichen (" ") verglichen. Falls dieser Vergleich positiv ausfiel, haben wir eines der gesuchten Leerzeichen im Satz gefunden, und die Anzahl muß um 1 nach oben verändert werden. Betrachten wir Zeile 3 einmal genauer. Etwas frei übersetzt steht dort "Wenn i-ter Buchstabe ein Leerzeichen ist". Das Wörtchen "dann" liegt Ihnen sicher auf der Zunge. Richtig! Sie finden das "dann" in der nächsten Zeile als "THEN" wieder. Zeile 4 wird nur dann ausgeführt, wenn die Bedingung MID\$(...)= " " erfüllt war; dies wird durch das vorgestellte THEN bewirkt. Fiel der Vergleich negativ aus, so wird Zeile 4 übersprungen.

Was passiert nun, falls ein Leerzeichen gefunden wurde? Hinter dem THEN steht die Anweisung "Anzahl=Anzahl+1". Mathematisch ist das natürlich totaler Quatsch, denn eine Zahl kann niemals gleich ihrem Nachfolger sein. Hier findet auch kein Vergleich statt, sondern eine Zuweisung wie in Zeile 1. Das BASIC soll zum Inhalt der Variablen Anzahl eine 1 hinzuaddieren und dann das Ergebnis wieder mit Anzahl bezeichnen. (Der alte Wert von Anzahl geht dabei verloren.). Auf diese Weise wird der Wert von Anzahl genau um 1 nach oben gezählt, also genau das, was wir wollten. Als kleine Eselsbrücke können Sie das Gleichheitszeichen hier in Gedanken durch einen Pfeil nach links ersetzen, etwa so: Anzahl <- Anzahl + 1.

Die Zeile 5 dient dem BASIC nun nur noch dazu, das Ende der IF-THEN-Konstruktion zu markieren. Sie sehen also, wenn das betrachtete Zeichen kein Leerzeichen war, tut der Computer nichts außer zum nächsten Zeichen überzugehen.

In Zeile 7 wird schließlich das Ergebnis unserer ganzen Rechnerei ausgegeben. PRINT bedeutet wörtlich "Drucke", im Fall von BASIC (und vielen anderen Sprachen) heißt es, etwas auf dem Bildschirm anzuzeigen. Was, das wird danach angegeben. PRINT Anzahl bedeutet dabei nicht etwa, daß das Wort Anzahl ausgegeben wird, sondern der Wert der Variablen Anzahl, und das ist nun einmal mehr oder minder zufällig das Ergebnis, das wir haben wollten.

9.7.3 C

C ist eine Compiler-Sprache, die sehr effiziente Programmierung erlaubt. Wie effizient dies sein kann, sehen Sie nicht zuletzt daran, daß weite Teile des GEM und des TOS in C geschrieben wurden. Es handelt sich hier also um die Leib- und Magensprache unseres Computers. Da C-Compiler recht teuer und kompliziert zu bedienen sind, werde ich mich im folgenden darauf beschränken, das Aussehen unseres obigen Problems in C zu demonstrieren. Falls Sie sich näher für diese Sprache interessieren, sollten Sie dann zu geeigneter Fachliteratur greifen. Ich empfehle Ihnen aber, die Programmierung erst anhand einer einfacheren Sprache zu erlernen (BASIC oder PASCAL), bevor Sie zu diesem Expertenmittel greifen.

So sieht das Programm in C aus:

```
char satz[]="In diesem Satz gibt es 6 Leerzeichen!";
main()
{
    int i,anzahl=0;
    for(i=0;satz[i]!='\0';i++)
        if (satz[i]==' ')
            anzahl=anzahl+1;
    printf ("%d\n",anzahl);
}
```

"main()" ist die Bezeichnung für das Hauptprogramm. C-Programme bestehen im allgemeinen aus mehreren Teilprogrammen, die sich gegenseitig aufrufen können. Das Hauptprogramm, welches als erstes ausgeführt wird, heißt aber immer "main". Die geschweiften Klammern umschließen dabei den Rumpf des Hauptprogrammes.

In den Zeilen 3 und 4 werden die Variablen definiert. In C muß jede Variable vor der ersten Benutzung angemeldet ("definiert") werden. Die Variable "satz" hat hier den Typ char (Character, zu deutsch "Zeichen"); der Stern kennzeichnet die Tatsache, daß es sich um eine ganze Zeichenkette und nicht nur ein Zeichen handelt. "int" steht für Integer (ganze Zahlen). Ansonsten wurden die Variablen wie in BASIC gewählt.

Das for-Statement kennen Sie schon. In C ist es aber ein wenig anders aufgebaut. Innerhalb der Klammer gibt es drei durch Semikola getrennte Angaben. "i=0" gibt an, daß die Zählung bei 0 beginnen soll. "satz[i]!=0" bedeutet, daß nur so lange gezählt werden soll, bis das i-te Zeichen im Satz den ASCII-Code 0 hat (wohlgemerkt den ASCII-Code, nicht das Zeichen 0). In C kennzeichnet dies das Ende einer Zeichenkette. "i++" gibt schließlich an, daß i in Einer-schritten erhöht werden soll.

Auch das if-statement dürfte Ihnen vertraut erscheinen. Wenn das i-te Zeichen im Satz ein Leerzeichen ist, wird in der darauf folgenden Zeile die Variable Anzahl um 1 erhöht. Wie Sie sehen, schert sich C auch nicht um organisatorische Sachen wie ein ENDIF oder NEXT. Das wird in C fast ganz allein durch die Struktur der Befehle ausgedrückt und verlangt vom Programmierer einige Übersicht.

"printf..." druckt schließlich - wie gehabt - die Anzahl auf dem Bildschirm aus.

9.7.4 Assembler

Es würde sicher zu weit führen, wollte ich das obige Problem auch in Assembler vorführen. Dazu wären seitenlange Erklärungen notwendig, die dieses Buch unnötig aufblähen würden. Zum Verständnis eines Teils der Prinzipien der Assembler-Programmierung soll uns ein einziger Befehl genügen. Sehen wir uns einmal an, wie man in Assembler zwei Zahlen addiert. Nehmen wir an, diese beiden Zahlen lägen auf den Speicherplätzen ZAHL1 und ZAHL2.

Bevor ein Prozessor irgendwelche Daten bearbeiten kann, muß er diese Daten in seine Register holen. Register sind interne kleine Speicher im Prozessor, die besonders schnell manipuliert werden können. Und vor allem: Die meisten Operationen können nur in einem Register, nicht im normalen Speicher durchgeführt werden. Bevor der Prozessor also ZAHL2 addieren kann, muß er ZAHL1 in ein Register (hier heißt es D0) holen. Das bewerkstelligt der folgende Befehl:

```
MOVE ZAHL1,D0
```

Beachten Sie bitte, daß jetzt die Zuweisungsrichtung von links nach rechts geht. (ZAHL1 wird nach D0 transportiert, in BASIC würde das ungefähr so aussehen: DO=ZAHL1.).

Jetzt kann die zweite Zahl schon addiert werden, denn der Additionsbefehl führt zwei Tätigkeiten aus: Die zweite Zahl holen und zum Inhalt von Register DO addieren. Der Befehl lautet dann:

```
ADD ZAHL2, DO
```

Das entsprechende BASIC-Äquivalent hieße $DO=DO+ZAHL2$. Damit enthält das Register DO den Wert $ZAHL1+ZAHL2$. Dieses Ergebnis soll nun noch an einen Speicherplatz mit dem Namen ERG gebracht werden. Auch dafür kann MOVE benutzt werden:

```
MOVE DO, ERG
```

In BASIC-Schreibweise:

```
ERG=DO.
```

Sie können sich vorstellen, daß die Programmierung in Assembler recht kompliziert werden kann. Dafür gibt es aber auch keine andere Sprache, die so effizient und schnell arbeiten kann. So hat jede Programmiersprache ihre Vor- und Nachteile.

10. Pflege und Wartung

Wie jedes technische Gerät braucht auch ein Computer ein gewisses Maß an Pflege. Aber bitte - stürzen Sie jetzt nicht in die Küche und holen Ihre Hausmittelchen hervor. Trotz aller weitentwickelten Technik sind Computer auch empfindlich. Wie es geht, zeigen die folgenden Abschnitte. Unabhängig von den verschiedenen Pflegehinweisen gibt es einige Regeln, die Sie im Umgang mit Computern immer beachten sollten:

1. Der Computerarbeitsplatz sollte trocken, staub- und fettfrei sein. Staub setzt sich mit der Zeit auf der Platine und in der Tastatur ab und kann die Funktionsfähigkeit beeinträchtigen.
2. Am Computerarbeitsplatz sollten Sie nicht rauchen und weder essen noch trinken. Rauch ist (neben der gesundheitsschädlichen Wirkung für den Menschen) Gift für viele Computer, weil die Platinen mit einer feinen Schmierschicht aus Teer und Rauchpartikeln überzogen werden. Ganz besonders allergisch reagieren Festplatten auf Rauch, auch wenn diese mit einem Luftfilter ausgestattet sind. Und sicherlich können Sie sich die Wirkung einer Tasse Kaffee auf der Hauptplatine des Rechners sehr plastisch vorstellen.
3. Setzen Sie Ihren Computer nicht starker Hitze oder Sonneneinstrahlung aus. Durch die Hitze kann sich die Platine verziehen, was oft zu Rissen in Leitungsbahnen führt. Diese Risse sorgen dann für Fehlschaltungen.
4. Verdecken Sie niemals Lüftungsschlitze an den Geräten. Betreiben Sie die Geräte vor allem nicht auf weichen Unterlagen wie Teppich oder dicken Tischdecken, da der Computer darin "einsinkt" und so die Lüftungsschlitze am Boden verdeckt werden.
5. Transporte sind Gift für einen Computer (die sogenannten Portables einmal ausgenommen). Die Geräte sollten einen festen Platz haben. Falls ein Transport doch einmal nötig werden sollte, nehmen Sie bitte die Originalverpackung als Transportvehikel. Vor allem bei der Festplatte sollten Sie Vorsicht walten lassen (siehe Kap. 3). Sie sollten beim Transport auch darauf achten, daß keine Disketten mehr in den Laufwerken stecken!
6. Knicken Sie niemals Kabel. Geknickte Kabel können Kurzschlüsse verursachen, und die sind der natürliche Feind eines jeden Computers.

10.1 Reinigung

Der Computer

Reinigung des Gehäuses

Das Gehäuse kann hin und wieder mit einem leicht feuchten Lappen abgewischt werden, um schmierigen Belag zu entfernen. Der Lappen sollte feucht und auf keinen Fall naß sein. Außerdem sollten Sie das Gerät während dieser Prozedur abschalten und erst nach dem Trocknen wieder einschalten.

Ist das Gerät aber nur staubig geworden, vergessen Sie bitte den Lappen. Ein kleiner Spezialstaubsauger oder auch der auf kleinste Leistungs gestellte Hausstaubsauger entfernt den Staub - aber bitte: Gehen Sie vorsichtig zu Werke.

Reinigung der Tastatur

Auch die Tastatur sollte von Zeit zu Zeit von Staub befreit werden. Auch hier können Sie wieder mit dem Staubsauger zu Werke gehen. Aber auch bei noch so guter Pflege wird sich die Tastatur mit einem schmierigen braunen Belag überziehen, der von unseren Händen stammt. Um diesen zu entfernen, nehmen Sie bitte ein Wattestäbchen und tränken Sie es in reinem Alkohol oder Naßabspießflüssigkeit für Schallplatten. Damit können Sie den Belag bei ausgeschaltetem Gerät abwischen.

Reinigung der Anschlüsse

Die Schnittstellen sind für unsere Reinigungsaktivitäten tabu. Hier können Sie mehr Schaden anrichten als Nutzen. Ganz abgesehen davon werden die Schnittstellen kaum verschmutzen. Sollte sich einmal in einem nicht benutzten Stecker ein wenig Staub absetzen, blasen Sie ihn einfach hinaus.

Die Maus

Reinigung des Gehäuses

Auch das Mausgehäuse kann von Zeit zu Zeit mit einem feuchten Lappen abgewischt werden. Aber bitte: nur bei ausgeschaltetem Gerät.

Reinigung der Mechanik

Die Mausmechanik ist recht empfindlich. Daher sollte regelmäßig (am besten einmal in der Woche) die Kugel gereinigt werden. Dazu gehen Sie bitte so vor:

1. Legen Sie die Maus auf den Rücken. Auf der Unterseite befindet sich ein Loch für die Rollkugel, darüber eine Abdeckplatte.

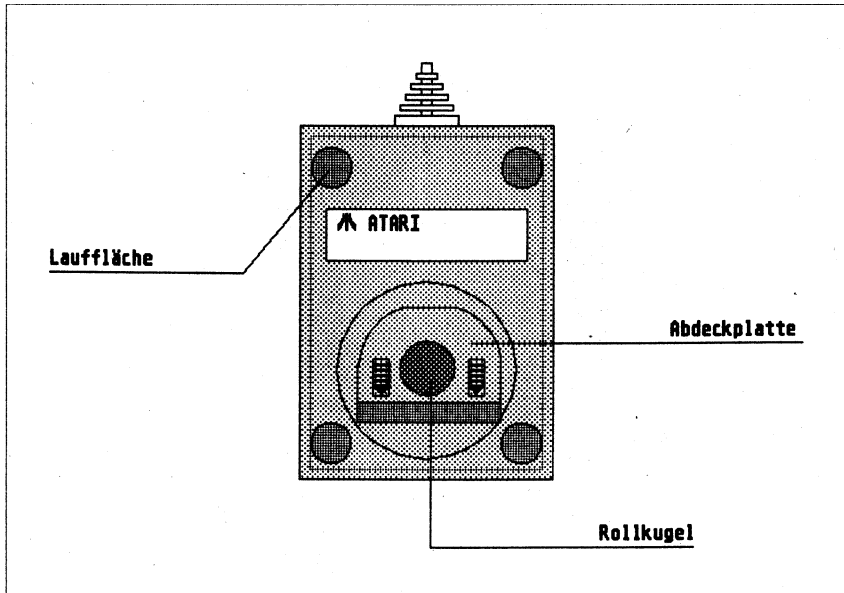


Abb. 10.1: Die Unterseite der Maus

2. Schieben Sie die Abdeckplatte auf, indem Sie mit zwei Fingern auf die Pfeilfelder drücken und schieben, bis die Platte ausrastet.
3. Drehen Sie die Maus vorsichtig herum, halten Sie dabei die andere Hand unter die Maus, so daß die Kugel in Ihre Hand fällt.
4. Reiben Sie die Kugel mit einem trockenen Tuch (wie wär's mit Ihrem Taschentuch?), und verwenden Sie dabei auf keinen Fall irgendwelche Reinigungsmittel. Die Rollkugel ist recht empfindlich und könnte durch die Reinigungsmittel angegriffen werden. Als Folge "eiert" die Maus, und die Steuerung wird sehr ungenau.

5. Blasen Sie Staub aus dem Gehäuse heraus. Achten Sie dabei auch auf die drei Walzen, die sich im Gehäuse befinden. Manchmal bildet sich darauf ein regelrechter Ring von Staub und Schmier, der die genaue Steuerung beeinträchtigen kann. Sie können diesen Ring notfalls mit dem Fingernagel vorsichtig abkratzen (aber niemals mit Gewalt oder Reinigungsmitteln vorgehen). Benutzen Sie dazu auf keinen Fall einen harten Gegenstand wie z.B. eine Schere.

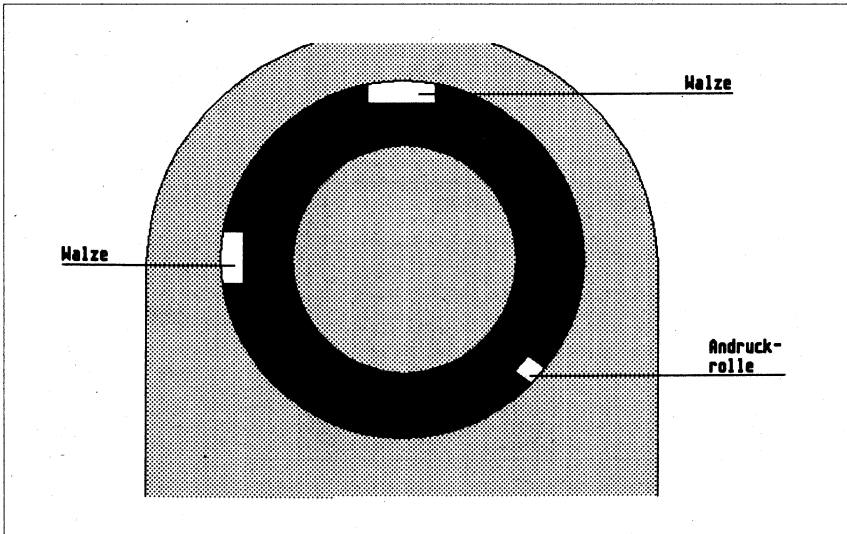


Abb. 10.2: Der "Käfig" für die Rollkugel

6. Sie können die Maus jetzt wieder zusammenbauen. Dazu brauchen Sie nur die Kugel wieder einzusetzen und die Abdeckplatte entgegen der Pfeilrichtung einrasten zu lassen.

Das Diskettenlaufwerk

Reinigung des Gehäuses

Hier können Sie wie auch beim Rechnergehäuse vorgehen. Gehen Sie aber mit dem Staubsauger besonders vorsichtig zu Werke, und versuchen Sie auf gar keinen Fall, den Staub aus dem Diskettenschlitz herauszusaugen.

Reinigung des Inneren

Das ist eine Aufgabe für Fachleute. Die Mechanik eines Diskettenlaufwerkes ist empfindlich, so daß Sie unter Umständen mehr Schaden anrichten als Reinigungswirkung erzielen. Wenn Ihre Floppy zu Lesefehlern neigt, so kann das an dejustierten oder verschmutzten Schreib-/Lese-Köpfen liegen. Was die Ursache ist, kann meist nur der Fachmann mit speziellen Prüfeinrichtungen feststellen.

Abzuraten ist von sogenannten Reinigungsdisketten. Wenn Sie Besitzer einer Hifi-Anlage sind, wissen Sie wahrscheinlich, daß Reinigungsdisketten mit Gewebebändern den Tonkopf regelrecht zerkratzen können. Das gleiche gilt für Reinigungsdisketten, die nach dem gleichen Prinzip arbeiten.

Wenn es dann doch einmal sein muß, nehmen Sie unbedingt solche Reinigungsdisketten, in die eine feuchte Scheibe eingelegt wird, die nur ein einziges Mal und sehr kurz verwendet werden darf. Bloßes Einlegen dieser Disketten in das Laufwerk bringt allerdings nichts.

Der Rechner muß versuchen, auf die Diskette zuzugreifen, um die Köpfe zu reinigen. Das erreichen Sie, indem Sie versuchen, das Inhaltsverzeichnis der Reinigungsdiskette anzeigen zu lassen. Nach spätestens 30 Sekunden sollten Sie Rechner und Laufwerk abschalten und erst nach einer weiteren Minute (Trockenzeit für das Reinigungsmittel) wieder einschalten - diesmal mit der Systemdiskette.

Disketten

Disketten können nicht gereinigt werden. Versuchen Sie bitte niemals, die Magnetscheibe im Inneren der Diskette zu berühren, es sei denn, Sie wollen unbedingt Ihre Daten zerstören.

Wenn der Aufkleber einer häufig benutzten Diskette abgegriffen sein sollte, können Sie ihn vorsichtig abziehen und durch einen neuen ersetzen.

Festplatten

Reinigung des Gehäuses

Wie oben beschrieben. Sie sollten aber auf keinen Fall versuchen, das Gerät zu öffnen. Ansonsten bieten sich bei der Festplatte wenig Möglichkeiten der Pflege. Achten Sie darauf, daß vor jedem Transport die Schreib-/Lese-Köpfe geparkt werden.

Drucker

Reinigung des Gehäuses

Genaue Angaben entnehmen Sie bitte dem Handbuch zu Ihrem Drucker. In den meisten Fällen können Sie den Drucker mit einem feuchten Tuch abwischen. Außerdem wird sich mit der Zeit im Inneren des Druckers eine Menge Papierstaub absetzen. Diesen können Sie mit einem Staubsauger sehr vorsichtig entfernen. Noch besser ist ein Pinsel mit eingebautem Blasebalg, wie er für die Reinigung von Kameras verwendet wird. Bevor Sie das tun, sollten Sie allerdings das Papier aus dem Drucker nehmen und das Farbband sowie die Abdeckhaube entfernen. Bedenken Sie, daß ein Drucker ein feinmechanisches Gerät ist, das empfindlich ist. Bei all diesen Aktionen sollten Sie es aber vermeiden, den Druckkopf zu bewegen. Natürlich gilt auch hier: alle Reinigungsaktivitäten nur bei ausgeschalteten Geräten.

Reinigung der Walze:

Sofern Ihr Drucker das Herausnehmen der Walze erlaubt, sollten Sie das nun anhand der Anleitung tun. Dann können Sie die Walze vorsichtig mit einem trockenen Tuch abreiben. Falls sie fest in das Gerät eingebaut ist, müssen Sie sie mit dem seitlichen Rad drehen und Stück für Stück reinigen.

Bildschirm

Hier ist eine Reinigung nur selten möglich - außer natürlich dem regelmäßigen Staubwischen. Feuchtigkeit hat an einem Monitor nichts zu suchen.

Steckmodule

Auch diese können eigentlich nicht gereinigt werden. Es kann aber vorkommen, daß die Kontakte vor allem bei häufigem Ein- und Ausstecken verschmutzen, was dann zu Fehlschaltungen führen kann. Einzige Möglichkeit: Mit einem harten Radiergummi, das möglichst wenig Krümel hinterläßt, kann der Belag auf den Kontakten abradert werden.

10.2 Die Betriebssystem-ROMs

Wie Sie vielleicht wissen, gibt es viele verschiedene Versionen des Atari ST. Eine Auflistung der ersten Typenbezeichnungen sieht ungefähr so aus:

520 ST, 520 ST+, 260 ST, 1040 ST, Mega ST 1, 2 oder 4, Mega STE 1,2 oder 4

Urahn aller ST-Modelle war der 520 ST, der 512 KByte Speicher besaß. Er wurde schon nach kurzer Zeit durch die beiden Modelle 260 ST (= 520 ST ohne gewisse Kinderkrankheiten) und 520 ST+ (260 ST mit 1 MByte Speicher) abgelöst. Diesen drei Veteranen-Modellen ist neben dem Gehäuse eines gemeinsam. Sie wurden (zumindest anfänglich) ohne Betriebssystem-ROMs ausgeliefert. Was heißt das? Beim Booten muß der Prozessor sich auf irgendeine Weise das Betriebssystem besorgen. Wenn es nicht in den ST eingebaut ist, muß es zwangsläufig auf der System-Diskette stehen, die diesem Umstand ursprünglich ihren Namen verdankt.

Das Laden des gesamten Betriebssystems mit immerhin 192 KByte Umfang dauert natürlich einige Zeit. Außerdem belegt es den wertvollen RAM-Speicher. Sobald das Betriebssystem von den größten Macken befreit war (auch Betriebssysteme sind Programme, und fehlerfreie Programme gibt es praktisch nicht), hat sich Atari dazu entschlossen, diese Software auf ROMs zu bannen und dann in alle ST-Modelle einzubauen. Die Folge: Atari-Rechner mit eingebauten ROMs brauchen im Grunde genommen gar keine Systemdiskette mehr, ihnen ist jede Platte recht. Die einzige Daseinsberechtigung schöpft die Systemdiskette noch aus der Tatsache, daß Accessories und Treiber geladen werden müssen.

Die alten Modelle ohne ROMs erkennt man daran, daß sie unbedingt ihre spezielle Systemdiskette haben wollen, auf der ein File namens "TOS.IMG" vorhanden ist. Falls Sie sich nicht sicher sind, ob Ihr ST ROMs eingebaut hat oder nicht (manchmal kann man solche Geräte noch gebraucht erwerben), können Sie folgenden Test anwenden:

1. Nehmen Sie eine frisch formatierte Diskette, auf der keine Dateien gespeichert sind. Im Fenster muß bei solchen Disketten "0 Bytes in 0 Dateien" angezeigt werden. Legen Sie diese in den Rechner ein.
2. Schalten Sie den Rechner aus und dann wieder ein. Jetzt gibt es zwei Möglichkeiten:

Nach einer kurzen Zeit erscheint ein Desktop-Bild, allerdings ohne Fenster. Herzlichen Glückwunsch! Ihr ST besitzt eingebaute ROMs.

Nach einer kurzen Zeit erscheint eine Grafik, die Sie auffordert, eine Systemdiskette einzulegen und dann mit der Maus einen Knopf anzuklicken. Pech gehabt! Ihr Veteranen-ST hat leider keine ROMs. Aber bitte keine Panik. Die ROMs kann man sehr schnell und einfach nachrüsten.

ROM einbauen

Besorgen Sie sich bitte zunächst die ROMs bei Ihrem Fachhändler. Dieser kann Ihnen die ROM-Bausteine auch in den Rechner einbauen, wenn Sie das möchten (gegen Aufpreis versteht sich). Wenn Sie die ICs haben, sollten Sie sich folgendes Werkzeug bereitlegen:

- Kreuzschlitzschraubendreher (in verschiedenen Größen),
- zwei Schraubendreher mit ca. 3 mm Breite,
- eine möglichst feine Zange (manchmal tut es auch eine sehr starke Pinzette),
- eine kleine Schale, um die Schrauben aufzubewahren.

Trennen Sie nun den ST von allen Verbindungen, insbesondere natürlich vom Netzstecker und der Maus. Legen Sie den Rechner auf die Tastatur, damit Sie an die Schrauben herankönnen. Lösen Sie nun bitte alle sechs Schrauben an der Unterseite, mit denen das Gehäuse befestigt ist. Dann drehen Sie bitte den Rechner wieder in die normale Gebrauchslage (Achtung - das Gehäuse ist bereits gelöst). Im folgenden gehe ich immer davon aus, daß der Rechner in der gleichen Position vor Ihnen liegt, in der er auch beim normalen Gebrauch ist. Sie können jetzt das Gehäuseoberteil nach oben abziehen.

Vor Ihnen liegt die Tastatur. Diese ist mit einem Strang von Kabeln mit der Platine verbunden. Sie können die Tastatur etwas beiseite schieben, um mehr Platz zu bekommen. Auf der Platine enden die Kabel in einem Stecker, den Sie nun sehr vorsichtig abziehen müssen. Am besten, Sie hebeln den Stecker mit einem Schraubenzieher millimeterweise nach oben, bis er sich ganz leicht abziehen läßt. Reißen Sie dabei auf keinen Fall an den Kabeln. Ist der Stecker gelöst, können Sie die Tastatur an die Seite legen, nicht ohne vorher die Gelegenheit beim Schopfe zu packen und alle Staubpartikel zwischen den Tasten wegzublasen. Der geöffnete Rechner müßte nun so aussehen:

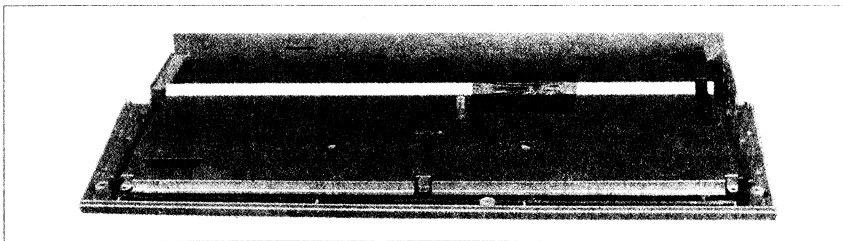


Abb. 10.3: Das geöffnete Rechnergehäuse ohne Tastatur

Das graue Ding ist eine Abschirmung, die den Rechner vor Störeinflüssen bewahren und Ihr Radio vor Rauschen aus dem Computer schützen soll - ob das gelingt, ist nicht immer sicher. Auf jeden Fall macht uns diese Abdeckung nun einige Schwierigkeiten, denn sie ist nicht ganz leicht zu entfernen. Am Rand der Abdeckung sehen Sie elf Laschen, die das Blechungsgewürm an die Platine klammern. Sie müssen alle elf Laschen, die teilweise sehr versteckt liegen, aufbiegen. Sollte Ihre Zange zu groß sein, um an die hinteren Laschen heranzukommen, versuchen Sie mit zwei Schraubendrehern, den gleichen Effekt nachzuahmen.

Sind die Laschen endlich aufgebogen, müssen Sie noch drei Schrauben am unteren Platinenrand lösen, die ebenfalls der Befestigung der Abdeckung dienen, außerdem aber die Platine am Gehäuseunterteil festhalten. Am oberen Rand befindet sich außerdem noch ein messingfarbener Klebestreifen. Dieser muß ebenfalls abgenommen werden, soweit das möglich ist. Die Abschirmung sollte nun vollständig von der Platine gelöst sein und sich relativ leicht nach oben abheben lassen. Geschafft: Vor uns liegt die Rechnerplatine.

Verschaffen wir uns erst einmal ein wenig Überblick. Anhand der aufgedruckten Nummern können Sie die meisten großen ICs aus dem Blockdiagramm wiedererkennen. Der größte IC ist der Prozessor (in der Mitte der Platine). Darüber sitzen Floppy-Controller und DMA-Chips. Die quadratischen Bauteile in der Mitte der Platine sind Spezialchips von Atari, die für besondere Aufgaben konstruiert wurden. Direkt vor Ihnen an der rechten Seite sehen Sie eine ganze Reihe kleinerer ICs: das RAM. Unter einer weiteren Abdeckung finden Sie dann noch die Chips für die Bilderzeugung.

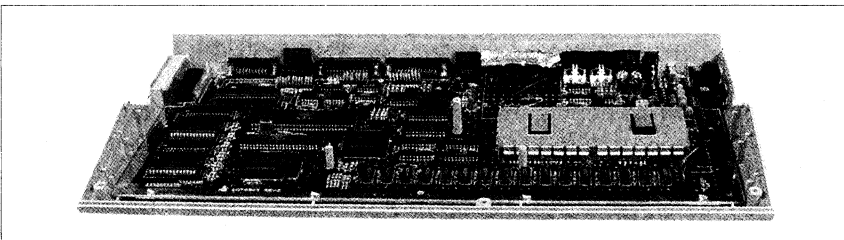


Abb. 10.4: Die Rechnerplatine



Die Platine ist immer noch mit dem Gehäuseunterteil verbunden. Versuchen Sie also nicht, sie aus dem Gehäuse herauszunehmen. Das können Sie erst, wenn Sie weitere drei Schrauben an der oberen Platinenkante gelöst haben. Übrigens sollten Sie auch die Platine bei dieser Gelegenheit von Staub befreien.

An der linken Seite neben dem Prozessor befinden sich sechs IC-Sockel von etwa doppelter Briefmarkengröße. Zwei davon sind mit ICs bestückt. (Das sind die alten Boot-ROMs, die das Betriebssystem von der Diskette laden.). Vier andere Sockel sind noch frei. Die Sockel sind auf der Platine jeweils links mit U2 bis U7 beschriftet. In diese Sockel werden später die neuen ROMs eingesetzt. Zuvor müssen wir aber die beiden alten ROMs entfernen. Fassen Sie dazu mit zwei Schraubendrehern vorsichtig zwischen IC und Sockel, und hebeln Sie die schwarzen ICs heraus. Sie sollten dabei sehr vorsichtig zu Werke gehen, denn vielleicht brauchen Sie die alten ROMs noch einmal. Verpacken Sie die alten Bausteine so, wie Ihnen die neuen geliefert wurden. Jetzt können wir endlich die neuen ROM-Bausteine einsetzen. Diese tragen auf der Oberseite eine Reihe von Zahlen und Buchstaben. Vor allem finden Sie auf jedem die Bezeichnung C02630x, wobei x für eine Ziffer zwischen 2 und 7 steht. Anhand dieser Ziffer können Sie feststellen, welches IC in welchen Sockel gehört. C026302 kommt in Sockel U2, C026303 in U3 usw. (C02630x in Ux). Wichtig ist auch, wie Sie die ICs einstecken. Falls die Bausteine eine halbrunde Kerbe auf der Oberseite besitzen, muß diese Kerbe nach rechts (in Richtung des Mausanschlusses) weisen. Besitzt ein Baustein keine Kerbe, so muß die Aufschrift für Sie auf dem Kopf stehen, also vom Floppy-Anschluß aus lesbar sein.

Drücken Sie nun die ICs der Reihe nach in die Sockel. Dabei dürfen Sie keine Gewalt anwenden. Unter Umständen müssen Sie die Pinreihen etwas zusammendrücken, damit der IC in den Sockel paßt. Achten Sie darauf, den Baustein möglichst gleichmäßig in die Fassung zu drücken. Aber Vorsicht: Ein abgebrochener Pin macht einen solchen Stein unbrauchbar. Die ICs müssen vollständig in den Sockel hineingedrückt werden, sonst gibt es Kontaktschwierigkeiten. Sind Sie erst einmal so weit gekommen, heißt es die eigene Arbeit zu überprüfen. Sehen Sie noch einmal nach, ob die Bausteine alle richtig herum in den Sockeln sitzen. Dann kommt der erste Test. Schließen Sie den Rechner (immer noch mit offenem Gehäuse und ohne Tastatur) an Floppy, Bildschirm sowie die Stromversorgung an. Schalten Sie ein. Wenn Sie alles richtig gemacht haben, sollte der Rechner jetzt mit dem Booten beginnen. Tut er das, ist alles O.K., tut er das nicht, haben Sie irgendwo einen Fehler gemacht oder die ROMs sind defekt. In diesem Fall hilft nur noch der Händler. Lassen Sie den Boot-Vorgang nicht bis zu Ende laufen, denn durch die fehlende Tastatur hat das kaum ein sinnvolles Ergebnis. Sie können jetzt daran gehen, den Rechner wieder zusammenzubauen. Dabei gehen Sie genau umgekehrt wie beim Aufschrauben vor. Übrigens paßt der Tastaturstecker nur in einer Richtung, so daß Sie hier nichts falsch machen können.

10.3 Rechner-Tuning

Wenn Sie mit dem Gehäuse Ihres ST nicht zufrieden sind, können Sie die Platinen und Peripheriegeräte in ein großes PC-Gehäuse einbauen. Das hat den Vorteil, daß Sie nur noch ein Gehäuse plus Tastatur auf dem Schreibtisch haben und bei geschicktem Vorgehen auch die Netzteile zu Ihren Füßen verschwinden. Die Tastatur kann dann in ein eigenes Gehäuse eingebaut werden, das über ein Spiralkabel mit dem Rechner verbunden wird. Aber für einen solchen Umbau müssen Sie etwas handwerkliches Geschick mitbringen. Es gibt auch fertige Umbausätze von einigen Herstellern, in die Sie dann nur noch die einzelnen Komponenten einstecken müssen. Beachten Sie bitte eins: Das Öffnen der Gehäuse läßt den Garantieanspruch erlöschen. Machen Sie solche Umbauten also erst nach dem Ablauf der Garantiezeit. Wenn Ihnen die Tastatur zu klapperig erscheint, gibt es eine einfache Abhilfe. Dazu müssen Sie das Gehäuse wie im letzten Kapitel beschrieben öffnen, um an die Tastatur heranzukommen. Diese muß aber nicht ausgebaut werden. Nehmen Sie dann selbstklebendes Dichtungsband für Fenster, und kleben Sie dies auf alle Stellen, an denen die Tastatur Kontakt mit dem Gehäuse hat. Um ein seitliches Verschieben der Tastatur zu vermeiden, können Sie auch noch links und rechts der Grundplatte solche Streifen anbringen. Nach dem Zusammenbau wird das Tastenfeld sehr viel fester im Gehäuse sitzen.

10.4 Kleine Reparaturen selbst gemacht

Viel kann der Anwender leider nicht selbst machen, in den meisten Fällen ist man auf die Fachwerkstatt angewiesen. Aber in manchen Fällen ist man selbst schneller. Bei Problemen mit Peripheriegeräten ist oft das Kabel die Ursache des Fehlers. Kontrollieren Sie bitte zunächst, ob alle Stecker korrekt an den dafür vorgesehenen Plätzen sitzen. Ist das Problem dann immer noch nicht beseitigt, sollten Sie das Kabel testen. Die eine Möglichkeit: Besorgen Sie sich von einem Bekannten ein nachweislich funktionierendes Kabel des gleichen Typs, und tauschen Sie das probeweise gegen Ihr eigenes aus. Tritt der Fehler nicht mehr auf, liegt es wirklich am Kabel. Das Kabel muß dann in den meisten Fällen gegen ein neues ausgetauscht werden, nur in Ausnahmefällen können Sie es reparieren (z.B. durch Austauschen der Litzen oder Reparatur defekter Lötstellen). Absolutes Reparaturverbot gilt aber beim Monitorkabel, aus diesem Grund ist es auch fest mit dem Bildschirm verbunden. Die Reparatur von Netzkabeln lohnt nicht, sie lassen sich preiswert gegen neue austauschen. Können Sie kein Austauschkabel zum Testen auftreiben, müssen Sie per Hand jede einzelne Litze prüfen. Dazu brauchen Sie einen Durchgangsprüfer oder ein Multimeter. Sehen Sie dann nach, welche Pinbelegungen die Stecker haben, und prüfen Sie, ob die Litzen wirklich bis zum anderen Kabelende eine

einwandfreie Verbindung ermöglichen. Dabei dürfen Sie nicht einfach testen, ob Pin x am einen Stecker mit Pin x am anderen Stecker verbunden ist, denn es könnte ja sein, daß sich innerhalb des Kabels Leitungen kreuzen (z.B. bei einem Nullmodem). Die Leitungen müssen von Stecker zu Stecker einen möglichst geringen Widerstand aufweisen. Liegt die Fehlerursache nicht im Kabel, ist eines der Geräte defekt. Lassen Sie in diesem Fall bitte die Finger davon. Sie werden mehr Schaden anrichten als Fehler ausmerzen, wenn Sie selbst versuchen sollten, ein Gerät zu reparieren oder eine Diskettenstation neu zu justieren. Dafür sind spezielle Einrichtungen und Anleitungen notwendig.

11. Hardware-Unterschiede STE und Mega ST

Wie so oft zählen auch bei Computern die inneren Werte - sprich die eingebaute Hardware. Im Laufe der Zeit wurden die Geräte immer wieder verbessert, so daß sich einige Unterschiede zwischen ST, Mega ST und STE ergeben.

Mega ST:

Außer dem veränderten Gehäuse und der hochwertigen Tastatur sind die Mega STs vor allem durch den Blitter gekennzeichnet. Dieser Baustein beschleunigt Bildschirmausgaben und kann wahlweise im Desktop abgeschaltet werden, wenn sich Probleme mit bestimmten Programmen ergeben sollten.

Zusätzlich haben die Mega STs eine Echtzeituhr, d.h. nach dem Ausschalten geht die eingestellte Uhrzeit nicht mehr verloren. Während der stromlosen Zeit übernimmt eine kleine Batterie die Versorgung der Uhr.

Im Gehäuseinneren befindet sich außerdem ein Erweiterungsanschluß, in den zusätzliche Platinen eingesteckt werden können. Damit wird der Anschluß z.B. eines Grafikadapters für Großbildschirme oder sonstiger Zusatzhardware möglich.

1040 STE:

Schon rein äußerlich zu erkennen sind die beiden neuen Joystickanschlüsse an der linken Seite. Hier können je zwei Steuerknüppel oder Drehregler angeschlossen werden - zusätzlich zu den beiden Anschlüssen unter der Tastatur, die auch schon bei den anderen Modellen vorhanden waren. Für die Nutzung dieser zusätzlichen Joysticks ist allerdings Software nötig, die besonders darauf eingerichtet ist.

Ebenso neu sind die beiden Cinch-Buchsen an der Rückseite, die Stereo-Ton liefern. Sie können den STE jetzt an Ihre Stereo-Anlage anschließen. Intern ist der Rechner sogar darauf eingerichtet, Video-Bilder zu übernehmen und diese ggf. verändert wieder auszugeben. Dazu ist allerdings spezielle Zusatzhardware erforderlich, die zur Drucklegung dieses Buches noch nicht verfügbar war.

Auch in der Bildschirmausgabe hat sich etwas getan, der STE kann nun aus bis zu 4096 Farbtönen auswählen (ST und Mega ST: 512). Außerdem wird das sogenannte Scrolling (Bewegung der Bildschirminhalte) durch den STE jetzt

besser unterstützt, so daß diese Vorgänge schneller ablaufen können. Diese Veränderungen schlagen sich natürlich nur in solchen Programmen nieder, die auch darauf eingerichtet sind, alle anderen müßten aber ohne Probleme weiterarbeiten. Selbstverständlich hat auch der STE einen Blitter.

Intern ist der STE wesentlich freundlicher geworden. Der Speicher von im Normalfall 1 MByte läßt sich durch sogenannte SIMM-Module ganz leicht auf bis zu 4 MByte aufstocken.

Stacy:

Hierbei handelt es sich um die tragbare Version des ST. Die eingebaute Hardware entspricht in den meisten Punkten der des Mega ST, das LCD-Display hat die gleiche Auflösung wie ein Monochrom-Monitor SM 124.

ST-Book:

Für diejenigen, denen selbst der Stacy zu groß oder zu schwer ist, bietet Atari den ST-Book an. Es ist ein normaler ST mit LCD-Display, hat jedoch z.B. kein Diskettenlaufwerk, sondern arbeitet mit sogenannten RAM-Karten. Ein Diskettenlaufwerk läßt sich jedoch extern anschließen. Intern ist das ST-Book noch mit einer Festplatte ausgerüstet, um schnell und sicher viele Daten abspeichern zu können. Im Gegensatz zum Stacy arbeiten die Akkus beim ST-Book jedoch wesentlich länger; außerdem ist das Gerät nur so groß wie ein DIN-A4-Blatt und sehr leicht. Das Display hat eine monochrome Auflösung von 640x400 Pixeln.

Teil B

12. Die eingebaute Software

Wer den ST genauer kennenlernen möchte, muß zuallererst mehr über dessen Innenleben wissen. Das wiederum teilt sich in zwei große Bereiche auf - die Hard- und die Software. Erstere wird in Teil C abgehandelt.

Nicht minder wichtig ist die im ST eingebaute Software. Sie werden staunen, wie viel das ist. Zu sehen bekommen Sie nämlich zunächst nur das Desktop, das ein Teil des GEM ist. Wenn man einen Experten fragt, was GEM eigentlich ist, wird dieser antworten "Bedienoberfläche für Betriebssysteme". Tatsächlich gibt es das GEM auch für andere Betriebssysteme, etwa das MS-DOS. Wenn also GEM nicht das Betriebssystem ist, wo ist es? Ganz einfach: Es steckt im gleichen ROM wie das GEM und nennt sich TOS (Tramiel Operating System). Aber so weit sind wir noch nicht.

Das GEM läßt sich noch weiter unterteilen. Neben dem Desktop (auch die Shell genannt) gibt es noch das AES (Application Environment System) und das VDI (Virtual Device Interface). Es stellt für alle Programme so nützliche Dinge wie Menübäume, Fenster usw. bereit (genauer: Funktionen zur Manipulation derselben). Das AES bedient sich dabei des VDI, damit es alle Ausgaben unabhängig vom verwendeten Gerät mit den gleichen Befehlen machen kann. Das VDI hat dann die Aufgabe, diese Befehle auf das gerade verwendete Gerät umzurechnen. Zu diesem Zweck ist das VDI ebenfalls zweigeteilt: das GIOS (Graphic Input/Output System) und das GDOS (Graphic Device Operating System). Letzteres teilt dem VDI im Einzelfall mit, wie denn nun konkret ein Gerät zu bedienen ist. Soweit es nicht schon im ROM vorhanden ist (für die Standardgeräte), muß es jeweils beim Booten von Diskette oder Festplatte geladen werden.

Das TOS besteht aus drei Teilen. Das GEMDOS ist das eigentliche Betriebssystem mit Funktionen zur Behandlung von Dateien, einfachen Bildschirmausgaben usw. GEMDOS bedient sich des BIOS (Basic Input/Output System) und des XBIOS (eXtended BIOS), die für eine Umsetzung der GEMDOS-Funktionen in kleinste Zugriffe auf die jeweiligen ICs sorgen.

Eine ganze Masse, was? Alles zusammen können Sie noch einmal in Abbildung 12.1 bewundern.

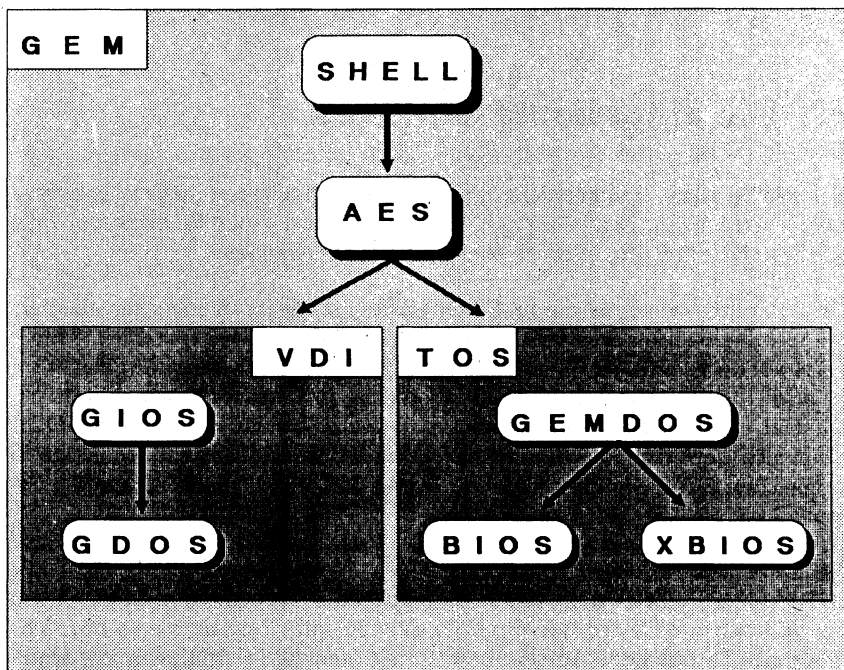


Abb. 12.1: Die Software im ROM des ST

Natürlich arbeiten alle Softwarekomponenten zusammen. GEMDOS benutzt die (X)BIOS-Funktionen, das GEMDOS wird vom VDI benutzt und dieses wiederum vom AES. Ihre Anwendungsprogramme nutzen schließlich das AES.

13. Programme unter GEM und TOS

Wer für den Atari ST Programme schreiben möchte, muß sich - wie unter fast allen Betriebssystemen - an gewisse Konventionen halten. Keine Angst, das ist nicht so kompliziert wie es sich zunächst anhört.

13.1 TOS-Programme

Beginnen wir mit Programmen, die ohne die GEM-Funktionen auskommen. Wird ein solches Programm gestartet, laufen eine ganze Reihe von Vorgängen ab.

Zum Laden eines Programms bedient sich das DESKTOP des Funktionsaufrufes PEXEC. Haben Sie ein Programm mit der Maus angewählt, so wird ein entsprechender PEXEC-Aufruf mit dem Dateinamen und eventuellen, per Dialog-Box eingegebenen, Parametern ausgeführt. Das GEMDOS bekommt durch den Aufruf die Kontrolle und sucht zunächst freien Speicher. Was aber ist "freier Speicher"? Da jedes Programm im Atari seinen Speicherbedarf anmeldet, ist der freie Speicher der Bereich, der von keinem Programm beansprucht wird. An den Anfang des freien Speichers, der sogenannten TPA, wird dann eine BASEPAGE aufgebaut. Diese BASEPAGE ist 256 Bytes (\$100 Bytes) groß und enthält spezielle Informationen über das geladene Programm. Den Aufbau der BASEPAGE entnehmen Sie bitte der folgenden Tabelle.

Offset	Größe	Bezeichnung	Funktion
0x00	LONG	p_lowtpa	Pointer auf den Anfang der Base Page
0x04	LONG	p_hitpa	Pointer auf das Ende des freien Speichers
0x08	LONG	p_tbase	Pointer auf Anfang des Programms (Text Segment)
0x0c	LONG	p_tlen	Größe des Programms (Text-Segment)
0x10	LONG	p_dbase	Pointer auf Anfang des Data-Segments
0x14	LONG	p_dlen	Größe des Data-Segments
0x18	LONG	p_bbase	Pointer auf Anfang des BSS-Segments
0x1c	LONG	p_len	Größe des BSS-Segments
0x20	LONG	p_dta	Pointer auf DTA-Buffer
0x24	LONG	p_parent	Pointer auf Parent's Base Page (reserviert)
0x28			
0x2c	LONG	p_env	Pointer auf Environment-String
0x30 *	BYTE	stdin-Handle	Handle für Standard-Eingabe
			...

...			
0x31 *	BYTE	stdout-Handle	Handle für Standard-Ausgabe
0x32 *	BYTE	aux:-Handle	Handle für Ein-/Ausgabe über AUX:
0x33 *	BYTE	prn:-Handle	Handle für Ausgabe über PRN:
0x37 *	BYTE	drive	aktuelles Laufwerk (Dsetdrv(), Dgetdrv())
0x40 *	BYTE		Tabelle für 16 (32?) Drives für Dateiverwaltung
0x68 *	LONG	D0-Save	enthält nach Trap Return-Wert der Funktion
0x6a *	LONG	A3-Save	A3 vor dem GEMDOS-Trap
0x70 *	LONG	A4-Save	A4 vor dem GEMDOS-Trap
0x74 *	LONG	A5-Save	A5 vor dem GEMDOS-Trap
0x78 *	LONG	A6-Save	A6 vor dem GEMDOS-Trap
0x7c *	LONG	A7-Save	Userstack-Zwischenspeicher
0x80	BYTE	cmdcnt	Anzahl der Zeichen in der Commandline
0x81	BYTE	cmdlin	Commandline

Der Bereich zwischen 0x30 und 0x7f wird vom Betriebssystem verwaltet. Sie als Programmierer sollten diesen Bereich nicht benutzen.

Wird die Basepage vom System aufgebaut, so sind zunächst viele Einträge noch nicht belegt. Erst nach dem Laden des Programms direkt hinter die Basepage stehen die Daten zur Verfügung und werden an den entsprechenden Stellen eingetragen. Die erforderlichen Daten wie z.B. Größe der verschiedenen Datenbereiche finden sich mit Datei-Header, der für ein unter GEMDOS lauffähiges Programm einen festgelegten Aufbau hat. Konkret sieht der Datei-Header wie folgt aus:

- 1 WORD \$601A als Kennzeichen für ein relocatibles File mit kontinuierlichem Text-, Data- und BSS-Bereich
- 1 LONG Größe des Text-Bereichs in Byte, das eigentliche Programm
- 1 LONG Größe des Data-Bereichs in Byte, die initialisierten Variablen wie Strings oder Konstanten
- 1 LONG Größe des BSS-Bereichs in Bytes für Variablen, die erst im Programmverlauf mit einem Wert versorgt werden
- 1 LONG Größe der (optionalen) Symbol-Tabelle in Bytes
- 1 LONG reserviert
- 1 LONG reserviert
- 1 WORD reserviert

Der Programmierer hat ja in der Regel keinen Einfluß darauf, in welchem Speicherbereich sein Programm läuft, da PEXEC selbsttätig freien Speicher ermittelt und das Programm an den Anfang dieses Bereichs lädt. Bei den klassischen 8-Bit-Betriebssystemen muß statt dessen beim Laden die gewünschte Adresse angegeben werden. Auf diese Weise kann es natürlich leicht zu Überschneidungen mit anderen Programmen kommen, die denselben Speicherbereich beanspruchen. Diese Überschneidungen lassen sich unter GEMDOS bei "richtiger" Programmierung vermeiden. Statt der Angabe einer festen Lade-Adresse wird das Programm so assembliert, daß die relevanten Adressen erst durch den Lader, also das PEXEC, ermittelt und in das Programm eingetragen werden. Dazu enthält die Programm-Datei noch eine weitere Tabelle. Über diese Tabelle werden die benötigten Offsets auf die zu verändernden Adressen in der Programm-Datei errechnet. Der Anfang der Tabelle für die Reloizierung liegt direkt hinter der Symboltabelle. Ist diese nicht vorhanden, dann liegt die Reloizertabelle direkt hinter dem DATA-Bereich, da der BSS-Bereich nicht mit auf der Diskette abgespeichert ist. Das ist ja für die nicht vorbesetzten Variablen nicht erforderlich. Sie werden nach dem Programmstart automatisch auf 0 gesetzt.

Nach dieser Arbeit wird das Programm gestartet. Es ist jetzt ein Child (Kind) des Programms, das es aufgerufen hat. Das aufrufende Program wird als Parent bezeichnet. Dieser Begriff bedeutet "Eltern" in der Einzahl, etwas, das es in der deutschen Sprache nicht gibt. Für uns erschwerend kommt hinzu, daß "Programm" weder männlich noch weiblich ist und aus diesem Grund weder Vater noch Mutter eine geeignete Bezeichnung darstellt. Wir bleiben besser bei den englischen Bezeichnungen Parent und Child und überlassen den Sprachwissenschaftlern die Lösung dieses Problems.

Wenden wir uns statt dessen dem soeben geschaffenen Child zu. Dieser Prozeß hat mit seinem Start fast den gesamten freien Speicher zur Verfügung bekommen. Die erste Aktion sollte darum in jedem Programm sein, den tatsächlich benötigten Speicherbedarf zu bestimmen und den Rest an das GEMDOS zurückzugeben. Und wie ermittelt man den Speicherbedarf? Ganz einfach, wenn man weiß, wie's geht.

Nach dem Start des Programms findet man auf dem Stack die Adresse der Base Page. Innerhalb der Base Page stehen aber alle Daten des Programms, die zur Berechnung des Speicherbedarfs nötig sind. Diese Daten sind p_tlen, p_dlen und p_blen. Addiert man die Werte, so hat man den Bereich, den das Programm tatsächlich benötigt. Zusätzlich muß man Platz für den Stack reservieren, der auch im geschützten Speicher liegen muß.

Wenn man einmal mit einem Disassembler den Anfang eines Programms analysiert, so findet man sehr häufig die folgende oder eine ähnliche Sequenz:

move.l	a7,a5	Stack retten um base page zu bestimmen
move.l	4(a5),a5	jetzt haben wir die base page in a5
move.l	\$c(a5),d0	in d0 steht die Länge des Text segments
add.l	\$14(a5),d0	dazu addieren wir die Länge des data-
add.l	\$1c(a5),d0	und die Länge des bss-Segments
add.l	#\$500,d0	dazu noch Platz für den Stack, hier lieber etwas mehr als ein zu kleiner Stack
move.l	d0,d1	
add.l	a5,d1	Länge + Adresse base page
and.l	#-2,d1	damit wird sicher gestellt, daß der Stack auf einer geraden Adresse beginnt
move.l	d1,a7	jetzt liegt der Stack wie gewünscht
move.l	d0,-(sp)	Größe des zu reservierenden Bereichs
move.l	a5,-(sp)	ab wo zu reservieren (base page)
clr.w	-(sp)	dummy
move.w	#\$4a,-(sp)	setblock-Funktionsnummer
trap	#1	gemdos aufrufen
add.w	#12,sp	und den Stack aufräumen

Dieser Programmteil übernimmt alle Aufgaben, wie sie vom GEMDOS gefordert werden. Wenn Sie mit einer höheren Programmiersprache wie Pascal, C oder BASIC arbeiten, müssen Sie sich darum natürlich nicht kümmern - das hat der Compiler bereits für Sie erledigt. Nachdem das GEMDOS den zur Verfügung stehenden Speicher entsprechend reduziert hat, kann das Programm so richtig losgehen. Machen Sie aber auf keinen Fall den Stack-Bereich zu klein. Sollte der Stack einmal zu groß werden und in Ihren Variablenbereich hineinlaufen, dann wird die Fehlersuche zum ganz besonderen Vergnügen. Diese Fehlerursache ist in der Regel nur sehr schwer zu finden. Dieser Ratsschlag gilt besonders auch für Anwender des DRI-C-Compilers (doch, die gibt es immer noch!). Dieser nur leidlich an GEMDOS angepaßte Compiler verwaltet nämlich alle zusätzlichen Datenbereiche zwischen dem Ende des BSS-Bereichs und dem Stack. Dazu gehören auch Buffer-Bereiche für C-Funktionen wie fopen() oder malloc(). Mit einem ausreichend bemessenen Stack (mehrere 10 KByte) kann man besonders auch mit dem malloc() und calloc() des DRI-Compilers gut arbeiten. (Achtung: malloc() ist eine Funktion der Compiler, Malloc() dagegen eine GEMDOS-Funktion.)

Und was ist mit dem freigegebenen Speicher? Der steht dem GEMDOS für weitere Pexec-Aufrufe zur Verfügung. Der Child-Prozeß hat jetzt keine Zu-

griffsberechtigung mehr auf den freigegebenen Speicher. Man sollte es auch tunlichst unterlassen, den freien Speicher ohne weitere Maßnahmen zu benutzen. Wenn man z.B. Daten irgendwo in diesem Bereich ablegt, dann kann es unvermittelt passieren, daß die Daten "übergebügelt" werden. Verschiedene Funktionen des GEMDOS, aber auch des VDI oder AES reservieren sich nämlich bei Bedarf mittels Malloc Speicher und legen Daten in dem erhaltenen Bereich ab. Wenn Sie Ihre Daten nicht entsprechend geschützt haben, so stehen die Chancen nicht schlecht, daß Sie die Daten verlieren.

Wenn Ihnen der zur Verfügung stehende Speicher nicht reicht, dann können auch Sie mit Malloc Speicher vom Betriebssystem anfordern. Nach dem Aufruf erhalten Sie die Anfangsadresse des reservierten Bereichs. In diesem Bereich können Sie sicher sein, daß Ihnen kein anderer Prozeß dazwischenfunk.

Über jede Speicheranforderung führt das GEMDOS genau Buch. Dazu verwaltet es zwei Listen, in denen die Adressen und Größen der belegten und freien Speicherbereiche vermerkt werden. Zusätzlich wird auch der "Besitzer" der belegten Bereiche in der Belegt-Liste vermerkt. Dazu wird die Basepage-Adresse des anfordernden Prozesses gespeichert. Damit ist ein begrenzter Schutz vor fehlerhafter Speicherfreigabe gegeben. Zusätzlich wird diese Information jedoch benötigt, um den von einem Programm belegten Speicherplatz beim Programmende wieder korrekt frei zu geben.

Leider sind die Belegt- und Frei-Listen in ihrer Größe festgelegt. Mehr als etwa 280 bis 290 aufeinanderfolgende Malloc-Aufrufe werden mit einem schnöden "Kein Speicher mehr frei" quittiert. Diese Werte beziehen sich auf die TOS-Version 1.0 vom 6.2.86. Die in den MEGA-STs verwendete Version 1.2 vom 22.4.87 ist in dieser Hinsicht etwas großzügiger. Hier sind maximal etwa 800 Malloc-Aufrufe möglich. Das hört sich schon nach einer ganzen Menge an, kann aber für viele Anwendungen noch zu wenig sein. Darum sollte man bei Bedarf den Speicher in relativ großen Portionen anfordern und sich eigene Routinen zur Verwaltung schreiben.

Wird ein einmal allozierter Speicher nicht mehr benötigt, dann geben Sie ihn am besten mit Mfree wieder zurück. Dann kann er von anderen Prozessen wieder verwertet werden. Dabei empfiehlt es sich, den Speicher nach Möglichkeit in der umgekehrten Reihenfolge freizugeben, in der er angefordert wurde. Nur so werden die freigegebenen Bereiche zuverlässig wieder zu einem großen freien Bereich zusammengefaßt.

Wird ein Programm mit Pterm0 oder Pterm beendet, dann wird, wie schon erwähnt, automatisch der vom Programm belegte Speicher freigegeben. Auch mit Malloc zusätzlich reservierter Speicher wird freigegeben. Zusätzlich wer-

den noch die eventuell geöffneten Dateien geschlossen. Dann wird die Kontrolle wieder an den Parent übergeben, der daraufhin wieder zum Child wird.

Wenn man sich die geschilderten Vorgänge einmal genau überlegt, so kommt man unwillkürlich zu dem Schluß, daß es (mindestens) einen "Vollwaisen", einen Prozeß ohne "Parent" geben muß. Die Kette kann ja schon aus Speicherplatzgründen nicht bis in die Unendlichkeit zurück reichen. Dieser Urprozeß wird bei der Initialisierung des Rechners angelegt, um das GEM zu starten.

In der oben gezeigten Startsequenz haben wir bereits einmal das Betriebssystem aufgerufen. Der Befehl, der uns dazu diente, ist "trap #1". Genauer gesagt tut dieser Maschinenbefehl nichts anderes, als eine sogenannte Exception zu erzeugen. Dabei handelt es sich - in der "Sprache anderer Prozessoren" gesprochen - um Interrupts, also Unterbrechungen des normalen Programmablaufs. Dabei sucht sich der Prozessor aus einem von 256 Vektoren am Beginn des Adreßbereichs eine Zieladresse und springt dann dorthin. Ist das dort stehende Exception-Programm abgearbeitet, geht es zurück an den ursprünglichen Startort.

Normalerweise werden Exceptions durch außergewöhnliche Situationen ausgelöst - etwa durch illegale Maschinenbefehle, ungültige Speicherzugriffe usw. Dabei wird in der Regel ein ROM-Programm gestartet, das die Aufgabe hat, die berühmt-berüchtigten Bomben auf dem Bildschirm zu zeigen.

Einige Exceptions können aber auch "per Hand" ausgelöst werden; das geschieht durch die TRAP-Befehle. TRAP #0 entspricht der Exception 32 usw. Insgesamt 16 Traps (zu deutsch: Fallen) gibt es. Uns interessiert zunächst der mit der Nummer 1, denn dieser startet das GEMDOS.

Um eine Betriebssystemfunktion zu starten, verfährt man dann wie folgt:

1. Alle Parameter in umgekehrter Reihenfolge auf den Stack bringen.
2. Funktionsnummer auf den Stack legen.
3. Trap #1 ausführen.
4. Stack wieder aufräumen.

Das Verfahren ändert sich nicht, wenn Sie Funktionen des BIOS oder des XBIOS starten möchten. Lediglich ein anderer Trap-Befehl muß benutzt werden. Für das BIOS nehmen Sie Trap #13, für das XBIOS Trap #14.

Und natürlich gibt es auch hier wieder die gute Nachricht für die Programmierer, die mit Hochsprachen arbeiten: Sie müssen in der Regel nur dafür sorgen, daß die richtige Library beim Linken mit eingebunden wird, damit die Betriebssystemaufrufe zur Verfügung stehen. Der gesamte Trap-Kram wird dann

automatisch erledigt. In einigen Sprachen (z.B. GFA-BASIC) brauchen Sie nicht einmal für das Linken zu sorgen.

13.2 GEM-Programme

Etwas komplizierter ist es da schon mit den GEM-Programmen. Aufgerufen wird auch das GEM mit einem TRAP-Befehl, genauer mit TRAP #2. Aber schon fangen die Unterschiede an. Befindet sich bei der Ausführung des obigen Befehls der Wert \$73 im Prozessorregister D0, wird das VDI aufgerufen; bei \$C8 in D0 wird das AES gestartet. Parameter werden nicht auf dem Stack übergeben, sondern in fünf speziellen Arrays. Sehen wir uns das einmal genauer an.

Wenn wir eine GEM-Routine in Assembler aufrufen wollen, dann müssen wir zunächst einmal eine ganze Menge Parameter vorbereiten. Diese Parameter werden irgendwo geordnet im Speicher abgelegt und unterteilen sich in 6 Datenblöcke, auch Arrays oder Felder genannt:

- Kontroll-Array (Abkürzung: contr)
- Global-Struktur (Abkürzung: global)
- Eingabe-Array (Abkürzung: intin)
- Eingabepunkte-Array (Abkürzung: ptsin)
- Ausgabe-Array (Abkürzung: intout)
- Ausgabepunkte-Array (Abkürzung: ptsout)

Diese Datenblöcke (alle bestehen jeweils aus einer Anzahl von Bytes) können völlig unabhängig voneinander irgendwo im Speicher liegen und haben die folgenden Funktionen:

a) Das Kontroll-Array

Das Kontroll-Array enthält Parameter allgemeiner Art. In ihm werden sowohl der aufzurufenden Funktion Parameter übergeben als auch diese Funktion Rückgabeparameter hier niederlegen kann. Die einzelnen Parameter besitzen jeweils die Länge von einem Wort (= 2 Byte) und haben die folgenden Funktionen:

Contrl-Feld für VDI-Aufrufe:

- | | |
|--------|---|
| Wort 0 | Nummer (Opcode) der aufzurufenden Funktion. |
| Wort 1 | Anzahl der Punkte im ptsin-Array. Ein Punkt besteht aus 2 Wörtern (x-/y-Koordinate) |

Wort 2	Anzahl der Punkte im ptsout-Array. Ein Punkt besteht aus 2 Wörtern (x-/y-Koordinate).
Wort 3	Anzahl der Wörter im intin-Array.
Wort 4	Anzahl der Wörter im intout-Array.
Wort 5	Unterfunktion-Befehlsnummer.
Wort 6	Gerätekennziffer "handle" (Device-handle).
Wort 7-10	Funktionsabhängige Zusatzinformationen.

Contrl-Feld für AES-Aufrufe:

Wort 0	Nummer (Opcode) der aufzurufenden Funktion.
Wort 1	Anzahl der Punkte im intin-Array.
Wort 2	Anzahl der Punkte im intout-Array.
Wort 3	Anzahl der Wörter im addrin-Array.
Wort 4	Anzahl der Wörter im addrout-Array.

Kontroll – Feld

Op - code	Länge Ptsin - Array	Länge Ptsout - Array	Länge Intin - Array	Länge Intout - Array	Sekundär - Opcode	Handl	Zusatzinformationen

Abb. 13.1: Contrl-Feld für VDI

Bei der Verwendung dieser Tabelle beachten Sie bitte, daß die Wörter 0, 1, 3 und 5 stets vom Programmierer belegt werden müssen. Die Wörter 2 und 4 sind Parameter, die die aufgerufene Funktion zurückgibt. Bei den Wörtern 6-10 können je nach Funktion Werte vom Programmierer oder vom Betriebssystem eingeschrieben werden.

Wort 0 bestimmt, welche Funktion des VDI oder AES überhaupt aufgerufen werden soll. Schließlich wird zum Aufruf einer Funktion immer das gleiche Unterprogramm angesprungen.

Da die untenstehenden Felder "intin", "intout", "ptsin" und "ptsout" von Funktion zu Funktion unterschiedliche Längen haben können, bzw. unterschiedlich viele Parameter enthalten können, wird in dem Kontrollfeld in den Wörtern 1-4 die jeweilige Anzahl von Parametern dieser Felder übergeben.

Wort 6 bestimmt die logische Nummer des angesprochenen Aus- oder Eingegerätes (Bildschirm, Plotter, Disk,...). Diese Nummer wird dem Programmierer einmal (in der Regel am Beginn) im Programm mitgeteilt. Er muß sich

diese Nummer über das gesamte Programm merken, um Kreise, Linien, Punkte etc. auf dem richtigen Ausgabegerät auszugeben.

b) Das intin-Array

Der Name intin kommt von integer input. In diesem Feld übergibt der Programmierer der aufzurufenden Funktion bestimmte Werte wie Farbnummer o.ä., die die Funktion benötigt. Die Anzahl der hier jeweils zu übergebenden Parameter muß im Kontroll-Array gespeichert werden, da sie ja von Funktion zu Funktion verschieden sein kann.

Man gibt diesem Feld normalerweise eine maximale Größe von 128 Wörtern (256 Bytes), um möglichst viel Spielraum zu haben. (Bei einem größeren intin-Array stürzt Ihr Rechner über kurz oder lang ab!)

intin-Feld:

Wörter 0-127 Ganzzahlige Übergabewerte

c) Das ptsin-Array

Dies ist das Feld, in dem der Programmierer der aufzurufenden VDI-Funktion eine je nach Funktion unterschiedlich große Anzahl der x- und y-Koordinaten von Punkten übergibt ("ptsin" kommt von Points input - Punkteingabe). Für jede Koordinate (x oder y) wird hier ein Wort, für jeden Punkt demnach zwei Wörter (= ein Langwort) reserviert. Die Anzahl der hier übergebenen Punkte (Langwörter) muß der Programmierer der VDI-Funktion im Kontrollfeld angeben (s.o.).

ptsin-Feld:

Wörter 0-255 x- und y-Koordinaten von max. 128 Punkten

d) Das intout-Array

Analog zum intin-Array, in dem der Programmierer der Funktion Parameter übergibt, kann hier die aufgerufene Funktion dem Programmierer Werte bzw. Informationen zurückgeben (z.B. gedrückte Taste etc.). Die Funktion gibt dem Programmierer im Kontroll-Array an, wie viele Parameter sie hier zurückgibt.

intout-Feld:

Wörter 0-127 Ganzzahlige Rückgabewerte einer Funktion

e) Das ptsout-Array

Dieses letzte Feld schließlich verwenden verschiedene VDI-Funktionen dazu, dem Programmierer eine Reihe von Punkten zurückzugeben. Sie haben die gleiche Struktur wie im ptsin-Array. Ihre Anzahl trägt die Funktion ebenfalls im Kontroll-Feld ein.

ptsout-Feld:

Wörter 0-11 Punktrückgabe einer VDI-Funktion

Diese fünf Felder dienen also der manchmal recht umfangreichen Parameterüber- und -rückgabe und den GEM-Bibliotheksfunktionen. Wie gesagt können diese Speicherfelder irgendwo im Rechner stehen. Aus diesem Grunde muß der aufrufenden Funktion noch mitgeteilt werden, wo sie sich genau befinden. Diese Aufgabe übernimmt der sogenannte Parameterblock. Er hat folgenden Aufbau:

Parameterblock für VDI-Aufrufe:

Wörter 0/1	Adresse des Kontroll-Arrays
Wörter 2/3	Adresse des intin-Arrays
Wörter 4/5	Adresse des ptsin-Arrays
Wörter 6/7	Adresse des intout-Arrays
Wörter 8/9	Adresse des ptsout-Arrays

Parameterblock für AES-Aufrufe:

Wörter 0/1	Adresse des Kontroll-Arrays
Wörter 2/3	Adresse der Global-Struktur
Wörter 4/5	Adresse des intin-Arrays
Wörter 6/7	Adresse des intout-Arrays
Wörter 8/9	Adresse des addrin-Arrays
Wörter 10/11	Adresse des addrout-Arrays

Dieser Parameterblock kann wiederum irgendwo im Speicher stehen. Seine Adresse wird der Funktion im d1-Register des Prozessors übergeben.

Sie merken schon, leichte Differenzen gibt es schon zwischen VDI und AES. Aufmerksamen Lesern wird aufgefallen sein, daß wir bisher die Global-Struktur nicht näher erläutert haben; auch die Arrays addrin und addrout haben wir mit keinem Wort gewürdigt. Nun, die Aufgabe der letzten beiden ergibt sich fast aus dem Namen. Addrin und addrout bestehen aus Zeigern, die

für die AES-Funktionen benötigt oder von diesen zurückgegeben werden. Jeder Eintrag hier hat also 32 Bits; ansonsten funktionieren sie wie die Brüder intin und intout.

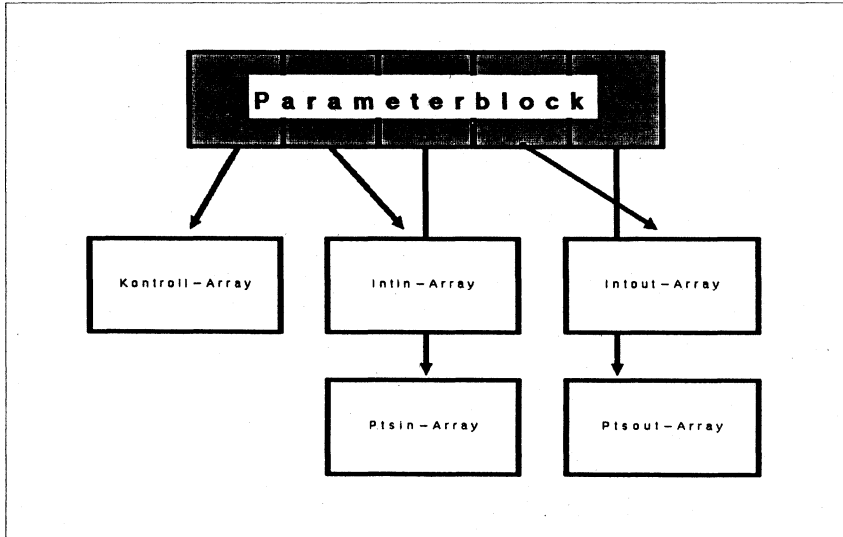


Abb. 13.2: Zusammenhang: Parameterblock <-> Arrays

Die Global-Struktur ist da schon schwieriger. Wohlgedemerk: Es handelt sich um eine Struktur, deren Mitglieder unterschiedliche Größen haben. Die Belegung sieht folgendermaßen aus:

Name	Breite	Inhalt
ap_version	Word	AES-Version
ap_count	Word	Max. gleichzeitig lauf. Prog.
ap_id	Word	Applikationsnummer
ad_privat	Long	frei
ad_ptree	Long	Zeiger auf Baumstruktur
ad_1res	Long	reserviert
ad_2res	Long	reserviert
ad_3res	Long	reserviert
ad_4res	Long	reserviert

Sie merken schon: Bei diesen Werten handelt es sich um globale Angaben (daher der Name) zu einer Applikation. Sie werden durch die AES-Funktion `Appl_init` gesetzt. Ausnahme: Die Adresse der Baumstruktur wird durch die Funktion `RSRC_LOAD` geliefert, die zum Laden der RSC-Dateien dient.

Wie ist nun aber ein Funktionsaufruf überhaupt zu bewerkstelligen? Nun, hier trennen sich die Sprachen C und Assembler. In Assembler müssen vor dem VDI-Funktionsaufruf die folgenden Prozessor-Register wie folgt belegt werden:

d0.l: VDI-Bibliotheksnnummer: \$73

d1.l: Adresse des Parameterblocks

Sind die Parameter-Arrays richtig ausgefüllt und diese beiden Register mit den entsprechenden Werten belegt, dann wird die Funktion mit einem

TRAP #2

ausgeführt.

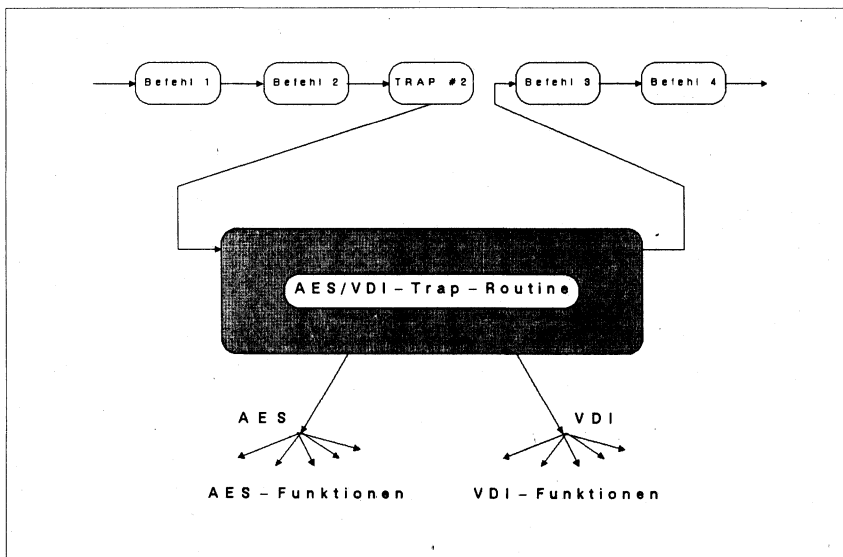


Abb. 13.3: TRAP #2

Nach dem Aufruf können dann eventuelle Rückgabeparameter in den out-Arrays bzw. im Kontroll-Feld nachgesehen werden.

Auch ein GEM-Programm muß natürlich seinen Speicherbedarf beim Betriebssystem anmelden - genau wie ein einfaches TOS-Programm. Das unten abgedruckte Programm dürfte Ihnen daher in einigen Teilen bekannt vorkommen.

Um ein GEM-Programm zu starten, bedarf es noch einiger weiterer Maßnahmen. So muß zunächst einmal das Programm (=Applikation) angemeldet werden. Bei dieser Anmeldung werden diverse Speicher usw. reserviert. Damit kann das Programm auf die AES-Library zugreifen. Der dafür zuständige AES-Library-Befehl heißt `appl_init` (Applikations-Initialisierung) und hat die Funktionsnummer 10. Diese Funktion liefert einen Wert in `intout(0)` zurück, der uns aber nicht weiter interessiert. Er stellt die Kennziffer Ihres Programms dar, die Ihnen von AES zugewiesen wurde, da ja gleichzeitig mehrere Programme (oder: Applikationen) im Speicher sein können. Außerdem werden, wie schon oben erwähnt, die Mitglieder der Global-Struktur initialisiert. Anhand dieser Werte erkennt das AES, von welcher Applikation es aufgerufen wurde.

Am Ende einer Applikation, also Ihres Programms, muß dann folgerichtig die Applikation wieder abgemeldet werden. Dies geschieht mit Hilfe des AES-Befehls `appl_exit` (Funktionsnummer: 19), dem kein Parameter übergeben werden muß. Erst jetzt darf das Programm z.B. ins Desktop zurückkehren.

Um nun aber auch auf die VDI-Befehle zugreifen zu können, müssen Sie Ihre Arbeitsstation öffnen. Das heißt, Sie müssen dem VDI mitteilen, ob Sie auf einem Drucker, Bildschirm oder Diskettenlaufwerk ausgeben wollen usw. VDI lädt dann die entsprechenden Treiber (soweit vorhanden) von Diskette. Wir wollen jedoch nur auf den Bildschirm ausgeben. Dafür existiert eine spezielle Funktion, die sich Open Virtual Screen Workstation (`v_opnvwk`) nennt und den Bildschirm als universelles Ausgabegerät bestimmt. Hier müssen Sie unter anderem in `intin(10)` angeben, ob Sie mit Rasterkoordinaten (`RC`, `intin(10)=2`) oder unter normalisierten Koordinaten (`NDC`, `intin(10)=0`) arbeiten wollen.

Die vollständige Liste der gesamten GEM-Funktionen finden Sie übrigens im Anhang.

Logischerweise müssen Sie diese Arbeitsstation vor Programmende wieder abmelden mit der Funktion `v_clsvwk` (Funktionsnummer: 101).

```

;*****
; ** **
; ** VDI- und AES-Ansteuerung **
; ** per Assemblerprogramm **
; ** **
;*****
;Konstanten definieren:

```



```

gemdos    = 1                ;TRAP-Nummer GEMDOS
xbios     = 14               ;TRAP-Nummer XBIOS
bios      = 13               ;TRAP-Nummer BIOS
text
;Programminitialisierung:
;-----
go:        move.l  sp,a5      ;alten Stackzeiger retten
          lea      stackend,sp ;Stackzeiger auf neuen Stapel
;Speicherplatzberechnung für SETBLOCK (GEMDOS):
          movea.l  4(a5),a5   ;a5 zeigt nun $100 Bytes unter
                               ;Programmstart = Programm-Header
          move.l   $c(a5),d0   ;Länge des Programmbereichs->d0
          add.l    $14(a5),d0  ;plus Länge des
                               ;initialisierten Datenbereichs
          add.l    $1c(a5),d0  ;plus Länge des
                               ;uninitialisierten Datenbereichs
          add.l    #$100,d0    ;plus Länge des Programm-Headers
          move.l   d0,-(sp)    ;d0=Länge des zu reservierenden
                               ;Speicherbereichs als
                               ;Parameter an TOS
                               ;Startadresse des zu reservierenden
          move.l   a5,-(sp)
Speicherbereichs an TOS
          move.w   #0,-(sp)    ;Wort ohne Bedeutung an TOS
          move.w   #$4a,-(sp)  ;Funktionsnummer für SETBLOCK
          trap     #gemdos     ;GEMDOS-Funktion
                               ;SETBLOCK aufrufen
          add.l    #12,sp      ;Stack in Ordnung bringen
          tst.l    d0          ;Fehler?
          bne     error        ;ja => kein Programmaufruf
;Programmaufruf:
          bsr      main        ;Eigentliches Programm aufrufen
;optionales Warten auf Tastendruck:
          move.w   #1,-(sp)    ;Funktionsnummer für
                               ;CONIN (GEMDOS)
          trap     #gemdos     ;GEMDOS-Funktion aufrufen
                               ;(Auf Taste warten)
          addq.l   #2,sp       ;Stack in Ordnung bringen
;Zurück zum Desktop (bzw. zum aufrufenden Programm):
error:     clr.l   (sp)        ;Funktionsnummer für
                               ;TERM (GEMDOS)
          trap     #gemdos     ;GEMDOS-Funktion aufrufen
;Routinen für späteren Aufruf:
;-----
;AES-Funktion aufrufen:
aes:       movem.l d0-d1,-(sp) ;d0 und d1 retten
          move.l   #aespb,d1   ;Adresse AES-Parameterblock
          move.w   #$c8,d0     ;AES-Bibliotheksnnummer
          trap     #2          ;Aufruf
          movem.l  (sp)+,d0-d1 ;d0 und d1 wiederholen
          rts
;VDI-Funktion aufrufen:
vdi:       movem.l d0-d1,-(sp) ;d0 und d1 retten
          move.l   #vdipb,d1   ;Adresse VDI-Parameterblock

```

```

        move.w  #$73,d0          ;VDI-Bibliotheksnnummer
        trap    #2              ;Aufruf
        movem.l (sp)+,d0-d1     ;d0 und d1 wiederholen
        rts
;Hier steht Ihr Programm:
;
;Initialisierung:
;-----
main:    move.w  #10,opcode      ;Opcode appl_init
        clr.w   aintin         ;Anzahl intin-Werte=0
        clr.w   aaddrin       ;Anzahl addrin-Adressen=0
        clr.l   aplresv       ;reservierte Bereiche
        clr.l   ap2resv       ;vorsorglich löschen
        clr.l   ap3resv
        clr.l   ap4resv
        jsr     aes           ;AES-Funktion aufrufen
        move.w  #77,opcode     ;Opcode graf_handle
        clr.w   aintin
        clr.w   aaddrin
        move.w  #5,aintout
        jsr     aes           ;AES-Funktion
        move.w  intout,handle  ;ermittelte Device-Nummer
                                ;nach handle

        move.w  #100,opcode    ;Opcode open_vwork
        clr.w   sptsin
        move.w  #11,sintin     ;11 Werte im intin-Array
        move.w  #1,intin
        move.w  #1,intin+2
        move.w  #1,intin+4
        move.w  #1,intin+6
        move.w  #1,intin+8
        move.w  #1,intin+10
        move.w  #1,intin+12
        move.w  #1,intin+14
        move.w  #1,intin+16
        move.w  #1,intin+18
        move.w  #2,intin+20    ;Rasterkoordinaten einstellen
        jsr     vdi           ;VDI-Funktion aufrufen
        move.w  #3,opcode     ;Opcode clear workstation
                                ;(v_clrwk)

        clr.w   sintin
        clr.w   sptsin
        jsr     vdi           ;Aufruf Bildschirm löschen
;eigentliches Hauptprogramm:
;-----
haupt:
;
;
;
;
;Terminierung:
;-----
        move.w  #101,opcode    ;close virtual screen

```

```

                                ;workstation
        clr.l    sptsin
        clr.l    sintin
        jsr     vdi                ;Aufruf
        move.w   #19,opcode        ;appl_exit
        clr.l    aintin
        clr.l    aaddrin
        jsr     aes
        rts                        ;mit RTS zurück

;Und nun die Datenbereiche:
;-----
;Initialisierte Daten:
data
align.w
;AES-Parameterblock:
aesp:  dc.l    contrl
        dc.l    global
        dc.l    intin
        dc.l    intout
        dc.l    aaddrin
        dc.l    addrout
;VDI-Parameterblock:
vdip:  dc.l    contrl
        dc.l    intin
        dc.l    ptsin
        dc.l    intout
        dc.l    ptsout
;AES- und VDI-Parameter-Arrays:
;Kontroll-Array:
contrl:
opcode: ds.w    1                ;Funktionsnummer
aintin:                ;Anzahl aes-intin-Werte
sptsin:  ds.w    1                ;Anzahl vdi-ptsin-Punkte
aintout:                ;Anzahl aes-intout-Werte
sptsout: ds.w    1                ;Anzahl vdi-ptsout-Punkte
aaddrin:                ;Anzahl aes-aaddrin-Adressen
sintin:  ds.w    1                ;Anzahl vdi-intin-Werte
aaddrout:                ;Anzahl aes-aaddrout-Adressen
sintout: ds.w    1                ;Anzahl vdi-intout-Werte
subcode: ds.w    1                ;Unterfunktion-Opcode
handle:  ds.w    1                ;Gerätekenziffer
        ds.w    4                ;sonstige Verwendung

;Global-Array (nur AES):
global:
apvers: ds.w    1
apcount: ds.w    1
apid:   ds.w    1
appriv: ds.l    1
apptree: ds.l    1
aplresv: ds.l    1
ap2resv: ds.l    1
ap3resv: ds.l    1
ap4resv: ds.l    1

```

```
;intin-Array:
intin:  ds.w   128
;ptsin-Array:
ptsin:  ds.w   128*2
;intout-Array:
intout: ds.w   128
;ptsout-Array:
ptsout: ds.w   12
;addrin-Array (nur AES):
addrin: ds.w   128
;addrout-Array (nur AES):
addrout: ds.w   128
;hier folgen Ihre initialisierten Daten ...

;Uninitialisierte Daten:
bss
align.w
;Stack reservieren:
stkstart: ds.l   300
stackend: ds.l   1
;Nun folgen Ihre uninitialisierten Daten...
end
```

Der hier verwendete Assembler ist der PROFI-ASS von DATA BECKER. Wenn Sie einen anderen Assembler besitzen, so sollte es kein großes Problem sein, dieses Programm entsprechend umzuschreiben.

Zwei kleine Besonderheiten des PROFI-ASS: Das Assembler-Kommando ALIGN.W heißt bei vielen anderen Assemblern auch EVEN. Ferner werden die Konstantendefinitionen am Programmanfang hier mit einem Gleichheitszeichen getätigt:

```
gemdos = 1
```

Bei anderen Assemblern wird dafür oft geschrieben:

```
gemdos EQU 1
```

Weiterhin verwendet PROFI-ASS ein Semikolon ";" als Kennzeichen für Kommentar. Hier gibt es Assembler, die sich der C-Syntax annähern und statt dessen ein Sternchen "*" benötigen.

Sie sehen, wir haben zum besseren Verständnis noch einmal die bereits oben besprochenen Programmstartkonventionen abgedruckt. Dies ist hier das letzte Mal passiert. In Zukunft werden wir nur noch den main-Teil und die zu den Datenbereichen hinzuzufügenden initialisierten und uninitialisierten Daten abdrucken. Die folgenden Programme sind also nicht für sich allein lauffähig, Sie müssen vielmehr die Start- und Endkonventionen hinzufügen!

Zu dem main-Teil des obigen Programms:

Hier sehen wir zum ersten Mal, wie wir eine AES- bzw. VDI-Funktion von Assembler aus aufrufen. Nehmen wir einmal den ersten Funktionsaufruf (`appl_init`). Zunächst einmal speichern wir die Nummer (hier 10) der aufzurufenden Funktion im ersten Kontroll-Array-Speicher, wir haben ihn `opcode` genannt. Dann teilen wir der Funktion mit, wie viele "intin"- und "addrin"-Werte (bzw. `ptsin`-Werte bei VDI-Funktionen) der Funktion übermittelt werden. Dies sind hier keine, also speichern wir in den beiden Speichern des Kontroll-Arrays den Wert Null. Da bei der AES-Funktion noch vier weitere Werte im global-Array für zukünftige Erweiterungen reserviert sind, löschen wir diese vorsorglich. Jetzt kann der Aufruf erfolgen.

Bevor wir - wie oben besprochen - unsere Arbeitsstation öffnen, müssen wir zunächst noch herausbekommen, welche Gerätekennziffer der Bildschirm besitzt, denn die müssen wir der Funktion `open_vwork` übergeben. Diese Gerätekennziffer erhalten wir durch den Aufruf der AES-Funktion `graf_handle` mit der Nummer 77.

Da uns die Funktion fünf Werte zurückgibt, wir auch einen zurückbekommen möchten und es sich um eine AES-Funktion handelt, speichern wir noch den Wert 5 ins Kontroll-Array als Wert für die Anzahl der `intout`-Werte. Dann folgt der Funktionsaufruf. Im ersten "intout"-Speicher steht dann die Gerätekennziffer (Device-Nummer oder `handle`), die wir sofort ins VDI-Kontroll-Array speichern, da sie jeder VDI-Funktion im Kontroll-Array übergeben werden muß, damit sie weiß, auf welches Gerät sie Grafik ausgeben soll etc. Diese Kennziffer bleibt dann hier im Kontroll-Array über das gesamte Programm erhalten und braucht - sofern wir nur mit einem einzigen Gerät arbeiten - nicht wieder geändert zu werden.

Jetzt können wir endlich unsere erste VDI-Funktion, nämlich `open_vwork` (Nummer: 100), aufrufen. Die Nummer geht wieder ins Kontroll-Array (`opcode`). Da wir keine Punkte übergeben, wird die Anzahl der im "ptsin"-Array angegebenen Koordinaten auf Null gesetzt. Da der Funktion elf Werte im "intin"-Array übergeben werden, geht diese Zahl in den entsprechenden Speicher des Kontroll-Arrays (`sintin`). Dann folgt die Belegung der elf "intin"-Werte. Die ersten zehn ("intin"(0)-"intin"(9)) sollten eigentlich eine Funktion haben, haben sie aber nicht - wir speichern einfach eine 1 hinein. Bleibt der elfte Wert. Er bestimmt, ob wir mit RC oder NDC arbeiten wollen:

=0 => Normalisierte Koordinaten (NDC)

=2 => Rasterkoordinaten (RC)

Wir wollen der Einfachheit halber mit RC hantieren, also packen wir vor dem Funktionsaufruf eine 2 in diesen Speicher.

Damit hätten wir eigentlich unsere VDI-Startkonventionen erledigt. Wir löschen aber noch eben den gesamten Bildschirm durch die Funktion "v_clrwk" (Clear workstation).

Jetzt könnte unser GEM-Programm starten. Nach der Beendigung dieses Programms bleibt uns nur noch die VDI-Terminierung, nämlich - wie besprochen - das Schließen der Arbeitsstation und das Abmelden der Applikation.

Prinzipiell muß sich die Programmiersprache C natürlich ebenfalls an die Parameterübergabe und den Funktionsaufruf halten, wie sie oben beschrieben wurde. Deshalb sollten Sie den obigen Abschnitt auch als C-Programmierer durchgearbeitet haben, wobei Sie natürlich die Assembler-spezifischen Dinge außer acht lassen können. Dem C-Programmierer wird die Sache jedoch meist durch C-Bibliotheken erleichtert. In diesen Bibliotheken werden die Parameter-Arrays und der Parameterblock bereits definiert. Der Sprung auf die Funktion wird dort durch ein kleines eingelinktes Assemblerprogramm realisiert, das unter anderem den notwendigen TRAP-Sprung enthält. Zudem erhält jede Funktion ihren eigenen Namen (mit denen wir stets arbeiten, wenn wir eine Funktion benennen), unter C brauchen Sie sich also nicht mit den Funktionsnummern herumzuschlagen, müssen allerdings die verschiedenen Namen im Gedächtnis halten. Auch die Variablen und Übergabeparameter können evt. Namen haben (z.B. handle für die Gerätekenzziffer oder pxyarray[] für Punkteangaben), damit Sie sich nicht merken müssen, in welchem Parameter-Array wo welche Parameter übergeben werden.

Die Parameterübergabe erfolgt dann beim Funktionsaufruf, z.B.:

```
v_pline(handle, count, pxyarray)
set_color(handle, color_index)
```

Im ersten Fall handelt es sich um eine Funktion, die Linienzüge beliebiger Länge zeichnet. Das Ausgabegerät wird - wie bei allen VDI-Funktionen - durch "handle" übergeben, handle ist natürlich nur das sechste Wort des Kontroll-Feldes. Die einzelnen Linienstart- und -endpunkte werden dann bekanntlich im "ptsin"-Array abgelegt, das in diesem Falle den Namen pxyarray[] bekommt (genauer: pxyarray[2*count], da es sich ja hier um ein Feld handelt). "count" ist die Anzahl der Punkte im pxyarray[] und wird dann natürlich in Wort 1 des Kontroll-Arrays übergeben (oben ptsin genannt). Die Funktions- und Variablennamen sind weitgehend standardisiert und damit unter den einzelnen Bibliotheken der verschiedenen C-Compiler identisch.

Muß doch einmal direkt auf die Parameter-Arrays zugegriffen werden, hier ihre C-Namen:

contrl[0] bis contrl[10]	- Kontroll-Array
intin[0] bis intin[127]	- intin-Array
ptsin[0] bis ptsin[127]	- ptsin-Array
intout[0] bis intout[127]	- intout-Array
ptsout[0] bis ptsout[11]	- ptsout-Array

Diese Arrays, genauso wie alle verwendeten Variablen (nicht aber die Funktionen), müssen in jedem Fall am Programmanfang als "integer" (int) deklariert werden. Natürlich müssen Sie die entsprechenden VDI-Bibliotheksfunktionen mit einbinden. (Beim C-Compiler des Entwicklungssystems z.B.: apstart, vdibind, aesbind, osbind, libf - Den genauen Compiler-Aufruf entnehmen Sie bitte der Anleitung Ihres C-Compilers bzw. Linkers.).

Um den Parameterblock und die Registerübergabe brauchen Sie sich in C nicht zu kümmern, das machen die Bibliotheksfunktionen selbst. Trotzdem auch hier einmal die C-Namen des Parameterblocks:

vdipb[0] bis vdipb[4]	- Parameterblock
-----------------------	------------------

Um Ihnen das ganze auch einmal zu demonstrieren, soll hier ein kurzes, aber unter Umständen nützliches Stückchen C-Code vorgestellt werden, das Sie in Ihren eigenen Programmen einsetzen können. Dieses Programmchen zeigt, wie Sie ein Accessory wecken können. Hierbei ist natürlich unterstellt, daß Sie CONTROL.ACC auch geladen haben, denn nach diesem Accessory wird hier gesucht.

Nützlich wäre solch ein Aufruf, wenn Sie in einem Programm plötzlich feststellen, daß die Systemzeit unmöglich stimmen kann. Rufen Sie einfach diese kleine Routine auf (die innerhalb Ihres Programms natürlich nicht "main" heißen darf), und schon erscheint das bekannte Control-Fenster, in dem Sie Ihre Einstellungen vornehmen können.

Die FOR-Schleife ist notwendig, weil Sie ja nicht wissen können, welchen Wert CONTROL.ACC bei seinem MENU_REGISTER zurückbekommen hat.

```
#include <stdio.h>
#include <gemdefs.h>
extern int gl_apid, global[];
main()
{
    register int i, id;
    int msg_buff[8];
```

```
appl_init();
id = appl_find("CONTROL ");
graf_mouse(ARROW, NULL);
if (id >= 0)
{
    for (i = 1; i < 7; i++)
    {
        msg_buff[0] = AC_OPEN;
        msg_buff[1] = gl_apid;
        msg_buff[2] = 0;
        msg_buff[3] = 0;
        msg_buff[4] = i;
        msg_buff[5] = 0;
        msg_buff[6] = 0;
        msg_buff[7] = 0;
        appl_write(id, 16, msg_buff);
    }
}
appl_exit();
}
```

Wie funktioniert das nun im Detail? Recht einfach. Die AES-Funktion dient dazu, einer Applikation eine Nachricht zu schicken - genau wie sie durch EVENT_MESAG oder ähnliche Funktionen empfangen werden kann. Unser Programm ahmt hier quasi das AES nach und schickt dem Kontrollfeld genau die Nachricht, die sonst beim Anklicken des entsprechenden Menüpunktes durch das AES erzeugt würde. Das Ergebnis sehen Sie selbst. Weil wir - wie oben erwähnt - nicht wissen, welchen Menüeintrag das Kontrollfeld besitzt, greifen wir zu einem Trick. Für jedes denkbare Accessory wird eine Nachricht abgeschickt. Aber nur beim Kontrollfeld paßt die ID, die im Aufruf von "appl_write" mitgegeben wurde, auch zum Programm. Deshalb wird auch nur dieses Accessory auf unsere Nachricht reagieren.

13.3 Accessories

Wir kommen jetzt zum Salz in der Atari-Suppe: Accessories. Die nützlichen Helfer im linken Teil des Menübaumes möchte wohl kein ST-Besitzer mehr missen. Verständlich, daß viele Programmierer gerne selbst ein solches Programm schreiben möchten. Das ist auch nicht besonders schwer!

Für das GEM ist ein Accessory zunächst nichts anderes als ein ganz normales Programm. Alle im Speicher befindlichen Programme werden vom AES in einer Liste verwaltet. Immer, wenn das AES von einer Applikation aufgerufen wird, sieht es in dieser Liste nach, für welche andere Anwendung eine Nachricht zur Verfügung steht. Dabei könnte es sich zum Beispiel um die Nachricht

"Accessory aufgerufen" handeln. Das entsprechende Accessory erhält Rechenzeit zugeteilt und kann dann reagieren.

Auch der Programmierer spürt nur wenige Unterschiede. Klar: Ein Accessory darf keine Menüleiste haben, denn die wird ja schon von der "Vordergrund-Applikation benutzt, und zwei Leisten wären wirklich zuviel des Guten. Ein wesentlicher Unterschied tritt beim Anmelden der Applikation durch `Appl_Init()` auf. Die sonst nicht mehr benötigte Applikationsnummer (`Appl_Id`) muß unbedingt zwischengespeichert werden, da sie später noch gebraucht wird. Der nächste Aufruf ist dann die Funktion `Menu_Register()`. Sie trägt den Namen des Accessories in die Menüleiste ein (siehe AES-Beschreibung). Accessory anmelden

Das war es dann auch schon fast. Alles was das Accessory nun noch zu tun hat, ist das Warten auf Nachrichten. Dazu muß in einer Endlosschleife die Funktion `Event_Multi()` oder `Event_Mesag()` aufgerufen werden. Wird das Accessory vom Benutzer angeklickt, kommt hier die Message `AC_OPEN` zurück. In diesem Fall kann aus der Endlosschleife in ein Unterprogramm verzweigt werden, das die gewünschte Aktion (z.B. Darstellen einer Dialogbox) ausführt. Danach geht es zurück in die Schleife.

Eine andere Nachricht ist `AC_CLOSE`. Sie wird gesendet, wenn ein Programm die Menüleiste löscht. In diesem Fall müssen alle Accessories den Dienst einstellen, d.h. evtl. noch offene Fenster schließen.

Nur eines darf ein Accessory niemals tun: zum Ende kommen. Das ist im AES schlicht nicht vorgesehen. Daher auch die oben erwähnte Endlosschleife.

In BASIC müssen Sie natürlich einen Compiler benutzen, um ein lauffähiges Programm zu erhalten.

Bevor Sie jetzt Ihr GFA-BASIC anwerfen, um ein eigenes Accessory zu programmieren, muß ich Ihnen leider die Freude etwas vermiesen. Bis einschließlich Version 2.0 kann in GFA-BASIC ein Accessory nur unter großem Aufwand erstellt werden. Grund: Sie haben keine direkte Möglichkeit, die `Appl_Id` zu erfahren, und die wird nun einmal für das Eintragen im Menübaum mittels `Menu_Register` gebraucht. Ab Version 3.0 ist aber auch das möglich.

13.4 Objekte

Wer GEM-Programme schreiben möchte, kommt zwangsläufig mit Objekten in Berührung. Wenn Sie Ihren C-Compiler fragen könnten, was ein Objekt ist, würde der antworten, daß es sich dabei um eine Speicherstruktur handelt. In dieser Speicherstruktur stehen Informationen, die das GEM zum Zeichnen des Objektes benötigt. Alles, was das GEM auf den Bildschirm bringt, ist ein Objekt. Also zum Beispiel die Dialogboxen, die Fenster, die Schieber in den Fenstern, die Menübäume, die Icons und und und...

Wie die Erwähnung der Schieber bereits angedeutet hat, können Objekte in anderen Objekten vorkommen. Genauer gesagt: Objekte können Unterobjekte besitzen, die ihrerseits wieder Unterobjekte besitzen dürfen usw. Das funktioniert so ähnlich wie ein Verzeichnisbaum. Folgerichtig spricht man auch genauer von Objektbäumen. Wenn Sie eine einfache Alarmbox betrachten, fällt Ihnen vielleicht auf, daß es aus drei Objekten besteht: dem Rahmen, dem Button darin und dem Text. Das Objekt an der Wurzel des Baums ist der Rahmen; in dessen Speicherstruktur steht ein Zeiger. Dieser zeigt auf das Objekt Text. Dieses Unterobjekt besteht natürlich aus einer eigenen Struktur. In dieser gibt es wiederum einen Zeiger, und der verweist auf das Button-Objekt, das unter dem Aktenschrank dargestellt werden muß. Dieses Geschwisterobjekt allerdings enthält keine weiteren Unterobjekte mehr, sondern zeigt zurück auf den Rahmen, also das Elternobjekt. Auf diese Weise schließt sich die Kette wieder. Sie fragen sich jetzt vielleicht, warum nicht das Elternobjekt auf alle Kindobjekte verweist. Ganz einfach: Deren Anzahl kann variieren. Mit der oben beschriebenen Technik wird aber in jeder Struktur nur eine feste Anzahl von Zeigern gebraucht. Hier sind es drei Zeiger. Der erste (`ob_head`) zeigt auf das erste Unterobjekt, der zweite (`ob_next`) gibt ein eventuelles Geschwisterobjekt an, und der letzte (`ob_tail`) verweist auf das letzte der möglichen Unterobjekte.

In diesen Zeigern wird nun nicht etwa eine Speicheradresse abgelegt, sondern einfach eine Nummer. Dabei werden alle Objekte eines Programmes mit 0 beginnend durchnummeriert. Zeiger, die ins Leere zeigen (z.B. wenn keine Unterobjekte vorhanden sind), erhalten den Wert -1.

In C sieht die Speicherstruktur für ein Objekt so aus:

```
struct objekt
{
    int    ob_next; /* Zeiger auf Geschwisterobjekt */
    int    ob_head; /* Zeiger auf erstes Unterobjekt */
    int    ob_tail; /* Zeiger auf letztes Unterobj. */
    unsigned ob_type; /* Objekttyp */
    unsigned ob_flags; /* Eigenschaften */
}
```

```

unsigned    ob_state; /* Aussehen */
long    ob_spec; /* Je nach Typ */
int    ob_x; /* X-Koordinate */
int    ob_y; /* Y-Koordinate */
int    ob_width; /* Breite */
int    ob_height; /* Höhe */
} OBJEKT;

```

Koordinaten werden immer relativ zum Elternobjekt angegeben. Der Objekttyp

335

kann folgende Werte annehmen:

20 G_BOX:

Rechteckiger Kasten, "ob_spec" ist eine 32 Bit lange Kodierung der Eigenschaften. Die Bits 0 bis 3 geben die Füllmusterfarbe an, die Bits 4 bis 6 das Füllmuster. Ist Bit 7 gleich 0, wird das Objekt transparent gezeichnet, sonst deckend. Die Bits 8 bis 11 nennen die Farbe des Zeichens im Box-Info-Word; dieses Element kommt nur im Typ G_BOXCHAR vor. Die Bits 12 bis 15 geben die Randfarbe an, in den Bits 16 bis 23 wird dessen Dicke gespeichert. Diese Dicke kann zwischen -127 und 128 liegen, bei negativen Werten wird der Rand nach außen gezeichnet. Die letzten 8 Bits (24 - 31) nennen das Zeichen für das Box-Info-Word.

21 G_TEXT:

Graphiktext, "ob_spec" ist ein Zeiger auf eine TEDINFO-Struktur (siehe unten).

22 G_BOXTEXT:

Rechteckiger Kasten mit Text. Ob_spec ist ein Zeiger auf eine TEDINFO-Struktur.

23 G_IMAGE:

Zeichnung im Bit-Raster-Modus. "Ob_spec" zeigt auf eine BITBLK-Struktur.

24 G_PROGDEF:

Durch Programmierer definiertes Objekt. "Ob_spec" zeigt auf eine APPBLK-Struktur.

25 G_IBOX:

Wie G_BOX, jedoch unsichtbar. Wird häufig als Bezugsobjekt benutzt.

26 G_BUTTON:

Rechteck mit zentriertem Text. "Ob_spec" zeigt auf den Text im C-Format (letztes Byte ist 0).

27 G_BOXCHAR:

Rechteck mit einem zentrierten Buchstaben. "Ob_spec" sieht unter G_BOX...

28 G_STRING:

Wie G_BUTTON, jedoch ohne Rechteck.

29 G_FTEXT:

Formatierter Text, "Ob_spec" zeigt auf TEDINFO-Struktur.

30 G_FBOXTEXT:

Wie G_FTEXT, jedoch mit umgebendem Rechteck.

31 G_ICON:

Icon. "Ob_spec" zeigt auf ICONBLK-Struktur.

32 G_TITLE:

Text für die Titel eines Drop-Down-Menüs. "Ob_spec" zeigt auf den Text im C-Format.

Mit den Objekt-Flags wird angegeben, welche Eigenschaften ein Objekt hat. Jedes Bit kann dabei wie ein Schalter betrachtet werden. Für jedes Bit gibt es in den meisten Compilern eine Konstante (z.B. NONE, wenn kein Bit gesetzt werden soll).

Bit 0: SELECTABLE

Eingabeobjekt: Beim Anklicken wird es invertiert.

Bit 1: DEFAULT

Aktivieren ist auch durch Return-Taste möglich. Darf nur einmal pro Objekt benutzt werden (meistens für den OK-Button in Dialogboxen).

Bit 2: EXIT

Durch Anklicken dieses Objekts wird das Objekt verlassen (für Dialogboxen).

Bit 3: EDITABLE

Objekt kann vom Benutzer verändert werden.

Bit 4: RBUTTON

Radiobutton, d.h. nur einer der Radiobuttons dieses Objekts kann invertiert erscheinen.

Bit 5: LASTOB

Dies ist das letzte Objekt in einem Baum.

Bit 6: TOUCHEXIT

Wie EXIT, aber Verlassen schon vor dem Loslassen des Knopfes.

Bit 7: HIDETREE

Unterobjekte werden versteckt und sind für die Funktionen "Objc_Draw" und "Objc_Find" nicht mehr auffindbar.

Bit 8: INDIRECT

Objektspezifikation ist ein Zeiger.

Der Objektstatus gibt an, wie das Objekt gegenwärtig aussieht. Auch hier sind die einzelnen Bits wie Schalter zu verstehen:

kein Bit gesetzt: NORMAL

Bit 0: SELECTED

Objekt erscheint invertiert und ist aktiviert.

Bit 1: CROSSED

Objekt wird durchgestrichen.

Bit 2: CHECKED

Objekt wird abgehakt.

Bit 3: DISABLED

Helle Darstellung (nicht anwählbar).

Bit 4: OUTLINED

Rahmen besteht aus einem dicken und einem dünnen Teil.

Bit 5: SHADOWED

Objekt besitzt einen Schatten.

Die verschiedenen Unterstrukturen (z.B. TEDINFO) sind in den nächsten Abschnitten dargestellt.

13.4.1 Die TEDINFO-Struktur

Für manche Objekte wird eine TEDINFO-Struktur gebraucht, die das Aussehen und den Inhalt eines Textes angibt. In C wird sie so definiert:

```
struct TEDINFO
{
    char *te_ptext; /* Zeiger auf den Text */
    char *te_ptmplt; /* Zeiger auf Eingabeschablone */
    char *te_pvalid; /* Zeiger auf zulässige Zeichen */
    int te_font; /* Zeichensatz */
    int te_res1; /* reserviert */
    int te_just; /* Text-Ausrichtung */
    int te_color; /* Farbe */
    int te_res2; /* reserviert */
    int te_thickness; /* Rahmenbreite */
    int te_txtlen; /* Textlänge inkl. 0 */
    int te_tmplen; /* Schablonenlänge inkl. 0 */
} tedinfo
```

Betrachten wir die einzelnen Elemente genauer. Der Zeiger "te_ptext" zeigt auf einen mit 0-Byte abgeschlossenen String, der im Objekt ausgegeben werden soll. Beginnt er mit einem Klammeraffen (@), werden alle Zeichen des Strings als Leerzeichen interpretiert. Der nächste Zeiger verweist auf die Eingabeschablone. Die Zeichen in diesem String regeln, ob der Benutzer den zuvor ausgegebenen Text verändern kann. Ist ein Unterstrich (_) gesetzt, kann die korrespondierende Position im Ausgabestring editiert werden (sofern eine der Formularfunktionen aufgerufen wurde). Der nächste String, durch "te_pvalid" markiert, gibt an, welche Editiermöglichkeiten der Benutzer hat. Wie bei der Schablone steht jedes Zeichen für das korrespondierende Byte des Ausgabetextes. Folgende Möglichkeiten bestehen:

- 9: Nur Ziffern sind zulässig
- A: Großbuchstaben oder Leerzeichen
- a: Buchstaben oder Leerzeichen
- N: Ziffern, Großbuchstaben oder Leerzeichen
- n: Ziffern, Buchstaben oder Leerzeichen
- F: File-Namen-Zeichen und "?", "*", ":"
- p: wie F, zusätzlich "I"
- P: File-Namen-Zeichen und "I", ":"
- X: alle Zeichen

Mit "te_font" wird die Zeichensatznummer angegeben. Die 3 steht für den großen, 5 für den kleinen Zeichensatz. Enthält "te_just" den Wert 0, wird linksbündig ausgegeben, bei 1 rechtsbündig und bei 2 zentriert.

Die Farbe ("te_color") kann aus den Werten 0 bis 15 ausgewählt werden. Welche Farbe erscheint, hängt von der Einstellung im Kontrollfeld ab. Die Rahmendicke wird durch den Wert in "te_thickness" bestimmt. Negative Werte bedeuten, daß der Rahmen nach außen gezeichnet wird, bei positivem Wert nach innen. Je größer der Absolutwert, desto dicker der Rahmen.

Bei den beiden Längenangaben am Schluß der Struktur muß beachtet werden, daß diese immer das Nullbyte mitzählen müssen, mit denen Strings in C abgeschlossen werden.

13.4.2 Die ICONBLK-Struktur

Mit dieser Struktur werden Icons definiert. Wie gewohnt hier zunächst die C-Definition:

```
struct ICONBLK
{
    int *ib_pmask; /* Zeiger auf Maske */
    int *ib_pdata; /* Zeiger auf Farbinfo */
    char *ib_ptext; /* Zeiger auf Text */
    int ib_char; /* Zeichen im Icon */
    int ib_xchar; /* X-Koordinate des Zeichens */
    int ib_ychar; /* Y-Koordinate des Zeichens */
    int ib_xicon; /* X-Koordinate des Icons */
    int ib_yicon; /* Y-Koordinate des Icons */
    int ib_wicon; /* Breite des Icons */
    int ib_hicon; /* Höhe des Icons */
    int ib_xtext; /* X-Koordinate des Textes */
    int ib_ytext; /* Y-Koordinate des Textes */
    int ib_wtext; /* Textbreite */
    int ib_htext; /* Texthöhe */
} iconblk;
```

Mit der Maske wird der Hintergrundumriß des Icons festgelegt. Eine Null steht für einen durchsichtigen Punkt, mit einer Eins wird ein Punkt in der Hintergrundfarbe gesetzt. Die Bits werden einfach als Folge hintereinander geschrieben. Die Umrechnung in Zeilen ergibt sich durch den Parameter "ib_wicon". Er gibt an, wieviele Bits eine Zeile bilden. Achtung: Dieser Wert muß unbedingt durch 16 teilbar sein! Daher muß die Maske auch aus ganzen Wörtern (je 16 Bit) bestehen. Das Aussehen des eigentlichen Icons ergibt sich durch das Bit-Muster, auf das "ib_pdata" zeigt. Auch hier steht wieder eine 1

für einen gesetzten Bildpunkt, dieser wird jedoch in der Vordergrundfarbe gezeichnet. Wird ein solches Icon angeklickt, kehren sich die Farbgebungen um, d.h. die Punkte der Maske tauschen ihre Farbe mit denen des Bit-Musters.

Alle anderen Parameter sind weitgehend selbsterklärend. Beachten Sie bitte, daß alle Koordinaten- und Weitenangaben in Bildschirmpunkten gemacht werden und daß die Position von Text und Einzelzeichen relativ zum Icon gemacht werden muß.

13.4.3 Die BITBLK-Struktur

Starke Ähnlichkeit mit der ICONBLK-Struktur zeigt diese Definition. Ihre Aufgabe ist die Beschreibung einfacher Bit-Mustergrafiken.

```
struct BITBLK
{
    int *bi_pdata; /* Zeiger auf Bit-Muster */
    int bi_wb;      /* Grafikbreite in Bytes */
    int bi_hl;      /* Grafikhöhe in Punkten */
    int bi_x;       /* Position x-Koordinate */
    int bi_y;       /* Position y-Koordinate */
    int bi_color;   /* Farbe */
} bitblock
```

Die einzelnen Werte haben annähernd die gleiche Bedeutung wie in ICONBLK. Aber Achtung: Die Breite der Grafik wird hier inkonsequenterweise in Bytes (nicht in Punkten) angegeben!

13.4.4 Die APPLBLK-Struktur

Für manche Anwendungen kann es notwendig sein, das Zeichnen eines Objektes durch eigene Routinen durchführen zu lassen. Mit Hilfe der APPLBLK-Struktur wird dem GEM bekanntgegeben, wo die Routine steht und welchen Parameter sie erwartet.

```
struct APPL_BLK
{
    int (*ub_code)(); /* Zeiger auf Routine */
    long ub_param; /* 4 Parameterbytes */
} appl_blk
```

Schön kurz, nicht wahr? Im Zeiger wird einfach die Adresse gespeichert, unter der die Routine im Speicher abgelegt wurde. Diese wird vom AES mit einem

einzigsten Parameter (einem Zeiger) aufgerufen. In diesen Zeiger wird auf eine weitere Struktur verwiesen, in die unter anderem der Wert `ub_parm` kopiert wird. In diesem Wert können Sie also bis zu vier Bytes mit verschiedenen Informationen für das Unterprogramm hinterlegen.

Die Zusatzstruktur ist folgendermaßen definiert:

```
struct PARMBLK
{
    long pb_tree;
    int  pb_obj;
    int  pb_prevstate;
    int  pb_currstate;
    int  pb_x, pb_y, pb_w, pb_h;
    int  pb_xc, pb_yc, pb_wc, pb_hc;
    long pb_parm;
} parm_blk;
```

"pb_tree" zeigt auf den Objektbaum, der das geforderte Objekt enthält. "pb_obj" ist die Ordnungsnummer des Objekts innerhalb des Baumes. "pb_prevstate" enthält den alten `ob_state`, "pb_currstate" den geforderten. Sind "pb_prevstate" und "pb_currstate" gleich, wurde der Aufruf von `OBJC_DRAW` verursacht; sind sie verschieden, war es `OBJC_CHANGE`. "pb_x" und "pb_y" stellen wie gehabt die Position des Objektes auf dem Bildschirm dar, "pb_w" die Breite des Objektes, "pb_h" die Höhe in Pixel.

"pb_xc", "pb_yc":	Position des "Clip-Rectangle" (siehe <code>OBJC_DRAW</code>)
"pb_wc":	Breite des "Clip-Rectangle"
"pb_hc":	dessen Höhe
"pb_parm":	enthält <code>ub_parm</code> (siehe oben)

13.5 Windows und Ereignisse

Die Windows (Fenster) des AES werden immer dann gebraucht, wenn sich ein Text oder eine Grafik nicht mehr in seiner Gesamtheit am Bildschirm anzeigen läßt oder wenn mehrere verschiedene Grafiken oder Texte gleichzeitig am Bildschirm bearbeitet werden sollen. Das Aussehen eines solchen Windows wird Ihnen sicherlich vom normalen Desktop her bekannt sein, auch die einzelnen Window-Elemente, die in der folgenden Zeichnung wiederzufinden sind:

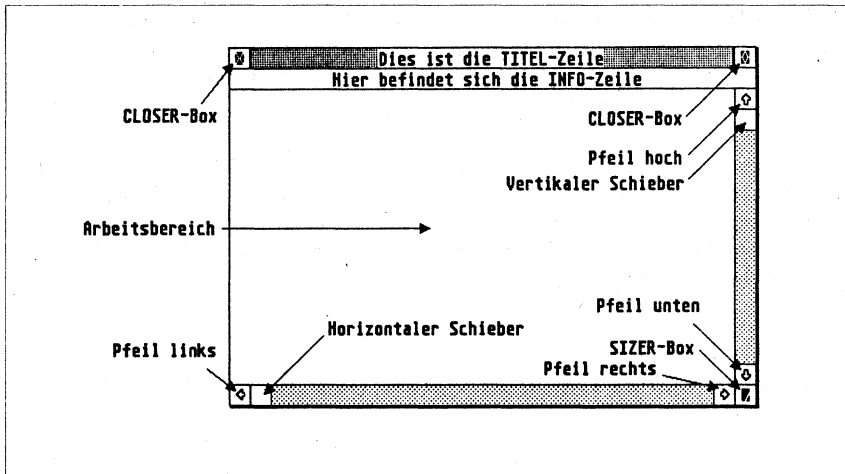


Abb. 13.4: Bildschirmfenster

Unser Problem soll es nun sein, diese Fenster zu programmieren und auf die Aktionen des Anwenders entsprechend zu reagieren.

Soll ein Fenster dargestellt werden, so muß sich der Programmierer zunächst über die Dimension und Beschaffenheit desselben im Klaren sein, also wissen, welche Fensterelemente sein Fenster besitzen soll und wie groß sein Fenster höchstens werden darf. Um das Ergebnis dieser Überlegungen dem AES mitzuteilen, verwendet der geübte Programmierer die Funktion **WIND CREATE**.

Damit ist unser Fenster definiert und seine Fensternummer (Window-Handle), die wir bei allen folgenden Fensterbefehlen angeben müssen, reserviert. Doch es ist noch keinesfalls auf dem Bildschirm dargestellt. Hierzu dient ein spezieller Befehl: **WIND OPEN**. Um die ganze Prozedur des Fenster-Öffnens auch für das Auge des Anwenders zu verschönern, kann vor der eigentlichen Darstellung des Fensters durch **WIND OPEN** ein größer werdendes Rechteck durch den **GRAF GROWBOX**-Befehl (siehe AES-Library im Anhang) gezeichnet werden, so daß auch wirklich der Eindruck eines sich öffnenden Fensters entsteht.

Jetzt, da wir ein Fenster dargestellt haben, müssen wir natürlich auch dafür Sorge tragen, daß uns AES mitteilt, ob der Benutzer unseres Programms eine bestimmte Fenster-Funktion (Größenfeld etc.) angewählt hat, damit wir darauf reagieren können. Hierzu legen wir einen sechzehn Bytes (acht Wörter) langen sogenannten Message-Buffer (Nachrichten-Puffer) an, in dem dann die Art der

Events (Ereignisse) abgelegt wird. Die Adresse dieses Message-Buffers übergeben wir der EVNT_MULTI-Routine, die dann auf ein beliebiges Ereignis (nicht nur Window-Ereignis) wartet und uns durch verschiedene Parameter im Message-Buffer mitteilt, welches Ereignis aufgetreten ist.

Wenn Sie sich den Anhang mit der Beschreibung dieser Funktion ansehen, stellen Sie schnell fest, wie komplex die Ansteuerung der Event-Multi-Routine ist. Wir wollen uns an dieser Stelle nur auf die Verwaltung des Message-Buffers konzentrieren. Die folgende Tabelle zeigt Ihnen, bei welchen Parametern welches Ereignis eingetreten ist, und wie Sie darauf entsprechend reagieren können:

Neuzeichnen eines Window-Bereichs (WM_REDRAW):

Inhalt des Message-Buffers:

Wort 0	20 (Event-Nummer)
Wort 3	Window-Handle des Fensters, das neu gezeichnet werden soll
Wort 4	x-Koordinate des neu zu zeichnenden Bereichs
Wort 5	y-Koordinate des neu zu zeichnenden Bereichs
Wort 6	Breite des neu zu zeichnenden Bereichs
Wort 7	Höhe des neu zu zeichnenden Bereichs

Ist das erste Wort des Message-Buffers = 20, so erkennen Sie daran, daß ein bestimmter Teil eines Fensters neu gezeichnet werden muß, zum Beispiel, weil das Kontroll-Feld über Ihrem Fenster gelegen hat. Durch die anderen Einträge im Message-Buffer teilt Ihnen das AES mit, welcher Bereich neu gezeichnet werden muß.

Aktivierung eines anderen Fensters (WM_TOPPED):

Inhalt des Message-Buffers:

Wort 0	21 (Event-Nummer)
Wort 3	Window-Handle des Fensters, das der Benutzer aktivieren möchte.

Durch Anklicken eines nicht aktiven Fensters (es können zwar mehrere Fenster gleichzeitig auf dem Bildschirm dargestellt werden, aber nur eins kann dabei aktiv sein), kann der Benutzer dieses aktivieren. Das AES teilt Ihnen diese Aktion des Anwenders durch die hier angegebenen Message-Buffer-Einträge mit. Durch die WIND_SET-Funktion (wi_sfield=10, s. Anhang) kann dem AES mitgeteilt werden, welches Fenster das aktive sein soll.

Schließen eines Fensters (WM_CLOSED):

Inhalt des Message-Buffers:

Wort 0 22

Wort 3 Window-Handle des Fensters, das der Benutzer schließen möchte.

Wird das erste Wort des Messagebuffers = 22, so hat der Benutzer die CLOSE-Box des Fensters betätigt, um das Fenster zu schließen. Durch die WIND_CLOSE-Funktion kann das Fenster geschlossen werden. Sollen auch Fensterdefinitionen und Window-Handle gelöscht werden, so muß anschließend noch ein WIND_DELETE erfolgen. Nach einem WIND_CLOSE sind die Handle-Nummer und die Fensterdefinitionen noch erhalten, das gleiche Fenster kann also wieder durch WIND_OPEN geöffnet werden. WIND_DELETE aber sorgt dafür, daß dieses Fenster nicht mehr so ohne weiteres geöffnet werden kann. Hierzu müssen Sie mit WIND_CREATE wieder ein ganz neues Fenster deklarieren. Spätestens am Programmende sollte jedes Fenster durch WIND_DELETE gelöscht werden.

Setzen der Maximalgröße eines Fensters (WM_FULLED):

Inhalt des Message-Buffers:

Wort 0 23

Wort 3 Window-Handle des Fensters, das auf Maximalgröße gebracht werden soll.

Gedenkt der Benutzer, das Fenster durch Anklicken der FULL-Box auf Maximalgröße zu bringen, so teilt Ihnen GEM das durch diese Parameter mit. Sie können dann zunächst durch WIND_GET mit wi_wgfield=4, und wi_ghandle=0 die Ausmaße des Desktop einholen, und anschließend mit dem WIND_SET-Befehl (wi_sfild=5) die neuen Fensterausmaße setzen.

Verschieben des Fensterinhalts:

Inhalt des Message-Buffers:

Wort 0 24

Wort 3 Window-Handle

Wort 4 Nummer des Feldes, das angeklickt wurde:

=0: Bereich über d. vertikalen Schieber (Seite nach oben)

=1: Bereich unter d. vertikalen Schieber (Seite nach unten)

=2: Pfeil nach oben wurde angeklickt (Zeile nach oben)

- =3: Pfeil nach unten wurde angeklickt (Zeile nach unten)
- =4: Bereich links neben horizontalem Schieber (Seite nach links)
- =5: Bereich rechts neben dem horizontalen Schieber (Seite nach rechts)
- =6: Pfeil nach links wurde angeklickt (Spalte nach links)
- =7: Pfeil nach rechts wurde angeklickt (Spalte nach rechts)

Hat der Benutzer einen der vier Pfeile oder einen der freien Bereiche neben den Schiebern angeklickt, um den Fensterinhalt zeilen- bzw. seitenweise zu verschieben, so erkennen Sie dies an den obenstehenden Message-Buffer-Einträgen.

Verschieben des horizontalen Schiebers (WM_HSLID):

Inhalt des Message-Buffers:

Wort 0 25
Wort 3 Window-Handle
Wort 4 Position (0-1000), auf die der Schieber gesetzt wurde.

Wurde der horizontale Schieber betätigt, so erkennen Sie das an den entsprechenden Eintragungen des Messagebuffers. Durch den WIND_SET-Befehl mit "wi_sfield=8" kann die neue Position des Schiebers gesetzt werden. Natürlich müssen Sie selbst dafür sorgen, daß der Fensterinhalt entsprechend verschoben wird.

Verschieben des vertikalen Schiebers (WM_VSLID):

Inhalt des Message-Buffers:

Wort 0 26
Wort 3 Window-Handle
Wort 4 Position (0-1000), auf die der Schieber gesetzt wurde.

Ist der vertikale Schieber betätigt worden, so erkennen Sie dies auch hier wieder an den Eintragungen des Message-Buffers. Durch den WIND_SET-Befehl mit "wi_sfield=9" kann die neue Position des Schiebers gesetzt werden. Auch hier müssen Sie wieder dafür sorgen, daß der Fensterinhalt entsprechend verschoben wird.

Vergrößern oder Verkleinern des Fensters (WM_SIZED):*Inhalt des Message-Buffers:*

Wort 0	27
Wort 3	Window-Handle
Wort 4	x-Koordinate des Fensters
Wort 5	y-Koordinate des Fensters
Wort 6	neue Breite des Fensters
Wort 7	neue Höhe des Fensters

Hat der Benutzer durch die SIZE-Box die Größe des Fensters verändert, so können Sie dies durch oben stehende Messagebuffer-Parameter erkennen. Durch den WIND_SET-Befehl mit "wi_sfield=5" können Sie AES die neue Größe des Fensters mitteilen.

Bewegen des Fensters (WM_MOVED):*Inhalt des Message-Buffers:*

Wort 0	28
Wort 3	Window-Handle
Wort 4	neue x-Koordinate des Fensters
Wort 5	neue y-Koordinate des Fensters
Wort 6	neue Breite des Fensters
Wort 7	neue Höhe des Fensters

Wurde das Fenster bewegt, so können Sie die neue Position des Fensters im Messagebuffer nachlesen. Auch hier kann wieder der WIND_SET-Befehl mit "wi_sfield=5" herangezogen werden, um die neue Fensterposition dem AES mitzuteilen.

Fenster wurde aktiviert (WM_NEWTOP):*Inhalt des Message-Buffers:*

Wort 0	30
Wort 3	neues Window-Handle

Wenn ein Fenster (z.B. durch Schließen eines anderen) zwangsweise aktiviert wurde, so teilt Ihnen dies das AES durch oben stehende Message-Buffer-Parameter mit. Hier brauchen Sie nichts weiter zu machen, als sich das neue Window-Handle zu merken.

Damit hätten wir die wichtigsten Dinge der Fensterprogrammierung hinter uns gebracht. Ein letztes Problem der Fensterprogrammierung bleibt die sogenannte Rechtecklistenverwaltung beim Erneuern mehrerer Fenster. Wenn Sie z.B. das Kontroll-Feld-Accessory über ein Fenster verschieben, so dürfen Sie natürlich nicht den gesamten Fensterinhalt erneuern, da ja sonst das Kontrollfeld selbst ebenfalls überschrieben würde. Deswegen verwaltet GEM Rechtecklisten, die uns nacheinander den zu erneuernden Bereich um eine bestimmte Fläche liefern. Wir brauchen diese dann nur noch mit dem Hauptrechteck (durch den Message-Buffer bei WM_REDRAW erhalten) zu clippen und dann die einzelnen Flächen zu erneuern. Clippen bedeutet hierbei, daß nur diejenigen Flächenbereiche neu gezeichnet werden, die sowohl durch das eine wie auch durch das andere Rechteck überdeckt werden (Schnittmenge). Wie eine solche Schnittmenge (oder die gemeinsame Fläche) ermittelt wird, können Sie der Redraw-Routine des Beispielprogramms (am Ende des Kapitels) entnehmen.

Bei der Programmierung einer solchen REDRAW-Routine muß dann wie folgt vorgegangen werden:

Zunächst wird durch den Messagebuffer das Hauptrechteck ermittelt. Mit diesem Rechteck müssen alle anderen Rechtecke der Rechteckliste geklippt werden. Anschließend wird die Rechteckliste durch den WIND_UPDATE-Befehl (mit "wi_ubegend=1") freigegeben und dadurch weitere Benutzer-Ereignisse ausgeschlossen, die den Redraw-Vorgang stören würden. Nun wird WIND_GET (wi_gfield=11) aufgerufen, um das erste Rechteck der Rechteckliste zu erhalten. Dieses erste Rechteck wird, wie alle anderen Rechtecke der Rechteckliste, mit dem Hauptrechteck geklippt und der resultierende Flächenbereich erneuert. Es wird jetzt so lange die Funktion WIND_GET aufgerufen und die entsprechenden mit dem Hauptrechteck geklippten Teile erneuert, bis die Breite und Höhe eines Rechtecks = 0 werden. Dies ist für uns das Zeichen, daß alle Rechtecke der Rechteckliste bearbeitet wurden. Schließlich wird nur noch der WIND_UPDATE-Befehl mit "wi_ubegend=0" aufgerufen, um Ereignisse wieder zu erlauben.

Weil das Ganze nicht so einfach ist, folgt hier ein einfaches Beispielprogramm. Es leistet nichts Großartiges, zeigt aber alle Schritte, die zur Behandlung eines Fensters nötig sind.

Die Elemente, die Sie in Ihrem Fenster benutzen wollen, müssen in der Variablen "wi_crkind" bei der "wind_create"()-Funktion angegeben werden. Dabei wird jedes Element durch ein Bit repräsentiert. Ist dieses Bit gesetzt, wird das entsprechende Element dargestellt. Es gelten folgende Bitbelegungen:

0x0001	NAME	(Titelbalken)
0x0002	CLOSER	(Schließfeld)
0x0004	FULLER	(Größeneinstellfeld)
0x0008	MOVER	(Bewegungsbalken)
0x0010	INFO	(Infozeile)
0x0020	SIZER	(Vergrößerungsfeld)
0x0040	UPARROW	(Pfeil nach oben)
0x0080	DNARROW	(Pfeil nach unten)
0x0100	VSLIDE	(vertikaler Schieber)
0x0200	LFARROW	(Pfeil nach links)
0x0400	RTARROW	(Pfeil nach rechts)
0x0800	HSLIDE	(horizontaler Schieber)

Diese Definitionen wurden Ihnen bereits von Ihrer Compiler-Bibliothek abgenommen. Dort finden Sie sie je nach Compiler z.B. in der GEMDEFS.H-Library. Um die entsprechenden Bits nun zu setzen, brauchen Sie nur noch die definierten Konstanten durch eine ODER-Operation miteinander zu verknüpfen. Dieser Vorgang würde dann beispielsweise so aussehen:

```
wi_crreturn = wind_create(NAME|CLOSER|MOVER|INFO,wi_crw,...
```

Damit würde später ein Fenster erzeugt, in dem die Elemente Titelzeile, Schließfeld, Bewegungsbalken und Infozeile dargestellt werden und anwählbar sind. Die in obiger Grafik eingezeichneten Scroll-Balken werden bei der Ereignisabarbeitung zunächst wie Pfeile behandelt, erst später wird differenziert.

Wir geben nun als erstes die Schritte an, die notwendig sind, bis ein Fenster auf dem Bildschirm dargestellt wird.

Zuerst müssen wir wissen, wie groß unser Fenster maximal werden darf. Da das Desktop seinerseits ebenfalls ein Fenster ist, welches selbständig vom AES verwaltet wird, können wir mit der "wind_get"-Funktion den Arbeitsbereich dieses Fensters ermitteln. Denn genau dieser Bereich umfaßt die maximale Größe eines vom Benutzer erzeugten Fensters.

Also rufen wir diese Funktion wie folgt auf:

```
"wind_get"(0,4,&x0max,&y0max,&w0max,&h0max);
```

Die 0 ist das stets gleichbleibende Fenster-Handle für das Desktop. Durch den Parameter 4 bekommen wir die Größe der Arbeitsfläche des mit dem Handle ausgewählten Fensters, hier also des Desktops. Die gelieferten Koordinaten werden in den Variablen x0max bis h0max abgelegt, wobei x0max und y0max

die Koordinaten für die linke obere Ecke beinhalten, w0max gibt die Breite und h0max die Höhe dieses Fensterbereiches wieder. Als nächstes müssen wir den Arbeitsbereich unseres Fensters berechnen, da die obigen Koordinaten den Arbeitsbereich plus den Randbereich umfassen. Diese Arbeit wird von der "wind_calc"-Funktion erledigt. Zur Berechnung muß diese Routine allerdings wissen, welche Elemente das Fenster besitzen soll. Nehmen wir einmal an, das Fenster sollte einen Titlbalken, eine Infozeile, ein Schließfeld, einen Pfeil nach oben, einen Pfeil nach unten und einen vertikalen Schieber aufweisen. Der Aufruf von "wind_calc" sähe dann so aus:

```
"wind_calc"(1,NAME|INFO|CLOSER|DNARROW|UPARROW|VSLIDE,x0max,y0max,w0max,h0max,&x1,
&y1,&w1,&h1)
```

Durch den Parameter 1 wird "wind_calc" veranlaßt, die Größe des Arbeitsbereiches des Fensters zu berechnen. Das Ergebnis wird in den Variablen x1 bis h1 abgelegt.

Neben dem maximalen Arbeitsbereich müssen wir noch den tatsächlich später benutzten Bildschirmbereich ermitteln. Dieser Bereich darf kleiner, aber nicht größer als der eben berechnete Arbeitsbereich sein. Das erreichen wir, indem wir nochmals die "wind_calc"-Funktion aufrufen. Nur wird jetzt als erster Parameter eine 0 und nicht wie vorhin eine 1 übergeben. Als Ergebnis bekommen wir die Koordinaten für den Bereich, den das Fenster einschließlich des Randbereiches einnimmt.

```
wind_calc(0,NAME|INFO|CLOSER|DNARROW|UPARROW|VSLIDE,x1,y1,w1,h1,&x1max,&y1max,&w1max,&h1max)
```

Mit diesen Koordinaten können wir nun die "wind_create"-Funktion aufrufen, die den notwendigen Platz, den das Fenster benötigt, reserviert. Bei Erfolg bekommt man von dieser Funktion eine Kennziffer, das sogenannte Window Handle, mit dem man später das Fenster ansprechen kann. Da GEM-AES max. acht Fenster gleichzeitig unterstützt, ist dieses Window Handle, kurz Handle, eine eindeutige Kennung für jedes einzelne Fenster. Sollte kein Fenster mehr verfügbar sein, wird ein negativer Funktionswert zurückgegeben.

Der Aufruf sieht folgendermaßen aus:

```
handle=wind_create(NAME|INFO|CLOSER|DNARROW|UPARROW|VSLIDE,x1max,y1max,w1max,h1max);
```

Bevor wir nun das Fenster mit "wind_open" öffnen, müssen wir dem AES die String-Adressen des Fensternamens und der Infozeile mitteilen. Dies geschieht

mit der "wind_set"-Funktion. Da wir hier nur die String-Adressen übergeben, müssen diese Variablen als static deklariert werden. Ein Aufruf sähe dann so aus:

```
static char titelzeile[] = "Fenstername";  
static char infozeile[] = "Fensterinfo";  
wind_set(handle,2,titelzeile,0,0);  
wind_set(handle,3,infozeile,0,0);
```

Nun kann endlich das Fenster mit "wind_open" geöffnet werden:

```
"wind_open"(handle,xlmax,ylmax,wlmax,hlmax);
```

Durch diesen Aufruf wird das Fenster in seiner größten Ausdehnung auf dem Bildschirm dargestellt. Wollen Sie ein kleineres Fenster, so teilen Sie die Variablen xlmax bis hlmax durch einen Wert, der der von Ihnen gewünschten Größe entspricht, und setzen diese Werte anstelle der Maximalwerte ein.

Wenn Sie sich das bis jetzt erzeugte Fenster anschauen würden, so wäre die Arbeitsfläche noch nicht zu gebrauchen. Das folgende Bild zeigt Ihnen den jetzigen Zustand.

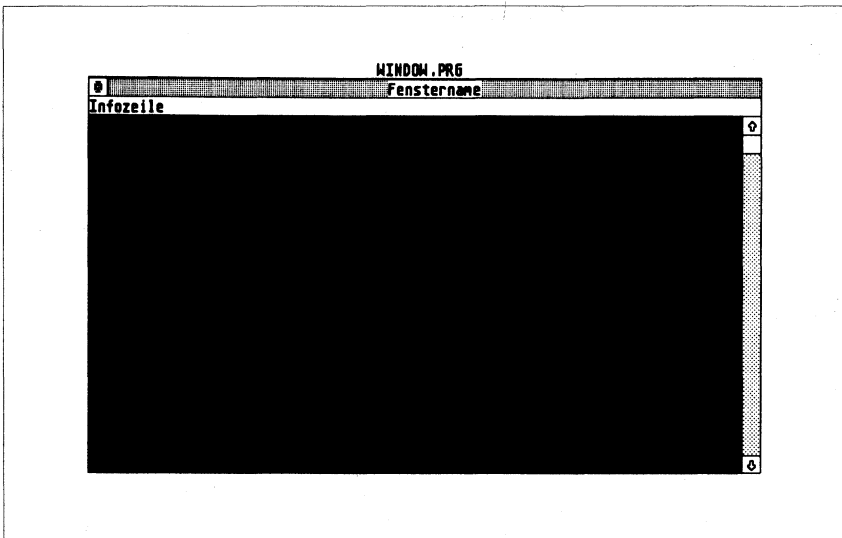


Abb. 13.5: Zwischenstufe bei der Fensterprogrammierung

Um die Arbeitsfläche z.B. mit Text beschreiben zu können, sollten wir diese Fläche vorher löschen. Dies geschieht mit Hilfe dreier VDI-Routinen: "vsf_color()", "vsf_interior" und "vsf_bar".

Mit "vsf_color()" wird die Farbe für das Füllmuster festgesetzt. Bei Monochrom-Monitoren ist nur schwarz 0 und weiß 1 möglich. Der "color_index" gibt also den gewünschten Farbindex an. In "set_color" wird nach dem Aufruf der gesetzte Farbindex abgelegt. Mit handle ist das VDI-Handle gemeint.

Mit "vsf_interior()" wird der Fülltyp festgelegt. Es gibt 5 Füllfunktionen

- 0 Fläche wird nicht ausgefüllt
- 1 Fläche wird mit der Füllfarbe ausgefüllt
- 2 Fläche wird mit dem unter "vsf_style" festgelegten Muster ausgefüllt
- 3 Fläche wird mit dem unter "vsf_style()" festgelegten schraffierten Muster ausgefüllt
- 4 Fläche wird mit selbstdefiniertem Muster ausgefüllt

Mit "vsf_bar()" können gemäß der gewählten Attributfunktionen ausgefüllte Rechtecke gezeichnet werden. Das pxyarray muß folgende Koordinaten beinhalten:

pxyarray[0]	- x-Koordinate des oberen linken Eckpunktes
pxyarray[1]	- y-Koordinate des oberen linken Eckpunktes
pxyarray[2]	- x-Koordinate des unteren rechten Eckpunktes
pxyarray[3]	- y-Koordinate des unteren rechten Eckpunktes

Übergeben wir nun die Koordinaten des Arbeitsbereiches (x1 bis h1), so müssen wir vorher noch die Koordinaten für den Eckpunkt unten rechts bestimmen. Diese Berechnung führen wir bei der Übergabe durch, die dann wie folgt aussieht:

```
pxyarray[0] = x1;  
pxyarray[1] = y1;  
pxyarray[2] = (x1+w1);  
pxyarray[3] = (y1+h1);
```

Ist diese Initialisierung des pxyarray beendet, rufen wir der Reihe nach die oben erwähnten VDI-Routinen auf, und der Fensterinhalt wird gelöscht.

```
vsf_color(vdi_handle,0); /*Farbindex "weiß" wird gewählt*/  
vsf_interior(vdi_handle,1); /*Fülltyp "solid" wird gewählt*/  
v_bar(vdi_handle,pxyarray); /*Rechteck bzw. Fenster wird gefüllt*/
```

Nun steht uns endlich ein blütenweißes Fenster zur Verfügung, das wir später mit Text füllen wollen.

Den Text, den wir im Fenster darstellen wollen, legen wir in einem Textpuffer ab, der aus einer Array-Struktur besteht. Der Text selber besteht der Einfachheit halber aus ein und dem selben Satz, dem eine vorlaufende Nummer vorangestellt wird. Dieses schien uns der einfachste Weg und reicht für Demonstrationszwecke vollkommen aus.

Um diesen Text nun ins Fenster einblenden zu können, wird eine weitere VDI-Funktion herangezogen, die "v_gtext"-Funktion. Damit wird der Text, der unter string[] abgelegt ist, an den Koordinaten x/y auf dem Grafikbildschirm ausgegeben.

Nachdem nun das Fenster geöffnet und der Text eingeblendet wurde, muß als nächster Schritt die Größe und Position des vertikalen Schiebers berechnet werden. Die Größe wird nach folgender Formel berechnet:

$$\text{Größe} = 1000 * \text{Anzahl der sichtbaren Zeilen} / \text{Gesamtzahl der Zeilen}$$

Bei unserem später folgenden Beispiel beträgt die Anzahl sichtbarer Zeilen 20 und die Gesamtzeilenzahl 50. Danach berechnet sich die Schiebergröße zu

$$\text{groesse} = 1000 * 20 / 50$$

Der hieraus ermittelte Wert von 400 muß mittels "wind_set" gesetzt werden:

```
wind_set(wi_handle, WF_VSLSIZE, groesse, 0, 0, 0);
```

Sollten weitere Textzeilen hinzukommen oder sich sonstwie die Größe des Textes ändern, muß die Schiebergröße jeweils neu berechnet und gesetzt werden. Die Schieberposition wird wie folgt ermittelt:

$$\text{place} = 1000 * (1. \text{ Position im Fenster} - 1. \text{ Textposition}) / (\text{Gesamtzahl der Textzeilen} - \text{sichtbare Textzeilen})$$

Dieser Wert muß nach jeder Bewegung des Schiebers neu berechnet und mittels "wind_set" gesetzt werden:

```
wind_set(wi_handle, WF_VSLIDE, place, 0, 0, 0);
```

Damit sind die Vorarbeiten abgeschlossen, und die Kontrolle über die Fenster- bzw. Benutzeraktionen werden von einer "evnt_mesag"- oder "evnt_multi"-Funktion übernommen. Wir haben in unserem Beispiel die "evnt_mesag"-Funktion gewählt, da sie für unsere Zwecke ausreichend ist. Diese Funktion

wartet, wie bereits oben erwähnt, nun darauf, daß der Benutzer irgendein Fensterelement betätigt. Diese Schleife wird so lange durchlaufen, bis das Schließfeld betätigt wird. Wurde ein Element angeklickt, so finden wir im Message-Puffer Nummer 0 msgbuff[0] die von uns benötigte und vom Programm dort abgelegte Nachricht, um welches Ereignis es sich handelt. Diese Nachricht wird durch eine Konstante repräsentiert:

```
#define MN_SELECTED 10 /*Menüeintrag wurde angeklickt*/  
#define WM_REDRAW 20 /*Fenster neu zeichnen*/  
#define WM_TOPPED 21 /*Fenster soll aktiviert werden*/  
#define WM_CLOSED 22 /*Schließfeld wurde betätigt*/  
#define WM_FULLED 23 /*Das "FULLER"-Feld wurde gewählt*/  
#define WM_ARROWED 24 /*Pfeil nach oben wurde angeklickt*/  
#define WM_HSLID 25 /*horizontaler Schieber wurde bewegt*/  
#define WM_VSLID 26 /*vertikaler Schieber wurde verschoben*/  
#define WM_SIZED 27 /*Größe des Fensters wurde verändert*/  
#define WM_MOVED 28 /*Fenster wurde bewegt*/  
#define WM_NEWTOP 29 /*Fenster wurde aktiviert*/  
#define AC_OPEN 40 /*Accessory-Eintrag wurde angeklickt*/  
#define AC_CLOSE 41 /*Accessory soll geschlossen werden*/
```

Diese Definitionen befinden sich wieder in einer Header-Datei des Compilers (z.B. GEMDEFS.H). Die von uns benötigten Konstanten werden nun mittels einer "switch()-case"-Konstruktion überprüft, um dann in die entsprechenden Routinen zu verzweigen.

Wurde in unserem Fensterprogramm z.B. der vertikale Schieber bewegt WM_VSLIDE, wird nach obiger Formel die neue Schieberposition berechnet und gesetzt. Anschließend wird noch die neue erste Zeile berechnet (Formel siehe Listing "neu_top"()), der Fensterinhalt gelöscht und der Text neu positioniert. Wurde hingegen einer der Pfeile gewählt (oder auch ein Scroll-Balken, s.a. Grafik zu den Fensterelementen) WM_ARROW, so wird in einer weiteren Routine pfeile() geprüft, welcher der Pfeile bzw. Scrollbalken angeklickt wurde. Diese Information finden wir im Message-Puffer Nummer 4 msgbuff[4].

Wurde einer der Scroll-Balken gewählt, wird der Text um eine Textseite voroder zurückgeblättert. Dazu wird einfach der Index der ersten Zeile um die entsprechende Zahl erhöht bzw. verringert. Bei der Wahl einer der Pfeile wird dieser Index nur jeweils um 1 verändert. Bei jeder Auswahl wird nach dieser Berechnung die neue Schieberposition berechnet (dieser wurde zwar nicht betätigt, da er aber einen Überblick geben soll, wo man sich im Text befindet, muß eben diese neue Position ermittelt werden) und durch "wind_set" wie oben gezeigt gesetzt.

Wurde hingegen das Schließfeld angeklickt, wird die "evnt_mesag"-Schleife verlassen, das Fenster geschlossen "wind_close" und gelöscht "wind_delete", die virtual screen workstation geschlossen "v_clsvwk" und schließlich zu allerletzt das Programm beim AES abgemeldet "appl_exit".

Das waren nun die wichtigsten Schritte, um ein Fenster zu erzeugen, Text einzublenden und die Fensterelemente bei ihrer Betätigung zu verwalten. Wie anfangs erwähnt, konnten wir hier nicht auf alle Möglichkeiten, die das AES zur Fenstermanipulation zur Verfügung stellt, eingehen, da es bei weitem den Rahmen dieses Buches sprengen würde. Sollten Sie aber durch dieses Beispielprogramm die Scheu vor der Fensterprogrammierung verloren haben und vielleicht tiefer in diese Materie eindringen wollen, so haben wir unser Ziel erreicht. Nun aber endlich das komplette Programmlisting, das wir zur besseren Übersicht so weit wie nötig kommentiert haben.

```
#include<stdio.h>
#include<osbind.h>
#include<gemdefs.h>
#include<gembind.h>
#include<obdefs.h>
#define WI_KIND (NAME|CLOSER|INFO|DNARROW|UPARROW|VSLIDE)/*Window-
Elemente*/

#define SP(x) 8*x-8 /*'X' gibt gewünschte Spalte an*/
#define ZE(y) 16*y+16 /*'Y' gibt gewünschte Zeile an*/
#define GZEILEN 20 /*Gesamtzeilenzahl im Window*/
#define DESK 0 /*Windowhandle für Desktop*/
int contrl [12], /*GEM-Arrays für VDI und AES*/
    intin [128],
    ptsin [128],
    intout [128],
    ptsout [128],
    work_in [11],
    work_out[57],
    msgbuff [8],
    vdi_handle, /*VDI-Handle*/
    phys_handle, /*phys.Workstation-Handle*/
    appl_id, /*Applikations-Identifikation*/
    wi_handle, /*Windowhandle*/
    xl,y1,w1,h1, /*Arbeitsbereich des Desktop*/
    x2,y2,w2,h2, /*max. Fensterarbeitsbereich*/
    x3,y3,w3,h3, /*Arbeitsfläche des Fensters*/
    pxyarray[4], /*Rechteck-Koordinatenfeld*/
    top, /*erste sichtbare Textzeile*/
    anzahl; /*Gesamtzahl der Textzeilen*/

typedef struct textrec /*Textpuffer wird angelegt*/
{
    char textzeile[40]; /*Zeilenlänge*/
}TEXT;
TEXT textarray[50]; /*Zeilenanzahl*/
```

```

main()
{
    appl_id=appl_init(); /*Programm anmelden*/
    open_vwork(); /*Initialisierungen*/
    text_init();
    event(); /*Ereignisverarbeitung*/
    close_window(); /*Fenster schließen und abmelden*/
    v_clswwk(vdi_handle); /*schließen der 'virtual screen workstation'*/
    appl_exit(); /*Programm abmelden*/
}

open_vwork()
{
    register int i; /*Laufindex*/
    int dummy;
    /*ermitteln des phys.Workstation-Handle*/
    phys_handle=graf_handle(&dummy,&dummy,&dummy,&dummy);
    vdi_handle=phys_handle;
    for(i=0;i<10;work_in[i++]=1); /*Feld initialisieren*/
    work_in[10]=2;
    v_opnwwk(work_in,&vdi_handle,work_out); /*öffnen der 'virtual screen
                                         workstation'*/
}

text_init() /*Text-Array wird initialisiert*/
{
    register int ii;
    char zahl[2];

    anzahl=50;
    for(ii=0;ii<=49;++ii)
    {
        itoa((ii+1),zahl);
        strcpy(((textarray+ii)->textzeile),zahl);
        strcat(((textarray+ii)->textzeile),". Textzeile in unserem ersten
        Fenster");
    }
}

event() /*Fensterverwaltung*/
{
    int slize, /*relative Slize-Größe*/
        groesse; /*absolute Slize-Größe*/
    register int tt, /*Laufindex*/
        zz;

    graf_mouse(M OFF,0L); /*Mauszeiger ausschalten*/
    wind_update(TRUE); /*keine Benutzerinteraktionen erlaubt*/
    fenster(); /*Fenster erzeugen*/
    groesse=1000*20/anzahl; /*absolute Slize-Größe berechnen*/
    slize=(groesse<1000) ? groesse : 1000; /*relative Slize-Größe
                                         berechnen*/
    wind_set(wi_handle,WF_VSLSIZE,slize,0,0,0); /*Größe des vertikalen
        Schiebers setzen*/
    wind_set(wi_handle,WF_VSLIDE,0,0,0,0); /*Position des vertikalen

```

```

                                Schiebers setzen*/
for(tt=0,zz=0;(zz<=GZEILEN)&&(tt<=anzahl);++zz) /*Ausgabe der
                                                gefundenen Einträge*/
{
    v_gtext(vdi_handle,(SP(1)+x2),(ZE(zz)+y2),
            ((textarray+tt)->textzeile));
    ++tt;
}
wind_update(FALSE); /*Benutzerinteraktionen wieder erlaubt*/
graf_mouse(M_ON,0L); /*Mauszeiger einschalten*/
top=0;
do
{
    evnt_mesag(msgbuff);
    graf_mouse(M_OFF,0L);
    wind_update(TRUE);
    switch(msgbuff[0]) /*was wurde ausgelöst*/
    {
        case WM_VSLID: /*Schieber wurde ausgewählt*/
            neu_top(); /*neue erste sichtbare Zeile berechnen*/
            break; /*und Text neu ausrichten*/
        case WM_ARROWED: /*Pfeil wurde ausgewählt*/
            pfeile(); /*neue Schieberposition setzen*/
            break; /*und Text neu ausrichten*/
        default:
            break;
    }
    wind_update(FALSE);
    graf_mouse(M_ON,0L); *
}while(msgbuff[0] != WM_CLOSED); /*solange Schließfeld nicht betätigt
                                wurde*/
}
fenster()
{
    static char name[]="Unser 1. Fenster"; /*String für Fenstertitel*/
    /*String für Infozeile*/
    static char info[]="Es befinden sich 50 Textzeilen im Text-Array";

    wind_get(DESK,4,&x1,&y1,&w1,&h1); /*Arbeitsbereich des Desktop
                                ermitteln*/
    wind_calc(1,WI_KIND,x1,y1,w1,h1,&x2,&y2,&w2,&h2); /*max. Größe des
                                Fenster-Arbeitsbereiches*/
    wind_calc(0,WI_KIND,x2,y2,w2,h2,&x3,&y3,&w3,&h3); /*gesamte
                                Arbeitsfläche des Fensters*/
    wi_handle=wind_create(WI_KIND,x3,y3,w3,h3); /*Platz für Fenster
                                reservieren*/
    wind_set(wi_handle,2,name,0,0); /*Titelzeile setzen*/
    wind_set(wi_handle,3,info,0,0); /*Infozeile setzen*/
    wind_open(wi_handle,x3,y3,w3,h3); /*Fenster öffnen*/
    pxyarray[0]=x2; /*Koordinaten ins Rechteck-Koordinatenfeld*/
    pxyarray[1]=y2; /*übertragen*/
    pxyarray[2]=(x2+(w2-1));
    pxyarray[3]=(y2+(h2-1));
}

```



```

    clear_window(); /*Fensterinhalt löschen*/
}
neu_top()
{
    int posneu; /*neuer erster Zeilenindex*/
    long posakt; /*aktuelle Schieberposition*/

    posakt=msgbuff[4];
    posneu=(int)(posakt*(anzahl-GZEILEN)/1000); /*neuen ersten Zeilenindex
                                                    berechnen*/
    wind_set(wi_handle,WF_VSLIDE,((int)posakt),0,0,0); /*aktuelle
                                                         Schieberposition setzen*/
    top=posneu; /*Zeilenindex sichern*/
    clear_window(); /*Fensterinhalt löschen*/
    show_text(); /*Text einblenden*/
}
pfeile()
{
    switch(msgbuff[4]) /*was wurde gewählt*/
    {
        case 0: /*oberer Scroll-Balken angeklickt*/
            top-=20; /*neuer Index der ersten sichtbaren Zeile*/
            if(top<0)
                top=0;
            set_vslid(); /*Schieber neu setzen*/
            break;
        case 1: /*unterer Scroll-Balken angeklickt*/
            top+=21; /*neuer Index der ersten sichtbaren Zeile*/
            if(top>(anzahl-20))
                top=(anzahl-20);
            set_vslid(); /*Schieber neu setzen*/
            break;
        case 2: /*oberer Pfeil angeklickt*/
            if(top>0) /*Textanfang noch nicht erreicht*/
            {
                --top; /*neuer Index der ersten sichtbaren Zeile*/
                set_vslid(); /*Schieber neu setzen*/
            }
            break;
        case 3: /*unterer Pfeil angeklickt*/
            if(top<(anzahl-GZEILEN)) /*Textende noch nicht erreicht*/
            {
                ++top; /*neuer Index der ersten sichtbaren Zeile*/
                set_vslid(); /*Schieber neu setzen*/
            }
            break;
    }
}
set_vslid()
{
    int posschieber; /*neue Schieberposition*/
    clear_window(); /*Fensterinhalt löschen*/
    show_text(); /*Text einblenden*/
}

```

```

/*neue Schieberposition berechnen*/
posschieber=(int)(1000*((double)top/((double)(anzahl-GZEILEN))));
wind_set(wi_handle,WF_VSLIDE,posschieber,0,0,0); /*neue
Schieberposition setzen*/
}
show_text()
{
    register int tt, /*Laufindex*/
               zz;

    for(tt=top,zz=0;(zz<=GZEILEN)&&(tt<=anzahl);++zz) /*die jeweils
                                                         sichtbaren Zeilen einblenden*/
    {
        v_gtext(vdi_handle,(SP(1)+x2),(ZE(zz)+y2),
                ((textarray+tt)->textzeile));
        ++tt;
    }
}
clear_window()
{
    vsf_color(vdi_handle,0); /*Farbindex 'weiß' wird gewählt*/
    vsf_interior(vdi_handle,1); /*Fülltyp 'solid' wird gewählt*/
    v_bar(vdi_handle,pxyarray); /*Rechteck wird ausgefüllt*/
}
close_window()
{
    wind_close(wi_handle); /*Fenster schließen*/
    wind_delete(wi_handle); /*Fenster löschen*/
}

```

13.6 Verwaltung der Dialogboxen

Sie kennen sicher die Dialogboxen des GEM-Desktop, mit denen Sie auf komfortable Art und Weise Einstellungen und Texteditierungen vornehmen können. Die Dialogboxen selbst werden in der Regel mit Hilfsprogrammen erstellt und anschließend in das eigentliche Programm eingefügt. Solche Hilfsprogramme zur Erstellung von Ressourcen werden Resource-Construction Sets genannt. Diese jedoch an dieser Stelle zu erklären würde eindeutig den Rahmen des Buches sprengen. Wichtig ist eigentlich im Moment nur, daß das RCS die oben beschriebenen Datenstrukturen zusammenstellt und Sie das nicht mühsam von Hand tun müssen.

Wenn Sie ein solches Resource-File erstellt haben, in dem natürlich auch mehrere Dialogboxen oder auch Pull-Down-Menüs enthalten sein können, so wird es zunächst mit dem `RSRC_LOAD`-Befehl (siehe Anhang und Beispielprogramm) zu dem GEM-Programm hinzugeladen.

Eine weitere Möglichkeit (insbesondere für die Menübäume) besteht darin, die erforderlichen Datenstrukturen für die Objekte in C oder einer anderen Sprache direkt zu definieren. Das ist etwas komplizierter, führt aber zu weniger Dateien. Vor allem aber geht das auch ohne RCS. Doch dazu später mehr.

Zur Darstellung einer Dialogbox wird zunächst die Adresse ihrer Daten ermittelt. Hierzu dient ein weiterer Resource-Befehl: `RSRC_GADDR`. Mit dieser Funktion können die Adressen von freien Strings, die z.B. die Texte von INFO- oder Titel-Zeilen der Fenster enthalten, oder aber auch Objektbäumen (z.B. Pull-Down-Menues) oder eben unsere Dialogboxen ermittelt werden. In unserem Falle wäre der `RSRC_GADDR`-Befehl mit `"re_gtype=0"` aufzurufen, da es sich bei unserer Dialogbox um einen Objektbaum handelt. Als Variable `"re_gindex"` übergeben wir der `RSRC_GADDR` die Nummer unseres Objektbaums, die uns das Resource-Construction-Set liefert. Genauer gesagt: Für jedes definierte Objekt wurde in einer Header-Datei eine Konstante definiert. Diese Konstante stellt eben die gesuchte Nummer dar. Jetzt erst können wir die eigentliche Darstellung der Dialogbox vornehmen.

Hierzu wird zunächst die `FORM_DIAL`-Routine aufgerufen, mit der Speicherplatz für die Dialogbox reserviert wird (Aufruf mit `intin(0)=0`). Anschließend wird dieselbe Routine nochmals aufgerufen, um eine sich öffnende Box darzustellen (optional, `initin(0)=1`). Jetzt sollten Sie die Funktion `OBJC_DRAW` ansteuern, damit die Dialogbox auf dem Bildschirm dargestellt wird.

Für den Benutzer ist bisher nicht viel passiert: Ein Rechteck hat sich geöffnet und den Blick auf die Dialogbox freigegeben. Änderungen darin sind noch nicht möglich. Mittels der `FORM_DO`-Routine kann nun die Überwachung der Dialogbox ans GEM abgegeben werden. Das GEM verwaltet alle Aktionen, die mit der Maus durchgeführt werden, ebenso alle Texteingaben. Der Benutzer kann also Knöpfe anklicken (die dabei invertiert werden), Eingabefelder editieren usw. Erst wenn der Benutzer ein Exit-Objekt anwählt (Objekt, das bei Anwahl zum Beenden des Formulars führt), kehrt diese Funktion wieder zurück. Sie liefert dabei die Nummer desjenigen Objektes, das für den Abbruch des Dialogs verantwortlich war (meistens ein Knopf). Nun wird wieder die `FORM_DIAL`-Routine aufgerufen, diesmal jedoch, um einen schrumpfenden Kasten zu zeichnen, und ein letztes Mal, um den zuvor reservierten Speicherbereich wieder freizugeben. Die Eingaben des Benutzers (wenn denn welche stattgefunden haben) stehen in den vorher definierten Strings.

Schon wesentlich schwieriger ist es, wenn ein Objektstatus oder gar ein eingegebener Text aus einem bestimmten Objekt des Formulars gelesen werden muß. Hierzu sollte zunächst ein wenig über die Struktur von Objektbäumen bekannt sein. Ein Objektbaum ist aus mehreren Objekten aufgebaut, die durch

bestimmte Vektoren (Zeiger) miteinander verbunden sind. Jedes Objekt kann Unterobjekte besitzen, die in ihm enthalten sind (z.B. ein Rechteck, das Text enthält etc.). Desweiteren umfaßt ein einzelnes Objekt mehrere Bytes, die sein Aussehen und seine Position bestimmen. Die folgende Tabelle verdeutlicht den genauen Aufbau dieser Objekte:

1. Wort	(ob_next)	Zeiger auf das nächste Objekt
2. Wort	(ob_head)	Zeiger auf das erste Objekt im Objektbaum
3. Wort	(ob_tail)	Zeiger auf das letzte Objekt im Objektbaum
4. Wort	(ob_type)	Objekt-Typ
5. Wort	(ob_flags)	besondere Objekteigenschaften
6. Wort	(ob_state)	Objekt-Status
7.+8. Wort	(ob_spec)	erfüllt je nach Objekt-Typ verschiedene Funktionen
9. Wort	(ob_x)	relative x-Koordinate zum übergeordneten Objekt
10. Wort	(ob_y)	relative y-Koordinate zum übergeordneten Objekt
11. Wort	(ob_width)	Breite des Objekts
12. Wort	(ob_height)	Höhe des Objekts

Wichtig für uns an dieser Tabelle ist lediglich der genauere Aufbau des Objektstatus, da alle anderen Einstellungen eines Objektes einmal mit dem Resource-Construction-Set durchgeführt und meist nicht wieder verändert werden. Immer jedoch, wenn z.B. ein Knopf angewählt wird, ändert sich ein bestimmtes Bit im Objekt-Status, an welchem wir auch zum Ende einer Dialogdurchführung erkennen können, welches Objekt (Knopf) angewählt ist, welches nicht.

Wenn wir den Status eines Objektes ("ob_state") ermitteln wollen, so verschaffen wir uns zunächst mit `RSRC_GADDR` die Adresse des ersten Objektes unseres Objektbaumes (auch Wurzelobjekt genannt). Anschließend multiplizieren wir die Nummer (Index) des Objektes, dessen Status wir zu ermitteln gedenken, mit 24, da ja die Eintragung eines einzelnen Objektes 24 Bytes (12 Wörter) lang ist. Damit sind wir in Besitz der Objektadresse, der Adresse also, bei der die obige Definition beginnt. Schließlich addieren wir zu dieser Adresse noch den Wert 12, um das sechste Wort, also die Adresse des Objektstatus zu bekommen. Zusammenfassend verwenden wir die folgende Formel:

$$\text{status_adr} = \text{obj_tree_adr} + 24 * \text{obj_nummer} + 12$$

Bei dieser Formel ist `obj_tree_adr` die Adresse des Objektbaumes (durch `RSRC_GADDR` ermittelt) und `obj_nummer` die Nummer (oder der Index) des Objekts, dessen Status wir ermitteln wollen. Die Adresse, die sich ergibt, braucht dann nur noch, z.B. mit `DPEEK` in BASIC, ausgelesen zu werden. Wurde der Status eines Objektes ermittelt, so repräsentiert das nullte. Bit des

Status-Wortes, ob das entsprechende Objekt angewählt ist (Bit = 1) oder nicht (Bit = 0). Damit hätten wir dieses Problem ebenfalls hinter uns gebracht.

Noch schwieriger ist es, wenn Texte aus Dialogboxen gelesen werden müssen. Hier muß nämlich zunächst der Zeiger auf eine sogenannte TED-Info-Struktur ermittelt werden. Dieses geschieht im Prinzip genauso, wie wir es schon mit der Adresse des Statuswortes gemacht haben. Wir erhalten hier die Formel

```
"ted_info_adr = obj_tree_adr + 24*obj_nummer + 14"
```

Wobei "obj_tree_adr" wieder die durch RSRC_GADDR ermittelte Adresse des Objektbaums und "obj_nummer" die Nummer des Objektes ist, dessen Textinhalt wir ermitteln. Der Inhalt der Adresse, die wir jetzt erhalten, ist erst ein weiterer Zeiger auf den eigentlichen Text. In BASIC erhielten wir also das erste Byte eines Dialogbox-Textes durch die Formel

```
byte = PEEK(LPEEK(LPEEK("obj_tree_adr + 24*obj_nummer+14"))).
```

Zugegeben ein komplizierter Vorgang, aber gerade dadurch wird die Verwaltung von Texten, gerade in C, wo das Ganze mit Strukturen programmiert wird, enorm flexibel.

Neben den AES-Routinen zur Dialogabwicklung mit selbst kreierte Objektbäumen gibt es solche, die bereits mit (fast) fertigen Dialogboxen arbeiten. So gibt es beispielsweise den ALERT-Befehl, mit dem Sie auf sehr einfache Art und Weise kleinere Dialoge durchführen können.

Auch hier können wir das Resource-Construction-Set (RCS) zu Hilfe ziehen, wenn wir die Texte unserer Dialogbox entwerfen. Hierzu erstellen wir mit dem RCS sogenannte freie Strings. (Strings, die in keinen Objektbaum gebunden sind.) Sie werden gemeinsam mit den anderen RCS-Definitionen durch RSRC_LOAD in den Speicher geladen. In diesem Fall wird der Alert-Routine die durch RSRC_GADDR (mit "re_gindex=15") ermittelte Adresse übergeben.

Es gibt eine weitere, sehr nützliche fertige Dialogbox, die sogenannte File-Selector-Box zur Auswahl beliebiger Files auf Disk. Die genauen Parameter dieser Funktion FSEL_INPUT können Sie im Anhang nachschlagen.

13.7 Programmierung der Pull-Down-Menüs

Kommen wir nun zum letzten wesentlichen Punkt der GEM-Programmierung, die Verwaltung der Pull-Down-Menüs. Sie alle kennen die verschiedenen Titel in der ersten Zeile des Bildschirms, die bei Berührung mit der Maus ein kleines

Fenster mit den sogenannten Menü-Einträgen sichtbar werden lassen. Diese Menüeinträge können nun wiederum angewählt werden, indem Sie mit der Maus auf sie zufahren.

Die Programmierung der Pull-Downs-Menüs gestaltet sich dabei mehr als einfach. Auch hier wird durch den `RSRC_LOAD`-Befehl die Resource geladen. Jetzt ermitteln wir wieder die Adresse des Pull-Down-Menüs (`RSRC_GADDR`), denn auch dieses ist, wie die Dialogboxen, ein Objektbaum, und übergeben sie der Funktion `MENU_BAR`. Trifft das Programm nun auf einen Befehl der `EVNT`-Gruppe (meistens `EVNT_MULTI`), so wird unter anderem auch auf das Anwählen eines Pull-Down-Menüs gewartet. Dabei legt `EVNT_MULTI` die Nummer des angewählten Menütitels sowie die Nummer des Menüeintrages im uns schon bekannten Message-Buffer wie folgt ab:

Inhalt des Message-Buffers:

Wort 0 10 (Event-Nummer)
Wort 3 Objekt-Index des Menütitels
Wort 4 Objekt-Index des angewählten Menüeintrags

Jetzt hat der Programmierer nur noch die Aufgabe, den invertiert dargestellten Menütitel mit `MENU_TNORMAL` wieder normal darzustellen. Anschließend springt er zu der Routine, die er für den jeweiligen Menüpunkt vorgesehen hat.

Weiterhin hat der Benutzer die Möglichkeit, durch `MENU_ICHECK` Menüeinträge abzuheben, bzw. Haken wieder zu entfernen. Auch können Menüeinträge durch `MENU_IENABLE` unwählbar (disable) oder anwählbar (enable) gemacht werden. Schließlich bietet das AES als letzte Möglichkeit der Menümanipulation die Änderung der Texte einzelner Menüeinträge durch den `MENU_TEXT`-Befehl.

13.8 Objekte ohne Resource-Dateien

Wie bereits erwähnt, ist es möglich, die Objekte des GEM zu nutzen, ohne ein RCS zu besitzen, um damit RSC-Dateien zu erzeugen. Alle Objekte stellen eigentlich nur Strukturen im RAM dar. Diese können Sie natürlich auch direkt in C definieren (in Pascal verwenden Sie dazu Records). Wie die einzelnen Strukturen aussehen, ist in den vorhergehenden Kapiteln beschrieben worden. Viele Compiler liefern auch schon vordefinierte Typen für diese Strukturen mit, so daß Sie diese nur noch mit Werten füllen müssen. Auch diese Werte können Sie den obigen Kapiteln entnehmen. Einzige Ausnahme: Pointer, die

auf andere Strukturen zeigen, können Sie natürlich nicht setzen, denn die Adressen der Strukturen werden natürlich erst zur Laufzeit bekannt. Alles was Sie also noch tun müssen, ist am Anfang des Hauptprogramms die noch fehlenden Adressen einzutragen. Achtung: Mit Adressen sind hier die echten RAM-Adressen gemeint und nicht die Verweise in den Objektbäumen, die mit einer einfachen Numerierung der Objekte arbeiten.

Wenn Sie auf diese sehr direkte (und leider auch umständliche Art) mit Objekten arbeiten, können Sie sich übrigens den Aufruf der Funktion `RSC_GADDR` sparen, denn die Adressen der Objekte können Sie ja ganz einfach durch die Adressen der Speicherstrukturen ermitteln.

13.9 Ein Beispiel-Accessory

Viele der oben vorgestellten Techniken leuchten nicht auf den ersten Blick ein. Um dem ein wenig abzuhelpen, zeigen wir Ihnen hier ein Accessory. Es hat natürlich keine Menüleiste, aber dafür zeigt es zum Beispiel den Umgang mit Objekten (ohne RCS). Und so ganz nebenbei ist es auch noch nützlich.

In den älteren Versionen des TOS ist es leider nicht möglich, einen einmal gewählten Ordnernamen umzubenennen. Bisher mußte man entweder den Namen mittels eines Diskmonitors per Hand ändern oder aber, falls kein Diskmonitor zur Verfügung stand, einen neuen Ordner erzeugen, den gesamten Inhalt des alten Ordners in den neuen kopieren und anschließend den alten Ordner samt Inhalt löschen. Um diese Prozedur zu umgehen, haben wir ein Programm geschrieben, mit dem man die Namensumbenennung bequemer durchführen kann. Der einzige Wermutstropfen ist der, daß nach durchgeführter Namensumbenennung der Rechner neu gebootet werden muß, damit der geänderte Directory-Aufbau auch erkannt und verwaltet wird.

Hat man das Accessory in der Menüleiste angeklickt, wird man als erstes nach dem alten und dann nach dem neuen Ordnernamen gefragt. Dazu muß das Laufwerk, auf dem sich der Ordner befindet, als aktuelles Laufwerk angemeldet sein, d.h. man muß sich beim Aufruf mindestens im Wurzelverzeichnis der zu ändernden Diskette bzw. Partition befinden.

Sind die Namen eingegeben, wird das aktuelle Laufwerk ermittelt und dessen Sektorenzahl berechnet (mittels der Informationen des BIOS-Parameter-Blocks BPB). Nun wird die Diskette sektorenweise in einen Puffer eingelesen, der dann nach dem zu ändernden Namen durchsucht wird. Ist er gefunden worden, wird ab der String-Anfangsadresse der neue Name in den Puffer eingetragen und der nun geänderte Puffer auf die Diskette zurückgeschrieben.

Zugegeben, das ist nicht gerade die feine englische Art, aber dieses Accessory soll ja auch in erster Linie ein paar Programmiertechniken (hier in C) demonstrieren.

Um die Berechnung der Sektorenzahl aus den Angaben des BPB besser nachvollziehen zu können, hier dessen Aufbau:

```
typedef struct bpb
{
    int sekteize;    /* Sektorgröße in Bytes */
    int clustsize;   /* Cluster-Größe in Sektoren */
    int clustbysize; /* Cluster-Größe in Bytes */
    int dirsize;     /* Directory-Größe in Sektoren */
    int fatsize;     /* Größe der File-Allocation-Tabelle */
    int fatsek;      /* Sektornr. der 2. FAT */
    int clustsek;    /* Sektornr. des 1. Daten-Cluster */
    int clustzahl;   /* Gesamtzahl der Daten-Cluster auf Disk */
    int flags;       /* Flags */
}BPB;
```

Mit der Funktion Getbpb() bekommt man die Adresse mittels eines Zeigers auf den BIOS-Parameter-Block des unter disk angegebenen Laufwerkes. Mit der Funktion Rwabs ist es möglich, Sektoren auf Disketten zu schreiben bzw. von Disketten zu lesen. Den Aufruf dieser Funktionen können Sie dem Anhang entnehmen.

Beachten Sie im Programmlisting vor allem die Definitionen der Speicherstrukturen am Anfang. Hier ist nicht einmal die Initialisierung der Pointer nötig, da die Strukturen alle als global deklariert wurden. Bei Arrays reicht der Name desselben, um dessen Adresse bekanntzugeben. (Siehe teds1, dort werden die Adressen der Zeichenarrays string2, string3 und string4 direkt eingesetzt.).

Die Funktion dialog() zeigt Ihnen, wie die so definierten Strukturen eingesetzt werden, um in Zusammenarbeit mit den AES-Funktionen eine Dialogbox zu bilden. Beachten Sie zum Beispiel, daß die angeklickten Buttons nach der Rückkehr aus dem Formular wieder auf normale Darstellung zurückgebracht werden müssen, da sie sonst beim nächsten Zeichnen immer noch invertiert erscheinen würden.

Natürlich finden Sie auch den klassischen Aufbau eines Accessories in der Hauptroutine wieder: Anmelden, Menüeintrag erzeugen und ab in die Endlosschleife.

Das Accessory wurde übrigens für den Megamax-C-Compiler geschrieben; wenn Sie mit anderen Compilern arbeiten, können sich die "include"-Dateien

ändern und eventuell die Namen von Funktionen oder Typen. Das können Sie dem jeweiligen Handbuch entnehmen.

```

/* Ordner umbenennen */
#include<stdio.h>
#include<osbind.h>
#include<gemdefs.h>
#include<gembind.h>
#include<define.h>
#include<obdefs.h>
#include<string.h>
int contrl [12], /*GEM-Arrays für VDI und AES*/
    intin [128],
    ptsin [128],
    intout [128],
    ptsout [128],
    work_in [11],
    work_out[57],
    msgbuff [8],
    vdi_handle, /*VDI-Handle*/
    phys_handle, /*physikalisches Workstation-
Handle*/
    appl_id, /*Programmkennung*/
    menu_id; /*Nummer des Menüeintrags*/

char name_alt[12],
    name_neu[12];

char string1[]="Eingabe des alten Ordnersnamen"; /*Stringaufbau der
Dialogbox1*/
char string2[]="@";
char string3[]="Dateiname : -----";
char string4[]="FFFFFFFFFFFF";
char string5[]=" Fertig ";
TEDINFO teds1[]={
string2,string3,string4,3,6,0,0x1180,0x0,-1,12,24}; /*Tedinfor der
Dialogbox1*/
OBJECT baum1[]={
-1, 1, 3,G_BOX,NONE,OUTLINED,0x21100L,150,100,300,160,
2,-1,-1,G_STRING,NONE,NORMAL,string1,40,16,200,16,
3,-1,-1,G_FTEXT,EDITABLE,NORMAL,&teds1[0],50,90,192,16,
0,-1,-1,G_BUTTON,SELECTABLE|DEFAULT|EXIT|LASTOB,NORMAL,
string5,120,130,64,16};
char string6[]="Eingabe des neuen Ordnersnamen"; /*Stringaufbau der
Dialogbox2*/
char string7[]="@";
char string8[]="Dateiname : -----";
char string9[]="FFFFFFFFFFFF";
char string0[]=" Fertig ";
TEDINFO teds2[]={
string7,string8,string9,3,6,0,0x1180,0x0,-1,12,24}; /*Tedinfor der
Dialogbox2*/

```

```

OBJECT baum2[]={
-1,1,3,G_BOX,NONE,OUTLINED,0x21100L,150,100,300,160,
2,-1,-1,G_STRING,NONE,NORMAL,string6,40,16,200,16,
3,-1,-1,G_FTEXT,EDITABLE,NORMAL,&teds2[0],50,90,192,16,
0,-1,-1,
G_BUTTON,SELECTABLE|DEFAULT|EXIT|LASTOB,NORMAL,string0,120,130,64,16);
main()
{
    appl_id=appl_init();                                /*Programm
                                                         anmelden*/
    open_vwork();                                       /*Workstation öffnen*/
    menu_init();                                       /*Menüeintrag
                                                         anmelden*/
    loop();                                           /*Hauptprogramm*/
}
open_vwork()
{
    register int i;                                    /*Laufindex*/
    int dummy;                                         /*Dummy-Wert*/
                                                         /*physikalisches
                                                         Workstation-Handle*/
    phys_handle=graf_handle(&dummy,&dummy,&dummy,&dummy);
    vdi_handle=phys_handle;
    for(i=0;i<10;work_in[i++]=1);                    /*Feld initialisieren*/
    work_in[10]=2;
    v_opnvwk(work_in,&vdi_handle,work_out);           /*öffnen der 'virtual
                                                         screen workstation'*/
}
menu_init()
{
    menu_id=menu_register(appl_id," Ordner-Rename"); /*Accessory-Ein-
                                                         trag anmelden*/
    if(menu_id== -1)                                  /*Fehler aufgetreten*/
    {
        form_alert(1,"[1][Kein Platz mehr in der Menüleiste][ OK ]");
        appl_exit();                                  /*Programm beenden*/
    }
}
loop()
{
    char buff1[513],
        buff2[513],
        ring[14];

    register int xx,tt,zz;

    int sektoren,
        laenge,
        start,
        disc,
        flag,
        *biospbptr;

```

```

while(TRUE)                                     /*Endlosschleife*/
{
    graf_mouse(0,0L);
    evnt_mesag(msgbuff);                        /*Ereignisabfrage*/
    if((msgbuff[0]==AC_OPEN)&&(msgbuff[4]==menu_id)) /*wurde Accessory
                                                    gewählt*/
    {
        wind_update(TRUE);                     /*Benutzeraktion unter-
                                                binden*/
        dialog();                             /*Ordernamen ermitteln*/
        wind_update(FALSE);                   /*Benutzeraktion er-
                                                lauben*/

        flag=FALSE;
        disc=Dgetdrv();                       /*aktuelles Laufwerk*/
        biospbptr=(int *)Getbpb(disc);        /*Diskinformationen in
                                                BPB-buff1 laden*/
                                                /*Gesamtzahl der Sektoren
                                                auf Disk*/
        sektoren=((*(biospbptr+1))*(*(biospbptr+7)))+(*(biospbptr+6));
        xx=0;
        while((xx<sektoren)&&(flag==FALSE))
        {
            Rwabs(0,buff1,1,xx,disc);          /*jeweils einen Sektor
                                                einlesen*/
            laenge=strlen(name_alt);           /*Länge des alten
                                                Ordernamens*/
            for(zz=0;zz<512;++zz)
                *(buff2+zz)=*(buff1+zz);      /*Pufferinhalt sichern*/
            tt=0;
            while((tt<512)&&(flag==FALSE))      /*Pufferinhalt
                                                überprüfen*/
            {
                for(zz=0;zz<laenge;++zz)
                    *(ring+zz)=*(buff2+tt+zz); /*entsprechend der
                                                Namenslänge*/
                *(ring+zz+1)='\0';             /*Bytes in Zwischenpuffer
                                                kopieren*/
                if((strcmp(name_alt,ring))==0) /*und mit Namen
                                                vergleichen*/
                {
                    start=tt;                  /*Name gefunden->
                                                Startadresse im*/
                    flag=TRUE;                 /*Puffer merken und
                                                Gefunden-Flag setzen*/
                }
                else
                    flag=FALSE;
                tt++;
            }
            xx++;
        }
        if(flag==TRUE)                         /*Name gefunden*/
        {

```

```

    zz=0;
    while((* (name_neu+zz)) != '\0') /*neuen Namen ab Start-
                                     adresse in Puffer kopieren*/
    {
        *(buff1+start+zz)=*(name_neu+zz);
        zz++;
    }
    Rwabs(1,buff1,1,xx-1,disc); /*und korrigierten Sektor
                                zurückschreiben*/
    form_alert(1,"[1][Ordner umbenannt| |Bitte Rechner neu
                                booten][ OK ]");
}
else
    form_alert(1,"[1][Ordner nicht gefunden][ OK ]");
}
}
}

dialog()
{
    GRECT box; /*Boxkoordinaten-
                Struktur*/

    char buffer[15];
    register int xx; /*Laufindex*/
    int laengel,
        laenge2;

    for(xx=0;xx<12;++xx) /*Dateiname-Array wird
                           gelöscht*/
        name_alt[xx]=EOS;

    form_center(baum1,&box.g_x,&box.g_y,&box.g_w,&box.g_h); /*Dialogbox zentrieren*/
    graf_mbox(20,20,50,16,320,200); /*ausdehnendes Rechteck
                                       zeichnen*/
    /*Bildschirmbereich wird
    reserviert*/
    form_dial(FMD_START,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
    /*ausdehnendes Rechteck
    zeichnen*/
    form_dial(FMD_GROW,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
    /*Dialogbox darstellen*/
    objc_draw(baum1,0,MAX_DEPTH,box.g_x,box.g_y,box.g_w,box.g_h);
    form_do(baum1,0); /*Benutzereingaben
                      abhandeln*/
    baum1[3].ob_state &= ~SELECTED; /*Button neu zeichnen*/
    /*schrumpfendes Rechteck
    zeichnen*/
    form_dial(FMD_SHRINK,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
    /*Bildschirmbereich
    freigeben*/
    form_dial(FMD_FINISH,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
    strcpy(buffer,(&tedsl[0])->te_ptext); /*Texteingabe sichern*/
    strcpy(name_alt,buffer);
}

```

```

for(xx=0;xx<12;++xx)                                /*Dateiname-Array wird
                                                         gelöscht*/

    name_neu[xx]=EOS;                                  /*Dialogbox zentrieren*/
                                                         form center(baum2,&box.g_x,&box.g_y,&box.g_w,&box.g_h);
                                                         graf_mbox(20,20,50,16,320,200);
                                                         /*ausdehnendes Rechteck
                                                         zeichnen*/
                                                         /*Bildschirmbereich wird
                                                         reserviert*/
form_dial(FMD_START,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
                                                         /*ausdehnendes Rechteck
                                                         zeichnen*/
form_dial(FMD_GROW,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
                                                         /*Dialogbox darstellen*/
objc_draw(baum2,0,MAX_DEPTH,box.g_x,box.g_y,box.g_w,box.g_h);
form_do(baum2,0);                                     /*Benutzereingaben
                                                         abhandeln*/
baum2[3].ob_state &= ~SELECTED;                       /*Button neu zeichnen*/
                                                         /*schrumpfendes Rechteck
                                                         zeichnen*/
form_dial(FMD_SHRINK,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
                                                         /*Bildschirmbereich
                                                         freigeben*/
form_dial(FMD_FINISH,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
strcpy(buffer,(&teds2[0])->te_ptext);                 /*Texteingabe sichern*/
strcpy(name_neu,buffer);
if((laengel=strlen(name_neu))<11)                     /*Namen mit Leerzeichen
                                                         auffüllen*/
{
    laenge2=11-laengel;
    for(xx=0;xx<laenge2;++xx)
        *(name_neu+laengel+xx)='\ ' ;
}
}

```

13.10 GEM und BASIC

Auch in der wohl am weitesten verbreiteten Programmiersprache der Welt läßt sich das GEM nutzen. Für den Atari ST kommen vor allem das GFA- und das Omikron-BASIC in Frage. Beide bieten die GEM-Funktionen an, so daß im Prinzip kein Unterschied zu C oder Assembler besteht. Allerdings verschluckt das BASIC gerne mal ein paar Parameter, die für die meisten Fälle unwichtig sind, in besonderen Situationen jedoch wichtig sind. Die Details zum Aufruf entnehmen Sie bitte den jeweiligen Handbüchern - für eine ausführliche Darstellung ist hier leider nicht genug Platz.

Einige prinzipielle Dinge gilt es jedoch noch zu beachten. BASIC ist traditionell eine Interpreter-Sprache. Daher stellt jedes BASIC-Programm quasi ein Pro-

gramm im Programm dar, mit allen Konsequenzen für das GEM. So könnte es zum Beispiel sein, daß sich bereits das BASIC (hier vor allem GFA) als Applikation beim GEM angemeldet hat. Wenn Sie das nun in Ihrem BASIC-Programm auch noch tun, kann das Konflikte hervorrufen. Einige der GEM-Aufrufe sind daher - obwohl prinzipiell aufrufbar - in BASIC verboten. Hier gilt es die Handbücher zu studieren.

Aus einem ähnlichen Grund kann man in BASIC nicht so ohne weiteres die Accessories programmieren. Hier muß auf jeden Fall zu einem Compiler gegriffen werden.

Noch ein Tip am Rande: Das GFA-BASIC bietet viele komfortable Befehle, die die Steuerung von GEM-Elementen ohne großartige Funktionsaufrufe übernehmen können. Man denke da nur an die Befehle zur Verwaltung der Menübäume oder für die Fenster. Wer die vollen Möglichkeiten ausnutzen will, muß in der Regel auf die "richtigen" Funktionen zurückgreifen. Wenn Sie dabei einmal die Funktion `EVENT_MULTI` suchen sollten, können Sie getrost auf den Befehl `ON MENU` und seine Verwandten zurückgreifen.

13.11 Ein eigenes Desktop

Bei der Erstellung von Programmen vergleicht man oft sein eigenes mit professionellen Anwendungen. Dabei sieht man des öfteren, daß diese Programme eigene Benutzeroberflächen, sprich Desktops, verwenden. Warum also nicht auch die eigenen Programme mit solch einem Desktop bereichern? Ein Desktop hat den Vorteil, daß es vom GEM automatisch verwaltet wird. Sind auf einem Desktop z.B. Funktionstasten dargestellt, die im Programmverlauf durch Fenster oder Dialogboxen verdeckt wurden, so werden diese Tasten, nachdem die Fenster bzw. Dialogboxen geschlossen wurden, automatisch neu gezeichnet. Diese Arbeit übernimmt das GEM, wenn Sie Ihr Desktop vorher als neues Desktop angemeldet haben. Doch bevor wir zur Vorgehensweise bei der Erstellung eines eigenen Desktops kommen, vorher ein wenig zum Aufbau eines solchen.

Das Desktop ist in seiner Struktur nichts anderes als ein Objektbaum, der aus einem (meistens grauen) Hintergrund (bei hoher Auflösung) und verschiedenen Objekten (Icons, Funktionstasten etc.) besteht. Diese Objekte werden in den Objektbaum des Hintergrundes, der vom Typ `G-BOX` ist, integriert. Mit einem RCS ist die Erstellung der entsprechenden Dateien (.H und .RSC) recht einfach. Da aber dieser Hintergrund im RCS nicht in der richtigen Größe erzeugt werden kann, muß diese Größenanpassung später im Programm erfolgen. Dort werden die entsprechenden Koordinaten für Breite und Höhe, nach-

dem das Resourcefile geladen und die Adresse des Objektbaumes ermittelt wurde, mittels Strukturverweisiger übergeben. Da das Desktop eine OBJECT-Struktur aufweist, die in der Header-Datei GEMDEFS.H Ihres C-Compilers definiert ist, können die Koordinaten wie folgt übergeben werden:

```
desktop->ob_width=640;
desktop->ob_height=400;
```

wobei desktop die Objektadresse beinhaltet.

Da die anderen Objekte Elemente dieses Objektbaumes sind, der ja als neues Desktop angemeldet wurde, bekommt der Screen-Manager auch eine entsprechende Nachricht, wenn diese Elemente im Verlaufe des Programms verdeckt waren und zeichnet sie automatisch neu.

Nun werden die einzelnen Schritte zur Erstellung eines eigenen Desktops genau beschrieben. Nachdem Sie mit einem RSC ein entsprechendes Resourcefile erstellt haben (leider lassen sich diese nicht abdrucken, so daß Sie das unten stehende Programm mit etwas Phantasie als ein Beispiel sehen sollten), muß es als nächstes vom Programm geladen werden. Danach wird die Adresse des Objektbaumes ermittelt, die der "wind_set"-Routine übergeben werden muß. Mittels dieser Routine ist es überhaupt möglich, ein eigenes Desktop anzumelden. Wird diese Funktion mit dem Parameter WF_NEWDESK aufgerufen, so wird unser übergebener Objektbaum als neues Desktop angemeldet. Nun muß das neue Desktop noch durch "objc_draw" gezeichnet werden, und wir haben ein eigenes Desktop zur Verfügung, dessen Verwaltung GEM nun automatisch übernimmt.

Das unten abgedruckte Beispiel erzeugt (mit dem richtigen RSC-File) ein Desktop mit zwei Icons, Funktionstasten und einer Dialogbox zur Informationsdarstellung. Durch Druck auf die Taste F10 erscheint eine Dialogbox, die Teile des Desktops verdeckt. Wird der OK-Button betätigt, so werden diese verdeckten Teile, nachdem die Dialogbox verschwunden ist, automatisch erneuert. Daran können Sie sehen, daß der Screen-Manager seine Arbeit, das neue Desktop zu verwalten, aufgenommen hat. Durch die Taste F1 beenden Sie dieses kurze Demoprogramm.

```
#include<stdio.h>
#include<gemdefs.h>
#include<osbind.h>
#include<obdefs.h>
#include<gembind.h>
#include"desktop.h"

int vdi handle;
int contrl[11],
    intin[256],

/*Resource-Header-Datei einlesen*/
/* VDI-Handle */
/* VDI-Parameter-Array */
```

```

    intout[45],
    ptsin[256],
    ptsout[12];
OBJECT *desktop;
main()
{
    GRECT box;
    int taste;
    int abbruch=FALSE;

    open_aes();
    rsrc_gaddr(R_TREE,FDESKTOP,&desktop); /*Adresse ermitteln*/
    desktop->ob_width=640;                /*Koordinaten der Breite*/
    desktop->ob_height=400;                /*und Höhe ändern*/
    wind_set(0,WF_NEWDESK,desktop,0,0); /*als neuen Desktop anmelden*/
                                        /*und zeichnen*/
    objc_draw(desktop,ROOT,MAX_DEPTH,desktop->ob_x,desktop->ob_y,
               desktop->ob_width,desktop->ob_height);
    do
    {
        taste=(evnt_keybd()/0x100);      /*auf Taste warten*/
        switch(taste)
        {
            case 0x44:                    /*F10 wurde gedrückt*/
                open_dialog();            /*Dialogbox öffnen*/
                abbruch=FALSE;
                break;
            case 0x3B:                    /*F1 wurde gedrückt*/
                abbruch=TRUE;             /*Programm beenden*/
                break;
            default:
                abbruch=FALSE;
                break;
        }
    }while(abbruch!=TRUE);
    aes_exit();                          /*Programm ordnungsgemäß verlassen*/
}

int open_aes()
{
    extern int gl_apid;                  /*AES Applikations-ID*/
    int work_in[11],work_out[57];       /*Arrays zum öffnen*/
    register int loop;                  /*Schleifen-Index*/
    int dummy;                          /*Dummy-Return-Variable*/

    appl_init();                        /*öffnet AES*/
    if(gl_apid==-1) exit(1);            /*falls !AES Initiierung->exit*/
    if(!rsrc_load("desktop.rsc")) exit(1); /*lädt Resource-File;
                                        sonst->exit*/
    vdi_handle=graf_handle(&dummy,&dummy,&dummy,&dummy); /*holt
                                                        physikalische Geräte-Handle*/
    for(loop=0;loop<10;loop++)
        work_in[loop]=1; /*initialisiert Work-Array zum öffnen von
                        'virtual work' */
}

```



```

work_in[10]=2;
v_opnvwk(work_in,&vdi_handle,work_out); /*öffnet 'virtual work
station'*/

if(!vdi_handle) exit(1);
graf_mouse(ARROW,0L); /*wandelt Maus in Zeigerform*/
}
int aes_exit()
{
    v_clsawk(vdi_handle); /*schließt 'virtual work station'*/
    rsrc_free(); /*gibt Platz der Resource frei*/
    appl_exit(); /*schließt AES*/
    exit(0); /*Rückkehr zum Desktop*/
}
int open_dialog()
{
    GRECT box; /*benutzter Bildschirmbereich*/
    OBJECT *tree; /*Objektbaum für Dialog*/
    int button; /*Index des angewählten Knopfes*/

    rsrc_gaddr(R_TREE,ABOUT,&tree); /*Dialogbaum-Adresse*/
    form_center(tree,&box.g_x,&box.g_y,&box.g_w,&box.g_h); /*zentriert
    Dialogbox auf Bildschirm*/
    /*und gibt Werte des benutzten Bereichs zurück*/
    box_draw(tree,box.g_x,box.g_y,box.g_w,box.g_h);

    button=form_do(tree,0); /*wartet, daß Exit angewählt wird*/
    tree[button].ob_state &= ~SELECTED; /*löscht 'Ausgewählt'-Flag
    im Baum*/
    box_undraw(box.g_x,box.g_y,box.g_w,box.g_h);
}
box_draw(baum,x,y,w,h) /*übergebene Dialogbox öffnen*/
long baum;
int x,y,w,h;
{
    form_dial(FMD_START,0,0,0,0,x,y,w,h);
    form_dial(FMD_GROW,0,0,0,0,x,y,w,h);
    objc_draw(baum,ROOT,MAX_DEPTH,x,y,w,h);
}
box_undraw(x,y,w,h) /*Dialogbox schließen*/
int x,y,w,h;
{
    form_dial(FMD_SHRINK,0,0,0,0,x,y,w,h);
    form_dial(FMD_FINISH,0,0,0,0,x,y,w,h);
}

```

13.12. Automatisch startende Programme

Verlassen wir nun das interessante Gebiet der GEM-Programmierung und wenden uns wieder dem TOS zu. Wie Sie wissen, startet der ST beim Booten

nacheinander die Programme, die im Boot-Laufwerk im Ordner mit dem Namen AUTO liegen. Allerdings werden dabei zwei Bedingungen an die Programme gestellt:

1. Das Programm muß die Endung .PRG haben.
2. Das Programm darf keine GEM-Funktionen verwenden.

Das ist eigentlich ein Widerspruch, denn normalerweise ist die Endung .PRG eben für GEM-Programme vorgesehen. Im Auto-Ordner dürften aber nur solche Programme stehen, die korrekt mit der Endung TOS oder TTP versehen sind. Achten Sie also darauf, und ändern Sie ggf. die Extension, wenn Sie ein Programm in den AUTO-Ordner legen.

Die nächsten beiden Abschnitte zeigen Ihnen nützliche Anwendungen dieses Mechanismus.

13.12.1 Aktuelle Uhrzeit resetfest

Als erstes wollen wir einmal ein kleines Programm schreiben, das nach dem Einschalten oder einem Reset testet, ob das Datum und die Uhrzeit bereits gesetzt wurden, und falls dies nicht der Fall ist, eine entsprechende Eingabe verlangt. Besitzer einer Echtzeituhr brauchen dieses Programm natürlich nicht, aber bei den meisten anderen ist es sicher schon oft vorgekommen, daß man beim Programmieren nach vielen Versionen eines Programmes in der Entwicklungsphase die neuste Version sucht. Wenn man konsequent das Datum und die Uhrzeit stellt, ist das kein Problem mehr.

Wichtig für die richtige Funktion des Programmes ist, daß sich das Control-Accessory nicht auf Ihrem Boot-Laufwerk befindet, da es sonst, nachdem Sie das Datum eingegeben haben, dieses direkt wieder überschreibt.

An diesem Programm sehen Sie, daß sich auch GFA-Programme in kompilierter Version für den Auto-Ordner eignen, solange sie nicht auf grafische Elemente zurückgreifen. Im Einzelfall können Sie das bei eigenen Programmen einfach ausprobieren.

Hier nun das versprochene Programm:

```
' Programm: AUTODATE.BAS
```

```
' Das Programm testet beim Booten das Datum und fordert z.B. beim  
' Einschalten des Rechners die aktuellen Werte an
```

```

,
,
D.ate%=Xbios(23)           ! Datum und Uhrzeit holen
J.ahr%=D.ate% Div 2^25      ! Jahreszahl isolieren
If J.ahr%<>48               ! Initialwert beim Einschalten
    ! noch vorhanden
    T.ime%=D.ate% And &HFFFF ! Zeit isolieren
    D.ate%=D.ate% Div &HFFFF ! Datum isolieren
    Void Gemdos(&H2D,T.ime%) ! und setzen
    Void Gemdos(&H2B,D.ate%) !
Else                         ! sonst altes Datum
Do                           ! ja, aktuelles Datum eingeben
    Cls                     ! Bildschirm löschen
    ' BOX 35,60,252,200      ! nur, wenn das Programm nicht
    ' LINE 35,84,252,84      ! im Auto-Ordner steht
    ,
Print At(6,5);"Datum und Uhrzeit stellen:"
Print At(7,8);"Datum:      --.---.---"
Print At(7,10);"Uhrzeit:   --:--:--"
,
Print At(20,8);             ! Datum eingeben
@Z eingabe(2)               ! Tage 2stellig
Datum$=Erg$+"."
Print ".";
@Z eingabe(2)               ! Monat 2stellig
Datum$=Datum$+Erg$+"."
Print ".";
@Z eingabe(4)               ! Jahr 4stellig
Datum$=Datum$+Erg$
,
Print At(20,10);            ! Uhrzeit eingeben
@Z eingabe(2)               ! Stunden 2stellig
Uhrzeit$=Erg$+":"
Print ":";
@Z eingabe(2)               ! Minuten 2stellig
Uhrzeit$=Uhrzeit$+Erg$+":"
Print ":";
@Z eingabe(2)               ! Sekunden 2stellig
Uhrzeit$=Uhrzeit$+Erg$
,
Print At(6,14);"Eingaben richtig ? (j/n) ";Chr$(8);
Ausflag%=1                 ! Flag für Cursor an/aus
Zaehler%=0                 ! Verzögerung für blinken
Do
    Taste$=Upper$(Inkey$)   ! Taste holen
    Exit If Taste$="J" Or Taste$="N" ! richtige Taste gedrückt ?
    Add Zaehler%,1          ! sonst Zaehler erhöhen
    If Zaehler%=100         ! Verzögerung zu Ende ?
        Zaehler%=0         ! Zähler zurücksetzen
        If Aus_flag%=1      ! War Cursor an ?
            Print " ";Chr$(8); ! Leerzeichen ausgeben
            Aus_flag%=0      ! Flag für Cursor aus
        Else

```

```

        Print " ";Chr$(8);          ! Unterstrich ausgeben
        Aus_flag%=1                 ! Flag für Cursor an
    Endif
Endif
Loop
,
If Taste$="J"                       ! Eingaben richtig ?
    Settime Uhrzeit$,Datum$         ! Datum und Uhrzeit setzen
    If Datum$=Date$ And Left$(Uhrzeit$,5)=Left$(Time$,5)
        End                         ! Datum neu gesetzt ?
    Endif
Endif
,
Loop                                ! Wiederholen, bis richtige
,                                   ! Eingabe
Endif
,
Procedure Z_eingabe(Anzahl%)        ! Zahleneingabe mit anzahl%
,                                   ! Stellen
,
Erg$=""                             ! Ergebnis-String löschen
For Lauf%=1 To Anzahl%             ! insgesamt anzahl% Stellen
    Ausflag%=1                     ! Flag für Cursor an setzen
    Zaehler%=0                     ! Zähler für Verzögerung löschen
    Do
        Taste$=Inkey$              ! Taste holen
        Exit If Taste$>="0" And Taste$<="9" ! Ziffer ?
        Add Zaehler%,1              ! Zähler erhöhen
        If Zaehler%=100              ! Verzögerung am Ende ?
            Zaehler%=0              ! Zähler löschen
            If Aus_flag%=1           ! Cursor an ?
                Print " ";Chr$(8);  ! Leerzeichen ausgeben
                Aus_flag%=0         ! Flag für Cursor aus
            Else                     ! Cursor aus ?
                Print " ";Chr$(8);  ! Unterstrich ausgeben
                Aus_flag%=1         ! Flag für Cursor an
            Endif
        Endif
    Loop
,
Print Taste$;                       ! Zeichen ausgeben
Erg$=Erg$+Taste$                    ! und an Ergebnis-String anhängen
Next Lauf%
Return

```

Am Anfang des Programmes wird zunächst einmal mit der Xbios(23)-Betriebssystemfunktion das zur Zeit eingestellte Datum abgefragt. Wenn der Rechner gerade eingeschaltet wurde, steht für das Jahr der Wert 48, den wir abfragen. Ist das Jahr ungleich diesem Wert, so wurde bereits einmal das aktuelle Datum und die Uhrzeit eingegeben (z.B. nach einem Reset), und es muß nur noch mit den beiden GEMDOS-Funktionen gesetzt werden. Ansonsten er-

folgt die Eingabemöglichkeit. Dazu haben wir eine kleine Prozedur geschrieben, der man die Anzahl der einzugebenden Ziffern übergeben kann. Die Prozedur sorgt für das kontinuierliche Blinken des Cursors und übergibt das Ergebnis in der Variablen Erg\$.

Nachdem Sie die Abfrage nach richtiger Eingabe mit j beantwortet haben, machen wir uns eine kleine Besonderheit des Settime-Befehls zunutze. Wenn man ein falsches Datum übergibt, so wird dies nicht übernommen. Daher kann mit der folgenden Abfrage leicht die richtige Eingabe überprüft werden, ohne eine eigene Routine dafür zu schreiben (z.B. 30.02).

Damit Sie verstehen, warum wir hier die Zeit gleich mit mehreren Betriebssystemroutinen setzen, wollen wir Ihnen die Hintergründe dazu liefern. Als erstes stellen wir Ihnen einmal die relevanten Funktionen mit genauer Parameterbeschreibung vor.

```
GFA: Void Xbios(22,L:Tida%)
C:  long tida;
   Settime(tida);      oder Xbios(22,tida);
ASS: move.l  tida,-(sp)
      move.w  #22,-(sp)
      trap   #14
      addq.l  #6,sp
```

Der Parameter tida ist bitweise aufgebaut und enthält verschlüsselt die Informationen für Zeit und Datum wie folgt:

Bit	Bedeutung
0- 4	Sekunden in Zweierschritten
6-10	Minuten
11-15	Stunden
16-20	Tag
21-24	Monat
25-31	Jahr minus 1980

Die entsprechend entgegengesetzte Abfrage der Uhrzeit erfolgt mit der Xbios(23)-Funktion Gettime. Der zurückgegebene Longwert ist äquivalent aufgebaut.

Bei den GEMDOS-Routinen werden jeweils nur Worte übergeben; so sind auch zwei verschiedene Routinen für Datum und Uhrzeit vorhanden:

Datum:

```
GFA: Void Gemdos(&H2B,D.ate%)
C:   int date;
      Tsetdate(date);           oder  gemdos(0x2B,date);
ASS: move.w   date,-(sp)
      move.w   #$2B,-(sp)
      trap     #1
      addq.l   #4,sp
```

Dabei ist date ein 16-Bit-Wert mit folgender Bedeutung:

Bit	Bedeutung
0- 4	Tag
5- 8	Monat
9-15	Jahr minus 1980

Die Umkehrfunktion, also das Lesen des Datums erfolgt mit GEMDOS(0x2A).

Zeit:

```
GFA: Void Gemdos(&H2D,T.time%)
C:   int time;
      Tsettime(time);           oder  gemdos(0x2D,time);
ASS: move.w   time,-(sp)
      move.w   #$2D,-(sp)
      trap     #1
      addq.l   #4,sp
```

Dabei ist time ein 16-Bit-Wert mit folgender Bedeutung:

Bit	Bedeutung
0- 4	Sekunden in Zweierschritten
5-10	Minuten
11-16	Stunden

Auch hier ist der Aufbau mit der XBIOS-Funktion identisch, jetzt jedoch mit dem unteren Wort. Die Umkehrfunktion zum Lesen der Zeit erreichen Sie mit GEMDOS(0x2C).

Nun werden Sie sich fragen, warum wir ausführlich beide Möglichkeiten zum Stellen der Uhr benutzt haben. Im Atari ST wird nicht nur eine Uhr verwaltet, sondern zwei. Die eine, die im Tastaturprozessor in Echtzeit und nach einem

Reset weiterhin richtig läuft, ist eine richtige Hardware-Uhr, die mit den XBIOS-Funktionen gesetzt und gelesen werden kann. Daneben existiert noch eine weitere, softwaremäßig realisierte Uhr, die durch die GEMDOS-Funktionen beeinflusst wird. Diese ist nicht resetfest, aber aus dieser Uhr bezieht das Betriebssystem die Informationen beispielsweise zur Datierung von Dateien. Sollten Sie also einen Reset gemacht haben, läuft die XBIOS-Uhr weiterhin richtig, so daß unser Programm diesen Wert ausliest und, da dieser vom Aufbau her völlig identisch ist mit den Werten für die GEMDOS-Funktionen, diese damit direkt setzt.

Im zweiten Fall, also direkt nach dem Einschalten, liegt das Datum, das Sie eingeben müssen, in String-Form vor. Um sich das Umrechnen zu sparen, benutzen wir den Settime-Befehl, der beide Uhren mit den neuen Werten versorgt. So, das war es eigentlich. Sie sehen an diesem Beispiel, daß manchmal für ein kleines Programm doch eine Menge Theorie notwendig ist.

13.12.2 Automatische Druckeranpassung statt Accessory

Die Voreinstellungen der Druckeranpassung sind für viele Drucker nicht günstig. Am meisten fällt wohl der Punkt "Punkte je Zeile" aus dem Rahmen, denn bei der Voreinstellung von 1280 Punkten kann man lediglich mit einem DIN-A3-Drucker eine vollständige Hardcopy erzeugen. Des weiteren holen sich viele Textverarbeitungsprogramme ihre Informationen z.B. über Einzel- oder Endlospapier aus den entsprechenden Standardwerten heraus. So war es bisher notwendig, die Druckeranpassung als Accessory parat zu haben, was Speicherplatz (einen der 6 Accessory-Plätze) belegte und mit unserer Auto-Uhr nicht läuft.

Um nun die Werte für das Betriebssystem zugänglich zu machen, kann man zwei Wege gehen. Der erste, der hier vorgestellt wird, liest sich die Informationen aus dem DESKTOP.INF-File heraus. Dazu brauchen Sie lediglich, nachdem Sie die Werte mit Hilfe des Accessory für Ihren Drucker eingestellt haben, den Punkt Arbeit zu sichern im Menü Extras anzuklicken. Neben weiteren Informationen z.B. der angemeldeten Laufwerke, Daten zur RS232-Schnittstelle usw. werden hier auch Daten zum Drucker abgespeichert. Löschen Sie nun einfach das Drucker-Accessory auf Ihrer Boot-Diskette, oder benennen Sie es um. Dann kopieren Sie das folgende Programm in kompilierter Version in den Auto-Ordner.

```

/
/ Programm: AUTODR.BAS
/
/ Das Programm liest die Daten für den Drucker aus dem DESKTOP.INF
/ und setzt entsprechend die Systemvariablen
/
Open "i",#1,"DESKTOP.INF"          ! Datei zum Lesen öffnen
Do
  Exit If Eof(#1)                   ! Ende, wenn alle Zeilen gelesen
  ! wurden
  Line Input #1,Zeile$              ! ganze Zeile lesen
  !
  If Left$(Zeile$,2)="#b"           ! Druckeranpassung
    Wert%=0                         ! Dualzahl in Dezimal wandeln
    For Schleife%=3 To 9            ! 3. bis 9. Zeichen
      Mul Wert%,2                   ! vorherige Zahl mit 2
      ! multiplizieren
      Add Wert%,Val(Mid$(Zeile$,Schleife%,1)) ! neue Zahl addieren
    Next Schleife%                 ! bis alle Ziffern durch sind
    !
    Void Xbios(33,Wert%)            ! Paramter setzen
  Endif
/
Loop

```

Die Werte für die Betriebssystemroutine sind identisch mit denen, die man aus dem DESKTOP.INF lesen kann. Lediglich eine Umrechnung in eine Zahl ist notwendig. Dabei bedeuten die einzelnen Bits der Zwei-Byte-Zahl:

Bitnummer	0	1
0	Matrixdrucker	Typenraddrucker
1	Einfarbdrucker	Farbdrucker
2	Atari-Drucker	Epson-Drucker
3	Test-Modus	Qualitätsmodus
4	Centronics-par.	RS232 seriell
5	Endlospapier	Einzelblatt
6-14	reserviert	
15	immer 0	

Die zweite Möglichkeit der Anpassung besteht darin, dem Betriebssystemaufruf direkt die Werte für Ihren Drucker zu übergeben. Dadurch wird das Programm auf eine Zeile gekürzt, und die Datei DESKTOP.INF muß nicht auf der Boot-Diskette vorhanden sein. Den Wert können Sie sich aus der obigen Beschreibung zusammensetzen. Der Aufruf lautet dann:

```
Void Xbios(33,Wert%)
```


Auch dieses "Programm" müssen Sie zunächst compilieren und dann in den Auto-Ordner kopieren.

13.12.3 GDOS und Accessory installieren

Einige Programme (z.B. die DTP- und Malprogramme) benötigen zum Arbeiten das Programm GDOS, andere wollen damit nicht zusammenarbeiten. Es ist aber sehr umständlich, gerade für Festplattenbesitzer, immer wieder das Programm in den Auto-Ordner zu kopieren bzw. es dort zu löschen. Hier wollen wir nun Abhilfe schaffen. Dabei soll das Programm zwei Fliegen mit einer Klappe schlagen. Die zweite Aufgabe wird sein, während des Boot-Vorgangs zu entscheiden, welche Accessories geladen werden sollen.

Dazu zunächst einige Erklärungen zur Wirkungsweise des Programmes. Wie Sie bereits wissen, werden im Auto-Ordner nur Programme gestartet, die die Endung .PRG haben. Ähnliches gilt auch für die Accessories, die sich unbedingt außerhalb des Auto-Ordners befinden müssen und die Endung .ACC tragen. Hier genau setzt das Programm an. Es fragt nach, ob mit GDOS gearbeitet werden soll und benennt das Programm GDOS.PRG entsprechend um in GDOS.PR, so daß es nicht mehr gestartet wird. Gleiches gilt für die Accessories, die bei Verneinung einfach die Endung .AC oder .ACX erhalten.

Um dieses Umbenennen zu realisieren, muß man noch wissen, daß die Programme im Auto-Ordner in der Reihenfolge abgearbeitet werden, in der sie in den Ordner kopiert worden sind, das heißt, nicht nach alphabetischer oder zeitlicher Reihenfolge. Deshalb ist es sehr wichtig, daß das folgende Programm vor dem Programm GDOS.PRG in den Auto-Ordner kopiert wird. Sollte das GDOS-Programm einen anderen Namen tragen, so nennen Sie dieses um oder ändern im Programm entsprechend den Namen ab.

Das Programm ist in bezug auf Komfort und grafische Darstellung sehr kurz gehalten, da im Auto-Ordner keine grafischen Elemente und auch die File-select-Box nicht erlaubt sind.

```

'
' Programm: GDOSMAKE.BAS
'
' GDOS und ACCESSORIES installieren
'
Print "   Gdos installieren ? (j/n) ";      ! Abfrage
Do
    U$=Upper$(Inkey$)                      ! Taste gedrückt ?
    Exit If U$="J" Or U$="N"                ! verlassen wenn 'j' oder 'n'
Loop
```

```

,
Chdir "\auto\"                ! in Auto-Ordner wechseln
,
If Exist("GDOS.PR")=0 And Exist("GDOS.PR")=0 ! Existiert eines von
    ! beiden
    Print "GDOS.PR' nicht gefunden" ! nein, Fehler ausgeben
    U=Inp(2) ! auf Taste warten
Else
,
    If U$="J" ! GDOS installieren ?
        If Exist("gdos.prg")=0 ! bereits richtig benannt
            Name "gdos.pr" As "gdos.prg" ! nein, umbenennen
        Endif
    Else
        If Exist("gdos.pr")=0 ! bereits richtig benannt
            Name "gdos.prg" As "gdos.pr" ! nein, umbenennen
        Endif
    Endif
,
Endif
,
Chdir "\" ! wieder ins Hauptverzeichnis
,
Dim A_name$(15) ! 15 Accessory-Namen
Dim A_flag$(15) ! 15 Flag für setzen/löschen
,
Cls ! Bildschirm löschen
Print "Accessories installieren"
,
Dta$=Space$(50) ! Buffer für Search
Void Gemdos(&H1A,L:Varptr(Dta$)) ! setzen
Such$="*.AC?" ! Suchname
Vorhanden%=Gemdos(&H4E,L:Varptr(Such$),0) ! 1. File
Lauf%=0
While (Vorhanden%=0 And Lauf%<15) ! max 15 ACC's
    A_name$(Lauf%)=Mid$(Dta$,31,14) ! Namen aus DTA herauslesen
    Print " ";A_name$(Lauf%); " (j/n) "; ! Namen ausgeben
,
    Do
        U$=Upper$(Inkey$) ! auf j oder n warten
        Exit If U$="J" Or U$="N"
    Loop
,
    Print U$ ! gedrückte Taste ausgeben
    If U$="J" ! j ?
        A_flag$(Lauf%)=1 ! flag setzen
    Else ! n ?
        A_flag$(Lauf%)=0 ! flag löschen
    Endif
    Inc Lauf% ! Zähler erhöhen
    Vorhanden%=Gemdos(&H4F) ! nächsten Acc-Namen lesen
Wend
,

```

```

For Lauf1%=0 To Lauf%                ! alle ACC's
  If A_flag%(Lauf1%)=0 And Instr(A_name$(Lauf1%),".ACC")<>0 ! schon
    ! gelöscht?
    Punkt%=Instr(A_name$(Lauf1%),".") ! Punkt suchen
    Neu_name$=Left$(A_name$(Lauf1%),Punkt%)+".ACC" ! neue Endung ACC
    Name A_name$(Lauf1%) As Neu_name$ ! umbenennen
  Endif
  If A_flag%(Lauf1%)=1 And Instr(A_name$(Lauf1%),".ACC")=0 ! schon
    ! gesetzt?
    Punkt%=Instr(A_name$(Lauf1%),".") ! Punkt suchen
    Neu_name$=Left$(A_name$(Lauf1%),Punkt%)+".ACC" ! neue Endung ACC
    Name A_name$(Lauf1%) As Neu_name$ ! umbenennen
  Endif
Next Lauf1%

```

Bevor wir nun zum Luxus-Auto-Starter für GEM-Programme kommen, noch kleine Anregungen. Oft benötigen Sie mehr als eins der vorgestellten Programme, so daß es sich lohnen würde, diese in ein Programm aufzunehmen. Zusätzlich könnten Sie dann auch noch eine Paßwortabfrage einbauen, sofern von der Festplatte gebootet wird.

13.12.4 Ein Autostarter für GEM-Applikationen

Wir haben bereits festgestellt, daß GEM-Programme nicht im Auto-Ordner gestartet werden können. Diese Einschränkung macht es aber unmöglich, auch gleich die Textverarbeitung oder die Dateiverwaltung zu laden, da solche Programme in der Regel GEM-Applikationen sind. Wie schön wäre es doch, wenn man die Diskette mit der Aufschrift Malprogramm einlegen und den Rechner anschalten könnte, um eine Minute später festzustellen, daß man sogleich mit dem Malen anfangen kann...

Mit dem folgenden Programm geht dieser Wunsch in Erfüllung. Zwar ist dieses Utility nicht das einzige seiner Art, doch kennen wir kein zweites, das derart zuverlässig und ausschließlich mit legalen (!) Mitteln einen GEM-Autostart ermöglicht. Was macht denn der Anwender, wenn er eine GEM-Applikation startet? Zerlegen wir den Arbeitsgang doch in vier Teile:

- Er wartet auf den grauen/grünen Schreibtisch.
- Er öffnet ein Fenster mit dem Inhaltsverzeichnis.
- Er zeigt mit der Maus auf das zu startende Programm.
- Er führt einen Doppelklick durch.

Wenn wir noch voraussetzen, daß der Benutzer durch den Menüpunkt "Arbeit sichern" dafür gesorgt hat, daß nach dem Einschalten automatisch ein Fenster

geöffnet wird und das Bild des zu startenden Programms genau in der Bildschirmmitte steht, dann entfallen die Schritte zwei und drei. Wir benötigen also nur noch ein TOS-Programm, das im Auto-Ordner untergebracht werden kann und die Schritte eins und vier durchführt. Das ist gar nicht so schwer, wie Sie gleich sehen werden.

Als erstes binden wir uns in den VBL ein, und zwar nicht in die VBL-Queue, sondern direkt über den zuständigen Vektor (Nr. 28, Adresse: \$70). Dieses unkonventionelle, aber durchaus legitime Verfahren ist dadurch begründet, daß wir den VBL nur für wenige Sekunden benötigen werden. Es bleiben also alle Slots für dauerhafte Programme frei. Diese VBL-Routine tut nun nichts weiter, als den Bildschirm - genauer gesagt: das letzte Byte des Video-RAM - daraufhin zu überprüfen, ob dort schon etwas steht. Wenn ja, kann das nur das Graumuster bzw. das Grün des Desktops sein. Für unseren Doppelklick ist es aber noch zu früh, denn der Rechner braucht etwas Zeit, um das Inhaltsverzeichnis darzustellen. Also wird eine zweite VBL-Routine installiert (die alte wird dabei inaktiv, wir brauchen sie nicht mehr), die 140 VBL - das sind beim Monochrom-Monitor zwei Sekunden - wartet.

Bis hierhin war die Sache kein Problem, aber wie soll man dem Rechner einen Doppelklick vortäuschen? Auch das ist recht einfach. Schauen Sie doch bitte mal unter der Nummer 34 im Verzeichnis der XBIOS-Routinen nach (kbdvbase). Diese Funktion liefert die Adressen einiger nützlicher Systemroutinen, so auch die für die Mausverarbeitung zuständige. Diese Adresse besorgen wir uns und springen sie viermal an: Maustaste gedrückt, losgelassen, gedrückt und losgelassen - ein Doppelklick also. Der Routine wird lediglich in A0 ein Zeiger auf ein Mausdatenpaket übergeben. Das war's denn auch schon, den Rest, sprich: den Start des Programms, überlassen wir gerne dem Betriebssystem.

Soviel zur Funktion des Programms. Was ist also zu tun, damit ein GEM-Programm automatisch gestartet wird?

- Legen Sie, falls noch nicht vorhanden, einen Auto-Ordner auf der Programmdiskette bzw. dem Boot-Laufwerk an.
- In diesen Ordner kopieren Sie das Programm AUTOGEM.PRG.
- Plazieren Sie das Inhaltsverzeichnis so auf dem Bildschirm, daß das gewünschte Programm-Bild (Icon) in der Mitte des Bildschirmes steht (da, wo der Mauszeiger nach dem Einschalten immer steht).
- Wählen Sie den Menüpunkt Arbeit sichern.

Das genügt schon; drücken Sie den Resetknopf, und erfreuen Sie sich des schönen Programms. Nur eins dürfen Sie auf keinen Fall tun: Die Maus in den zwei Sekunden bewegen, bis das Programm gestartet wird. Hier nun das Assembler-Listing:

```

;
; GEM-Programme automatisch starten
; MP 29-04-88
;
gemdos      = 1
xbios       = 14
keep        = $31
supexec     = 38
kbdvbase    = 34
vbl         = $70
v_bas_ad    = $44e
last_byte   = 31999

text
move.l 4(a7),a0 ; Speicherbedarf ausrechnen
move.l #$100,d6
add.l 12(a0),d6
add.l 20(a0),d6
add.l 28(a0),d6
pea init_vbl ; Initialisierung im Supervisor
move.w #supexec,-(sp)
trap #xbios
addq.l #6,sp
clr.w -(sp) ; Beenden, aber resident halten
move.l d6,-(sp)
move.w #keep,-(sp)
trap #gemdos
init_vbl: move.l vbl,old_vbl ; keine VBL-Queue, sondern die
move.l #new_vbl,vbl ; 'harte' Methode...
move.l old_vbl,j_vbl+2 ; direkt in den Sprungbefehl
; schreiben

rts
new_vbl: move.l a0,savereg ; Register müssen gerettet
; werden!

move.l v_bas_ad,a0
tst.b last_byte(a0) ; letztes Byte des Bildschirms
beq weiter ; Null, dann Desktop nicht
; sichtbar
move.l #newest,vbl ; Neuen VBL (Timer) vorschalten
move.w #140,counter ; 2 Sekunden (monochrom)
bra.s weiter

newest: move.l a0,savereg ; Register nochmal retten...
subi.w #1,counter ; Zähler dekrementieren
bne.s weiter
move.l old_vbl,vbl ; VBL normalisieren
move.w #kbdvbase,-(sp) ; Adresse der Mausverarbeitung
; suchen
trap #xbios

```

```

addq.l #2,sp
move.l d0,a0          ; steht in einer Tabelle...
move.l 16(a0),a0
move.l a0,j1+2
move.l a0,j2+2
move.l a0,j3+2
move.l a0,j4+2
lea    pressed,a0     ; Doppelklick vorgaukeln
j1:    jsr    $12345678 ; Dummy-Adressen, werden zur
                        ; Laufzeit eingesetzt
        lea    released,a0
j2:    jsr    $12345678
        lea    pressed,a0
j3:    jsr    $12345678
        lea    released,a0
j4:    jsr    $12345678
weiter: move.l savereg,a0 ; Register zurück
j_vbl: jmp    $12345678
        data
pressed: dc.b    $fa,00,00 ; Ereignis: Maustaste gedrückt
released: dc.b    $f8,00,00 ; Ereignis: Maustaste losgelassen
        bss
        align
old_vbl: ds.l    1
savereg: ds.l    2
counter: ds.w    1
end

```

13.13 GFA-BASIC und Assembler

Wir wollen an zwei Beispielen zeigen, wie eigene Assembler-Routinen in GFA-BASIC benutzt werden können. Die vorgestellten Programme sind jedoch nicht nur in diesem Zusammenhang interessant, sondern in praktisch jeder Programmiersprache - also auch in Assembler selbst - zu gebrauchen. Es handelt sich dabei um eine Methode, um wirklich zufällige Zufallszahlen zu ermitteln, und um einen Timer, etwa für Benchmarks von eigenen Programmen.

Beginnen wir mit den Zufallszahlen. Die Ankündigung, daß diese in unserem Programm wirklich zufällig seien, ist keineswegs doppelt gemoppelt, denn ein Computer ist überhaupt nicht in der Lage, sich rein zufällig eine Zahl ausdenken; er ist und bleibt nun mal eine Maschine, und Maschinen sind immer berechenbar. So verhält es sich denn auch mit den Zufallszahlen, die gewöhnlich nach einer Formel berechnet werden. Diese Formel, die Sie an anderer Stelle in diesem Buch finden, errechnet Zahlen aus einem bestimmten Wertebereich so, daß sich bei mehrfacher Anwendung eine möglichst gleichmäßige statistische Verteilung der Ergebnisse ergibt. Zufällig ist das aber keineswegs.

Wenn wir also wirklich zufällige Zufallszahlen benötigen und der Computer diese - wie oben gezeigt - nicht erstellen kann, dann brauchen wir jemanden, der nicht berechenbar ist, den Anwender nämlich, der vor seinem Computer sitzt. Nun soll eine derartige Aktion natürlich nicht so aussehen, daß der Anwender nach einer zufälligen Zahl gefragt wird, das wäre zu einfach, sondern wir benutzen die Zeit, die der Anwender vertrödelt, wenn der Computer auf ihn wartet.

Der Anwender ist ein träges Glied im Dialog zwischen Mensch und Maschine. Wenn er eine Eingabe tätigen soll, so braucht er dafür Zeit, und zwar jedesmal unterschiedlich viel Zeit. Diese Zeit bringt für uns einen nicht berechenbaren, also rein zufälligen Faktor in das Computersystem. Lassen wir nämlich mit hoher Geschwindigkeit einen Zähler herabzählen, so ist der Zählerstand, nachdem der Rechner z.B. auf einen Tastendruck gewartet hat, undefiniert und damit zufällig! Damit liegt natürlich auch der Nachteil dieses Systems auf der Hand: Werden zwei Zufallszahlen benötigt, ohne daß der Anwender das Programm verzögert, so mag die zweite Zufallszahl zwar im statistischen Sinne noch zufällig sein; sie ist aber berechenbar.

Kommen wir zum Programm: Als erstes wird Ihnen auffallen, daß keine Berechnung des Speicherbedarfs vorgenommen wird und das Programm mit einem simplen RTS endet. Vom Desktop aus könnten wir es also gar nicht aufrufen. Es initialisiert den Timer des MFP (siehe Kapitel residente Programme) und stellt ihn auf ca. 500 Hz ein, d.h. der oben erwähnte Zähler wird 500mal in der Sekunde dekrementiert. Als Bereich des Zählers haben wir Null bis 95 gewählt.

Wichtig für dieses Programm ist, daß der Assembler auf PC-relative Adressierung eingestellt wird, damit es später in jedem beliebigen Speicherbereich ablaufen kann. Zusätzlich sollte das erzeugte Programm ohne Programm-Header abgespeichert werden. Beim Profimat wählen Sie dazu beim Abspeichern die Endung .B, und beim GST-Assembler wird mit der Option -NOLINK assembliert; der Linker wird hier nicht benötigt.

Das so erzeugte File muß nun irgendwie in den BASIC-Interpreter hinein - Sie erinnern sich doch an den Titel dieses Kapitels? Der wohl einfachste Weg ist, die vom Assembler erzeugte Programmdatei durch einen Data-Generator zu jagen, der in diesem Buch schon vorgestellt wurde. Der Data-Generator generiert ein BASIC-Programm, das den Code des Assembler-Programms in Data-Zeilen enthält. Diese Data-Zeilen enthalten aber auch noch eine Zeilennummer und eine Prüfsumme (siehe Beschreibung des Data-Generators), so daß wir die erste und letzte Zahl einer jeden Zeile löschen müssen. Die Zeilen vor den Datas, die dazu bestimmt sind, ein lauffähiges Programm auf die Diskette zu

schreiben, brauchen wir natürlich auch nicht mehr. In unserem Fall bleiben 5 Zeilen à 16 Zahlen übrig, das macht 80 Bytes.

Nun müssen wir diese 80 Bytes im Speicher ablaufen lassen. Dazu sind sie zunächst einmal in eine computergerechte Form zu bringen, d.h. wir benötigen einen 80 Bytes großen Speicherbereich, in dem diese Zahlen byteweise abgespeichert werden können. Hier bietet sich ein String an. Wie die Bytes im einzelnen dort hinein kommen, schauen Sie sich am besten im Programm-Listing an. Der Aufruf gestaltet sich nun sehr einfach: Der Befehl "Call", versehen mit der Startadresse des Programms, die gleich der Adresse des ersten Zeichens unseres Strings ist, setzt die ganze Maschinerie in Gang. Beachten Sie bitte, daß am Ende des Programms der Timer unbedingt wieder abgeschaltet werden muß. (Wir verweisen nochmals auf das Kapitel Residente Programme, der MFP.). Täten wir das nicht, so würde beim nächsten Programmstart die Variable "Random\$" gelöscht, und das macht sich als Maschinenprogramm nicht sonderlich gut...

So bleibt noch zu klären, wie wir an die aktuelle Zufallszahl herankommen. Dazu empfiehlt es sich, ein Assembler-Listing mit den Adressen jedes Maschinenbefehls ausgeben zu lassen. Diesem Listing können wir z.B. entnehmen, daß die Variable Zahl aus unserem Assembler-Programm \$44 Bytes nach der Startadresse des Programms liegt. Da wir aber die Startadresse kennen, müssen wir nur noch die \$44 dazu addieren und können durch ein Lpeek (Basis+&h44) die Zufallszahl auslesen. Alles weitere ist den Listings zu entnehmen:

```

;
; zufällige Zufallszahlen          RANDOM.Q
; PC-relativ assemblieren
;      MP      02-05-88
;
xbios      = 14
xbtimer    = 31
text
lea        zahl,a0
move.l     startwert,(a0)    ; Randomize
pea        random            ; Timer A initialisieren lassen
move.w     #49,-(sp)         ; Datenregister (49mal
                             ; dekrement)
move.w     #6,-(sp)          ; Vorteiler teilt durch 100
clr.w      -(sp)             ; Timer A
move.w     #xbtimer,-(sp)
trap       #xbios
add.l      #12,sp
rts
random:    move.l     a0,-(sp)    ; A0 retten
           lea        zahl,a0

```



```

        subi.l #1,(a0)          ; Zufallszahl 500mal in der
                                ; Sekunde
        bpl.s ok                ; erniedrigen
        move.l startwert,(a0)   ; Abgelaufen? Dann neu anfangen
ok:      move.l (sp)+,a0
        bclr #5,$fffffa0f      ; Etwas Verwaltungsarbeit...
        rte                    ; und abtreten
        data
startwert: dc.l 95              ; Zahlen von 0 bis ...
                                ; (einschließlich)
        bss
zahl:    ds.l 1
        end

```

```

' Demoprogramm für RANDOM                      RND_DEMO.BAS
' MP 19-05-88
'
Random$=Space$(80)
Basis=Varptr(Random$)
'
For I%=0 To 79
    Read X%
    Poke Basis+I%,X%
Next I%
'
Call Basis
'
Data 65,250,0,66,32,186,0,58,72,122,0,26,63,60,0,49
Data 63,60,0,6,66,103,63,60,0,31,78,78,223,252,0,0
Data 0,12,78,117,47,8,65,250,0,28,4,144,0,0,0,1
Data 106,4,32,186,0,12,32,95,8,184,0,5,250,15,78,115
Data 0,0,0,95,0,0,0,0,0,0,0,0,0,0,0,0
'
Print "Zufallszahlengenerator: Drücken Sie eine Taste. (E=Ende) "
Print
'
Do
    A%=Inp(2)
    Exit If Upper$(Chr$(A%))="E"
    Print "Neue Zufallszahl ist ";Lpeek(Basis+&H44)
Loop
'
Void Xbios(31,0,0,0,L:0)      ! Timer ausschalten
End

```

Sehr komfortabel ist dieses Verfahren sicherlich nicht, auch wenn es sich für sehr kurze Assembler-Routinen anbietet. Werden die Programme jedoch größer, so ist der Aufwand mit allen Datas, Startadressen usw. nicht mehr zu vertreten. Es gibt deshalb eine weitere Möglichkeit, Assembler-Programme von GFA-BASIC aus aufzurufen: der Monitor-Befehl.

Monitor kann einen Parameter (in Klammern) erhalten, der vor Aufruf der Routine automatisch ins Register D0 kopiert wird (Langwort). GFA-BASIC erzeugt dann bewußt einen Fehler durch einen ungültigen Maschinenbefehl, der gewöhnlich mit 4 Bomben bestraft wird. Wir benötigen also ein residentes Programm, das den Illegal-Instruction-Vektor (Nummer 4) auf die gewünschte Routine lenkt. Dieses Programm wird gestartet, bevor GFA-BASIC geladen wird. Trifft der BASIC-Interpreter auf einen Monitor-Befehl, so wird das zuvor geladene Maschinenprogramm ausgeführt. Das umständliche In-den-Speicher-Poken entfällt; das BASIC-Programm sieht sauberer aus. Außerdem bietet es die wertvolle Möglichkeit, einen Parameter zu übergeben.

Als Beispiel haben wir schon einen Timer angekündigt. Als Zeiteinheit wählen wir eine 200stel Sekunde, da dies eine der wenigen glatten Werte ist, die der Timer des MFP erzeugen kann. Das anzuspringende Programm beginnt beim Label "new_illegal". Hier wird erst einmal der übergebene Parameter geprüft. Wir beabsichtigen, folgende Funktionen zu implementieren:

Parameter	Funktion
0	Timer auf Null stellen
1	Timer (wieder) anstellen
2	Timer anhalten
>2	Aktuelle Zeit in 200stel Sekunden wird in der Adresse, die der Parameter darstellt, abgespeichert

Wie das realisiert wird, ist im folgenden Listing zu sehen:

```

;
; Zeitmessung für Benchmarks u.Ä.                TIME.Q
;      MP      02-05-88
;
gemdos      = 1
bios        = 13
xbios       = 14
xbtimer     = 31
setexception = 5
keep        = $31
text
move.l      4(a7),a0      ; Speicherbedarf berechnen
move.l      #$100,d6
add.l       12(a0),d6
add.l       20(a0),d6
add.l       28(a0),d6
pea         timer        ; Timer A initialisieren lassen
move.w      #192,-(sp)    ; Datenregister 192 und
move.w      #5,-(sp)     ; Vorteiler teilt durch
```

```

                                ; 64 => 200 Hz
                                ; Timer A
clr.w    -(sp)
move.w   #xbtimer,-(sp)
trap     #xbios
add.l    #12,sp
pea      new_illegal          ; Illegal-Instruction Vektor
move.w   #4,-(sp)             ; auf eigene Routine
move.w   #setexception,-(sp)
trap     #bios
addq.l   #8,sp
clr.w    -(sp)
move.l   d6,-(sp)             ; Abbruch, aber resident halten
move.w   #keep,-(sp)
trap     #gemdos
timer:   tst.w    flag         ; Timer angeschaltet?
        beq.s     ok           ; nein, dann Ende
        addi.l    #1,zeit      ; sonst Uhr weiterzählen
ok:       bclr    #5,$ffffa0f  ; abmelden beim MFP
        rte       ; und Ende
new_illegal: addq.l #2,2(sp)    ; 2(sp) ist geretteter Programm-
                                ; Zähler; wir setzen ihn auf den
                                ; Befehl hinter ILLEGAL
                                ; Parameter größer als 2?
cmpi.l   #2,d0               ; Ja, dann als Adresse gemeint
bgt.s     read_timer         ; Opcode * 4 = Offset
lsl.w     #2,d0
lea       tabelle,a0
move.l    (a0,d0.w),a0        ; Adresse der Unterroutine
jmp       (a0)
read_timer: move.l d0,a0      ; Uhr auslesen nach (d0)
        move.l    zeit,(a0)
        rte
                                ; RTE springt zurück ins
                                ; GFA-BASIC
                                ; hinter den ILLEGAL-Befehl
                                ; (s.o.)
reset_timer: clr.l    zeit    ; Timer auf 0 stellen
        rte
start_timer: move.w   #1,flag ; Timer aktivieren
        rte
stop_timer:  clr.w    flag    ; Timer abstellen
        rte
        data
zeit:       dc.l      0
flag:       dc.w      0       ; 0 = Timer ausgeschaltet
tabelle:    dc.l      reset_timer ; Startadressen der Routinen
        dc.l      start_timer
        dc.l      stop_timer
end

```

Dazu ein BASIC-Programm, das die Anwendung verdeutlicht. Es soll zeigen, welche Zeiten für eine leere For-Schleife (10000 Durchläufe) benötigt werden, wenn die Laufvariable vom Typ Integer bzw. vom Typ Real ist:

```
'
' Demoprogramm für Zeitmessung          TIMEDEMO.BAS
'   MP   19-05-88
'
Monitor (0)                      ! Timer auf Null setzen
Monitor (1)                      ! Timer anschalten
'
For I%=1 To 10000
Next I%
'
Monitor (2)
Monitor (Varptr(A%))             ! Ergebnis nach A%
'
Print "Zeit für INTEGER-Laufvariable: ";A%/200;" Sekunden"
'
'
Monitor (0)
Monitor (1)
'
For I=1 To 10000
Next I
'
Monitor (2)
Monitor (Varptr(A%))
'
Print "Zeit für REAL-Laufvariable: ";A%/200;" Sekunden"
End
```


14. Massenspeicher

Massenspeicher wie Disketten oder Festplatten sind für die Arbeit mit dem Computer unverzichtbar - ganz klar. Aber da steckt dann doch noch ein wenig mehr dahinter. Es klingt ja so einfach, wenn man sagt, daß eine Datei auf einer Diskette angelegt wird.

Erst einmal muß die Verteilung auf der Diskette so erfolgen, daß man die Daten jederzeit wiederfinden kann. Dazu sind einige Vorbereitungen erforderlich, um die sich der Computerbenutzer zwar kaum kümmern muß, die aber das Betriebssystem und die Hardware des Rechners bzw. der Diskettenstationen und Festplatte übernehmen muß.

Die Diskette muß vor der Benutzung erst einmal formatiert werden. Dabei wird die Fläche der Disketten in einzelne Sektoren unterteilt, deren Position durch das verwendete Format festgelegt werden.

Dieses Format muß nun auch für den Rechner feststellbar sein, da er ja mit verschiedenen Formaten arbeiten können muß. Dabei sind die Anzahl der verwendeten Seiten der Diskette ebenso wichtig wie die Anzahl der Sektoren und deren Länge. Diese Informationen enthält der sogenannte Boot-Sektor, den wir gleich unter die Lupe nehmen werden.

Danach müssen für jede Datei bzw. Programm, welches auf der Diskette gespeichert werden soll, die verwendeten Sektoren zugewiesen und markiert werden. Diese Informationen sind im Inhaltsverzeichnis der Diskette und im sogenannten FAT, der "File-Allocation-Table", verborgen. Diese werden ebenfalls in diesem Kapitel besprochen.

Sie sehen schon: Auch auf einer Diskette gibt es noch viel zu entdecken.

14.1 Das Diskettenformat

Beim Formatieren einer Diskette (oder auch einer Festplattenpartition) wird diese, wie bereits erwähnt, in einzelne Abschnitte aufgeteilt. Die grobe Unterteilung ist die Spureinteilung. Diese Spuren liegen wie konzentrische Ringe auf der Scheibe und werden von außen nach innen gezählt. Auf einer normal formatierten Diskette befinden sich 80 solche Spuren, auch Tracks genannt, die von 0 bis 79 nummeriert sind. Es ist zwar möglich, bis zu 82 Tracks zu formatieren, jedoch nimmt die Datensicherheit zur Mitte hin wegen der immer gerin-

geren verfügbaren Fläche dermaßen ab, daß die Tracks 80 bis 82 nicht mehr verwendet werden. Dennoch könnten sie benutzt werden, wenn man entsprechend formatiert.

Die einzelnen Tracks werden nun ihrerseits in Sektoren unterteilt, die je einen Abschnitt des Ringes darstellen. Diese Sektoren werden wiederum zu sogenannten Clustern zusammengefaßt, üblicherweise 2 Sektoren pro Cluster. Die Bedeutung dieser Cluster ist jedoch nicht so wichtig, so daß wir nur die Sektoren betrachten.

Im normalen Format befinden sich 9 Sektoren auf jedem Track, die 512 Bytes fassen. Somit ergibt sich bei einer einseitigen Diskette eine Speicherkapazität von $80 \times 9 \times 512 = 368640$ Bytes.

Dies ist allerdings nicht die wirkliche Anzahl der auf der Diskette gespeicherten Daten. Pro Track und nochmals pro Sektor werden beim Formatieren einige zusätzliche Daten geschrieben.

Diese Daten werden vom Disk-Controller, dem die Floppy steuernden Chip im ST, benötigt, um aus dem Track den richtigen Sektor herauszufinden. Betrachten wir nun den vollständigen Aufbau eines normalen Tracks.

Anzahl	Bytes	Bemerkungen
60 pro Sektor:	\$4E	Track-Anfang
12	\$00	
3	\$F5	werden als \$A1 geschrieben
1	\$FE	ID Adress Mark
1	Track#	Tracknummer 0 bis 79
1	Seiten#	Seitennummer 0 oder 1
1	Sektor#	Sektornummer 1 bis 9
1	\$02	*\$100=512 Bytes pro Sektor
1	\$F7	CRC-Prüfsumme schreiben (werden 2 Bytes)
22	\$4E	Füllbytes
12	\$00 "	
3	\$F5	werden zu \$A1
1	\$FB	Markierung (Data-Adress-Mark)
512	Daten	hier liegen die eigentlichen Daten
1	\$F7	CRC-Prüfsumme schreiben
40	\$4E	Füllbytes
usw...		
Trackende:		
1401	\$4E	Füllbytes

Zählt man all diese Bytes zusammen, so kommt man auf 6969 Bytes pro Track, was einer (unformatierten) Diskettenkapazität von 557520 Bytes entspricht (bei einseitiger Nutzung). Leider kann man diese Kapazität nicht für seine Daten nutzen, da der Controller sie dann nicht mehr finden könnte (wie könnte er Anfang und Ende eines Sektors erkennen?). Möglich ist es jedoch, die letzten 1401 Bytes jedes Tracks für einen zusätzlichen Sektor zu benutzen. Das würde die benutzbare Kapazität auf 409600 Bytes erhöhen. Nimmt man zusätzlich noch drei weitere Tracks (80 bis 82) dazu, kommt man sogar auf 424960 Bytes. Aber wie gesagt, die Datensicherheit ist nicht mehr so besonders, wenn auch vertretbar.

Um ein solches eigenes Format zu erstellen, ist ein kleines Programm nötig. Bevor wir uns ein solches Programm ansehen, müssen wir jedoch die einzelnen Schritte, in denen das Formatieren abläuft, genauer ansehen. Mit dem bloßen Formatieren der Tracks ist es nämlich nicht getan. Die verwendeten Parameter wie Anzahl der Tracks und Sektoren müssen ebenfalls auf die Diskette geschrieben werden, da der Rechner sonst nicht feststellen kann, wie die Diskette formatiert ist. Dazu dient der Boot-Sektor.

14.2 Der Boot-Sektor

Der Boot-Sektor liegt immer ganz am Anfang einer Diskette bzw. Festplatte, d.h. auf Track 0, Seite 0, Sektor 1 einer Diskette oder Sektor 0 der Harddisk. Er ist, wie alle anderen Sektoren auch, 512 Bytes lang und wird nach jedem Diskettenwechsel vom Betriebssystem überprüft. Außerdem ist er für das "Booten" der Diskette entscheidend. Booten bedeutet hier das Laden des Betriebssystems von Diskette nach dem Einschalten. Dabei wird zuerst von der Diskette in Laufwerk A der Boot-Sektor geladen und geprüft, ob die Diskette ein Betriebssystem enthält. Außerdem enthält der Boot-Sektor noch weitere Informationen. Sollte in Laufwerk A: kein Betriebssystem zu finden sein, macht der ST das gleiche Spielchen mit dem Boot-Sektor der Harddisk (falls eine angeschlossen ist).

Insgesamt enthält der Boot-Sektor eine Seriennummer der Diskette, einen Parameterblock für das BIOS des Rechners und evtl. ein Boot-Programm mit Bootparametern. Sollte ein solches Programm enthalten sein, so muß die Summe aller enthaltenen Bytes des Sektors (Checksumme) die "magische" Zahl \$1234 ergeben. Stimmt diese Zahl, so wird das Programm ab dem Anfang des Sektors ausgeführt, wo dann üblicherweise ein BRA (Branch always)-Befehl steht. Das Programm muß dabei so gestaltet sein, daß es an jeder beliebigen Speicherstelle laufen kann, da der Sektor ja irgendwohin geladen wurde.

Solch ein Boot-Programm ist normalerweise nicht im Boot-Sektor enthalten. Wichtiger sind die verschiedenen Parameter, die sich ebenfalls auf dem Sektor befinden. Diese Parameter werden bei einem "Get BPB"-Aufruf des Betriebssystems geladen und in den sogenannten BPB (BIOS-Parameter-Block) geladen. Sind diese Parameter unglaublich, so wird aus der "Get BPB"-Funktion nicht die Adresse des BPB übergeben, sondern eine 0.

Die weitere Information im Boot-Sektor ist die Seriennummer der Diskette. Es handelt sich dabei um eine 24-Bit-Zahl, die beim Formatieren ermittelt und auf die Diskette geschrieben wurde. Diese Nummer dient zur Erkennung eines Diskettenwechsels.

Hier nun der gesamte Aufbau des Boot-Sektors:

Byte Nr.	Name	Bedeutung (normal 1/2-seitig)
\$00	BRA	Sprungbefehl in das Boot-Programm (evtl.)
\$02	Füller	reservierte Füllbytes oder "Loader"
\$08	Serien#	Seriennummer
* \$0B	BPS	Anzahl Bytes pro Sektor (512)
* \$0D	SPC	" Sektoren pro Cluster (2)
* \$0E	RES	" reservierte Sektoren (1)
* \$10	NFATS	" FATs (File Allocation Tables) (2)
* \$11	NDIRS	" mögliche Directory-Einträge (112)
* \$13	NSECTS	" Sektoren auf Diskette (720/1440)
* \$15	MEDIA	Medium-Beschreibung (unbenutzt)
* \$16	SPF	Anzahl Sektoren der FAT (5)
* \$18	SPT	" Sektoren pro Track (9)
* \$1A	NSIDES	" Seiten der Diskette (1/2)
* \$1C	NHID	" versteckte Sektoren (0)
\$1E	EXECFLG	Flag für COMMAND.PRg
\$20	LDMODE	Flag für File- oder Sektorboot
\$22	SSECT	erster zu ladender Sektor
\$24	SECTCNT	Anzahl der zu ladenden Sektoren
\$26	LDADDR	Ladeadresse
\$2A	FATBUF	FAT-Adresse
\$2E	FNAME	File-Name (meist TOS.IMG)
\$39	RES	reserviert
\$3A	BOOTIT	Boot-Programm
-\$1FD		
\$1FE		Ausgleichswort für die Checksumme

Die mit einem Sternchen (*) gekennzeichneten Einträge entsprechen dem BPB der Diskette. Diese Tabelle ist fast identisch mit derjenigen des MS-DOS, des

Betriebssystems der IBM-PCs. Dadurch ist auch zu beachten, daß ein 16-Bit-Wort hier in der Bytes-Reihenfolge low-high vorliegt (z.B. BPS= \$00 \$02 ergibt \$200 Bytes pro Sektor). Somit ist der ATARI ST in der Lage, IBM-Disketten zu lesen.

Hier noch einige Anmerkungen zu den Einträgen im Boot-Sektor:

Die Zahlen in Klammern, die hinter einigen Einträgen stehen, stellen den üblichen Inhalt dieser Einträge bei einer einseitig formatierten Diskette dar.

NHID, die Anzahl der versteckten Sektoren, wird vom BIOS des ATARI ST bei Disketten nicht verwendet.

Die ab \$1E liegenden Daten sind nur interessant, wenn es sich bei der Diskette um eine boot-fähige Diskette handelt. Eine solche Diskette enthält normalerweise das Betriebssystem in Form eines Daten-Files, Imagefile (.IMG) genannt. Man erkennt einen ausführbaren Boot-Sektor auch daran, daß in ihm ab dem dritten Byte der Text "Loader" steht. Das Boot-Programm, welches in den älteren ATARI STs in den zwei ROMs steckt, erkennt einen solchen Boot-Sektor außerdem an der Prüfsumme, welche für einen ausführbaren Boot-Sektor \$1234 betragen muß. Liegt also ein solcher Fall vor, bekommen die weiteren Daten im Boot-Sektor folgende Bedeutungen:

EXECFLG	wird in die Systemvariable "cmdload" kopiert. Dieses Flag entscheidet, ob nach dem Laden des Betriebssystems das Programm COMMAND.PRG geladen werden soll oder nicht.
LDMODE	bestimmt den Lademodus. Ist dieses Flag Null, wird das mit FNAME benannte File gesucht und geladen. Dieses File ist üblicherweise TOS.IMG. Ist LDMODE ungleich null, so werden Sektoren in Abhängigkeit von SECTCNT und SSECT direkt geladen.
SSECT	ist der logische Sektor, ab dem gebootet wird. Diese Variable ist nur gültig, wenn LDMODE ungleich null ist.
SECTCNT	gibt die Anzahl der zu bootenden Sektoren an. Dies ist ebenfalls nur bei LDMODE ungleich null gültig.
LDADDR	ist die Speicheradresse, ab der das File oder die Sektoren geladen werden.
FATBUF	gibt die Adresse an, an die die FAT- und die Directory-Sektoren geladen werden sollen.

- FNAME** ist der File-Name des "Image-Files", welches geladen werden soll (LDMODE = 0). Es ist genauso aufgebaut wie ein normaler File-Name, also acht Zeichen als Name und drei Zeichen als Extender.
- BOOTIT** ist ein eventuelles Boot-Programm, welches nach dem Laden des Boot-Sektors ausgeführt wird.

So ist also der Boot-Sektor aufgebaut. Zusammen mit den Kenntnissen des Diskettenformates ist nun genug Theorie vorhanden, um einen Schritt in die Praxis zu tun. Wir wollen diesen Schritt mit einem Programm vornehmen, mit dem wir unsere Disketten formatieren können.

Mit Hilfe des "File"-Menüs des Desktop sind wir auch ohne spezielles Programm in der Lage, Disketten zu formatieren. Wie jedoch bereits anfangs erwähnt, ist das vom ATARI-Betriebssystem TOS verwendete Format auf 80 Tracks und neun Sektoren pro Tracks festgelegt. Physikalisch passen aber wesentlich mehr Tracks und Sektoren auf eine Diskette.

14.2.1 Ein Formatierungsprogramm

Das nun folgende Programm bietet einige Möglichkeiten, die Kapazität einer normalen Diskette zu erhöhen. Es wird über ein kleines Menü gesteuert, in dem man einige Parameter für die Formatierung einstellen kann. Die Auswahl der gewünschten Einstellungen erfolgt dabei über die Funktionstasten.

Das Menü, welches nach dem Aufruf des Programmes erscheint, sieht folgendermaßen aus:

```
*** Formatierungs-Programm S.D. ***
[F1] Seite(n) .....: 2
[F2] Tracks .....: 80
[F3] Sektoren/Track ...: 9
[F4] Laufwerk .....: A
[F8] Formatieren ...
[F10] Quit !
```

Durch Druck einer der angegebenen Funktionstasten wird nun entweder eine Einstellung verändert oder eine Funktion ausgelöst. Folgende Einstellungen sind möglich:

- F1:** Mit dieser Taste wird zwischen ein und zwei Seiten umgeschaltet. Bei Verwendung eines einseitigen Laufwerks sollte auch nur eine Seite eingestellt werden.

- F2: Hier wird zwischen 80 (Normaleinstellung) und 82 Tracks ausgewählt. Die Verwendung von 83 Tracks wäre auch möglich, aus Gründen der zu geringen Datensicherheit habe ich jedoch auf diese Auswahlmöglichkeit verzichtet. Sie können natürlich dennoch durch geringfügige Änderungen im Programm mit 83 Tracks arbeiten. Aber Achtung: Einige der von Atari verwendeten Floppy-Laufwerke sind physikalisch nicht in der Lage, die 83. Spur anzufahren. Versucht man es dennoch, fährt der Kopf gegen einen Anschlag. Passiert das häufiger, riskiert man eine Dejustierung, die nur vom Profi mit Spezialwerkzeug rückgängig gemacht werden kann.
- F3: Mit der Funktionstaste F3 wird zwischen neun und zehn Sektoren pro Track hin- und hergeschaltet.
- F4: Hiermit können Sie zwischen Laufwerk A und B auswählen. Dieser Punkt ist vor dem Starten der Formatierung besonders zu kontrollieren, damit nicht versehentlich Ihre im anderen Laufwerk steckende Systemdiskette formatiert wird...
- F8: Diese Taste löst die Formatierung selbst aus. Dieser Vorgang beginnt unmittelbar auf Tastendruck und wird durch die Meldung "Formatierung läuft. Bitte warten..." angezeigt. Sollte ein Fehler auftauchen, erscheint die Meldung "*** Es ist ein Fehler aufgetreten !! ***". Sie sollten daraufhin die Diskette überprüfen, ob nicht z.B. der Schreibschutz zurückgeschoben ist. Die Fehlermeldung bleibt so lange auf dem Bildschirm, bis irgendeine Taste gedrückt wird. Somit braucht der Formatierungs-Vorgang nicht ständig beobachtet zu werden.
- F10: Wenn Sie alle Disketten formatiert haben, können Sie durch die Betätigung der F10-Taste das Programm verlassen.

Durch die flexible Auswahlmöglichkeit dieses Programmes sind verschiedene Kapazitäten der formatierten Disketten einstellbar. Hier einige Werte für einseitige Formate:

Tracks	Sektoren pro Track	Kapazität in Bytes
80	9 (normal)	357376
82	9	366592
80	10	398336
82	10	408576

Wie Sie an obiger Tabelle erkennen können, ist ein Gewinn an Kapazität schon bei einseitigen Disketten bis zu 51200 Bytes möglich. Bei doppelseitigen Disketten verdoppelt sich diese Ausbeute noch, so daß über 100 KByte dazu kommt.

Hier nun das Programm. Erstellt wurde es mit dem Programm SEKA, welches geringe Abweichungen zum DRI-Assembler zeigt. Sollten Sie das Programm mit dem DRI-Assembler übersetzen wollen, so müssen lediglich die Kommentarzeilen mit einem Sternchen (*) beginnen und die "blk.b"-Anweisung in "ds.b" geändert werden.

```

    ;** Formatierungs-Programm S.D. **
run:
    move.l #menue,d0
    bsr    print           ;Menü ausgeben
    bsr    getkey
    cmp.b  #$3b,d0
    blt    run             ;falsche Taste
    cmp.b  #$44,d0
    bgt    run             ;falsche Taste
    cmp.b  #$3b,d0         ;F1 ?
    bne    notf1
    eor    #3,sds          ;1/2 Seiten
    eor    #1,sdsf
    bra    run
notf1:
    cmp.b  #$3c,d0         ;F2 ?
    bne    notf2
    eor    #2,trs          ;80/82 Tracks
    eor    #2,trsf
    bra    run
notf2:
    cmp.b  #$3d,d0         ;F3 ?
    bne    notf3
    eor    #3,sptf
    eor    #$1109,spt      ;9/10 Sektoren pro Track
    bra    run
notf3:
    cmp.b  #$3e,d0         ;F4 ?
    bne    notf4
    eor    #3,lw
    eor    #1,lwf          ;Laufwerk A/B
    bra    run
notf4:
    cmp.b  #$42,d0         ;F8 ?
    bne    notf8
    bsr    format          ;=> Formatierung
    bra    run
notf8:
    cmp.b  #$44,d0         ;F10 ?
    bne    run

```

```

        clr      -(sp)
        trap     #1          ;Quit, zurück zum Desktop
format:      ;* Formatieren *
        move.l   #wait,d0
        bsr      print      ;"Formatierung läuft.."
        move     trsf,trsf1
        subq     #1,trsf1
floop:
        move     sdsf,seite  ;Seite bestimmen
floop1:
        bsr      fmttr      ;formatiere einen Track
        bne      error
        subq     #1,seite    ;ggf. noch andere Seite
        bpl      floop1     ;formatieren
        subq     #1,trsf1
        bpl      floop      ;nächster Track
setboot:     ;Boot-Sektor erstellen
        clr      -(sp)      ;Execute-Flag: nicht ausführbar
        moveq    #2,d0
        or       sdsf,d0
        move     d0,-(sp)    ;Disktyp- und Seitenauswahl
        move.l   #$1000000,-(sp) ;Seriennr. erstellen
        pea      puffer     ;Puffer-Adresse
        move     #$12,-(sp)
        trap     #14        ;Boot-Sektor erstellen
        add.l    #14,sp
        lea      puffer,a0   ;Zeiger auf Boot-Sektor-Puffer
        clr.l    d0
        cmp      #9,sptf     ;9 Sektoren pro Track ?
        beq      sok        ;ja
        move.b   #10,24(a0,d0) ;sonst 10 SPT einsetzen
        move     trsf,d1     ;Anzahl der Tracks in D1
        tst      sdsf       ;1 Seite ?
        beq      sd11       ;ja
        lsl      #1,d1      ;sonst doppelter Zuwachs
sd11:
        bsr      addsec     ;SEC + Anzahl der Tracks (D1)
sok:
        cmp      #80,trsf    ;80 Tracks ?
        beq      trok       ;ja
        move     #18,d1
        tst      sdsf       ;1 Seite ?
        beq      sd12       ;ja
        lsl      #1,d1      ;sonst doppelter Zuwachs
sd12:
        bsr      addsec     ;SEC + 2*9 oder 4*9
trok:
        move     #1,-(sp)    ;1 Sektor
        clr.l    -(sp)      ;Seite 0, Track 0
        move     #1,-(sp)    ;Sektor 1
        move     lwf,-(sp)   ;Laufwerk
        clr.l    -(sp)
        pea      puffer     ;Puffer

```

```

move    #9,-(sp)
trap    #14                ;flopwr, Boot-Sektor schreiben
add.l   #20,sp
tst     d0                ;Fehler aufgetreten ?
bne     error             ;ja: Fehlermeldung
bra     run               ;Neustart
addsec:                ;SEC = SEC + D1
move.b  20(a0,d0),d2      ;HI
lsl     #8,d2
move.b  19(a0,d0),d2      ;LO
add     d1,d2
move.b  d2,19(a0,d0)      ;set LO
lsr     #8,d2
move.b  d2,20(a0,d0)      ;set HI
rts

error:
move.l  #errtxt,d0
bsr     print             ;Fehlermeldung ausgeben
bsr     getkey            ;auf Taste warten
bra     run               ;und Neustart
fmttr:                ;einen Track formatieren
clr     -(sp)             ;Virgin-Daten
move.l  #$87654321,-(sp)  ;Magic-Zahl
move    #1,-(sp)          ;interleave
move    seite,-(sp)        ;Seite
move    trsf1,-(sp)        ;Track
move    sptf,-(sp)         ;Sektoren/Track
move    lwf,-(sp)          ;Laufwerk
clr.l   -(sp)
pea     puffer             ;Track-Puffer
move    #10,-(sp)
trap    #14               ;flopfmt, Track formatieren
add.l   #26,sp
tst     d0                ;Test auf Error
rts

print:                ;Text ab (D0) ausgeben
move.l  d0,-(sp)
move    #9,-(sp)
trap    #1
addq.l  #6,sp
rts

getkey:                ;auf Tastendruck warten,
move.w  #1,-(sp)
trap    #1
addq.l  #2,sp
swap    d0                ;Tastencode in D0.b
rts

; Texte und Variablen:
menue:  dc.b  $1b,"E*** Formatierungs-Programm S.D. ***"
        dc.b  10,13,10,13
        dc.b  " [F1] Seite(n) .....: "
sds:    dc.b  " 2",10,13
        dc.b  " [F2] Tracks .....: "

```

```
trs:  dc.b  "80",10,13
      dc.b  " [F3] Sektoren/Track ...: "
spt:  dc.b  " 9",10,13
      dc.b  " [F4] Laufwerk .....: "
lw:   dc.b  " A",10,13
      dc.b  " [F8] Formatieren ...",10,13
      dc.b  "[F10] Quit !",10,13,10,13,0
wait: dc.b  "Formatierung läuft. Bitte warten...",10,13,0
errtxt: dc.b  "*** Es ist ein Fehler aufgetreten !! ***",10,13,0
even
sdsf:  dc.w 1
trsf:  dc.w 80
trsf1: dc.w 80
sptf:  dc.w 9
lwf:   dc.w 0
seite: dc.w 0
data
puffer: blk.b 8000
```

Das Programm ist in folgende Abschnitte eingeteilt:

1. Menüsteuerung: Das Menü wird ausgegeben (es löscht dabei den Bildschirm) und auf einen Tastendruck gewartet. Nach einer Eingabe wird der Tastencode, welcher in D0 übergeben wurde, ausgewertet. Trifft einer der CMP.B #\$xx,D0-Vergleiche zu, so wird die gewählte Funktion ausgelöst. Bei einer Umschalt-Funktion (F1-F4) wird durch den EOR-Befehl diese Umschaltung sowohl im Menütext als auch in der entsprechenden Parameterzelle vorgenommen. Nach einer erfolgten Umschaltung wird wieder zum Start (run) gesprungen, außer bei der Taste F10, die mittels der TERM-Funktion des GEMDOS das Programm beendet.
2. Formatierung: Nach Ausgabe der Meldung "Formatierung läuft..." wird die Diskette ab dem eingestellten maximalen Track-1 bis Track 0 formatiert. Ist zweiseitige Formatierung gewählt, so wird jeder Track erst auf Seite 1 (Rückseite) und dann auf Seite 0 formatiert.
3. Erstellung des Boot-Sektors: Zuerst wird ein normler Boot-Sektor über die "protobt"-Funktion des XBIOS erzeugt. Dabei wird lediglich die Anzahl der Seiten berücksichtigt.
4. Korrektur des Boot-Sektors: Sind von der Norm abweichende Einstellungen vorhanden (zehn Sektoren pro Track, 82 Tracks), so wird der erstellte Boot-Sektor entsprechend korrigiert. Dazu wird zunächst die Anzahl der Sektoren pro Track getestet. Ist sie zehn, so wird dies zuerst in die SPT-Zelle des Boot-Sektors eingesetzt und dann die Anzahl der Tracks zur Anzahl der Sektoren auf der Diskette addiert. Danach wird die gewählte

Track-Anzahl getestet und bei Bedarf die zusätzliche Sektorenanzahl aufaddiert.

5. Abspeichern des Boot-Sektors: Mit Hilfe der "flopwr"-Funktion des XBIOS wird der aufbereitete neue Boot-Sektor auf Seite 0, Track 0, Sektor 1 geschrieben. Sollte ein Fehler auftreten, so wird dieser angezeigt.
6. Datenbereich: Hier sind die Texte des Menüs bzw. der Meldungen und die Variablen untergebracht. Der Puffer wird zwar in der Länge eingestellt, jedoch nicht auf die Diskette geschrieben, da er im ".bss"-Bereich liegt. Bei Verwendung des DRI-Assemblers muß daher hier statt "data" ein ".bss" eingesetzt werden.

Hier nun ein ST-BASIC-Programm, welches das Formatierungs-Programm mit dem Namen "Bigformat.prg" auf der Diskette erstellt:

```

10 **** Erstellung des BIGFORMAT-Programms ***
15 '
20 '
25 print "File >> bigformat.prg << wird erzeugt":print:print:print
30 dim c%( 441):cs#=0
35 for i=0 to 441
40 read a$:c%(i)=val("&H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 4376703 then 70
60 print"Geht leider noch nicht, da etwas mit den DATA's nicht stimmt."
65 goto 80
70 bsave "bigformat.prg",varptr(c%(0)), 883
75 print "Das Programm >> bigformat.prg << ist nun geschrieben."
80 print:print"Bitte Taste drücken":a=inp(2):end
85 '
90 ***** DATA für bigformat.prg *****
95 '
100 DATA 601A,0000,0332,0000,0000,0000,0000,0000
101 DATA 0000,0000,0000,0000,0000,0000,203C,0000
102 DATA 020A,6100,01EA,6100,01F2,0C00,003B,6D00
103 DATA FFEC,0C00,0044,6E00,FFE4,0C00,003B,6600
104 DATA 0016,0A79,0003,0000,024E,0A79,0001,0000
105 DATA 0326,6000,FFC8,0C00,003C,6600,0016,0A79
106 DATA 0002,0000,026C,0A79,0002,0000,0328,6000
107 DATA FFAC,0C00,003D,6600,0016,0A79,0003,0000
108 DATA 032C,0A79,1109,0000,028A,6000,FF90,0C00
109 DATA 003E,6600,0016,0A79,0003,0000,02A8,0A79
110 DATA 0001,0000,032E,6000,FF74,0C00,0042,6600
111 DATA 000A,6100,0012,6000,FF64,0C00,0044,6600
112 DATA FF5C,4267,4E41,203C,0000,02D6,6100,0140
113 DATA 33F9,0000,0328,0000,032A,5379,0000,032A
114 DATA 33F9,0000,0326,0000,0330,6100,00E6,6600
115 DATA 00D0,5379,0000,0330,6A00,FFF0,5379,0000

```

116 DATA 032A,6A00,FFDC,4267,7002,8079,0000,0326
117 DATA 3F00,2F3C,0100,0000,4879,0000,0332,3F3C
118 DATA 0012,4E4E,DFFC,0000,000E,41F9,0000,0332
119 DATA 4280,0C79,0009,0000,032C,6700,001E,11BC
120 DATA 000A,0818,3239,0000,0328,4A79,0000,0326
121 DATA 6700,0004,E349,6100,0050,0C79,0050,0000
122 DATA 0328,6700,0016,323C,0012,4A79,0000,0326
123 DATA 6700,0004,E349,6100,0030,3F3C,0001,42A7
124 DATA 3F3C,0001,3F39,0000,032E,42A7,4879,0000
125 DATA 0332,3F3C,0009,4E4E,DFFC,0000,0014,4A40
126 DATA 6600,001E,6000,FE76,1430,0814,E14A,1430
127 DATA 0813,D441,1182,0813,E04A,1182,0814,4E75
128 DATA 203C,0000,02FC,6100,0046,6100,004E,6000
129 DATA FE4C,4267,2F3C,8765,4321,3F3C,0001,3F39
130 DATA 0000,0330,3F39,0000,032A,3F39,0000,032C
131 DATA 3F39,0000,032E,42A7,4879,0000,0332,3F3C
132 DATA 000A,4E4E,DFFC,0000,001A,4A40,4E75,2F00
133 DATA 3F3C,0009,4E41,5C8F,4E75,3F3C,0001,4E41
134 DATA 548F,4840,4E75,1B45,2A2A,2A20,466F,726D
135 DATA 6174,6965,7275,6E67,732D,5072,6F67,7261
136 DATA 6D6D,2020,532E,442E,202A,2A2A,0A0D,0A0D
137 DATA 205B,4631,5D20,2053,6569,7465,286E,2920
138 DATA 2E2E,2E2E,2E2E,2E2E,3A20,2032,0A0D,205B
139 DATA 4632,5D20,2054,7261,636B,7320,2E2E,2E2E
140 DATA 2E2E,2E2E,2E2E,3A20,3830,0A0D,205B,4633
141 DATA 5D20,2053,656B,746F,7265,6E2F,5472,6163
142 DATA 6B20,2E2E,3A20,2039,0A0D,205B,4634,5D20
143 DATA 204C,6175,6677,6572,6B20,2E2E,2E2E,2E2E
144 DATA 2E2E,3A20,2041,0A0D,205B,4638,5D20,2046
145 DATA 6F72,6D61,7469,6572,656E,202E,2E2E,0A0D
146 DATA 5B46,3130,5D20,2051,7569,7420,210A,0D0A
147 DATA 0D00,466F,726D,6174,6965,7275,6E67,206C
148 DATA 8475,6674,2E20,4269,7474,6520,7761,7274
149 DATA 656E,2E2E,2E0A,0D00,2A2A,2045,7320,6973
150 DATA 7420,6569,6E20,4665,686C,6572,2061,7566
151 DATA 6765,7472,6574,656E,2021,2120,2A2A,0A0D
152 DATA 0000,0001,0050,0050,0009,0000,0000,0000
153 DATA 0002,2808,1408,1408,1408,260A,0406,0604
154 DATA 0E0A,0E0E,120A,1006,120E,1A08,341E,0606
155 DATA 0608,0006

Nun noch einige Bemerkungen zum Programm:

- Das Kopieren einer normalen auf eine erweiterte Diskette ist nur File für File möglich, da das Betriebssystem wegen der unterschiedlichen Diskettenformate nicht die Disketten direkt kopiert.
- Die Verwendung einer erweiterten Diskette als TOS-Systemdiskette ist nicht ohne weiteres möglich, da der Loader im Boot-Sektor fehlt. Daher muß für das Booten dieser Diskette der Boot-Sektor einer anderen Systemdiskette kopiert werden und danach die Einstellungen der erweiterten Diskette mit einem geeigneten Disketten-Monitor wieder eingegeben werden.
- Verwenden Sie die erweiterten Disketten nicht unbedingt als Träger von sehr wichtigen und einmaligen Daten. Sollte die von Ihnen verwendete Diskettenmarke nämlich nicht allzuviel taugen, so kann schon mal der eine oder andere Sektor in den obersten Tracks ausfallen...

Doch nun zurück zur leidigen Theorie. Wie bereits erwähnt, wird aus den diversen Informationen der BIOS-Parameter-Block, kurz BPB, erstellt. Sehen wir uns jetzt diesen BPB etwas genauer an.

14.2.2 Der BIOS-Parameter-Block BPB

Einige Einträge dieses Parameterblocks sind uns ja schon bekannt, da sie auch im Boot-Sektor auftauchen. Der BPB wird beim Aufruf des BIOS-Kommandos "Get BPB" (Nr. 7) erstellt, wenn die Diskette zwischenzeitlich gewechselt wurde. Der BPB enthält im Gegensatz zum Boot-Sektor seine Daten im normalen 16-Bit-Format, und zwar in folgender Reihenfolge:

recsize	Sektorgröße in Bytes	(512)
clziz	Cluster-Größe in Sektoren	(2)
clsizb	Cluster-Größe in Bytes	(1024)
rdlen	Anzahl der Directoriesektoren	(7)
fsiz	FAT-Größe in Sektoren	(5)
fatrec	Startsektor des zweiten FATs	(6)
datrec	erster Datensektor	(rdlen+fsiz+fatrec=18)
numcl	Anzahl der Daten-Cluster	(711)
bflags	FAT-Eintragsgröße in Bit 0: 0=12 Bit, 1=16 Bit	(0)

Die Zahlen in Klammern geben den typischen Inhalt der Einträge bei einer doppelseitig formatierten Diskette an.

Nun wollen wir wieder ein Programm betrachten, mit dem dieser BIOS-Parameter-Block BPB eingelesen und analysiert werden kann. Das Programm ist recht einfach aufgebaut. Zuerst wird ein Prompt ausgegeben, welches auch die Überschrift beinhaltet. Dieses Prompt fordert nun zur Eingabe eines Buchstaben in die Tastatur auf. Dabei ist entweder eine Laufwerksbezeichnung (a,b,c oder d) oder der Buchstabe "q" gültig. "q" beendet das Programm und führt wieder zum Desktop zurück.

Nach der Eingabe testet das Programm, ob ein gültiger Buchstabe eingegeben wurde. Wenn nicht, so wird erneut gestartet, wenn ein 'q' eingegeben wurde, so wird das Programm abgebrochen. Der Buchstabe wird dann durch Subtraktion von "a" in die für den GETBPB-Aufruf nötigen Wert (0-3) umgewandelt. Damit wird nun die GETBPB-Funktion aufgerufen. Man erhält die Adresse des BPB im Register D0 zurück. Die Einträge des BPB werden nun nacheinander ausgelesen, sedezimal (= hexadezimal) ausgegeben und mit dem entsprechenden Text versehen. Damit werden alle wichtigen Informationen über die Diskette auf einen Blick überschaubar. Hier das Programm, welches wiederum mit dem SEKA geschrieben wurde:

```
;** BPB-Analysator S.D. **
```

```
run:
```

```
move.l #prompt,d0
bsr pmsg ;Prompt ausgeben
bsr getkey ;Eingabe des Laufwerks A-D
cmp #'q',d0 ;Quit ?
beq quit ;ja => Desktop
move d0,d6 ;Zeichen retten
bsr pcrLf ;CR ausgeben
sub #'a',d6 ;Wert umwandeln
bmi run ;falsche Eingabe
cmp #3,d6
bgt run ;falsche Eingabe
move d6,-(sp) ;Device-Nr.
move #7,-(sp)
trap #13 ;GETBPB-Funktion
addq.l #4,sp
tst.l d0
beq run ;Error !
move.l d0,a5 ;BPB-Adresse retten
bsr pnext
move.l #bps,d0
bsr pline ;"Bytes pro Sektor"
bsr pnext
move.l #spc,d0
```

```

    bsr    pline                ;"Sektoren pro Cluster"
    bsr    pnext
    move.l #bpc,d0
    bsr    pline                ;"Bytes pro Cluster"
    bsr    pnext
    move.l #dirsec,d0
    bsr    pline                ;"Directory-Sektoren"
    bsr    pnext
    move.l #fatsec,d0
    bsr    pline                ;"FAT-Sektoren"
    bsr    pnext
    move.l #fat2s,d0
    bsr    pline                ;"Start-Sektor des 2. FAT"
    bsr    pnext
    move.l #datsec,d0
    bsr    pline                ;"Start-Sektor der Daten"
    bsr    pnext
    move.l #datc,d0
    bsr    pline                ;"Daten-Cluster"
    move    #'$',d0
    bsr    pchar                ;"$" ausgeben
    move    #12,d0              ;12 Bit annehmen
    btst    #0,(a5)             ;richtig ?
    beq     bits12              ;ja
    move     #16,d0              ;sonst 16 Bit
bits12:
    bsr    phexbyt
    move.l #fatbit,d0
    bsr    pline                ;"Bits pro FAT-Eintrag"
    bra     run                 ;fertig => Neustart
quit:
    clr     -(sp)
    trap    #1
getkey:
    move    #1,-(sp)
    trap    #1
    and.l   #$ff,d0
    addq.l  #2,sp
    rts
pline:
    bsr    pmsg                 ;Print Line/CR
pcrlf:
    move    #10,d0
    bsr    pchar                ;Print CR,LF
    move    #13,d0
pchar:
    move    d0,-(sp)
    move    #2,-(sp)
    trap    #1
    addq.l  #4,sp
    rts
pmsg:
    move.l  d0,-(sp)            ;Print Line (D0)

```

```

        move    #9,-(sp)
        trap    #1
        addq    #6,sp
        rts

pnext:                                ;nächstes Wort holen und ausgeben
        move    #'$',d0
        bsr     pchar                  ;"$" ausgeben
        move    (a5)+,d0
phexword:                                ;Print Hex-Word D0
        moveq    #3,d1
        bra     phex1
phexbyt:                                ;Print Hex-Byte
        moveq    #1,d1
        rol.l    #8,d0
phex1:
        rol.l    #4,d0
        move.l    d0,-(sp)
        move.l    d1,-(sp)
        bsr     phexnib                ;ein Nibble (0-F) ausgeben
        move.l    (sp)+,d1
        move.l    (sp)+,d0
        dbra     d1,phex1
        rts
phexnib:
        and.l    #$7f,d6
        swap     d0
        and.l    #$0f,d0
        add.b     #$30,d0
        cmp.b     #$3a,d0
        bcs     phexn
        add.b     #7,d0
phexn:
        bra     pchar                  ;Nibble ausgeben
prompt:  dc.b     "**** BPB-Analysator S.D. ****",10,13
        dc.b     "Bitte Laufwerk eingeben (a-d) oder",10,13
        dc.b     "'q' für Quit : ",0
bps:     dc.b     " Bytes pro Sektor",0
spc:     dc.b     " Sektoren pro Cluster",0
bpc:     dc.b     " Bytes pro Cluster",0
dirsec:  dc.b     " Directory-Sektoren",0
fatsec:  dc.b     " FAT-Sektoren",0
fat2s:   dc.b     ": Start-Sektor 2.FAT",0
datsec:  dc.b     ": Start-Sektor der Daten",0
datc:    dc.b     " Daten-Cluster",0
fatbit:  dc.b     " Bits pro FAT-Eintrag",10,13,0

```

Hier auch gleich den BASIC-Lader, welcher das BPB-Analyseprogramm als BPBANA.TOS auf der Diskette erstellt:

```

10  *** Erstellung des BPB-Analysators ***
15  '
20  '

```

```

25 print "File >> BPBANA.TOS << wird erzeugt"
30 dim c%( 331):cs#=0
35 for i=0 to 331
40 read a$:c%(i)=val("&H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 4548987 then 70
60 print"Geht leider noch nicht, etwas mit den DATAs stimmt nicht."
65 goto 80
70 bsave "BPBANA.TOS",varptr(c%(0)), 664
75 print "Das Programm >> BPBANA.TOS << ist nun geschrieben."
80 print"Bitte Taste drücken":a=inp(2):end
85 '
90 '***** DATAs für BPBANA.TOS *****
95 '
100 DATA 601A,0000,026E,0000,0000,0000,0000,0000
101 DATA 0000,0000,0000,0000,0000,0000,203C,0000
102 DATA 015E,6100,0100,6100,00D0,0C40,0071,6700
103 DATA 00C4,3C00,6100,00D6,0446,0061,6B00,FFDE
104 DATA 0C46,0003,6E00,FFD6,3F06,3F3C,0007,4E4D
105 DATA 588F,4A80,6700,FFC6,2A40,6100,00D4,203C
106 DATA 0000,01B2,6100,00A2,6100,00C6,203C,0000
107 DATA 01C5,6100,0094,6100,00B8,203C,0000,01DC
108 DATA 6100,0086,6100,00AA,203C,0000,01F0,6100
109 DATA 0078,6100,009C,203C,0000,0205,6100,006A
110 DATA 6100,008E,203C,0000,0214,6100,005C,6100
111 DATA 0080,203C,0000,0229,6100,004E,6100,0072
112 DATA 203C,0000,0242,6100,0040,303C,0024,6100
113 DATA 0048,303C,000C,0815,0000,6700,0006,303C
114 DATA 0012,6100,005C,203C,0000,0252,6100,001A
115 DATA 6000,FF2A,4267,4E41,3F3C,0001,4E41,0280
116 DATA 0000,00FF,548F,4E75,6100,001A,303C,000A
117 DATA 6100,0006,303C,000D,3F00,3F3C,0002,4E41
118 DATA 588F,4E75,2F00,3F3C,0009,4E41,5C4F,4E75
119 DATA 303C,0024,6100,FFE2,301D,7203,6000,0006
120 DATA 7201,E198,E998,2F00,2F01,6100,000C,221F
121 DATA 201F,51C9,FFF0,4E75,0286,0000,007F,4840
122 DATA 0280,0000,000F,0600,0030,0C00,003A,6500
123 DATA 0006,0600,0007,6000,FFA0,2A2A,2A20,2042
124 DATA 5042,2D41,6E61,6C79,7361,746F,7220,2053
125 DATA 2E44,2E20,202A,2A2A,0A0D,4269,7474,6520
126 DATA 4C61,7566,7765,726B,2065,696E,6765,6265
127 DATA 6E20,2861,2D64,2920,6F64,6572,0A0D,2771
128 DATA 2720,6681,7220,5175,6974,203A,2000,2020
129 DATA 4279,7465,7320,7072,6F20,5365,6B74,6F72
130 DATA 0020,2053,656B,746F,7265,6E20,7072,6F20
131 DATA 436C,7573,7465,7200,2020,4279,7465,7320
132 DATA 7072,6F20,436C,7573,7465,7200,2020,4469
133 DATA 7265,6374,6F72,792D,5365,6B74,6F72,656E
134 DATA 0020,2046,4154,2D53,656B,746F,7265,6E00
135 DATA 3A20,5374,6172,742D,5365,6B74,6F72,2032
136 DATA 2E46,4154,003A,2053,7461,7274,2D53,656B
137 DATA 746F,7220,6465,7220,4461,7465,6E00,2020

```

```
138 DATA 4461,7465,6E2D,436C,7573,7465,7200,2020
139 DATA 2020,4269,7473,2070,726F,2046,4154,2D45
140 DATA 696E,7472,6167,0A0D,0000,0000,0002,420E
141 DATA 0E0E,0E0E,0E0E,2600
```

Beim Einschalten des Rechners sind die Daten des BPB nicht vorhanden. Das Betriebssystem erstellt die BPB erst nach dem Booten, wenn es die Anzahl und Kennung der angeschlossenen Laufwerke feststellt. Doch dazu muß erst einmal ein Betriebssystem vorhanden sein.

Ist das TOS nicht im Rechner eingebaut, so muß es erst gebootet werden. Ebenso wird gebootet, wenn zwar ein Betriebssystem eingebaut ist, die Diskette jedoch ein bootbares Betriebssystem (TOS.IMG) enthält und der Boot-Sektor ausführbar ist. Der Vorgang des Bootens läuft in folgenden vier Schritten ab:

1. Der Boot-Sektor wird geladen, und das auf ihm befindliche Boot-Programm wird ausgeführt.
2. Die FAT und das Directory werden von der aktuellen Diskette geladen. Der Lader sucht nun nach dem angegebenen File-Namen (meist TOS.IMG). Findet er es nicht, so gibt er eine Fehlermeldung zurück.
3. TOS.IMG wird ab der Speicheradresse \$40000 geladen.
4. Das geladene Programm wird am Anfang gestartet.

Das TOS.IMG besteht nun seinerseits aus drei Teilen:

- Einem Relocator, einem Programm, welches das Betriebssystem an die eigentlich vorgesehene Adresse (\$6100) schiebt. Dieses Programm löscht den Bildschirm, schiebt den TOS-Imageblock an seine Heimatadresse und startet es dort.
- Die Daten des Betriebssystems (BIOS, XBIOS).
- Die Daten des GEM und des Desktop-Programms.

Wie man also sieht, ist der Aufbau des Betriebssystems im File TOS.IMG recht kompliziert. Das eingebaute TOS, welches in den sechs PROMs (Programmable Read Only Memory) liegt, ist natürlich etwas kürzer, da es nur das Betriebssystem mit GEM enthält und keinen Relocator.

Gehen wir nun über in den nächsten Abschnitt der Datenstrukturen auf Disketten, und zwar dem Aufbau und der Verwaltung des Inhaltsverzeichnisses.

14.3 Das Inhaltsverzeichnis

Das Inhaltsverzeichnis beginnt auf einseitig formatierten Disketten auf Track 1, Sektor 3, und belegt 7 Sektoren einer einseitigen Diskette. Es enthält pro Eintrag außer dem File-Namen und der Extension noch eine Reihe weitere Daten, die für die Verwaltung der Diskette mehr oder weniger wichtig sind.

Jeder Eintrag im Inhaltsverzeichnis besteht aus 32 Bytes, die alle Informationen über das File enthalten, die das Betriebssystem benötigt. Diese 32 Bytes unterteilen sich in acht Datenfelder, die folgendermaßen aufgebaut sind:

1	File-Name	8 Bytes
2	Filetyp (Extender)	3 Bytes
3	Attribut	1 Bytes
4	Reserviert	10 Bytes
5	Uhrzeit	2 Bytes
6	Datum	2 Bytes
7	erster Cluster	2 Bytes
8	File-Größe	4 Bytes

Das erste Feld enthält also den File-Namen. Dieser Name besteht aus ASCII-Zeichen, also nur Buchstaben und Ziffern. Dabei werden auch nur Großbuchstaben verwendet. Der Name ist auf acht Zeichen begrenzt; hat er weniger als acht Zeichen, so wird der Rest mit Leerzeichen (Blanks) aufgefüllt.

Ist das erste Byte des Namens eine Null, so bedeutet dies, daß der Eintrag bisher nie benutzt wurde. Wurde das File bereits verwendet und wieder gelöscht, so findet sich hier eine 229 (\$E5).

Ist das erste Zeichen des Namens ein Punkt (.), so steht dieser Eintrag für ein spezielles Unterverzeichnis, einen Ordner.

Das darauffolgende Feld enthält den File-Typ, auch Extender genannt. Dieser Typ ist auf drei Buchstaben begrenzt (z.B. PRG, TOS, BAS usw.) und wird ebenfalls bei Bedarf mit Leerzeichen aufgefüllt. Auch hier werden nur Großbuchstaben verwendet.

Nun folgt das Byte des File-Attributes. Es enthält bitweise kodiert den Status dieses Eintrages bzw. des Files. Die Bedeutung dieser Bits ist folgende:

Bit	Bedeutung wenn gesetzt (1)
0	nur lesen erlaubt
1	verstecktes File
2	System-File
3	Eintrag ist Disketten-Name
4	Eintrag ist ein Ordner
5	File wurde geändert

Nach diesem Byte folgen zehn Bytes, welche keine Bedeutung haben. Sie gelten lediglich als Reserve-Bytes, die vielleicht für spätere Anwendungen verwendet werden sollen.

Nun folgen zwei Bytes, welche die Uhrzeit der letzten Modifikation des Files enthalten. Hierbei wurde zur Platzeinsparung eine spezielle Kodierung der Zeit verwendet.

Die 16 Bit des Uhrzeiteintrages teilen sich in drei Sektionen, Stunden, Minuten und Sekunden. Diese Aufteilung sieht folgendermaßen aus:

Beispiel: 19:21:34 Uhr

Stunde	Minuten	Sekunden/2
10011	010101	10001

Die Sekunden werden nur in Zweierschritten gewertet, daher steht in den unteren fünf Bits der Uhrzeit eine 17.

Das nächste Feld im Directory enthält das Datum der letzten Änderung der Datei. Die Aufteilung in Jahr, Monat und Tag geschieht hier auf ähnliche Weise wie bei der Uhrzeit. Dabei sind für das Jahr nur sieben Bits reserviert, weshalb grundsätzlich zu der hier enthaltenen Zahl 1980 addiert werden muß. Somit ergibt sich folgender Eintrag:

Beispiel: 12.05.1986

Jahr	Monat	Tag
0000110	0101	01100

Das siebente Feld im Directory enthält die Nummer des ersten Clusters auf der Diskette, der von dem File belegt wird. In diesem üblicherweise aus zwei Sektoren bestehenden Cluster beginnt somit die Speicherung des Files. Wie es ab dort weitergeht, erfahren Sie im nächsten Kapitel über die FAT.

Das letzte Feld enthält schließlich die Länge des Files in Bytes. Hierbei ist zu beachten, daß eventuell weniger Bytes gelesen werden als hier steht, was auch von der FAT abhängt. Die File-Länge ist somit nur als maximale Länge anzusehen.

Anhand dieser Kenntnisse über den Aufbau des Inhaltsverzeichnisses auf den Disketten sind Sie nun in der Lage, mit Hilfe eines Diskettenmonitors die Aufteilung der Diskette zu analysieren. Durch Änderungen der Werte können vielfältige Manipulationen vorgenommen werden, deren Wirkungen jedoch manchmal unangenehm sein können. Aus diesem Grunde empfiehlt es sich, vor solchen Manipulationen eine Kopie der Diskette anzufertigen.

Will man nun ein Programm schreiben, in dem das Inhaltsverzeichnis einer Diskette ausgelesen werden soll, so muß man vor dem Aufruf der entsprechenden Funktion des Betriebssystems einen Puffer für die erwarteten Daten bereitstellen. Der Anfang dieses Puffers wird als die "Disk Transfer Adress (DTA)" bezeichnet.

Dieser Puffer ist 44 Bytes lang und muß dem Betriebssystem durch den Aufruf einer besonderen Funktion angegeben werden. Danach kann die Suche nach Directory-Einträgen beginnen. Dafür wird mit der Funktion SFIRST (search first) der erste passende Eintrag, mit SNEXT (search next) die weiteren Einträge gesucht und in den Puffer an der DTA geladen werden.

Der Puffer enthält nach dem Aufruf alle Informationen, die auch im Directory-Fenster des Desktop erscheinen. Die Aufteilung der Daten ist folgende:

Byte(s)	Inhalt
0...20	reserviert
21	File-Attribut
22,23	Uhrzeit
24,25	Datum
26...29	File-Größe in Bytes (LO,HI)
30...43	File-Name und Extender

Nun zum Programm. Diese Maschinenroutine setzt zuerst die DTA und sucht dann nach dem angegebenen File-Namen im Directory. Ist der angegebene

Name nur " *.*", so wird der erste Eintrag, der dem vorgegebenen Attribut entspricht, geliefert. Ist überhaupt kein passender Eintrag vorhanden, so liefert die Funktion im Datenregister D0 die Fehlernummer -33, File nicht gefunden, zurück. Andernfalls ist dieses Register null.

MOVE.L	#PUFFER, -(SP)	* DTA übergeben
MOVE	#\$1A, -(SP)	* SETDTA-Funktionsnummer
TRAP	#1	* Betriebssystem aufrufen
ADDQ.L	#6, SP	* Stack reparieren
MOVE	##11001, -(SP)	* Dateityp: alle Dateien
MOVE.L	#NAME, -(SP)	* Adresse des File-Namens
MOVE	#\$4E, -(SP)	* SFIRST-Funktionsnummer
TRAP	#1	* Betriebssystem aufrufen
ADDQ.L	#8, SP	* Stack reparieren
TST	D0	* gefunden
BNE	WARNIX	* nein
USW.		
.		
.		
PUFFER:	.ds.b 44	* Platz für die Daten
NAME:	.dc.b " *.*", 0	* alle Namen erlaubt

Um danach den nächsten Eintrag zu suchen, genügt einfach der Programmteil:

MOVE	#\$4F, -(SP)	* SNEXT-Funktionsnummer
TRAP	#1	* Betriebssystem aufrufen
ADDQ.L	#2, SP	* Stack reparieren
TST	D0	* gefunden
BNE	WARNIX	* nein, das war's wohl

Auf diese Weise läßt sich also leicht ein Programm schreiben, welches z.B. das Inhaltsverzeichnis einer Diskette auf dem Drucker ausgibt.

Läßt man sich nun im Desktop das Inhaltsverzeichnis der Diskette anzeigen, so werden Name, Extender, Datum, Uhrzeit und Länge der Files ausgegeben. Wenn Sie dann ein Programm anklicken, so muß das Betriebssystem nicht nur wissen, wo auf der Diskette das File beginnt, sondern auch, wo die weiteren Daten des Files stehen. Diese Informationen beinhaltet die FAT, die wir nun betrachten wollen.

14.4 Die FAT

Die FAT (File Allocation Table) belegt normalerweise fünf Sektoren auf einer (einseitigen) Diskette und beginnt normalerweise auf Track 0, Sektor 2 der Seite 0. Die Größe dieser Tabelle variiert je nach verwendetem Format. Sie wird verwendet, um die Verteilung jedes Files auf der Diskette zu speichern.

Der Grund dafür liegt darin, daß ein File nicht unbedingt Sektoren belegt, die direkt hintereinander liegen. Schließlich werden diejenigen Sektoren, die ein gelöscht File beinhaltet hatten, wieder zur Speicherung neuer Daten freigegeben. Ein neues File, welches auf die Diskette geschrieben wird, wird auf solche freien Sektoren verteilt. Dabei werden belegte Sektoren einfach übersprungen.

Jeder Sektor muß also einen eigenen Eintrag in der FAT besitzen, um als frei oder belegt erkannt werden zu können. Um den Umfang der FAT geringer zu halten, werden immer zwei Sektoren zusammengefaßt und als Cluster bezeichnet, die von 2 bis zum Diskettenende durchnummeriert sind. Die FAT enthält somit nur noch einen Eintrag für zwei Sektoren.

Jeder Eintrag der FAT ist normalerweise zwölf Bytes lang. Einige Formate verwenden 16-Bit-Einträge, was wir jedoch hier vernachlässigen können. Durch die Aufteilung in zwölf Bit ergibt sich, daß je zwei FAT-Einträge drei Bytes einnehmen.

Die ersten zwei Einträge der FAT enthalten Format-Informationen, weshalb die Numerierung auch erst bei 2 beginnt.

Jeder weitere Eintrag repräsentiert nun einen Cluster. Eine Null in einem Eintrag bedeutet, daß der entsprechende Cluster frei ist. Dies bedeutet natürlich nicht, daß die Sektoren keine Daten enthalten, da ein gelöscht File nicht wirklich von der Diskette gelöscht wird. Das Löschen eines Files geschieht lediglich dadurch, daß im Inhaltsverzeichnis der erste Buchstabe des Namens durch eine \$E5 ersetzt wird und die freiwerdenden Cluster durch eine Null in der FAT freigegeben werden. Die Daten selbst sind dennoch vorhanden, aber schwer zu finden.

Enthält ein FAT-Eintrag eine \$FF7, so bedeutet dies einen unbrauchbaren Cluster. Solche Cluster werden beim Formatieren erkannt und markiert. Wenn solche Fehler auf der Diskette vorkommen, z.B. durch einen kleinen Kratzer auf der Diskette, erkennt man dies an der verminderten Kapazität, die nach dem Formatieren gemeldet wird. Sollte ein solcher Fehler jedoch in Track 0 oder 1 auftauchen, so ist die Diskette unbrauchbar, da dort der Boot-Sektor, die FAT und das Directory liegen müssen.

Soll nun ein File geladen werden, so entnimmt das Betriebssystem dem Directory die Nummer des ersten Clusters, der die gewünschten Daten enthält. Der FAT-Eintrag dieses Clusters enthält nun seinerseits die Nummer des nächsten Clusters des Files. Dessen FAT-Eintrag enthält dann wieder die nächste Nummer und so weiter, bis ein Eintrag eine \$FFF enthält. Dies bedeutet, daß der Cluster der letzte des Files ist.

Auch in der FAT können mittels eines Diskettenmonitors Änderungen vorgenommen werden. Hierbei ist die Wahrscheinlichkeit eines Datenverlustes jedoch so hoch, daß man unbedingt vorher eine Kopie der Diskette anfertigen sollte.

14.5 Programmaufbau

Der ATARI ST besitzt einen recht großen Speicher, in den mehrere Programme passen. In der Tat ist es möglich, mehrere Programme gleichzeitig in den Speicher zu legen und ablaufen zu lassen. Ein einfaches Beispiel hierfür sind die Accessories, die ja im Hintergrund laufen.

Diese offene Speicherezuteilung wirft allerdings ein Problem auf. Von den 8-Bit-Rechnern war man gewöhnt, daß ein Programm (Maschinenprogramm) an einer ganz bestimmten Stelle im Speicher liegen muß, damit es auch läuft. Das liegt daran, daß Maschinenprogramme den Speicher direkt adressieren bzw. zu bestimmten Adressen verzweigen, an denen dann das entsprechende Teilprogramm liegen müßte.

Beim ATARI ST ist das jedoch nicht möglich. Woher sollte ein Programmierer auch im voraus wissen, wohin sein Programm geladen wird und ob dort nicht schon ein anderes Programm liegt?

Ein anderes Problem ist es, daß das Betriebssystem wissen muß, wie groß das geladene Programm ist und wieviel Speicher es zum Laufen benötigt. Braucht das Programm nämlich einen zusätzlichen Speicher zum Abspeichern von z.B. eingegebenen Texten, so darf dieser Speicher nicht beim Dazuladen eines anderen Programmes überschrieben werden.

Wie man sieht, reicht es nicht aus, wenn eine Programmdatei auf Diskette nur die Programmdatei selbst enthält. Der Aufbau einer solchen Datei soll in diesem Kapitel erläutert werden. Ein lauffähiges Programm auf Diskette, also .PRG-, .TOS- und .TTP-Dateien, ist in vier Abschnitte unterteilt. Diese Abschnitte sind die folgenden:

Anfang der Datei:	File-Header
	Programm mit Datenfeld
	Symbol-Tabelle (wenn eine existiert)
	Relocation-Daten (wenn vorhanden)

Ende der Datei:

Betrachten wir zunächst den ersten Teil: den Header.

14.5.1 Der Programm-Header

Der Header ist ein 14 Worte langer Programmvorspann, der die Längen der einzelnen Segmente enthält. Der Aufbau des Headers ist folgender:

Byte Nr.	Inhalt
\$00-\$01	\$601A, der Maschinenbefehl BRA *+ \$1A
\$02-\$05	Länge des Programm-Segmentes (text)
\$06-\$09	Länge des Daten-Segmentes (data)
\$0A-\$0D	Länge des Zusatzspeicher-Segmentes (bss)
\$0E-\$11	Länge der Symboltabelle
\$12-\$1B	00, reserviert

Der erste Eintrag ist ein Maschinenbefehl, der den Programmablauf zum Anfang des Programm-Segmentes verzweigt.

Es folgt die Länge des Programm-Segmentes. Dieses Segment, allgemein "text"-Segment genannt, enthält das Programm selbst. Alle Adressen, die das Programm verwendet, sind darin so abgelegt, daß der Programmstart als Adresse 0 angenommen wird. Daten, die dieses Segment erhält, sind unverändert.

Der nächste Eintrag enthält die Länge des Daten-Segmentes, "data"-Segment genannt. Dieses Segment muß unmittelbar im Anschluß an das Programm liegen. In einem Maschinenprogramm wird mit einer "data"-Anweisung die Trennung zwischen text- und data-Segment vorgenommen. Es handelt sich bei den Daten um initialisierte Daten, wie z.B. Texte oder Tabellen. Uninitialisierte Daten wie Datenpuffer für Diskettenoperationen oder Zwischenspeicher enthält das nächste Segment.

Die Länge dieses Zusatzspeichers enthält der vierte Eintrag des Headers. Diesen Speicherbereich nennt man "bss". Nach dem Laden des Programms wird dieser Speicherbereich dem Programm zur Verfügung gestellt und gleichzeitig für andere Anwendungen gesperrt. Sein Inhalt ist allerdings nicht definiert - er muß erst vom Programm beschrieben werden. Der Vorteil des bss-Segmentes gegenüber dem data-Segment ist der, daß dieser Bereich nicht in dem Diskettenfile enthalten zu sein braucht. Dadurch wird ein Programm-File nicht länger als nötig.

Eintrag Nummer fünf enthält die Länge der Symbol-Tabelle. Eine solche Tabelle ist selten vorhanden, da sie für die Funktion des Programms keine Rolle

spielt. Eine Symbol-Tabelle wird, wenn vom Programmierer gewünscht, von einem Compiler bzw. einem Assembler an das compilierte bzw. assemblierte Programm angehängt. Die Symbole entsprechen dabei den in dem Quellprogramm verwendeten Labels von Routinen oder Daten. Der Vorteil einer solchen Tabelle ist z.B. für die Fehlersuche nicht zu verachten, da ein symbolischer Debugger wie der SID beim Disassembler-Listing zu jeder vom Programm verwendeten Adresse den symbolischen Namen anhängt. Ist die Test- und Fehlerphase in der Programmentwicklung jedoch abgeschlossen, so empfiehlt es sich, die Symbol-Tabelle wegzulassen, da sie das Programm-File unnötig lang macht.

Jeder Eintrag in der Symbol-Tabelle ist sieben Worte lang und enthält den Namen, Typ und Wert des Labels:

Byte	Inhalt
\$0-\$7	Symbol-Name, endet mit einer Null
\$8-\$9	Symbol-Typ, wie relocatable, global oder extern
\$A-\$C	Wert, wie Adresse, Register-Nr., Direktwert usw.

Die gesamte Symboltabelle eines Programm-Files kann z.B. mit dem Programm NM68 ausgelesen und ggf. ausgedruckt werden.

Zurück zum Aufbau des Programm-Headers. Die verbleibenden Bytes von \$12 bis \$1B sind reserviert für spätere Anwendungen, müssen jedoch null sein.

Auf den Header folgt nun direkt das Programm selbst, welches wie gesagt eigentlich nur an der Adresse \$0000 funktionieren kann. Um es an der jeweiligen Adresse lauffähig zu machen, an die es geladen wurde, müssen nun alle absoluten Adressen, die in dem Programm auftauchen, geändert werden. Dazu braucht nur die wirkliche Startadresse zu den im Programm enthaltenen Adressen addiert werden. Aber woher weiß das Betriebssystem, das diese Änderungen ja vornehmen muß, wo die absoluten Adressen im Programm stehen? Die Antwort heißt Relocation-Tabelle.

14.5.2 Die Relocation-Tabelle

Hinter der Symboltabelle folgt die Relocation-Tabelle in der Programmdatei. Diese Tabelle enthält die Abstände zwischen den Langworten, die reloziert werden müssen. Das erste Langwort in dieser Tabelle gibt das erste zu ändernde Langwort ab dem Programmanfang an. Danach werden Bytes ver-

wendet, deren Wert wiederum den Abstand zwischen dem gefundenen und dem nächsten zu ändernden Langwort angibt. Ist der Abstand zwischen zwei solchen Langworten größer als 254, so wird ein Byte mit dem Wert 1 eingesetzt und das so oft, bis wieder mit einem Wert kleiner als 255 das nächste betroffene Langwort gefunden werden kann.

Das erste Byte, das eine Null enthält, zeigt das Ende der Relocation-Tabelle an. An dieser Stelle endet auch die gesamte Programmdatei auf der Diskette.

Wird ein Programm nun geladen, so legt das Betriebssystem dieses Programm an eine freie Stelle im Speicher ab und reloziert es. Die Aufteilung des Programms im Speicher ist danach etwas anders als es vorher auf der Diskette war. Vor dem eigentlichen Programm, dem das data- und das bss-Segment folgt, liegt nämlich noch die sogenannte Basepage. Diese 256 Bytes lange Basepage stellt wieder einen Vorspann dar, der Informationen über die aktuelle wirkliche Aufteilung des Programms im Speicher enthält.

Die Basepage ist folgendermaßen organisiert:

Byte	Länge	Inhalt
00	4	Startadresse des Arbeitsspeichers
04	4	HI-Adresse des Arbeitsspeichers +1
08	4	Startadresse des Programms
0C	4	Länge des Programm-Segmentes in Bytes
10	4	Startadresse des data-Segmentes
14	4	Länge des data-Segmentes in Bytes
18	4	Startadresse des bss-Segmentes
1C	4	Länge des bss-Segmentes in Bytes
2C	4	Zeiger auf den "Environment-String"
80	80	Text der Kommandozeile (z.B. bei .TTP)

Alle ungenannten Einträge der Basepage sind reserviert. Nicht nur der Rechner benötigt die Daten aus dieser Tabelle. Ein Programm kann sie ebenfalls sehr gut brauchen. Das beste Beispiel dafür ist die Kommandozeile. Nimmt man für sein Programm den Typ .TTP, so gibt das Betriebssystem beim Aufruf des Programms ein Dialogfenster aus, in dem man Eintragungen machen kann. Diese Zeile kann das Programm nun auswerten.

Um an die Adresse der Kommandozeile heranzukommen, muß man in seinem Programm am Anfang etwa folgende Programmsequenz stehen haben:

```
run:  MOVE.L   4(SP),A0      ;Adresse der Basepage
      LEA      $80(A0),A0
```

A0 enthält die Adresse der Kommandozeile. Damit kann nun weitergearbeitet werden.

14.6 Festplattenformat

Wenden wir uns nun der Festplatte zu. Hier ist es wegen der enormen Speicherkapazität nicht so einfach gestaltet wie auf einer Diskette. Eine Festplatte ist nämlich in bis zu vier Bereiche aufgeteilt, von denen jeder einen Boot-Sektor enthält. Diese Bereiche werden "Partitions" genannt.

Partitionssektor

Der erste Sektor auf der Harddisk (logischer Sektor 0) enthält die Informationen über die Aufteilung der Festplatte. Diese Informationen liegen wie folgt:

Byte	Name	Bedeutung
\$1C2	hd_siz	Gesamtgröße der Harddisk in logischen Vektoren
\$1C6	p0_flg	Partition 0 existiert, wenn p0_flg >0 Ist Bit 7 gesetzt, wird hier gebootet
\$1C7	p0_id	Bezeichnung der Partition (GEM)
\$1CA	p0_st	logische Sektornummer des ersten Sektors in der Partition
\$1CE	p0_siz	Größe der Partition in Sektoren
\$1D2	p1_flg	s.o., Partition 1
\$1D3	p1_id	
\$1D6	p1_st	
\$1DA	p1_siz	
\$1DE	p2_flg	s.o., Partition 2
\$1DF	p2_id	
\$1E2	p2_st	
\$1E6	p2_siz	
\$1EA	p3_flg	s.o., Partition 3
\$1EB	p3_id	
\$1EE	p3_st	
\$1F2	p3_siz	
\$1F6	bsl_st	Start-Sektor der 'bad sector list'
\$1FA	bsl_cnt	Anzahl der defekten Sektoren

Die "bad sector list" wird beim Formatieren der Festplatte angelegt. Sie enthält eine Liste der defekten Sektoren, die sich nicht formatieren ließen. Die Tabelle liegt meist am Ende der Festplatte.

An der Variablen `p*_flg` erkennt das Betriebssystem, ob die Partition existiert (`p*_flg` ungleich Null). Der erste Sektor einer jeden Partition enthält einen Boot-Sektor, in dem die BPB liegt.

Das Betriebssystem bootet von dem ersten Boot-Sektor, dessen `p*_flg` das Bit 7 gesetzt hat.

14.7 Der Disk-Controller

Dieses, zugegebenermaßen große, Kapitel handelt von dem Floppy-Disk-Controller WD1772 (im weiteren nur noch FDC oder Controller genannt), der im ATARI ST verwendet wird.

Für den normalen Datenaustausch zwischen Diskettenlaufwerk und ATARI ST ist es nicht erforderlich, den FDC in Eigenregie zu programmieren. Diese Aufgabe kann, durch entsprechende Aufrufe des BIOS bzw. XBIOS, dem Betriebssystem übergeben werden.

Von Seiten des Betriebssystems werden jedoch nicht alle Möglichkeiten, die der FDC bietet, unterstützt. Für den Anwender, der z.B. ein schnelles Kopierprogramm oder einen Kopierschutz entwickeln möchte, sind die fehlenden Funktionen allerdings von großer Bedeutung. Wer besondere Diskettenformate erstellen möchte, kann nicht die vorhandene Betriebssystemroutine zur Spur-Formatierung benutzen, sondern muß hier selbst aktiv werden.

Um all diese Funktionen in ein Anwenderprogramm einzubinden, muß man sich der direkten Programmierung des Controllers bedienen. Dies ist jedoch nur bei genauer Kenntnis der FDC-Kommandos und deren Abläufe möglich.

14.7.1 Organisation

Um die Programmierung des FDC, die später ausführlich erklärt wird, besser zu verstehen, wäre es günstig, sich zunächst einmal ein Blockschaltbild des WD1772 anzuschauen und danach die einzelnen Funktionsblöcke zu erklären.

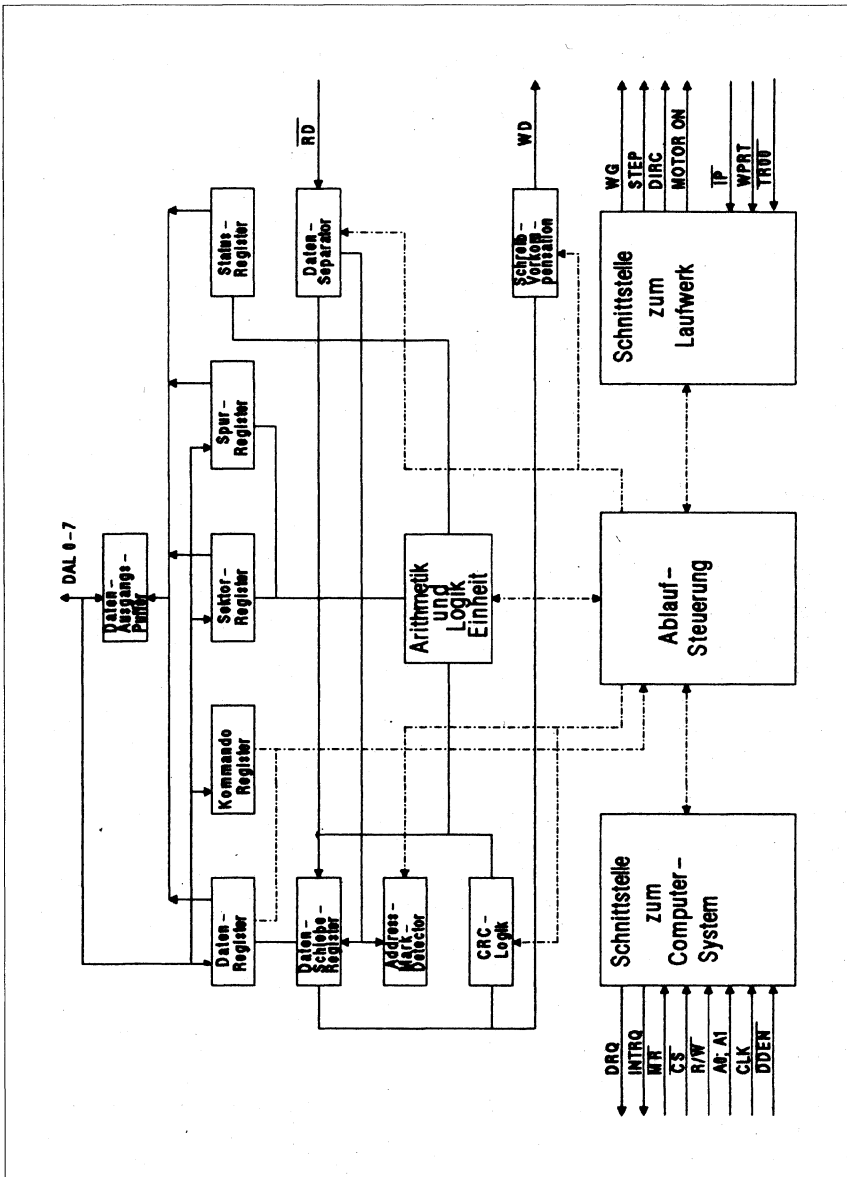


Abb. 14.1: Das Blockschaltbild des WD 1772

Data-Shift-Register (DSR)

In diesem 8-Bit-Schiebe-Register werden während einer Lese-Operation die seriellen Daten, die über den READ-DATA-Eingang (RD) eintreffen, gesammelt. Bei Schreiboperationen wird der Inhalt dieses Registers über den WRITE-DATA Ausgang (WD) seriell ausgegeben. Bei einigen Operationen wird das Data-Shift-Register auch als Zwischenspeicher benutzt.

Data-Register (DR)

Bei Schreib-/Lese-Operationen wird dieses Register als Zwischenspeicher benutzt. Wenn bei einer Lese-Operation das DSR ein 8-Daten-Bit empfangen hat, so wird diese Information in das Daten-Register übertragen. Bei Schreib-Operationen wird, nachdem das DSR ein Byte ausgesendet hat, das nächste Byte vom Daten-Register in das DSR transferiert.

Bei einer Such-Operation (siehe "SEEK-Kommando") enthält das Daten-Register die Nummer der gewünschten Spur.

Track-Register (TR)

Dieses Register enthält normalerweise die Spur-Nummer, über der sich der Schreib-/Lese-Kopf befindet. Wie das "normalerweise" schon vermuten läßt, gibt es hier auch Ausnahmen. Um nämlich das Track-Register auf dem aktuellen Stand zu halten, wird es bei einem Schritt nach innen um 1 erhöht, bzw. bei einem Schritt nach außen um 1 vermindert.

Während dies bei den Befehlen RESTORE und SEEK in jedem Fall geschieht, ist es bei den Befehlen STEP, STEP-IN und STEP-OUT nur dann der Fall, wenn in den Befehlsworten das Update-Flag (u-Bit) gesetzt ist. Bei Schreib-, Lese- und Verify- Operationen wird der Inhalt des Spurregisters mit der aufgezeichneten Spurnummer im ID-Feld verglichen.

Das Spurregister kann gelesen und beschrieben werden, sollte während einer Operation jedoch nicht geladen werden.

Sector-Register (SR)

Bei Schreib- und Leseoperationen enthält dieses Register die Nummer des gewünschten Sektors, welche mit der im ID-Feld aufgezeichneten Sektornummer verglichen wird. Nach einem READ ADDRESS-Befehl befindet sich die Spurnummer aus dem ID-Feld in diesem Register.

Das Sektor-Register kann gelesen und beschrieben werden. Während einer Operation sollte das Spurregister jedoch nicht geladen werden.

Command-Register (CR)

Dieses Register enthält das gerade in der Ausführung befindliche Kommando. Es kann nur beschrieben werden, da ein Lesen automatisch das Status-Register selektiert. Das Kommando-Register sollte während der Ausführung einer Operation nicht geladen werden, außer wenn es sich bei dem neuen Kommando um den FORCE-INTERRUPT-Befehl handelt.

Status-Register (STR)

Die in diesem Register befindliche Information gibt Aufschluß über den Zustand des FDC bzw. des Laufwerks. Die einzelnen Bits dieses Registers werden teilweise abhängig vom bearbeiteten Befehl gesetzt. Das Status-Register kann nur gelesen werden, wobei das gelesene Byte folgende Bedeutung(en) hat:

Bit 7 MOTOR ON

Dieses Bit reflektiert den Zustand des MOTOR-ON-Ausgangs. Es ist nach einem Kommando, also wenn das Busy-Bit schon gelöscht ist, noch für ca. ein bis zwei Sekunden gesetzt.

Bit 6 WRITE PROTECT

Dieses Bit zeigt nach Schreib-Operationen an, ob sich im Laufwerk eine schreibgeschützte Diskette befindet. Ist es gesetzt, so bedeutet das gleichzeitig, daß die gewünschte Schreib-Operation nicht ausgeführt wurde.

Das WPRT-Bit wird ebenfalls (im Falle einer schreibgeschützten Diskette) nach einem Typ-1-Kommando gesetzt.

Zurückgesetzt wird dieses Bit, wenn eine Operation mit einer, nicht schreibgeschützten, Diskette stattgefunden hat.

Bit 5 SPIN UP / RECORD TYPE

SPIN UP: Bei Typ-1-Kommandos wird dieses Bit nach Abschluß der Spin-Up-Sequenz gesetzt. Damit soll angezeigt werden, daß der Laufwerksmotor seine Nenndrehzahl (wahrscheinlich) erreicht hat.

RECORD TYPE: Nach einem READ-SECTOR-Befehl läßt sich hieran erkennen, ob das Daten-Feld mit einem "normalen" oder "gelöschten" Data-Mark beginnt.

Bit 5 = 0, "normales" Data-Mark (\$FB)

Bit 5 = 1, "gelöschtes" Data-Mark (\$F8)

Bit 4 RECORD NOT FOUND (RNF)

Wird kein korrektes ID-Feld gefunden, so wird dieses Bit gesetzt. Dies kann nach einem READ-SECTOR-, WRITE-SECTOR- oder READ-ADDRESS-Kommando der Fall sein.

Nach einen READ SECTOR-Befehl kann das RNF-Bit aber auch trotz eines korrekten ID-Feldes gesetzt sein. Dies ist dann der Fall, wenn innerhalb der 43 Bytes, die dem letzten CRC-Byte des ID-Feldes folgen, kein Data-Mark gefunden wurde.

Bit 3 CRC ERROR

Dieses Bit wird gesetzt, wenn der Inhalt des CRC-Feldes (im Daten- oder ID-Feld) nicht mit dem Inhalt des CRC-Registers übereinstimmt.

Bit 2 LOST DATA / TRACK 00

LOST DATA: Wird bei Typ-2- und Typ-3-Kommandos nicht innerhalb der erforderlichen Zeit auf eine Datenanforderung (angezeigt durch DRQ-Ausgang bzw. DRQ-Statusbit) reagiert, so wird dieses Bit gesetzt.

TRACK 00: Bei Typ-1-Kommandos ist dieses Bit gesetzt, wenn sich der Schreib-/Lese-Kopf über der Spur Null befindet.

Bit 1 DATA REQUEST / INDEX

DATA REQUEST: Bei Typ-2- und Typ-3-Kommandos wird dieses Bit gesetzt, wenn Daten bereitstehen bzw. benötigt werden. Es wird durch Lesen oder Schreiben des Daten-Registers zurückgesetzt.

INDEX: Bei Typ-1-Kommandos ist dieses Bit während eines eintreffenden Index-Impulses gesetzt.

Bit 0 BUSY

Während der Ausführung eines Kommandos ist dieses Bit gesetzt.

CRC-Logic

Um Lesefehler zu vermeiden, muß man sich eines Verfahrens bedienen, das eine hohe Datensicherheit gewährleistet. Das hier angewandte Verfahren funktioniert so, daß aus den geschriebenen Daten, nach einem bestimmten Algorithmus eine 16-Bit-Prüfsumme gebildet wird, welche zusätzlich, im Anschluß an die Daten, auf die Diskette geschrieben wird.

Werden nun diese Daten wieder gelesen, so wird nach dem gleichen Algorithmus erneut die Prüfsumme gebildet. Stimmt diese mit der aufgezeichneten überein, so hat man eine fast 100prozentige Sicherheit, daß die Daten richtig gelesen wurden. Durch den relativ komplizierten Algorithmus ist es nämlich sehr unwahrscheinlich, daß sich trotz Lesefehlers die gleiche Prüfsumme ergibt.

Diese 16-Bit-Prüfsumme wird "Cyclic Redundancy Check (CRC)" genannt. Für die Erzeugung und Kontrolle der Prüfsumme ist die CRC-Logik zuständig. Die Berechnung erfolgt aus Geschwindigkeitsgründen durch eine Hardware, die die Summe nach dem Polynom : $CRC(x) = x^{16} + x^{12} + x^5 + 1$ berechnet.

ARITHMETIC/LOGIC UNIT (ALU)

Die ALU wird einerseits zur Registermodifikation (erhöhen, vermindern), andererseits für Vergleiche zwischen Register und den auf der Diskette enthaltenen Informationen in den ID-Feldern benutzt.

ADDRESS-MARK-DETECTOR

Das wahrscheinlich wichtigste Teil im FDC ist dieser Detektor. Wie der Name schon verrät, besitzt dieses Teil die Fähigkeit, ein Address Mark zu erkennen. Eine solche Markierung kennzeichnet den Anfang eines ID-Feldes (Index-Address-Mark) bzw. den Anfang eines Datenfeldes (Data-Address-Mark).

Doch wozu braucht man einen speziellen Detektor? Nehmen wir als Beispiel einmal den Wert "\$FE", den der Controller als ein Index-Address-Mark interpretiert. Auch ohne besonderen Detektor ist es keine Schwierigkeit dieses "\$FE" zu finden.

Das Problem wird allerdings dann deutlich, wenn man bedenkt, daß dieser Wert aber auch durchaus in einem Daten-Feld vorkommen kann. An dieser

Stelle darf er jedoch nicht als Index-Address-Mark gewertet werden. Wie ist es also möglich, ein "\$FE" von einem anderen "\$FE" zu unterscheiden? Nun, genau diese Funktion erfüllt der Address-Mark-Detector.

Wie zuvor schon erwähnt, besteht die geschriebene Information nicht nur aus Daten-Impulsen. Es sind gleichermaßen auch Takt-Impulse darin enthalten. Diese werden, bei Lese-Operationen, vom Datenseparator aus dem Signal herausgefiltert und dem Address-Mark-Detector zugeführt.

Bei einem WRITE-TRACK-Kommando werden Werte, die größer als als "\$F4" sind, in besonderer Weise behandelt. Gemeinsam gilt für diese Werte jedoch, daß sie ohne Takt-Impulse geschrieben werden. Die so geschriebenen Werte bestehen also nur aus Daten-Impulsen.

Wird nun diese Information gelesen, so werden keine Takt-Impulse vom Datenseparator geliefert, da ja keine in dem Signal vorhanden sind. Erst durch das Fehlen der Takt-Impulse wird der Address-Mark-Detector aktiviert. Er kann also ein Address Mark nur dann erkennen, wenn es ohne Takt-Impulse geschrieben wurde.

Es existiert aber noch ein weiteres Problem, das ihnen wahrscheinlich noch gar nicht bewußt wurde, weil bisher nur von "vollständigen" Daten-Bytes die Rede war. Diese werden allerdings seriell aufgezeichnet, wobei jedoch der Anfang eines Bytes in keiner Weise gekennzeichnet ist.

Wenn nun bei einer Lese-Operation 8 Bit im DSR gesammelt wurden, so kann man nicht davon ausgehen, daß diese tatsächlich zu einem einzigen Daten-Byte gehören. Es könnten genauso gut jeweils vier Bit, aus zwei verschiedenen Daten-Bytes, darin enthalten sein.

Der Address-Mark-Detector würde ein Address-Mark nur dann erkennen, wenn das Sammeln der Daten-Bits zufällig Byte-synchron verläuft. Daß hier nichts dem Zufall überlassen werden darf, ist leicht einzusehen. Man muß also eine Möglichkeit haben, den Anfang eines Bytes zu erkennen. Dies geschieht durch Synchronisations-Bytes. Diese (jeweils drei) werden beim Formatieren einer Spur, vor jedes Address-Mark geschrieben. Die "SYNC-Bytes" enthalten, genau wie die Address-Marks, keine Takt-Impulse und aktivieren dadurch den Address-Mark-Detector.

Der Controller, der natürlich ständig über den Zustand des Address-Mark-Detectors unterrichtet ist, liest nun so lange die seriellen Daten-Bits, bis der Inhalt im DSR einem SYNC-Byte, dessen Wert er ja kennt, entspricht. Ab diesem Punkt muß zwangsläufig das nächste eintreffende Bit das erste Bit des folgenden Bytes sein.

Eines ist aber noch zu bemerken. Während der Detektor eingeschaltet ist, können die gelesenen Bytes verfälscht werden (siehe "READ-TRACK-Kommando"). Weshalb und wann? Wenn der Detektor durch die fehlenden Takt-Impulse aktiviert wird, so geht der FDC davon aus, daß SYNC-Bytes mit einem anschließenden Address-Mark folgen. In der Synchronisationsphase, also während der Rekonstruktion eines SYNC-Bytes, werden einige der gelesenen Daten-Bits verworfen. Sollte sich der Detektor "aus Versehen" in einem Daten-Feld auf die Suche nach einem Address-Mark begeben, so werden die Daten bis zum Erkennen des "falschen Alarms" natürlich verändert.

Da der Address-Mark-Detektor, empfindlich wie er nun mal ist (und auch sein muß), dazu neigt, übersensibel auf fehlende Takt-Impulse zu reagieren, wird er vorsorglich (während ID-bzw. Daten-Felder gelesen werden) ausgeschaltet.

DATA SEPARATOR

Der Datenseparator ist eigentlich schon bei den Ausführungen zum Address-Mark-Detektor beschrieben worden. Deshalb sei hier nur kurz erwähnt, daß er die Aufgabe hat, aus dem gelesenen Signal die Takt-Impulse zu entfernen, die, sozusagen als Abfallprodukt, zur Steuerung des Address-Mark-Detectors benutzt werden.

Die Schnittstelle zum Computer-System

Das Prozessor-Interface besteht aus den acht bidirektionalen Daten-Leitungen (DAL0- DAL7), den beiden Adressen-Leitungen (A0,A1), der Datenanforderung (DRQ), der Interrupt-Anforderung (INTRQ), dem Chip Select (CS), der Schreib-/Lese-Leitung (R/W), dem Clock-Eingang (CLK) und dem Master-Reset-Eingang (MR). Über diese Anschlüsse wird der Austausch von Daten und Steuersignalen, zwischen Prozessor und FDC, abgewickelt.

Die Schnittstelle zum Laufwerk

Die Informationen, die der Controller vom Laufwerk erhält sind:

- a) ob eine schreibgeschützte Diskette eingelegt ist (WPRT=1),
- b) ob sich der Schreib-/Lese-Kopf über der Spur 0 befindet (TR00=0),
- c) ob die Diskette eine vollständige Umdrehung beendet hat (IP=0)

und

- d) die seriellen Daten, die gelesen wurden (RD).

Die Signale, die der Controller zum Laufwerk überträgt sind:

- a) Einschalten des Laufwerksmotors (MOTOR ON=1),
- b) Ausführen eines Schrittes des Schreib-/Lese-Kopfes (STEP=1),
- c) die Richtung des Schrittes (DIRC=0 oder 1),
- d) Einschalten der Schreiblogik (WG=1)

und

- e) die seriellen Daten, die geschrieben werden sollen (WD).

Die Ablaufsteuerung

Wird ein Kommando vom FDC ausgeführt, so müssen natürlich die einzelnen Funktionsteile, in einer bestimmten Reihenfolge, ein- bzw. ausgeschaltet werden. Ferner werden auch Daten zwischen den einzelnen Registern des FDC transferiert, Berechnungen ausgeführt, Eingangsleitungen abgefragt und die Zustände der Ausgangsleitungen geändert. Der gesamte zeitliche Ablauf, der ja vom übergebenen Kommando abhängig ist, wird von der Ablaufsteuerung ausgeführt bzw. überwacht.

Lese-Operationen

Lese-Operationen finden im allgemeinen nur durch das READ-SECTOR-Kommando statt. Die anderen Befehle, durch welche auch Daten von der Diskette gelesen werden können, dienen nur Diagnosezwecken und haben für den normalen Betrieb keine Bedeutung.

Die Sektorlängen können 128, 256, 512 oder 1024 Byte betragen. Die Sektorlänge wird beim Formatieren durch das "Längen-Feld" (das vierte Byte im ID-Feld) bestimmt.

Soll ein Sektor gelesen werden, so erkennt der Controller anhand des Längenfeldes, welche Anzahl von Daten-Bytes ab dem DATA-ADDRESS-MARK gelesen werden müssen. Voraussetzung für ein fehlerfreies Lesen der Daten-Bytes ist, daß während dieser Zeit der ADDRESS-MARK-DETECTOR ausgeschaltet wird, da durch ihn Lesefehler verursacht werden können.

Schreib-Operationen

Bevor auf die Diskette, aufgrund eines Schreib-Befehls, geschrieben werden kann, muß der WRITE GATE-Ausgang vom FDC aktiviert werden.

Als Vorsichtsmaßnahme gegen unbeabsichtigtes Schreiben geschieht dies jedoch erst, nachdem als Reaktion auf den vom FDC gesetzten DRQ-Ausgang

das Datenregister geladen wird. Sollte dies nicht geschehen, so wird die Befehlsausführung beendet, INTRQ gesetzt, das LOST-DATA-Status-Bit gesetzt und das BUSY-Status-Bit gelöscht.

Wird auf die erste Datenanforderung reagiert, so wird das Schreib-Kommando ausgeführt. Sollte dem FDC, auf eine folgende Datenanforderung, kein weiteres Daten-Byte übertragen werden, so wird der Befehl nicht abgebrochen, sondern ein "Null-Byte" statt dessen geschrieben. Auch in diesem Fall ist nach Beendigung des Befehls das LOST-DATA-Bit gesetzt. Würden bei einem WRITE-SECTOR-Kommando also nur 112 Byte übertragen, so würde der Controller die restlichen 400 Byte (bei einer Sektorgröße von 512 Byte) mit Nullen füllen.

Man sollte es unterlassen, diesen Sachverhalt dazu auszunutzen, einen Sektor teilweise zu löschen. Da das LOST-DATA-Bit in jedem Fall gesetzt ist, läßt sich nicht erkennen, ob nicht vielleicht bei der Übertragung der ersten Bytes ein Fehler auftrat.

Das Schreiben wird generell verhindert, wenn der WRITE-PROTECT-Eingang auf LOW liegt. In diesem Fall wird jeglicher Schreib-Befehl sofort abgebrochen, INTRQ auf HIGH gelegt, das WRITE PROTECT-Status-Bit gesetzt und das BUSY-Status-Bit gelöscht.

Um bei einer höheren Schreibdichte, wie sie auf den inneren Spuren existiert, die Datensicherheit zu erhöhen, gibt es die Möglichkeit, eine Schreib-Vorkompensation einzuschalten. Ist in den Schreibkommandos das Bit für die Schreib-Vorkompensation gelöscht, so wird der Datenstrom über WRITE-DATA, abhängig vom zu schreibenden Bit-Muster, 125 ns früher oder später ausgesendet. Die folgende Tabelle zeigt, wann welcher Fall gegeben ist:

X 1 1 0	früher	X 0 1 1	später
0 0 0 1	früher		
1 0 0 0	später		
....			
....	nächstes zu übertragendes Bit		
....	das Bit, das gerade gesendet wird		
..	zuvor übertragene Bits		

Die Schreib-Vorkompensation wird bei 5,25-Zoll-Disketten normalerweise auf den inneren Spuren, auf denen ja eine höhere Datendichte herrscht, eingeschaltet. Bei dem 3,5-Zoll-Format, auf dem die Datendichte der äußeren Spuren schon die Datendichte der mittleren Spuren des 5,25-Zoll-Formats erreicht, wird die Vorkompensation im allgemeinen ständig eingeschaltet.

14.7.2 Kommando-Beschreibung

Nachdem nun der interne Aufbau des FDC und auch einige grundlegende Abläufe bei Schreib- und Leseoperationen erklärt wurden, kommen wir zu den Kommandos.

Der WD1772 kennt elf verschiedene Befehle (Kommandos), die in vier Gruppen bzw. Typen unterteilt sind. Die folgende Tabelle zeigt diese im Gesamtüberblick.

		Bit							
Typ	Kommando	7	6	5	4	3	2	1	0
I	Restore	0	0	0	0	h	V	r1	r0
I	Seek	0	0	0	1	h	V	r1	r0
I	Step	0	0	1	u	h	V	r1	r0
I	Step-in	0	1	0	u	h	V	r1	r0
I	Step-out	0	1	1	u	h	V	r1	r0
II	Read Sector	1	0	0	m	h	E	0	0
II	Write Sector	1	0	1	m	h	E	P	a0
III	Read Address	1	1	0	0	h	E	0	0
III	Read Track	1	1	1	0	h	E	0	0
III	Write Track	1	1	1	1	h	E	P	0
IV	Force Interrupt	1	1	0	1	I3	I2	I1	I0

Diese Kommandos haben jeweils mehrere Flag-Bits, die im einzelnen folgende Bedeutung haben:

h = Motor On Flag

h = 0, Motor On-Test einschalten

h = 1, Motor On-Test ausschalten

Wird der Laufwerksmotor eingeschaltet, so sollte eine Wartezeit eingelegt werden, bis der Motor seine Nenndrehzahl erreicht hat. Dies wird vom WD1772 so gehandhabt, daß er nach dem Einschalten des Motors sechs Index-Impulse abwartet. Bei einer Nenndrehzahl des Motors von 300 UPM beträgt diese Wartezeit mindestens eine Sekunde. Dieser Vorgang (Spin-Up-Sequenz genannt) soll sicherstellen, daß die Motoren ihre Solldrehzahl erreicht haben, wenn die Schreib-/Lese-Operationen stattfinden.

Nach Beendigung eines Kommandos werden die Laufwerksmotoren erst nach zehn weiteren Umdrehungen der Diskette (ca. zwei Sekunden) ausgeschaltet.

Folgt in der Nachlaufphase ein weiteres Kommando, wäre es natürlich reine Zeitverschwendung, erneut eine Spin-Up-Sequenz einzulegen. Aus diesem Grunde wurde in den Controller ein Motor-On-Test implementiert.

Ist dieser eingeschaltet ($h=0$), so wird erst einmal der Motor-On-Ausgang getestet. Nur falls Motor-On auf LOW liegt, legt der FDC die zuvor beschriebene Spin-Up-Sequenz ein. Liegt Motor-On jedoch auf High, so nimmt der Controller an, daß die Motoren mit Solldrehzahl laufen und fährt mit der Abarbeitung seines Kommandos fort.

Wenn der FDC ein Kommando (mit gesetztem h -Bit) empfängt, schaltet er zunächst die Laufwerksmotoren ein, indem er den MOTOR-ON-Ausgang auf HIGH legt. Dies ist unabhängig davon, ob der MOTOR-ON-Ausgang vielleicht schon auf High liegt. Danach beginnt er sofort mit der Ausführung des Kommandos. Er wartet also nicht bis sechs Index-Impulse eingetroffen sind.

V = Verify Flag

V = 0, Verify ausschalten

V = 1, Verify einschalten

Dieses Flag-Bit existiert nur in der Gruppe der Typ-I-Kommandos. Ist es gesetzt, so nimmt der Controller, nach einem Step-Kommando bzw. nach dem letzten Step in einem Restore- oder Seek-Kommando, eine Spurverifizierung vor. Dies geschieht in der Form, daß nach dem Step ein korrektes ID-Feld gesucht wird, dessen Spur-Nummer mit dem Inhalt des Spur-Registers übereinstimmt.

Ob man nun ein Kommando mit oder ohne Verify ausführt, sollte von dem nachfolgenden Kommando abhängig gemacht werden, da ein Verify nicht unbedingt nötig und auch nicht immer sinnvoll ist.

Folgt ein READ- oder WRITE-SECTOR-Befehl und der Schreib-/Lese-Kopf befindet sich nicht über der gewünschten Spur, so wird keinesfalls ein falscher Sektor gelesen oder geschrieben, da bei diesen Kommandos generell ein Verify ausgeführt wird, welches außerdem noch etwas ausführlicher ist. Hier erübrigt sich also ein STEP-Befehl mit Verify.

Völlig unsinnig ist ein Verify z.B., wenn eine neue Diskette formatiert wird. Das Verify nach jedem Step-Befehl würde sowieso negativ ausfallen. Da der Controller allerdings für die Zeit von fünf Umdrehungen nach einem korrekten ID-Feld sucht, werden beim Formatieren von 80 Spuren ca. 1,5 Minuten kostbarer Zeit sinnlos vergeudet.

Wird auf einer, schon mit Daten beschriebenen, Diskette nachträglich eine einzelne Spur neu formatiert, dann sollte sicherheitshalber ein Verify veranlaßt werden. Denn der WRITE-TRACK-Befehl, der ja zum Formatieren einer Spur verwendet wird, nimmt vor seiner Ausführung keinerlei Tests an der Spur vor. Es wird also die Spur formatiert, über der sich der Schreib-/Lese-Kopf gerade befindet.

r1, r0 = Stepping Rate

0 0 2ms

0 1 3ms

1 0 5ms

1 1 6ms

Mit diesen beiden Bit läßt sich die Stepping-Rate programmieren. Das ist die Verzögerungszeit, die bei einem SEEK- bzw. RESTORE-Kommando zwischen den einzelnen Step-Impulsen eingelegt wird. Man hat dadurch die Möglichkeit, den FDC in gewissen Grenzen an die mechanischen Gegebenheiten des Laufwerks anzupassen.

Nehmen wir als Beispiel einmal ein Laufwerk, dessen Kopfmechanik für einen Schritt 6 ms benötigt. Wenn nun die Stepping-Rate auf 3 ms programmiert ist und der Kopf durch ein Seek-Kommando von Spur 0 auf Spur 40 bewegt werden soll (was 39 Step-Impulsen entspricht), so würde der Kopf nur die Spur 20 erreichen, da aufgrund der mechanischen Trägheit, jeder zweite Impuls "verschluckt" wird.

Mit einzelnen Step-Impulsen (Step, Step-in, Step-out) wird der Kopf wahrscheinlich korrekt gesteuert, da die Dauer der Step-Impulse durch die Stepping-Rate nicht beeinflußt wird. Durch das interne Timing des FDC bestimmt, beträgt diese einheitlich 4 ms.

u = Update Flag

u = 0, kein Update des Spur-Registers

u = 1, Update des Spur-Registers

Ist bei einem STEP-, STEP-IN- oder STEP-OUT-Kommando das u-Bit gesetzt, so wird nach der Operation das Spur-Register um 1 erhöht bzw. um 1 vermindert. Das bedeutet allerdings nicht, daß der Inhalt des Spur-Registers mit der tatsächlichen Spur übereinstimmt. Hierzu müssen zwei Voraussetzungen erfüllt sein:

- 1) Die tatsächliche Spur-Nummer muß vor dem Step mit dem Inhalt des Spur-Registers übereingestimmt haben.
- 2) Es darf nicht versucht werden, eine Spur-Nummer >82 anzusteuern. Wenn sich der Schreib-/Lese-Kopf z.B. auf der Spur 82 (die letzte vom Laufwerk erreichbare Spur) befindet und auch der Inhalt des Spur-Registers 82 beträgt, so würde nach fünf STEP-IN Kommandos das Spur-Register die Spur-Nummer 87 enthalten, während sich der Kopf immer noch auf Spur 82 befindet.

m = Multiple Sector

m = 0, einen Sektor lesen bzw. schreiben

m = 1, mehrere Sektoren lesen bzw. schreiben

Durch dieses Bit hat man die Möglichkeit, mit nur einem READ- bzw. WRITE-SECTOR-Befehl, maximal alle Sektoren der Spur zu lesen oder zu schreiben. Voraussetzung dazu ist aber, daß die Sektor-Nummern eine lückenlose Reihenfolge bilden. Die Nummer des ersten zu lesenden bzw. zu schreibenden Sektors wird zuvor in das Sektor-Register geschrieben. Nachdem der FDC diesen Sektor gelesen bzw. geschrieben hat, erhöht er das Sektor-Register, und versucht den nächsten Sektor zu lesen bzw. zu schreiben. Dies setzt sich so lange fort, bis kein weiterer Sektor mehr gefunden wird, bzw. das Kommando durch einen FORCE-INTERRUPT-Befehl beendet wird.

a0 = Data-Address-Mark

a0 = 0, normales Data Mark schreiben (\$FB)

a0 = 1, gelöschttes Data Mark schreiben (\$F8)

Das Data-Address-Mark kennzeichnet den Beginn des Datenfeldes. Da durch das a0-Bit (je nachdem, ob es gesetzt oder gelöscht ist) unterschiedliche Data-Address-Marks geschrieben werden können, ist man in der Lage, einen Sektor auf einfache Weise zu kennzeichnen. Nach einem READ SECTOR-Kommando wird die Art des Data-Address-Marks im RECORD-TYPE-Statusbit angezeigt.

E = 30ms Settling Delay

E = 0, keine Kopfberuhigungszeit

E = 1, 30ms Kopfberuhigungszeit

Es gibt Laufwerkstypen, bei denen der Schreib-/Lese-Kopf nicht ständig das Speichermedium berührt. Bei diesen läßt sich der Kopf durch einen Hubmagneten anheben bzw. absenken. Diese, Head Load genannte, Einrichtung soll den Verschleiß der Diskette verringern, indem man den Kopf nur dann auf die

Diskette absenkt, wenn tatsächlich Lese- oder Schreiboperationen erfolgen. Durch dieses Absenken treten jedoch Schwingungen in der Kopf-Mechanik auf, die einen optimalen Kontakt zwischen Speichermedium und Kopf für eine gewisse Zeit verhindern. Durch Setzen des E-Bits läßt sich ein Delay einschalten, welches diese Zeit überbrückt.

P =	Write Precompensation
P = 0,	Schreib-Vorkompensation einschalten
P = 1,	Schreib-Vorkompensation ausschalten
I0-I3 =	Interrupt-Bedingungen
I0 = 1,	Keine Bedeutung
I1 = 1,	Keine Bedeutung
I2 = 1,	Interrupt bei jedem Index-Impuls
I3 = 1,	Sofortiger Interrupt
I0-I3 = 0,	Laufendes Kommando ohne Interrupt beenden

Die Typ-I-Kommandos

Die Typ-2- und Typ-3-Kommandos, die für das Lesen und Schreiben von Daten zuständig sind, beziehen sich immer auf die Spur, über der sich der Schreib-/Lese-Kopf momentan befindet. Für dessen Positionierung ist die, aus den Kommandos "Restore", "Seek", "Step", "Step-in" und "Step-out" bestehende, Gruppe zuständig.

RESTORE (SPUR 0 suchen)

Kommando-Wort:	7	6	5	4	3	2	1	0

	0	0	0	0	h	v	r1	r0

Wird dem FDC dieses Kommando übergeben, so testet er zunächst den TR00-Eingang. Sollte sich dieser auf LOW befinden, da der Schreib-/Lese-Kopf bereits über der Spur Null steht, so wird lediglich das Spur-Register auf Null gesetzt.

Steht der Schreib-/Lese-Kopf nicht über der Spur Null, so werden so lange STEP- OUT Impulse erzeugt, bis der TR00-Eingang auf LOW ist. Sollte dies nach 255 Step-Impulsen noch nicht der Fall sein, so wird das Kommando abgebrochen. Die Beendigung des Kommandos wird durch Setzen des INTRQ-Ausgangs und Löschen des BUSY-Statusbits angezeigt

SEEK (Spur suchen)

```
Kommando-Wort:  7  6  5  4  3  2  1  0
                 -----
                 0  0  0  1  h  V  r1 r0
```

Mit diesem Befehl kann man den Schreib-/Lese-Kopf direkt über eine bestimmte Spur steuern. Dazu wird die gewünschte Spur-Nummer ins Daten-Register geladen. Voraussetzung für ein ordnungsgemäßes Funktionieren dieses Befehls ist, daß sich die aktuelle Spur-Nummer im Spur-Register befindet. Werden mehrere Laufwerke betrieben, muß also eventuell auch das Spur-Register geladen werden, damit in diesem auch tatsächlich die aktuelle Spurnummer steht.

Erhält der FDC ein SEEK-Kommando, so vergleicht er das Spur-Register mit dem Daten-Register, wodurch er feststellt, ob Step-in- oder Step-out-Impulse erforderlich sind. Danach werden Step-Impulse für die entsprechende Richtung ausgegeben. Ein UPDATE des Spur-Registers erfolgt nach jedem Step-Impuls. Sobald die Inhalte von Spur- und Daten-Register gleich sind, ist die Ziel-Spur erreicht und das Kommando beendet. Dies wird durch Setzen des INTRQ-Ausgangs und löschen des BUSY-Statusbits angezeigt.

STEP

```
Kommando-Wort:  7  6  5  4  3  2  1  0
                 -----
                 0  0  1  u  h  V  r1 r0
```

Dieser Befehl veranlaßt den FDC einen Step-Impuls auszugeben. Die Richtung ist dabei die gleiche wie bei einem vorangegangenen Step-Impuls, da der Zustand des DIRECTION-Ausgangs nicht verändert wird. Der INTRQ-Ausgang wird danach auf HIGH gelegt und das BUSY-Statusbit gelöscht.

STEP-IN

```
Kommando-Wort:  7  6  5  4  3  2  1  0
                 -----
                 0  1  0  u  h  V  r1 r0
```

Bei einem "Step-In"-Kommando wird der DIRECTION-Ausgang, unabhängig vom vorherigen Zustand, auf HIGH gelegt und ein Step-Impuls ausgesendet. Durch diesen wird der Schreib-/Lese-Kopf einen Schritt, in Richtung Diskettenmitte, bewegt. Der INTRQ-Ausgang wird auf HIGH gelegt und das BUSY-Statusbit gelöscht.

STEP-OUT

Kommando-Wort:	7	6	5	4	3	2	1	0

	0	1	1	u	h	V	r1	r0

Bei diesem Befehl wird der DIRECTION-Ausgang, unabhängig vom vorherigen Zustand, auf LOW gelegt, bevor der Step-Impuls ausgegeben wird. Der Schreib-/Lese-Kopf wird durch den Impuls einen Schritt, in Richtung Diskettenrand, bewegt. Der INTRQ-Ausgang wird danach auf HIGH gelegt und das BUSY-Statusbit gelöscht.

Die Verify-Sequenz bei den Typ-I-Kommandos

Wurde im Kommandowort das V-Bit gesetzt, so wird, wenn die gewünschte Spur erreicht ist, folgende Sequenz, nach Einlegen einer 30ms Verzögerung, durchlaufen:

Die Spur-Nummer aus dem ersten gelesenen ID-Feld wird mit dem Inhalt des Spur-Registers verglichen. Bei Übereinstimmung werden die CRC-Bytes des ID-Feldes getestet. Sollten diese mit den, durch die CRC-Logik, berechneten übereinstimmen, so ist die Verify-Sequenz fehlerfrei abgeschlossen. Der INTRQ-Ausgang wird auf HIGH gelegt und das BUSY-Statusbit gelöscht.

Ist in dem ID-Feld entweder die Spur-Nummer falsch oder die CRC-Prüfsumme nicht korrekt, wird das nächste ID-Feld gelesen und ein erneuter Test vollzogen.

Wurde nicht innerhalb von fünf Umdrehungen der Diskette ein ID-Feld mit richtiger Spur-Nummer und gültiger Prüfsumme gefunden, so wird das RECORD-NOT-FOUND-Bit im Status-Register gesetzt. Das Kommando wird abgebrochen, was durch Setzen des INTRQ-Ausgang und Löschen des BUSY-Statusbit angezeigt wird.

Ablaufdiagramm der Typ-I-Kommandos

Um den Ablauf der Typ-I-Kommandos deutlicher zu machen, liefern wir ihnen hierzu das entsprechende Ablauf-Diagramm.

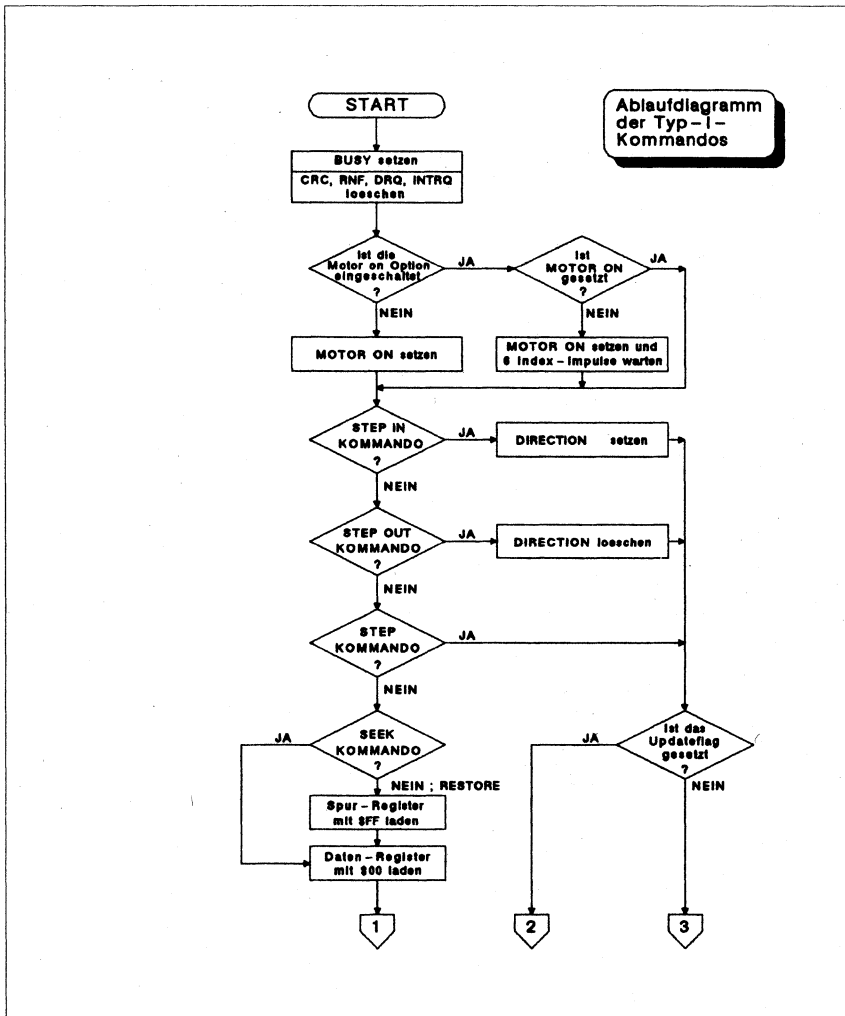


Abb. 14.2: Ablaufdiagramm der Typ-I-Kommandos I

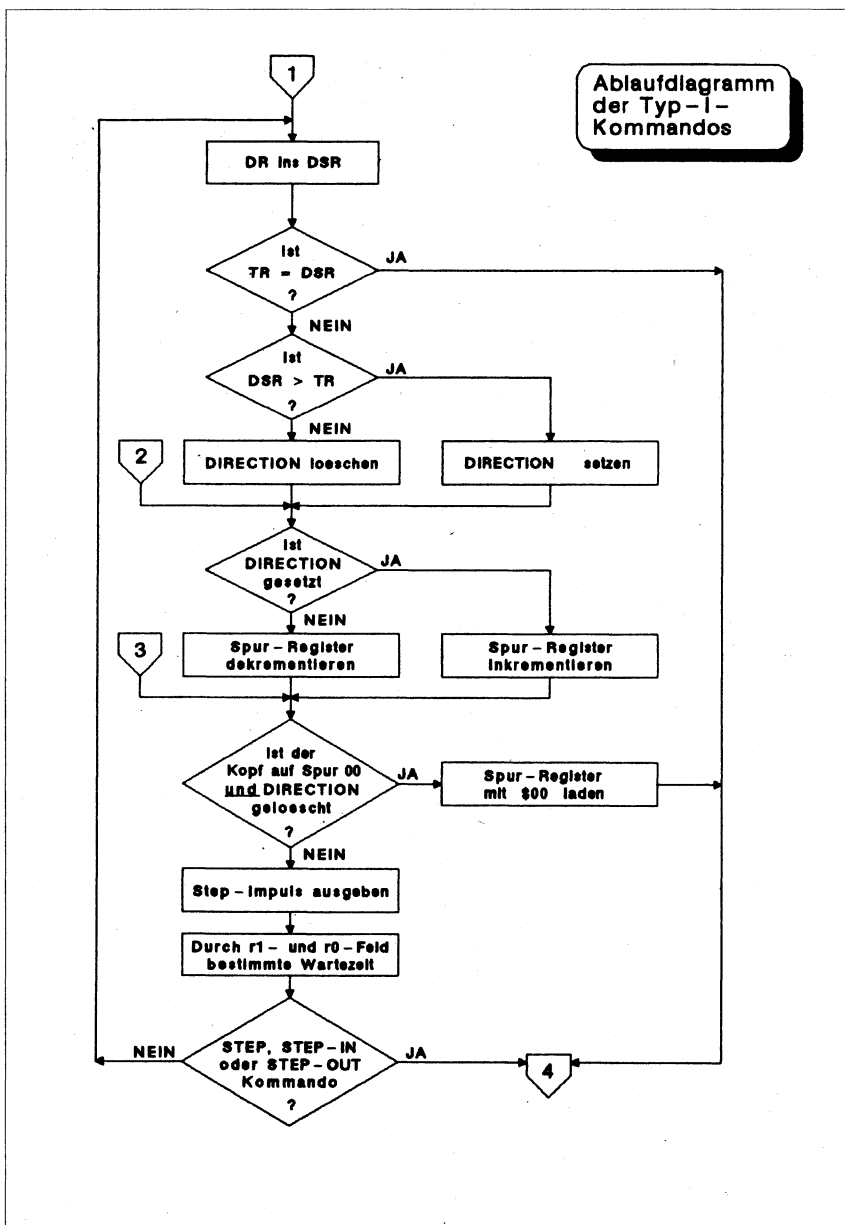


Abb. 14.3: Ablaufdiagramm der Typ-I-Kommandos II

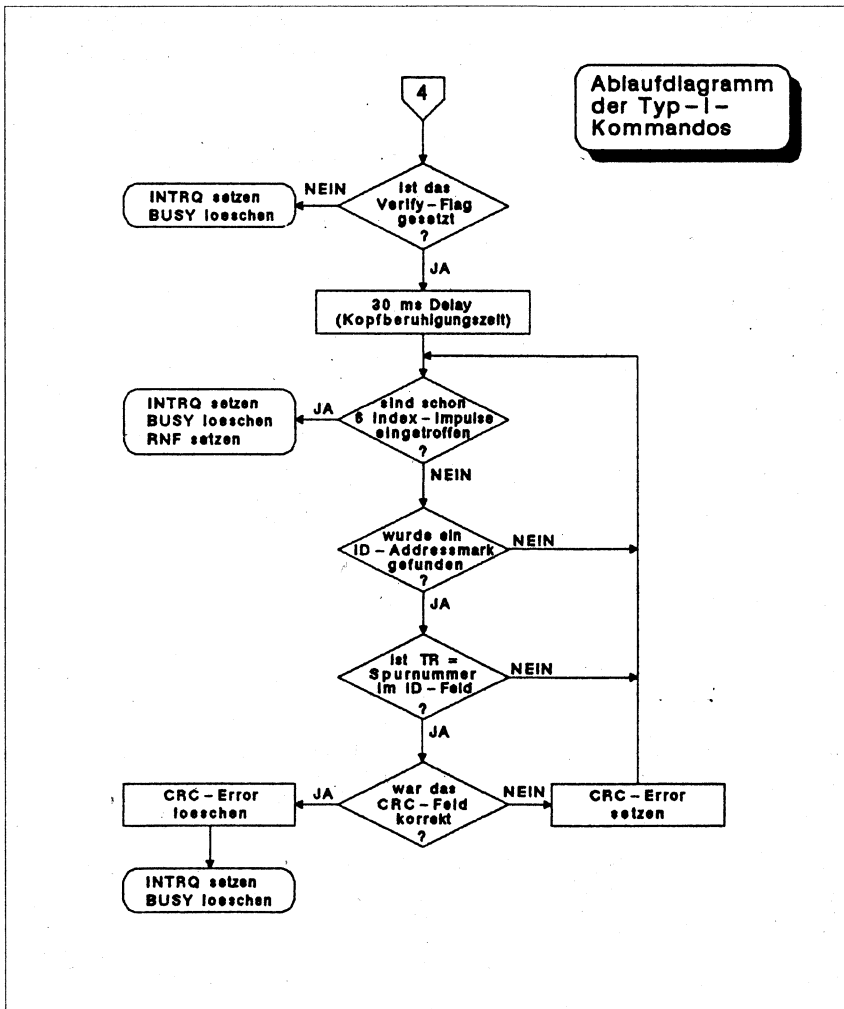


Abb. 14.4: Ablaufdiagramm der Typ-I-Kommandos III

Die Typ-II-Kommandos

Diese Gruppe ist für das Lesen und Schreiben von Sektoren, welche die logische Dateneinheit darstellen, zuständig. Der Datenaustausch wird deshalb ausschließlich mit diesen Kommandos abgewickelt.

Wie alle Kommandos, die für das Lesen und Schreiben von Informationen zuständig sind, nehmen auch diese immer auf die Spur Bezug, über der sich der Schreib-/Lese-Kopf gerade befindet.

Daher ist es Aufgabe des Programmierers, mit Hilfe der Typ-I-Kommandos dafür zu Sorgen, daß der Kopf vor Anwendung eines Schreib-/Lese-Befehls über der entsprechenden Spur positioniert wird.

READ SECTOR

Kommando-Wort:	7	6	5	4	3	2	1	0

	1	0	0	m	h	E	0	0

Dem FDC wird die Nummer des zu lesenden Sektors im Sektor-Register übergeben. Wird dann ein READ-SECTOR-Kommando gestartet, so liest der Controller ein ID-Feld und testet, ob dessen Spur- und Sektor-Nummer mit den Inhalten von Spur- und Sektor-Register übereinstimmen. Ist das nicht der Fall, so wird das nächste ID-Feld gelesen.

Waren sie jedoch gleich, so wird die Angabe der Sektor-Länge zwischengespeichert und die CRC-Prüfsumme des ID-Feldes mit der, aus den gelesenen Daten, berechneten verglichen. Sollten sich hier keine Abweichungen ergeben, dann wurde das, zu dem gewünschten Sektor "passende", ID-Feld gefunden; andernfalls wird ein weiteres ID-Feld gelesen.

Bevor nun das Lesen des Daten-Feldes beginnen kann, muß sich unter den nächsten 43 Byte ein DATA-ADDRESS-MARK (DAM) befinden. Wenn nicht, so wird (wie sollte es anders sein) wiederum ein ID-Feld gelesen.

Wurde nach fünf Umdrehungen kein "passendes" ID-Feld gefunden, dem nicht innerhalb von 43 Byte ein DAM folgt, so wird das RNF-Statusbit gesetzt und das Kommando abgebrochen.

Wie Sie sehen, müssen schon einige Bedingungen erfüllt sein, bevor der FDC ein Daten-Feld, in dem sich ja ein Sektor befindet, liest. Ist bis hierhin aber alles ordnungsgemäß verlaufen, dann geht's wie folgt weiter:

Die Art des DATA-ADDRESS-MARK ('normal' = 0; 'gelöscht' =1) wird in das Status-Bit 5 übertragen.

Der Address-Mark-Detector wird ausgeschaltet und die entsprechende Anzahl (aus der Angabe im Sektor-Längen-Feld berechnet) von Daten-Bytes gelesen.

Für jedes gelesene Byte wird ein DRQ ausgelöst. Sollte auf einen dieser DRQ's nicht reagiert werden, so setzt der FDC das LOST-DATA-Statusbit.

Nachdem alle Daten-Bytes gelesen sind, wird auch hier eine CRC-Prüfsumme getestet. Diese befindet sich in den beiden, dem Sektor folgenden, Byte und wird mit der, aus den gelesenen Sektor-Daten, berechneten verglichen. Bei Abweichung wird das CRC-ERROR-Statusbit gesetzt.

Das Ende der Operation wird durch Setzen des DRQ-Ausgangs und Löschen des BUSY-Statusbits angezeigt.

READ SECTOR (mit gesetztem m-Bit)

Wird im Kommandowort des READ-SECTOR-Befehls das m-Bit gesetzt, so wird der FDC versuchen, mehrere Sektoren (maximal alle der Spur) zu lesen. Die Nummer des ersten zu lesenden Sektors wird dem Controller im Sektor-Register übergeben. Der Ablauf des Kommandos ist zunächst mit dem zuvor beschriebenen identisch. Nachdem jedoch der Sektor gelesen wurde, erhöht der Controller automatisch das Sektor-Register und startet ein weiteres READ-SECTOR-Kommando.

Dies setzt sich so lange fort, bis kein weiterer Sektor mehr gefunden wird. Das heißt mit anderen Worten, daß dieses Kommando immer mit einem RECORD-NOT-FOUND-ERROR abgebrochen wird.

Da dem Controller nicht die Anzahl der zu lesenden Sektoren angegeben werden kann, muß es eine andere Möglichkeit geben, mehrere Sektoren zu lesen. Dieses erreicht man durch das FORCE-INTERRUPT-Kommando. Es wird also nicht darauf gewartet, bis der FDC das Kommando von alleine abbricht, sondern bestimmt diesen Moment selbst.

Wie, soll dieses Beispiel verdeutlichen:

Voraussetzung: Der Kopf befindet sich über einer beliebigen, nach ATARI-FORMAT formatierten Spur.

Die Sektoren dieser Spur sind demnach von 1-9 numeriert und wir möchten die Sektoren 3-7, also 5 Stück, lesen.

Der Sektor-Nr. 3 wird ins Sektor-Register übertragen, und die Lese-Operation durch Übergabe des Kommandowortes gestartet.

Wird nichts weiter unternommen, so würden jetzt sieben Sektoren (Sektor 3-9) gelesen und das Kommando nach fünf weiteren Umdrehungen mit RNF-

ERROR abgebrochen (da Sektor-Nr.10 nicht gefunden werden kann). Es sollen aber nur die Sektoren 3-7 (und zwar ohne Fehlermeldung) gelesen werden. Deshalb kommen wir zu folgendem Ablauf:

Da die Datenübertragung durch den DMA-Controller abgewickelt wird, wurde dieser ja vor dem Start des FDC-Kommandos mit einer Anfangs-DMA-Adresse initialisiert. Während der FDC die Daten an den DMA-Controller übergibt, wird diese Adresse fortlaufend erhöht. Die aktuelle DMA-Adresse kann daher durch Auslesen des DMA-Adressen-Registers in Erfahrung gebracht werden.

Dieses Register wird nun ständig abgefragt, bis dessen Inhalt gegenüber der Start-Adresse um \$A00 (5 * \$200) erhöht ist. Sobald das der Fall ist, wird das READ-SECTOR-Kommando durch einen FORCE-INTERRUPT-BEFEHL abgebrochen.

WRITE-SECTOR

Kommando-Wort:	7	6	5	4	3	2	1	0

	1	0	1	m	h	E	P	a0

Wir möchten hierzu nur die Unterschiede gegenüber dem READ-SECTOR-Kommando beschreiben, da der größte Teil des Ablaufs mit diesem übereinstimmt.

Der FDC testet zu Anfang des Kommandos, ob der WRITE-PROTECT-Eingang auf LOW liegt. Sollte das der Fall sein (schreibgeschützte Diskette), so wird das WPRT-Statusbit gesetzt und das Kommando abgebrochen.

Ist die Diskette nicht schreibgeschützt, dann wird mit der ID-Feld-Suche begonnen.

Wurde das "passende" ID-Feld gefunden, wird ein Delay, 23 Byte entsprechend, eingelegt. Danach werden zwölf "Null-Bytes" und ein DATA-ADDRESS-MARK (die Art ist vom a0-Bit abhängig) geschrieben. Es folgt die eigentliche Sektor-Information, der sich die CRC-Prüfsumme anschließt. Zu guter Letzt wird noch ein "\$FF-Byte" geschrieben. Ob der Sektor richtig geschrieben wurde, kann nur durch ein READ-SECTOR-Kommando festgestellt werden.

WRITE-SECTOR (mit gesetztem m-Bit)

Es gilt hier analog das gleiche wie bei den Ausführungen zum READ-SECTOR-Kommando mit gesetztem m-Bit.

Ablaufdiagramm der Typ-2-Kommandos

Auch für die Typ-2-Kommandos haben wir ein Ablauf-Diagramm anzubieten.

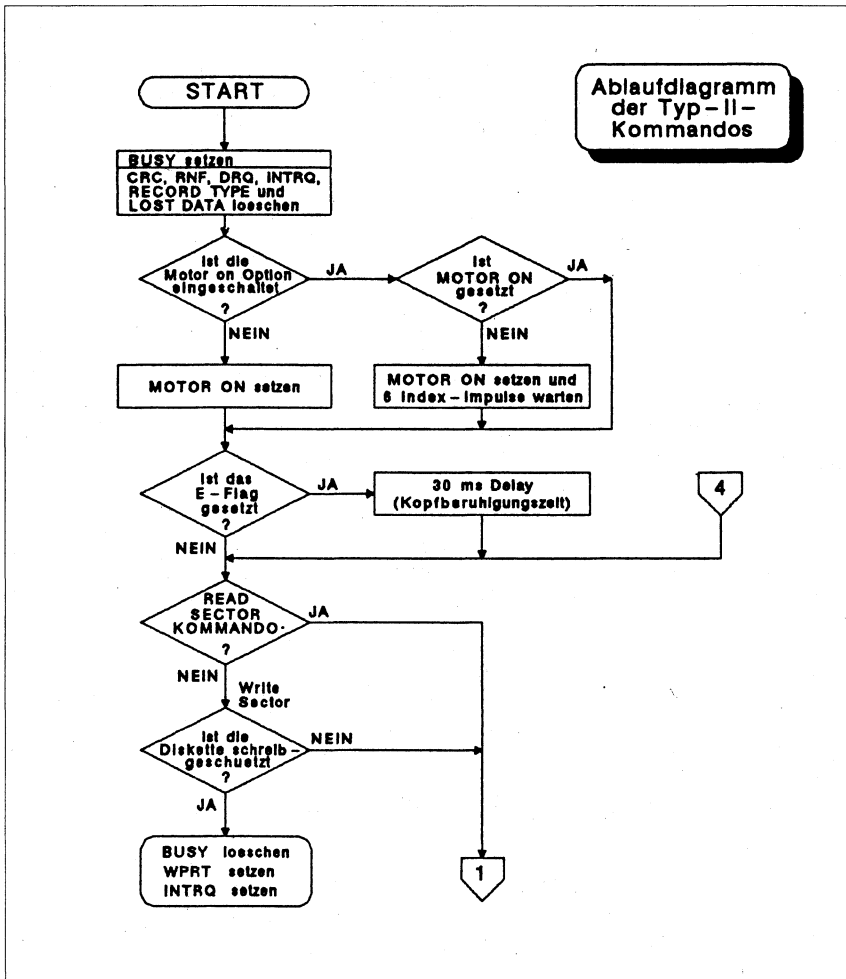


Abb. 14.5: Ablaufdiagramm der Typ-II-Kommandos I

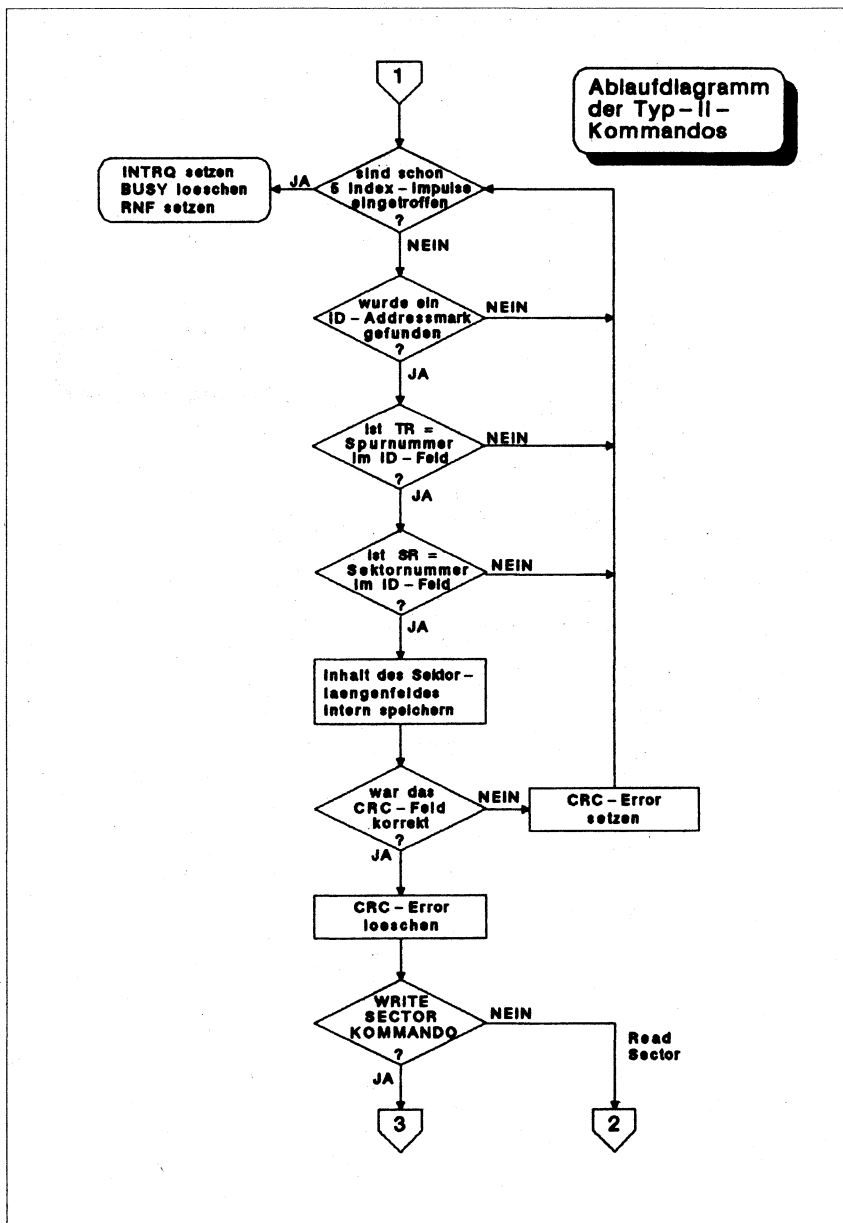


Abb. 14.6: Ablaufdiagramm der Typ-II-Kommandos II

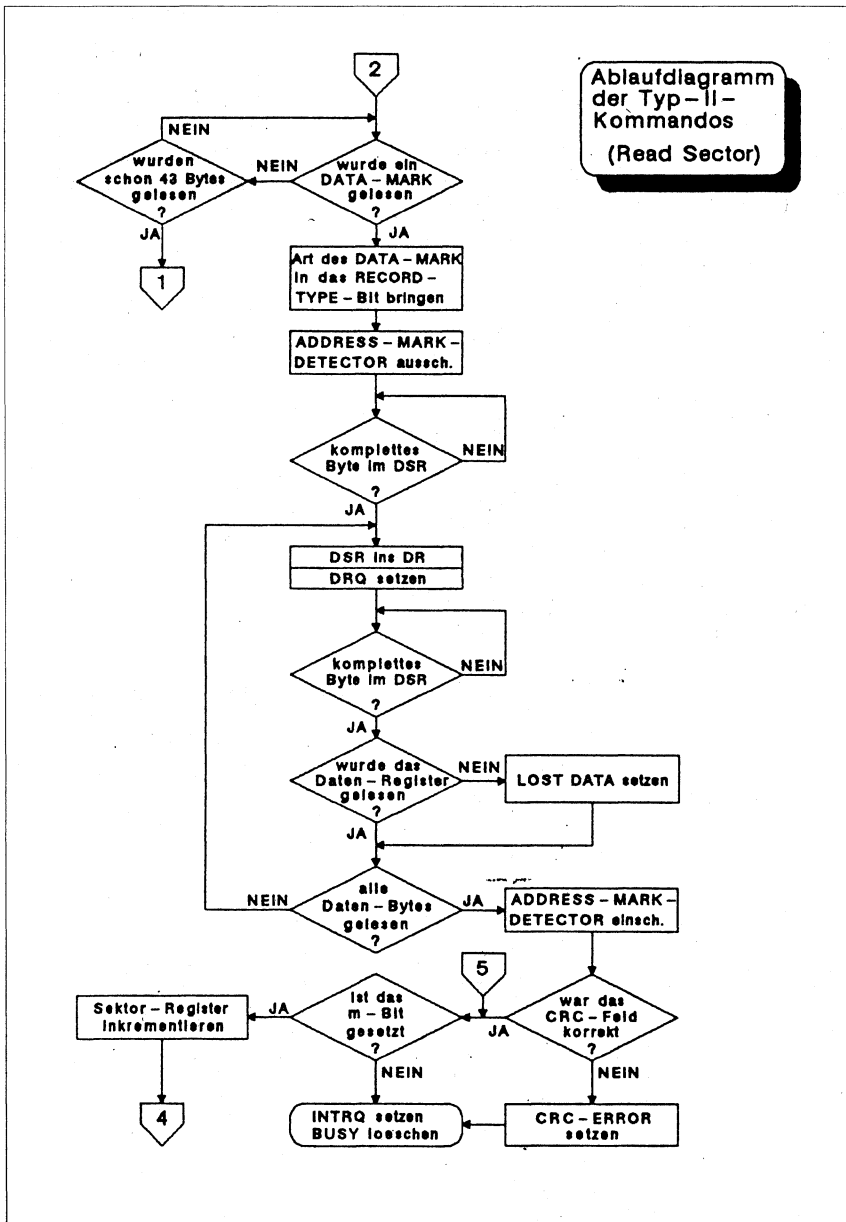


Abb. 14.7: Ablaufdiagramm der Typ-II-Kommandos III

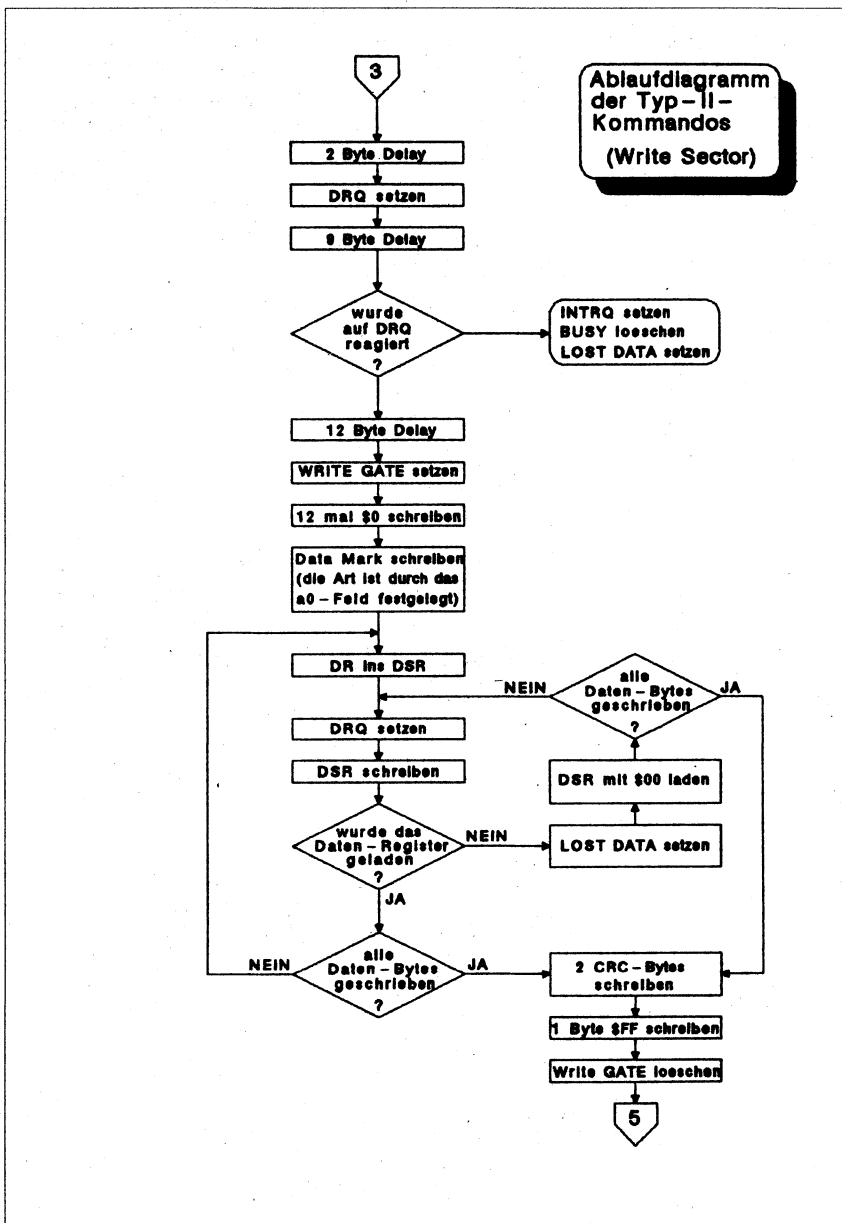


Abb. 14.8: Ablaufdiagramm der Typ-II-Kommandos IV

Die Typ-III-Kommandos

Der in dieser Gruppe enthaltene WRITE-TRACK-Befehl dient dem Formatieren einer Spur, während mittels READ-TRACK und READ-ADDRESS das Format einer Spur analysiert werden kann.

READ ADDRESS

Kommando-Wort:	7	6	5	4	3	2	1	0

	1	1	0	0	h	E	0	0

Zu diesem Kommando müssen unbedingt ein paar erklärende Worte, die den DMA-Controller betreffen, vorausgeschickt werden. Es könnte sonst leicht passieren, daß Sie bei der Programmierung des READ-ADDRESS-Befehls an den Rand des Wahnsinns getrieben werden.

Wird das READ-ADDRESS-Kommando gestartet, so stellt man nach dessen Ausführung nämlich erstaunt fest, daß sich das ID-Feld nicht im RAM befindet.

Ein Test der Status-Register (DMA und FDC) wird allerdings keinen Fehler anzeigen. Wenn man nun die Start-DMA-Adresse von der End-DMA-Adresse subtrahiert, dann ergibt die Differenz Null. Es sind also nur 0 Bytes übertragen worden. Das ist natürlich etwas wenig.

Wo sind also die sechs Byte des ID-Feldes geblieben? Nun, ganz einfach, im DMA-Controller! Dieser überträgt nämlich die Bytes nicht einzeln, sondern wartet, bis er 16 Byte erhalten hat. Erst dann stellt er eine Bus-Anforderung an den 68000-Prozessor, um diese 16 Byte in das RAM übertragen zu können.

Wie kommt man nun an das gelesene ID-Feld, welches sich ja noch im DMA-Controller befindet? Hier gibt es nur eine Möglichkeit. Es müssen weitere Daten gelesen werden. Dies wird dadurch erreicht, indem mehrere ID-Felder, hintereinander, gelesen werden.

Nach z.B. drei READ-ADDRESS-Befehlen (18 Byte) befinden sich dann 16 Byte im RAM und zwei Byte im DMA-Controller. Damit alle gelesenen Bytes ins RAM übertragen werden, muß deren Anzahl demzufolge ein ganzzahliges Vielfaches von 16 sein.

Es gibt aber noch einen weiteren Fallstrick. Dieser besteht im Löschen des DMA- Statusregisters, welches durch "toggeln", also dem Ein- und Ausschalten, der Schreib-/Lese-Leitung erreicht wird. Wer, vor jedem DMA-Transfer, "sicherheitshalber" dieses Register zurücksetzt, erlebt die nächste Überraschung.

schung. Hierdurch wird nämlich nicht nur das Status-Register gelöscht, sondern auch alle Bytes, die sich noch im DMA-Controller befinden. In diesem Fall könnten so lange ID-Felder gelesen werden, bis die Diskette durchgeschliffen ist. Der DMA-Controller würde nicht ein einziges Byte in den Speicher übertragen. Das DMA-Statusregister darf also nur vor dem ersten READ-ADDRESS-Befehl gelöscht werden.

Doch nun zum READ-ADDRESS-Kommando selbst. Durch dieses Kommando wird jeweils das nächste ID-Feld, das der Schreib-/Lese-Kopf erreicht, gelesen. Es kann in Verbindung mit dem READ-TRACK-Kommando dazu benutzt werden, das Format einer Spur zu analysieren. Ferner ist es auch möglich, ein Spur-Verify vorzunehmen, ohne die Spur zu verlassen.

Das READ-ADDRESS-Kommando liest ein ID-Feld, ohne zu testen, ob ein zugehöriges Daten-Feld existiert. Es werden dabei sechs Byte gelesen, die folgende Bedeutung haben:

Byte-Nr.	Bedeutung
1	Spur-Nummer
2	Seiten-Nummer
3	Sektor-Nummer
4	Sektorgroße
5	CRC-Byte 1
6	CRC-Byte 2

Die Spur-Nummer (Byte 1) wird außerdem in das Sektor-Register geschrieben. Man kann dadurch also ein Spur-Verify vornehmen, ohne die gelesenen Daten zu verwenden.

Dies erscheint im ersten Moment als unnötig, da es ja keine Rolle spielt, ob man für ein Spur-Verify den Inhalt des Sektor-Registers oder das erste übertragene Byte, mit dem Inhalt des Spur-Registers, vergleicht. Außerdem ist es einfacher, das übertragene Byte zu verwenden, da in diesem Fall nicht erst das Sektor-Register selektiert werden muß.

Nach einem einzelnen READ-ADDRESS-Befehl wird jedoch noch keine Information ins RAM übertragen, da, wie zuvor schon beschrieben, der Speicher-Transfer erst dann beginnt, wenn der DMA-Controller 16 Byte "gesammelt" hat. Doch dadurch, daß der FDC das erste Byte des ID-Feldes zusätzlich noch in das Sektor-Register schreibt, hat man trotzdem eine Möglichkeit, durch ein einziges READ-ADDRESS-Kommando, ein Spur-Verify durchzuführen.

Bei dem Lesen des ID-Feldes wird eine Prüfsumme gebildet, die mit der Aufgezeichneten (CRC-Byte 1 und 2) verglichen wird. Sollten diese nicht identisch sein, wird das CRC-ERROR-Statusbit gesetzt.

Ist bis zum Eintreffen von sechs Index-Impulsen (das entspricht mindestens fünf Umdrehungen), kein ID-Feld gefunden worden, so wird das RNF-Statusbit (Record not found) gesetzt.

Hat der Controller das Kommando abgearbeitet, so zeigt er dies durch Setzen des INTRQ-Ausgangs und Löschen des BUSY-Statusbits an.

READ TRACK

Kommando-Wort:	7	6	5	4	3	2	1	0

	1	1	1	0	h	E	0	0

Auch das READ-TRACK-Kommando dient nur der Spur-Diagnose. Es wird eine komplette Spur, inklusiv aller GAP-, SYNC- und Daten-Bytes, gelesen.

Das Lesen beginnt mit der steigenden Flanke des nächsten Index-Impulses, den der Controller vom Laufwerk erhält. Es werden so lange Daten gelesen, bis ein weiterer Index-Impuls den Controller erreicht. Das Ende der Operation wird, wie üblich, durch Setzen des INTRQ-Ausgangs und Löschen des BUSY-Statusbits angezeigt.

Für jedes gelesene Byte wird ein DRQ erzeugt. Wie bei allen Kommandos, die Daten transferieren, wird auch hier das LOST-DATA-Statusbit gesetzt, wenn nicht auf den DRQ reagiert wird.

Bezüglich des LOST-DATA-Bits stellten wir fest, daß es des öfteren, aus uns unbekannten Gründen, gesetzt wird. Das hat zur Folge, daß dieses Bit nach einem READ-TRACK-Kommando keinen Aufschluß darüber gibt, ob tatsächlich Daten verloren gegangen sind. Merkwürdigerweise traten solche Fälle bei READ-TRACK-Versuchen an einer unformatierten Diskette nie auf.

Mit jedem empfangenen Address-Mark wird das "Sammeln" der Daten-Bits synchronisiert. Der ADDRESS-MARK-DETECTOR, der hierfür zuständig ist, wird danach aber nicht (wie z.B. bei einem READ-SECTOR-Kommando) ausgeschaltet, sondern bleibt während des gesamten Lesevorgangs aktiviert. Er ist ständig auf der Suche nach einem Address-Mark und verursacht dadurch Lesefehler.

Nach Herstellerangaben sollen alle Informationen, mit Ausnahme der GAP-Bytes, korrekt gelesen werden. Unsere Versuche haben allerdings andere Ergebnisse erbracht. In der Praxis ist es so, daß nur die ID-Felder richtig gelesen werden. Aber schon bei der CRC-Prüfsumme der ID-Felder treten zeitweilig Lesefehler auf.

Es erhebt sich die Frage, wozu das READ-TRACK-Kommando eigentlich verwendet werden kann. Die Daten selbst sind ja durch mögliche Lesefehler nur von zweifelhaftem Wert und korrekt gelesene ID-Felder erhält man durch das READ-ADDRESS-Kommando wesentlich einfacher, denn die Suche nach einem ID-Feld in der gesamten Spur-Information wirft ein weiteres Problem auf. So läßt sich bei einer Byte-Folge von FE-01-00-01-02-BC-DB nicht erkennen, ob es sich hier tatsächlich um ein ID-Feld handelt. Diese Byte-Folge könnte ja auch in einem Daten-Feld vorkommen. Auch wenn kein ID-Feld existiert, könnte Sie zufällig auf der Diskette stehen. Ein "echtes" ID-Feld ist nur ein solches, welches der Controller als ID-Feld erkennt.

Der READ-TRACK-Befehl ist, alleine angewandt, nicht sehr sinnvoll. Im Zusammenwirken mit dem READ-ADDRESS-Kommando läßt sich eine Spur jedoch ziemlich genau analysieren. Wenn die Daten selbst zum größten Teil keine Aussagekraft besitzen, so ist deren Anzahl jedoch von großer Bedeutung. Man kann dadurch die Abstände zwischen "markanten" Punkten, was für die Spuranalyse wichtig ist, sehr genau berechnen. Hierdurch wird der Nachteil des READ-ADDRESS-Kommandos, welches nur ID-Felder liest, aber nicht testet ob ein zugehöriges Daten-Feld existiert, ausgeglichen.

Als Beispiel möchten wir im Ansatz den Ablauf einer Spuranalyse schildern:

- a) Alle ID-Felder der Spur werden durch READ-ADDRESS-Kommandos gelesen.
- b) Die gesamte Spur-Information wird mit einem READ-TRACK-Befehl gelesen.
- c) Alle Sektoren der Spur (Spur- und Sektor-Nummer erhält man aus den gelesenen ID-Feldern) werden durch READ-SECTOR-Kommandos gelesen.

Sind in der Spur keine ID-Felder enthalten, so ist unsere Analyse schon beendet, da wir es dann mit einem unlesbaren Format zu tun haben.

Es wird nun in der Spur-Information nach dem ersten ID-Feld gesucht. Hier muß man sich an den eben erwähnten "markanten" Punkten orientieren.

Der erste dieser Punkte ist eine Byte-Folge von \$A1,\$FE oder \$C2,\$FE, also SYNC-Byte und ID-ADDRESS-MARK. Nur einem solchen Punkt kann ein ID-Feld folgen.

Wurde es gefunden, so wird der zweite "markante" Punkt interessant. Dieser befindet sich max. 42 Byte hinter dem ID-Feld. Es muß hier ein SYNC-Byte, gefolgt von einem DATA-ADDRESS-MARK, gefunden werden. Falls nicht, so existiert für das ID-Feld kein gültiges Daten-Feld.

Ein weiterer Test betrifft die Plausibilität des ID-Feldes. Zeigt dieses z.B. eine Sektor-Größe von 512 Byte an und das nächste ID-Feld folgt in einem Abstand von 200 Byte, so ist hier etwas im Argen.

Das Lesen der Sektoren erfolgt in diesem Zusammenhang aus zwei Gründen. Zum einen kann die Sektor-Information (wegen der schon beschriebenen Datenverfälschungen) nicht den durch READ-TRACK erhaltenen Daten entnommen werden. Zum anderen ist nur durch READ-SECTOR eine Kontrolle der CRC-Prüfsumme des Datenfeldes möglich.

Auf diese oder ähnliche Weise wird der gesamte Spur-Inhalt analysiert. Die Auswertung der daraus erhaltenen Informationen gibt die Möglichkeit, die soeben analysierte Spur zu reproduzieren, bzw. die Erkenntnis, daß sich die Spur, mit den Möglichkeiten des WD 1772, nicht reproduzieren läßt.

WRITE TRACK (FORMAT TRACK)

Kommando-Wort:	7	6	5	4	3	2	1	0

	1	1	1	1	h	E	P	0

Bevor eine Diskette zur Datenspeicherung benutzt werden kann, muß diese als erstes formatiert werden. Doch was geschieht dabei eigentlich? Die Sache ist im Grunde genommen sehr leicht zu erklären.

Die logische Dateneinheit, mit der ein Datentransfer zwischen Laufwerk und Controller stattfindet, ist der Sektor. Da sich auf einer neuen Diskette aber keine Informationen über den Startpunkt eines Sektors befinden, muß diese erst mit solchen Startpunkten versehen werden.

Zu jedem Sektor gehört ein Feld, welches Informationen über diesen beinhaltet. Ferner sind die Daten durch Prüfsummen gesichert. Auch das fehlt auf einer neuen Diskette und muß zuvor auf diese geschrieben werden. Synchronisations-Bytes sind dort natürlich auch nicht zu finden. Diese sind jedoch emi-

nent wichtig, um den Anfang eines Bytes, welcher sich, bei späteren Leseoperationen, irgendwo im seriellen Bit-Strom "versteckt", zu erkennen.

Ziel des Formatierens ist es nun, alle hier aufgeführten Informationen bzw. Markierungen auf die Diskette zu schreiben. Dies muß jedoch nach bestimmten Regeln geschehen, da später sonst kein Sektor-Transfer möglich ist.

Hält man sich an diese Regeln, so lassen sich zahlreiche - auch vom Standard abweichende - einwandfrei funktionierende, Formate erzeugen.

Dem FDC wird grundsätzlich ein Daten-Byte, also ein Wert zwischen \$00 und \$FF, übergeben. Um aber die erforderlichen Markierungen, die sich von "normalen" Daten-Bytes unterscheiden, auf die Diskette zu schreiben, muß es eine Möglichkeit geben, den Controller so zu steuern, daß nicht ein Datenbyte, sondern eine Markierung auf die Diskette geschrieben wird. Wir möchten deshalb zunächst die Steuer-Bytes behandeln, welche durch Werte zwischen \$F5 und \$FF repräsentiert werden.

Im Gegensatz zu den READ-SECTOR- bzw. WRITE-SECTOR-Kommandos, bei welchen diese als "normale" Daten-Bytes geschrieben werden, veranlassen sie den Controller bei einem WRITE-TRACK-Kommando, die Diskette mit besonderen Markierungen zu versehen. Gemeinsam für diese gilt, daß sie ohne Takt-Impulse geschrieben werden und daher, bei späteren Lese-Operationen, von Daten-Bytes, die eventuell den gleichen Wert besitzen, unterschieden werden können (siehe "Address-Mark-Detector"). Was diese Steuer-Bytes, bei einem WRITE-TRACK-Befehl, im einzelnen bewirken, bzw. welche Bedeutung sie dadurch bei späteren Lese-Operationen erlangen, zeigt die folgende Übersicht:

an den FDC übergebenes Byte	vom FDC geschriebene(s) Byte(s)	Bedeutung
\$F5	\$A1	Sync-Byte, CRC-Reg. löschen
\$F6	\$C2	Sync-Byte
\$F7	\$XX,\$XX	2 CRC-Bytes
\$F8	\$F8	"gelöschtes" Data-Address-Mark
\$F9	\$F9	Data-Mark
\$FA	\$FA	Data-Mark
\$FB	\$FB	"normales" Data-Address-Mark
\$FC	\$FC	Data-Mark
\$FD	\$FD	Data-Mark
\$FE	\$FE	Index-Address-Mark
\$FF	\$FF	

Es soll nun erklärt werden, wie mit Hilfe dieser Steuer-Bytes ein beliebiges Format erzeugt werden kann. Da hierzu einige Beispielwerte erforderlich sind, werden jeweils solche benutzt, die zur Erstellung des ATARI-Formates nötig sind. An den entsprechenden Stellen wird darauf hingewiesen, in welchem Maße Abweichungen zulässig sind.

Beginnen wir mit einem Puffer, der alle Informationen einer kompletten Spur, die später durch das WRITE-TRACK-Kommando auf die Diskette geschrieben wird, aufnehmen kann. Dazu sollte dessen Größe mindestens 6250 Byte betragen.

Es gilt nun, diesen Puffer derart mit Daten zu füllen, daß er einem Format entspricht, welches neun Sektoren von je 512 Byte Länge aufnehmen kann.

Teilen wir unseren Puffer in zwei verschiedene Komponenten auf, so erhalten wir folgendes Schema, welches für alle Formate Gültigkeit besitzt. Unterschiede in der Anzahl der Records sind natürlich möglich.

| GAP1 | RECORD 1 | RECORD 2 |//.....| RECORD 9 | GAP5 |

Eine Spur beginnt und endet mit einem, GAP bezeichneten, Block. Wie wir später noch sehen werden, sind diese GAP's auch in den RECORD's vorhanden. Ein GAP ist ein Zwischenraum, der die einzelnen Komponenten in der Spur trennt. Es enthält keine Nutzinformation, sondern nur Füllbytes bzw., wenn ein GAP vor einem ID- oder Daten-Feld steht, auch SYNC-Bytes. Dem FDC wird dadurch eine, durch die Länge des GAP's bestimmte, Zeit eingeräumt, in der er seine Funktionsteile, auf die Erfordernisse der nachfolgenden Komponente, einstellen kann.

Bezeichnung	Wert	Gap-Länge	Gap-Länge
		ATARI-Format	Fremd-Format
GAP1 (Spur-Vorspann)	\$4E	60 Byte	min. 32 Byte
GAP5 (Spur-Nachspann)	\$4E	ca.664 Byte	min. 16 Byte

Die Länge von GAP5 ist im Moment irrelevant. GAP5 ist, einfach formuliert, das, was übrig bleibt. Bei unseren Berechnungen der Puffer-Aufteilung, müssen wir lediglich darauf achten, daß wenigstens 16 Byte für GAP5 vorhanden sind.

Subtrahieren wir von unserem Puffer die Anzahl der Bytes, die wir für GAP1 reservieren, so stehen uns noch 6190 Byte zur Verfügung, um sie auf die An-

zahl der Records aufzuteilen. Diese Aufteilung läßt sich jetzt noch nicht vornehmen, weil die Länge eines Records noch nicht bekannt ist.

Wie Sie wahrscheinlich schon richtig vermutet haben, ist in jedem Record einer unserer neun Sektoren enthalten. Deshalb steht eines fest; die Record-Länge ist in jedem Fall größer als die Sektorlänge. Wenn wir einen Record weiter aufschlüsseln, ergibt sich folgendes Bild:

| GAP2 | INDEX-FELD | GAP3 | DATEN-FELD | GAP4 |

Hier sind zunächst die bereits angekündigten GAP's, die in der gezeigten Reihenfolge mit "Pre-Record"-, "Inter-Record"- und "Post-Record"-Gap bezeichnet werden.

Bezeichnung	Wert	Gap-Länge ATARI-Format	Gap-Länge Fremd-Format
GAP2	\$00	12 Byte	min. 8 Byte
SYNC	\$F5	3 Byte	3 Byte
GAP3	\$4E	22 Byte	22 Byte
	\$00	12 Byte	12 Byte
SYNC	\$F5	3 Byte	3 Byte
GAP4	\$4E	40 Byte	min. 24 Byte

Summe der GAP-Bytes pro RECORD:

92 Byte

min. 72 Byte

Die Synchronisations-Bytes (\$F5) in Gap2 und Gap3 sorgen dafür, daß das Lesen, der seriell eintreffenden Daten-Bits, mit dem Byte-Anfang geschieht. Ferner haben sie die Aufgabe, den FDC auf ein folgendes ADDRESS-MARK aufmerksam zu machen und dessen CRC-Logik zu initialisieren. Um ein SYNC zu schreiben wird dem FDC der Wert \$F5 übergeben, der daraufhin ein "\$A1-Byte" ohne Takt-Impulse schreibt.

Das Daten-Feld

Die GAPs, die sich in einem RECORD befinden, sind nun aufgeschlüsselt. Sehen wir uns nun das Daten-Feld, in dem sich ja unser Sektor befindet, genauer an.

| DAM | Sektor | CRC | | | |
 | \$FB | 512 Datenbyte | \$F7 |

Das Daten-Feld beginnt mit einem DATA-ADDRESS-MARK, welches den Start des Sektors kennzeichnet. Der Wert \$FB wird von einem späteren READ-SECTOR-Kommando als ein "normales" DATA-ADDRESS-MARK interpretiert, wogegen der Wert \$F8, der statt \$FB eingetragen werden kann, als ein "gelöschtes" DAM betrachtet wird.

Das Sektor-Feld wird beim Formatieren mit "Dummy-Bytes" gefüllt. Die Werte können zwar frei bestimmt werden, sollten aber keinesfalls größer als \$F4 sein. Die Anzahl kann 128, 256, 512 oder 1024 betragen. Wie der FDC die unterschiedlichen Sektor-Längen erkennt, wird im "ID-Feld" erklärt. Durch Übergabe des Wertes "\$F7" veranlaßt, schreibt der FDC den Inhalt seines 16-Bit-CRC-Registers, welches eine Prüfsumme enthält, auf die Diskette. Obwohl nur ein Byte übergeben wird, werden vom Controller zwei Byte geschrieben. Die Gesamtlänge des Datenfeldes beträgt bei einer Sektorgröße von 512 Bytes, wie auch in unserem Beispiel, 515 Byte.

Das Index-Feld

Das Index-Feld oder kürzer ID-Feld, enthält Informationen über das nachfolgende Datenfeld.

ID-AM	Spur	Seite	Sektor	Länge	CRC
\$FE	00-79	00-01	01-09	00-03	\$F7

Das Index-Address-Mark (ID-AM) ist die Startmarkierung des ID-Feldes. Trifft der Controller, bei späteren Lese-Operationen, auf ein ID-AM, so wird er die sechs folgenden Byte bei ausgeschaltetem Address-Mark-Detector lesen. Das ID-AM wird ebenfalls ohne Takt-Impulse geschrieben. Die drei Byte, die sich dem ID-AM anschließen, beschreiben die Lage des RECORDS. Hier wird zuerst die Spur-Nummer, auf der er sich befindet, angegeben. In unserem Fall ein Wert zwischen 0 und 79, abhängig davon, welche Spur formatiert wird.

Das Seiten-Feld gibt an, ob sich der Record auf der Vorder-oder auf der Rückseite befindet. Dieses Byte wird, vom Controller selbst, in keiner Weise bei irgendwelchen Operationen benutzt. Das Sektor-Feld schließlich enthält die Nummer des Sektors (1-9).

Da der FDC unterschiedliche Sektorgrößen unterstützt, muß ihm mitgeteilt werden, wieviele Daten-Byte im folgenden Sektor enthalten sind. Dies geschieht durch die Angabe im Längen-Feld.

Tabelle der Sektorlängen

Längen-Feld	Byte pro Sektor
00	128
01	256
02	512
03	1024

Für eine Sektorgröße von 512 Byte steht in diesem Feld also eine "02". Es fehlt nun nur noch die Prüfsumme. Das Schreiben dieser Summe wird, genau wie im Daten-Feld, durch Übergabe des Wertes "\$F7" erreicht. In der Addition ergibt sich, daß die Länge eines ID-Feldes immer sieben Byte beträgt. Nachdem jetzt alle Komponenten und deren Anordnung innerhalb der Spur erklärt wurden, sind wir in der Lage, die Record-Länge zu berechnen.

Daten-Feld		515 Byte
ID-Feld	+	7 Byte
GAP2-GAP4	+	92 Byte

Record-Länge = 614 Byte

Dies ist die tatsächliche Größe des Records. In unserem Puffer beträgt dessen Länge nur 612 Byte, da für das Schreiben der beiden Prüfsummen (vier Byte), jeweils nur ein Byte an den FDC übergeben wird. In unserer Berechnung ist jedoch der Platz entscheidend, den ein Record auf der Spur einnimmt, also 614 Byte. Für neun Records benötigen wir also $9 * 614 \text{ Byte} = 5526 \text{ Byte}$. Subtrahieren wir diese von den 6190 zur Verfügung stehenden Byte, so bleiben noch 664 Byte für den Spur-Nachspann (GAP5) übrig. Das ist mehr als ausreichend, wenn man bedenkt, daß hierfür nur 16 Byte erforderlich sind. Selbst ein Format, welches zehn Sektoren à 512 Byte benutzt, würde die Länge von GAP5 mit 50 Byte nicht unter das Mindestmaß sinken lassen. Sehen wir uns zum Schluß an, mit welchen Daten unser Puffer aufbereitet wird. Zu der Tabelle, die wir hierfür angelegt haben, sind noch einige Erläuterungen nötig:

Die Daten von GAP2 bis einschließlich GAP4 (ein vollständiger Record) wiederholen sich für jeden Sektor. So muß z.B. bei dem Format mit 29 Sektoren, der entsprechende Block, 29mal hintereinander in den Puffer geschrieben werden. Die mit "\$XX" angegebenen Werte müssen von Ihnen selbst bestimmt werden, was aber nicht sonderlich kompliziert ist. Wird z.B. die Spur 54 formatiert, so wird als Wert für die Spur-Nr. jeweils eine "54" eingetragen. Ein Wert für die Seiten-Nr. ist im allgemeinen "0" für die Vorderseite und "1" für die Rückseite. Die Sektor-Nr. wird fortlaufend, normalerweise mit "1" be-

ginnend, vergeben. Die Reihenfolge ist variabel und könnte, bei einem Format mit neun Sektoren, z.B. 3,6,9,14,7,2,5,8 lauten. Wichtig ist nur, daß die Folge vollständig ist. Werden die Sektoren z.B. mit 1,2,3,5,6,7,8,9,10 bezeichnet, so wird ein späterer READ-SECTOR- bzw. WRITE-SECTOR-Befehl mit gesetztem m-Bit, nach dem dritten Sektor mit RECORD-NOT-FOUND-Error abgebrochen, da der FDC keinen Sektor mit der Nr.4 finden kann.

Das ATARI-FORMAT ist in der erste Spalte aufgeführt. Wer einen Blick in "A Hitchhiker's Guide to the BIOS" wirft, wird in einem Punkt eine Abweichung zu unserer Tabelle entdecken. Die Länge des Spur-Nachspanns (GAP5) ist nicht mit 664 Byte, sondern mit 1401 Byte angegeben. Wenn man alle dort angegebenen Byte addiert, so hat es den Anschein, daß die Spur-Länge ca. 7000 Byte beträgt, was aber nicht der Fall ist. Vielmehr wird nur ein Puffer aufbereitet, der eben etwas größer ist als die Spur tatsächlich aufnehmen kann. Aber mehr als ca. 6250 Byte passen nicht in eine Spur. Selbst eine Puffer-Größe von 50 KByte ändert diese Tatsache nicht.

	Anzahl der Sektoren / Sektorgroße			
	9 / 512	18 / 256	29 / 128	5 / 1024
GAP1	60 * \$4E	42 * \$4E	40 * \$4E	60 * \$4E
GAP2	12 * \$00	11 * \$00	10 * \$00	40 * \$00
SYNC	3 * \$F5	3 * \$F5	3 * \$F5	3 * \$F5
ID-AM	1 * \$FE	1 * \$FE	1 * \$FE	1 * \$FE
Spur-Nr.	1 * \$XX	1 * \$XX	1 * \$XX	1 * \$XX
Seiten-Nr.	1 * \$XX	1 * \$XX	1 * \$XX	1 * \$XX
Sektor-Nr.	1 * \$XX	1 * \$XX	1 * \$XX	1 * \$XX
Sektor-Länge	1 * \$02	1 * \$01	1 * \$00	1 * \$03
ID-CRC	1 * \$F7	1 * \$F7	1 * \$F7	1 * \$F7
GAP3	22 * \$4E	22 * \$4E	22 * \$4E	22 * \$4E
	12 * \$00	12 * \$00	12 * \$00	12 * \$00
SYNC	3 * \$F5	3 * \$F5	3 * \$F5	3 * \$F5
DAM	1 * \$FB	1 * \$FB	1 * \$FB	1 * \$FB
Daten	512 * \$E5	256 * \$E5	128 * \$E5	1024 * \$E5
DATA-CRC	1 * \$F7	1 * \$F7	1 * \$F7	1 * \$F7
GAP4	40 * \$4E	26 * \$4E	25 * \$4E	40 * \$4E
GAP5	664 * \$4E	34 * \$4E	33 * \$4E	420 * \$4E

Ablauf-Diagramm der Typ-3-Kommandos

Natürlich haben wir auch für die Typ-3-Kommandos ein Ablauf-Diagramm erstellt.

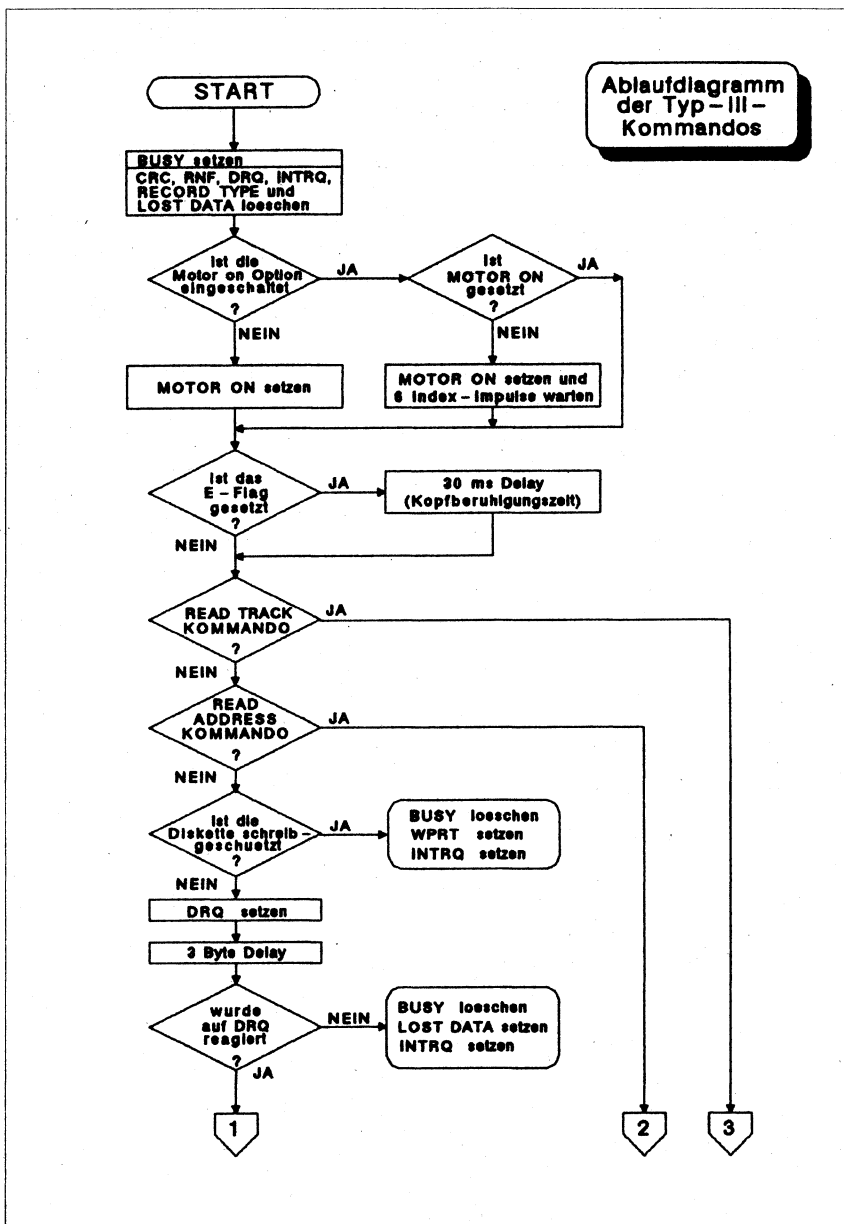


Abb. 14.9: Ablaufdiagramm der Typ-III-Kommandos I

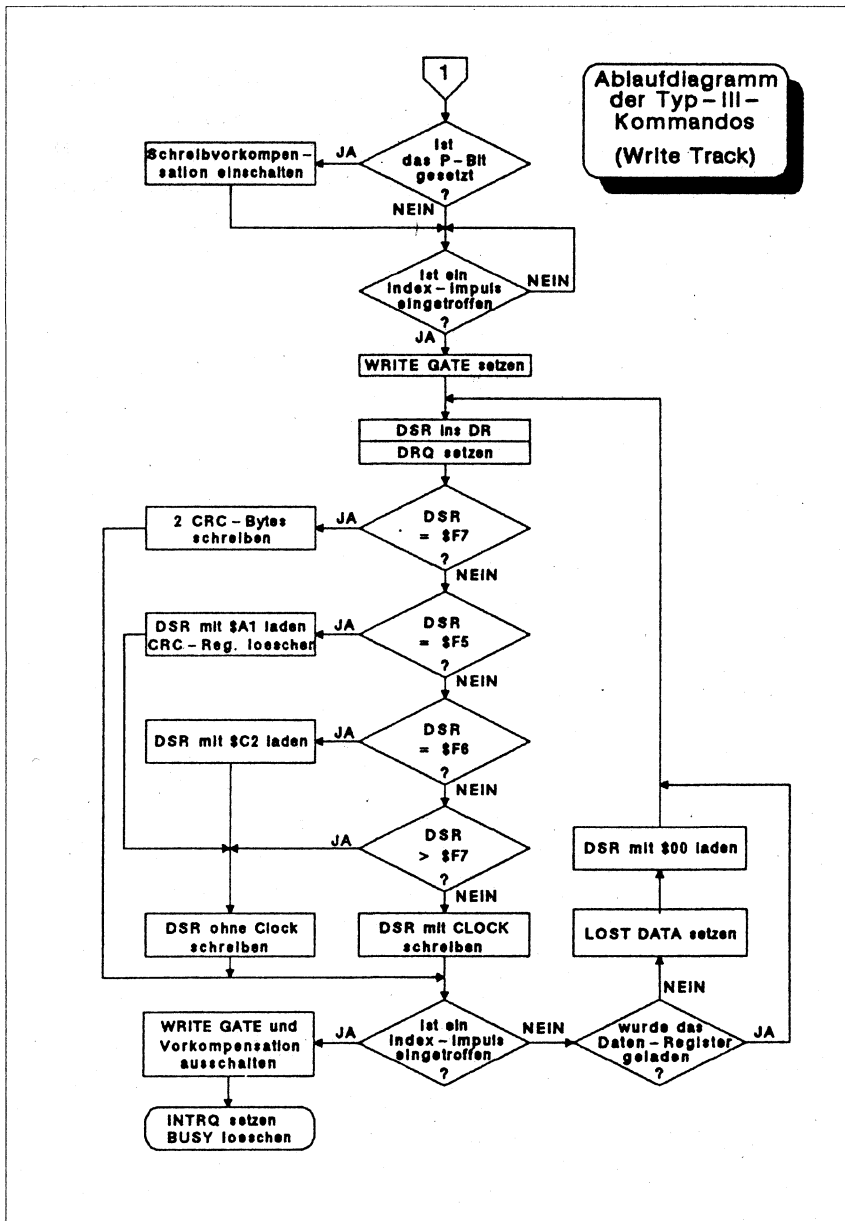


Abb. 14.10: Ablaufdiagramm der Typ-III-Kommandos II

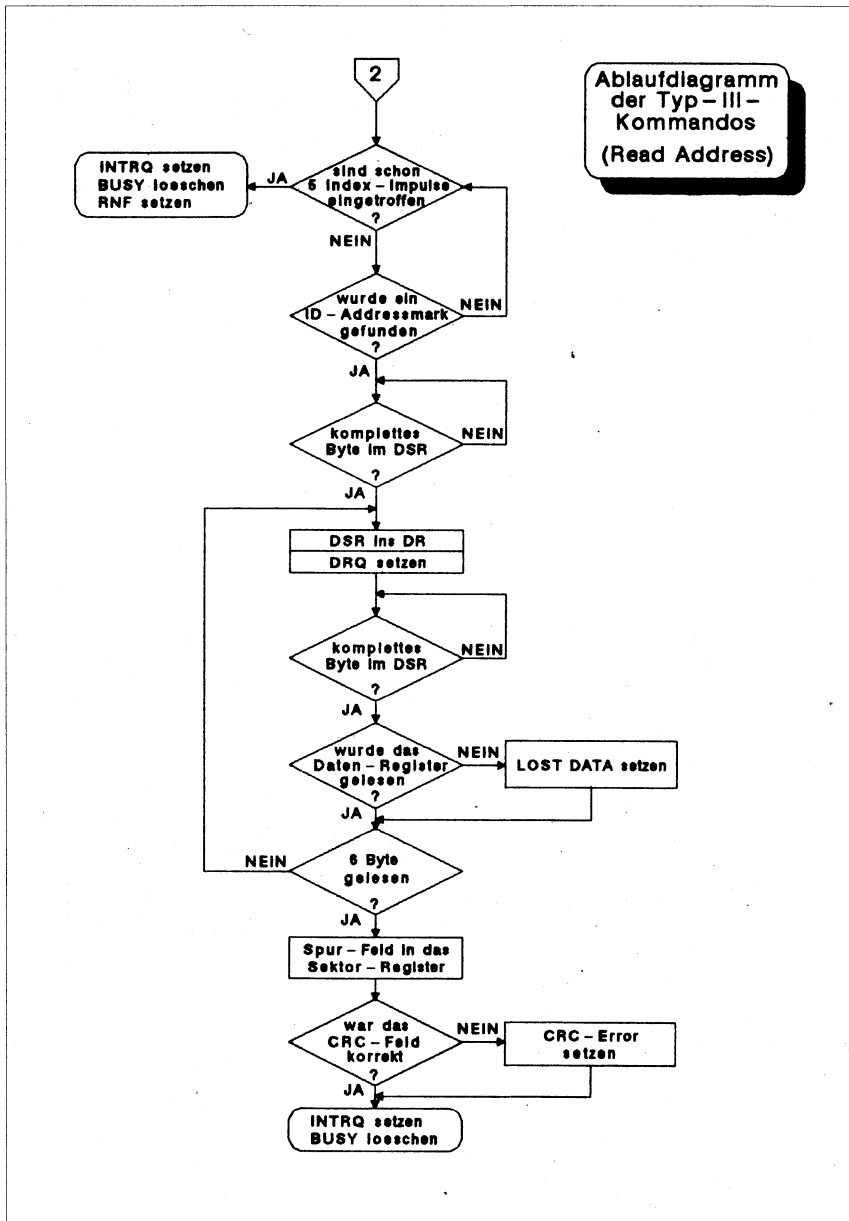


Abb. 14.11: Ablaufdiagramm der Typ-III-Kommandos III

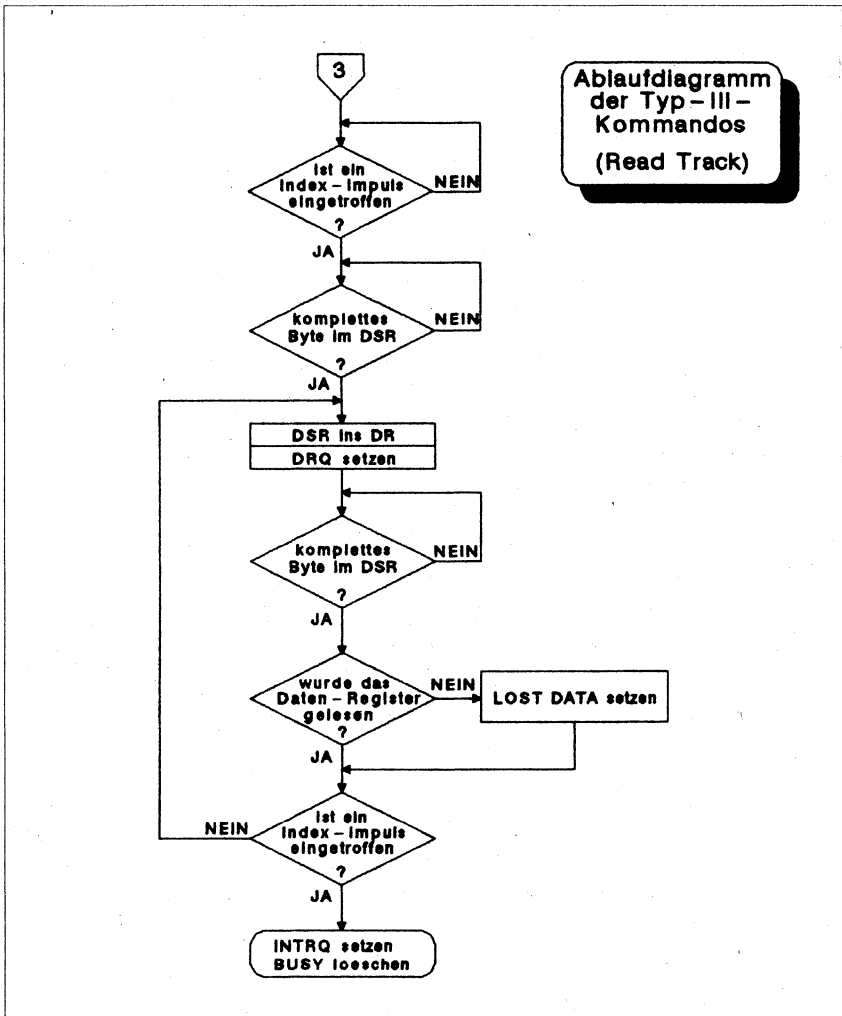


Abb. 14.12: Ablaufdiagramm der Typ-III-Kommandos IV

Das Typ-IV-Kommando

FORCE INTERRUPT

Kommando-Wort: 7 6 5 4 3 2 1 0

 1 1 0 1 I3 I2 I1 I0

Dieses Kommando ist das einzige, das an den Controller übergeben werden darf, während sich ein anderes Kommando in der Ausführung befindet. Es wird in der Hauptsache dazu benutzt, ein READ-SECTOR- oder WRITE-SECTOR-Kommando mit gesetztem m-Bit zu beenden.

Es gibt drei verschiedene Möglichkeiten des Interrupts. Diese werden durch die Bedingungsbits (I0-I3) im Kommandowort bestimmt. I0 und I1 haben keine Bedeutung und sollten gelöscht sein. Mit den Bits I2 und I3 wird die Art des Interrupts wie folgt selektiert:

- (\$D4) I2 = 1, Interrupt bei jedem Index-Impuls
- (\$D8) I3 = 1, Laufendes Kommando mit Interrupt beenden
- (\$D0) I2-I3 = 0, Laufendes Kommando ohne Interrupt beenden

Der Interrupt bei jedem Index-Impuls (\$D4) kann z.B. dazu verwendet werden, die Drehzahl des Laufwerks festzustellen. Eine andere Anwendung wäre das Synchronisieren des READ-ADDRESS-Kommandos mit dem Spur-Anfang. Bei READ-TRACK oder WRITE-TRACK ist das natürlich nicht nötig, da sie ohnehin erst mit dem Index-Impuls gestartet werden.

Durch die Interrupt's "\$D8" und "\$D0" kann ein, in der Ausführung befindliches, Kommando abgebrochen werden. Es ist zu beachten, daß der INTRQ-Ausgang nach einem "\$D8"-Interrupt nicht, wie üblich, durch Lesen oder Schreiben des Kommando-Registers zurückgesetzt wird. Dies kann nur erreicht werden, wenn dem "\$D8"- ein "\$D0"-Interrupt folgt und danach das Status-Register gelesen wird.

Wird dem FDC ein FORCE-INTERRUPT-Kommando übergeben, so muß vor dem nächsten Befehl eine Wartezeit von 16ms eingelegt werden, da sonst das Interrupt- Kommando nicht ausgeführt wird.

14.7.3 Status-Interpretation

Es hat sich herausgestellt, daß das Programmieren von Peripheriebausteinen, die für einen Daten-Transfer zuständig sind, fast ausschließlich dem Betriebs-

system überlassen wird. Allerdings bieten diese Bausteine - in den meisten Fällen - erheblich mehr "Features" als für den normalen Systembetrieb erforderlich sind. Dies ist der Grund dafür, daß die weniger gebräuchlichen Fähigkeiten nicht vom Betriebssystem ausgenutzt werden.

In Einzelfällen können diese "schlummernden" Talente der Peripherie-Chips aber ein Programmierproblem drastisch vereinfachen oder gar erst lösen. Warum trauen sich also nur - vergleichsweise wenige - Programmierer, diese "Talente" zu wecken? Nur etwa, weil es keinen passenden Betriebssystem-Aufruf dafür gibt? Nein - die Scheu eine solche Programmierung selbst vorzunehmen liegt meist in einer "Angst vor dem Status" begründet.

Das äußert sich dann so, daß man zwar weiß, wie man den Chip dazu veranlaßt, eine bestimmte Aufgabe zu erledigen, aber den Status, den man zurückerhält, nicht interpretieren kann.

Oft ist nicht bekannt, welchen Status man nach einer fehlerfreien Ausführung erhält, denn irgendwelche Bits sind im Statusregister fast immer gesetzt. Ein O.K.-Status kann also durchaus variieren. Wer schon hier nicht weiter weiß - wie soll er dann erst nach Erhalt eines Fehler-Status den Programmteil, der dem Aufruf folgt, weitercodieren?

Es ist zeitraubend, alle Möglichkeiten durchzuspielen und somit experimentell zu ermitteln, in welchen Fällen man welchen Status erhält. Aber das muß auch nicht sein. Diese Aufgabe haben wir - jedenfalls was den Floppy-Controller betrifft - schon für Sie erledigt. Wir zeigen an dieser Stelle nochmals die Bedeutung der einzelnen Status-Bits, die bereits eingehend beschrieben wurden:

Status-Register des FDC

Bit	Benennung	Bit=1 bedeutet
7	Motor On (MO)	Motor läuft
6	Write Protect (WPRT)	Disk schreibgeschützt
5	Record Type / Spin Up bzw. Spin Up	DATA-MARK gelöscht Drehzahl erreicht
4	Record not found (RNF)	Sektor nicht gefunden
3	CRC-Error (CRC)	Prüfsummen-Fehler
2	Spur 0 bzw. Lost Data	Kopf auf Track 0 Datenverlust
1	Index-Impuls bzw. Data Request	Index-Puls-Status Übertragungsbereit
0	Busy	Kommando aktiv

Es ist wissenswert, daß schon nach einer Positionierung des Schreib-/Lese-Kopfes festgestellt werden kann, ob eine schreibgeschützte Diskette (oder keine) in das Laufwerk eingelegt wurde.

Gemeinsam für alle Kommandos (Typ-1 bis Typ-3) gilt, daß im Statuswort, welches direkt im Anschluß nach einem Kommando gelesen wird, Bit-7 gesetzt (der Motor wird nicht sofort ausgeschaltet) und Bit-0 gelöscht ist (der FDC hat das Kommando ja beendet). Nach einem Typ-1-Kommando ist außerdem noch Bit-5 (Spin-Up) gesetzt.

Der Status nach einem Typ-1-Kommando

Beginnen wir mit dem korrekten Status nach einem Typ-1-Kommando. In den verwendeten Kommandoworten sind die "Stepping-Rate-Bits" für 3ms "Track to Track" gesetzt (r0=1, r1=0).

RESTORE-Kommando	normal	schreibgeschützt
01 (mit MO-Option, ohne Verify)	A4	E4
01 (siehe (1))	A6	E6
05 (mit MO-Option, mit Verify)	A4	E4
09 (ohne MO-Option, ohne Verify)	A4	E4
0D (ohne MO-Option, mit Verify)	A4	E4

SEEK-Kommando	normal	schreibgeschützt
11 (mit MO-Option, ohne Verify)	A0	E0
11 (siehe (3))	A2	E2
11 (siehe (2))	A4	E4
11 (siehe (1))	A6	E6
15 (mit MO-Option, mit Verify)	A0	E0
15 (siehe (2))	A4	E4
19 (ohne MO-Option, ohne Verify)	A0	E0
19 (siehe (2))	A4	E4
1D (ohne MO-Option, mit Verify)	A0	E0
1D (siehe (2))	A4	E4

STEP, STEP-IN, STEP- OUT	normal	schreibgeschützt
x1 (mit MO-Option, ohne Verify)	A0	E0
x1 (siehe (2))	A4	E4
x5 (mit MO-Option, mit Verify)	A0	E0
x5 (siehe (2))	A4	E4
x9 (ohne MO-Option, ohne Verify)	A0	E0
x9 (siehe (2))	A4	E4
xD (ohne MO-Option, mit Verify)	A0	E0
xD (siehe (2))	A4	E4

- (1) Dieser Wert gilt, wenn sich der Schreib-/Lese-Kopf schon vor dem RE-STORE-, bzw. einem SEEK-Kommando nach Spur 0, über der Spur 0 befand. Es ist, neben dem Spur-0-Bit, das IP-Bit gesetzt. Das liegt daran, das bei der Motor-On-Option, sechs Index-Impulse abgewartet werden. D.h., der FDC stellt während eines Index-Impulses fest, daß die gewünschte Spur erreicht ist, und beendet das Kommando.
- (2) Diesen Status trifft man nach einem SEEK-, STEP-, oder STEP-OUT-Kommando an, wenn der Schreib-/Lese-Kopf über Spur 0 positioniert wird.
- (3) Befindet sich der Schreib-/Lese-Kopf bei einem SEEK-Kommando bereits über der gewünschten Spur (außer Spur 0), so ist im Status-Wort das IP-Bit gesetzt. Es liegt hier im Prinzip der gleiche Sachverhalt wie unter (1) beschrieben zugrunde.

Ein Fehler-Status nach einem Typ-1-Kommando kann im allgemeinen nur erhalten werden, wenn im Kommandowort das Verify-Bit gesetzt war. Im Statuswort ist dann noch zusätzlich:

- (a) Falls kein ID-Feld gefunden wurde, das RNF-Bit gesetzt.

und

- (b) Falls kein korrektes ID-Feld gefunden wurde, das RNF- und das CRC-Bit gesetzt.

Somit ergibt sich ein Status von "B2" oder "BA", bzw. "F2" oder "FA" bei einer schreibgeschützten Diskette.

Geschieht das Ganze auf Spur 0, so ist natürlich auch noch das Spur-0-Bit gesetzt. Das Statuswort hat dann den Wert "B6" oder "BE" bzw. "F6" oder "FE" bei einer schreibgeschützten Diskette.

Es fällt auf, daß im Fehlerfall immer das IP-Bit gesetzt ist. Das hat hier aber nichts mit der Motor-On-Option zu tun, sondern erklärt sich dadurch, daß die (vergebliche) Suche nach einem ID-Feld mit dem sechsten Index-Impuls abgebrochen wird.

Der Status nach einem Typ-2-Kommando

Bei den Typ-2-Kommandos gestaltet sich die Status-Interpretation etwas einfacher.

Nach einem erfolgreichen WRITE-SEKTOR-Kommando enthält das Status-Register immer den Wert "80". Nach einem READ-SEKTOR-Kommando kann das Statuswort, falls ein Sektor mit "gelöshtem" Data-Mark gelesen wurde, auch "A0" betragen. Ansonsten ist hier ebenfalls eine "80" zu finden.

Wenn das Kommando nicht erfolgreich war, so ist der Status nach einem WRITE-SEKTOR-Kommando entweder:

- (a) "C0" nach dem Versuch eine schreibgeschützte Diskette zu beschreiben;
- (b) "90", wenn das zum gewünschten Sektor gehörende ID-Feld nicht gefunden wurde;
- (c) "98", falls die Prüfsumme (CRC) des ID-Feldes nicht korrekt war;

oder

- (d) "84", wenn nicht auf ein "DATA-REQUEST" des FDC reagiert wurde.

Nach einem fehlerhaften READ-SEKTOR-Kommando:

- (a) "90", wenn das zum gewünschten Sektor gehörende ID-Feld oder das DATA-MARK nicht gefunden wurde;
- (b) "98", falls die Prüfsumme (CRC) des ID-Feldes nicht in Ordnung war;
- (c) "88", wenn die Prüfsumme (CRC) im Daten-Feld einen Fehler aufwies;

oder

- (d) "84", wenn nicht auf ein "DATA-REQUEST" des FDC reagiert wurde.

Der Status nach einem Typ-3-Kommando

Noch einfacher ist der Status nach einem Typ-3-Kommando auszuwerten. Der Wert "80" zeigt auch hier eine erfolgreiche Ausführung an.

Für den Fehlerfall nach einem WRITE-TRACK-Kommando ergibt sich:

(a) "C0", bei einer schreibgeschützten Diskette;

oder

(b) "84", es wurde nicht auf einen DRQ des FDC reagiert.

Ein fehlerhafte Ausführung des READ-TRACK-Kommandos gibt es eigentlich gar nicht. Der FDC liest einfach nur, zwischen zwei Index-Impulsen, den RD-Eingang. Ganz gleich, ob eine Diskette im Laufwerk ist oder nicht.

Der einzige Fehler der vorstellbar ist, nämlich ein LOST-DATA (Status "84"), kann aufgrund eines Softwarefehlers in einem Mikroprogramm des FDC nicht ausgewertet werden. Das LOST-DATA-Bit wird, außer bei einem Datenverlust, auch noch abhängig vom gelesenen Format gesetzt.

Es kann also nach einem READ-TRACK-Kommando nicht festgestellt werden, ob ein gesetztes LOST-DATA-Bit tatsächlich einen Datenverlust anzeigt.

Es bleibt noch das READ-ADDRESS-Kommando übrig. Auch hier ist ein Status von "80" als gut zu werten. Ansonsten könnte sich noch ergeben:

(a) "90", wenn kein ID-Feld gefunden wurde;

(b) "88", falls der FDC einen Prüfsummen-Fehler im ID-Feld erkannt hat;

oder

(c) "84", wenn nicht auf eine Datenanforderung reagiert wurde.

14.8 Der Harddisk-Controller

In diesem Kapitel soll die direkte Programmierung des Harddisk-Controllers beschrieben werden. Im Normalfall ist die Programmierung mit den BIOS- oder GEMDOS-Funktionen sehr viel einfacher; doch es gibt Situationen, wo Sie direkten Einfluß auf den Controller nehmen möchte. Für diese Fälle sind

die unten abgedruckten Beispiele gedacht. Sehen Sie sie bitte nicht als den Normalfall der Programmierung an.

Der in den ATARI ST-Festplatten verwendete Controller ist ein sehr leistungsfähiges Gerät. Dieser Controller schafft eine Datenübertragungsrate von bis zu 8 MBit pro Sekunde, das ist etwa 1 MByte/Sekunde. Eine solche Datenflut würde den Speicher eines 1 MByte-ATARI in einer Sekunde füllen! Leider ist diese Zahl nicht für den wirklichen Datentransfer maßgeblich.

Eine starke "Bremse" der Datenübertragung ist die Mechanik, die in der Festplatte steckt. Gemeint ist die Rotationsgeschwindigkeit der Platte und der Schrittmotor, der den Schreib-/Lesekopf erst einmal an die richtige Stelle, also über den richtigen Track fahren muß. Alle diese Punkte verringern die tatsächlich erreichbare Geschwindigkeit des Datenaustauschs, die dennoch sehr hoch ist.

Der Controller hat eine recht einfache innere Struktur. Dennoch ist sein Befehlssatz so vielseitig, daß er sogar Fehlerkorrektur unterstützt.

Die Hardware des Controllers besteht hauptsächlich aus einem Disk-Controller, einem Kodierer/Dekodierer und einem Mikro-Controller. Diese Einzelteile haben die folgenden Aufgaben bzw. Funktionen:

Der Disk-Controller wandelt die Daten vom seriellen ins parallele Format um und umgekehrt. Außerdem wandelt er die Daten selbst in ein anderes Bit-Muster um, welches dann wirklich auf die Platte geschrieben wird. Dieses andere Format ermöglicht durch einen Trick die Erkennung von einfachen Fehlern beim Lesen.

Der Kodierer/Dekodierer wandelt die vom Disk-Controller erhaltenen Daten in die elektrischen Signale um, die den Schreibkopf steuern. Umgekehrt wandelt er die Signale, die beim Lesen vom Kopf kommen, in Bits um, wobei er gleichzeitig als Datenseparator dient (s. Floppy-Disk-Controller).

Der Mikro-Controller arbeitet wie ein eigentlicher Disk-Controller. Seine Aufgaben sind:

- Interpretation der vom Rechner kommenden Kommandos.
- Auswahl des angesprochenen Laufwerks (normalerweise ist nur eines vorhanden).
- Auswahl des Kopfes in dem Laufwerk (obere oder untere Plattenseite).

- Steuerung des Steppermotors, der den Schreib-/Lesekopf in die richtige Position fährt.
- Statusermittlung.

Hier nun ein einfaches Blockdiagramm des ATARI Hard-Disk-Controllers:

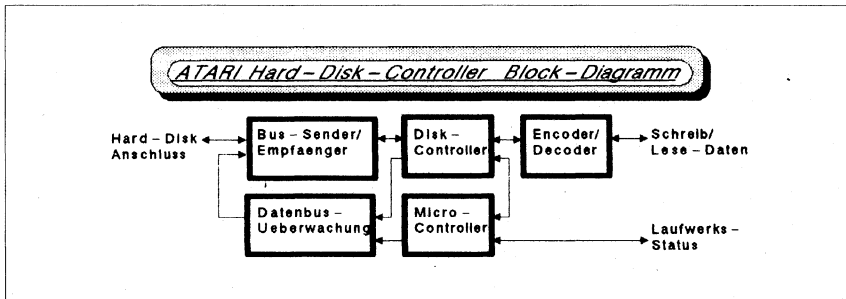


Abb. 14.13: Der Harddisk-Controller

Die Operationen, die man über den DMA-Bus mit der Harddisk durchführt, sind in fünf verschiedene Phasen aufgeteilt. Diese Phasen sind folgendermaßen definiert:

Reset-Phase

tritt auf, wenn entweder die RESET-Taste am ST betätigt bzw. der Rechner eingeschaltet wird oder der RESET-Befehl der 68000-Maschinensprache auftritt. Der Bus und damit der HDC werden in den Grundzustand versetzt.

Bus-frei-Phase

liegt vor, wenn kein Gerät auf den Bus zugreift.

Ziel-Anwahl-Phase

beginnt durch den Aufruf eines Gerätes durch Rücksetzen der SEL-Leitung. Die Adressierung des gewünschten Gerätes geschieht durch ein gesetztes Daten-Bit der 8-Bit-Parallel-Leitung. Das adressierte Gerät (hier: HDC) antwortet mit einem BUSY-Signal, worauf die SEL-Leitung wieder gesetzt wird. Danach beginnt die

Informations-Übertragungs-Phase

Während dieser Phase werden übertragen:

- Der Kommando-Block, sechs Bytes vom ST zum HDC.
- Der oder die Daten-Blöcke, wenn das gewählte Kommando dies erfordert.
- Das Status-Byte vom HDC zum ST, welches die erfolgreiche Operation anzeigt bzw. einen Fehler anzeigt. Dieses Byte ist jedoch immer Null, so daß die Status-Ermittlung nur durch die Erkennung eines eventuellen Timeout erfolgen kann.
- Das Completion-Byte vom HDC zum ST, ein Null-Byte, welches das Ende der gesamten Operation signalisiert.

Bus-Auslösungs-Phase

wird durch Setzen der BUSY-Leitung ausgelöst und bedeutet, daß der Bus nun frei für die nächste Operation ist. Danach befindet sich der Bus wieder in der BUS-frei-Phase.

14.8.1 Befehlsstruktur

Die Übertragung von Befehlen an den Hard-Disk-Controller ist genau festgelegt. Jedes Kommando wird in einem sechs Byte langen Block gesendet, dem "Command-Descriptor-Block". Hat der Controller ein solches Kommando erhalten, so teilt er dies dem Initiator, also dem ATARI ST, durch einen Interrupt mit.

Enthält das Kommando einen Befehl, einen bestimmten Track zu suchen (Verify, Format Track, Read, Write), so wird dies automatisch ausgeführt. Dabei wird der angegebene logische Datenblock, der gewünscht wird, in physikalische Größen wie Plattenseite und Tracknummer vom Controller umgewandelt.

Das folgende Diagramm zeigt die Struktur eines Kommandoblocks:

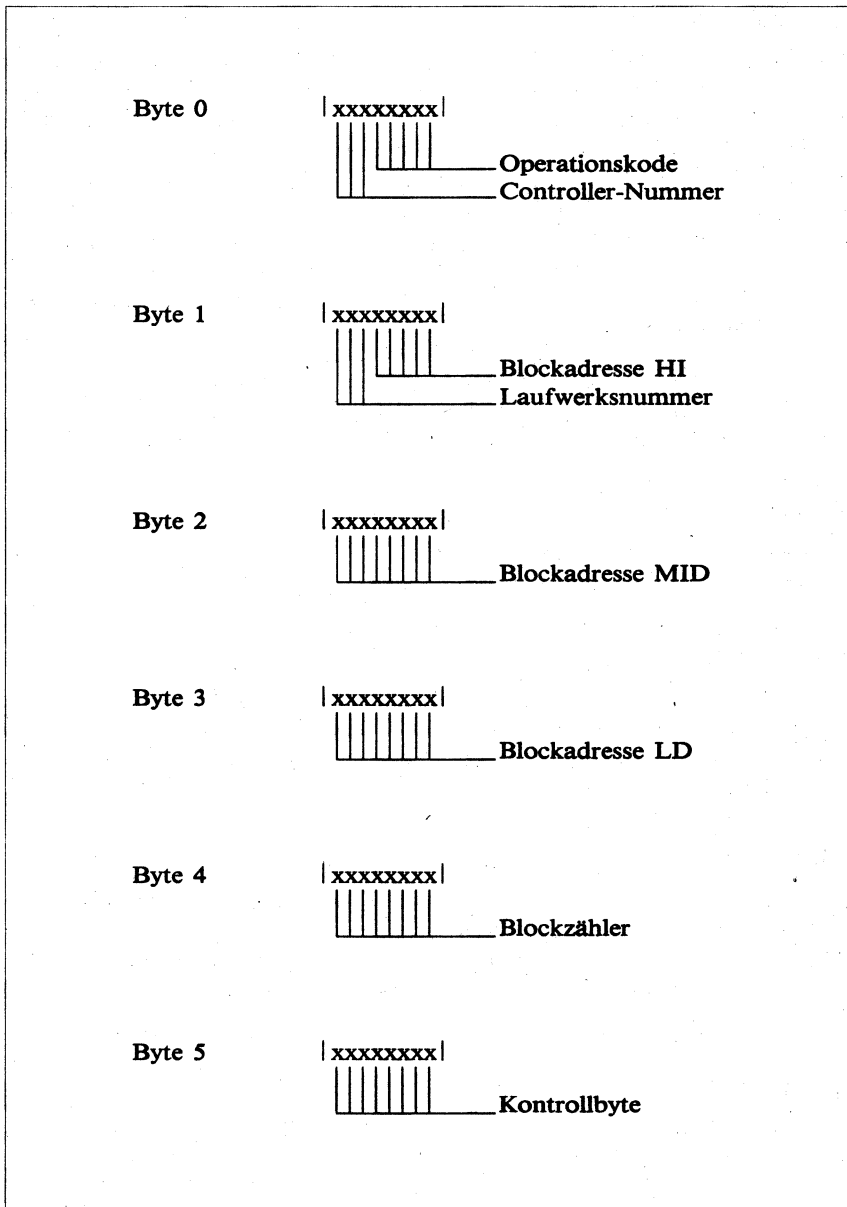


Abb. 14.14: Kommandoblock

Controller-Nummer

Dies ist ein 3-Bit-Wert (0-7), welcher die Nummer des gewählten Controllers darstellt. Somit ist es möglich, bis zu acht verschiedene Controller anzuschließen und zu bedienen. Die Nummer, bei der sich der einzelne Controller angesprochen fühlt, wird mit Hilfe von drei Schaltern auf der Platine des Controllers eingestellt. Kommt dann ein Kommando über die gemeinsame Bus-Leitung der Controller an, so testet jeder, ob er gemeint ist. Wenn nicht, so verhält er sich so, als ob er nicht existiere. Wenn doch, so beginnt die Kommunikation zwischen dem Rechner (Initiator) und dem angesprochenen Controller (Target = Ziel). Antwortet kein Controller auf das Kommando, so gibt der Rechner nach etwa vier Sekunden auf: Timeout.

Eines ist hierbei noch zu erwähnen: Die Rollen der einzelnen Geräte als Initiator und Ziel sind festgelegt. Eine Kommunikation zwischen gleichen Geräten ist somit unmöglich.

Operationskode

Dieser Kode, auch OpCode genannt, enthält in fünf Bits das auszuführende Kommando. Dadurch sind nur Kommandos von 0 bis 31 möglich.

Laufwerksnummer

Ähnlich wie die Controller-Nummer ist dies eine 3-Bit-Zahl, die das gewählte Laufwerk bezeichnet. Jeder der bis zu acht Controller kann somit jeweils bis zu acht Laufwerke steuern, wodurch also theoretisch 64 Laufwerke an den ATARI anschließbar sind.

Blockadresse

Diese 21-Bit-Zahl bezeichnet den gewählten logischen Datensektor. Die Umrechnung dieser Zahl (bis zu 2097151) in die physikalischen Werte erledigt der Controller. Die ATARI-Festplatte enthält 41616 Sektoren, so daß die Blockadresse diesen Wert nicht überschreiten und, da die Blocknummer bei 0 beginnt, auch nicht erreichen.

Blockzähler

Dieser Zähler bestimmt die Anzahl der zu lesenden bzw. zu schreibenden Sektoren. Der Zähler muß einen Wert ungleich 0 enthalten (1-255).

Kontrollbyte

Dieses Byte enthält verschiedene Angaben, je nach dem verwendeten Kommando. Um nun einen so aufgebauten Kommando-Block an den HDC zu übertragen, muß folgendermaßen vorgegangen werden:

Zunächst wird der Prozessor durch die SUPER-Funktion (\$20) des GEMDOS (TRAP #1) in den Supervisor-Modus geschaltet, da einige privilegierte Zugriffe auf Hardware-Register erfolgen müssen.

Danach werden durch Setzen der System-Variablen FLOCK (\$43E) die Routinen der Floppy-Bearbeitung gesperrt. Dies ist nötig, da sowohl der HDC als auch der FDC über die selben Hardware-Register gesteuert werden. Damit es dabei keine Überschneidungen geben kann, wie z.B. ein OK-Signal des einen Controllers, wenn auf das OK des anderen gewartet wird, wird der FDC quasi aus dem System entfernt und kann nicht mehr mitmischen. In dem Hardware-Register \$FF8606, im folgenden WDL genannt, werden durch Einschreiben des Wertes \$88 die Bits 7 und 3 gesetzt, alle anderen gelöscht. Dadurch wird einerseits der HDC selektiert und andererseits die Leitung A1, die durch Bit 1 angesprochen wird, auf 0 gelegt.

Diese Leitung A1 dient zur Signalisierung an den HDC, daß nun ein Kommando-Byte (das erste Byte eines Kommando-Blocks) übertragen wird.

Danach wird im Register \$FF8604, im folgenden WDC genannt, das Kommando-Byte übergeben. Der HDC übernimmt dieses Byte und meldet dies durch ein 0-Signal an der HDC-Interrupt-Leitung. Diese Leitung liegt an Bit 5 des I/O-Portes des Multi-Funktions-Chip MFP und ist somit an der Adresse \$FFFA01 zu finden.

Dieser Interrupt findet auch nach jedem weiteren übertragenen Byte statt. Kommt er nicht, so wurde das übertragene Byte entweder nicht erkannt, oder der HDC ist nicht bereit, Daten zu empfangen.

Während der Übertragung der Kommando-Bytes wird auf den Interrupt maximal 100 Millisekunden lang gewartet, nach der vollständigen Übertragung des Kommando-Blocks sogar bis zu drei Sekunden, da dann ja das Kommando vollständig ausgeführt werden muß, bis der HDC ein OK melden kann. Ist nach dieser Zeit immer noch kein Interrupt erfolgt, so wird die Übertragung der Kommando-Bytes abgebrochen und ein sogenanntes Timeout gemeldet. Ist das Kommando-Byte übertragen und rechtzeitig durch den Interrupt quittiert, so wird eine \$8A in das WDL geschrieben, wodurch die A1-Leitung wieder zu 1 wird.

Die restlichen fünf Bytes des Kommando-Blocks werden nun nach dem gleichen Schema, nur mit gesetztem Bit 1 (A1), übertragen. Zusammen mit einer \$8A in WDL wird das Byte in WDC geschrieben, bis zu 100 Millisekunden auf den Interrupt gewartet (sonst Timeout) und dann das nächste Byte übertragen. Nach Übertragung des letzten Bytes (Byte 5) des Kommando-Blocks wird nun länger (max. drei Sekunden) auf den Interrupt gewartet, damit der HDC Zeit genug hat, das Kommando auszuführen.

Ist der Interrupt in der Zeit erfolgt, wird eine \$80 in das WDL-Register geschrieben, um Bit 3 zurückzusetzen und damit den HDC zu deselektieren. Dadurch wird der FDC, der Floppy-Disk-Controller, wieder freigegeben. Anschließend wird die System-Variable FLOCK (\$43E) wieder auf 0 gesetzt, um die Floppy-Operationen wieder zu ermöglichen. Schließlich und endlich kann danach der Prozessor wieder in den User-Modus zurückgeschaltet werden.

Hier nun ein kleines Programm, welches die oben beschriebenen Schritte durchführt und einen Kommando-Block zum HDC sendet. Bitte beachten Sie, daß dies nur vollständig funktioniert, wenn das Kommando keine Datenübertragung über die DMA beinhaltet (z.B. READ, WRITE), da dann die DMA ebenfalls programmiert werden muß. Dazu kommen wir später.

```
; ** Hard-Disk-Zugriff S.D. **
; ** sendet Kommando-Bytes aus COM-Feld zum HDC **
wdc      = $ff8604
wdl      = $ff8606
wdcwdl   = wdc
port     = $fffa01
flock    = $43e
run:
    move.b    #'0',num        ;Timeout-Meldung vorbereiten
    clr.l     -(sp)
    move      #$20,-(sp)
    trap      #1              ;in Super-Modus umschalten
    addq.l    #6,sp
    move.l    d0,spsave       ;alten Stackpointer retten
    lea       com,a0          ;Zeiger auf Kommando-Block
    bsr       send            ;Kommando-Block an HDC senden
    bra       exit            ;fertig
send:
    st        flock           ;Kommando-Block an HDC senden *
    move      #$88,wdl        ;Floppy sperren
    clr.l     d0              ;HDC selektieren, A1=0
    moveq     #5,d2           ;Zähler: sechs Bytes
loop:
    clr.l     d0
    move.b    (a0)+,d0        ;Byte holen
    bsr       send_byte       ;Byte an HDC senden
    bmi       error           ;Timeout !
```

```

cont:      dbra          d2,loop          ;weitermachen

          move          #$8a,wdl
          bsr           wait1            ;max. 3s auf Interrupt warten
          bmi           error            ;Timeout !
          move          #$8a,wdl
          move          wdc,d0           ;Status-Byte holen
          move          #$80,wdl         ;HDC deselektieren
          move          wdc,d1           ;Completion-Byte holen
          clr           flock            ;Floppy freigeben
          rts            ;fertig

exit:      move.l       spsave,-(sp)
          move          #$20,-(sp)
          trap          #1               ;Umschalten in User-Modus
          addq.l        #6,sp
          rts                           ;Ende

error:     clr          flock            ;Error melden
          move.l        #senderr,d0      ;Floppy freigeben
          bsr           pline            ;Fehlermeldung ausgeben
          bra           exit             ;und Schluß

send_byte: * ein Byte zum HDC senden *
          swap          d0               ;Byte ins HI-Wort
          move          #$8a,d0          ;$8A ins LO-Wort
          move.l        d0,wdcwdl       ;WDC und WDL setzen
          bra           wait            ;warten auf OK (Interrupt)

wait1:     add.b         #1,num           ;Durchlauf-Nummer+1
          move.l        #45000,d3        ;Timeout nach 3 Sekunden
          bra           wait1            ;warten...

wait:      add.b         #1,num           ;Durchlauf-Nummer+1
          move.l        #1500,d3         ;Timeout nach 100 ms

wait1:     subq.l        #1,d3            ;Timeout-Zähler-1
          bmi           timeout          ;Timeout !
          move.b        port,d0          ;I/O-Port laden
          and.b         #$20,d0          ;Bit 5 ausblenden
          bne           wait1            ;noch gesetzt, weiterwarten
          moveq         #0,d3            ;OK übergeben
          rts                           ;fertig

timeout:   moveq         #-1,d3           ;nicht OK übergeben

pline:     rts                           ;* Zeile auf Bildschirm ausgeben *

          move.l        d0,-(sp)
          move          #9,-(sp)
          trap          #1
          addq.l        #6,sp
          rts

spsave:    dc.l 0
senderr:   dc.b "ERROR bei send_byte "

```

```
num:    dc.b "1. mal !",10,13,0
com:    dc.b $b,$0,$0,0,0,$0
even
```

Die Bytes des hier übertragenen Kommando-Blocks lassen den Schreib-/Lesekopf der Harddisk auf Spur 0 fahren (\$B=Seek). Das Programm beinhaltet zu Testzwecken noch eine Fehler-Ausgabe, die ein Timeout mit Angabe des Zeitpunktes auf dem Bildschirm ausgibt. Dieser Teil kann natürlich entfallen, er dient nur zur Kontrolle der ordnungsgemäßen Übertragung des Kommando-Blocks.

Etwas komplizierter wird die Übertragung eines Schreib- bzw. Lese-Befehls an den HDC. Dabei muß zusätzlich zur Übertragung des Kommando-Blocks noch die DMA (Direct Memory Access) programmiert werden, die für die Übertragung der Daten zwischen Harddisk und Computer-Speicher zuständig ist. Die DMA benötigt folgende Informationen:

- Die Speicher-Adresse, aus der bzw. in den die zu übertragenen Daten-Bytes zu lesen sind. Diese Adresse wird in den Hardware-Registern \$FF8609, \$FF860B und \$FF860D übergeben, und zwar erst das LO-, dann das MID- und dann das HI-Byte der Adresse. Da dabei ein Byte zu einer vollständigen 32-Bit-Adresse fehlt, kann diese Adresse "nur" in einem Bereich zwischen 0 und \$FFFFFF liegen (s. auch FDC-Programmierung).
- Die Richtung, in der die Daten zu übertragen sind, d.h. Lesen oder Schreiben. Diese Information erhält die DMA aus Bit 8 des WDL-Wortes, eine 0 bedeutet Lesen in und eine 1 meint Schreiben aus dem Speicher.
- Den Zustand, ob die DMA überhaupt eingeschaltet ist. Dies erfährt die DMA aus Bit 6 des WDL-Registers \$FF8606. Normalerweise ist die DMA immer eingeschaltet, d.h. Bit 6=0.

Es ist auch wichtig, in welchem Moment der DMA diese Informationen gegeben werden, damit nicht Überschneidungen mit vorhergehenden DMA-Aktionen stattfinden können. Soll von der Harddisk gelesen werden, so wird erst das Kommando-Byte an den HDC übergeben und dann erst die DMA-Adresse gesetzt. Dadurch wird ausgeschlossen, daß die DMA unerwünschte Daten in den Speicher lädt, da der HDC nach Empfang des Kommando-Bytes erst auf die weiteren Bytes des Kommando-Blocks wartet.

Um auf die Festplatte zu schreiben, wird dagegen erst die DMA-Adresse gesetzt und danach das Kommando-Byte übertragen. Wie dies praktisch zu bewerkstelligen ist, können Sie anhand des Programmes "HDC-Tools" im Kapitel

14.8.3 erkennen. Zunächst wollen wir jedoch bei der leidigen Theorie bleiben und die Kommandos für den HDC betrachten.

14.8.2 Liste der Befehle

Der beim ATARI ST verwendete Befehlssatz enthält nur neun verschiedene Kommandos. In den verschiedenen Handbüchern der Harddisk sind zwar einige andere Kommandos aufgeführt, die jedoch nicht in der angegebenen Weise bzw. gar nicht funktionieren. Hier eine Übersicht über die funktionierenden Kommandos mit dem dazugehörigen sedezimalen OpCode:

Harddisk-Kommandos

OpCode	Kommando
00	Test Unit Ready
01	Restore
03	Request Sense
04	Format Drive
08	Read
0A	Write
0B	Seek
15	Mode Select
1B	Seek to Shipping-Position

Es folgt nun eine Erklärung der einzelnen Kommandos mit ihren Parameter-Bytes. Das "-"-Zeichen bedeutet, daß das Bit keine Bedeutung hat. Diese Bits sollten dann auf 0 gelegt werden.

Test Unit Ready (00)

Mit diesem Kommando kann der Rechner den Bus ansprechen und feststellen, welche Geräte angeschlossen sind.

Byte 0: xxx00000
 Kommando 00, Test Unit Ready
 Controller-Nummer

Byte 1: xxx----
 Laufwerks-Nummer

Byte 2 bis 5: -----

Ist das angegebene Laufwerk eingeschaltet und bereit, so wird im Status-Byte eine Null übergeben, andernfalls wird das Check Condition-Bit gesetzt.

Restore (01)

Diese Kommando setzt den HDC in den Grundzustand zurück und läßt den Schreib-/Lesekopf des Laufwerks zur Spur 0 zurückfahren.

Byte 0: xxx00001
 Kommando 01, Restore
 Controller-Nummer

Byte 1: xxx-----
 Laufwerks-Nummer

Byte 2 bis 5: -----

Request Sense (03)

Dieses Kommando gibt 4 Bytes zurück (viermal WDC auslesen!), wovon nur das erste Byte eine Bedeutung hat. Es enthält den Error-Code des zuletzt ausgeführten Kommandos. Ist kein Fehler aufgetreten, so erhält man dort eine 0.

Byte 0: xxx00011
 Kommando 03, Request Sense
 Controller-Nummer

Byte 1: xxx-----
 Laufwerks-Nummer

Byte 2: -----

Byte 3: -----

Byte 4: 00000100 = \$04 Bytes werden zurückgegeben

Byte 5: -----

Format Drive (04)

Dieses Kommando veranlaßt den HDC, die gesamte (!) Harddisk zu formatieren. Es ist somit nicht besonders empfehlenswert, dieses Kommando auszuprobieren!

Es werden dem Kommando einige Parameter mitgegeben:

- Das Data-Pattern-Flag, welches aus zwei Bits besteht und bestimmt, welche Daten auf die leeren Sektoren geschrieben werden sollen. Sind die Bits nicht gesetzt (0), so werden in alle Sektoren \$6C geschrieben. Sind die Bits gesetzt, so wird das in Kommando-Byte 2 übergebene Byte geschrieben.
- Data-Pattern: Hier steht das Byte, mit dem bei gesetztem Data-Pattern-Flag die formatierten Sektoren gefüllt werden. Ist das Flag nicht gesetzt, so hat dieses Byte keine Bedeutung.
- Interleave-Faktor: Dieser Wert gibt den Abstand zwischen zwei der Nummer nach aufeinanderfolgenden Sektoren an. Ist der Faktor 1, so werden die Sektoren der Reihe nach auf die Tracks geschrieben. Ist er z.B. 2, so wird zwischen Sektor 1 und 2 ein anderer Sektor gelegt. Die Reihenfolge der 17 Sektoren eines Tracks wäre dann folgende:

laufende Nummer	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Sektor-Nummer	1	10	2	11	3	12	4	13	5	14	6	15	7	16	8	17	9

Somit benötigt man zwei Umdrehungen der Platte, um alle Sektoren eines Tracks zu lesen. Der Vorgang wird dadurch zwar langsamer, allerdings auch sicherer, da nach jedem gelesenen Sektor eine kleine Pause zum nächsten entsteht. Normalerweise ist dieser Faktor auf 1 gesetzt - und dabei kann es auch bleiben.

Byte 0:	xxx00100 Kommando 04, Format Drive Controller-Nummer
Byte 1:	xxx--xx- Data-Pattern-Flag Laufwerks-Nummer
Byte 2:	xxxxxxx Data-Pattern
Byte 3:	xxxxxxx Interleave-Faktor HI (sollte 0 sein)
Byte 4:	xxxxxxx Interleave-Faktor LO (normalerweise 1)
Byte 5:	-----

Read Sectors (08)

Dieses Kommando veranlaßt den Controller, den Schreib-/Lesekopf auf die Spur des gewünschten Startsektors zu bewegen und dann die angegebene Anzahl von Sektoren zu lesen und an den Computer zu übertragen. Zusätzlich zur Übertragung des Kommando-Blocks muß die DMA programmiert werden, damit die ankommenden Daten auch in den entsprechenden Speicherbereich geschrieben werden.

Byte 0: xxx01000
 Kommando 08, Read Sectors
 Controller-Nummer

Byte 1: xxxxxxxx
 Sektor-Nummer HI
 Laufwerks-Nummer

Byte 2: xxxxxxxx Sektor-Nummer MID

Byte 3: xxxxxxxx Sektor-Nummer LO

Byte 4: xxxxxxxx Anzahl der zu lesenden Sektoren

Byte 5: -----

Write Sectors (0A)

Durch dieses Kommando werden Sektoren beschrieben. Der Kopf wird auf den entsprechenden Track gefahren und die über die DMA abgeschickten Daten empfangen und auf die Sektoren geschrieben. Die DMA muß hierbei ebenfalls zusätzlich zur Kommando-Block-Übertragung programmiert werden.

Byte 0: xxx0101
 Kommando 0A, Write Sectors
 Controller-Nummer

Byte 1: xxxxxxxx
 Sektor-Nummer HI
 Laufwerks-Nummer

Byte 2: xxxxxxxx Sektor-Nummer MID

Byte 3: xxxxxxxx Sektor-Nummer LO

Byte 4: xxxxxxxx Anzahl der zu schreibenden Sektoren

Byte 5: -----

Seek (0B)

Durch dieses Kommando wird der Schreib-/Lesekopf des Laufwerks bewegt. Der Controller errechnet aus der mit dem Kommando übergebenen Sektornummer den entsprechenden Track und fährt den Kopf dort hin.

Byte 0: xxx01011
 Kommando 0B, Seek
 Controller-Nummer

Byte 1: xxxxxxx
 Sektor-Nummer HI
 Laufwerks-Nummer

Byte 2: xxxxxxx Sektor-Nummer MID

Byte 3: xxxxxxx Sektor-Nummer LO

Byte 4: -----

Byte 5: -----

Mode Select (15)

Dieses Kommando wird zur Einstellung der Parameter für die Formatierung der Harddisk verwendet. Es wird nach der Übergabe des Kommandos (mit DMA-Programmierung !) ein 16-Byte-Block an den HDC übergeben.

Byte 0: xxx10101
 Kommando \$15, Mode Select
 Controller-Nummer

Byte 1: xxx----
 Laufwerks-Nummer

Byte 2: -----

Byte 3: -----

Byte 4: 00010110 = 16 Bytes werden übergeben

Byte 5: -----

Seek to Shipping-Position (1B)

Durch dieses Kommando wird der Schreib-/Lesekopf auf eine Position gefahren, in der er sicher gegen Erschütterungen des Laufwerks ist. Diese Position nennt man auch "Shipping-Position", da sie für den Transport des Laufwerks vorgesehen ist.

Das Programm SHIP.PRГ läßt alle angeschlossenen Festplatten-Laufwerke auf diese Position fahren. Es sollte deshalb unbedingt vor dem Transport der Harddisk aufgerufen werden. Dabei ist zu beachten, daß der Aufruf dieses Programms von einer Diskette aus geschehen soll und kein Inhaltsverzeichnis-Fenster auf dem Bildschirm ist, da sonst nach der Rückkehr aus dem SHIP-Programm wieder das Inhaltsverzeichnis der Festplatte ausgelesen wird. Dieser Zugriff auf die Festplatte wird aber den Kopf wieder aus der sicheren Position herausfahren.

Außerdem funktioniert das erste Kommando nach dem 1B-Kommando nicht korrekt, da der Kopf erst aus seiner Shipping-Position fahren muß.

Byte 0: xxx11011
 Kommando 1B
 Seek to Shipping-Position
 Controller-Nummer

Byte 1: xxx-----
 Laufwerks-Nummer

Byte 2 bis 5: -----

14.8.3 HDC-Tools

Zur Demonstration des Schreib- bzw. Lesezugriffes auf die Harddisk folgt nun ein Programm, welches wahlweise einen oder mehrere Sektoren liest und in den Speicher überträgt oder umgekehrt Sektoren mit Daten aus dem Speicher beschreibt. Im Beispiel werden acht Sektoren ab Sektor 132 in den Speicher geladen. Dort liegt auch das Inhaltsverzeichnis des ersten Harddisk-Teils (Partition). Natürlich könnten Sie das Ganze auch sehr einfach mit einem BIOS-Aufruf bewerkstelligen - aber das hier ist eben selbst gemacht.

Außerdem ist in diesem Programm die einfache Übertragung eines Kommandoblocks integriert, ähnlich wie im Beispiel des Kapitels 14.8.1.

```

; ** Harddisk-Sektor lesen/schreiben, Kommando senden **
wdc      = $ff8604 ; FDC/HDC-Access, DMA-Sector-Count
wdl      = wdc+2 ; DMA-Mode/Status
dma      = $ff8609 ; DMA-Adresse HI
flock    = $43e ; Floppy-VBL-Flag
port     = $ffffa01 ; Parallel-Port, Bit 5=HDC-IRQ
run:
        clr.l -(sp)
        move    #$20, -(sp)
        trap     #1 ; in Supervisor-Mode schalten

```

```

        addq.l    #6,sp
        move.l    d0,spsave    ;User-Stackpointer retten
bp1:
        bra      put          ;für Nur-Übertragung
        pea      puffer       ;Puffer-Adresse
        move     #8,-(sp)      ;8 Sektoren
        move.l    #132,-(sp)   ;ab Sektor 132
        bsr      read         ;Sektor(en) in Puffer lesen
        bra      bp2
put:
        bsr      send         ;Kommando-Block übertragen
bp2:
        move.l    spsave,-(sp)
        move     #20,-(sp)
        trap     #1           ;in User-Mode schalten
        addq.l    #6,sp
        rts                ;Rückkehr zum Aufrufer
                           ; oder
        clr      -(sp)
        trap     #1           ;Rückkehr zum Desktop
send:
        lea      wdc,a0
        lea      com,a1       ;Zeiger auf Kommando-Block
        st       flock        ;Floppy sperren
        move     #88,wdl      ;HDC selektieren, A1=0
        clr.l    d0
        moveq     #5,d1
loop:
        clr.l    d0
        move.b    (a1)+,d0
        bsr      send_byte    ;Byte an HDC senden
        bmi      tout         ;Timeout !
        dbra     d1,loop      ;sonst weitermachen
        bsr      waitl        ;max. 3 Sekunden warten
        bmi      tout         ;Timeout !
        move     wdc,d6
        move     #80,wdl      ;sonst
        clr      flock        ;Floppy freigeben
        rts                ;fertig
read:   ; * Sektor(en) lesen *
        lea      wdc,a0
        st       flock        ;Floppy-VBL-Routine sperren
        move     #88,2(a0)    ;HDC-Zugriff, A1=0
        nop
        move.l    #08008a,(a0) ;READ-Kommando
        move.l    10(sp),-(sp) ;Puffer-Adresse
        bsr      setdma       ;DMA setzen
        addq.l    #4,sp
        bsr      set_parameters ;Sektoranzahl und -Nummer
        bmi      tout         ;Timeout aufgetreten !
        move     #190,2(a0)
        nop
        move     #90,2(a0)    ;Umschalten auf READ

```

```

        nop
        move      8(sp), (a0)      ;Sector-Count an DMA senden
        nop
        move      #$8a, 2(a0)
        nop
        move.l    #0, (a0)         ;übertragung starten
        bsr       waitl           ;max. 3 Sekunden warten
        bmi       tout           ;Timeout !
        move      #$8a, 2(a0)
        bra       exec
write:   ; * Sektor(en) schreiben *
        lea       wdc, a0
        st        flock           ;Floppy-VBL sperren
        move.l    10(sp), -(sp)
        bsr       setdma         ;DMA-Adresse setzen
        addq.l    #4, sp
        move      #$88, 2(a0)     ;HDC-Zugriff, A1=0
        nop
        move.l    #$0a008a, (a0) ;WRITE-Kommando
        bsr       set_parameters ;Sektoranzahl und -nummer
        bmi       tout           ;Timeout !
        move      #$90, 2(a0)
        nop
        move      #$190, 2(a0)    ;Umschalten auf WRITE
        nop
        move      8(sp), (a0)     ;Sector-Count an DMA senden
        nop
        move      #$18a, 2(a0)
        nop
        move.l    #$100, (a0)     ;übertragung starten
        bsr       waitl           ;max. 3 Sekunden warten
        bmi       tout           ;Timeout !
        move      #$18a, 2(a0)
exec:
        nop
        move.l    (a0), d6         ;HDC/DMA-Status in D6 holen
        and.l     #$ff00ff, d6    ;HI=HDC, LO=DMA
tout:
        move      #$80, 2(a0)     ;auf FDC umschalten
        nop
        move.l    (a0), d7         ;Completion-Byte holen
        and.l     #$ff00ff, d7    ;HI=HDC (0), LO=DMA
        clr       flock           ;Floppy-VBL-Routine freigeben
        rts        ;fertig
set_parameters: ;Sektor-Anzahl und Sector-Count setzen
        move      #$8a, 2(a0)
        bsr       wait           ;warten auf HDC-OK
        bmi       setpx          ;Timeout !
        clr       d0
        move.b    4+5(sp), d0     ;Sektornr. HI
        bsr       send_byte
        bmi       setpx
        move.b    4+6(sp), d0    ;Sektornr. MID

```

```

        bsr      send_byte
        bmi      setpx
        move.b   4+7(sp),d0      ;Sektornr. LO
        bsr      send_byte
        bmi      setpx
        move     4+8(sp),d0      ;Anzahl der Sektoren
        bsr      send_byte

setpx:
        rts                      ;fertig
send_byte: ; * 1 Byte zum HDC senden *
        swap     d0
        move     #$8a,d0
        move.l   d0,(a0)
        bra      wait
wait1:  ;max. 3 Sekunden auf OK warten
        move.l   #450000,count
        bra      wait1
wait:   ;max. 100 ms auf OK warten
        move.l   #15000,count
wait1:
        subq.l   #1,count
        bmi      timeout
        move.b   port,d0
        and.b    #$20,d0        ;HDC-Interrupt ?
        bne      wait1          ;nein
        moveq    #0,d0          ;ja => OK
        rts

timeout:
        move.l   #errline,d0
        bsr      pline           ;'Timeout' ausgeben
        moveq    #-1,d0          ;Timeout anzeigen
        rts
setdma: ; * DMA-Adresse setzen *
        move.b   7(sp),dma+4     ;LO
        move.b   6(sp),dma+2     ;MID
        move.b   5(sp),dma      ;HI
        rts
pline:  ; * Zeile auf Bildschirm ausgeben *
        move.l   d0,-(sp)
        move     #9,-(sp)
        trap     #1
        addq.l   #6,sp
        rts
errline: dc.b    "Timeout aufgetreten !",10,13,0
com:     dc.b    $b,0,0,132,0,0
even
count:   dc.l    1                ;Timeout-Counter
spsave:  dc.l    0                ;User-Stackpointer
puffer:  blk.b    512*8,$FF        ;Puffer für 8 Sektoren

```

Mit diesem Programm ist man nun in der Lage, direkt Sektoren von der Hard-disk zu laden bzw. zu schreiben. Die Status-Information, die eigentlich erfol-

gen sollte, wird in das Register D6 geschrieben, was bei Verwendung eines Debugger-Monitors (z.B. das SID-Programm oder K-SEKA) abgefragt werden kann.

14.8.4 Partitions-Analysator

Der Sektor 0 der Harddisk ist recht interessant, da er Informationen über die Partitionen enthält. Um diese Informationen zu lesen und auszuwerten, kann das nun folgende Programm verwendet werden. Es enthält u.a. auch Teile des obigen Programmes (read), so daß diese evtl. direkt übernommen werden können.

Das Programm liest auf die bekannte Weise den Sektor 0 der Harddisk aus und interpretiert die darin enthaltenen Daten. Diese werden dann auf dem Bildschirm ausgegeben, wobei alle Zahlen sedezimal (hexadezimal) dargestellt werden.

```

;** Partitions-Analysator S.D. **
wdc      = $ff8604          ;FDC/HDC-Access, DMA-Sector-Count
wdl      = wdc+2            ;DMA-Mode/Status
dma      = $ff8609          ;DMA-Adresse HI
flock    = $43e             ;Floppy-VBL-Flag
port     = $fffa01          ;Parallel-Port, Bit 5=HDC-IRQ
run:
        lea        stp,sp
        clr.l      -(sp)
        move       #$20,-(sp)
        trap       #1          ;in Supervisor-Mode schalten
        addq.l     #6,sp
        move.l     d0,spsave    ;User-Stackpointer retten
        pea        puf         ;Puffer-Adresse
        move       #1,-(sp)     ;1 Sektor
        move.l     #0,-(sp)     ;ab Sektor 0
        bsr        read        ;Sektor(en) in Puffer lesen
        move.l     spsave,-(sp)
        move       #$20,-(sp)
        trap       #1          ;in User-Mode schalten
        addq.l     #6,sp
        move.l     #head,d0
        bsr        pline       ;überschrift ausgeben
        move.l     #hi_cc,d0
        bsr        pmsg
        move       puf+$1b6,d0
        bsr        pword       ;Zylinder-Anzahl ausgeben
        bsr        pcrLf
        move.l     #hi_dhc,d0
        bsr        pmsg

```

```

move.b    puf+$1b8,d0
bsr       pbyt           ;Anzahl der Köpfe ausgeben
bsr       pcrLf
move.l    #hi_lz,d0
bsr       pmsg
move.b    puf+$1be,d0
bsr       pbyt           ;Landeposition ausgeben
bsr       pcrLf
move.l    #hi_rt,d0
bsr       pmsg
move.b    puf+$1bf,d0
bsr       pbyt           ;Seek-Rate ausgeben
bsr       pcrLf
move.l    #hi_in,d0
bsr       pmsg
move.b    puf+$1c0,d0
bsr       pbyt           ;Interleave-Faktor ausgeben
bsr       pcrLf
move.l    #hi_spt,d0
bsr       pmsg
move.b    puf+$1c1,d0
bsr       pbyt           ;Sektoren/Track ausgeben
bsr       pcrLf
move.l    #hd_size,d0
bsr       pmsg
move.l    puf+$1c2,d0
bsr       plong           ;Sektoren auf Harddisk
                                ;ausgeben
bsr       pcrLf
move.l    #bsl_count,d0
bsr       pmsg
move.l    puf+$1fa,d0
bsr       plong           ; tote Sektoren ausgeben
bsr       pcrLf
clr       d5
clr.l     d6
lea       puf+$1c6,a6      ;Partitions-Feld 0

loop:
bsr       pcrLf
move.b    d5,px_on
add.b     #'0',px_on
cmp.b     #0,0(a6,d6)      ;Partition aktiv ?
bne       pon              ;ja
move.l    #' aus',px_on+14 ;sonst 'aus' melden
move.l    #px_on,d0
bsr       pline
bra       nextp

pon:
move.l    #' an ',px_on+14
move.l    #px_on,d0
bsr       pline           ;'Partition an' ausgeben
and.b     #$80,0(a6,d6)   ;Boot-bar ?
beq       noboot          ;nein

```

```

                                move.l    #boot,d0
                                bsr        pline          ;sonst 'Boot-bar' ausgeben
nobot:
                                move.b    1(a6,d6),px_id+18
                                move      2(a6,d6),px_id+19
                                move.l    #px_id,d0
                                bsr        pline
                                move.l    #px_start,d0
                                bsr        pmsg
                                move.l    4(a6,d6),d0
                                bsr        plong          ;Startsektor ausgeben
                                bsr        pcrlf
                                move.l    #px_size,d0
                                bsr        pmsg
                                move.l    8(a6,d6),d0
                                bsr        plong          ;Sektoren/Track ausgeben
                                bsr        pcrlf
nextp:
                                addq       #1,d5
                                add        #12,d6
                                cmp        #4*12,d6
                                blt        loop
                                move      #1,-(sp)
                                trap       #1            ;warten auf Tastendruck
                                addq       #2,sp
                                clr        -(sp)
                                trap       #1            ;Rückkehr ins Desktop
read:  ; Sektor(en) lesen (wie oben !)
                                lea        wdc,a0
                                st         flock          ;Floppy-VBL-Routine sperren
                                move       #88,2(a0)      ;HDC-Zugriff, A1=0
                                nop
                                move.l    #$8008a,(a0)    ;read-Command
                                move.l    10(sp),-(sp)    ;Puffer-Adresse
                                bsr        setdma          ;DMA setzen
                                addq.l    #4,sp
                                bsr        set_parameters  ;Sektoranzahl und -Nummer
                                                    setzen
                                bmi        tout            ;Timeout aufgetreten !
                                move       #$190,2(a0)
                                nop
                                move       #$90,2(a0)    ;Umschalten auf READ
                                nop
                                move       8(sp),(a0)      ;Sector-Count an DMA übergeben
                                nop
                                move       #$8a,2(a0)
                                nop
                                move.l    #0,(a0)          ;Übertragung starten
                                bsr        waitl
                                bmi        tout
                                move       #$8a,2(a0)
                                move.l    (a0),d6          ;HDC/DMA-Status holen
                                and.l     #$ff00ff,d6      ;HI=HDC, LO=DMA

```

```

tout:
    move    #$80,2(a0)    ;auf FDC umschalten
    nop
    move.l  (a0),d7        ;Completion-Byte holen
    and.l   #$ff00ff,d7   ;HI=FDC, LO=DMA
    clr     flock         ;Floppy-VBL-Routine freigeben
    rts      ;fertig
set_parameters:    ;Sektor-Anzahl und Sector-Count setzen
    move    #$8a,2(a0)
    bsr     wait         ;warten auf HDC-OK
    bmi     setpx        ;Timeout !
    clr     d0
    move.b  4+5(sp),d0    ;Sektornr. HI
    bsr     send_byte
    bmi     setpx
    move.b  4+6(sp),d0    ;Sektornr. MID
    bsr     send_byte
    bmi     setpx
    move.b  4+7(sp),d0    ;Sektornr. LO
    bsr     send_byte
    bmi     setpx
    move    4+8(sp),d0    ;Anzahl der Sektoren
    bsr     send_byte

setpx:
    rts
send_byte:    ;1 Byte zum HDC senden
    swap    d0
    move     #$8a,d0
    move.l   d0,(a0)
    bra     wait
wait1:    ;max. 3 Sekunden auf OK warten
    move.l   #450000,count
    bra     wait1
wait:    ;max. 100 ms auf OK warten
    move.l   #15000,count
wait1:
    subq.l   #1,count
    bmi     timeout
    move.b   port,d0
    and.b    #$20,d0        ;HDC-Interrupt ?
    bne     wait1           ;nein
    moveq    #0,d0          ;ja => OK
    rts
timeout:
    move.l   #errline,d0
    bsr     pline
    moveq    #-1,d0        ;Timeout anzeigen
    rts
setdma:    ;DMA-Adresse setzen
    move.b   7(sp),dma+4    ;LO
    move.b   6(sp),dma+2    ;MID
    move.b   5(sp),dma      ;HI
    rts

```



```

; ** weitere Unterrountinen **
pline:  ; Print Line/CR
        bsr          pmsg
pcrlf:  ;Print CR,LF
        move         #10,d0
        bsr          pchar
        move         #13,d0
pchar:  ;Print Character D0
        move         d0,-(sp)
        move         #2,-(sp)
        trap         #1
        addq.l       #4,sp
        rts
pmsg:   ;Print Line (D0)
        move.l       d0,-(sp)
        move         #9,-(sp)
        trap         #1
        addq         #6,sp
        rts
plong:  ;D0 8-stellig hex ausgeben
        moveq        #7,d1
        bra          phexw11
pword:  ;Print Hex-Word D0
        swap         d0
        moveq        #3,d1
        bra          phexw11
pbyt:   ; Print Hex-Byte D0
        moveq        #1,d1
        ror.l        #8,d0
phexw11:
        rol.l        #4,d0
        move.l       d0,-(sp)
        move.l       d1,-(sp)
        bsr          phexnib
        move.l       (sp)+,d1
        move.l       (sp)+,d0
        dbra         d1,phexw11
        rts
phexnib:
        and.l        #$0f,d0
        add.b        #$30,d0
        cmp.b        #$3a,d0
        bcs          phexn1
        add.b        #7,d0
phexn1:
        bra          pchar ;Zeichen ausgeben
head:   dc.b         "*** Harddisk-Analyse 8/86 S.D. ***",0
hi_cc:  dc.b         "Zylinder      : ",0
hi_dhc: dc.b         "Köpfe         : ",0
hi_lz:  dc.b         "Park-Position : ",0
hi_rt:  dc.b         "Seek-Rate     : ",0
hi_in:  dc.b         "Interleave    : ",0
hi_spt: dc.b         "Sektoren/Track : ",0

```

```

hd size:  dc.b  "Gesamt-Sektoren : ",0
bsl_count: dc.b  "tote Sektoren   : ",0
even
px_on:    dc.b  "1. Partition :   ",0
boot:     dc.b  "Boot-bar ",0
px_id:     dc.b  "Partition-ID   : ",0
px_start: dc.b  "Start-Sektor   : ",0
px_size:   dc.b  "Sektoren-Anzahl : ",0
errline:   dc.b  "Timeout aufgetreten !",10,13,0
even
data
count:     dc.l  1                                ;Timeout-Counter
spsave:     dc.l  0                                ;User-Stackpointer
           blk.l  200
stp:       blk.l  1
puf:       blk.b  512                                ;Puffer für einen Sektor

```

Und auch gleich das BASIC-Lader-Programm, welches das Programm ANA-PART.TOS auf der Diskette erstellt:

```

10  '*** Erstellung von ANAPART.TOS ***
15  '
20  print
25  print "File >> ANAPART.TOS << wird erzeugt"
30  dim c%( 612):cs#=0
35  for i=0 to 612
40  read a$:c%(i)=val("%H"+a$)
45  check#+=check#+(c%(i))
50  next i
55  if check# = 6435851 then 70
60  print "Geht leider nicht, da etwas mit den DATA's nicht stimmt."
65  goto 80
70  bsave "anapart.tos",varptr(c%(0)), 1226
75  print "Das Programm >> anapart.tos << ist nun geschrieben."
80  print "Bitte Taste drücken":a=inp(2):end
85  '
90  '***** DATAs für anapart.tos *****
95  '
100 DATA 601A,0000,0484,0000,0000,0000,0000,0000
101 DATA 0000,0000,0000,0000,0000,0000,4FF9,0000
102 DATA 07AC,42A7,3F3C,0020,4E41,5C8F,23C0,0000
103 DATA 0488,4879,0000,07B0,3F3C,0001,2F3C,0000
104 DATA 0000,6100,01A0,2F39,0000,0488,3F3C,0020
105 DATA 4E41,5C8F,203C,0000,0354,6100,02AC,203C
106 DATA 0000,0377,6100,02BE,3039,0000,0966,6100
107 DATA 02C6,6100,0298,203C,0000,038A,6100,02A6
108 DATA 1039,0000,0968,6100,02B6,6100,0280,203C
109 DATA 0000,039D,6100,028E,1039,0000,096E,6100
110 DATA 029E,6100,0268,203C,0000,03B0,6100,0276
111 DATA 1039,0000,096F,6100,0286,6100,0250,203C
112 DATA 0000,03C3,6100,025E,1039,0000,0970,6100
113 DATA 026E,6100,0238,203C,0000,03D6,6100,0246

```

114 DATA 1039,0000,0971,6100,0256,6100,0220,203C
115 DATA 0000,03E9,6100,022E,2039,0000,0972,6100
116 DATA 0230,6100,0208,203C,0000,03FC,6100,0216
117 DATA 2039,0000,09AA,6100,0218,6100,01F0,4245
118 DATA 4286,4DF9,0000,0976,6100,01E2,13C5,0000
119 DATA 0410,0639,0030,0000,0410,0C36,0000,6800
120 DATA 6600,001A,23FC,2061,7573,0000,041E,203C
121 DATA 0000,0410,6100,01B2,6000,0070,23FC,2061
122 DATA 6E20,0000,041E,203C,0000,0410,6100,019A
123 DATA 0236,0080,6800,6700,000C,203C,0000,0423
124 DATA 6100,0186,13F6,6801,0000,043F,33F6,6802
125 DATA 0000,0440,203C,0000,042D,6100,016C,203C
126 DATA 0000,0445,6100,017E,2036,6804,6100,0182
127 DATA 6100,015A,203C,0000,0458,6100,0168,2036
128 DATA 6808,6100,016C,6100,0144,5245,0646,000C
129 DATA 0C46,0030,6D00,FF52,3F3C,0001,4E41,544F
130 DATA 4267,4E41,41F9,00FF,8604,50F9,0000,043E
131 DATA 317C,0088,0002,4E71,20BC,0008,008A,2F2F
132 DATA 000A,6100,00EA,588F,6100,0058,6B00,003C
133 DATA 317C,0190,0002,4E71,317C,0090,0002,4E71
134 DATA 30AF,0008,4E71,317C,008A,0002,4E71,20BC
135 DATA 0000,0000,6100,0076,6B00,0010,317C,008A
136 DATA 0002,2C10,0286,00FF,00FF,317C,0080,0002
137 DATA 4E71,2E10,0287,00FF,00FF,4279,0000,043E
138 DATA 4E75,317C,008A,0002,6100,0050,6B00,0030
139 DATA 4240,102F,0009,6100,0028,6B00,0022,102F
140 DATA 000A,6100,001C,6B00,0016,102F,000B,6100
141 DATA 0010,6B00,000A,302F,000C,6100,0004,4E75
142 DATA 4840,303C,008A,2080,6000,0010,23FC,0006
143 DATA DDD0,0000,0484,6000,000C,23FC,0000,3A98
144 DATA 0000,0484,53B9,0000,0484,6B00,0014,1039
145 DATA 00FF,FA01,0200,0020,6600,FFEA,7000,4E75
146 DATA 203C,0000,046B,6100,0020,70FF,4E75,13EF
147 DATA 0007,00FF,860D,13EF,0006,00FF,860B,13EF
148 DATA 0005,00FF,8609,4E75,6100,001A,303C,000A
149 DATA 6100,0006,303C,000D,3F00,3F3C,0002,4E41
150 DATA 588F,4E75,2F00,3F3C,0009,4E41,5C4F,4E75
151 DATA 7207,6000,000E,4840,7203,6000,0006,7201
152 DATA E098,E998,2F00,2F01,6100,000C,221F,201F
153 DATA 51C9,FF00,4E75,0280,0000,000F,0600,0030
154 DATA 0C00,003A,6500,0006,0600,0007,6000,FFAA
155 DATA 2A2A,2048,6172,6464,6973,6B2D,416E,616C
156 DATA 7973,6520,2038,2F38,3620,2053,2E44,2E20
157 DATA 2A2A,005A,796C,696E,6465,7220,2020,2020
158 DATA 2020,203A,2000,4B94,7066,6520,2020,2020
159 DATA 2020,2020,2020,3A20,0050,6172,6B2D,506F
160 DATA 7369,7469,6F6E,2020,203A,2000,5365,656B
161 DATA 2D52,6174,6520,2020,2020,2020,3A20,0049
162 DATA 6E74,6572,6C65,6176,6520,2020,2020,203A
163 DATA 2000,5365,6B74,6F72,656E,2F54,7261,636B
164 DATA 2020,3A20,0047,6573,616D,742D,5365,6B74
165 DATA 6F72,656E,203A,2000,746F,7465,2053,656B
166 DATA 746F,7265,6E20,2020,3A20,0000,312E,2050

```
167 DATA 6172,7469,7469,6F6E,203A,2020,2020,0042
168 DATA 6F6F,742D,6261,7220,0050,6172,7469,7469
169 DATA 6F6E,2D49,4420,2020,203A,2020,2020,2020
170 DATA 0053,7461,7274,2D53,656B,746F,7220,2020
171 DATA 203A,2000,5365,6B74,6F72,656E,2D41,6E7A
172 DATA 6168,6C20,3A20,0054,696D,656F,7574,2061
173 DATA 7566,6765,7472,6574,656E,2021,0A0D,0000
174 DATA 0000,0002,1006,140E,0A0A,0E0A,0E0A,0E0A
175 DATA 0E0A,0E0A,0E0A,0E0A,120A,0814,0612,0614
176 DATA 0C08,060A,16FC,0E06,1C00
```

Wie bereits im Kapitel über Boot-Sektoren beschrieben, gibt das "px_flag" für jede Partition (max. 4) an, ob sie aktiv und boot-bar ist. Die Harddisk des ATARI ST enthält üblicherweise keine boot-bare Partition, da der ST nicht von der Harddisk booten kann ohne den Harddisk-Treiber (AHDL.PRG).

Mit der Bezeichnung "Seek-Rate" wird normalerweise eine 2 ausgegeben, was 3 Millisekunden pro Schritt (Step) bedeutet. Interleave kann von 1 bis 16 (Sektoren/Track-1) sein, beträgt hier jedoch üblicherweise 1. Dies stellt den Abstand zwischen zwei der Nummer nach aufeinanderfolgenden Sektoren auf dem Track dar, ebenso wie bei Disketten.

Der Wert hinter "tote Sektoren" gibt die Anzahl der defekten Sektoren auf der gesamten Harddisk an. Diese Sektoren werden vom HDX.PRG-Programm erkannt und markiert. Eine 0 bedeutet hier, daß die Harddisk vollständig in Ordnung ist. Ansonsten ist ein defekter Sektor pro Megabyte noch ganz normal.

14.9 Die RAM-Disk

Als Dritter im Bunde der Massenspeicher für den ATARI ST steht die RAM-Disk. Eine solche scheinbare Diskettenstation im Speicher stellt eine interessante und vor allem sehr schnelle Möglichkeit der Datenspeicherung dar. Wie funktioniert das?

Zunächst einmal brauchen wir einen Speicherbereich, der von keiner anderen Anwendung des Computers verwendet werden kann. Dort hinein legen wir die Daten, die sonst auf eine Diskette geschrieben würden. Der Vorteil liegt auf der Hand: Daten im Speicher hin- und herzuschieben ist für den 68000-Prozessor des ST eine leichte Sache und geht daher mit einer enormen Geschwindigkeit vor sich. Zusätzlich entfallen alle mechanischen Vorgänge, die eine Diskettenstation bremsen (Koppositionierung, Motor anlaufen lassen usw.). Das Ergebnis: Eine RAM-Disk ist sehr schnell.

Was wir noch brauchen, ist ein Programm. Dieses Programm muß die Verwaltung des RAM-Disk-Speichers übernehmen und die Daten je nach Bedarf im Speicher verschieben. Solche Programme sind bereits mehrere auf dem Markt, einige sind auch in der Literatur zu finden. Sie arbeiten alle nach dem gleichen Prinzip.

Zuerst muß der als RAM-Disk zu verwendende Speicher initialisiert werden. Dazu muß ein Boot-Sektor erstellt werden, der alle Informationen über die Art, Aufteilung und Größe der "Disk" enthält. Dieser Sektor ist auf richtigen Disketten der allererste, also müssen diese Parameter an den Anfang des RAM-Disk-Speichers geschrieben werden.

Danach muß sich das Programm installieren, d. h. es muß Vorbereitungen treffen, damit es immer erfährt, ob eine Datenübertragung stattfinden soll und wenn ja, wohin und in welcher Richtung. Dies wird erreicht, indem drei Zeiger des Betriebssystems auf eigene Routinen gerichtet werden. Diese Zeiger sind Speicherzellen, in denen Adressen von Programmen stehen. Will das Betriebssystem ein solches Programm aufrufen, so wird der entsprechende Zeiger ausgelesen und zur erhaltenen Programmroutine verzweigt.

Die Zeiger, die bei der Installation einer RAM-Disk verwendet werden, sind für die Bedienung der Hard-Disk vorgesehen. Sie liegen an den Speicheradressen \$472 bis \$47E und zeigen auf Routinen folgender Bedeutungen:

Adresse	Name	Bedeutung
\$472	hdv_bpb	Feststellung und Übergabe des Parameterblockes, der Angaben über die Diskette bzw. Harddisk enthält. Schreib-/Leseroutine für die Harddisk. Dort wird der Datentransfer abgewickelt.
\$476	hdv_rw	
\$47A	hdv_boot	Boot-Routine für die Harddisk. Wird von der RAM-Disk nicht benötigt, da von ihr nicht gebootet werden kann.
\$47E	hdv_mediach	Feststellung, ob das Medium (Diskette) zwischenzeitlich gewechselt wurde.

Sind die Zeiger schließlich alle neu eingestellt und ihre alten Inhalte gerettet, so kann das Programm verlassen werden. Dafür wird jedoch ein spezieller Aufruf des BIOS verwendet, mit dem sich ein bestimmter Speicherplatz reservieren läßt. Damit ist die RAM-Disk schließlich installiert.

Nun muß man noch ein Diskettensymbol des Desktop für die RAM-Disk vorbereiten. Dazu klickt man irgendeines der Diskettensymbole an und ändert

nach Wahl des Menüpunktes "Disk anmelden" den Namen und den Kennbuchstaben der Diskettenstation. Nach Wahl des OK-Knopfes erscheint nun ein weiteres Symbol auf dem Bildschirm. Dieses kann nun nach gewohnter Manier zum Laden und Speichern von Daten und Programmen verwendet werden. Lediglich die Funktionen Formatieren und Disk-Copy funktionieren nicht, so daß nur einzelne Files bearbeitet oder gelöscht werden können.

Will nun das Betriebssystem auf die Hard- oder RAM-Disk zugreifen, so wird über einen der oben erwähnten Zeiger in das immer noch im Speicher stehende RAM-Disk-Programm gesprungen. Dort wird dann geprüft, ob sich die RAM-Disk angesprochen fühlen soll oder nicht. Wenn nicht, so wird zu der eigentlichen Routine verzweigt, deren Adresse ja gerettet wurde.

Ist doch die RAM-Disk gemeint, so beginnt das Programm mit seiner Arbeit. Bei einem Schreib-/Lesezugriff werden die Parameter wie Sektor, Anzahl der zu lesenden Sektoren und Datenübertragungs-Richtung vom Stack gelesen und die entsprechenden Daten im Speicher kopiert.

Handelt es sich um die "Media-Change"-Anfrage, bei der nach einem eventuellen Wechsel des Speichermediums gefragt wird, so gib das RAM-Disk-Programm grundsätzlich eine 0 zurück. Diese bedeutet, daß nichts gewechselt wurde, was ja bei einer RAM-Disk auch nicht möglich ist.

Die dritte Art des Aufrufes bedeutet, daß das Betriebssystem die Speicheradresse des Parameterblockes erfahren möchte. Dafür wird die gewünschte Adresse im Register D0 übergeben.

Das waren schon alle Aufgaben eines RAM-Disk-Programmes. Was es jedoch nicht kann, ist die Erhaltung der Daten nach dem Abschalten des Rechners. Das ist nämlich der Haken bei der Sache: wirklich abgespeichert sind die Daten nicht. Aus diesem Grunde müssen die Daten, die z.B. mit einem Texteditor erstellt wurden, unbedingt vor dem sicher verdienten Feierabend noch von der RAM-Disk auf eine wirkliche Diskette bzw. Harddisk kopiert werden!

Doch nun genug der bloßen Theorie. Nun wollen wir uns ein RAM-Disk-Programm ansehen, in dem alle diese Sachen vorkommen.

14.9.1 Ein komfortables RAM-Disk-Programm

Das in diesem Kapitel vorgestellte Programm enthält einige Merkmale, die für die reine Anwendung einer RAM-Disk eigentlich nicht nötig wären. Da sie jedoch recht nützlich sind, ist das Programm zwar etwas umfangreicher, dafür

aber komfortabler. Es ist für die Verwendung der RAM-Disk als Laufwerk C vorgesehen, läßt sich jedoch auch leicht auf eine andere Laufwerksnummer anpassen.

Das Programm ist ein Accessory, welches nach dem Booten in dem "Desk"-Menü unter dem Punkt "RAM-Disk" auftaucht. Wird dieser Menüpunkt angeklickt, so öffnet sich ein kleines Dialogfenster, welches drei Auswahlpunkte beinhaltet.

Der erste Punkt, welcher auch stark umrandet ist, trägt die Bezeichnung "Exit". Wenn man nun diesen Punkt anklickt oder auch nur die Return-Taste betätigt, so wird das Fenster gelöscht, und das Desktop meldet sich wieder. Passiert ist dabei gar nichts. Dieser Punkt ist nur dafür vorgesehen, wenn der Menüeintrag "RAM-Disk" versehentlich angewählt wurde. Dann ist "Exit" der Notausgang.

Im mittleren Auswahlpunkt steht die Inschrift "mehr". Ein Anklicken dieses Knopfes verändert die Zahl in dem rechten Auswahlkästchen. Diese Zahl bedeutet die Größe der zu installierenden RAM-Disk. Durch Anwahl von "mehr" wird diese Zahl in 100er-Schritten erhöht, bis nach dem Wert 800 wieder eine Null angeboten wird.

Hat man die gewünschte RAM-Disk-Kapazität eingestellt, so wählt man den Knopf mit der Zahl an. Da für die Installation eines neuen Speicherbereiches der alte Inhalt der RAM-Disk gelöscht wird, erscheint zur Sicherheit ein weiteres Dialog-Fenster. In diesem Fenster muß auf die Frage "Alten Inhalt der RAM-Disk löschen?" mit Ja geantwortet werden; andernfalls bleibt die alte RAM-Disk mit ihrer alten Kapazität und Belegung erhalten.

Nachdem durch die Wahl von "Ja" nun alle Einstellungen erledigt sind, macht sich das Programm an die Arbeit. Als erstes wird der Speicherbereich, den die RAM-Disk vorher belegt hat, wieder an das Betriebssystem zurückgegeben.

Danach versucht das Programm, den gewünschten Speicherbereich für sich zu reservieren. Sollte nicht genügend Speicherplatz vorhanden sein, so erhält man die Meldung "Nicht genug RAM". Nachdem diese Meldung quittiert wurde, ist sowohl die Meldung als auch die RAM-Disk verschwunden. Man muß dann einen kleineren Speicher auswählen, indem man nach erneuter Wahl des "RAM-Disk"-Menüpunktes so oft die "mehr"-Taste betätigt, bis der neue gewählte Wert erscheint.

Wählt man als Kapazität der RAM-Disk die Null an, so wird sie vollständig abgemeldet und belegt somit keinen Speicher mehr. Somit ist man mit diesem Programm in der Lage, seine RAM-Disk je nach Bedarf beliebig oft in der

Größe zu verändern oder ab- und anzumelden. Diese Möglichkeit haben die meisten auf dem Markt befindlichen RAM-Disk-Programme nicht, und bei häufigem Arbeiten mit diesem Programm werden Sie die Vorteile zu schätzen wissen.

Noch ein Punkt sollte erwähnt werden, bevor wir uns das Programm selbst ansehen. Da eine RAM-Disk nicht formatiert werden kann (bitte erst gar nicht probieren, da dies die Diskettenstationen ansprechen kann...), muß zum Löschen einer solchen "Disk" jedes File einzeln gelöscht werden. Bei dem vorliegenden Programm jedoch brauchen Sie lediglich in dem Dialogfenster dieselbe Kapazität auszuwählen, und schon ist die gesamte RAM-Disk leer.

Doch nun zum Programm:

```
,***** RAM-Disk mit Komfort S.D. *****
hdv_bpb      = $472
hdv_rw       = $476
hdv_mediach  = $47e
drvbits      = $4c2
start:
    move.l    #nstapel,a7      ;neuen Stack einstellen
    move      #10,opcode      ;appl_init
    move      #0,sintin
    move      #1,sintout
    move      #0,saddrin
    move      #0,saddrout
    bsr       aes
    move      intout,appid     ;Application-ID
    move      #77,opcode      ;graf_handle
    move      #5,sintout
    move      #0,saddrin
    move      #0,saddrout
    bsr       aes
    move      intout,grhandle  ;Graphic-Handle
    move      #35,opcode      ;Menu_Register
    move      #1,sintin
    move      #1,sintout
    move      #1,saddrin
    move      appid,intin
    move.l    #accname,addrin
    bsr       aes
    move      intout,accid     ;Accessory-Nummer
; ** Ab hier die Bereitschafts-Schleife **
loop:
    bsr       event           ;Event_Multi
    cmp       #40,msgbuff     ;Acc_open ?
    bne       loop           ;nein
    move      msgbuff+8,d0
    cmp       accid,d0        ;unsere Accessory-Nummer ?
    bne       loop           ;nein
    bsr       run             ;Menü darstellen
```



```

        addq.l    #6,sp
        rts
fehler:
        move.l    #error,addrin
        bsr      formalert      ;'Nicht genug RAM !'
        bra      ende           ;Abbruch
init:
        move.l    hdv_bpb,bpbsave ;alte Vektoren retten
        move.l    #bpb,hdv_bpb
        move.l    hdv_rw,rwsave  ;Vektoren auf neue Routinen
                                   ;setzen
        move.l    #rw,hdv_rw
        move.l    hdv_mediasave,hdv_mediasave
        move.l    #media,hdv_mediasave
        move.l    puffer,a0
        move.l    #10240/4,d0
iloop1:
        clr.l     (a0)+          ;Boot-Sektor + FATs löschen
        dbra     d0,iloop1
        ; * Boot-Sektor generieren *
        move.l    puffer,a0
        add.l     #11,a0         ;ab Puffer+11
        lea      boottab,a1
        moveq     #tabend-boottab-1,d0
bloop:
        move.b    (a1)+,(a0)+    ;Daten in Boot-Sektor kopieren
        dbra     d0,bloop
        move      kapazi,d7
        move      d7,numcl       ;Kapazität in KByte in BPB
        lsl      #1,d7          ;Kapazität in Sektoren
        add       #18,d7        ;plus 18 Sektoren
        move.l    puffer,a0
        add.l     #19,a0         ;in Puffer+19 und +20
        move.b    d7,(a0)+      ;LO
        lsr      #8,d7
        move.b    d7,(a0)       ;HI
        bset     #2,drvbits+3   ;Drive C anmelden
        rts          ;fertig
; * Funktion: Get BPB *
bpb:
        cmp      #2,4(sp)       ;Drive C ?
        beq      bpb1          ;ja
        move.l    bpbsave,a0    ;alte Routine
        jmp      (a0)           ;aufrufen
bpb1:
        move.l    #bpbtab,d0    ;Zeiger auf BIOS-Parame-
                                   ;ter-Block
        rts
; * Funktion: Read/Write *
rw:
        cmp      #2,14(sp)      ;Drive C ?
        beq      rw1          ;ja
        move.l    rwsave,a0     ;alte Routine
        jmp      (a0)           ;aufrufen
rw1:
        move     12(sp),d0      ;reco, logische Sektor-Nummer
        ext.l     d0

```

```

        lsl.l      #8,d0
        lsl.l      #1,d0          ;mal 512
        move.l     6(sp),a0       ;Puffer-Adresse
        move       10(sp),d1      ;Anzahl Sektoren
        subq       #1,d1
        move.l     puffer,a1      ;Basis-Adresse
        add.l      d0,a1          ;plus relative Adresse in
                                ;RAM-Disk
        move       4(sp),d0       ;rwflag
        btst       #0,d0          ;lesen ?
        beq        rloop0        ;ja
        exg        a0,a1          ;Ziel und Quelle tauschen
rloop0:  move.l     #511,d0        ;ein Sektor
rloop:   move.b     (a1)+,(a0)+   ;Puffer kopieren
        dbra       d0,rloop0
        dbra       d1,rloop0      ;nächster Sektor
        clr        d0            ;OK
        rts
; * Funktion: Media-Change *
media:   cmp        #2,4(sp)      ;Drive C ?
        beq        medial        ;ja
        move.l     mediasave,a0  ;alte Routine
        jmp        (a0)          ;aufrufen
medial:  move       changed,d0    ;Diskette evtl. gewechselt
        clr        changed       ;aber nur einmal
        rts
event:   move       #25,opcode    ;Event_Multi, GEM-Ereignis
                                ;feststellen
        move       #16,sintin
        move       #7,sintout
        move       #1,saddrin
        move.l     #msgbuff,addrin
        lea        table,a1
        lea        intin,a2
        moveq      #15,d0
lopl:    move       (a1)+,(a2)+   ;Parameter setzen
        dbra       d0,lopl
        bsr        aes
        rts
aes:     ; AES-Aufruf
        move.l     #aespb,d1
        move       #$c8,d0
        trap       #2
        rts
mfree:   ; Speicher freigeben
        tst.l      puffer
        beq        ende          ;ist bereits abgemeldet
        move.l     #reinit,-(sp)
        move       #38,-(sp)     ;Reinitialisierung
        trap       #14           ;im Supervisor-Modus
        addq.l     #6,sp

```

```

move.l    puffer,-(sp)
move      #$49,-(sp)          ;MFREE-Funktion, Speicher
                                ;freigeben

trap      #1
addq.l    #6,sp
tst.l     d0                  ;Error?
beq       ende                ;nein
move.l    #error1,addrin
bsr       formalert           ;Error-Meldung

ende:

clr.l     puffer              ;kein Speicher mehr reserviert
rts

reinit:

move.l    bpbsave,hdv_bpb
move.l    rwsave,hdv_rw       ;Vektoren auf alte
                                ;Routinen setzen

move.l    mediasave,hdv_mediach
bclr      #2,drvbits+3        ;Drive C abmelden
rts

formalert:

move      #52,contrl          ;form alert, Alarmfenster
                                ;darstellen

move      #1,contrl+2
move      #1,contrl+4
move      #1,contrl+6
move      #0,contrl+8
move      #1,intin
bsr       aes
rts

table:    dc.w $13,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
accname:   dc.b " RAM-Disk ",0
even
wieviel:   dc.b "[1][Wieviel RAM-Disk-KByte ?]"
dc.b "[Exit| mehr |"
offer:     dc.b " 100 |",0,0
clear:     dc.b "[1][Alten Inhalt der| RAM-Disk löschen ?]"
dc.b "[ Ja ! | Nein ]",0,0
error:     dc.b "[2][Nicht genug RAM !]"
dc.b "[Dann nicht..]",0,0
error1:    dc.b "[2][Fehler bei MFREE !]"
dc.b "[Dann nicht..]",0,0
even
kapazi:    dc.w 100
size:      dc.w 2
sizes:     dc.w 0,100,200,300,400,500,600,700,800
dezi:      dc.b ' 0 100 200 300 400 500 600 700 800'
puffer:    dc.l 0              ;RAM-Disk-Puffer-Adresse
changed:   dc.l 0              ;Flag für 'Diskette gewechselt'
bpbtabs:   dc.w $200           ;Sektorgroße
clsiz:     dc.w 2              ;Cluster-Größe in Sektoren
clsizb:    dc.w $400           ;Cluster-Größe in Bytes
rdlen:     dc.w 7              ;Directory-Länge in Sektoren

```

```

fsiz:      dc.w 5          ;FAT-Größe
fatrec:    dc.w 6          ;FAT-Sektoren
datrec:    dc.w 18         ;Sektoren für Verwaltung
numcl:     blk.w 1         ;Kapazität in KByte
flags:     dc.l 0,0,0,0
boottab:   ; Daten in 8086-format
           dc.b 0,2        ;Bytes pro Sektor
           dc.b 2          ;Sektors pro Cluster
           dc.b 1,0        ;reserved Sektors
           dc.b 2          ;FATs
           dc.b 112,0      ;Directory entries
           blk.b 2         ;Sectors on media
           dc.b 0          ;media descriptor
           dc.b 5,0        ;Sectors pro FAT
           dc.b 9,0        ;Sectors pro Track
           dc.b 1,0        ;Sides
           dc.b 0          ;hidden

tabend:
even
bpbsave:   blk.l 1         ;Platz für alte Vektoren
rwsave:    blk.l 1
mediasave: blk.l 1
aespb:     dc.l contrl,global,intin,intout,addrin,addrout
data
choice:    blk.w 1         ;Auswahl
grhandle:  blk.w 1
appid:     blk.w 1         ;Application-ID
accid:     blk.w 1         ;Accessory-Unit
msgbuff:   blk.b 16
           blk.l 128       ;neuer Stack
nstapel:   blk.l 1
contrl:    ;GEM-Parameter-Block
opcode:    blk.w 1
sintin:    blk.w 1
sintout:   blk.w 1
saddrin:   blk.w 1
saddrout:  blk.l 1
           blk.w 5
global:    blk.l 8
intin:     blk.w 80
ptsin:     blk.w 80
intout:    blk.w 80
ptsout:    blk.w 80
addrin:    blk.w 80
addrout:   blk.w 80

```

Dieses Programm wurde auf dem Macro-Assembler SEKA erstellt, wodurch einige Punkte anders aussehen als bei Programmen für den DRI-Assembler, der in dem ATARI-Entwicklungspaket enthalten ist. Zu ändern wären jedoch

lediglich die Kommentarzeilen, die für den DRI-Assembler mit einem Sternchen (*) beginnen müssen, und die "blk"-Anweisung, die bei DRI "ds" lauten muß.

Das Programm ist in mehrere Teile untergliedert:

1. Anmeldung des Accessories.
2. Bereitschafts-Schleife, die im normalen Betrieb des ATARI ST ständig im Hintergrund mitläuft und daher kein Ende haben darf.
3. Auswahl-Dialogfenster darstellen und bedienen, dabei die gewählte Kapazität in "kapazi" ablegen.
4. Sicherheits-Abfrage darstellen und bearbeiten.
5. Vorher benutzten Speicher abmelden (MFREE).
6. Neuen Speicher reservieren, ggf. Fehlermeldung ausgeben.
7. BIOS-Vektoren für die Disk-Routinen retten und neue Vektoren einstellen.
8. Get BPB-Funktion.
9. Read/Write-Funktion.
10. Media Change-Funktion.
11. Datenfelder für Parameterblöcke.

Die Punkte 7 bis 10 wurden bereits im vorigen Kapitel besprochen. Die Funktionen der Punkte 1 bis 6 sind für die Beschreibung an dieser Stelle zu umfangreich. Die vollständige Beschreibung der verwendeten Funktionen finden Sie im Teil D.

Und hier wieder ein BASIC-Lader-Programm, welches das Accessory-Programm RAMDISK.ACC auf der Diskette erstellt:

```
10  '*** Erstellung des RAM-Disk-Accessories ***
15  '
20  print
25  print "File >> RAMDISK.ACC << wird erzeugt"
30  dim c%( 741):cs#=0
35  for i=0 to 741
40  read a$:c%(i)=val("&H"+a$)
45  check#=check#+(c%(i))
50  next i
55  if check#= 5104824 then 70
60  print"Geht leider nicht, da etwas mit den DATAs nicht stimmt."
65  goto 80
70  bsave "RAMDISK.ACC",varptr(c%(0)), 1483
75  print "Das Programm >> RAMDISK.ACC << ist nun geschrieben."
```

```
80 print"Bitte Taste drücken":a=inp(2):end
85 '
90 '***** DATAs für RAMDISK.ACC *****
95 '
100 DATA 601A,0000,0546,0000,0000,0000,0000,0000
101 DATA 0000,0000,0000,0000,0000,0000,2E7C,0000
102 DATA 0B60,33FC,000A,0000,0546,33FC,0000,0000
103 DATA 0548,33FC,0001,0000,054A,33FC,0000,0000
104 DATA 054C,33FC,0000,0000,054E,6100,02F6,33F9
105 DATA 0000,06BC,0000,094C,33FC,004D,0000,0546
106 DATA 33FC,0005,0000,054A,33FC,0000,0000,054C
107 DATA 33FC,0000,0000,054E,6100,02C8,33F9,0000
108 DATA 06BC,0000,094A,33FC,0023,0000,0546,33FC
109 DATA 0001,0000,0548,33FC,0001,0000,054A,33FC
110 DATA 0001,0000,054C,33F9,0000,094C,0000,057C
111 DATA 23FC,0000,03F4,0000,07FC,6100,0286,33F9
112 DATA 0000,06BC,0000,094E,6100,0234,0C79,0028
113 DATA 0000,0950,6600,FFF2,3039,0000,0958,B079
114 DATA 0000,094E,6600,FFE2,6100,0006,6000,FFDA
115 DATA 23FC,0000,0402,0000,07FC,6100,02BE,33F9
116 DATA 0000,06BC,0000,0948,0C79,0001,0000,0948
117 DATA 6700,0278,0C79,0003,0000,0948,6700,0044
118 DATA 5479,0000,04BE,0C79,0012,0000,04BE,6000
119 DATA 0008,4279,0000,04BE,41F9,0000,04C0,4280
120 DATA 3039,0000,04BE,33F0,0800,0000,04BC,E348
121 DATA 41F9,0000,04D2,23F0,0800,0000,042C,6000
122 DATA FF90,23FC,0000,0434,0000,07FC,6100,024C
123 DATA 0C79,0002,0000,06BC,6700,0010,6100,01D2
124 DATA 4A79,0000,04BE,6600,0004,4E75,33FC,0002
125 DATA 0000,0544,4287,3E39,0000,04BC,0687,0000
126 DATA 0009,EB87,EB87,2F07,3F3C,0048,4E41,5C8F
127 DATA 4A80,6700,0018,23C0,0000,04F6,2F3C,0000
128 DATA 01C2,3F3C,0026,4E4E,5C8F,4E75,23FC,0000
129 DATA 046E,0000,07FC,6100,01E2,6000,01AE,23F9
130 DATA 0000,0472,0000,093C,23FC,0000,0258,0000
131 DATA 0472,23F9,0000,0476,0000,0940,23FC,0000
132 DATA 0272,0000,0476,23F9,0000,047E,0000,0944
133 DATA 23FC,0000,02C2,0000,047E,2079,0000,04F6
134 DATA 203C,0000,0900,4298,51C8,FFFC,2079,0000
135 DATA 04F6,D1FC,0000,000B,43F9,0000,0532,7011
136 DATA 10D9,51C8,FFFC,3E39,0000,04BC,33C7,0000
137 DATA 0520,E34F,0647,0012,2079,0000,04F6,D1FC
138 DATA 0000,0013,10C7,E04F,1087,08F9,0002,0000
139 DATA 04C5,4E75,0C6F,0002,0004,6700,000A,2079
140 DATA 0000,093C,4ED0,203C,0000,0512,4E75,0C6F
141 DATA 0002,000E,6700,000A,2079,0000,0940,4ED0
142 DATA 302F,000C,48C0,E188,E388,206F,0006,322F
143 DATA 000A,5341,2279,0000,04F6,D3C0,302F,0004
144 DATA 0800,0000,6700,0004,C348,203C,0000,01FF
145 DATA 10D9,51C8,FFFC,51C9,FFF2,4240,4E75,0C6F
146 DATA 0002,0004,6700,000A,2079,0000,0944,4ED0
147 DATA 3039,0000,0544,4279,0000,0544,4E75,33FC
148 DATA 0019,0000,0546,33FC,0010,0000,0548,33FC
```

```

149 DATA 0007,0000,054A,33FC,0001,0000,054C,23FC
150 DATA 0000,0950,0000,07FC,43F9,0000,03D4,45F9
151 DATA 0000,057C,700F,34D9,51C8,FFFC,6100,0004
152 DATA 4E75,223C,0000,04FA,303C,00C8,4E42,4E75
153 DATA 4AB9,0000,04F6,6700,0032,2F3C,0000,0376
154 DATA 3F3C,0026,4E4E,5C8F,2F39,0000,04F6,3F3C
155 DATA 0049,4E41,5C8F,4A80,6700,0010,23FC,0000
156 DATA 0494,0000,07FC,6100,0032,42B9,0000,04F6
157 DATA 4E75,23F9,0000,093C,0000,0472,23F9,0000
158 DATA 0940,0000,0476,23F9,0000,0944,0000,047E
159 DATA 08B9,0002,0000,04C5,4E75,33FC,0034,0000
160 DATA 0546,33FC,0001,0000,0548,33FC,0001,0000
161 DATA 054A,33FC,0001,0000,054C,33FC,0000,0000
162 DATA 054E,33FC,0001,0000,057C,6100,FF56,4E75
163 DATA 0013,0001,0001,0001,0000,0000,0000,0000
164 DATA 0000,0000,0000,0000,0000,0000,0000,0000
165 DATA 2020,5241,4D2D,4469,736B,2043,0000,5B31
166 DATA 5D5B,5769,6576,6965,6C20,5241,4D2D,4469
167 DATA 736B,2D4B,4279,7465,203F,5D5B,4578,6974
168 DATA 7C20,6D65,6872,207C,2031,3030,205D,0000
169 DATA 5B31,5D5B,416C,7465,6E20,496E,6861,6C74
170 DATA 2064,6572,7C20,5241,4D2D,4469,736B,206C
171 DATA 9473,6368,656E,203F,5D5B,204A,6120,2120
172 DATA 7C20,4E65,696E,205D,0000,5B32,5D5B,4E69
173 DATA 6368,7420,6765,6E75,6720,5241,4D20,215D
174 DATA 5B44,616E,6E20,6E69,6368,742E,2E5D,0000
175 DATA 5B32,5D5B,4665,686C,6572,2062,6569,204D
176 DATA 4652,4545,2021,5D5B,4461,6E6E,206E,6963
177 DATA 6874,2E2E,5D00,0000,0064,0002,0000,0064
178 DATA 00C8,012C,0190,01F4,0258,02BC,0320,2020
179 DATA 3020,2031,3030,2032,3030,2033,3030,2034
180 DATA 3030,2035,3030,2036,3030,2037,3030,2038
181 DATA 3030,0000,0000,0000,0546,0000,055C,0000
182 DATA 057C,0000,06BC,0000,07FC,0000,089C,0200
183 DATA 0002,0400,0007,0005,0006,0012,FFFF,0000
184 DATA 0000,0000,0000,0000,0000,0000,0000,0002
185 DATA 0201,0002,7000,FFFF,0005,0009,0001,0000
186 DATA 0000,0000,0002,0808,0808,080A,0408,0808
187 DATA 080A,0408,0808,0806,0406,040A,040C,0A06
188 DATA 1204,0A04,080C,0A08,0A06,0808,0808,0A04
189 DATA 0C0E,0E08,2006,1004,1206,0E06,0E06,0A12
190 DATA 0C0E,060C,2608,121C,3408,060A,0808,0806
191 DATA 0406,0614,0E0A,0E14,040A,080A,0A16,0808
192 DATA 0808,0801,3204,0404,0404,0000

```

14.10 Automatisches RAM-Disk-Kopierprogramm

Installiert sich Ihre RAM-Disk beim Boot-Vorgang automatisch, so wäre es doch sehr komfortabel, wenn die Programme, mit denen man ständig arbeitet (Textverarbeitung, Compiler, Linker etc.), auch direkt bei diesem Boot-Vor-

gang von der Boot-Diskette in die RAM-Disk kopiert würden. Ein derartiges Kopierprogramm und dessen Arbeitsweise wird im Folgenden beschrieben.

Die groben Arbeitsschritte sind dabei folgende:

1. Es muß eine Textdatei, die die Namen der zu kopierenden Dateien enthält, erzeugt werden (dies kann mit jeder Textverarbeitung geschehen), die in unserem Programm den Namen LISTE.LST haben muß und sich im Wurzelverzeichnis der Boot-Diskette befindet.
2. Die zu kopierenden Dateien müssen ebenfalls auf der Boot-Diskette im Wurzelverzeichnis vorliegen.
3. Nach dem Start wird die Verzeichnisdatei LISTE.LST geöffnet und in einen Puffer eingelesen. Danach wird ein RAM-Puffer angelegt, der die Dateien gemäß der Verzeichnisdatei der Reihe nach aufnimmt und aus dem sie dann in die RAM-Disk übertragen werden. (Der RAM-Puffer hat in unserem Programm eine Größe von 200 KByte, kann aber je nach Größe des Arbeitsspeichers beliebig erweitert werden.)
4. Sind alle Dateien in die RAM-Disk kopiert worden, wird der reservierte Pufferbereich wieder freigegeben und das Programm beendet.

Das Programm beginnt mit der Definition der Struktur, die uns als DTA-Puffer dienen soll und in die die Ffirst-Funktion die von uns benötigten dateispezifischen Informationen ablegt.

Danach wird ein Puffer (datpuffer) angelegt, in den der Inhalt der Verzeichnisdatei LISTE.LST eingelesen werden soll. Da es an dieser Stelle gut paßt, wollen wir hier kurz den Aufbau dieser Datei beschreiben:

In der ersten Zeile wird die Kennung der RAM-Disk gefolgt von einem Doppelpunkt und einem Backslash angegeben, z.B. F:\. Ab der zweiten Zeile folgen nun die zu kopierenden Dateien, bei denen ebenfalls die Laufwerkskennung vorangestellt werden muß, z.B.

```
A:\PROGRAMM1.PRG
A:\PROGRAMM2.PRG
```

Eine komplette Verzeichnisdatei sähe dann folgendermaßen aus:

```
F:\
A:\PROGRAMM1.PRG
A:\PROGRAMM2.PRG
```

Dies zum Aufbau der Verzeichnisdatei. Doch nun weiter mit dem Programmbau. Es wird mit der Funktion `such_file()` überprüft, ob sich die Verzeichnisdatei auf der Boot-Diskette befindet. Dazu wird der DTA-Puffer mit der Funktion `Fsetdata` festgelegt. Mit der Funktion `Fsfirst` wird nun nach der Datei, die unter File-Name übergeben wurde, hier `LISTE.LST`, im aktuellen Verzeichnis, hier das Wurzelverzeichnis, gesucht. Wurde die Datei gefunden, ist der Funktionswert 0, andernfalls kleiner 0. Dies zur Funktionsweise der Funktion `such_file()`.

Wurde also die Verzeichnisdatei nicht gefunden, wird das Programm mit einer Fehlermeldung abgebrochen, ansonsten wird nun die Größe dieser Datei mittels der Informationen, die durch `Fsfirst` in den DTA-Puffer geladen wurden, ermittelt. Die Größe wurde von uns auf 4000 Byte beschränkt, kann aber beliebig verändert werden. Auch hierbei wird bei Überschreitung dieser vorgegebenen Größe eine Fehlermeldung ausgegeben und das Programm beendet.

Ist bis hierhin alles glatt gelaufen, reservieren wir nun mittels `"Malloc"` einen 200 KByte großen RAM-Puffer, der anschließend die zu kopierenden Dateien zwischenspeichern muß. Sollte der freie Arbeitsspeicher zu klein sein, wird eine entsprechende Fehlermeldung ausgegeben und das Programm beendet. Die Größe dieses RAM-Puffers ist natürlich auch von Ihnen frei wählbar. Dazu brauchen Sie nur Ihre gewünschte RAM-Puffergröße als Long-Wert in die `Malloc`-Funktion einzutragen.

Durch `"Fopen"` wird nun die Verzeichnisdatei geöffnet, mit `"Fread"` in den vorgesehenen Puffer `datapuffer` eingelesen und durch `"Fclose"` wieder ordnungsgemäß geschlossen.

Durch zwei Schleifen wird nun nacheinander jede Zeile der Verzeichnisdatei ermittelt und in das C-String-Format umgewandelt. (Es wird an jede ermittelte Zeile ein Stringendezeichen `\0` angehängt.) Schließlich wird ein Zeiger auf den jeweiligen Zeilenanfang an die Funktion `"filecopy()"` übergeben. Dieser Vorgang wird so lange wiederholt, bis alle Zeilen abgearbeitet wurden, d.h. bis der Laufindex `index1` gleich der Dateilänge `laenge` ist.

Die Funktion `"filecopy()"` überprüft zunächst mittels der Funktion `"suche-file()"`, deren Arbeitsweise bereits beschrieben wurde, ob die übergebene Datei sich auch auf der Boot-Diskette befindet. Bei nicht erfolgreicher Suche wird eine entsprechende Fehlermeldung ausgegeben.

Ist die Datei vorhanden, wird aus den Informationen des DTA-Puffers, der ja durch die `"Fsfirst"`-Funktion mit den dateispezifischen Daten gefüllt wurde, die jeweilige Größe der gerade zu bearbeitenden Datei ermittelt.

Danach wird durch die Funktionen "Fopen", "Fread" und "Fclose" die Datei in den RAM-Puffer gelesen.

Um die im RAM-Puffer befindliche Datei nun in die RAM-Disk zu kopieren, muß sie mit der "Fcreate"-Funktion auf der RAM-Disk erzeugt werden. Da hierzu der komplette Dateipfad, der ja aus Laufwerkskennung plus Pfad- und Dateiname besteht, notwendig ist, muß der vorliegende Dateipfad abgeändert werden, weil hier als Laufwerkskennung das Boot-Laufwerk eingetragen wurde (in unserem obigen Beispiel A:\PROGRAMM1.PRG also die Kennung A). Wir brauchen nun nur diese Kennung A mit der für die RAM-Disk entsprechende (in unserem Beispiel F) Kennung auszutauschen. Da in der ersten Zeile der Verzeichnisdatei die Kennung des Ziellaufwerkes angegeben wurde, steht diese Kennung also an der ersten Stelle des Dateipuffers datapuffer. Da der Name eines Arrays auch gleichzeitig die Adresse auf das erste Element des Arrays ist, brauchen wir nur den Inhalt dieses Elementes an die entsprechende Stelle der momentanen Zeile zu übertragen. Dies wird mit der Programmzeile

```
*File-Name=*datapuffer
```

erreicht. Somit ist der Dateipfad komplett, und die Datei kann mittels Fcreate auf der RAM-Disk erzeugt werden. Mit der so erhaltenen Handle-Nummer wird mit der Funktion Fwrite die Datei aus dem RAM-Puffer in die erzeugte Datei auf der RAM-Disk geschrieben. Nachdem dann die Datei noch mit "Fclose" geschlossen wurde, liegt sie zur weiteren Benutzung auf der RAM-Disk vor. Dieser Vorgang wird für jede Zeile in der Verzeichnisdatei durchgeführt, bis alle Programme kopiert wurden.

Mit dem nun folgenden Programm-Listing und den obigen Erklärungen haben wir hoffentlich etwas Licht in die Vorgehensweise eines einfachen Kopierprogramms gebracht und Ihnen die Arbeit beim Umgang mit der RAM-Disk ein wenig erleichtert.

```
/* Auto-Ramdisk-Copyprogramm */
#include<stdio.h>
#include<osbind.h>
#include<ctype.h>
typedef struct dtarec          /*DTA-Puffer, in den der Dateieintrag
                               eingelesen wird*/
{
    char reserved[21];          /*reserviert*/
    char attribut;              /*Dateiattribut*/
    int time;                   /*Erstellungs-Uhrzeit*/
    int date;                   /*Erstellungs-Datum*/
    long size;                  /*Dateigröße*/
    char name[14];              /*Dateiname*/
} DTAREC;
```

```

DTAREC dtabuffer;
char datpuffer[1000];
long rampuffer;

/*Puffer für die Verzeichnisdatei*/
/*Puffer für die zu kopierenden
Dateien*/

main()
{
    long laenge;
    int fhandle;
    index1;
    index2;
    ret;
    char liste[10];

    /*Länge der Verzeichnisdatei*/
    /*Dateihandle der Verzeichnisdatei*/
    /*erster Laufindex*/
    /*zweiter Laufindex*/
    /*Rückgabewert der Dateisuchfunktion*/
    /*String für den Namen der Verzeichnis-
    datei*/

    index1=3;
    printf("Ramdisk-Copy aktiv !\n"); /*Programm beginnt*/
    strcpy(liste,"LISTE.LST"); /*Name der Verzeichnisdatei*/
    ret=suche_file(liste); /*Datei vorhanden ?*/
    if(ret!=0) /*nein, dann*/
        exit(1); /*Programm beenden*/
    else /*ja, dann*/
    {
        laenge=dtabuffer.size; /*Länge der Verzeichnisdatei
        ermitteln*/
        if(laenge>1000) /*falls Länge größer als Puffer, dann*/
        {
            printf("\nIhre Verzeichnis-Liste ist zu lang !\n");
            printf("\nTaste\n");
            Cconin(); /*auf Tastendruck warten*/
            exit(1); /*Programm beenden*/
        }
    }
    rampuffer=Malloc(51200L); /*Anfangsadresse des 200K Dateien-
    puffers ermitteln*/
    if(rampuffer==0L) /*falls Speicherplatz zu klein, dann*/
    {
        printf("\nSpeicherplatz zu klein !\n");
        printf("\nTaste\n");
        Cconin(); /*auf Tastendruck warten*/
        exit(1); /*Programm beenden*/
    }
    fhandle=Fopen(liste,0); /*öffnen der Verzeichnisdatei und
    Handle-übergabe*/
    Fread(fhandle,laenge,datpuffer);/*Verzeichnisdatei in Puffer
    einlesen*/
    Fclose(fhandle); /*Verzeichnisdatei schließen*/
    while(index1 < laenge) /*solange erster Laufindex < Verzeichnis-
    datei-Länge*/
    {
        while(((datpuffer+index1) < 32) && (index1 < laenge))
            ++index1; /*ersten Laufindex erhöhen*/
        index2=index1; /*Position des ersten Laufindex sichern*/
        if(index1 < laenge) /*falls erster Laufindex < Verzeichnis-
        datei-Länge*/

```

```

    {
        while((*datpuffer+index1) > 32) /*solange Zeichen im Puffer
                                         kein Steuerzeichen*/
            ++index1; /*ersten Laufindex erhöhen*/
        *(datpuffer+index1)=0; /*Stringendezeichen '\0' anhängen*/
        filecopy((datpuffer+index2)); /*Name der zu kopierenden Datei
                                         übergeben*/
    }
}
Mfree(rampuffer); /*Freigabe des Dateienpuffers*/
exit(1); /*Programm beenden*/
}

suche_file(File-Name)
char *File-Name;
{
    int ret;

    Fsetdta(&dtabuffer); /*Anfangsadresse des DTA-Puffers
                           festlegen*/
    ret=Ffirst(File-Name,0); /*Dateinamen suchen*/
    if(ret!=0) /*Datei nicht gefunden*/
    {
        printf("\nDatei %s konnte nicht gefunden werden !\n",File-Name);
        printf("\nTaste\n");
        Cconin(); /*auf Tastendruck warten*/
    }
    return(ret); /*Datei gefunden - ja (==0) oder nein
                  (!=0)*/
}
filecopy(File-Name)
char *File-Name; /*Name der zu kopierenden Datei*/
{
    long laenge; /*Länge der Datei*/
    int fhandle, /*Datei-Handle der alten Datei*/
        rhandle, /*Datei-Handle der kopierten Datei*/
        ret;
    ret=suche_file(File-Name); /*Datei gefunden ?*/
    if(ret == 0) /*falls ja, dann*/
    {
        laenge=dtabuffer.size; /*Länge der Datei ermitteln*/
        fhandle=Fopen(File-Name,0); /*Datei öffnen*/
        Fread(fhandle,laenge,rampuffer); /*Datei in Puffer einlesen*/
        Fclose(fhandle); /*Datei schließen*/
        *File-Name=*datpuffer; /*neue Laufwerkskennung übergeben*/
        if((rhandle=Fcreate(File-Name,0))<= 0) /*falls Datei nicht angelegt
                                                werden kann*/
        {
            printf("\nDatei %s konnte nicht erzeugt werden !\n",File-Name);
            printf("\nTaste\n");
            Cconin(); /*auf Tastendruck warten*/
        }
    }
    else /*Datei konnte angelegt werden*/

```

```
{
    Fwrite(rhandle,laenge,rampuffer);/*zu kopierende Datei aus
                                   Puffer schreiben*/
    Fclose(rhandle);               /*Datei schließen*/
}
}
```

14.11 Harddisk-Backup-Programm

Wohl jeder Atari-ST-Besitzer, der auf große Datenbestände, sei es bei Datenbankprogrammen, Grafik- bzw. CAD-Programmen etc., schnell zugreifen will, wird sich über kurz oder lang eine Harddisk zulegen wollen, sofern die Geldbörse mitspielt. Ist diese Anschaffung dann getätigt, sind die wertvollen Daten in mühevoller Arbeit eingegeben und dann auf der Harddisk abgespeichert worden, möchte man natürlich auch, daß diese Arbeit nicht durch irgendeinen Umstand umsonst war. Nur zu gut kennen wir das Problem, daß man aus der anerzogenen Ordnungsliebe heraus die Verzeichnisse der Harddisk von allen mit der Zeit angefallenen .BAK-Files etc. säubern will und dabei ein Daten-File des Datenprogramms versehentlich mitlöscht. Da nach dem Kauf der Harddisk das Geld ein wenig knapper ist, haben wir ein kurzes, aber völlig ausreichendes Kopierprogramm geschrieben, das bei diesem Problem Abhilfe schafft.

Damit kommen wir zu der Funktionsweise des Backup-Programmes, mit dem Sie Ihre Harddisk Partition für Partition sichern können. Nach dem Start des Programmes werden Sie aufgefordert, Ihr Quell- und Ihr Ziellaufwerk anzugeben. Dies geschieht, indem Sie die entsprechenden Kästchen mit vor bzw. zurück mit dem Mauszeiger anklicken. Ist die gewünschte Laufwerkskennung in dem stark umrandeten Kästchen, bestätigen Sie es durch bzw. durch Anklicken. Danach werden Sie gefragt, ob Sie ein Backup Ihrer Harddisk machen wollen oder Ihre auf Diskette gesicherten Daten wieder zurück auf die Harddisk überspielen wollen (Restore). Nach dieser Auswahl wird der noch zur Verfügung stehende Speicherplatz ermittelt und als Datenpuffer reserviert.

Sollten Sie Backup angewählt haben, wird nun die Größe der zu sichernden Harddisk-Partition ermittelt. Dies geschieht durch die Informationen aus dem DAI-Puffer und dem BPB-Puffer. Die Größe errechnet sich wie folgt:

$$\text{Größe} = \text{Gesamt-Cluster-Zahl} * \text{Cluster-Größe} + \text{Sektornr. des 1. Daten-Cluster}$$

Nach dieser Berechnung wird die Gesamtzahl der zum Backup benötigten Disketten berechnet. Diese Zahl bezieht sich nur auf Disketten, die mit 80 Tracks und neun Sektoren formatiert wurden. Dieses Format wurde aus Sicherheitsgründen gewählt, da die Atari-Laufwerke bei höherer Formatierungsdichte manchmal Schwierigkeiten haben. Wurde Ihre Diskette vorher nicht formatiert, erscheint die Meldung Daten auf Disk x defekt. Wählen Sie dann Abbruch; nun wird die Formatierung mit dem obigen Format nachgeholt. Das Formatieren geschieht mit dem oben beschriebenen "Flopfmt"()-Befehl. Nach diesem Vorgang wird noch der Boot-Sektor mit dem "Protobt"()-Befehl erzeugt und auf die Diskette übertragen.

Nun werden die Daten der Partition in Portionen, die vom Diskettenformat abhängen, in den RAM-Puffer geladen, um dann auf die einzelnen Backup-Disketten kopiert zu werden. Auf jeder Diskette wird in einem Infosektor (Sektor 0) die aktuelle Diskettennummer, die Gesamtzahl der Backup-Disketten und die Zahl der kopierten Sektoren festgehalten. Diese Informationen werden beim späteren Restore benötigt. Nach der letzten Diskette wird durch eine Alertbox das Ende des Backup-Vorgangs gemeldet und der reservierte RAM-Puffer wieder freigegeben.

Sollten Sie statt Backup aber Restore ausgewählt haben, so werden Sie jetzt aufgefordert, die erste Backup-Diskette einzulegen. Die Überprüfung der eingelegten Diskette erfolgt über die im Infosektor gespeicherten Informationen, die ja beim Backup gesichert wurden. Durch diese Infos bekommen wir unter anderem die Gesamtzahl der Backup-Disketten, die nun eine nach der anderen eingelegt werden muß. Deren Inhalt wird wiederum in den RAM-Puffer gelesen und von dort aus auf die ausgewählte Partition der Harddisk. Damit ist der Restore-Vorgang auch schon beendet, der reservierte RAM-Puffer wird wieder freigegeben und das Programm beendet.

Mit dem dokumentierten Programm-Listing und dieser Beschreibung können Sie die Arbeitsweise eines Kopierprogrammes nachvollziehen und es vielleicht Ihren eigenen Vorstellungen und Bedürfnissen anpassen.

Es muß noch darauf hingewiesen werden, daß aus Geschwindigkeitsgründen kein unnötiges Accessory installiert sein sollte, da dadurch der RAM-Puffer zu klein werden könnte und kein kompletter Backupdisk-Inhalt eingelesen werden kann. Bei dem Programm sind wir ferner von einem doppelseitigen Laufwerk ausgegangen. Sollten Sie Besitzer eines einseitigen Laufwerkes sein, so müssen Sie im Listing im Unterprogramm backup() die Variable discsize mit dem Wert 719 initialisieren und bei der Formatierungsroutine die Formatierung der Seite 1 (ab page=1 bis return(FALSE)) streichen.

```
/*Harddisk-Backup-Programm*/
#include<stdio.h>
#include<osbind.h>
#include<define.h>
typedef struct dfrei
{
    long freie_cluster;
    long gesamt_cluster;
    long sektor_groesse;
    long cluster_groesse;
}DFREE;
long ramgroesse;
int trackbuffer[0x3000];
int *rampuffer;
char text[200];
main()
{
    int disk1,
        disk2,
        alertret,
        backret;

    appl_init();
    disk1=disk2=0;
    do
    {
        sprintf(text,"[1][ Geben Sie bitte Ihr | Quellaufwerk an ]
                [%c|zurück|vor]",disk1+65);

        alertret=form_alert(1,text);    /*Eingabe des Quellaufwerkes*/
        switch(alertret)
        {
            case 2:
                if(disk1>0)
                    --disk1;
                break;
            case 3:
                if(disk1<5)
                    ++disk1;
                break;
            default:
                break;
        }
    }while(alertret!=1);
    do
    {
        sprintf(text,"[1][ Geben Sie bitte Ihr | Ziellaufwerk an ]
                [%c|zurück|vor]",disk2+65);

        alertret=form_alert(1,text);    /*Eingabe des Ziellaufwerkes*/
        switch(alertret)
        {
```



```

        case 2:
            if(disk2>0)
                --disk2;
            break;
        case 3:
            if(disk2<5)
                ++disk2;
            break;
        default:
            break;
    }
}while(alertret!=1);
alertret=form_alert(0, "[1][ Wollen Sie Ihre Harddisk | sichern oder
                        neu beschreiben][ Backup | Restore ]");
ramgroesse=(long)Malloc(-1L); /*Ermitteln des freien Speicher-
                               platzes*/
rampuffer=(int *)Malloc(ramgroesse); /*ermittelten Speicherplatz
                                       reservieren*/

if(alertret==1)
    backret=backup(disk1,disk2); /*Backup-Vorgang starten*/
else
    restore(disk1,disk2); /*Restore-Vorgang starten*/
if(backret==FALSE) /*falls beim Backup Fehler
                  auftreten*/
    form_alert(1,text); /*Ausgabe der Fehlermeldung*/
Mfree(rampuffer); /*Freigabe des RAM-Puffers*/
appl_exit(); /*Programm abmelden*/
}
backup(disc1,disc2)
int disc1,
    disc2;
{
    DFREE dai; /*Puffer der Disk Allocation
                Information (DAI)*/

    long dummy,
        magic; /*Konstante bei Formatierung*/
    int page, /*Ziel-Disketten-Seite*/
        tracknumber,
        discsize, /*Anzahl der Sektoren der
                  Ziel-Disketten*/
        copysize, /*Anzahl der Sektoren der HD-
                  Partition*/
        *biospbptr, /*Zeiger auf Bios-Parameter-Block
                    (BPB)*/
        gesdisk, /*benötigte Disketten zum
                 Backup*/
        aktuell, /*aktuelle Ziel-Disk-Nummer*/
        rest, /*noch zu kopierende Sektoren*/
        index, /*schon kopierte Sektoren*/
        interleave, /*Reihenfolge der Sektoren beim
                    Formatieren*/

```

```

    virgin,                /*Standardwert beim Formatieren*/
    sektor,               /*Sektoren pro Track*/
    format;               /*Diskette formatiert - ja/nein*/
long error;               /*Nummer der Fehlermeldungen*/
int dbuff[256];           /*Puffer zum Boot-Sektor
                           generieren*/
                           /*Initialisierung der
                           Konstanten*/

interleave=0x1;
virgin=0xE5E5;
magic=0x87654321;
sektor=0x9;
index=0;
format=FALSE;
discsize=1439;           /*bei 80 Tracks mit neun Sektoren
                           und zwei Seiten*/

if(((long)discsize*512)>ramgroesse) /*genügend Speicherplatz
    vorhanden?*/
{
    sprintf(text, "[1][Nicht genug Speicherplatz|vorhanden !][ OK ]");
    return(FALSE);
}
Dfree(&dai, disc1+1);      /*Disk-Informationen Quelldisk in
                           DAI-Puffer laden*/
biospbptr=(int *)Getbpb(disc1); /*Disk-Informationen in BPB-Puffer
    laden*/
                           /*Anzahl der Sektoren auf HD*/
copysize=(dai.gesamt_cluster * dai.cluster_groesse)+(*biospbptr+6);

                           /*Gesamtzahl der Backup-Disketten
                           ermitteln*/
if((copysize>discsize)&&((copysize%discsize)!=0))
    gesdisk=copysize/discsize+1;
else if(copysize<discsize)
    gesdisk=1;
else
    gesdisk=copysize/discsize;
sprintf(text, "[1][Es werden %d Disketten |zum Backup benötigt|
    [ OK ]", gesdisk);
form_alert(1, text);
rest=copysize;           /*noch zu kopierende Sektoren*/
aktuell=1;
do
{
    error=Rwabs(2, rampuffer, 1, 0, disc2); /*Auf Formatierung prüfen*/
    if(error!=0)          /*nicht formatiert*/
    {
        format=TRUE;
        discsize=1439;    /*Anzahl der Sektoren auf
                           Diskette minus Sektor '0'*/
    }
    else                  /*bereits formatiert - dann

```

```

                                Ziel-Disk-Größe ermitteln*/
{
    Dfree(&dai,disc2+1);          /*Disk-Informationen Ziel-Disk in
                                DAI-Puffer laden*/
    biospbptr=(int *)Getbpb(disc2); /*Diskinformationen in BPB-
                                Puffer laden*/
    discsize=(dai.gesamt_cluster * dai.cluster_groesse)+
                (*(biospbptr+6)); /*Anzahl der Sektoren auf Disk*/
    discsize-=1;
    format=FALSE;
}
sprintf(text,"[1][%d.Diskette ins|Laufwerk %c legen|Achtung!|Daten
auf Disk gehen verloren][ OK ]",aktuell,disc2+'A');
form_alert(1,text);
tracknumber=0;
if(format==TRUE)
{
    do                            /*Ziel-Disk formatieren*/
    {
        page=0;
        error=Flopfmt(trackbuffer,0l,disc2,sektor,tracknumber,page,
                        interleave,magic,virgin);
        sprintf(text,"[1][Formatierungsfehler|Seite 0
|Error=%ld][ok]",error);
        if(error<0)
            return(FALSE);
        page=1;
        error=Flopfmt(trackbuffer,0l,disc2,sektor,tracknumber,page,
                        interleave,magic,virgin);
        sprintf(text,"[1][Formatierungsfehler|Seite 1
|Error=%ld][ok]",error);
        if(error<0)
            return(FALSE);
        tracknumber++;
    }while(tracknumber<=79);
}

Protobt(dbuff,0x01000001,3,0); /*Boot-Sektor erzeugen*/
error=Flopwr(dbuff,0l,disc2,1,0,0,1);/*Boot-Sektor auf Ziel-Disk
                                schreiben*/
sprintf(text,"[1][Schreibfehler im
Boot-Sektor|error=%ld][ok]",error);
if(error<0)
    return(FALSE);
if(discsize>rest)
    discsize=rest;
error=Rwabs(2,rampuffer,1,0,disc2);
rampuffer=aktuell; /*aktuelle Disk-Nummer*/
*(rampuffer+1)=gesdisk; /*Gesamtzahl der Ziel-Disketten*/
*(rampuffer+2)=discsize; /*Zahl der zu kopierenden
Sektoren*/
error=Rwabs(3,rampuffer,1,0,disc2); /*Header-Info's der Zieldis-
ketten in Sektor '0'*/

```

```

error=Rwabs(2,rampuffer,discsize,index,disc1); /*Sektoren von
                                                Quelldiskette in Puffer einlesen*/
error=Rwabs(3,rampuffer,discsize,1,disc2);      /*Puffer auf Ziel-
                                                disk schreiben*/
index+=discsize;                               /*schon kopierte Sektoren*/
rest-=discsize;                                /*noch zu kopierende Sektoren*/
sprintf(text,"[1][Diskettenfehler !|Error = %ld|Backup wird
        abgebrochen][ OK ]",error);
if(error!=0)
    return(FALSE);
    aktuell++;
}while(rest>0);
form_alert(1,"[1][ Backup beendet ][ OK ]");
return(TRUE);
}
restore(disc1,disc2)
int disc1,
    disc2;
{
    int aktuell,
        gesdisk,
        discsize,
        index;

    aktuell=1;
    index=0;
    sprintf(text,"[1][Legen Sie die 1.Backup-Disk|in Laufwerk %c|
        [ OK ]",disc1+'A');
    form_alert(1,text);
    which_disk(disc1,aktuell);
    gesdisk =*(rampuffer+1);                /*Gesamtzahl der Backup-
Disketten*/
    do
    {
        Rwabs(2,rampuffer,1,0,disc1);
        discsize=*(rampuffer+2);           /*zu kopierende Sektoren*/
        sprintf(text,"[1][Aktuelle Backup-Disk = %d|Gesamtzahl der
            |Backup-Disketten = %d][ OK ]",aktuell,gesdisk);
        form_alert(1,text);
        Rwabs(2,rampuffer,discsize,1,disc1);
        Rwabs(3,rampuffer,discsize,index,disc2);
        index+=discsize;
        if(aktuell<gesdisk)
            which_disk(disc1,aktuell+1);
        aktuell++;
    }while(aktuell<=gesdisk);
    form_alert(1,"[1][Restore beendet !! |Achtung!!
        |System muß neu gebootet werden][ OK ]");
}
which_disk(diskette,nummer)
int diskette,
    nummer;
{
    /*Quelldisk*/
    /*aktuelle Disk-Nummer*/

```

```
int number;                                /*gefundene Disk-Nummer in
                                           Header-Info*/
do
{
    Rwabs(2,rampuffer,1,0,diskette); /*Headerinfo's einlesen*/
    number=*rampuffer;                /*aktuelle Disk-Nummer ermitteln*/
    if(number!=nummer)
    {
        sprintf(text,"[1][Das ist die %d.Backup-Disk
            |%d.Backup-Disk einlegen][OK]",number,nummer);
        form_alert(1,text);
    }
}while(number!=nummer);
}
```

15. Grundlegende Grafikprogrammierung in BASIC und mit VDI

Der Bildschirm des ST ist geradezu prädestiniert für die Darstellung von Grafik. Anders als bei den Massenspeichern sind hier schnelle Erfolge möglich - und vor allem auch direkt zu sehen.

In den folgenden Abschnitten werden wir gemeinsam in die Geheimnisse der Computergrafik mit Hilfe der Befehle des GFA-BASIC und den Routinen des VDI, die uns das Betriebssystem zur Verfügung stellt, einsteigen. Wir werden anhand kleiner oder größerer Beispielprogramme in BASIC, Assembler oder C die verschiedenen Effekte und Techniken kennenlernen, die Grafik an sich ausmachen.

15.1 Kleine Grafik-Einführung

Dieses Kapitel können Sie ruhig überschlagen, falls Sie an einer Kurzeinführung nicht interessiert sind. Trotzdem wird es Ihnen bei dem Verständnis der nachfolgenden Programme in diesem Buch sehr hilfreich sein, sich mit diesem Thema einmal ganz kurz auseinandergesetzt zu haben.

Ihr Atari ST bietet Ihnen drei Grafikarten an, die Sie nach Bedarf bzw. hardwaremäßiger Ausstattung (Monochrom- oder Farbmonitor) auswählen können:

15.1.1 640x400 - hochaufgelöste Grafik (HGR)

Dieser Modus ist nur mit einem Monochrom-Monitor erreichbar. Hier ist es Ihnen möglich, jeden einzelnen Punkt des Bildschirms anzusprechen. Ihr Atari ST verwendet dazu ein Bit jedes Bytes des Grafikspeichers als Punktraster (0 = Punkt nicht gesetzt; 1 = Punkt gesetzt). Es können nur die beiden Grundfarben des Monitors verwendet werden (beim originalen Atari SM124 die beiden Farben Schwarz und Weiß). Dabei können Sie einstellen, welche Farbe die gesetzten und welche die nicht gesetzten Punkte haben sollen.

Das zum Setzen eines Punktes notwendige Koordinatensystem sieht dann folgendermaßen aus:

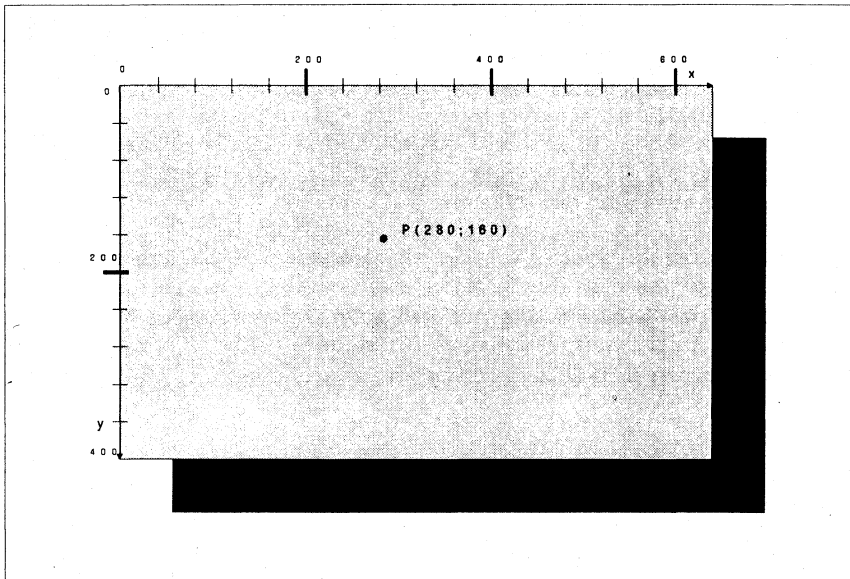


Abb. 15.1: 640x400-Koordinatensystem

Auffällig ist, daß sich der Nullpunkt dieses Koordinatensystems links oben in der Ecke des Bildschirms befindet. Diese Einteilung hat sich in der Computerwelt geradezu standardisiert und wird uns noch in vielen Anwendungen und Programmen begegnen. Da uns der Atari unter VDI zusätzlich noch eine weitere Möglichkeit des Koordinatensystems an die Hand gibt, unterscheiden wir folgende zwei Koordinatensysteme:

a) Rasterkoordinaten (RC)

Unter Rasterkoordinaten verstehen wir z.B. die gerade vorgestellte Einteilung des Bildschirms in Koordinaten 640x400 mit dem Nullpunkt in der oberen linken Ecke des Bildschirms. Sie richtet sich nach den hardwaremäßigen Gegebenheiten des Bildschirms.

b) Normalisierte Koordinaten (NDC)

Die normalisierten Koordinaten stellen ein für jeden Modus und jedes grafikfähige Gerät (Bildschirm, Plotter etc.) einheitliches Koordinatensystem dar, dessen Ursprung (0,0) in der unteren linken Ecke und dessen Endkoordinaten (32767,32767) in der oberen rechten Ecke des Bildschirms bzw. allgemein des

Grafikgerätes liegen. Die normalisierten Koordinaten werden im Falle des Bildschirms in die Rasterkoordinaten umgerechnet, wobei der jeweiligen Bildschirmauflösung Rechnung getragen wird. Es handelt sich also um ein geräteunabhängiges Koordinatensystem. Bei der Verwendung der VDI-Routinen können die Grafikausgaben, die sonst auf den Bildschirm gehen, dann bei Vorhandensein der entsprechenden Gerätetreiber auch unverändert auf ein anderes Ausgabegerät (Plotter, Drucker, Disk) umgeleitet werden.

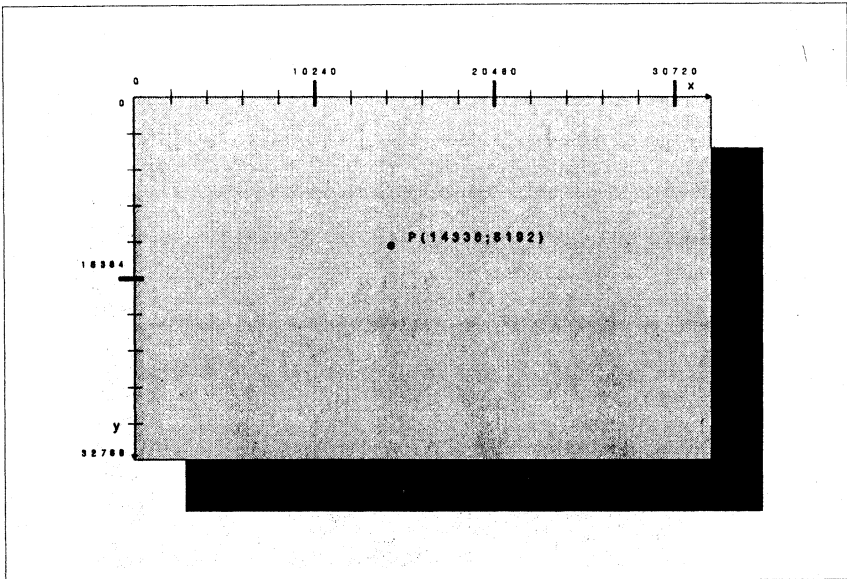


Abb. 15.2: NDC-Koordinatensystem

15.1.2 640x200-mittlere Grafikauflösung (MGR)

In diesem Grafikmodus, der nur auf einem Farbmonitor verfügbar ist, wird die Farbauflösung auf Kosten der Grafikauflösung erhöht. Auf einem Bild können hier insgesamt vier verschiedene Farben von 512 möglichen gleichzeitig dargestellt werden. Dementsprechend wurde in diesem Modus die Punkte-Auflösung in y-Richtung herabgesetzt. Ein Punkt ist demnach doppelt so hoch wie breit, die Objekte erscheinen in Folge dessen verzerrt auf dem Bildschirm, wenn man nicht entsprechende Vorkehrungen trifft.

Jeder Punkt wird dabei durch zwei verschiedene Bits des 16-KByte-Grafikspeichers kodiert, die beide in zwei aufeinanderfolgenden Wörtern stehen. Damit können alle vier verschiedenen Farben bestimmt werden (%00 = Hintergrundfarbe, %01 = Farbe 1, %10 = Farbe 2, %11 = Farbe 3). Welche der 512 möglichen Farben die Punkte annehmen können, wird in einer sogenannten Farbpalette festgelegt.

Das zum Setzen eines Punktes notwendige Koordinatensystem sieht dann folgendermaßen aus:

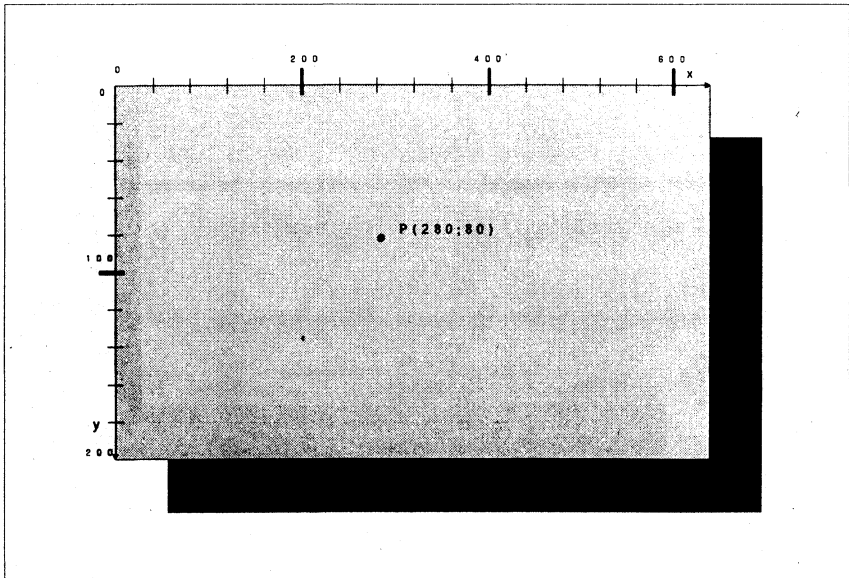


Abb. 15.3: 640x200-Koordinatensystem

15.1.3 320x200-niedrig aufgelöste Grafik (NGR)

In diesem letzten Grafikmodus erhält jeder Punkt nun wieder die gleiche Breite wie Höhe, ist allerdings jeweils doppelt so hoch und doppelt so breit wie unter HGR. Dafür können gleichzeitig auf dem Bildschirm 16 verschiedene Farben von den bekannten 512 möglichen erscheinen. Die Farbmöglichkeiten sind hier also bereits recht ordentlich.

Ein Punkt wird im Speicher durch jeweils vier Bits kodiert, die sich in vier aufeinanderfolgenden Speicher-Wörtern befinden. Mit den vier Bits können alle 16 Farben dargestellt werden. Die oben genannte Farbpalette umfaßt hier also nun 16 Farben, die eine Auswahl der 512 möglichen Farben darstellen.

Das zum Setzen eines Punktes notwendige Koordinatensystem sieht dann folgendermaßen aus:

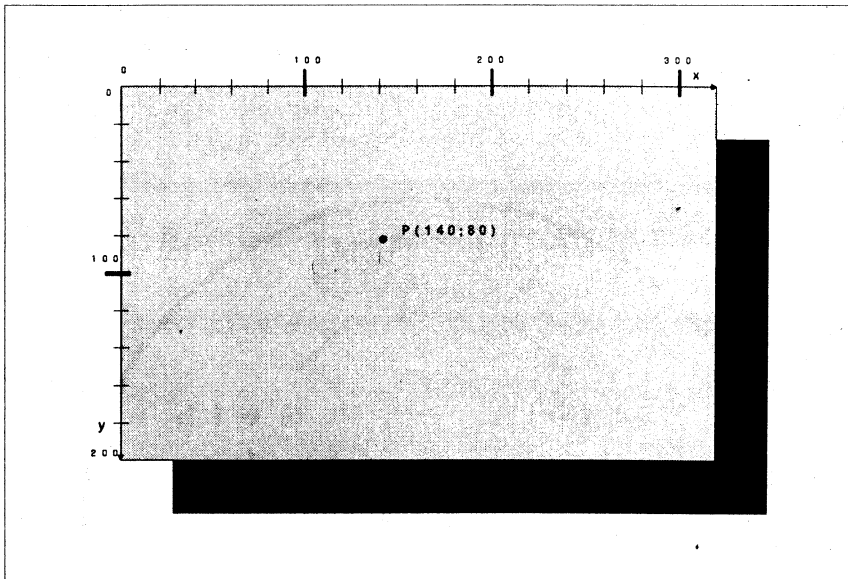


Abb. 15.4: 320x200-Koordinatensystem

15.1.4 Die Farben

Ihr Atari ST kann nicht nur einfache Schwarzweiß-Bilder auf dem Bildschirm darstellen, wie dies vom Monochrom-Monitor bekannt ist. Enorm sind auch die Fähigkeiten Ihres Rechners, mit Farben zu hantieren. Es stehen Ihnen - sofern Sie in Besitz eines anschlußfähigen Farbmonitors sind (oder einen Fernseher und einen Rechner mit UHF-Modulator besitzen) - insgesamt 512 verschiedene Farben zur Verfügung, die sich aus verschiedenen Helligkeitsstufen der drei Grundfarben Rot, Grün und Blau zusammensetzen. Jede dieser drei Grundfarben kann in acht solcher Schattierungen zur Gesamtfarbe zugemischt werden. Daraus ergeben sich $8 \times 8 \times 8 = 512$ Kombinationen.

Die Farbmischung erfolgt nach dem sogenannten additiven Prinzip (im Gegensatz zum subtraktiven Prinzip bei der Mischung von z.B. Wasserfarben). Bei der additiven Farbmischung wird die Mischfarbe um so heller, je intensiver die verwendeten Grundfarben sind (bei der subtraktiven Farbmischung ist das Gegenteil der Fall). Das folgende Schaubild soll Ihnen erläutern, welche Mischfarben durch den Einsatz welcher Grundfarben entstehen. Die Tabelle geht dabei nur von zwei Intensitäten für jede Grundfarbe aus (ein oder aus). Je mehr Intensitätsunterteilungen (beim Atari jeweils acht) für eine Grundfarbe existieren, desto mehr Zwischen-Schattierungen gibt es natürlich auch bei den Mischfarben. Interessant ist die Tatsache, daß bei jeweils gleichen Intensitäten der drei Grundfarben stets eine Grau-Schattierung resultiert (von schwarz bis weiß). Weiß entsteht dabei, wenn alle Grundfarben maximale Helligkeit haben, Schwarz dagegen, wenn alle Intensitäten auf Null stehen.

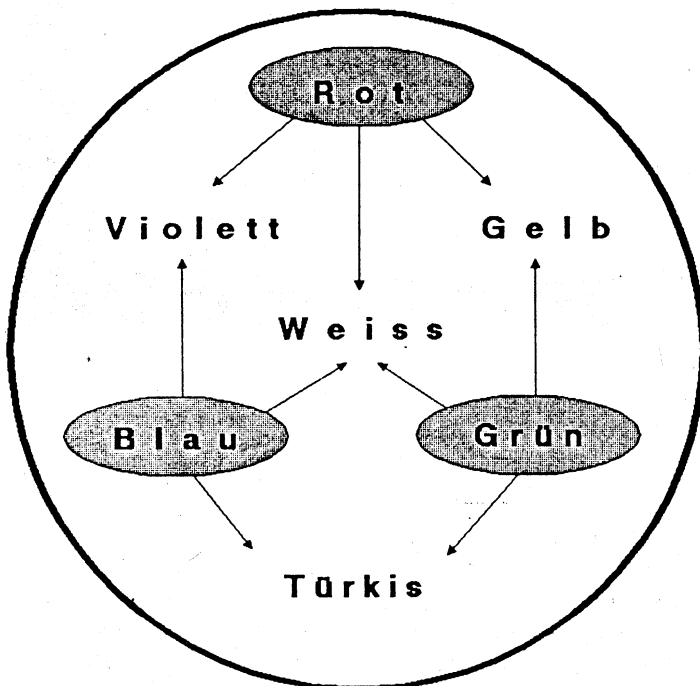


Abb. 15.5: Additive Farbmischung (modifizierter Goethe-Farbkreis)

Da nun aber in den verschiedenen Grafik-Modi ohne Umwege nie alle 512 Farben gleichzeitig auf dem Bildschirm erscheinen können, wird in einer rech-

nerinternen Tabelle, der sogenannten Farb-Palette, festgelegt, welche vier oder 16 Farben nun für die oben erwähnten Bit-Kombinationen eines Punktes erscheinen sollen. Man kann das Ganze mit einem Tuschekasten mit 512 Farben, aber nur z.B. 16 Pinseln vergleichen. Jeder dieser Pinsel kann nun in eine Farbe getaucht werden. Damit hätte man 16 verschiedene Farben zur Verfügung, ein Bild zu gestalten.

Für den NGR-Modus sähe das dann etwa so aus:

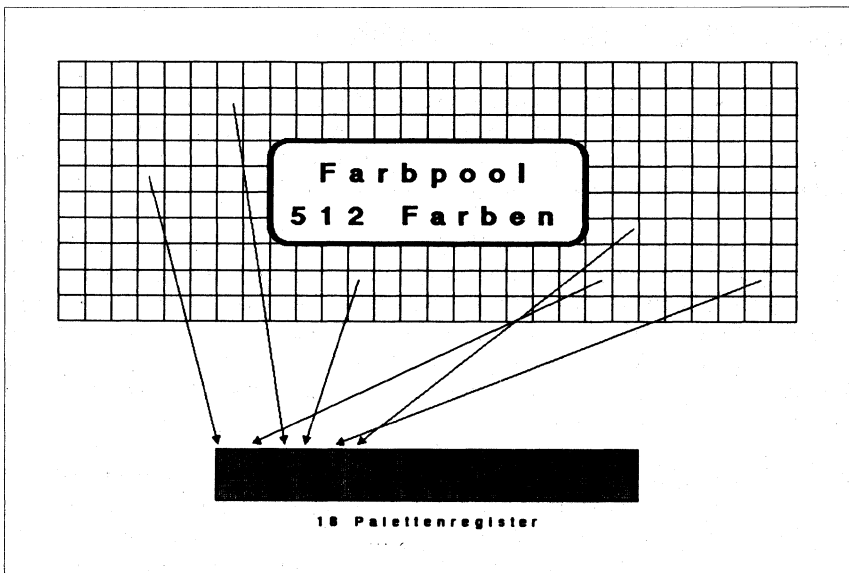


Abb. 15.6: Farbpalette NGR

15.2 Der Punkt als Urgrafik

Als der große griechische Philosoph Euklid vor mehr als 2000 Jahren einmal die Eindrücke einer wunderschönen griechischen Landschaft in den Sand zeichnen wollte und er mit seinem Holzstab zum Malen ansetzte, hielt er einen Moment inne, setzte den Stab wieder ab und betrachtete den Eindruck, den dieser im Sand hinterlassen hatte. Von diesem Moment an wußte er, daß alles mit dem Punkt beginnt. Obwohl diese Entdeckung lange Zeit in Vergessenheit geraten war, bekommt sie im Zeitalter der Computer wieder zentrale Bedeutung. Hier sind nämlich alle Grafiken, Bilder und Buchstaben im Endeffekt aus

Punkten zusammengesetzt. So wie also die Materie aus kleinsten Teilchen besteht, so verhält es sich auch mit der Grafik. Die glatteste Fläche, die ebenste Linie ist nur eine Annäherung, die unser Auge täuscht. Damit ist ein Befehl, der einen einzelnen Punkt auf dem Bildschirm setzt, die Grundlage aller Grafik. Alle anderen Figuren beruhen auf dieser Rechenvorschrift und stellen lediglich eine Erleichterung für uns dar. Wollen wir sehen, wie es sich damit verhält!

15.2.1 PLOT oder POLYMARKER?

Das Interessante am Atari-Betriebssystem ist die Tatsache, daß es eigentlich keinen Befehl gibt, um einzelne Punkte auf den Bildschirm zu bringen. Trotzdem gibt es natürlich die Möglichkeit, die wir auch sofort - der Einfachheit halber zunächst in BASIC - vorstellen möchten:

Mit dem GFA-BASIC-Befehl PLOT, der einen einzelnen Punkt, und DRAW, dem Befehl, der sowohl einen Punkt als auch eine Linie auf den Bildschirm zaubern soll (damit werden wir uns etwas später beschäftigen), wollen wir unsere Einführung in die Grafik auf dem Atari ST beginnen. Mit diesem Befehl werden Sie eine Menge über die wesentliche Struktur der Grafik und ihrer Programmierung in BASIC, Assembler und C lernen. Dieses Kapitel sollten Sie also besonders aufmerksam lesen.

Zunächst aber probieren Sie doch einmal das folgende Programmbeispiel, das Ihnen das Weiterlesen schmackhaft machen soll. Starten Sie also GFA-BASIC aus dem Desktop heraus. Beachten Sie bitte, daß es sich hierbei um ein Programm für die hochauflösende Grafik des Monochrom-Monitors handelt. Wollen Sie es für Ihren Farbmonitor umschreiben, so ändern Sie bitte die Initialisierungswerte für die Grafikauflösung am Anfang des Programms:

```
' *****  
' **  
' ** PLOT-DEMO **  
' **  
' *****  
'  
' Bildschirmauflösung:  
xauf1%=DPEEK(&H2686)  
yauf1%=DPEEK(&H2688)  
'  
xa=xauf1%/2  
ya=yauf1%/2  
xb=xauf1%*0.3  
yb=yauf1%*0.2  
'
```

```

/
FOR x=0 TO 943 STEP 0.3
  PLOT xb*SIN(x/15)+xa,ya*SIN(x/50)+ya
NEXT x
/
FOR x=0 TO 943 STEP 0.3
  PLOT ya*SIN(x/50)+xa,xb*SIN(x/15)+ya
NEXT x
/
' Auf Taste warten:
c=INP(2)
EDIT

```

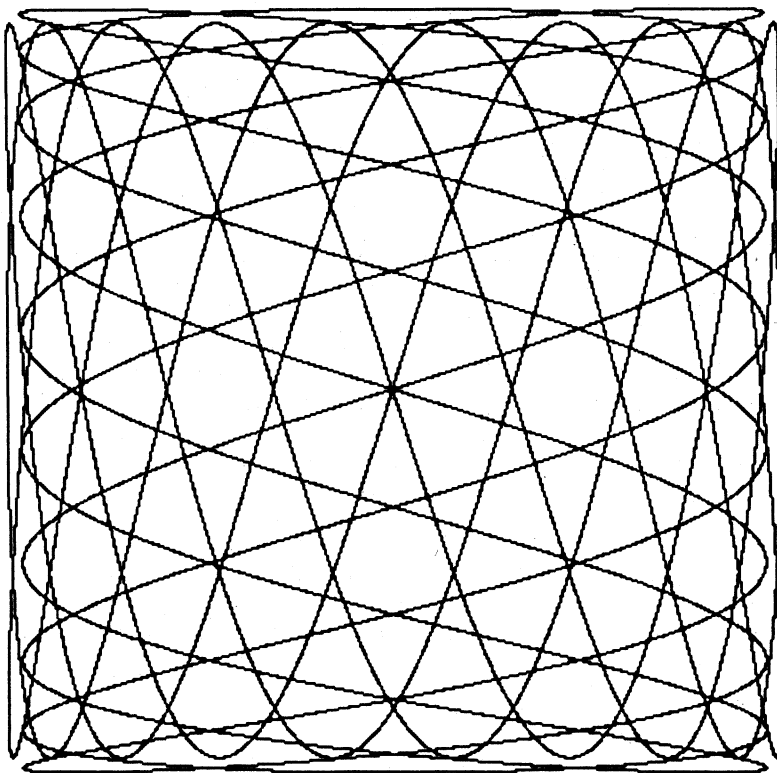


Abb. 15.7: PLOT-DEMO

Sie brauchen nicht unbedingt zu verstehen, was es mit den merkwürdig erscheinenden Formeln in diesem Programm auf sich hat. Es handelt sich hier um eine Form der sogenannten Lissajous-Kurven. Diese kleine Demonstration sollte Sie nur auf das Thema einstimmen. Sprechen wir das Programm doch einmal kurz durch:

Bevor wir zur Sache gehen, muß zunächst einmal die momentane Bildschirm-auflösung erfragt werden. Das Programm soll schließlich unter allen Auflösungen laufen. Dies können wir mit Hilfe des Befehles OPEN VIRTUAL SCREEN WORKSTATION feststellen. Doch das ist ziemlich umständlich. Wir werden aus diesem Grunde in Zukunft bei BASIC-Programmen die Auflösung direkt aus den Workstation-Informationen im Speicher holen. Diese Informationen, die sämtliche Werte enthalten, die als Ausgabeparameter in OPEN VIRTUAL SCREEN WORKSTATION mitgeteilt werden, liegen ab Adresse \$2686 (dezimal: 9862) im Speicher (Wortlänge). Durch ein

```
DPEEK(&H2686)
```

erfahren wir die Bildschirmbreite in Pixel, durch ein

```
DPEEK(&H2688)
```

erfahren wir die Bildschirmhöhe in Pixel. Damit können wir die Koordinaten für unsere Gebilde jeweils auf Bildschirmgröße umrechnen. Das wird - soweit möglich - in allen Demo-Programmen getan, damit Sie unter jeder Auflösung Spaß an den verschiedenen Routinen haben. Schwierig wird die Umrechnung - das kann ich Ihnen bereits jetzt verraten - oft in der mittleren Auflösung, so daß hier in seltenen Fällen einmal Ausnahmen gemacht werden müssen.

Durch das Starten des Programms wird der gesamte Grafikbildschirm zunächst einmal gelöscht. Wir haben also freie Bahn und können loslegen:

Durch die FOR...NEXT-Schleife wird die Variable X ständig um 0,3 erhöht, die in die Formeln zur Koordinatenberechnung des PLOT-Befehls einfließt. Um den PLOT-Befehl, den Befehl also, der einen einzigen Punkt auf den Bildschirm zeichnet, näher zu verstehen, schauen wir ihn uns doch einmal etwas genauer an:

PLOT x,y (GFA-BASIC-Befehl)

Setzen (Löschen etc.) eines Punktes

x: x-Koordinate
y: y-Koordinate

Dieser Befehl ermöglicht Ihnen also, einen Punkt mit den Koordinaten x,y auf dem Bildschirm in der entsprechenden Farbe (s.u.) zu zeichnen. Um das mit den Koordinaten zu verstehen, rufen wir uns noch einmal das Atari-interne Koordinatensystem in Erinnerung, das bereits in Kapitel 15.1. vorgestellt wurde. In diesem Koordinatensystem lag der Nullpunkt ja links oben in der Ecke des Bildschirmfensters. Die Koordinaten der y -Achse wurden nach unten (!) hin, die der x -Achse nach rechts immer größer. Dabei kamen folgende Werte für x und y in Frage, wenn ein Punkt innerhalb des Bildschirms liegen sollte:

HGR: x : 0 bis 639
 y : 0 bis 399

MGR: x : 0 bis 639
 y : 0 bis 199

NGR: x : 0 bis 319
 y : 0 bis 199

Dabei bedeuten:

HGR: Hoch aufgelöste Grafik
MGR: Mittel aufgelöste Grafik
NGR: Niedrig aufgelöste Grafik

In unserem Beispiel haben wir es mit HGR zu tun. Damit befindet sich der Punkt mit den Koordinaten $x=0$ und $y=399$ in der unteren linken und der Punkt mit den Koordinaten $x=639$ und $y=399$ in der unteren rechten Ecke des Bildschirms. In dem Befehl `PLOT  $x,y$` geben Sie also für x die x - und für y die y -Koordinate des Punktes an, der gezeichnet werden soll. Statt diese Werte direkt mit Zahlen anzugeben, können Sie natürlich dafür auch jede Variable oder Formel einsetzen, die Ihnen in den Sinn kommt, wie dies auch in dem obigen Beispiel geschehen ist, z.B.

```
PLOT (x+1)*5,b^2
```

Unter BASIC sieht das Ganze also recht einfach aus, solange wir mit den vorgesehenen BASIC-Befehlen hantieren. Wie aber haben wir uns die Programmierung eines Punktes in Assembler und wie in C vorzustellen? Hier sind wir ja gezwungen, auf VDI direkt zuzugreifen. (Das können wir in BASIC natürlich auch, doch dazu später.)

Nun, VDI bietet uns eine recht komfortable Funktion, die es uns erlaubt, eine beliebige Anzahl von Markierungen auf dem Bildschirm zu erzeugen. Diese

Markierungen können z.B. Punkte sein. Es stehen aber noch eine ganze Reihe anderer Figuren wie Kreuze oder Sternchen zur Verfügung. Zudem können diese Markierungen mit zwei anderen VDI-Funktionen noch in der Größe und in der Farbe verändert werden. Die dazu notwendigen vier VDI-Funktionen lauten:

```
POLYMARKER (7 VDI)
SET POLYMARKER TYPE (18 VDI)
SET POLYMARKER HEIGHT (19 VDI)
SET POLYMARKER COLOR INDEX (20 VDI)
```

Eine Bemerkung zu den Ausgabeparametern der ersten drei Funktionen. Wenn Sie den Funktionen einen Wert übergeben, der zu hoch oder zu niedrig ist, dann wählen sie sich jeweils einen plausiblen gültigen Wert. In diesem Fall unterscheidet sich natürlich der Ausgabewert von Ihrem Eingabewert. Und nur in diesem Fall haben die Ausgabewerte für uns einen Sinn.

Bei der Funktion zur Farbgebung muß eine Kleinigkeit beachtet werden. Wir werden später noch verschiedene Möglichkeiten zur Farbmanipulation kennenlernen. Dabei werden wir mit den sogenannten Farbpalettenregistern zu tun haben. Die Nummern dieser Register stimmen nicht mit dem im VDI anzugebenen Farbindex überein, obwohl es sich rein logisch um ein und dasselbe handeln sollte. VDI hat jedoch (aus recht undurchsichtigen Gründen) eine Umrechentabelle, die dafür sorgt, daß sich die durch einen VDI-Farbindex angesprochenen Farbregister nicht mit den Registernummern decken. Mehr hierzu jedoch im Farbkapitel.

Wie Sie sehen, ist doch eine ganze Menge nötig, um unser Vorhaben zu verwirklichen, einen einzigen Punkt auf den Bildschirm zu bringen. Doch wenn es erst einmal geschafft ist, dann bieten sich mit minimalen Mitteln enorme Möglichkeiten. Wir werden jetzt nacheinander mit Hilfe dieser vier Funktionen drei Beispielprogramme jeweils in Assembler, C und BASIC vorstellen, die alle genau das gleiche bewirken: Sie zeichnen vier Punkte mit den Koordinaten P1(100,100), P2(150,100), P3(120,145) und P4(130,145) auf den Bildschirm:

Zunächst einmal das Assembler-Programm (Bitte beachten Sie dabei, daß dieses Programm nur zusammen mit den Start- und Endkonventionen aus dem Kapitel 2 läuft! Sie müssen es also ab "main" zu diesen Konventionen hinzufügen.):

```
;Initialisierung:
```

```
;-----
```

```
main:    move.w    #10,opcode      ;Opcode appl_init
          clr.w     aintin         ;Anzahl intin-Werte=0
          clr.w     aaddrin        ;Anzahl addrin-Adressen=0
```

```
clr.l  ap1resv      ;reservierte Bereiche
clr.l  ap2resv      ;vorsorglich löschen
clr.l  ap3resv
clr.l  ap4resv
jsr    aes          ;AES-Funktion aufrufen
move.w #77,opcode   ;Opcode graf_handle
clr.w  aintin
clr.w  aaddrin
move.w #5,aintout
jsr    aes          ;AES-Funktion
move.w intout,handle ;ermittelte Device-Nummer
                           ;nach handle
move.w #100,opcode  ;Opcode open_vwork
clr.w  sptsin
move.w #11,sintin   ;11 Werte im intin-Array
move.w #1,intin
move.w #1,intin+2
move.w #1,intin+4
move.w #1,intin+6
move.w #1,intin+8
move.w #1,intin+10
move.w #1,intin+12
move.w #1,intin+14
move.w #1,intin+16
move.w #1,intin+18
move.w #2,intin+20  ;Rasterkoordinaten einstellen
jsr    vdi          ;VDI-Funktion aufrufen
move.w #3,opcode    ;Opcode clear workstation
                           ;(v_clrwk)

clr.w  sintin
clr.w  sptsin
jsr    vdi          ;Aufruf Bildschirm löschen

;Programm:
;-----
move.w #18,opcode   ;set polymarker type
clr.w  sptsin
move.w #1,sintin
move.w #1,intin     ;Markierungstyp = 1
                           ; = Punkt
jsr    vdi          ;Aufruf
move.w #19,opcode   ;set polymarker height
move.w #1,sptsin
clr.w  sintin
clr.w  ptsin        ;ungenutzt
move.w #8,ptsin+2   ;Markierungshöhe
jsr    vdi          ;Aufruf
move.w #20,opcode   ;set polymarker color index
clr.w  sptsin
move.w #1,sintin
move.w #1,intin     ;Farbe = 1
jsr    vdi          ;Aufruf
move.w #7,opcode    ;Polymarker
clr.w  sintin
```

```

move.w #4,sptsin      ;4 Punkte
move.w #100,ptsin     ;x-Wert Punkt 1
move.w #100,ptsin+2   ;y-Wert Punkt 1
move.w #150,ptsin+4   ;x-Wert Punkt 2
move.w #100,ptsin+6   ;y-Wert Punkt 2
move.w #120,ptsin+8   ;x-Wert Punkt 3
move.w #145,ptsin+10  ;y-Wert Punkt 3
move.w #130,ptsin+12  ;x-Wert Punkt 4
move.w #145,ptsin+14  ;y-Wert Punkt 4
jsr     vdi           ;Funktion aufrufen
;Terminierung:
;-----
move.w #101,opcode    ;close virtual screen
                        ;workstation
clr.l  sptsin
clr.l  sintin
jsr    vdi             ;Aufruf
move.w #19,opcode     ;appl_exit
clr.l  aintin
clr.l  aaddrin
jsr    aes
rts                                ;mit RTS zurück

```

Wenn Sie bisher alles verstanden haben, dann sollte dieses Programm für Sie keine besonderen Schwierigkeiten mehr darstellen. Die Initialisierung und die Terminierung kennen Sie bereits. Neu ist lediglich der unter "Programm" stehende Teil mit den oben erläuterten VDI-Funktionsaufrufen. Dabei werden zunächst nacheinander die Funktionen SET POLYMARKER TYPE, SET POLYMARKER HEIGHT und SET POLYMARKER COLOR INDEX aufgerufen, die das nähere Aussehen der Markierungen festlegen sollen. Dabei spielt in diesem Fall die Polymarker-Höhe keine Rolle, da es sich in jedem Fall um einen einzelnen Punkt handelt, der nicht vergrößert oder verkleinert werden kann. Die Funktion wurde jedoch direkt in das Programm aufgenommen, um die Option der Vergrößerung zu haben, wenn Sie bei der Funktion SET POLYMARKER TYPE statt des Punktes eine andere Figur angeben. Sie sollten überhaupt einmal die verschiedenen Parameter der einzelnen Befehle variieren, um zu sehen, wie sich das dann auf das Ergebnis auswirkt. Nur so werden Sie die einzelnen Möglichkeiten der Befehle verstehen und später damit arbeiten können.

Bei der Farbe sehen wir uns für den Monochrom-Monitor auf die beiden Werte 0 oder 1 beschränkt, wobei der Wert 0 die Punkte in der Hintergrundfarbe erscheinen läßt, Sie also auf dem Bildschirm kaum etwas erkennen werden. Das ändert sich erst, wenn Sie einen Farbmonitor anschließen. Dazu brauchen Sie nicht einmal Änderungen an Ihrem Programm vorzunehmen, da die Koordinaten der einzelnen Punkte auch für die anderen Grafikauflösungen passen.

Wenn Sie also Ihren Farbmonitor anschließen, können Sie auch den Farbwert für die Color-Funktion des VDI entsprechend ändern und schauen, was passiert.

Die aktuelle Auflösung des Bildschirms können Sie übrigens auch durch die Funktion OPEN VIRTUAL SCREEN WORKSTATION erfahren. Diese Funktion gibt neben vielen anderen Dingen auch die aktuelle Auflösung des Bildschirms in den Variablen

```
intout(0) - x-Auflösung
intout(1) - y-Auflösung
```

zurück.

Nach der Bestimmung der einzelnen veränderbaren Polymarker-Merkmale können wir die Markierungen (hier Punkte) auf den Bildschirm bringen. Dazu übergeben wir der Funktion POLYMARKER die Anzahl der zu zeichnenden Punkte (hier vier) in "contrl(1)" (also: sintin). Damit der Rechner auch weiß, wo er die vier Punkte hinsetzen soll, müssen wir ihre Koordinaten im "ptsin"-Array ablegen. Erst jetzt können wir die Funktion aufrufen, die Punkte (oder anderen Markierungen) erscheinen auf dem Bildschirm.

Für die C-Programmierer sehen wir uns nun einmal das Ganze aus der C-Perspektive an:

```
/******  
/**                               **/  
/** Polymarker-Types  **/  
/**                               **/  
/******  
#define MPUNKT      1  
#define MPLUS       2  
#define MSTERN      3  
#define MQADRAT     4  
#define MDKREUZ     5  
#define MDIAMANT    6  
  
int contrl[12],  
    intin[128],  
    ptsin[128],  
    intout[128],  
    ptsout[128];  
  
int gl_apid;  
int gem_handle;  
int handle;  
int symbol;  
int height;  
int color_index;
```

```

int count;
int gl_wchar, gl_hchar, gl_wbox, gl_hbox;
int work_in[12],
    work_out[57],
    pxyarray[10];

/*****
/*      Initialisiere Applikation      */
*****/
aes_init()
{
    gl_apid = appl_init();
    gem_handle = graf_handle(&gl_wchar, &gl_hchar,
                             &gl_wbox, &gl_hbox);
}

/*****
/*      Terminiere Applikation      */
*****/
aes_exit()
{
    gemdos(0x1);                /* Auf Taste warten */
    appl_exit();
}

/*****
/*      öffne virtuelle Workstation    */
*****/
open_vwork()
{
    int i;

    for(i=0; i<10; work_in[i++]=1);
    work_in[10] = 2;
    handle = gem handle;
    v_opnvwk(work_in, &handle, work_out);
}

/*****
/*      Schließe virtuelle Workstation */
*****/

#define close_vwork() v_clswwk(handle);

/*****
/*      Hauptprogramm                    */
*****/

main()
{
    aes_init();
    open_vwork();                /* Initialisierung      */
}

```

```
v_clrwk(handle);          /* Bildschirm löschen */
symbol = MPUNKT;
vsm_type(handle, symbol); /* Punkt als Symbol */
height = 8;
vsm_height(handle, height); /* Höhe (bei Punkt egal) */
color_index = 1;
vsm_color(handle, color_index); /* Farbe = 1 */
pxyarray[0] = 100;         /* x-Wert Punkt 1 */
pxyarray[1] = 100;         /* y-Wert Punkt 1 */
pxyarray[2] = 150;         /* x-Wert Punkt 2 */
pxyarray[3] = 100;         /* y-Wert Punkt 2 */
pxyarray[4] = 120;         /* x-Wert Punkt 3 */
pxyarray[5] = 145;         /* y-Wert Punkt 3 */
pxyarray[6] = 130;         /* x-Wert Punkt 4 */
pxyarray[7] = 145;         /* y-Wert Punkt 4 */
count = 4;                 /* Anzahl Punkte = 4 */
v_pmarker (handle, count, pxyarray);
/* Terminierung: */
close_vwork();
aes_exit();
}
```

Wie Sie sehen, haben wir ganz zu Anfang alle Parameter-Felder sowie alle Variablen, die in den einzelnen Funktionen verwendet werden, als int definiert. Voraussetzung für die Verwendung der obigen VDI- und AES-Funktionen ist natürlich eine vorhandene Funktions-Bibliothek, die eigentlich zu Ihrem C-Compiler gehören und sich auch nach dem obigen offiziellen Standard richten sollte. Bitte schauen Sie dazu in Ihrem Compiler-Handbuch nach, oder sehen Sie sich die Library-Files an. Bei manchen Compilern müssen Sie beim Linken noch die verschiedenen Library-Teile angeben, wie z.B. "vdibind" oder "aesbind" etc. (so z.B. beim Digital Research Entwicklungssystem), andere Linker setzen dies automatisch um (z.B. Megamax). Dazu sollten Sie in der Anleitung nachschauen.

Als nächstes definieren wir einige nützliche Funktionen, die Aufgaben übernehmen wie: Applikationsinitialisierung und -Terminierung sowie das Öffnen und Schließen der Arbeitsstation ("aes_init()", "aes_exit()", "open_vwork()" und "close_vwork()"). Es werden dabei genau die gleichen Konventionen eingehalten wie bereits besprochen. Beachten Sie bitte bei "aes_exit()", daß hier mittels der GEMDOS-Funktion Nummer 1 auf eine Taste gewartet wird. Wir haben hier absichtlich die direkte Ansteuerung über die Funktionsnummer gewählt und werden es in Zukunft auch immer tun (der Ausdruck "0x1" bedeutet soviel wie: die Hexadezimalzahl 1, also: \$1). Hat Ihr Compiler auch eine GEMDOS-Bibliothek (meist osbind.h genannt, sie enthält dann GEMDOS-, BIOS-, und XBIOS-Funktionen), so können Sie statt

```
gemdos(0x1);
```

auch eingeben:

```
Cconin();
```

Bei unserer Funktion "open_vwork"() werden in einer FOR-Schleife die zehn "nutzlosen" Parameter in das work-Array (also intin-Array) übergeben. "work_in"[10] wird dann auf 2 gesetzt, um das RC-Koordinatensystem zu aktivieren. Dann wird das in "aes_init"() ermittelte handle gesetzt und die Station geöffnet.

Im folgenden passiert eigentlich genau das gleiche wie beim obigen Assemblerprogramm. Sie brauchen also nur in den dortigen Erläuterungen nachzuschauen.

Zum guten Schluß wollen wir die VDI-Funktion POLYMARKER auch von BASIC aus nutzen. Dazu möge das folgende Beispiel dienen:

```

/ *****
/ **
/ ** POLYMARKER-Demo **
/ **
/ *****
/
DIM work_in%(11),pxyarray%(20)
/
/ Initialisierung:
/ *****
/
@graf_handle          !handle holen
/
FOR x%=0 TO 9
  work_in%(x%)=1      !Array mit Einsen füllen
NEXT x%
work_in%(10)=2        !Rasterkoordinaten

@v_opnvwk(*work_in%(),*handle%)
@v_s_clip(handle%,0,0,0,0,0) !kein Clipping
/
@v_clrwk(handle%)      !Bildschirm löschen
/
/ Zeichnen:
/ *****
/
@vswr_mode(handle%,mode%) !Zeichenmodus
@vsm_color(handle%,1)     !Farbe setzen
@vsm_type(handle%,1)      !Typ setzen
/
pxyarray%(0)=100         !x-Koordinate Punkt 1
pxyarray%(1)=100         !y-Koordinate Punkt 1

```

```
pxyarray%(2)=150      !...
pxyarray%(3)=100
pxyarray%(4)=120
pxyarray%(5)=145
pxyarray%(6)=130
pxyarray%(7)=145
'
@v_pmarker(handle%,4,*pxyarray%()) !4 Punkte zeichnen
'
'
' Terminieren:
' *****
'
@v_clswk(handle%)
'
END
'
'
'
' *****
' Unterprogrammbibliothek
' *****
'
'
' AES-Funktionen definieren:
' *****
'
'
' GRAF_HANDLE:
'
PROCEDURE graf_handle
'
  DPOKE GCONTRL+2,0      !Einträge in int_in
  DPOKE GCONTRL+4,5      !Einträge in int_out
  DPOKE GCONTRL+6,1      !Einträge in addr_in
  DPOKE GCONTRL+8,0      !Einträge in addr_out
'
  GEMSYS 77
'
  handle%=DPEEK(GINTOUT)
'
RETURN
'
'
'
' VDI-Funktionen definieren:
' *****
'
'
' OPEN VIRTUAL SCREEN WORKSTATION:
```



```

PROCEDURE v_opnvwk(awork_in%,ahandle%)
,
DPOKE CONTRL+2,0           !Anzahl Punkte in ptsin
DPOKE CONTRL+6,11          !Anzahl der Werte in intin
DPOKE CONTRL+12,*ahandle%  !handle
SWAP *awork_in%,x%()        !Austausch: Arraywerte/Dimension
FOR x%=0 TO 10              !11 Werte übertragen
    DPOKE INTIN+2*x%,x%(x%) !Array übertragen
NEXT x%
SWAP *awork_in%,x%()
,
VDISYS 100                  !Funktion 100 aufrufen
,
*ahandle%=DPEEK(CONTRL+12) !handle neu besetzen
,
RETURN
,
,
' CLOSE VIRTUAL SCREEN WORKSTATION:
,
PROCEDURE v_clsvwk(handle%)
,
DPOKE CONTRL+2,0
DPOKE CONTRL+6,0
DPOKE CONTRL+12,handle%
,
VDISYS 101
,
RETURN
,
,
' CLEAR WORKSTATION:
,
PROCEDURE v_clrwk(handle%)
,
DPOKE CONTRL+2,0
DPOKE CONTRL+6,0
DPOKE CONTRL+12,handle%
,
VDISYS 3
,
RETURN
,
,
' SET CLIPPING RECTANGLE:
,
PROCEDURE vs_clip(handle%,clip_flag%,x1%,y1%,x2%,y2%)
,
DPOKE CONTRL+2,2
DPOKE CONTRL+6,1
DPOKE CONTRL+12,handle%
DPOKE INTIN,clip_flag%      !Clipping ein/aus

```

```
DPOKE PTSIN,x1%           !Clipp-Fenster-Koordinaten
DPOKE PTSIN+2,y1%
DPOKE PTSIN+4,x2%
DPOKE PTSIN+6,y2%
,
VDISYS 129
,
RETURN
,
,
' SET WRITING MODE:
,
PROCEDURE vswr_mode(handle%,mode%)
,
DPOKE CONTRL+2,0
DPOKE CONTRL+6,1
DPOKE CONTRL+12,handle%
DPOKE INTIN,mode%
,
VDISYS 32
,
RETURN
,
,
' POLYMARKER:
,
PROCEDURE v_pmarker(handle%,count%,apxyarray%)
,
DPOKE CONTRL+2,count%      !Anzahl Markierungen
DPOKE CONTRL+6,0
DPOKE CONTRL+12,handle%
SWAP *apxyarray%,x%()      !Austausch: Feldinhalte/Dimension
FOR x%=0 TO count%-1
    DPOKE PTSIN+4*x%,x%(2*x%) !Array nach ptsin übertragen
    DPOKE PTSIN+4*x%+2,x%(2*x%+1)
NEXT x%
SWAP *apxyarray%,x%()
,
VDISYS 7
,
RETURN
,
,
' SET POLYMARKER COLOR INDEX:
,
PROCEDURE vsm_color(handle%,color_index%)
,
DPOKE CONTRL+2,0
DPOKE CONTRL+6,1
DPOKE CONTRL+12,handle%
DPOKE INTIN,color_index%
,
VDISYS 20
```

```

,
RETURN
,
,
' SET POLYMARKER TYPE:
,
PROCEDURE vsm_type(handle%,symbol%)
,
    DPOKE CONTRL+2,0
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,symbol%
,
    VDISYS 18
,
RETURN

```

Eine interessante Variation des obigen Programms erhalten wir, wenn Sie den Teil des Programms, der unter dem Punkt "Zeichnen" steht, durch den folgenden ersetzen:

```

' Zeichnen:
,
@vswr_mode(handle%,mode%) !Zeichenmodus
@vsm_color(handle%,1) !Farbe setzen
@vsm_type(handle%,1) !Typ setzen
,
FOR xk%=0 TO 640
    pxyarray%(0)=xk%
    pxyarray%(1)=60*SIN(xk%/40)+150
    @v_pmarker(handle%,1,*pxyarray%())
NEXT xk%

```

Auch hier können Sie natürlich die verschiedenen Parameter nach Lust und Laune verändern.

Sicher werden Ihnen die vielen platzaufwendigen Funktionsdefinitionen und -aufrufe genauso wie mir langsam die gute Laune verderben. Nun, GFA-BASIC hat eine ganze Menge VDI-Befehle in seinen Befehlssatz aufgenommen. So können wir das obige BASIC-Programm auch in aller Kürze und Knappheit wie folgt verfassen:

```

' *****
' **                                     **
' ** POLYMARKER-Demo II **
' **                                     **
' *****
,
,
DIM xkoord%(10),ykoord%(10)
,

```

```
' Zeichnen:
' *****
'
GRAPHMODE 1           !Zeichenmodus setzen
DEFMARK 1,1,1         !Farbe, Typ, Größe setzen
'
xkoord%(0)=100        !x-Koordinate Punkt 1
ykoord%(0)=100        !y-Koordinate Punkt 1
xkoord%(1)=150        !...
ykoord%(1)=100
xkoord%(2)=120
ykoord%(2)=145
xkoord%(3)=130
ykoord%(3)=145
'
POLYMARK 4,xkoord%(),ykoord%() !4 Punkte zeichnen
'
' 4 Punkte verschoben zeichnen:
POLYMARK 4,xkoord%(),ykoord%() OFFSET -80,50
'
END
```

Sie sehen also: Es entfallen alle VDI-Start-Konventionen (wie z.B. OPEN VIRTUAL SCREEN WORKSTATION etc.) sowie die Definition der Bibliotheksfunktionen und alles wird schön kurz und übersichtlich. In diesem Programm verwenden wir einige neue GFA-Befehle, die wir kurz vorstellen wollen:

GRAPHMODE m (GFA-BASIC-Befehl)

Einstellen des Zeichenmodus für alle Grafik-Zeichenoperationen

m: Zeichenmodus:
m=1: Replace
m=2: Transparent
m=3: exklusiv ODER
m=4: Invers Transparent

DEFMARK (fi),(typ),(gr) (GFA-BASIC-Befehl)

Einstellen der Polymarker-Eigenschaften

fi: Polymarker-Farbindex
typ: Polymarker-Typ
gr: Polymarker-Größe

POLYMARK n,xa,ya (OFFSET xo,yo) (GFA-BASIC-Befehl)

Setzen (Löschen etc.) mehrerer Markierungen evtl. mit Offset

n: Anzahl der Markierungen (1-128)
xa: Array der x-Koordinaten aller Markierungen
ya: Array der y-Koordinaten aller Markierungen
xo: x-Koordinaten-Verschiebungswert
yo: y-Koordinaten-Verschiebungswert

Die Befehle sollten Ihnen vom VDI her bekannt sein. Neu ist lediglich die Möglichkeit, beim Befehl POLYMARK einen gemeinsamen x- bzw. y-Offset anzugeben, der jeweils zu den Markierungskordinaten addiert wird. Damit wird es möglich, eine definierte Figur auf einfache Weise auf dem Bildschirm zu verschieben.

Interessant ist auch die Möglichkeit, mit DEFMARK gleich drei verschiedene Markierungseigenschaften auf einmal einzustellen. Sollten Sie einen Parameter nicht ändern wollen, so lassen Sie ihn einfach weg.

Wenn Sie innerhalb eines GFA-BASIC-Programms zusätzlich zu den normalen BASIC-Befehlen noch VDI-Funktionen aufrufen wollen (denn alle VDI-Funktionen sind auch in GFA-BASIC nicht in den Befehlssatz aufgenommen), dann können Sie das (fast) ohne weiteres tun. Nur ein Problem taucht auf: Woher nehmen Sie die Handle-Nummer, die GFA-BASIC vom VDI zugewiesen wurde und mit der GFA-BASIC selbst die verschiedenen VDI-Funktionen aufruft? Nun, da hilft uns eine Sondervariable, die seit der Version II erreichbar ist: VDIBASE. Diese Variable gibt uns die Adresse eines Registerblockes an, in dem einige nützliche Dinge gespeichert sind, die spezielle Einstellungen (Linienform, Textparameter etc.) für die jeweilige Workstation speichert. (Jede Workstation kann schließlich unterschiedlichste Einstellungen tätigen, unabhängig von eventuellen anderen Workstations.) Die Adresse

VDIBASE+40

gibt dann die Handle-Nummer dieser Workstation an (Wortlänge). Mit einem

DPEEK(VDIBASE+40)

kann aus GFA-BASIC heraus die aktuelle Handle-Nummer erfragt werden. Wir werden dies später in einigen Programmen anwenden.

15.2.2 Farben bringen Farbe hinein

Dieses Kapitel ist natürlich nur für diejenigen interessant, die glückliche Besitzer eines Farbmonitors (oder natürlich eines TV-Anschlusses) sind und den Atari ST einmal aus einer ganz anderen Perspektive betrachten können. Diejenigen unter Ihnen, die nur einen Monochrom-Monitor anschließen können, sollten sich diesen Abschnitt jedoch ebenfalls durchlesen, da vieles hier Gesagte auch für die hohe Auflösung zutrifft. Außerdem erhöht es eventuell sogar den Reiz, sich einen farbigen Monitor zuzulegen.

Nun, Ihr Rechner kann tatsächlich eine ganze Menge, was Farben betrifft. Immerhin haben Sie die Wahl zwischen 512 verschiedenen Farben, die Sie allerdings nicht ohne weiteres gleichzeitig auf den Bildschirm bringen können. An dieser Stelle sollten Sie sich vielleicht noch einmal den kleinen Abschnitt 15.1.4. "Die Farben" durchlesen, damit Sie mit dem folgenden etwas vertrauter sind.

Sie wissen also, daß sich in Ihrem Rechner eine sogenannte Farbpalette von insgesamt 16 Farben befindet, in der Sie die Farben der 512 möglichen auswählen können. Im Bildschirmspeicher steht also nicht die Farbe eines einzelnen Punktes, sondern die Nummer des Farbregisters, in dem die zuständige Farbe steht. Ändert man die Farbe innerhalb des Farbregisters, so ändert sich die Farbe jedes einzelnen Punktes, der die Farbe dieses Farbregisters besitzt.

Die Farbzusammenstellung in der Palette können Sie selbstverständlich jederzeit durch entsprechende VDI-Funktionen festlegen und auch ändern. Dabei ändern sich dementsprechend alle Punkte des Bildschirms, die jemals mit diesem Farbregister gezeichnet wurden. Gleichzeitig können Sie angeben, aus welchem Farbregister VDI die Farbe z.B. für POLYMARKER nehmen soll, um die entsprechenden Figuren zu zeichnen.

Im Vorkapitel haben wir beim Zeichnen einer POLYMARKER-Markierung bereits die Farbgebung ins Spiel gebracht - Sie werden sich erinnern. Im folgenden zeigen wir Ihnen eine kleine Variation der drei angegebenen Programme. Beginnen wir wieder mit dem Assembler-Listing. Ersetzen Sie bitte den unter "Programm:" (nicht unter main!) stehenden Teil durch den folgenden:

```
;Programm:
;-----
      move.w #18,opcode      ;set polymarker type
      clr.w  sptsin
      move.w #1,sintin
      move.w #6,intin        ;Markierungstyp = 6
                              ; = Diamant
      jsr    vdi             ;Aufruf
```

```

        move.w #19,opcode      ;set polymarker height
        move.w #1,sptsin
        clr.w  sintin
        clr.w  ptsin          ;ungenutzt
        move.w #100,ptsin+2    ;Markierungshöhe
        jsr    vdi            ;Aufruf
        move.w #1,d6          ;Farbinitialisierungswert
loop:   move.w #20,opcode      ;set polymarker color index
        clr.w  sptsin
        move.w #1,sintin
        move.w d6,intin       ;Farbe
        jsr    vdi            ;Aufruf
        move.w #7,opcode      ;Polymarker
        clr.w  sintin
        move.w #4,sptsin      ;4 Punkte
        move.w #100,ptsin     ;x-Wert Punkt 1
        move.w #100,ptsin+2   ;y-Wert Punkt 1
        move.w #150,ptsin+4   ;x-Wert Punkt 2
        move.w #100,ptsin+6   ;y-Wert Punkt 2
        move.w #120,ptsin+8   ;x-Wert Punkt 3
        move.w #145,ptsin+10  ;y-Wert Punkt 3
        move.w #130,ptsin+12  ;x-Wert Punkt 4
        move.w #145,ptsin+14  ;y-Wert Punkt 4
        jsr    vdi            ;Funktion aufrufen
        move.w #1,-(sp)       ;Funktionsnummer für
                                ;CONIN (GEMDOS)
        trap   #gemdos        ;GEMDOS-Funktion aufrufen
                                ;(Auf Taste warten)
        addq.l #2,sp          ;Stack in Ordnung bringen
        addq.w #1,d6          ;nächster Farbindex
        cmp.w  #16,d6         ;d6=16?
        bne    loop          ;nein! -> weiter

;Terminierung: ...
;-----

```

Wie Sie sehen, haben wir den POLYMARKER-Typ in "Diamant" geändert sowie als Größenfaktor 100 angegeben. Zudem haben wir eine kleine Schleife einprogrammiert, die die Farbe der POLYMARKER bei jedem Tastendruck hochzählt.

Das gleiche für das C-Programm:

Ersetzen Sie doch den main()-Teil des obigen C-Programms durch die folgende Befehlssequenz (der folgende Programmteil läuft also nicht für sich allein!):

```

/*****
/*      Hauptprogramm      */
*****/

main()
{
    aes_init();
    open_vwork();           /* Initialisierung      */
    v_clrwk(handle);        /* Bildschirm löschen  */
    symbol = MDIAMANT;
    vsm_type(handle, symbol); /* Punkt als Symbol    */
    height = 100;
    vsm_height(handle, height); /* Höhe (bei Punkt egal) */
    for (color_index = 1; color_index < 16; color_index++)
    {
        vsm_color(handle, color_index); /* Farbe = 1          */
        pxyarray[0] = 100;             /* x-Wert Punkt 1     */
        pxyarray[1] = 100;             /* y-Wert Punkt 1     */
        pxyarray[2] = 150;             /* x-Wert Punkt 2     */
        pxyarray[3] = 100;             /* y-Wert Punkt 2     */
        pxyarray[4] = 120;             /* x-Wert Punkt 3     */
        pxyarray[5] = 145;             /* y-Wert Punkt 3     */
        pxyarray[6] = 130;             /* x-Wert Punkt 4     */
        pxyarray[7] = 145;             /* y-Wert Punkt 4     */
        count = 4;                     /* Anzahl Punkte = 4  */
        v_pmarker (handle, count, pxyarray);
        gemdos(0x1);                   /* Auf Taste warten   */
    }

    /* Terminierung: */
    close_vwork();
    aes_exit();
}

```

Wie Sie sehen, haben wir (wie beim Assembler-Programm) statt vier Punkte vier Diamanten (oder Karos) auf den Bildschirm gezeichnet. Dabei haben wir ihnen die Größe 100 gegeben. Das alles können Sie natürlich auf Ihrem Monochrom-Monitor ebenfalls vollbringen. Der interessante Teil des Programms folgt aber nun in der FOR-Schleife. Hier wird der Farbindex "color_index" von 1 bis 15 hochgezählt und jedesmal neu gezeichnet, sobald Sie eine Taste betätigen. Am besten sehen Sie den Effekt, wenn Sie NGR einschalten. Beachten Sie bitte, daß aus dem Palettenregister 0 (= VDI-Farbindex 0) stets die Hintergrundfarbe stammt. Aus diesem Grunde haben wir die Nummer 0 ausgelassen.

Die erscheinenden Farben hängen natürlich von Ihrer Farbeinstellung im Kontrollfeld-Accessory ab (einzustellen im Desktop), da ja bei den obigen Pro-

grammen nicht etwa die Nummer der Farbe, sondern nur der VDI-Farbindex, also die Nummer des Registers, aus dem die Farbe stammt, geändert wird.

Dabei ist eine Besonderheit zu vermerken! Die Farbindizes des VDI stimmen nicht mit der Numerierung der Farbpalettenregister überein. Aus irgendwelchen dubiosen Gründen wird im VDI noch eine Umwandlungstabelle zwischengeschoben. Folgende Zuordnung wurde getroffen:

PR:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FI:	0	2	3	6	4	7	5	8	9	10	11	14	12	15	13	1

Dabei bedeuten:

PR: Palettenregisternummer

FI: VDI-Farbindex

Das heißt also: Wenn ich eigentlich Palettenregister 1 ansprechen möchte, so muß ich unter VDI Farbindex 2 angeben usw. Das hat jedoch noch weitere Konsequenzen. Arbeiten wir im 640x200-Vierfarb-Modus, so sind für uns normalerweise nur Palettenregister 0-3 zuständig. Aufgrund der obigen Zuordnung arbeiten wir unter VDI allerdings mit den Farbindizes 0,2,3,6. Welchen Sinn diese Zuordnung haben soll, ist mir nicht ganz klar. Wollen wir diese Tabelle umgehen, so müssen wir sozusagen eine Gegentabelle erstellen, die unsere gewünschte Registernummer in den Index umwandelt, der vom VDI dann wieder in die richtige Registernummer umgewandelt wird.

Wie aber ändern wir die Belegung dieser Farbpalette? Wir wollen ja schließlich einmal alle 512 Farben auf dem Bildschirm sehen. Nun, dazu müssen wir wissen (s. Kapitel 15.1.4.), daß jede Farbe des Bildschirms aus den drei Grundfarben Rot, Grün und Blau gemischt wird. Jede dieser Grundfarben ist in acht Helligkeitsstufen (von 0=AUS (=schwarz) bis 7=voll AN) verfügbar. Damit ergeben sich die bekannten $8 \times 8 \times 8 = 512$ verschiedenen Farben. VDI stellt uns nun eine Funktion zur Verfügung, in der wir einem Palettenregister die gewünschte Farbkombination zuordnen können: SET COLOR REPRESENTATION (14 VDI).

In dieser Funktion geben Sie jeweils den Rot-, Grün- und Blauanteil einer Farbe an und ordnen ihn einem Farbindex, also einem Palettenregister zu. (Denken Sie stets an die oben beschriebene Umwandlungstabelle!) Dabei werden die Werte für die Intensitäten nach einem recht merkwürdigen Verfahren angegeben. Für unseren Bildschirm sind nur jeweils acht verschiedene Intensitätswerte möglich. Nun könnte man auf den Gedanken kommen, diese Werte als 0,1,2,...,7 anzugeben. Doch weit gefehlt! VDI soll ja bekanntermaßen

auf möglichst viele Geräte zugeschnitten sein. Dementsprechend wird der Wertebereich für Intensitäten gewaltig gestreckt:

Intensität	VDI-Wert
0	- 70
1	71 - 213
2	214 - 356
3	357 - 499
4	500 - 642
5	643 - 785
6	786 - 928
7	929 - 1000

Wenn Sie also beispielsweise die vierte Intensitätsstufe der Grundfarbe Blau einstellen wollen, so müssen Sie der Funktion in "intn(3)" (für C-Freunde: "rgb_in"[2]), also dem Blau-Register, den Wert 500 übergeben (oder alle größeren Werte bis 642). Negative Werte erzeugen automatisch die Intensität 0 (also schwarz). Damit können wir folgende Formel aufstellen:

$$\text{VDI-Wert} = \text{Intensität} * 143 - 72$$

Damit lassen sich alle VDI-Werte der obigen Tabelle berechnen, wenn man nur die Intensitätsstufe kennt. Bei der Intensität 0 käme man dann allerdings auf einen negativen Wert (-72), was allerdings nicht viel ausmacht, da es auch bei Integerzahlen negative Werte gibt (oberstes Bit gesetzt).

Bei der Untersuchung des VDI-Listings wurde diese merkwürdige Formel bestätigt. Alle positiven Werte größer 1000 werden auf die Intensität 1000 zurückgebracht.

Haben Sie die mittlere Auflösung gewählt, so geben natürlich nur die Palettenregisternummern 0-3 einen Sinn, da Punkte auf dem Bildschirm gar nicht in anderen Nummern erscheinen können, laut Umrechnungstabelle wären das dann die Indizes 0,2,3,6. Entsprechend haben Sie beim Monochrom-Monitor nur mit dem Palettenregister 0 zu tun. Hier allerdings ist ebenfalls die Farbwahl eingeschränkt (schließlich gibt es insgesamt nur zwei mögliche Farben, z.B. weiß und schwarz). In diesem Modus bestimmt allein der Index (0 oder 1) die dargestellte Farbe. Weiß (oder eine analoge Farbe) entsteht, wenn Sie alle Farbtintensitäten auf 7, Schwarz dagegen, wenn Sie alle auf 0 stellen (dies ist übrigens auch im Farbmodus so!).

Um uns die Verwendung dieses Befehls einmal vor Augen zu führen, wollen wir wieder ein kleines BASIC-, gefolgt vom entsprechenden C-Programm durchsprechen:

```

' *****
' **          **
' ** Farb-Demo **
' **          **
' *****
'
DIM work_in$(11),pxyarray$(20),farben$(8),rgb_in$(3)
'
@graf_handle          !handle holen
'
FOR x%=0 TO 9
    work_in$(x%)=1      !Array mit Einsen füllen
NEXT x%
work_in$(10)=2         !Rasterkoordinaten
'
@v_opnvwk(*work_in$(),*handle$)
@vs_clip(handle$,0,0,0,0,0) !kein Clipping
'
@v_clrwk(handle$)      !Bildschirm löschen
'
' Zeichnen:
' *****
'
' Hintergrundfarbe auf schwarz:
rgb_in$(0)=0
rgb_in$(1)=0
rgb_in$(2)=0
@vs_color(handle$,0,*rgb_in$())
'
'
FOR x%=0 TO 7
    READ farben$(x%)    !Farbwerte einlesen
NEXT x%
'
@vswr_mode(handle$,mode$) !Zeichenmodus
'
@vsm_type(handle$,3)     !Typ setzen
'
ind%=0
FOR xkoord%=0 TO 319 STEP 30
    FOR ykoord%=10 TO 199 STEP 30
        '
        ind%=ind%+1      !Farbindex hochzählen
        IF ind%=16
            ind%=1
        ENDIF
        @vsm_color(handle$,ind%) !Farbe setzen
    
```

```
    pxyarray%(0)=xkoord%      !x-Wert-Markierung
    pxyarray%(1)=ykoord%      !y-Wert-Markierung
    @v_pmarker(handle%,1,*pxyarray%())
NEXT ykoord%
NEXT xkoord%
,
,
' Farben der Sterne verändern:
,
ind%=0
FOR rgb0%=1 TO 7              !Rot-Schleife
  FOR rgb1%=1 TO 7            !Grün-Schleife
    FOR rgb2%=1 TO 7          !Blau-Schleife
      ind%=ind%+1             !Farbindex hochzählen
      IF ind%=16
        ind%=1
      ENDIF
      ind%=ind%
      rgb_in%(0)=farben%(rgb0%)
      rgb_in%(1)=farben%(rgb1%)
      rgb_in%(2)=farben%(rgb2%)
      ,
      @vs_color(handle%,ind%,*rgb_in%()) !Farbe setzen
      ,
      c=INP(2)                  !Auf Taste warten
      ,
    NEXT rgb2%
  NEXT rgb1%
NEXT rgb0%
,
,
' Hintergrundfarbe auf weiß:
rgb_in%(0)=1000
rgb_in%(1)=1000
rgb_in%(2)=1000
@vs_color(handle%,0,*rgb_in%())
,
,
' Terminieren:
' *****
,
@v_clsawk(handle%)
,
END
,
,
DATA 0,71,214,357,500,643,786,929
,
,
,
' *****
' Unterprogramm-bibliothek
```

```

' *****
'
'
'
' AES-Funktionen definieren:
' *****
'
'
' GRAF_HANDLE:
'
PROCEDURE graf_handle
'
  DPOKE GCONTRL+2,0      !Einträge in int_in
  DPOKE GCONTRL+4,5      !Einträge in int_out
  DPOKE GCONTRL+6,1      !Einträge in addr_in
  DPOKE GCONTRL+8,0      !Einträge in addr_out
'
  GEMSYS 77
'
  handle%=DPEEK(GINTOUT)
'
RETURN
'
'
'
' VDI-Funktionen definieren:
' *****
'
'
' OPEN VIRTUAL SCREEN WORKSTATION:
'
PROCEDURE v_opnvwk(awork_in%,ahandle%)
'
  DPOKE CONTRL+2,0      !Anzahl Punkte in ptsin
  DPOKE CONTRL+6,11     !Anzahl der Werte in intin
  DPOKE CONTRL+12,*ahandle% !handle
  SWAP *awork_in%,x%()  !Austausch: Arraywerte/Dimension
  FOR x%=0 TO 10        !11 Werte übertragen
    DPOKE INTIN+2*x%,x%(x%) !Array übertragen
  NEXT x%
  SWAP *awork_in%,x%()
'
  VDISYS 100            !Funktion 100 aufrufen
'
  *ahandle%=DPEEK(CONTRL+12) !handle neu besetzen
'
RETURN
'
'
' CLOSE VIRTUAL SCREEN WORKSTATION:
'
PROCEDURE v_clsvwk(handle%)
'

```

```
DPOKE CONTRL+2,0
DPOKE CONTRL+6,0
DPOKE CONTRL+12,handle%
,
VDISYS 101
,
RETURN
,
,
' CLEAR WORKSTATION:
,
PROCEDURE v_clrwk(handle%)
,
DPOKE CONTRL+2,0
DPOKE CONTRL+6,0
DPOKE CONTRL+12,handle%
,
VDISYS 3
,
RETURN
,
,
' SET CLIPPING RECTANGLE:
,
PROCEDURE vs_clip(handle%,clip_flag%,x1%,y1%,x2%,y2%)
,
DPOKE CONTRL+2,2
DPOKE CONTRL+6,1
DPOKE CONTRL+12,handle%
DPOKE INTIN,clip_flag%      !Clipping ein/aus
DPOKE PTSIN,x1%              !Clipp-Fenster-Koordinaten
DPOKE PTSIN+2,y1%
DPOKE PTSIN+4,x2%
DPOKE PTSIN+6,y2%
,
VDISYS 129
,
RETURN
,
,
' SET WRITING MODE:
,
PROCEDURE vswr_mode(handle%,mode%)
,
DPOKE CONTRL+2,0
DPOKE CONTRL+6,1
DPOKE CONTRL+12,handle%
DPOKE INTIN,mode%
,
VDISYS 32
,
RETURN
,
```

```

/
/ POLYMARKER:
/
PROCEDURE v_pmarker(handle%,count%,apxyarray%)
/
    DPOKE CONTRL+2,count%      !Anzahl Markierungen
    DPOKE CONTRL+6,0
    DPOKE CONTRL+12,handle%
    SWAP *apxyarray%,x%()      !Austausch: Feldinhalte/Dimension
    FOR x%=0 TO count%-1
        DPOKE PTSIN+4*x%,x%(2*x%) !Array nach ptsin übertragen
        DPOKE PTSIN+4*x%+2,x%(2*x%+1)
    NEXT x%
    SWAP *apxyarray%,x%()
/
    VDISYS 7
/
RETURN
/
/
/ SET POLYMARKER COLOR INDEX:
/
PROCEDURE vsm_color(handle%,color_index%)
/
    DPOKE CONTRL+2,0
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,color_index%
/
    VDISYS 20
/
RETURN
/
/
/ SET POLYMARKER TYPE:
/
PROCEDURE vsm_type(handle%,symbol%)
/
    DPOKE CONTRL+2,0
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,symbol%
/
    VDISYS 18
/
RETURN
/
/
/ SET COLOR REPRESENTATION:
/
PROCEDURE vs_color(handle%,index%,rgb_in%)
/
    DPOKE CONTRL+2,0

```

```

/*****/
/**                               **/
/** Polymarker-Types           **/
/**                               **/
/*****/

#define MPUNKT      1
#define MPLUS       2
#define MSTERN      3
#define MQUADRAT    4
#define MDKREUZ     5
#define MDIAMANT    6

int contrl[12],
    intin[128],
    ptsin[128],
    intout[128],
    ptsout[128];

int gl_apid;
int gem_handle;
int handle;
int symbol;
int height;
int color_index;
int count;
int gl_wchar, gl_hchar, gl_wbox, gl_hbox;
int work_in[12],
    work_out[57],
    pxyarray[10];

```



```

int index,
    rgb_in[3];

/*****
/*      Initialisiere Applikation      */
*****/
aes_init()
{
    int i;
    gl_apid = appl_init();
    gem_handle = graf_handle(&gl_wchar, &gl_hchar,
                             &gl_wbox, &gl_hbox );
}

/*****
/*      Terminiere Applikation      */
*****/
aes_exit()
{
    gemdos(0x1);                /* Auf Taste warten */
    appl_exit();
}

/*****
/*      öffne virtuelle Workstation      */
*****/
open_vwork()
{
    int i;

    for(i=0; i<10; work_in[i++]=1);
    work_in[10] = 2;
    handle = gem handle;
    v_opnvwk(work_in, &handle, work_out);
}

/*****
/*      Schließe virtuelle Workstation      */
*****/

#define close_vwork()  v_clsvwk(handle);

/*****
/*      Hauptprogramm      */
*****/

main()
{
    int i;
    /* Farbnummern für vs color: */
    static int farben[8]= { 0, 71, 214, 357, 500, 643, 786, 929};
    int xkoord, ykoord;

```

```

int ind, rgb[3];
aes_init();
open_vwork();                /* Initialisierung */
v_clrwk(handle);             /* Bildschirm löschen */
symbol = MSTERN;
vsm_type(handle, symbol);    /* STERN als Symbol */
height = 20;
vsm_height(handle, height);  /* Höhe */

/* Sterne in verschiedenen Farben zeichnen: */
/*****
for (color_index=0, xkoord=10; xkoord<319; xkoord+=30)
    for (ykoord=10; ykoord<199; ykoord+=30)
    {
        if (++color_index==16)        /* Farbindex hochzählen */
            color_index=1;
        vsm_color(handle, color_index); /* Farbe setzen */
        pxyarray[0] = xkoord;          /* x-Wert Punkt */
        pxyarray[1] = ykoord;          /* y-Wert Punkt */
        count = 1;                     /* Anzahl Punkte = 1 */
        v_pmarker (handle, count, pxyarray);
    }

/* Farben der Sterne ändern: */
/*****
for (rgb[0]=1, ind=0; rgb[0]<8; rgb[0]++) /* Rot-Schleife */
    for (rgb[1]=1; rgb[1]<8; rgb[1]++) /* Nummern 0-7 hochzählen */
        for (rgb[2]=1; rgb[2]<8; rgb[2]++) /* Grün-Schleife */
            for (rgb[2]=1; rgb[2]<8; rgb[2]++) /* Nummern 0-7 hochzählen */
                for (rgb[2]=1; rgb[2]<8; rgb[2]++) /* Blau-Schleife */
                    for (rgb[2]=1; rgb[2]<8; rgb[2]++) /* Nummern 0-7 hochzählen */
                    {
                        if (++ind==16) /* Farbindex hochzählen */
                            ind=1;
                        /* Parameter übergeben: */
                        index=ind;
                        for (i=0; i<3; i++)
                            rgb_in[i]=farben[ rgb[i] ];
                        vs_color(handle, index, rgb_in); /* Farbe setzen */
                        gemdos(0x1); /* Auf Taste warten */
                    }

/* Terminierung: */
close_vwork();
aes_exit();
}

```

Wie im obigen BASIC-Programm werden zunächst (nach der üblichen Initialisierung) durch zwei ineinander verschachtelte FOR-Schleifen eine ganze Menge Sterne auf den niedrig aufgelösten Grafikbildschirm gezeichnet. Dabei werden die x- und y-Koordinaten bei jedem Schleifendurchlauf um jeweils 30 erhöht, um die Koordinaten für den nächsten Stern zu berechnen. Gleichzeitig

wird bei jedem Stern die Nummer des Palettenregisters geändert, aus dem der einzelne Stern seine Farbe bezieht. Auf diese Weise erscheinen die vielen Sterne zunächst einmal recht bunt (je nachdem, welche Farben Sie im Kontrollfeld eingestellt haben).

Erst jetzt tritt unsere neue Funktion in Kraft: In drei ineinander verschachtelten FOR-Schleifen zählen wir die Farbtintensitäten der drei Grundfarben von 1 bis 7 hoch (0 wird ausgelassen, da es sich um schwarz handelt). Diese Intensitäten (jeweils gespeichert in "rgb"[]) dienen als Index für die Tabelle "farben[]", die die Werte der obigen Tabelle enthält, sie wurde als "static int" für die "main"-Funktion definiert. Diese Werte schließlich werden dann den endgültigen Farbspeichern "rgb_in[]" zugewiesen und der Funktion "vs_color()" zusammen mit dem ebenfalls wechselnden Index übergeben.

Danach wartet das Programm auf Ihren Tastendruck, um die nächste Farbänderung vorzunehmen.

Neben der gerade vorgestellten VDI-Funktion gibt es noch eine XBIOS-Funktion, mit der Sie die Farbpalette ändern können. Dies geht dann allerdings nur für den Bildschirm. (Die Funktion SET COLOR REPRESENTATION kann bekanntlich für alle möglichen anderen Geräte - sofern vorhanden und unterstützt - verwendet werden.) Nun, die besagte XBIOS-Funktion nennt sich schlicht und greifend: SETPALETTE (6 XBIOS).

Wie Sie wissen, werden alle Eingabeparameter inklusive des Opcodes bei XBIOS-Funktionen auf dem Stack übergeben, das ist natürlich nur für Assembler wesentlich. Damit müßte ein Assembler-Aufruf etwa so aussehen:

```
...
move.l #palette,-(sp) ;Adresse der neuen Farbpalette auf den Stack
move.w #6,-(sp)      ;Opcode für Setpalette
trap #14              ;XBIOS-Aufruf
addq.l #6,sp          ;Stack restaurieren
...
palette dc.w $000,$012,$123,$234,$345,$456,$567,$670
          dc.w $700,$070,$007,$770,$707,$077,$777,$555
Entsprechend der C-Aufruf:
static int palette[16]=
    {0x000,0x012,0x123,0x234,0x345,0x456,0x567,0x670,
     0x700,0x070,0x007,0x770,0x707,0x077,0x777,0x555}
long int paletteptr = &palette
...
Setpalette(paletteptr);
Unter GFA-BASIC sähe das dann so aus:
s_palette$=string$(32,0)
FOR x%=0 TO 15
```

```
    READ palette%
    DPOKE VARPTR(s_palette$)+x%*2,palette%
NEXT x%
/
a=XBIOS(6,L:VARPTR(s_palette$))
/
DATA &h000,&h012,&h123,&h234,&h345,&h456,&h567,&h670
DATA &h700,&h070,&h007,&h770,&h707,&h077,&h777,&h555
```

Wie Sie sehen, müssen Sie der Funktion die Adresse einer ganz neuen Farbta-
belle übergeben, die insgesamt 16 Wörter enthält, die die RGB-Farbkombina-
tionen angeben. Im untersten Nibble (= unterste vier Bits) jedes Wortes steht
dann der Intensitätswert für die Grundfarbe Blau, dann folgt der für die Farbe
Grün. Im dritten Nibble schließlich wird der Intensitätswert für Rot angege-
ben, das oberste Nibble bleibt ungenutzt. Nach dem Aufruf sind sämtliche
Farbregister entsprechend der Tabelle geändert.

Wichtig hierbei ist die Tatsache, daß es sich bei der Reihenfolge der Paletten-
werte um die originale Reihenfolge der Palettenregister im Rechner handelt -
nicht also um die Reihenfolge gemäß VDI-Farbindex!

Ebenfalls die originalen Farbregisterwerte verwendet die XBIOS-Funktion
SETCOLOR (Nr. 7), mit der Sie (ähnlich der Funktion SET COLOR REPRESENTATION)
nur ein einziges Farbregister ändern können. Der entsprechende
BASIC-Befehl:

SETCOLOR fn,f (GFA-BASIC-Befehl)

SETCOLOR fn,fr,fg,fb

SETCOLOR; Ändern einer Palettenfarbe

fn: Farbregisternummer
f: RGB-Farbintensitätswert
fr: Rot-Intensitätsanteil
fg: Grün-Intensitätsanteil
fb: Blau-Intensitätsanteil

Dieser Befehl bietet gleich zwei Möglichkeiten: Entweder die Farbangabe in
einem Wert (dieser Wert ist genauso aufgebaut wie bereits besprochen) oder
die Angabe der drei Einzelintensitäten. Schreiben Sie doch einmal eines der
obigen Programme unter Verwendung dieser Funktionen um!

Wir werden uns im Laufe des Buches noch mit einer Reihe anderer Funktionen
und Möglichkeiten der Farbeeinflussung beschäftigen. Einen VDI-Befehl je-

doch möchte ich Ihnen an dieser Stelle noch vorstellen: CELL ARRAY (10 VDI). Diese Funktion soll folgende Aufgaben erfüllen:

Es wird ein logisches Rechteck in weitere Rechtecke unterteilt. Jedem dieser kleineren Rechtecke kann ein Farbindex im Farbindexfeld zugeordnet werden. Alle Punkte, die dann innerhalb des großen Rechteckes liegen, nehmen automatisch die Farbe der inneren Rechtecke an.

Durch die Funktion OPEN VIRTUAL SCREEN WORKSTATION wird in "intout(38)" angegeben, ob die Funktion CELL ARRAY für das betreffende Gerät verfügbar ist. Der Parameter "intout(38)" ist gleich 0, wenn dies nicht, und gleich 1, wenn dies doch der Fall ist. In der mir zur Verfügung stehenden GEM-Version war die Funktion CELL ARRAY weder auf dem monochromen noch auf dem Farbmonitor verfügbar. Aus diesem Grunde werden wir uns auch nicht weiter mit ihr beschäftigen.

15.2.3 Die Zeichenmodi

Wir wissen nun, wie wir einen Punkt oder bestimmte andere Markierungen auf den Bildschirm bringen können. Wir wissen ferner, wie wir mit den verschiedenen Möglichkeiten umgehen, Farben ins Bild zu bringen. Eine weitere Modifikationsmöglichkeit, die sich auf alle zukünftigen programmierbaren Figuren bzw. Grafikfunktionen bezieht, ist die Wahl des Zeichenmodus. Wir verstehen unter dem Zeichenmodus die Art und Weise, auf die bereits auf dem Bildschirm befindliche Grafik von neu eingezeichneten Punkten oder Figuren beeinflusst wird. VDI bietet uns da einen interessanten Befehl an, der Ihnen nicht unterschlagen werden soll. In vielen Büchern zum Atari ST werden oft recht unverständliche, ja sogar ziemlich fehlerhafte Beschreibungen dieser Zeichenmodi geliefert. Dies ist wahrscheinlich die Folge daraus, daß die verschiedenen Funktionen des VDI von den Buchautoren nicht ausprobiert werden. Ich hoffe, die folgenden Beschreibungen sind etwas verständlicher. Sie sind aber in jedem Fall fehlerfrei, da sie von mir "untersucht" wurden.

Die VDI-Funktion SET WRITING MODE dient zum Festlegen des Zeichenmodus für alle folgenden Grafikoperationen. Die entsprechende GFA-Funktion lautet:

GRAPHMODE m (GFA-BASIC-Befehl)

Festlegen des Grafikmodus

- m: Grafikmodus:
 - m=1 = Replace
 - m=2 = Transparent
 - m=3 = XOR (Revers)
 - m=4 = Reverse Transparent

Sie sehen, die Funktionen stellen uns vier verschiedene Zeichenmodi zur Verfügung, die wir natürlich beschreiben wollen. Vorher muß jedoch ein zentraler Begriff geklärt werden: die Objektmaske.

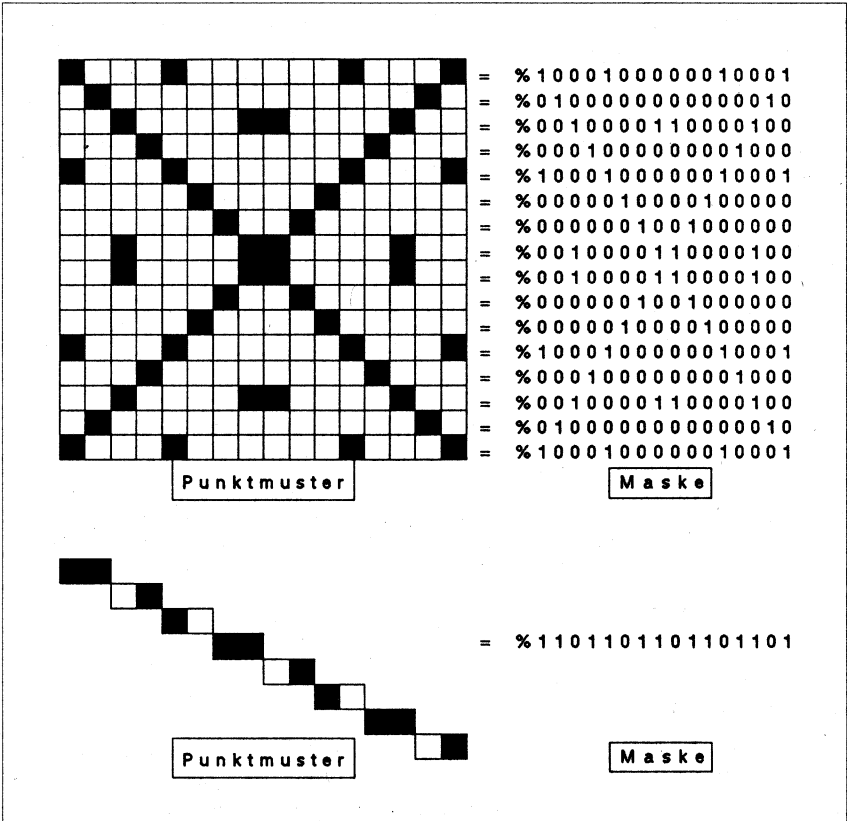


Abb. 15.8: Linien- und Flächenmaske

Wir werden im Laufe des Buches einige Grafikobjekte kennenlernen (Linie, Fläche,...), die eine Menge von Gestaltungsvariationen erlauben. Eine davon (neben Farbe etc.) ist die Objektmaske. Leider haben wir noch keine Funktion kennengelernt, die uns ein Grafikobjekt auf den Bildschirm bringt, das eine variable Maske besitzt. Trotzdem kommen wir an dieser Stelle nicht um diesen Begriff herum.

Stellen Sie sich ein beliebiges Muster vor, das wir auf den Bildschirm bringen. (Wir werden später sehen, wie ein solches Muster erzeugt werden kann.) In diesem Muster sind Punkte entweder gesetzt oder gelöscht. Zwar können wir den gesetzten Punkten eine bestimmte Farbe zuordnen, das spielt hier in diesem Zusammenhang jedoch noch keine Rolle. Die Information nun, welcher Punkt gesetzt und welcher nicht gesetzt wird, wird durch gesetzte bzw. nicht gesetzte Bits innerhalb eines Wortes kodiert, das seinerseits wiederum ein Teil der Gesamt-Musterdefinition ist. Kurz: Wir definieren das Muster in einer sogenannten Bit-Maske. Für jede Farbebene kann sogar eine getrennte Maske existieren. Ähnliches findet bei Linien statt, die über eine solche Maske (z.B. für gestrichelte Linien) gezeichnet werden. Bemerkenswert ist bei dieser Organisation das Setzen oder Löschen eines einzelnen Punktes mittels der POLY-MARKER-Funktion. Die Objektmaske setzt sich nämlich hierbei nur aus einem gesetzten Bit (=ein Punkt) zusammen und kann nicht geändert werden. Auch wenn Sie also in Hintergrundfarbe einen Punkt quasi unsichtbar machen, so gilt dieser Punkt trotzdem als ein zu setzender.

Doch nun starten wir mit unseren Erläuterungen:

a) Replace-Modus:

Dies ist die normalerweise eingestellte Art und Weise, Punkte auf den Bildschirm zu bringen. In diesem Modus wird im wahrsten Sinne des Wortes "ohne Rücksicht auf Verluste" gezeichnet. Die zu zeichnenden oder auch zu löschenden Punkte werden in die Grafik gesetzt, ohne eventuell bereits gezeichnete Punkte zu berücksichtigen. Auch die Farbe der zu setzenden Punkte ändert sich nicht in Abhängigkeit von der Farbe der bereits auf dem Bildschirm befindlichen Punkte, wie das bei den anderen Modi z.T. der Fall ist. Ein Muster wird also völlig den darunter liegenden Bildschirmteil überdecken.

b) Transparent-Modus:

Nun wird es interessanter - aber auch schwieriger. Der Transparent-Modus ist allgemein mit einer ODER-Verknüpfung des zu zeichnenden Masken(!)punktes und des bereits im Bildschirmspeicher befindlichen Punktes zu vergleichen.

Jeder in der Objekt-Maske gesetzte Punkt erscheint auch - ohne Wenn und Aber - auf dem Bildschirm. Er erscheint dabei in der ihm zugeordneten Farbe. Alle Punkte jedoch, die in der Maske nicht gesetzt sind, bleiben völlig unberücksichtigt. Das hat zur Folge, daß das Muster an den "leeren" Stellen durchsichtig (transparent) erscheint. Noch einmal zu bemerken ist, daß ein einzelner durch POLYMARKER gesetzter Punkt stets als zu setzender Punkt gilt - egal, welche Farbe er besitzt. Für einen solchen Punkt besteht somit kein Unterschied zum Replace-Modus.

Die Aufgabe eines solchen Transparent-Modus ist schnell erläutert: Wenn wir bestimmte Figuren auf den Bildschirm zeichnen wollen, die aus mehreren Farben bestehen (z.B. Buchstaben oder gestrichelte Linien), so werden diese nicht einfach eingezeichnet. Vielmehr schaut das bisher dort Stehende noch durch die übergezeichnete Figur hindurch. Das heißt also, die nicht gesetzten Punkte eines Buchstabenkastens lassen die bisher an dieser Stelle stehenden Punkte des Bildschirms durchscheinen.

c) XOR-Modus:

Auch dieser Modus bewirkt eine logische Verknüpfung des zu zeichnenden Masken(!)punktes mit der Farbe(!) des bereits auf dem Bildschirm stehenden Punktes. In diesem Fall handelt es sich um eine Exklusiv-ODER-Verknüpfung. Doch differenzieren wir hier - der Anschaulichkeit halber - in monochrome und farbige Darstellung:

- Monochrom:

Soll an eine Stelle ein in der Maske gesetzter Punkt gesetzt werden, so wird der Zustand des an dieser Stelle bereits befindlichen Punktes umgekehrt. D.h. befand sich an dieser Stelle ein Punkt in Zeichenfarbe, so wechselt er zur Hintergrundfarbe, war es ein Punkt in Hintergrundfarbe, so wechselt er zur Zeichenfarbe. Es findet also ein Farbwechsel statt. Dieser Farbwechsel findet - wie immer - auch dann statt, wenn ein Punkt die Hintergrundfarbe besitzt (also eigentlich löscht), in der Farbmaske jedoch als gesetzt gilt!

Soll jedoch ein in der Maske gesetzter Punkt in den Bildschirm gezeichnet werden, so ändert sich nichts.

Kreuzen sich also beispielsweise zwei durchgezogene Linien, die beide normale Zeichenfarbe besitzen, so wird der Schnittpunkt in der Hintergrundfarbe erscheinen.

- Farbe:

Hier müssen wir uns die einzelnen Punkte als Bit-Muster im Speicher vorstellen, das die Nummer des Farbregisters bezeichnet, aus dem der Punkt seine Farbe bezieht. Trifft nun der zu zeichnende Punkt auf einen bereits im Speicher befindlichen, so wird eine Exklusiv-ODER-Verknüpfung (EXOR oder XOR) zwischen der Farbnummer des bereits gesetzten Punktes und der Maske des zu setzenden Punktes hergestellt. Nehmen wir das Beispiel von oben: Ein in der Maske gesetzter Punkt trifft auf einen mit Farbe %0101 (5):

%1111	- Auf vier Bits erweiterte Maske
EXOR %0101	- Farb-Bitmuster
%1010	- Resultierendes Farb-Bitmuster

Der Ergebnispunkt hat also die Farbe des Registers mit der Nummer %1010 (dezimal: 10). Sie sehen also, jedes einzelne Masken-Bit wird auf vier (bei NGR) oder zwei (bei MGR) Bits erweitert und dann mit dem Farbmuster des zu überschreibenden Punktes verknüpft. Damit wird stets eine komplementäre Farbe zur Ursprungsfarbe das Ergebnis sein.

Ist dagegen ein zu setzender Punkt in der Objekt-Maske nicht gesetzt, so hat er keinerlei Einfluß auf den bereits an der betreffenden Stelle liegenden Punkt.

Bemerkenswert ist die Tatsache, daß die Farbe des zu setzenden Punktes keinerlei Rolle spielt.

Der Vorteil dieses Zeichenmodus besteht darin, daß jede Zeichenoperation durch Wiederholung rückgängig gemacht werden kann. Sie können also unabhängig vom bereits bestehenden Bildschirminhalt Figuren (z.B. den Mauszeiger) auf den Bildschirm setzen. Beim nochmaligen Zeichnen dieser Figur wird der ursprüngliche Bildschirminhalt wieder automatisch hergestellt. Auf diese Weise werden z.B. Cursor oder Mauszeiger gezeichnet und wieder gelöscht. Damit funktioniert das Ganze auch mit dem POLYMARKER-Befehl, da hier bekanntlich stets ein Punkt gesetzt wird, stets also invertiert wird.

d) Reverse-Transparent-Modus:

Dies ist der komplizierteste aller vier Modi, dennoch recht einfach zu erklären: Zunächst wird die Maske des zu zeichnenden Punktes umgedreht (aus 1 (gesetzt) wird 0 (gelöscht) und umgekehrt). Mit diesem neuen Punkt wird nun exakt dasselbe durchgeführt wie beim Transparent-Modus. Wir erhalten dann praktisch im Transparent-Modus das Negativ der Ursprungsfigur auf dem Bildschirm.

Um Ihnen die Anwendung dieses neuen Befehls ein wenig vor Augen zu führen, wollen wir hier wieder einige kleine Programme vorführen (jeweils in C und in BASIC), die sich mit diesem Thema beschäftigen. Zunächst folgen ein paar kleine BASIC-Routinen, die Ihnen die vier verschiedenen Modi etwas transparenter machen sollen:

```
' *****
' **                               **
' **   Der Grafikmodus   **
' **                               **
' *****
'
DIM x%(1),y%(1)
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
CLS                               !Bildschirm löschen
'
xg%=xauf1%/2                       !Arbeitswerte berechnen
yg%=yauf1%/2
'
' 4 Kästen (Rahmen) zeichnen:
GRAPHMODE 1
COLOR 1
FOR x%=0 TO xg% STEP xg%/2
  FOR y%=0 TO yg%
    PLOT x%,y%
  NEXT y%
NEXT x%
'
FOR y%=0 TO yg% STEP yg%/2
  FOR x%=0 TO xg%
    PLOT x%,y%
  NEXT x%
NEXT y%
'
' 4 gestreifte Kästen zeichnen:
@ksten(5,5,xg%/2-5,yg%/2-5,1,3)
@ksten(xg%/2+5,5,xg%-5,yg%/2-5,1,3)
@ksten(5,yg%/2+5,xg%/2-5,yg%-5,1,3)
@ksten(xg%/2+5,yg%/2+5,xg%-5,yg%-5,1,3)
'
'
dritt%=(yg%-10)/6                  !Arbeitswert berechnen
'
DEFLINE -&X11111111                !Punktmuster: 000000011111111
'
' Kasten 1: Replace-Modus
GRAPHMODE 1
FOR n%=1 TO 2
```

```

COLOR 2-n%
@kasten(5,5+n%*dritt%,xg%/2-5,5+(n%+1)*dritt%-1,1,1)
NEXT n%
'
' Kasten 2: Transparent-Modus
GRAPHMODE 2
FOR n%=1 TO 2
  COLOR 2-n%
  @kasten(xg%/2+5,5+n%*dritt%,xg%-5,5+(n%+1)*dritt%-1,1,1)
NEXT n%
'
' Kasten 3: XOR-Modus
GRAPHMODE 3
FOR n%=1 TO 2
  COLOR 2-n%
  @kasten(5,10+(n%+3)*dritt%,xg%/2-5,10+(n%+4)*dritt%-1,1,1)
NEXT n%
'
' Kasten 4: Invers-Transparent-Modus
GRAPHMODE 4
FOR n%=1 TO 2
  COLOR 2-n%
  @kasten(xg%/2+5,10+(n%+3)*dritt%,xg%-5,10+(n%+4)*dritt%-1,1,1)
NEXT n%
'
c=INP(2)          !Auf Taste warten
'
EDIT
'
'
'
' Prozedur: Kasten zeichnen:
PROCEDURE kasten(x1%,y1%,x2%,y2%,xnetz%,ynetz%)
'
  FOR x%=x1% TO x2% STEP xnetz%
    FOR y%=y1% TO y2% STEP ynetz%
      PLOT x%,y%
    NEXT y%
  NEXT x%
RETURN

```

Nun, das Programm sieht aufgrund der vielen komplizierten Rechnungen viel schwieriger aus als es ist. Was da im einzelnen alles berechnet wird, das soll uns hier nämlich gar nicht interessieren. Einzig und allein interessant ist der Effekt auf dem Bildschirm. Stören sollten Sie sich auch nicht daran, daß in diesem Programm Befehle bzw. Grafikoperationen verwendet wurden, die wir noch nicht besprochen haben. Lassen Sie sich einfach nur von den folgenden Ausführungen und dem Programmergebnis leiten:

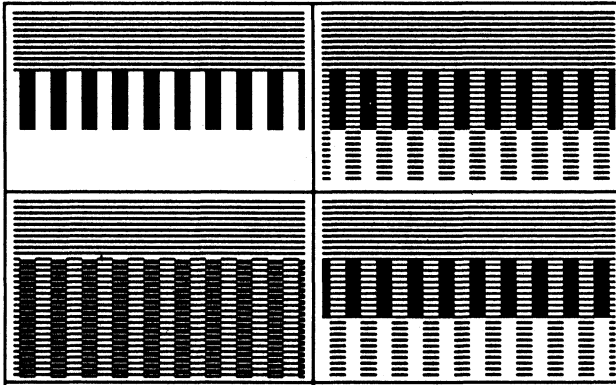


Abb. 15.9: Der Grafikmodus

Bevor Sie jedoch das Programm starten, stellen Sie doch bitte am Programm-anfang die von Ihnen verwendete Auflösung fest, und tragen Sie sie dort ein. Hier wurde zunächst von einer hohen Auflösung mit monochromer Darstellung ausgegangen.

Zunächst einmal werden - der schönen Optik wegen - vier Rahmen auf den Bildschirm gezeichnet. Dann folgen vier gestreifte Kästen. Sie dienen uns als Hintergrund. Jetzt wird es interessant. Wir legen nämlich ein eigenes Linienmuster bzw. eine Linienmaske fest:

```
0000000011111111
```

An diese 16 Bit lange Linienmaske hält sich nämlich im Gegensatz zum POLYMARK- der PLOT-Befehl des GFA-BASICs. Er ist - wie wir im nächsten großen Kapitel sehen werden - eigentlich ein verkappter Befehl zum Zeichnen einer Linie. Man muß sich den gesamten Bildschirm nun so vorstellen, als sei er mit dieser Maske vollgeschrieben. Wenn nun ein Punkt mit PLOT gesetzt wird, so wird er nur dort wirklich gesetzt, wo die Maske gerade 1 ist. Ansonsten gilt er als gelöscht Punkt.

Wie sich das dann in den einzelnen Zeichenmodi darstellt, das zeigt Ihnen das Programm. Hier sehen Sie, wie sich die einzelnen Punkte unter Berücksichtigung der gestreiften Kästen als Hintergrund verhalten. Vergleichen Sie einmal das Ergebnis mit den theoretischen Beschreibungen von oben.

Um Ihnen zu verdeutlichen, wie das alles in Farbe funktioniert, sei nun das folgende kleine BASIC-Programm, das nur unter NGR (320x200) funktioniert, vorgestellt:

```

' *****
' **                               **
' ** Zeichenmodi in Farbe **
' **                               **
' *****
'
SETCOLOR 0,0                !Hintergrundfarbe auf Schwarz
SETCOLOR 1,7,7,7
'
FOR n=1 TO 4                !Modus-Schleife
  CLS                       !Bildschirm löschen
  '
  GRAPHMODE 1               !Replace-Modus ein
  FOR y=0 TO 14             !15 Balken in 15 Farben
    DEFFILL y+1,1           !Einstellen: Farbe, gefüllter Balken
    PBOX 0,y*13,319,y*13+10 !waagerechten Balken zeichnen
  NEXT y
  '
  GRAPHMODE n               !Grafikmodus setzen
  FOR x=0 TO 14             !15 Balken in 15 Farben
    DEFFILL x+1,2,9         !Einstellen: Farbe, Mauermuster
    PBOX x*21,0,x*21+15,199 !senkrechten Balken zeichnen
  NEXT x
  '
  c=INP(2)                  !Auf Taste warten
NEXT n
'
SETCOLOR 0,7,7,7           !Hintergrund wieder weiß
EDIT                       !sofort in Editor zurück

```

Auch hier sollten Sie sich nicht daran stören, daß wir die Befehle DEFFILL und PBOX verwenden, um ein mit einem Muster ausgefülltes Rechteck auf den Bildschirm zu bringen. Wichtig ist hier wiederum das Bildschirmergebnis.

Hier werden - auf schwarzem Hintergrund - zunächst einmal 15 waagerechte, völlig ausgefüllte Balken in 15 verschiedenen Farbindizes gezeichnet. Sie werden dann von 15 weiteren senkrechten und ebenfalls verschiedenfarbigen Balken überschritten. Letztere Balken werden jeweils auf Tastendruck in den vier verschiedenen Zeichenmodi aufgebracht. Hier können Sie direkt sehen, wie sich welche Farben wie auf welchem Hintergrund verhalten. Studieren Sie die Bildschirmausgabe genau, und vergleichen Sie sie wieder mit dem oben Gesagten.

Zum Abschluß sei ein kleines C-Programm angeführt, das die Grundlage für Ihr nächstes CAD-Programm sein könnte. Es macht Ihnen deutlich, wie ein Cursor im XOR-Modus durch zweimaliges Zeichnen wieder vom Bildschirm verschwindet, ohne den Bildschirminhalt zu zerstören. Es sollte mit Ihrem jetzigen Wissen nicht so schwer sein, dieses C-Programm mit Hilfe eines normalen C-Lehrbuches relativ schnell in ein BASIC-Programm umzuformen, zumal

es gut dokumentiert ist. Die BASIC-Fans unter Ihnen sollten sich also einmal daran versuchen, es lohnt sich:

```
/* **** */
/*      */
/*  Z E I C H N E R  */
/*      */
/* **** */

/* Polymarker-Types */
#define MPUNKT      1
#define MPLUS       2
#define MSTERN      3
#define MQUADRAT    4
#define MDKREUZ     5
#define MDIAMANT    6

int contrl[12],
    intin[128],
    ptsin[128],
    intout[128],
    ptsout[128];
int gl_apid;
int gem_handle;
int handle;
int symbol;
int height;
int color_index;
int count;
int mode;
int gl_wchar, gl_hchar, gl_wbox, gl_hbox;
int work_in[12],
    work_out[57],
    pxyarray[10];
int index,
    rgb_in[3];
int xaupfl, yaupfl;

/* **** Initialisiere Applikation **** */
aes_init()
{
    int i;
    gl_apid = appl_init();
    gem_handle = graf_handle(&gl_wchar, &gl_hchar,
                           &gl_wbox, &gl_hbox );
}

/* **** Terminiere Applikation **** */
```

```

aes_exit()
{
    gemdos(0x7);          /* Auf Taste warten */
    appl_exit();
}

/*****
/*      Öffne virtuelle Workstation      */
*****/
open_vwork()
{
    int i;

    for(i=0; i<10; work_in[i++]=1);
    work_in[10] = 2;
    handle = gem_handle;
    v_opnvwk(work_in, &handle, work_out);
}

/*****
/*      Schließe virtuelle Workstation    */
*****/

#define close_vwork() v_clsvwk(handle);

/*****
/*      Hauptprogramm                      */
*****/

main()
{
    int xkoord, ykoord;
    int xalt, yalt;
    unsigned int flag, plotflag;
    int speed;
    aes_init();
    open_vwork();          /* Initialisierung          */
    xafl = work_out[0];
    yafl = work_out[1];    /* Bildschirmauflösung feststellen */
    v_clrwk(handle);       /* Bildschirm löschen          */
    symbol = MPUNKT;
    vsm_type(handle, symbol); /* PUNKT als Symbol          */
    color_index=1;
    vsm_color(handle, color_index);
    /* Netz zeichnen: */
    /*****/
    for (xkoord=0; xkoord<=xafl; xkoord+=30)
        for (ykoord=0; ykoord<=yafl; ykoord++)
            marker(xkoord, ykoord); /* Einen Punkt setzen          */
    for (ykoord=0; ykoord<=yafl; ykoord+=30)
        for (xkoord=0; xkoord<=xafl; xkoord++)
            marker(xkoord, ykoord); /* Einen Punkt setzen          */
}

```

```
/* Stern über Bildschirm bewegen: */
/*****/
symbol = MSTERN;
vsm type(handle, symbol);      /* STERN als Symbol          */
height = 20;
vsm height(handle, height);    /* Höhe                  */
mode=3;
vswr mode(handle, mode);       /* Zeichenmodus=XOR      */
xkoord = (xauf1+1)/2;
ykoord = (yauf1+1)/2;          /* Stern zur Bildschirmmitte */
marker(xkoord, ykoord);        /* Stern setzen           */
flag=0;
plotflag=0;
speed=1;
/* Schleife solange wie definierte Tasten: */
while (flag == 0)
{
  xalt = xkoord;
  yalt = ykoord;                /* Koordinaten merken          */
  switch (gemdos(0x7) & 0xff)    /* Zeichen holen               */
  {
    case '8':                    /* hoch                        */
      ykoord -= speed;
      break;
    case '2':                    /* Cursor runter               */
      ykoord += speed;
      break;
    case '4':                    /* Cursor links                 */
      xkoord -= speed;
      break;
    case '6':                    /* Cursor rechts                */
      xkoord += speed;
      break;
    case '5':                    /* Zeichnen ein/aus            */
      plotflag += 0x8000;       /* zwischen 0 und $8000 wechseln */
      break;
    case '*':                    /* Geschwindigkeit groß/klein  */
      if (speed == 5)
        speed=1;               /* klein                        */
      else
        speed=5;               /* groß                         */
      break;
    default:
      flag=1;
  }
  /* Koordinaten überprüfen: */
  if (xkoord < 0)
    xkoord = xauf1;
  if (xkoord > xauf1)
    xkoord = 0;
  if (ykoord < 0)
    ykoord = yauf1;
  if (ykoord > yauf1)
```



```

    ykoord = 0;
    if (plotflag == 0x8000) /* Zeichenflag? */
    /* Punkt an Cursorstelle plotten: */
    {
        symbol = MPUNKT;
        vsm_type(handle, symbol); /* PUNKT als Symbol */
        marker(xkoord, ykoord); /* Punkt setzen */
        symbol = MSTERN;
        vsm_type(handle, symbol); /* wieder STERN als Symbol */
    }
    marker(xalt,yalt); /* Alten Stern löschen */
    /* durch nochmaliges Zeichnen */
    marker(xkoord, ykoord); /* Stern setzen */
}

/* Terminierung: */
close_vwork();
aes_exit();
}

/* Einen Marker auf Bildschirm setzen: */
marker(x,y)
int x, y;
{
    count=1;
    pxyarray[0] = x,
    pxyarray[1] = y;
    v_pmarker(handle, count, pxyarray);
}

```

Wir haben es bei dieser C-Applikation mit einem kleinen Benutzer-gesteuerten Programm zu tun, das zunächst einmal ein Netz auf den Bildschirm zeichnet. Dies wird mit Hilfe der beiden doppelten FOR-Schleifen bewältigt. Innerhalb dieser Schleifen rufen wir die C-Funktion `marker(x,y)` auf, die an eine bestimmte Stelle auf dem Bildschirm eine Markierung (z.B. einen Punkt) setzt. Durch Verwendung dieser Funktion ersparen wir uns die umständliche Initialisierung der `POLYMARKER`-Funktion.

Damit unser Programm jedoch in allen Auflösungen funktioniert, haben wir diese noch etwas früher festgestellt. Die notwendigen Werte (x- und y-Auflösung) liefert uns die gute alte Funktion `OPEN VIRTUAL SCREEN WORKSTATION`. Mit Hilfe der Anzahl der möglichen Punkte in x- und y-Richtung konnten wir z.B. das Netz in der richtigen Größe zeichnen.

Nachdem nun das Netz auf den Bildschirm gezeichnet ist (wir werden hierzu später natürlich noch viel einfachere und bedeutend schnellere Methoden kennenlernen), wenden wir uns unserer eigentlichen Aufgabe zu, dem Bewegen eines Cursors über den Bildschirm.

Die Form, Größe und Start-Position dieses Cursors, der ebenfalls mit der POLYMARKER-Funktion gezeichnet werden soll, wird zunächst festgelegt. Dann wird er in Form eines Sternes sichtbar gemacht. Dies können wir wieder mit Hilfe der Funktion "marker()" realisieren.

Doch bevor dies geschieht, schalten wir den Zeichenmodus mit unserer neuen Funktion "vswr_mode()" auf XOR. Wenn wir nun also irgendwann einmal den so gezeichneten Stern noch einmal an diese Position zeichnen, so verschwindet er wieder, ohne Spuren zu hinterlassen.

Die Bewegung dieses Sternchens wird nun in einer großen while-Schleife realisiert, in der durch Aufruf der Funktion "gemdos(0x7)" die Tastatur abgefragt wird. Je nachdem, welche Taste nun betätigt wird, werden die Koordinaten des Sternes verändert. (Die alten Koordinaten des Sterns werden vorher natürlich in zwei Speichern mit den Namen xalt und yalt zwischengespeichert.) Die so veränderten Koordinaten werden zunächst überprüft, ob sie sich überhaupt noch auf dem Bildschirm befinden. Im Falle des Falles werden sie wieder auf den Bildschirm gebracht.

Hier folgt die Abfrage des sogenannten "Plot-Flags". Dieses Flag gibt an, ob sich das Programm im Zeichenmodus befindet oder nicht. Ist dies der Fall, so wird an die Stelle des Grafik-Cursors ein Punkt gesetzt (oder gelöscht).

Nun kann mit den alten, oben zwischengespeicherten Koordinaten der Stern noch einmal an derselben Stelle gezeichnet werden, was ein Löschen desselben bewirkt. (Wir befinden uns bekanntlich im XOR-Modus.) Erst jetzt werden die neuen Koordinaten dazu verwendet, den Cursor an der aktuellen Stelle zu zeichnen.

Das Ganze geht so lange, bis eine nicht definierte Taste betätigt wird. Folgende Tasten des Ziffernblocks sind erlaubt:

- 8 - Cursor hoch bewegen
- 2 - Cursor hinunter bewegen
- 4 - Cursor nach links bewegen
- 6 - Cursor nach rechts bewegen
- 5 - Plot-Modus ein/aus
- * - Bewegungsgeschwindigkeit des Cursors auf schnell/langsam

Mit diesem Programm können Sie bereits ein wenig spielen. Es ist sozusagen das erste Grundgerüst eines Zeichenprogramms (das allererste!).

15.3 Die Linie - Grundlage aller Grafik

So, wie im ideellen Sinne der Punkt als das kleinste unteilbare Grafik-Teilchen angesehen werden kann, so stellt die Linie in der Praxis das grundlegendste und wohl wichtigste grafische Element dar. So kann ein Punkt gleichwohl als Sonderfall einer Linie angesehen werden - wie es im VDI auch gehandhabt wird. Eine Reihe von Punkten jedoch zu einer Linie zu formen, dazu gehört bereits einiges mehr. Doch auch die anderen grafischen Objekte wie Kreise, Ellipsen, Kurven und viele mehr werden meist aus einer Vielzahl aneinanderhängender Linien gezeichnet. Damit steigt die Bedeutung einer schnellen internen Linienroutine um ein Vielfaches. Je schneller ein System eine Linie zeichnen kann, desto schneller werden meist auch die anderen grafischen Figuren auf dem Bildschirm darstellbar.

Nun, VDI gibt uns nicht einfach eine Funktion an die Hand, mit der wir möglichst reibungslos einfache Linien in die Gegend zeichnen können. Unter VDI haben wir mannigfaltige Möglichkeiten, diese Linien zu gestalten. Ob dick oder dünn, ob gestrichelt oder gar gepunktet - alles wird nach unseren Wünschen erledigt.

Schauen wir doch einmal hinein in diese, demnächst unsere Welt der Linien.

15.3.1 Linien, Linienzüge, Polygone

Zum Linienziehen dienen im VDI die Funktion POLYLINE (6) und im GFA-BASIC:

`POLYLINE count,x(),y() (OFFSET ox,oy) (GFA-BASIC-Befehl)`

Verbinden von Punkten

`count`: Anzahl der Punkte

`x()`: Array mit allen x-Koordinaten

`y()`: Array mit allen y-Koordinaten

`ox`: x-Verschiebewert

`oy`: y-Verschiebewert

Da haben wir ihn, den Linienbefehl, jeweils als VDI-Funktion und als GFA-BASIC-Befehl. Von Stund' an brauchen wir uns nicht mehr mit der langsamen POLYMARKER-Funktion herumzuärgern.

Der POLYLINE-Befehl ermöglicht uns nicht nur, einzelne Linien zu zeichnen, er bietet uns gleichzeitig die Möglichkeit, ganze Linienscharen zu einem Linienzug miteinander zu verbinden. Die Punkte, die jeweils nacheinander durch Linien verbunden werden sollen, werden im PTSIN-Array (für C-Freunde: PXYARRAY) je mit x- und y-Koordinate abgelegt. Die Anzahl dieser Punkte geben wir mit `count (intin(0))` an, und ab geht die Post.

Recht raffiniert ist das im GFA-BASIC gelöst. Hier geben Sie neben der Punktezahl jeweils ein Array an, in dem die x-Koordinaten, und eines, in dem die y-Koordinaten der zu verbindenden Punkte gespeichert werden. Zusätzlich sind Sie aber in der glücklichen Lage, einen sogenannten Offset zu den x- und y-Koordinaten hinzuzufügen. Dies sind zwei Werte, die zu allen x- bzw. y-Koordinaten addiert werden. Auf diese Weise ist es ein einfaches Unterfangen, eine in den Arrays definierte Figur auf dem Bildschirm zu verschieben. Wir werden das an dem folgenden Beispiel sehen.

C- oder Assembler-Fans werden wahrscheinlich kaum Probleme haben, eine solche Funktion auch für ihre Zwecke zu programmieren. Sie brauchen lediglich zu sämtlichen Koordinatenwerten im PXYARRAY- bzw. PTSIN-Array entsprechende Konstanten zu addieren - fertig. Wir werden dies ein wenig später bei unserem C-Beispiel zum Thema Linien & Co. demonstrieren.

```
' *****  
' **  
' ** Sterne am Horizont **  
' **  
' *****  
'  
'  
' Bildschirmauflösung:  
xauf1%=DPEEK(&H2686)  
yauf1%=DPEEK(&H2688)
```

```

'
xgroesse%=30*(xauf1%+1)/640      !Sterngröße festlegen
ygroesse%=30*(yauf1%+1)/400
'
DIM x%(20),y%(20)                !Koordinaten-Array dimensionieren
'
' Eckpunkte einlesen:
READ anzahl
FOR i%=0 TO anzahl-1
    READ x%(i%),y%(i%)            !Koord. aus DATA-Zeilen lesen
    x%(i%)=x%(i%)*xgroesse%      !richtige Größe berechnen
    y%(i%)=y%(i%)*ygroesse%
NEXT i%
'
ya2%=yauf1%/3                    !Hilfsspeicher
pe=80*(xauf1%+1)/640
'
' Hauptschleife:
' (Abstände ständig vergrößern)
'
FOR s%=1 TO 350                  !Abstandsschleife
    CLS                          !Bildschirm löschen
    '
    FOR x%=0 TO xauf1%/s%        !Zeichenschleife
        '
        ' Gebilde zeichnen:
        POLYLINE anzahl,x%(),y%() OFFSET x%*s%,ya2%*SIN(x%*s%/pe)+ya2%-10
    NEXT x%
NEXT s%
'
c=INP(2)                        !Auf Taste warten
EDIT
'
' relative Eck-Koordinaten des Sterns:
'
' Anzahl der Eckpunkte:
DATA 13
'
' Koordinaten:
DATA 0,1, 1,2, 0,3, 1,3, 2,4
DATA 3,3, 4,3, 3,2, 4,1, 3,1
DATA 2,0, 1,1, 0,1

```

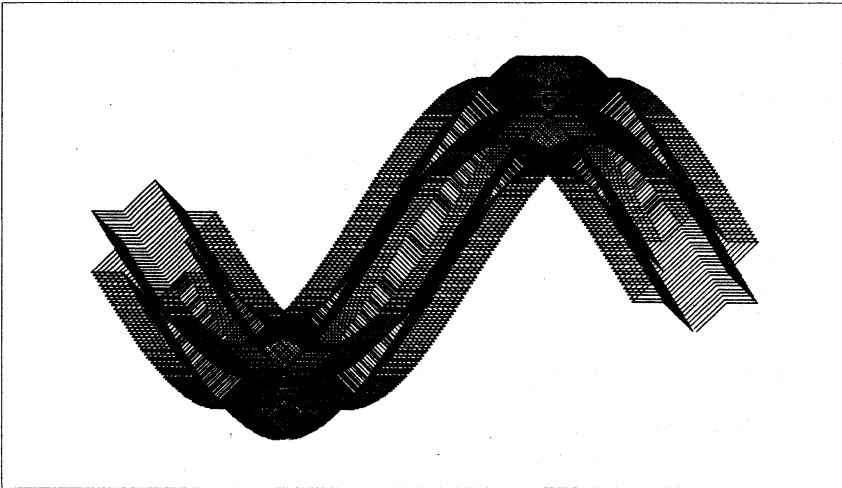


Abb. 15.10: Sterne am Horizont

Dieses kleine, aber sehenswerte Programm zeichnet uns eine geschwungene Reihe schöner, großer Sterne auf den Bildschirm, die sich stets verändert. Zentrum des ganzen Programms ist die Definition des Sternes in den DATA-Zeilen am Programmende. Hier wird nur ein ganz winziger Stern in der Größe von fünf[^] Punkten definiert, um erst im Programm nachträglich auf die Größe vergrößert zu werden, die angenehm ist. Die x- und y-Vergrößerungsfaktoren "xgroesse%" und "ygroesse%" des Sternes werden direkt am Programmanfang unter Berücksichtigung der Bildschirmauflösung berechnet. Die komplizierte Formel hierzu sollte Sie nicht weiter stören. Solche Umstände werden später noch öfter auf uns zukommen, damit die Programme mit allen Auflösungen lauffähig sind.

Als nächstes werden die DATA-Koordinatenwerte in die Koordinaten-Arrays eingelesen und gleichzeitig vergrößert. Jetzt erst beginnt die Hauptarbeit. In einer inneren Schleife werden viele Sterne entlang einer Sinuskurve mittels POLYLINE gezeichnet. Die Position der Sterne wird durch den OFFSET-Zusatz berechnet. Dabei gibt s% den Abstand der Sterne voneinander an. Dieser Abstand wird in der äußeren Schleife ständig erhöht. Den dadurch entstehenden Effekt haben Sie direkt vor Augen.

Unter GFA-BASIC gibt es jedoch noch zwei weitere Befehle, die Linien auf den Bildschirm zaubern können:

LINE x1,y1,x2,y2 (GFA-BASIC-Befehl)

Setzen (Löschen etc.) einer Linie

x1: x-Koordinate Startpunkt
 y1: y-Koordinate Startpunkt
 x2: x-Koordinate Endpunkt
 y2: y-Koordinate Endpunkt

DRAW (x1,y1) (TO x2,y2 (TO ... TO ...)) (GFA-BASIC-Befehl)

Setzen (Löschen etc.) eines Punktes, einer Linie, eines Linienzuges

x1: x-Koordinate Startpunkt
 y1: y-Koordinate Startpunkt
 x2: x-Koordinate Punkt 2
 y2: y-Koordinate Punkt 2

Der erste Befehl zieht eine Linie von Punkt P1(x1,y1) nach Punkt P2(x2,y2) und bedarf eigentlich keiner weiteren Erläuterungen. Ein kleines Beispielprogramm soll die Verwendung von Linien demonstrieren:

```

' *****
' **
' ** Linienzeichner **
' **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
nx=0 !Zähler für Liniendrehung
ny=1.5
'
lang=20 !Linienlänge
xo=0 !Koordinaten-Offsets
yo=0
gm=1 !Start-Grafikmodus
'
' Rahmen zeichnen:
neu:
CLS !Bildschirm löschen
LINE 0,0,0,yauf1%
LINE 0,yauf1%,xauf1%,yauf1%

```

```
LINE xauf1%,yauf1%,xauf1%,0
LINE xauf1%,0,0,0
,
,
' Linien Zeichnen:
,
' Hauptschleife:
schleife:
GRAPHMODE gm          !Grafikmodus setzen
xo=INT(nx)*lang        !Koordinaten-Offsets berechnen
yo=INT(ny)*lang
,
' Innere Schleife:
REPEAT
  IF MOUSEK<>0          !nur zeichnen, Mausknopf gedrückt
    x=MOUSEX            !Mausposition holen
    y=MOUSEY
    LINE x-xo,y-yo,x+xo,y+yo !Linie zeichnen
  ENDIF
  c$=INKEY$            !evtl. Taste holen
UNTIL c$<>" "          !bis Taste betätigt
,
,
' Tasten bearbeiten:
IF VAL(c$)<>0            !Zifferntaste?
  gm=VAL(c$)           !Grafikmodus setzen
ENDIF
,
IF c$="*"              !Sternchen? (Linienwinkel)
  nx=nx+0.5            !Multiplikand für
  ny=ny+0.5            !Koordinaten-Offsets
  IF nx=2
    nx=0
  ENDIF
  IF ny=2
    ny=0
  ENDIF
ENDIF
,
IF c$="+"              !Pluszeichen?
  lang=lang+1          !Linienlänge erhöhen
ENDIF
,
IF c$="-"              !Minuszeichen?
  lang=lang-1          !Linienlänge verringern
ENDIF
,
IF c$="("              !Klammer auf?
  GOTO neu             !Bildschirm löschen
ENDIF
,
IF c$=" "              !Leerzeichen?
  EDIT                 !Ende!
```



```
ENDIF
```

```
GOTO schleife !weiter
```

In diesem Programm können Sie mit Ihrer Maus Linien auf den Bildschirm zeichnen. Sobald Sie auf einen Mausknopf drücken, erscheint an der momentanen Mausposition eine Linie, die Sie mit folgenden Tasten des Ziffernblockes variieren können:

Ziffer 1:	Grafikmodus auf Replace
Ziffer 2:	Grafikmodus auf Transparent
Ziffer 3:	Grafikmodus auf XOR
Ziffer 4:	Grafikmodus auf Invers Transparent
Taste *:	Linienwinkel ändern
Taste +:	Linienlänge erhöhen
Taste -:	Linienlänge verringern
Taste (:	Bildschirm löschen
Space:	Programm beenden

Sie können somit Linien aus der Hand malen oder einzelne Linien setzen. Das Programm ist gut dokumentiert und sollte nicht allzu schwer zu verstehen sein. Die MOUSE-Befehle sollten Sie nicht irritieren, wir werden später darauf zurückkommen. Hier holen Sie nur die aktuelle Mausposition (MOUSEX, MOUSEY) und den Knopf-Zustand (MOUSEK, =0: kein Knopf gedrückt).

Der zweite Befehl kann im Gegensatz zu LINE auf verschiedene Arten und Weisen verwendet werden. Zum einen gibt es ihn in der Version

```
DRAW x,y
```

in der er genau dasselbe bewirkt wie ein

```
PLOT x,y
```

Er zeichnet also einen Punkt. Wichtig zu wissen ist, daß dieser Punkt (sowohl bei DRAW als auch bei PLOT) Interpreter-intern mit Hilfe der VDI-Funktion POLYLINE gezeichnet wird, indem eine Linie mit zwei identischen Endpunkten gezogen wird. Dies ist deshalb erwähnenswert, da alle späteren Linien-Einstellungen wie Form oder Farbe auch auf die Punkte zutreffen, die mit diesen beiden Befehlen gezeichnet werden. In der Variation

```
DRAW TO x2,y2
```

zieht der Befehl eine Linie vom zuletzt mit DRAW, LINE oder PLOT gezeichneten Punkt zum Punkt P2(x2,y2). Dies ist - wie wir im folgenden Beispiel sehen werden - recht nützlich.

```

' *****
' **
' ** Funktionsdesigner **
' **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
' Funktion definieren:
DEFN f(x,s)=yauf1%/4*SIN(LOG(x)*5)+yauf1%/3+s*5
'
' Zeichnen:
FOR s=41 TO 1 STEP -3      !Schrittweite
  PLOT 1,FN f(1,s)         !Startpunkt für x=1 berechnen
  FOR x=2 TO xauf1% STEP s
    DRAW TO x,FN f(x,s)    !Linie vom letzten Punkt zeichnen
  NEXT x
NEXT s
c=INP(2)
EDIT

```

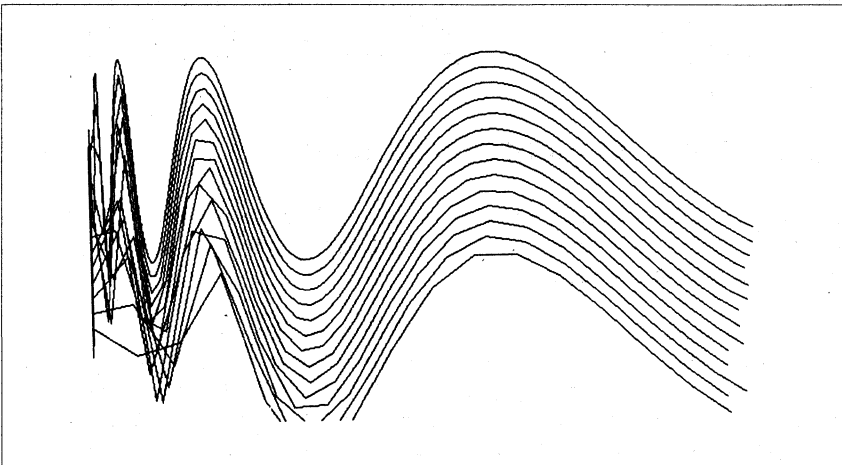


Abb. 15.11: Funktionsdesigner

Gezeichnet wird - unter Berücksichtigung der Bildschirmauflösung - eine Reihe ganz normaler Sinus-Kurven. Dabei wird stets ein neu berechneter Punkt mit dem vorher berechneten durch eine Linie verbunden (DRAW TO ...). Damit Sie sehen, daß tatsächlich eine Linie gezeichnet wird, werden zunächst nur wenige Punkte der Kurve gezeichnet, die dadurch ein sehr zackiges Aussehen erhält. Das ändert sich von Kurve zu Kurve, bis eine sehr schöne, gleichmäßige Kurve entsteht.

An diesem Beispiel soll Ihnen klar werden, wie viele scheinbar runde Grafik-Objekte nur eine Annäherung durch Linien darstellen. Das ist ein Hauptprinzip der Computergrafik. Es erinnert mit Recht an die Quadratur des Kreises bzw. die Annäherung (Approximation) eines Kreises mit Hilfe immer mehr-eckiger Polygone.

Der DRAW-Befehl kann aber noch mehr. Mit ihm können Sie ganze Linienzüge programmieren. In der Form

```
DRAW x1,y1 TO x2,y2
```

verrichtet er exakt dasselbe wie der Befehl

```
LINE x1,y1,x2,y2
```

Sie können aber nun noch so viele "TO xn,yn" hinten anhängen wie Sie wollen, und schon bekommt er wieder Ähnlichkeit mit POLYLINE. Damit hat sich hier unser Kreis geschlossen, und wir erwarten neue Abenteuer.

15.3.2 Linie ist nicht gleich Linie

Tja, einfache Linien kennen wir - wenn auch nicht in der rasenden Zeichengeschwindigkeit - von anderen, sehr viel preisgünstigeren Rechnern. Wir fordern mehr - wir bekommen mehr.

Das VDI unseres Atari ST bietet uns nämlich noch einige sogenannte Attribut-Funktionen. Dies sind Funktionen, die Eigenschaften von zu zeichnenden Linien verändern können (ähnlich wie die POLYMARKER-Eigenschaften). Diese Linien-Attribut-Funktionen gelten für alle sogenannten Linien-Objekte, die uns auch später noch über den Weg laufen werden (Linien, Rechtecke, Kreise, Ellipsen etc.). Sie gelten jedoch nicht - das sei hier schon einmal gesagt - für die Flächen-Funktionen, die zwar auch umrahmte Flächen zeichnen können, die aber nicht unter die Linien-Funktionen fallen. Hierzu gehören die im nächsten Kapitel zu besprechende Polygon-Füll-Funktion, gefüllte Rechtecke, Kreise,

Ellipsen etc. Für diese Funktionen gelten spezielle Füll-Attribute, die wir ebenfalls noch kennenlernen werden.

Doch bleiben wir auf dem Teppich und schauen uns erst einmal die verschiedenen Attribute (Eigenschaften) an, die wir an Linienobjekten ändern können. Dazu dienen die VDI-Funktionen

```
SET POLYLINE LINE TYPE (15 VDI)
SET USER-DEFINED LINE STYLE PATTERN (113 VDI)
SET POLYLINE LINE WIDTH (16 VDI)
SET POLYLINE END STYLES (108 VDI)
SET POLYLINE COLOR INDEX (17 VDI)
```

Das sind auf den ersten Blick natürlich erschlagend viele Möglichkeiten, die uns offenstehen. Doch keine Angst, wir werden auch damit noch fertig. Gehen wir doch alles einmal der Reihe nach durch. Die Farbfunktionen werden wir in einem gesonderten Kapitel behandeln.

Zunächst einmal läßt uns VDI nicht mit einfach durchgezogenen Linien alleine. Wir können vielmehr bestimmen, ob unsere Linie punktiert, gestrichelt oder sonst was ist. Dies legen wir mit der Funktion SET POLYLINE LINE TYPE fest. Um diese Funktion zu verstehen, müssen wir zunächst erläutern, wie unser Rechner eine Linie zeichnet.

Grundsätzlich beginnt er bei dem von uns angegebenen Startpunkt der Linie und setzt nach und nach jeden einzelnen Punkt der Linie. Dabei werden jeweils 16 hintereinanderfolgende Punkte zunächst über eine sogenannte Maske ausmaskiert. Diese Maske besteht aus 16 Nullen oder Einsen, die angeben, ob ein Punkt der Linie gesetzt (1) oder nicht gesetzt (0) werden soll, hier spielt natürlich wieder der Zeichenmodus eine Rolle, s.o.

Das Aussehen dieser Maske legen wir nun mit der Funktion SET POLYLINE TYPE fest. Dabei stehen uns beim Atari-Bildschirm bereits sechs fertige Masken (Linientypen) zur Verfügung:

Maske 1:	*****	durchgezogen
Maske 2:	*****---	unterbrochen
Maske 3:	***-----***	kurzgestrichelt
Maske 4:	*****---***	lang-kurz-gestrichelt
Maske 5:	*****-----	lang-gestrichelt
Maske 6:	****-**-**---	lang-kurz-kurz-gestrichelt

Viele solcher Masken hintereinander ergeben eine Linie. Das oberste Bit einer Maske ist dann der erste Punkt, der gezeichnet wird.

Neben diesen fertigen Masken oder Linienformen können wir aber noch unsere eigene Maske kreieren. Dies geht mit Hilfe der Funktion mit dem unmöglichen Namen SET USER DEFINED LINE STYLE PATTERN. Wählen wir in der Funktion SET POLYLINE TYPE den Typ Nummer 7, so findet unsere eigene Maskendefinition Anwendung. Wir müssen eben genannter Funktion nur einen Wert (pattern) übergeben, der sich aus dem Bit-Muster für unsere Linie ergibt. Denkbar wäre beispielsweise:

```
****--****--****--  
= %1100110011001100  
= $CCCC  
= 52428 (dez) oder -13108
```

In diesem Fall müssten Sie der VDI-Funktion 113 also den Wert 52428 oder hexadezimal \$CCCC als pattern übergeben.

Können wir auf diese Linientypen verzichten, so haben wir die Möglichkeit, unsere Linien zu verbreitern. Dies geschieht durch die Funktion SET POLYLINE LINE WIDTH. Hier geben wir die Breite einer Linie eines Linien-Objektes in Pixel an. Breite 1 wäre demnach eine ganz normale Linie, die auch wieder punktiert oder gestrichelt sein könnte. Effekt haben nur ungerade Werte, da eine Linie jeweils rechts und links von der Linienachse um ein Pixel erweitert wird. Der nächstkleinere Wert wäre dann also 3 usw. Auf dem Atari-Bildschirm ist die maximale Liniendicke bei "width=39" erreicht. Größere Werte sind nur bei anderen Geräten denkbar.

Einen interessanten Zusatz stellen die Linien-End-Typen dar, die mit der VDI-Funktion Nummer 108 (SET POLYLINE END STYLES) variiert werden können. Diese Funktion stellt uns die folgenden drei Anfangs- oder Endformen zur Verfügung:

- eckig (normal)
- Pfeil
- rund

Wie das aussieht, werden Sie noch im nächsten Programm begutachten können. Wählen Sie z.B. "Pfeil", so erscheint tatsächlich ein kleiner oder großer (bei großer Liniendicke) Pfeil am Linienende und/oder -anfang. Bei "rund" sind die jeweiligen Enden tatsächlich abgerundet.

Wo Anfang und Ende liegen, hängt von der jeweiligen Linienfigur ab. Bei einfachen Linien sind es natürlich die beiden Endpunkte, bei mit einem Befehl

gezeichneten Linienzügen hingegen sind es jeweils Start- und Endpunkte des gesamten Linienzuges, also nicht der einzelnen Linien selber! Doch sehen Sie selbst:

```
' *****
' **                **
' **  Linienattribute  **
' **                **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
' Attributwerte initialisieren:
st=1                !Linienstil
di=1                !Liniendicke
fu=0                !Linienfuß
sp=0                !Linienspitze
'
' Hauptschleife:
schleife:
CLS                  !Bildschirm löschen
DECLINE st,di,fu,sp  !Attribute realisieren
'
' Linie zeichnen:
x1%=xauf1%*0.1
y1%=yauf1%*0.1
x2%=xauf1%*0.9
y2%=yauf1%*0.9
x3%=xauf1%*0.8
y3%=yauf1%*0.1
'
DRAW x1%,y1% TO x2%,y2% TO x3%,y3%
'
' Statuszeile schreiben:
PRINT
PRINT AT(1,23);"Stil: ";st;TAB(20);"Dicke: ";di
PRINT AT(1,24);"Fuß:  ";fu;TAB(20);"Spitze: ";sp
'
' Auf Taste warten:
REPEAT
  c$=INKEY$
UNTIL c$<>" "
'
' eventuelle restliche Tasten löschen:
REPEAT
  UNTIL INKEY$=""
'
' Tasten bearbeiten:
IF c$="("            !Klammer auf?
  st=st+1            !Stilform erhöhen
```

```

ENDIF
IF c$=")"
    st=st-1
ENDIF
'
IF c$="/"
    st=st+200
ENDIF
IF c$="*"
    st=st-200
ENDIF
'
IF c$="+"
    di=di+1
ENDIF
IF c$="-"
    di=di-1
ENDIF
'
c=VAL(c$)
IF c<>0 AND c<7
    IF c<4
        fu=c-1
    ELSE
        sp=c-4
    ENDIF
ENDIF
'
IF c$=" "
    EDIT
ENDIF
'
GOTO schleife

```

Mit diesem Programm können Sie direkt am Bildschirm und per Tastendruck die verschiedenen Linienparameter verstellen. Sie sehen das Ergebnis stets aktuell auf dem Bildschirm, egal, ob Sie dicke oder dünne, durchgezogene oder gestrichelte Linien erzeugen. Dieses Programm hilft Ihnen, die vielen verschiedenen Möglichkeiten der Linien-Attribute zu durchschauen.

Bevor wir uns dem Programm zuwenden, müssen wir kurz darauf eingehen, auf welche Weise die Linienattribute aus BASIC heraus gehandhabt werden. Es sind dafür nur zwei Befehle zuständig, die alle (oder einzelne) Linien-Eigenschaften einstellen:

DEFLINE st,di,ls,le (GFA-BASIC-Befehl)

Einstellen der Linienattribute

st: Linienstil:
=1-6: vordefinierte Linienstile
<0: eigendefinierter Linienstil
di: Liniendicke
ls: Linienstarttyp
le: Linienendtyp

COLOR fi (GFA-BASIC-Befehl)

Einstellen des Farbindexes für Linien-Objekte

fi: Farbindex

Wie Sie sehen, können Sie hier beim ersten Befehl alle möglichen Linientypen einstellen. Wollen Sie die vordefinierten Linienstile benutzen, so geben Sie für st einen Wert von 1 bis 6 an. Wollen Sie dagegen eine eigene Linienmaske erstellen, so errechnen Sie den Maskenwert (s.o.) und kehren einfach das Vorzeichen auf negativ.

Im Programm wird ein Linienzug aus zwei Linien auf dem Bildschirm dargestellt. Mit Hilfe der folgenden Tasten des Ziffernblockes können Sie die verschiedenen Attribute einstellen, die sowohl als Zahlenwerte in den untersten zwei Bildschirmzeilen auftauchen als auch direkt in ihrem Effekt auf dem Bildschirm sichtbar werden:

(-	Linientyp erhöhen
) -	Linientyp verringern
/ -	Linientyp um 200 erhöhen
* -	Linientyp um 200 verringern
+ -	Liniendicke erhöhen
--	Liniendicke verringern
1 -	Linienstart auf eckig
2 -	Linienstart auf Pfeil
3 -	Linienstart auf rund
4 -	Linienende auf eckig
5 -	Linienende auf Pfeil
6 -	Linienende auf rund
Space -	Programmende

Spielen Sie ein bißchen mit diesem Programm herum, und Sie bekommen einen umfassenden Überblick über die mannigfaltigen Möglichkeiten.

15.3.3 Das, was drin ist - Flächen ausfüllen

Wir kommen nun zu einem höchstinteressanten Kapitel, das uns wieder eine ganze Schublade voll Möglichkeiten beschert. Haben wir uns bisher mit einfachen Linien, also mehr oder weniger einfacher Grafikdarstellung zufrieden gegeben, so kommen wir nun zu den Flächen, die uns eine ganz neue Dimension erschließen. Allgemein werden wir feststellen, daß es zu jeder Linien-Funktion, also jeder Funktion, die ein Linien-Objekt zeichnet, eine entsprechende Flächen-Funktion gibt, die also das gleiche Linien-Objekt nur mit ausgefüllter Fläche zeichnet. Wir haben bereits erwähnt, daß zu diesen Flächen-Funktionen spezielle Attribut-Funktionen existieren, ähnlich wie es - uns inzwischen bekannte - Attribut-Funktionen für die Linien-Funktionen gibt. Der Vorteil dieses Systems liegt darin, daß wir durch die Kombination der Linien mit den entsprechenden Flächen-Funktionen alle Linien-Attribute mit den Flächen-Attributen kombinieren können. So kann die Fläche z.B. eine andere Farbe besitzen als die Umrandung dieser Fläche, die extra mit der entsprechenden Linien-Funktion gestaltet wurde, aber dazu später.

Die korrespondierende Flächen-Funktion zu POLYLINE heißt FILLED AREA (9 VDI)

Wir geben hier - genauso wie bei POLYLINE - die Anzahl der zu verbindenden Punkte sowie die Koordinaten der einzelnen Punkte an. Der Aufruf der Funktion bringt dann das gewünschte Ergebnis auf den Bildschirm bzw. auf das angesprochene Gerät.

Doch beim einfachen Ausfüllen der Flächen bleibt es nicht. Mit den folgenden Funktionen werden Flächen-Attribute manipuliert:

```
SET FILL STYLE INDEX (24 VDI)
SET FILL INTERIOR STYLE (23 VDI)
SET FILL PERIMETER VISIBILITY (104 VDI)
SET USER DEFINED FILL PATTERN (112 VDI)
SET FILL COLOR INDEX (25 VDI)
```

Damit haben wir eine Übersicht über alle möglichen Flächen-Attribute, die wir im folgenden kurz besprechen wollen, um dann gleich mit einem Programm-Beispiel in die Praxis zu gehen.

Die erste VDI-Funktion, die uns interessiert, heißt **FILL INTERIOR STYLE**. Sie bietet uns die Möglichkeit, auszuwählen, wie unsere Fläche ausgefüllt werden soll:

- mit Hintergrundfarbe
- mit Füllfarbe
- punktiert
- schraffiert
- selbstdefiniert

Bevor wir die einzelnen Möglichkeiten durchgehen, müssen wir hier erst den Begriff der Maske klären. Er ist uns bereits öfter (z.B. beim Linienstil) über den Weg gelaufen und hat auch hier eine analoge Funktion. Beim Füllen von Flächen wird stets mit einer 16x16 Punkte großen Maske gearbeitet. Diese Maske gibt - wie beim Linienstil - an, welche Punkte als gesetzt und welche als nicht gesetzt gelten. Wird nun eine Fläche gezeichnet, so wird praktisch über die gesamte Fläche Maske an Maske gesetzt. Dadurch ist es möglich, ein bestimmtes Muster innerhalb der Fläche zu erzeugen.

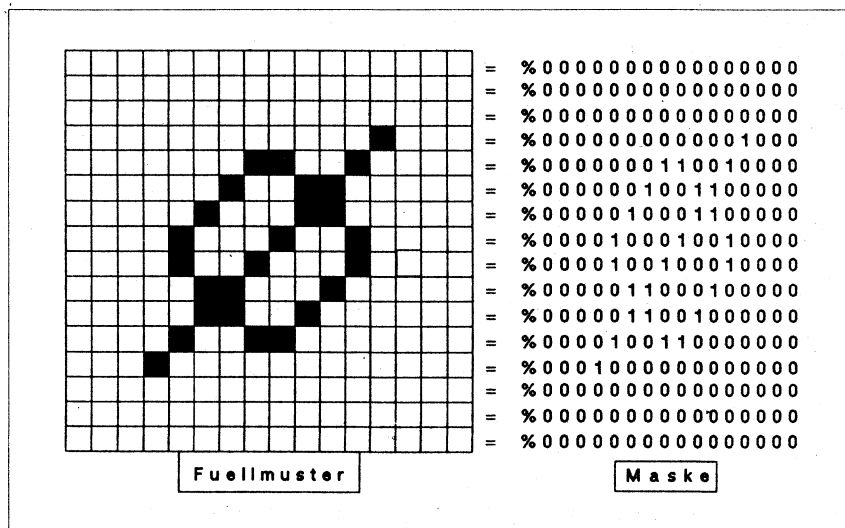


Abb. 15.12: 16x16-Punktemaske (Flächen)

Eine Fläche besteht unter VDI aus zwei unabhängigen Teilen:

- Flächenumrandung
- Flächeninhalt

Die Umrandung gilt nicht als Linien-Objekt, sondern ist - unabhängig von den Linien-Attributen - stets durchgezogen. Sie nimmt dabei die Füllfarbe an.

Schauen wir uns die einzelnen Möglichkeiten der Füllstilgestaltung einmal an: Im ersten Fall wird praktisch alles, was unter unserer Fläche liegt, gelöscht - sofern wir uns im Replace-Modus befinden - da sie mit Hintergrundfarbe gefüllt wird. Es wird mit einer "leeren" Maske gezeichnet, mit einer Maske also, in der alle Bits gelöscht sind. Übrig bleibt meist nur noch der Rahmen der Fläche (obwohl dieser ebenfalls noch ausgeschaltet werden kann).

Die zweite Füllart füllt eine Fläche mit der normalen Füllfarbe ganzflächig und einheitlich aus. Es wird in diesem Fall mit einer "vollen" Maske, einer Maske also, in der alle Bits gesetzt sind, gezeichnet. Der Rahmen der Fläche geht dabei logischerweise unter, obwohl er auch gezeichnet wird.

Bei der dritten Füllweise wird es interessanter. Hier steht einfach "punktiert". Hinter dieser Einstellung verbergen sich jedoch insgesamt 24 verschiedene fest eingestellte Füllmuster, die im weitesten Sinne mit Punktierung zu tun haben. Welches dieser 24 Füllmuster nun tatsächlich verwendet wird, um eine Fläche auszufüllen, entscheidet die Funktion SET FILL STYLE INDEX. Hier geben Sie einfach die Nummer des gewünschten Füllmusters an. Die alleinige Einstellung in der VDI-Funktion 23 also reicht hier noch nicht aus.

Die vierte Füllweise ähnelt der dritten. Auch hier stehen (diesmal 12) verschiedene zusätzliche Füllmuster zur Verfügung, die hier unter "Schraffur" eingeordnet werden und ebenfalls mit der VDI-Funktion 24 (SET FILL STYLE INDEX) angewählt werden müssen. Die Unterscheidung zwischen punktierten und schraffierten Mustern liegt daran, daß bestimmte Geräte (z.B. Plotter o.ä.) hier verschiedene Füll-Algorithmen verwenden müssen.

Die fünfte und letzte Einstellung der VDI-Funktion 23 erweitert die Füllmöglichkeiten enorm. Wenn Sie diese Einstellung vornehmen, dann können Sie mit Hilfe der Funktion SET USER DEFINED FILL PATTERN eine eigene, selbst erstellte Füll-Maske erstellen. Sie kreieren also Ihr eigenes Füllmuster. Das bietet natürlich enorme Möglichkeiten. Das besondere hieran ist die Möglichkeit, sogar mehrfarbige Füllmuster erstellen zu können - je nach Auflösung.

Um ein solches eigenes Muster zu erstellen, übergeben Sie der genannten Funktion SET USER DEFINED FILL PATTERN zunächst einmal die Anzahl der sogenannten Farbebenen (für C-Anwender wurde diese Vereinfachung eingeführt) bzw. die Anzahl der Farbebenen mal 16 (für Assembler- oder BA-

SIC-Anwender) in "contrl(3)" (sintin). Unter Farbebenen versteht man hier genau dasselbe wie bei der Farborganisation des Bildschirmspeichers. Beim Bildschirm-RAM liegen bekanntlich die zusammengehörigen Wörter der verschiedenen Bildschirmenebenen nebeneinander. Hier jedoch sind die Ebenen jeweils getrennt zusammengefaßt. In NGR bedeutet das nun: Die Nummer des Farbindex eines Maskenpunktes wird aus je einem korrespondierenden Bit der vier Ebenen zusammengesetzt. Jeweils die Bits 0 aus Wort 0 der vier Maskenebenen stellen die 4-Bit-Nummer des Farbregisters für den ersten Maskenpunkt usw.

Arbeiten wir in HGR oder wollen nur einfarbige Farbmuster verwenden, dann genügt eine Musterebene.

Eine Ebene setzt sich bekanntlich aus 16x16 Punkten bzw. Bits zusammen, umfaßt also genau 16 Wörter. (Daraus resultiert auch der Wert, der in contrl(3) übergeben werden muß.) Wie solche eigenen Füllmuster erstellt und programmiert werden, zeigen Ihnen die weiter unten folgenden Beispielprogramme.

Es bleibt neben der Füllfarb-Funktion noch ein weiterer Attribut-Befehl zu beschreiben: SET FILL PERIMETER VISIBILITY. Mit ihm kann die oben genannte Umrahmung von Flächen ein- oder ausgeschaltet werden. Bekanntlich wird diese Umrahmung unabhängig von den Linien-Attributen stets durchgezogen und mit der Breite von einem Pixel gezeichnet.

Eine mögliche Anwendung dieser Funktion könnte darin bestehen, eine Fläche auf dem Bildschirm mit einer Umrahmung zu bringen, die die Linien-Attribute berücksichtigt. Dazu wird das gewünschte Flächenobjekt ohne Rahmen auf den Bildschirm gebracht und anschließend die entsprechende Linienfunktion (gleiche Übergabeparameter!) aufgerufen, die die Fläche nun nach allen Richtlinien der Linien-Attribute umrahmt.

Das folgende BASIC-Programm erlaubt Ihnen, alle möglichen Füllmuster-Attribute direkt am Bildschirm zu testen, ähnlich wie wir das bereits für die Linienattribute ausprobiert haben. Vorher aber wollen wir kurz die GFA-BASIC-Befehle durchgehen, die die obigen VDI-Funktionen dem BASIC-Programmierer zur Verfügung stellen. Die entsprechende Flächenfunktion zu POLYLINE lautet:

POLYFILL count,x(),y() (OFFSET ox,oy) (GFA-BASIC-Befehl)

Ausfüllen eines Polygons

count: Anzahl der Eck-Punkte

x(): Array mit allen x-Koordinaten

y(): Array mit allen y-Koordinaten

ox: x-Verschiebewert

oy: y-Verschiebewert

DEFFILL (fa),(st),(mu) (GFA-BASIC-Befehl)

DEFFILL (fa),mu\$

Einstellen der Füllmusterattribute

fa: Füllmusterfarbe

st: Füllmusterstil:

=0: Fläche wird mit Hintergrundfarbe gefüllt

=1: Fläche wird mit Füllfarbe gefüllt

=2: Fläche wird mit Punktmuster gefüllt

=3: Fläche wird mit Schraffiermuster gefüllt

=4: Fläche wird mit selbstdefiniertem Füllmuster gefüllt

mu: Füllmuster für Punktierung/Schraffur

mu\$: Füllmusterdefinition als String

Der erste Befehl funktioniert haargenau, wie Sie das bereits von POLY-MARK und POLYLINE gewohnt sind. Der zweite Befehl sollte Ihnen auch klar sein, wenn Sie die obigen Ausführungen zu den entsprechenden VDI-Funktionen gelesen haben. Der Befehl hat allerdings eine kleine Besonderheit: Sie können ihn in zwei verschiedenen Variationen ansprechen. Die erste stellt die Farben und die Füllmusterattribute ein. Die zweite übergibt dem GFA-BASIC ein neues Füllmuster (allerdings nur einfarbig), das in einer String-Variable übermittelt wird. Darin werden insgesamt 32 ASCII-Werte gespeichert, die jeweils den aus einem Bit-Muster resultierenden Byte-Wert darstellen. Wie das in der Praxis zu verstehen ist, entnehmen Sie bitte dem zweiten Programm.

Hier jedoch zunächst einmal das angekündigte Programm zur Verstellung aller Füllmusterparameter:

```
' *****
' **
' ** Füllmuster-Männchen **
' **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
DIM x%(30),y%(30) !Koordinatenfelder dimensionieren
'
' Laufende Werte initialisieren:
xgroesse%=18 !Größe in x-Richtung
ygroesse%=18 !Größe in y-Richtung
xpos%=MOUSEX !x-Position
ypos%=MOUSEY !y-Position
'
' Füllmusterattribute:
farbind%=1 !Farbindex
mustil%=1 !Füllmusterstil
muwahl%=1 !Füllmusterwahl
'
CLS !Bildschirm löschen
'
' Eckpunktkoordinaten einlesen:
start:
RESTORE
READ anzahl% !Eck-Punkteanzahl
FOR i%=1 TO anzahl%
    READ x%(i%-1),y%(i%-1) !einlesen
    x%(i%-1)=x%(i%-1)*xgroesse% !Vergrößern
    y%(i%-1)=0-y%(i%-1)*ygroesse%
NEXT i%
'
' Statuszeilen löschen:
FOR i%=24 TO 23 STEP -1
    FOR k%=1 TO 40
        PRINT AT(k%,i%);" ";
    NEXT k%
NEXT i%
'
' Statuszeilen schreiben:
PRINT
PRINT AT(1,23);"Farbe: ";farbind%;TAB(12);"Musterstil:
";mustil%;TAB(26);"Musterwahl: ";muwahl%
PRINT AT(1,24);"X-Größe: ";xgroesse%;TAB(12);"Y-Größe: ";ygroesse%;
'
' Füllmusterattribute einstellen:
DEFFILL farbind%,mustil%,muwahl%
```

```

'
' Zeichenprogramm:
GRAPHMODE 3 !XOR-Modus einstellen
' Objekt einmal zeichnen:
POLYFILL anzahl%,x%(),y%() OFFSET xpos%,ypos%
'
' Hauptschleife:
REPEAT
  x%=MOUSEX !aktuelle Maus-Koordinaten
  y%=MOUSEY
  IF xpos%<>x% OR ypos%<>y% !geändert?
    ' Altes Objekt durch nochmaliges überzeichnen löschen:
    POLYFILL anzahl%,x%(),y%() OFFSET xpos%,ypos%
    ' neue Koordinaten übernehmen:
    xpos%=x%
    ypos%=y%
    ' Neues Objekt an die neue Stelle zeichnen:
    POLYFILL anzahl%,x%(),y%() OFFSET xpos%,ypos%
  ENDIF
'
IF MOUSEK<>0 !Mausknopf gedrückt?
  GRAPHMODE 1 !Replace-Modus ein
  ' Objekt an die Stelle fest zeichnen:
  POLYFILL anzahl%,x%(),y%() OFFSET xpos%,ypos%
  '
  GRAPHMODE 3 !XOR-Modus ein
  ' Objekt einmal im XOR-Modus zeichnen:
  POLYFILL anzahl%,x%(),y%() OFFSET xpos%,ypos%
ENDIF
'
c$=INKEY$ !evtl. Taste übernehmen
UNTIL c$<>" !bis Taste gedrückt
'
' Alle anderen Tasten im Tastatur-Buffer löschen:
REPEAT
UNTIL INKEY$=""
'
' Objekt durch nochmaliges Zeichnen löschen:
POLYFILL anzahl%,x%(),y%() OFFSET xpos%,ypos%
'
' Tastenbearbeitung:
'
cscan%=ASC(RIGHT$(c$)) !Scan-Code berechnen
'
IF cscan%=&H4B !Cursor links?
  xgroesse%=xgroesse%-1 !x-Größe verringern
ENDIF
IF cscan%=&H4D !Cursor rechts?
  xgroesse%=xgroesse%+1 !x-Größe erhöhen
ENDIF
'

```

```
IF cscan%=&H50                !Cursor ab?
    ygroesse%=ygroesse%-1      !y-Größe verringern
ENDIF
IF cscan%=&H48                !Cursor auf?
    ygroesse%=ygroesse%+1      !y-Größe erhöhen
ENDIF
'
IF c$="+"                    !Pluszeichen?
    muwahl%=muwahl%+1          !Musterwahlnummer erhöhen
ENDIF
IF c$="-"                    !Minuszeichen?
    muwahl%=muwahl%-1          !Musterwahlnummer verringern
ENDIF
'
IF c$="("                    !Klammer auf?
    farbind%=farbind%-1        !Farbindex verringern
ENDIF
IF c$=")"                    !Klammer zu?
    farbind%=farbind%+1        !Farbindex erhöhen
ENDIF
'
IF c$="/"                    !Geteilt-Zeichen?
    mustil%=mustil%-1          !Musterstilnummer verringern
ENDIF
IF c$="*"                    !Mal-Zeichen?
    mustil%=mustil%+1          !Musterstilnummer erhöhen
ENDIF
'
IF c$=" "                    !Leertaste?
    EDIT                        !Ende
ENDIF
'
GOTO start
'
' Eckpunktkoordinaten der Figur:
' Anzahl der Punkte:
DATA 13
' Eckpunktkoordinaten:
DATA 0,2, 1,6, 5,8, 3,9, 1,8
DATA 5,6, 6,2, 5,4, 4,0, 3,3
DATA 2,0, 1,4, 0,2
```

Lassen Sie uns das Programm kurz durchgehen. Zunächst werden einige Werte initialisiert, die die verstellbaren Parameter vorbestimmen. Jetzt liest das Programm die Definition des mit POLYFILL zu zeichnenden Objektes aus den DATA-Zeilen am Programmende mit der entsprechenden Vergrößerung in die Koordinaten-Arrays. Wir kennen dieses Verfahren bereits aus anderen Programmen zu POLYLINE und sparen uns hier eine nähere Erläuterung.

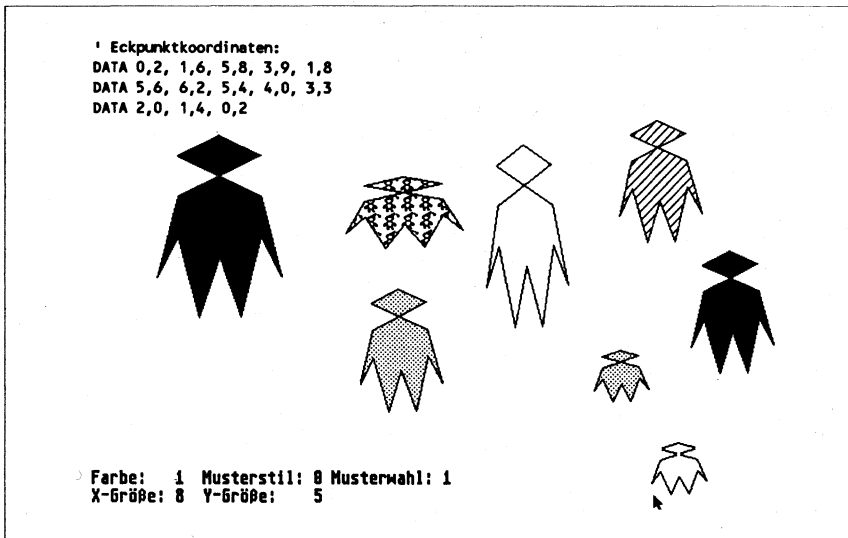


Abb. 15.13: Füllmuster-Männchen

Als nächstes werden die beiden Zeilen auf den Bildschirm gebracht, die die aktuellen Attribut-Werte enthalten. So können Sie jederzeit sehen, welche Attribute welchen Effekt bewirken.

Das Objekt ist mit der Maus auf dem Bildschirm verschiebbar. Wie bereits demonstriert, wird es im XOR-Modus stets durch zweimaliges Zeichnen wieder gelöscht. Die Bewegung und das ständige Löschen und Zeichnen findet in der Hauptschleife statt, in der die Maus abgefragt und entsprechend verzweigt wird.

Mit folgenden Tasten können Sie die verschiedenen Attribute verändern:

- | | |
|------------|---|
| Mausknopf | - Objekt auf dem Bildschirm fest zeichnen |
| Cursor re | - x-Größe des Objektes erhöhen |
| Cursor li | - x-Größe des Objektes verringern |
| Cursor auf | - y-Größe des Objektes erhöhen |
| Cursor ab | - y-Größe des Objektes verringern |
| Taste "*" | - Musterstilnummer erhöhen |
| Taste "/" | - Musterstilnummer verringern |
| Taste "+" | - Nummer des Punkt-/Schraffurmusters erhöhen |
| Taste "-" | - Nummer des Punkt-/Schraffurmusters verringern |
| Taste ")" | - Farbindex erhöhen |

Taste "(" - Farbindex verringern
Leertaste - Programmende

Und jetzt das angekündigte Programm zur Definition eines eigenen Füllmusters:

```
' *****  
' **  
' ** Tapetenwechsel **  
' **  
' *****  
'  
' Bildschirmauflösung:  
xauf1%=DPEEK(&H2686)  
yauf1%=DPEEK(&H2688)  
'  
' Musterdefinition aus DATA-Zeilen einlesen:  
FOR i%=0 TO 31  
  READ a                    !Einlesen  
  mu$=mu$+CHR$(a)        !an ASCII-String anhängen  
NEXT i%  
'  
' Neue Musterdefinition übernehmen:  
DEFFILL 1,mu$  
'  
DIM x%(3),y%(3)  
'  
' Eckkoordinaten Dreieck:  
x%(0)=10  
y%(0)=10  
x%(1)=80  
y%(1)=80  
x%(2)=150  
y%(2)=10  
'  
' Zeichnen:  
FOR gm%=1 TO 4            !in verschiedenen Grafikmodi  
  CLS                    !Bildschirm löschen  
  GRAPHMODE gm%        !Grafikmodus einstellen  
'  
  FOR x%=0 TO 470 STEP 2  
    ' Ausgefülltes Dreieck zeichnen:  
    POLYFILL 3,x%(),y%() OFFSET x%*xauf1%/640,60*SIN(x%/100)+yauf1%/2  
  NEXT x%  
  ' Warteschleife:  
  FOR t=1 TO 20000  
    NEXT t  
'  
NEXT gm%  
'  
c=INP(2)                !Auf Taste warten  
EDIT                    !Ende
```

```
' Musterdefinition (16x16 Wörter):
```

```
DATA &X00000000,&X00100000  
DATA &X00000000,&X11010000  
DATA &X00000011,&X11000110  
DATA &X00001111,&X11000110  
DATA &X00011000,&X11000000  
DATA &X00010000,&X01000000  
DATA &X00001000,&X10000000  
DATA &X00000111,&X00000000  
DATA &X00001000,&X10000000  
DATA &X00110000,&X01100000  
DATA &X01101000,&X10110000  
DATA &X01101000,&X10110000  
DATA &X00001111,&X10000000  
DATA &X00001000,&X10000000  
DATA &X00001100,&X11000000  
DATA &X00000000,&X00000000
```

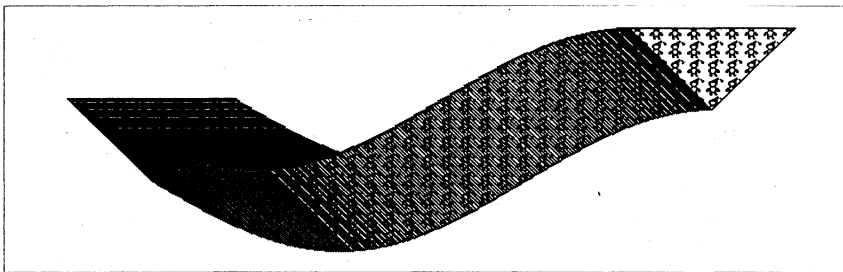


Abb. 15.14: Tapetenwechsel

Die wichtigen Teile in diesem Programm stehen zum einen am Programmende. Hier sehen Sie, wie Sie am besten Ihr Füllmuster in DATA-Zeilen organisieren. Indem Sie es wie hier Byte für Byte in binärer Schreibweise abspeichern, können Sie leicht direkt im Programm Änderungen vornehmen. Dabei haben Sie stets einen ungefähren Überblick, wie Ihr Muster später in Realität aussehen wird.

Der zweite Teil ist das Einlesen dieser DATAs. Hier sehen Sie, wie Sie langsam einen String aus 32 Byte zusammenbauen, indem Sie die DATA-Definitionen als ASCII-Werte für ein Zeichen ansehen.

Im Anschluß an diese Definition können von nun an alle Flächen mit dem neuen Muster ausgefüllt werden, wie die darauffolgenden Programmzeilen veranschaulichen.

15.4 Rechtecke im Rampenlicht

Nun, da wir uns bestens mit Linien und Flächen auskennen, wird es uns ein Leichtes sein, die Rechteckbefehle kennenzulernen. Rechtecke sind besondere Formen von Polygonen, die jedoch so oft verwendet werden, daß ein eigener Befehl sinnvoll erscheint. Wie Sie an unseren bisherigen Beispielprogrammen feststellten, mußten wir uns - da wir ja noch keine Rechteckfunktionen kennengelernt hatten - stets mit Polygondefinitionen über Wasser halten.

Rechtecke finden in vielerlei Hinsicht Anwendungen. So sind sie beispielsweise unentbehrlich bei Balkendiagrammen, die wir später noch kennenlernen werden. Aber auch sonst stellen sie wichtige Ergänzungen zu den normalen Grafikbefehlen dar.

Bevor wir uns jedoch ins Getümmel der Beispielprogramme stürzen, wollen wir an dieser Stelle die verschiedenen Befehle vorstellen, die uns das Hantieren mit dem neuen Werkzeug ermöglichen. Beginnen wir also mit den hierzu anzuführenden VDI-Befehlen:

```
FILL RECTANGLE (11/4 VDI)
BAR (11/1 VDI)
ROUNDED RECTANGLE (11/8 VDI)
FILLED ROUNDED RECTANGLE (11/9 VDI)
```

Die erste der genannten VDI-Funktionen zeichnet ein ausgefülltes Rechteck auf den Bildschirm. Dabei werden alle Flächen-Attribute berücksichtigt:

- Füllmusterfarbe
- Füllmusterstil
- Punkt-/Schraffur-Füllmusterart
- selbstdefiniertes Füllmuster

Ein einziges Flächenattribut jedoch findet keine Verwendung: Die Umrandung (perimeter). Die mit diesem Befehl gezeichnete Fläche wird nämlich niemals umrandet, es wird also lediglich die Fläche gezeichnet. Damit fällt er aus unserem Schema der Linien- und Flächenobjekte ein wenig heraus. Er ist wohl dazu gedacht, ganze Bildschirmbereiche mit einem Füllmuster (oder einer Farbe) zu unterlegen.

Der zweite Befehl (BAR) dagegen gliedert sich ohne Ausnahmen in dieses Schema ein. Hier wird ein umrahmtes (oder nicht umrahmtes, je nach perimeter-Einstellung) Rechteck unter Berücksichtigung aller Flächen-Attribute auf den Bildschirm gebracht. Interessant in diesem Zusammenhang ist die Tat-

sache, daß diese Funktion exakt dieselben Resultate bringt wie FILL RECTANGLE, sofern man die Umrandung mittels SET FILL PERIMETER VISIBILITY ausschaltet. Wir haben es hier also mit einer merkwürdigen Funktionsüberschneidung zu tun.

Für C-Programmierer wäre an dieser Stelle nichts mehr zu sagen. Wollen Sie die BAR-Funktion jedoch von Assembler oder BASIC aus nutzen, so sollten Sie folgendes noch wissen:

BAR gehört zu einer Reihe von sogenannten grafischen Grundfunktionen, die alle dieselbe Funktionsnummer (nämlich 11) besitzen. Diese Grundfunktionen, zu denen noch die in einem der nächsten Kapitel besprochenen Kreis- und Ellipsenfunktionen gehören, unterscheiden sich in dem sogenannten Secundär-Opcode. Er wird im Control-Array an Position 5 (also im fünften Wort = 10. Byte) gespeichert (contrl(5)). Wie Sie wissen, wird der eigentliche Opcode (hier wie gesagt 11) an die Position 0 gesetzt. Der Secundär-Opcode der Funktion BAR hat die Nummer 1.

Bemerkenswert und ebenfalls aus der Rolle fallend ist die Tatsache, daß es zum Zeichnen einfacher Rechtecke keine entsprechende Linienfunktion gibt. Wollen wir also ein nicht ausgefülltes Rechteck zeichnen, so sind wir wieder auf die POLYLINE-Funktion angewiesen. Es sollte aber nicht so schwierig sein, eine Funktion in C oder ein Unterprogramm in Assembler zu schreiben, die bzw. das eine solche Funktion simuliert.

Eine Besonderheit des VDI sind die beiden folgenden Funktionen, die ebenfalls mit dem Opcode 11 zu den grafischen Grundfunktionen gehören, also noch einen Secundär-Opcode benötigen. Sie zeichnen ebenfalls Rechtecke. Diese Rechtecke besitzen aber aus lauter Luxus abgerundete Ecken, was zur Dekoration von Programm-Outputs beiträgt. Die Zeichengeschwindigkeit dieser Objekte ist jedoch um einige Faktoren geringer als bei normalen Rechtecken, wie Sie an den verschiedenen Demo-Programmen feststellen werden.

Die Übergabeparameter entsprechen denen der Funktion BAR. Der einzige Unterschied: Hier gibt es wieder eine Linien- und eine Flächenfunktion - ganz nach Regel.

Ganz kurz wollen wir hier noch die Befehle vorstellen, die GFA-BASIC zu diesem Thema bietet:

BOX x1,y1,x2,y2 (GFA-BASIC-Befehl)

Zeichnen eines Rechtecks als Linienobjekt

x1: x-Koordinate einer Ecke E1
y1: y-Koordinate einer Ecke E1
x2: x-Koordinate der E1 gegenüberliegenden Ecke E2
y2: y-Koordinate der E1 gegenüberliegenden Ecke E2

PBOX x1,y1,x2,y2 (GFA-BASIC-Befehl)

Zeichnen eines ausgefüllten Rechtecks als Flächenobjekt

x1: x-Koordinate einer Ecke E1
y1: y-Koordinate einer Ecke E1
x2: x-Koordinate der E1 gegenüberliegenden Ecke E2
y2: y-Koordinate der E1 gegenüberliegenden Ecke E2

RBOX x1,y1,x2,y2 (GFA-BASIC-Befehl)

Zeichnen eines Rechtecks mit runden Ecken als Linienobjekt

x1: x-Koordinate einer Ecke E1
y1: y-Koordinate einer Ecke E1
x2: x-Koordinate der E1 gegenüberliegenden Ecke E2
y2: y-Koordinate der E1 gegenüberliegenden Ecke E2

PRBOX x1,y1,x2,y2 (GFA-BASIC-Befehl)

Zeichnen eines ausgefüllten Rechtecks mit runden Ecken als Flächenobjekt

x1: x-Koordinate einer Ecke E1
y1: y-Koordinate einer Ecke E1
x2: x-Koordinate der E1 gegenüberliegenden Ecke E2
y2: y-Koordinate der E1 gegenüberliegenden Ecke E2

Zu diesen Befehlen ist wohl nichts zu sagen, da sie nicht viel anders funktionieren als die VDI-Befehle.

Rechtecke können Sie sich sicher alle vorstellen, aus diesem Grunde fangen wir gleich mit unserem ersten Beispielprogramm an. Hier ist es:

```

/ *****
/ **           **
/ ** Rechtecke **
/ **           **
/ *****
/
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
/
i%=1
REPEAT
  DEFFILL 1,2,i%      !Füllmusterwahl
  MOUSE x%,y%,k%      !Mauspos. und Knöpfe abfragen
  /
  IF k%=0              !Kein Knopf gedrückt
    GRAPHMODE 3        !=> XOR-Modus
  ELSE
    IF k%=1             !linker Knopf gedrückt
      GRAPHMODE 1      !=> Replace-Modus
    ELSE
      i%=i%+1          !rechter Knopf => Füllmuster ändern
      IF i%>24
        i%=1
      ENDIF
      PAUSE 10          !"Antiprell-Pause"
    ENDIF
  ENDIF
ENDIF

' Gefülltes Rechteck zeichnen:
PBOX x%,y%,x%+xauf1%*0.1,y%+yauf1%*0.1
' PRBOX x%,y%,x%+xauf1%*0.1,y%+yauf1%*0.1
/
' Bei XOR-Modus wieder löschen:
PBOX x%,y%,x%+xauf1%*0.1,y%+yauf1%*0.1
' PRBOX x%,y%,x%+xauf1%*0.1,y%+yauf1%*0.1
/
UNTIL INKEY$<>"      !Ende bei Taste
EDIT

```

Auch an diesem Demonstrationsprogramm können Sie direkt aktiv teilhaben. Nach dem Start wird nämlich an die aktuelle Mausposition ein gefülltes Rechteck gezeichnet. Sie können dieses Rechteck mit der Maus über den gesamten Bildschirm verschieben. (Beachten Sie auch hier den Einsatz des XOR-Zeichenmodus.) Sobald Sie den linken Mausknopf betätigen, wird an der momentanen Stelle das bewegbare Rechteck fest auf den Bildschirm gezeichnet. Halten Sie den Mausknopf gedrückt, so können Sie mit Ihrem Rechteck sogar zeichnen. Mit der rechten Maustaste jedoch wechseln Sie die Nummer des Punktmusters, mit dem das besagte Objekt gefüllt ist. So können Sie schöne Bilder erzeugen und den Rechteck-Befehl auf kennenlernen.

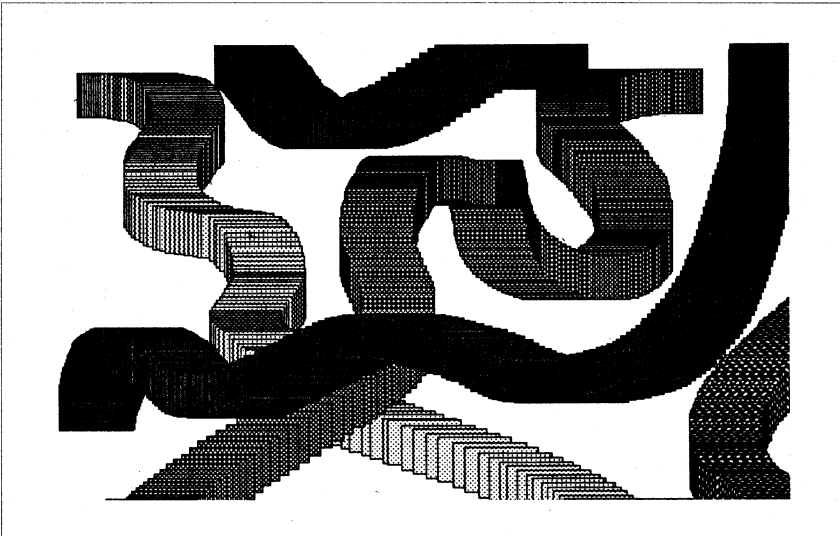


Abb. 15.15: Rechtecke

Eine kleine Variation des Programms wurde direkt eingebaut. Wenn Sie die beiden PBOX-Befehle durch den Befehl PRBOX ersetzen, dann sehen Sie das erste Mal, wie ordentlich ein Rechteck mit abgerundeten Ecken aussehen kann. Mit diesem kleinen Programm aber noch nicht genug. Das folgende Programm, das nur unter der hohen oder der niedrigen Auflösung läuft, sollten Sie in jedem Fall einmal ausprobieren, auch wenn Sie die internen Rechnungen nicht ganz verstehen:

```

/ *****
/ **
/ ** Rechtecke - einmal anders **
/ **
/ *****
/
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
IF xauf1%=639 AND yauf1%=199
  PRINT "Bitte unter 320x200-Auflösung starten!"
  VOID INP(2)
  EDIT
ENDIF
'
zwis=PI/180                                !Arbeitswert
    
```



```

,
DEFBN f(i,k,h)=yauf1%+k*SIN(h-zwis*i)
,
' Schlangen zeichnen:
@schlange(xauf1%/3,0)
@schlange(-xauf1%/3,0)
@schlange(xauf1%/3,PI/2)
@schlange(-xauf1%/3,PI/2)
,
VOID INP(2)
EDIT
,
,
PROCEDURE schlange(k,h)
  DEFFILL ,2,11          !Füllmuster
  DEFLINE ,3             !Umrandungsdicke
  FOR i=0 TO 360 STEP 8+2*INT(320/(xauf1%+1))
    x=xauf1%/12+i*3*xauf1%/640
    y=FN f(i,k,h)
    dd=yauf1%/5*SIN(zwis*i/2)
    x1=(x+10+dd)/2
    x2=(x+20-dd)/2
    y1=(y+dd)/2
    y2=(y-dd)/2
    PRBOX x1,y1,x2,y2    !Fläche füllen
    RBOX x1,y1,x2,y2     !Fläche umranden
  NEXT i
,
RETURN

```

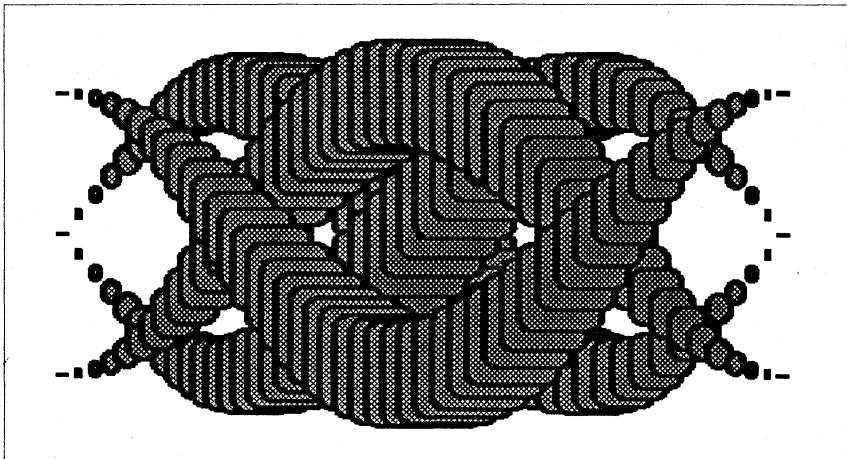


Abb. 15.16: Rechtecke - einmal anders

Und damit die C-Freunde auch nicht zu kurz kommen, hier ein kleines C-Demoprogramm:

```
/*  
**  
** Rechtecke aus C-Sicht **  
**  
***/  
int contrl[12],  
    intin[128],  
    ptsin[128],  
    intout[128],  
    ptsout[128];  
int gl_apid;  
int gem_handle;  
int handle;  
int count;  
int gl_wchar, gl_hchar, gl_wbox, gl_hbox;  
int work_in[12],  
    work_out[57],  
    pxyarray[10];  
  
/*  
*      Initialisiere Applikation      *  
*/  
aes_init()  
{  
    int i;  
    gl_apid = appl_init();  
    gem_handle = graf_handle(&gl_wchar, &gl_hchar,  
                             &gl_wbox, &gl_hbox );  
}  
  
/*  
*      Terminiere Applikation      *  
*/  
aes_exit()  
{  
    gemdos(0x1);          /* Auf Taste warten */  
    appl_exit();  
}  
  
/*  
*      öffne virtuelle Workstation      *  
*/  
open_vwork()  
{  
    int i;  
  
    for (i=0; i<10; work_in[i++]=1);  
    work_in[10] = 2;  
    handle = gem_handle;
```

```

    v_opnvwk(work_in, &handle, work_out);
}

/*****
/*      Schließe virtuelle Workstation      */
*****/

#define close_vwork()  v_clsvwk(handle);

/*****
/*      Hauptprogramm      */
*****/

main()
{
    int  xaupl,
        yaupl,
        x,
        y,
        taste,
        flag;
    int  d=5;                /* Anzahl der Durchläufe */
    aes_init();              /* Initialisierung      */
    open_vwork();            /* Auflösung feststellen */
    xaupl = work_out[0];
    yaupl = work_out[1];
    vswr_mode(handle, 1);    /* Schreibmodus: Replace */
    vsf_color(handle, 1);    /* Füllfarbindex = 1     */
    vsf_interior(handle, 2); /* Punktierung ein      */
    /* 5 Durchläufe: */
    for (flag=0; flag < 5; flag++)
    {
        v_clrwk(handle);    /* Bildschirm löschen   */
        /* Zeichenschleife: */
        for (x=0, y=0; x < xaupl/2; x=x+d, y=y+d)
        {
            vsf_style(handle, (x/d)%24+1); /* Punkt-Muster einstellen */
            pxyarray[0] = x;                /* Eckpunktkoordinaten    */
            pxyarray[1] = y;                /* festlegen              */
            pxyarray[2] = xaupl-x;
            pxyarray[3] = yaupl-y;
            switch (flag)
            {
                case 0:
                    vr_recfl(handle, pxyarray); /* FILL RECTANGLE */
                    break;
                case 1:
                    vsf_perimeter(handle, 0);    /* Umrahmung aus */
                    v_bar(handle, pxyarray);    /* BAR           */
                    break;
                case 2:
                    vsf_perimeter(handle, 1);    /* Umrahmung ein */
                    v_bar(handle, pxyarray);    /* BAR           */
            }
        }
    }
}

```

```

        break;
case 3:
    vsf_perimeter(handle, 0);    /* Umrahmung aus */
    v_rfbbox(handle, pxyarray); /* FILLED ROUNDED */
                                /* RECTANGLE */
    break;
case 4:
    vsf_perimeter(handle, 1);    /* Umrahmung ein */
    v_rfbbox(handle, pxyarray); /* FILLED ROUNDED */
                                /* RECTANGLE */
    break;
    }
}
gemdos(1);                      /* Auf Taste warten. */
}
close_vwork();                  /* Programmende */
aes_exit();
}

```

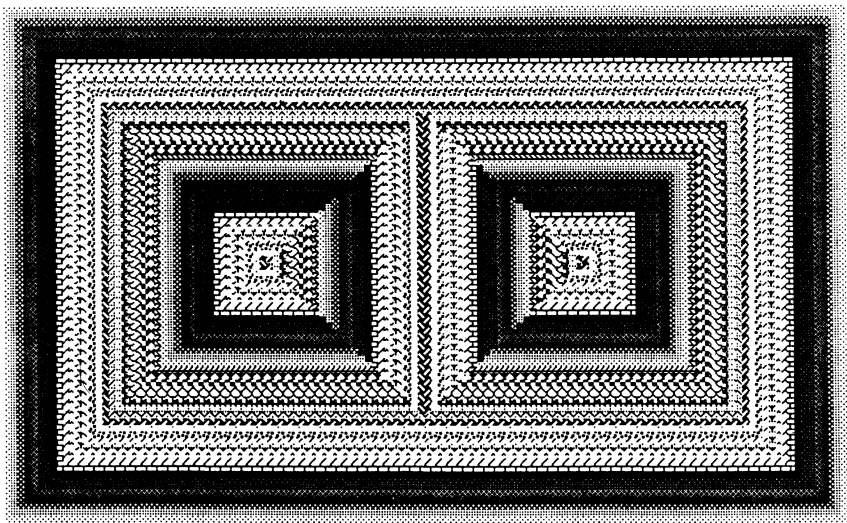


Abb. 15.17: Rechtecke aus C-Sicht

Dieses Programm zeigt Ihnen die Programmierung der verschiedenen Rechteck-Funktionen und demonstriert den Zusammenhang zwischen ihnen (z.B. was die Umrandung angeht etc.).

15.5 Beliebige Flächen ausfüllen

Ein sehr begehrter Befehl ist die Funktion zum Ausfüllen beliebiger Flächen. Sie können sich ja ein paar Minuten einmal Gedanken darüber machen, wie eine beliebige, vorher umrahmte Fläche ausgefüllt werden könnte. Sie werden sehen, das ist nicht so einfach. Nun, eigentlich brauchen Sie auch gar nicht über die Lösung dieses Problems nachzudenken, da es bereits von Ihrem VDI in komfortabler Weise gelöst ist. Hierzu existiert nämlich ein eigener VDI-Befehl namens **CONTOUR FILL** (103 VDI).

Wie gesagt, haben Sie mit diesem Befehl die Möglichkeit, beliebige Flächen auszufüllen. Sicher werden Sie ihn sehr viel weniger benötigen als auf anderen Rechnern, da der Atari ST bereits sehr viele Funktionen bietet, die von Haus aus Flächen ausfüllen.

Nun, wie geht das alles vonstatten? Sie haben sich also eine Fläche ausgedacht. Diese Fläche muß nun einige Bedingungen erfüllen, um durch **CONTOUR FILL** ausgefüllt zu werden. Sie muß, und das ist allererstes Gebot, als abgeschlossene Fläche erkennbar sein. Das heißt, Sie sollte eine einheitliche Farbe besitzen und erkennbare und ununterbrochene Farbabgrenzungen zu anderen Flächen haben, da sonst die Gefahr besteht, daß diese anderen Flächen ebenfalls ausgefüllt werden. Eine andere Möglichkeit ist die Umrahmung einer auszufüllenden Fläche mit Linien-Objekten. Diese Umrahmung muß ebenfalls lückenlos sein, um ein "Ausfließen" der Füllfarbe zu verhindern. Ein weiteres Kriterium: Die Fläche darf nicht zu kompliziert gerändert sein, da sich der Computer alle Einbuchtungen und Nischen merken muß, um sie später zu füllen. Sind nun zu viele Einbuchtungen vorhanden, dann läuft irgendwann einmal der Speicher über, und das Ausfüllen wird abgebrochen.

Die auszufüllende Fläche darf allerdings durchaus eine bestimmte Farbe haben (möglichst ohne Muster).

Sie übergeben dem Befehl einfach die Koordinaten (x,y) eines beliebigen Startpunktes innerhalb der auszufüllenden Fläche. Damit weiß der Computer, welche Fläche Sie ausgefüllt haben wollen und beginnt seine Arbeit. Sie müssen jedoch noch einen weiteren Parameter (index) übergeben. Dieser Parameter gibt den Index der Farbe an, die für Sie die Begrenzung Ihrer auszufüllenden Fläche darstellt. Damit können Sie sogar mehrere verschiedene, zusammenhängende Flächen unterschiedlicher Farbe durch einen **CONTOUR FILL**-Befehl ausfüllen. Allerdings ist diese Option in den beiden niedrigen Auflösungen mit Vorsicht zu genießen, da sich der Rechner bei zu großen auszufüllenden Flächen unterschiedlicher Farbe schnell "verhaspelt", nicht mehr weiter

kommt und schließlich aufgibt, indem er zurückkehrt, ohne die Fläche ordnungsgemäß ausgefüllt zu haben.

Wenn Sie dem Parameter index jedoch einen negativen Wert übergeben, so merkt sich der Rechner die Farbe des Startpunktes (x,y) und sieht alles als Begrenzung an, was ungleich dem Wert dieser Farbe ist. Dies ist der sicherste und auch am häufigsten verwendete Weg, er wird auch im GFA-BASIC verwendet.

Bemerkenswert an der VDI-Funktion CONTOUR FILL ist die Tatsache, daß er sich an alle Flächenattribute (außer perimeter - es wird keine Umrandung gezeichnet) hält. D.h. Sie können mit jeder beliebigen Farbe füllen (unabhängig von der momentanen Farbe der Fläche) und sogar jedes beliebige Füllmuster verwenden. Da sogar der Zeichenmodus berücksichtigt wird, sind Sie mit diesem Befehl in der Lage, im Transparent- oder Invers-Transparent-Modus auf eine Fläche beliebiger Farbe ein beliebiges Muster einer beliebigen anderen Farbe zu setzen. (Dieses Muster kann natürlich wiederum ein Vielfarbmuster sein.) Sie sehen, welche Möglichkeiten dieser Befehl bietet.

Um Ihnen CONTOUR FILL in der Praxis zu demonstrieren, haben wir uns etwas ausgedacht, was Ihnen sicher Spaß machen und Ihnen auch noch einmal die anderen Grafik-Funktionen etwas näherbringen wird. Doch zuvor sollten wir auf die GFA-Implementierung des CONTOUR FILL eingehen:

FILL x,y (GFA-BASIC-Befehl)

Ausfüllen einer beliebigen definierten Fläche mit Füllmuster

x: x-Koordinate des Startpunktes
y: y-Koordinate des Startpunktes

Wie Sie sehen, wurde der Parameter index fortgelassen. GFA-BASIC übergibt der Routine CONTOUR FILL einfach einen negativen Wert als index. Die daraus resultierenden Folgen haben Sie gerade kennengelernt.

Ein - wirklich kleiner - Grafik-Editor, der all das bisher in diesem Buch Beschriebene an Grafik-Know-How verwendet, soll Sie hier schon einmal mit den Grundzügen der CAD-Programmierung vertraut machen:



Abb. 15.18: Mini-Mini-CAD-Zeichner

```

' *****
' **                                     **
' ** Der Mini-Mini-CAD-Zeichner **
' **                                     **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
handle%=DPEEK(VDIBASE+40) !aktuelle Handle-Nummer holen
'
@menuezeile               !Menüzeile zeichnen
men%=4                    !Start auf Handzeichnen
fuellm1%=2                !Füllmusterstil
fuellm2%=1                !Füllmusterwahl
'
REPEAT
  MOUSE x1%,y1%,b%
  '
  IF b%<>0                 !Mausknopf gedrückt?
    ' ja:
    IF y1%>mhoehe%         !unter Auswahlzeile?
      ' Menü-Routinen aufrufen:
      ON men%+1 GOSUB dummy,linie,rechteck,flaeche,handzeichnen,dummy
    '
    ELSE                   !nein => Grafikfunktion wählen
      men2%=men%           !alte Menüpunktnummer merken
      men%=x1%/mbreite%    !neue Menüpunktnummer feststellen
      ' Menü-Routinen direkt nach Anklicken aufrufen:
      ON men%+1 GOSUB loeschen,dummy,dummy,dummy,dummy,fuellmuster
    '
  ENDIF
ELSE
  ' Mausknopf nicht gedrückt:
  x0%=x1%                 !Koordinaten merken (z.B. für Handzeichnen)
  y0%=y1%
ENDIF
'
UNTIL INKEY$<>" "        !bis Taste betätigt

```



```

    @menuezeile                !Menüzeile zeichnen
RETURN
'
' Linie zeichnen:
PROCEDURE linie
    GRAPHMODE 3                !XOR-Modus
    ,
    REPEAT
        MOUSE x2%,y2%,b%      !neue Koordinaten holen
        DRAW x1%,y1% TO x2%,y2% !Linie zeichnen
        DRAW x1%,y1% TO x2%,y2% !und wieder löschen
    UNTIL b%=0                !solange bis Mausknopf wieder los
    ,
    GRAPHMODE 1                !Replace-Modus
    DRAW x1%,y1% TO x2%,y2%    !Linie zeichnen
    ,
RETURN
'
' Rechteck zeichnen:
PROCEDURE rechteck
    GRAPHMODE 3                !XOR-Modus
    ,
    REPEAT
        MOUSE x2%,y2%,b%      !neue Koordinaten holen
        BOX x1%,y1%,x2%,y2%    !Rechteck zeichnen
        BOX x1%,y1%,x2%,y2%    !und wieder löschen
    UNTIL b%=0                !solange bis Mausknopf wieder los
    ,
    GRAPHMODE 1                !Replace-Modus
    BOX x1%,y1%,x2%,y2%        !Rechteck zeichnen
    ,
RETURN
'
' Handzeichnen:
PROCEDURE handzeichnen
    DRAW x0%,y0% TO x1%,y1%    !Linie vom letzten Punkt
    x0%=x1%                    !und Endkoordinaten merken
    y0%=y1%
RETURN
'
' Fläche ausfüllen:
PROCEDURE flaeche
    DEFFILL ,fuellm1%,fuellm2% !Füllmuster einstellen
    FILL x1%,y1%                !Fläche ausfüllen
RETURN
'
' Füllmuster festlegen:
PROCEDURE fuellmuster
    @vsf_perimeter(handle%,0)  !Umrahmung aus
    DEFFILL ,0                  !Muster auf leer
    PBOX 0,0,xaufl%,mhoeh%     !Menüzeile löschen
    @vsf_perimeter(handle%,1)  !Umrahmung ein
    ,

```

```

REPEAT
  DEFFILL ,fuellm1%,fuellm2%  !akt. Füllmuster einstellen
  ' Kasten zeichnen:
  PBOX xaupl%*0.4,0,xaupl%*0.6,mhoehe%
  ,
  b%=MOUSEK
  IF b%=1                      !linker Mausknopf
    fuellm2%=fuellm2%+1      !Musterwahl erhöhen
    IF fuellm1%=2             !punktiert?
      ' ja:
      IF fuellm2%>24          !Musterwahl Ende?
        fuellm1%=3           !auf Schraffierung
        fuellm2%=1           !und Musterwahl 1
      ENDIF
    ELSE
      ' nein:
      IF fuellm2%>11          !Musterwahl Ende?
        fuellm1%=2           !auf Punktierung
        fuellm2%=1           !und Musterwahl 1
      ENDIF
    ENDIF
  ENDIF
  PAUSE 10
  UNTIL b%=2                  !bis rechter Mausknopf
  @menuezeile
RETURN
' SET FILL PERIMETER VISIBILITY
PROCEDURE vsf_perimeter(handle%,per_vis%)
  DPOKE CONTRL+2,0
  DPOKE CONTRL+6,1
  DPOKE CONTRL+12,handle%
  DPOKE INTIN,per_vis%
  VDISYS 104
RETURN

```

Wir haben es hier mit einem kleinen Programm zu tun, mit dessen Hilfe Sie einfachste Grafiken mit Ihrer Maus erstellen können.

Die möglichen Zeichenfunktionen werden Ihnen dabei in der obersten Menüzeile zur Auswahl gestellt. Durch Anklicken wählen Sie eine dieser Optionen. So können Sie den Bildschirm löschen, Linien und Rechtecke zeichnen, beliebige Flächen ausfüllen, aus freier Hand zeichnen und ein beliebiges Füllmuster für Flächen wählen. Die Bedienung dürfte nicht allzu schwierig sein.

Um Ihr Bild zu löschen, klicken Sie einfach den entsprechenden Menüpunkt an. Wählen Sie den Modus "Linien zeichnen", so können Sie mit Betätigung der linken Maustaste den Startpunkt der Linie festlegen. Lassen Sie den Maus-

knopf wieder los, so bestimmen Sie den Endpunkt gemäß der Position des Mauszeigers. In dem Zeitraum zwischen Drücken und Loslassen sehen Sie die Linie Ihrem Mauszeiger folgen.

Rechtecke werden auf die gleiche Weise "zu Bildschirm" gebracht. Haben Sie den Menüpunkt "Flächen ausfüllen" angewählt, so brauchen Sie mit dem Mauszeiger nur auf die auszufüllende Farbe zu tippen, und schon wird Ihre Fläche ausgefüllt - aber wehe Sie haben ein Loch in der Umrandung gelassen. Wenn Sie die Option "Handzeichnen" ausprobieren, dann wird stets Punkt für Punkt entlang den Bewegungen des Mauszeigers gezeichnet, solange Sie die linke Maustaste niedergedrückt halten.

Die letzte Funktion erlaubt es Ihnen, ein Füllmuster auszuwählen. Hierzu klicken Sie den Punkt an, und es erscheint ein gefülltes Rechteck in der Menüzeile. Sie können nun mit Betätigung der linken Maustaste ein beliebiges Füllmuster anwählen. Tippen Sie dagegen auf die rechte, so gelangen Sie wieder zurück ins normale Bild. Mit einem einfachen Tastendruck verlassen Sie, wie so oft, das Programm.

Das Programm ist durchgehend dokumentiert, trotzdem wollen wir es an dieser Stelle durchsprechen. Es wurde bewußt so programmiert, daß Sie es sehr leicht durch eigene Funktionen erweitern können, was Sie vielleicht sogar einmal tun sollten.

Zu Programmanfang wird zunächst die selbstdefinierte Menüzeile auf den Bildschirm gebracht. Hierfür ist die Prozedur "menuezeile" zuständig. In ihr findet nach der Initialisierung verschiedener Variablen, wie die Breite und die Höhe eines Menüpunktes, das Übertragen der Menünamen aus den dafür zuständigen DATA-Zeilen auf den Bildschirm statt. Es existieren dabei stets zwei Sätze von Menünamen, einer für die beiden höheren Auflösungen mit je 80 Zeichen pro Zeile (HGR, MGR) und einer für NGR mit 40 Zeichen pro Zeile.

Die Texte werden mit einem Ihnen noch unbekannten Befehl (TEXT) geschrieben, der verschiedene besondere Möglichkeiten der Textgestaltung erlaubt und in einem späteren Kapitel gesondert besprochen wird.

Das Rückgrat des gesamten Programms befindet sich in den ersten Zeilen. Es wird durch die große REPEAT...UNTIL-Schleife repräsentiert. Hier wird ständig die Maus abgefragt. Wurde ein Mausknopf betätigt, so gibt es zwei Möglichkeiten der Verzweigung:

Steht der Mauszeiger bei dem Mausklick auf der Menüzeile ($0 < y < \text{Menüpunkthöhe}$), dann wurde ein Menüpunkt angewählt, dessen Nummer aus Mausposition und Menüpunktbreite errechnet wird. Als dann

findet ein Sprung auf die entsprechende Routine statt (ON...GOSUB). Diese Routine tätigt dann Vorgänge, die direkt nach Menüpunktanwahl ausgeführt werden müssen (z.B. Bildschirm löschen oder Füllmusterwahl). Die Vorgänge, die erst später, z.B. nach Eintritt in das Zeichenfeld stattfinden sollen, werden hier allerdings noch nicht aufgerufen, sondern durch dummy-Sprünge (leere Funktion) überbrückt.

Sie finden erst dann statt, wenn der Mauszeiger beim Klick unter der Menüzeile steht (erstes ON...GOSUB). Dies sind hier das Zeichnen der Linien, Rechtecke, das Füllen von Flächen und das freie Zeichnen aus der Hand.

Damit wäre zu diesem Programm eigentlich schon fast alles gesagt. Was bleibt, sind die einzelnen Routinen, die durch die Menüwahl aufgerufen werden. Sie sind leicht zu durchschauen. Lediglich die Routine zur Musterwahl scheint auf den ersten Blick ein wenig komplizierter. Das hat allerdings seinen Grund nur darin, daß sowohl Punkt- als auch Schraffurmuster eingestellt werden können.

Wenn Sie einen eigenen Menüpunkt hinzufügen wollen, so schreiben Sie seinen Namen zunächst in die dafür zuständigen DATA-Zeilen (jeweils in 80- und in 40-Zeichen-Version). Sie müssen nur aufpassen, daß er nicht zu lang wird, da er ansonsten kaum noch lesbar ist. Vergessen Sie auch nicht, die Anzahl der Menüpunkte zu verändern.

Jetzt kommt die Hauptarbeit. Das Schreiben der eigentlichen Routine kann Ihnen niemand abnehmen. Den Namen der zuständigen Prozedur müssen Sie dann in die ON...GOSUB-Zeilen mit aufnehmen (an dieselbe Stelle wie in den DATA-Zeilen). Dabei können Sie wählen, ob Ihre Routine direkt nach dem Anwählen des Menüpunktes aufgerufen werden soll (zweites ON...GOSUB) oder erst, wenn sich der Mauszeiger im Zeichenfeld befindet. Vielleicht braucht Ihre Routine ja auch beide Aufrufe. In jedem Fall müssen Sie im unbenutzten ON...GOSUB einen dummy-Sprung eingliedern.

Sollte der Platz in der Menüzeile einmal nicht mehr ausreichen, dann programmieren Sie doch einfach eine zweite hinzu. Eine andere Möglichkeit wäre das Zeigen einer Untermenüleiste nach dem Anklicken eines Hauptmenüpunktes.

15.6 Kreise und andere runde Sachen

In diesem Kapitel geht es um Kreise, Ellipsen und sonstige Figuren, die damit zusammenhängen. Runde Objekte zählen zu den rechenintensiven und komplizierteren grundsätzlichen Grafikfiguren, da zu ihrer Berechnung meist re-

chenintensive Funktionen wie Sinus oder Cosinus herangezogen werden müssen. Es gibt zwar Algorithmen, die helfen, solche mathematischen Klimmzüge zu vermeiden, die aber auch meist nur spezielle Figuren fertigen (z.B. nur vollständige Kreise etc.).

Runde Figuren sind aber in vielen Bereichen der Grafikdarstellung sehr gefragt und oft unabdingbar (Kuchendiagramme etc.). Aus diesem Grunde stellt Ihnen VDI eine Reihe von verschiedenen Funktionen zur Verfügung, die solche Aufgaben bewältigen. Dabei sind an einigen Stellen, nämlich wenn es um Kreis- oder Ellipsenausschnitte, Ellipsenbögen bzw. Kreisbögen geht, Winkelangaben erforderlich. Diese Winkel werden grundsätzlich in 1/10 Altgradeinheiten angegeben. Einen 90-Gradwinkel müssten Sie also mit 900, einen 180-Gradwinkel mit 1800 angeben. Das folgende Schaubild zeigt Ihnen den gesamten Wertebereich von 0-3600:

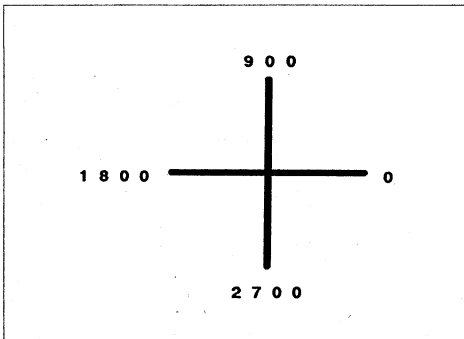


Abb. 15.19: Gradeinteilung VDI-Kreisfunktion

Wenn Sie größere Winkel als 3599 (359,9 Grad) angeben, so wird so lange 3600 abgezogen, bis das Ergebnis im erlaubten Wertebereich liegt, es erscheint also kein "Illegal quantity error" oder so etwas. Ein Winkel von z.B. 3610 Einheiten entspräche dann also einem Winkel von 10 Einheiten (= 1 Grad).

Vorsicht! Bei einigen der folgenden Funktionen kann es zu Systemabstürzen kommen, wenn Teile von Objekten extrem über den Bildschirmrand hinausragen. Das heißt allerdings nicht, daß sie gar nicht darüber hinausragen dürfen. Die Befehle sind da - wie alle anderen Grafikbefehle auch - sehr tolerant. Die beobachteten Abstürze liegen daran, daß in Fällen, in denen solche extrem großen Auslagerungen stattfinden, Werte außerhalb des Bildschirms gePOKET werden. Die RAM-Disk des Autors hatte darunter einige Male böse zu leiden. Verschaffen wir uns doch einmal einen Überblick über die vorhandenen Funktionen:

ARC (11/2 VDI)

Zeichnen eines Kreisbogens

CIRCLE (11/4 VDI)

Zeichnen eines ausgefüllten Kreises

PIE (11/3 VDI)

Zeichnen eines ausgefüllten
Kreisausschnittes

ELLIPSE (11/5 VDI)

Zeichnen einer ausgefüllten Ellipse

ELLIPTICAL ARC (11/6 VDI)

Zeichnen eines Ellipsenbogens

ELLIPTICAL PIE (11/7 VDI)

Zeichnen eines ausgefüllten
Ellipsenausschnittes

Für Assembler-Programmierer ist es bei diesen Funktionen notwendig, die unbenutzten Array-Elemente auch mit Nullen zu füllen!

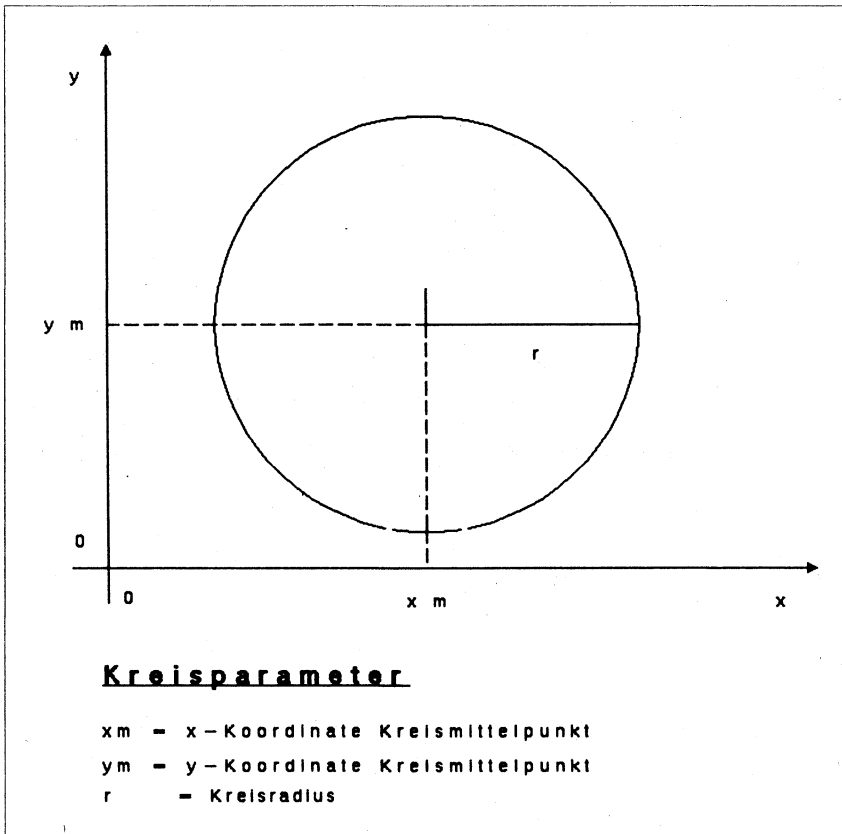


Abb. 15.20: Kreis-Parameter

Den folgenden Schaubildern können Sie die Bedeutung der einzelnen Parameter entnehmen. Zunächst die Schaubilder für die Parameterübergabe der Kreis-Objekte mit den Übergabeparametern Kreismittelpunktskoordinaten und Kreisradius für den Kreis, zusätzlich Winkelstart und -ende für Kreisbögen und -ausschnitte:

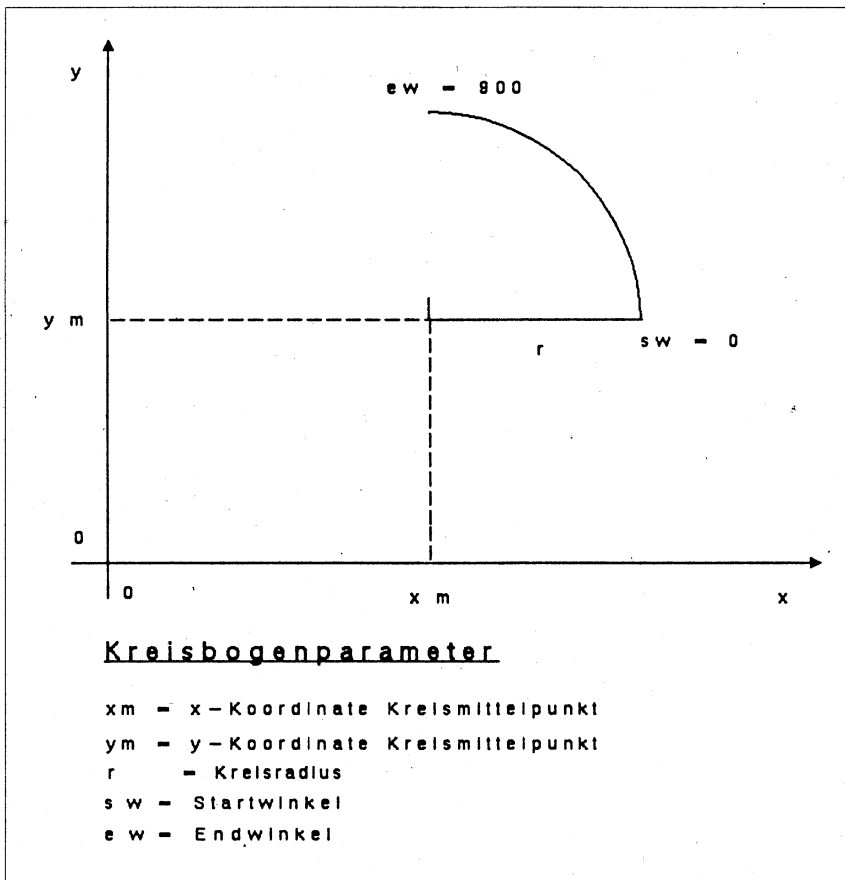


Abb. 15.21: Kreisbogen-Parameter

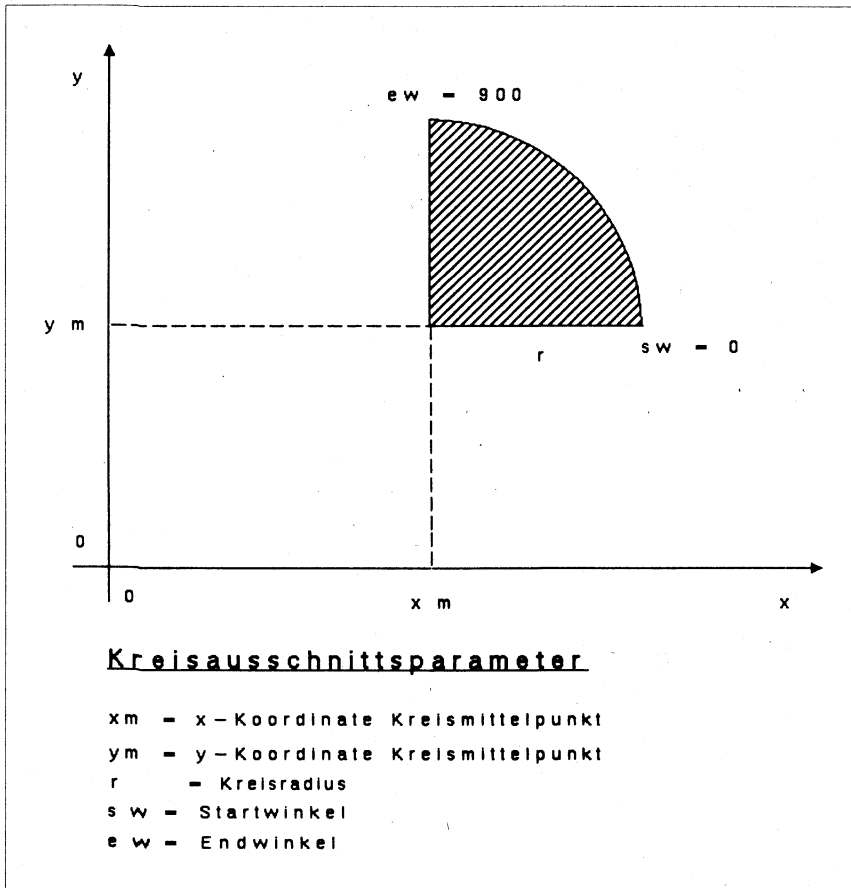


Abb. 15.22: Kreisausschnitt-Parameter

Bei den elliptischen Objekten sieht das Ganze eigentlich recht ähnlich aus. Der Radius einer Ellipse kann jedoch zwangsläufig nicht mehr so einfach wie beim Kreis angegeben werden. Vielmehr werden hier zwei Radien benötigt, die die Ausdehnung einer Ellipse in x- und in y-Richtung festlegen:

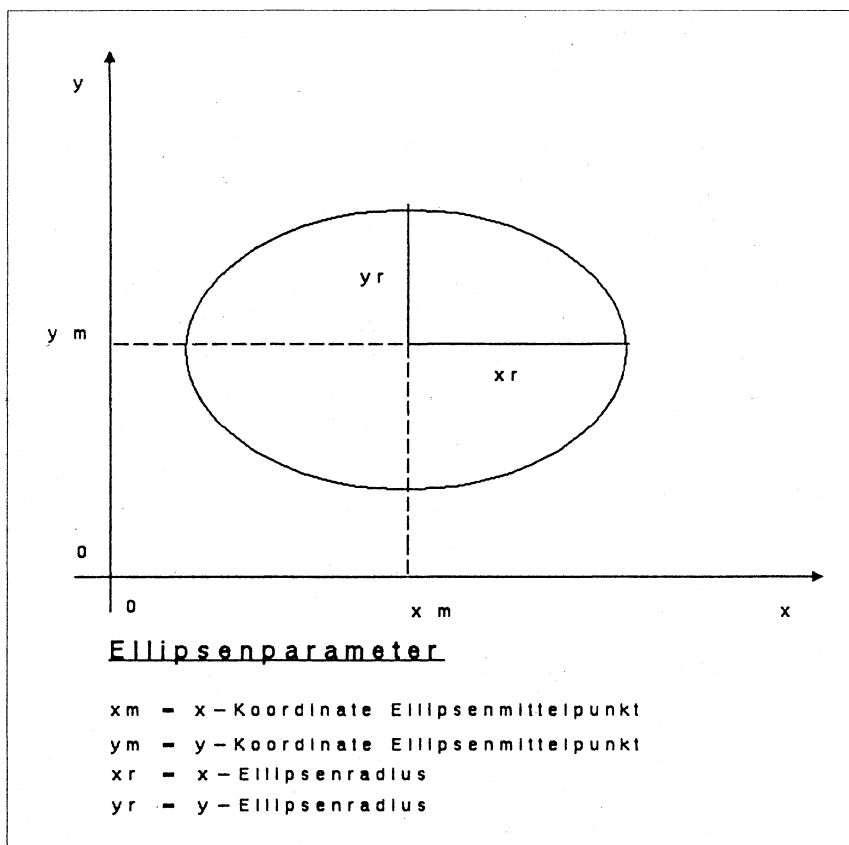


Abb. 15.23: Ellipsen-Parameter

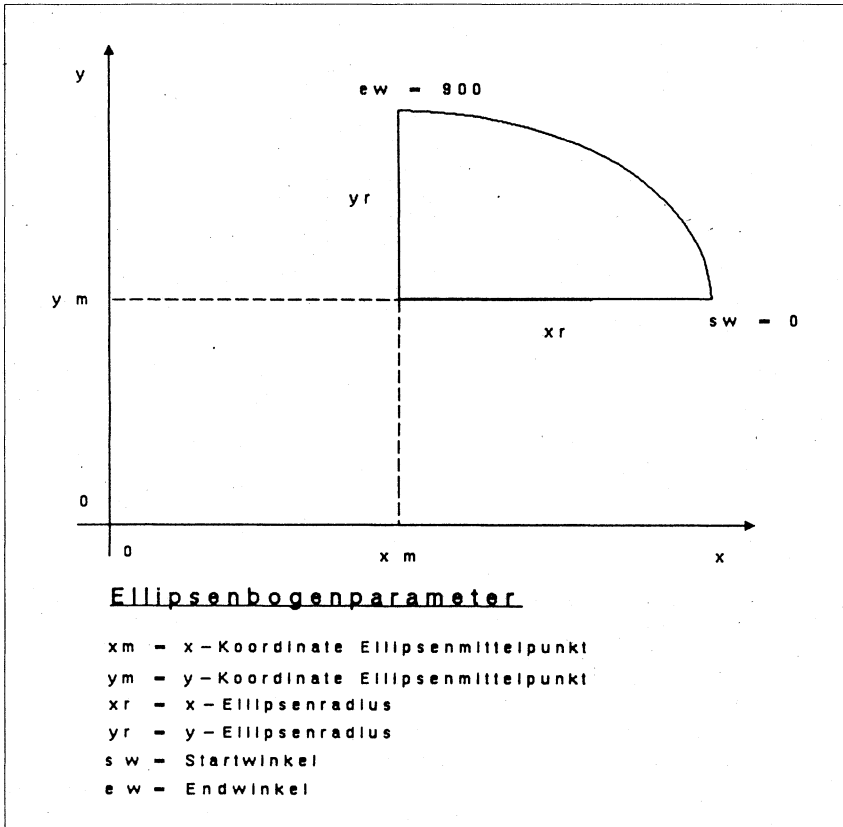


Abb. 15.24: Ellipsenbogen-Parameter

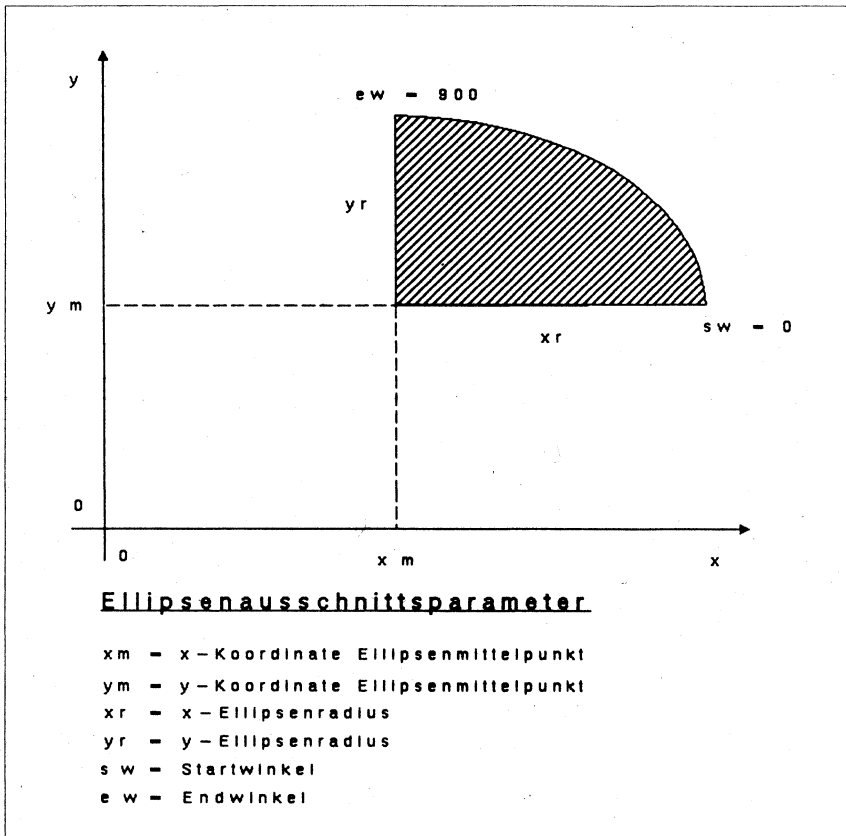


Abb. 15.25: Ellipsenausschnitt-Parameter

Wie Sie sehen, haben Sie mannigfaltige Möglichkeiten der Gestaltung. Interessant ist hier wiederum die Tatsache, daß es für Kreis und Ellipse nur die hier vorgestellten Flächenoperationen gibt. Wollen Sie dagegen auch die dazugehörigen Linien-Objekte (also Kreislinie, Ellipsenlinie) darstellen, so müssen Sie sich der Bogen-Funktionen (ARC und ELLIPTICAL ARC) mit dem Startwinkel 0 und dem Endwinkel 3600 bedienen. Genügt Ihnen allerdings eine Kreislinie ohne Berücksichtigung der Linien-Attribute, so können Sie diese auch zeichnen, indem Sie die CIRCLE/ELLIPSE-Funktion mit Fülltyp 0 (Hintergrundfarbe bzw. leer) und im Transparent-Modus anwenden.

Eine wichtige Information sollte hier nicht fehlen, da der Autor selbst von ihr auf dem falschen Fuße erwischt wurde. Solange Sie im hochaufgelösten, mo-

nochromen Modus oder in der niedrigsten Auflösung von 320x200 Punkten arbeiten, läuft alles nach Plan. Zeichnen Sie die Gebilde jedoch in der Auflösung 640x200 Punkte, müßte ein Kreis mit dem Radius r rein theoretisch zur Ellipse werden, eine Ellipse mit den identischen Radien x_r und y_r wäre dann trotz der beiden gleichen Werte ebenfalls eine Ellipse, da ein Punkt des Bildschirms doppelt so hoch wie breit ist. Dem ist allerdings nicht so. Das Betriebssystem halbiert vor dem Zeichnen den Radius in y -Richtung, so daß ein Kreis wieder ein Kreis wird. Das ist natürlich auf den ersten Blick sehr schön. Wenn Sie allerdings eigene Kreisberechnungen und die Kreisfunktionen nebeneinander verwenden, wie dies aus bestimmten Gründen durchaus der Fall sein kann, dann kompliziert dieser Tatbestand die Programmierung zum Teil nicht unerheblich.

Diese Probleme sind natürlich nur dann relevant, wenn Sie im Raster-Koordinatensystem (RC) arbeiten, wie wir das bisher immer getan haben. Bei den normalisierten Koordinaten müssen Sie sowieso andere Berechnungsgrundlagen annehmen.

Die folgenden Ausführungen machen Sie mit den entsprechenden GFA-Befehlen vertraut:

CIRCLE x_m, y_m, r (GFA-BASIC-Befehl)

Zeichnen eines Kreises als Linien-Objekt

x_m : x -Mittelpunktskoordinate
 y_m : y -Mittelpunktskoordinate
 r : Kreisradius

PCIRCLE x_m, y_m, r (GFA-BASIC-Befehl)

Zeichnen eines ausgefüllten Kreises als Flächen-Objekt

x_m : x -Mittelpunktskoordinate
 y_m : y -Mittelpunktskoordinate
 r : Kreisradius

ELLIPSE x_m, y_m, x_r, y_r (sw,ew) (GFA-BASIC-Befehl)

Zeichnen einer Ellipse oder eines Ellipsenbogens als Linien-Objekt

x_m : x -Mittelpunktskoordinate

ym: y-Mittelpunktskoordinate
 xr: Ellipsenradius in x-Richtung
 yr: Ellipsenradius in y-Richtung
 sw: Startwinkel
 ew: Endwinkel

PELLIPSE xm,ym,xr,yr (,sw,ew) (GFA-BASIC-Befehl)

Zeichnen einer ausgefüllten Ellipse oder eines Ellipsenausschnittes als Flächen-Objekt

xm: x-Mittelpunktskoordinate
 ym: y-Mittelpunktskoordinate
 xr: Ellipsenradius in x-Richtung
 yr: Ellipsenradius in y-Richtung
 sw: Startwinkel
 ew: Endwinkel

Wie Sie sicher bemerkt haben, können Sie bei diesen Befehlen sogar die entsprechenden Linien-Objekte erzeugen, ohne erst - wie oben besprochen - umständlich über die ARC-Funktion zu gehen (obwohl das GFA-intern natürlich getan wird). Die kreisförmigen Bögen und Ausschnitte müssen Sie hier allerdings über die Ellipsen-Funktion mit identischen Radien anwählen. Dabei geben Sie dann die Parameter sw und ew an, die Sie sonst auch weglassen können. Ansonsten erkennen Sie sicher die Parallelen zu den VDI-Funktionen.

Kommen wir jetzt aber zu der praktischen Anwendung der Befehle. Wir stellen Ihnen an dieser Stelle ein GFA-BASIC-Programm vor, das nur in der 320x200-Auflösung läuft, aber trotzdem - oder gerade deshalb - erstaunliche Effekte in Ihr Heim bringt. Anschließend sehen Sie, wie das alles unter C funktioniert.

```
' *****
' **                               **
' ** Farbe, Kreise und Flächen **
' **                               **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
IF yauf1%<>199 OR xauf1%<>399
  PRINT "Nur lauffähig in Auflösung: 320x200"
  VOID INP(2)
```

```
EDIT
ENDIF
,
handle%=DPEEK(VDIBASE+40)
,
DIM drad%(15),index%(15),farbe%(15)
,
@vsf_perimeter(handle%,0)           !keine Flächenränder!
,
SETCOLOR 0,0                        !Hintergrundfarbe=schwarz
,
' 15 Farben einlesen und setzen:
FOR i%=1 TO 15
    READ index%(i%),farbe%(i%)
    SETCOLOR index%(i%),farbe%(i%)
NEXT i%
,
' 16 Radiusdifferenzen einlesen:
FOR i%=0 TO 15
    READ drad%(i%)
NEXT i%
,
GRAPHMODE 1                        !Replace-Modus
,                                  !Versuchen Sie andere!
rad1%=100                          !Startradien
rad2%=40
,
' Zeichnen von 4 Bögen und 1 Kreis:
FOR i%=15 DOWNT0 0
    DEFFILL i%,1                    !Füllmuster einstellen
    ,                              !Versuchen Sie andere!
    PCIRCLE 0,0,rad1%               !Fäche zeichnen
    DEFFILL i%,2,10
    PCIRCLE xaufl%,yauf1%,rad1%
    DEFFILL i%,2,5
    PCIRCLE xaufl%,0,rad1%
    DEFFILL i%,2,2
    PCIRCLE 0,yauf1%,rad1%
    DEFFILL i%,3,1
    PCIRCLE xaufl%/2,yauf1%/2,rad2%
    ' Neue Radien berechnen:
    rad1%=rad1%-drad%(i%)
    rad2%=rad2%-drad%(i%)/2.5
NEXT i%
,
' Farben rollen:
REPEAT
    PAUSE MOUSEX/50                !Geschwindigkeit
    index%=index%(1)               !Index 1 retten
    VSYNC                           !Synchronisation
    FOR i%=1 TO 14
        index%(i%)=index%(i%+1)   !Indizes verschieben
```

```

        SETCOLOR index%(i%),farbe%(i%)!und Farbe setzen
NEXT i%
index%(15)=index%           !Index 1 nach Index 15
SETCOLOR index%(15),farbe%(15) !und Farbe setzen
'
UNTIL INKEY$<>" "           !bis Taste betätigt
'
'
SETCOLOR 0,7,7,7             !Hintergrundfarbe weiß
EDIT
'
'
' SET PERIMETER VISIBILITY
PROCEDURE vsf_perimeter(handle%,per_vis%)
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,per_vis%
    VDISYS 104
RETURN
'
'
' DATAs:
'
' Farbzuteilungstabelle: Index, Farbe
DATA 15,&H300,01,&H500,02,&H700,04,&H730
DATA 06,&H750,03,&H770,05,&H570,07,&H370
DATA 08,&H070,09,&H073,10,&H077,12,&H057
DATA 14,&H037,11,&H007,13,&H003
'
'
' Radiendifferenzentabelle:
DATA 1,2,2,3,3,5,6,8
DATA 10,8,6,5,3,3,2,2

```

Das Programm zeichnet eine Reihe ausgefüllter, immer kleiner werdender und in der Farbe wechselnder Kreise an mehrere Stellen des Bildschirms. Anschließend gerät das Ganze durch Farbwechsel in Bewegung. Die damit verbundenen Effekte können sich sehen lassen.

Das Programm selbst dürfte nicht allzu schwer zu verstehen sein. Der Farbrolleffekt wurde - mit anderen Farben und Objekten - bereits durchgesprochen. Aus diesem Grunde verzichten wir an dieser Stelle auf eine ausführliche Beschreibung. Notwendige Informationen entnehmen Sie bitte den Kommentarzeilen.

Damit wären wir bei unserem Beispiel für C-Programmierer, das ebenfalls keine besonderen Verständnisschwierigkeiten aufwerfen sollte. Hier werden Ihnen nur die Anwendung und Parameterübergabe der Kreis- bzw. Ellipsen-Objekte demonstriert:

```
/*
**
**  Kreise, Ellipsen und Anderes in C
**
**
**
**
*/
int contrl[12],
    intin[128],
    ptsin[128],
    intout[128],
    ptsout[128];
int gl_apid;
int gem_handle;
int handle;
int count;
int gl_wchar, gl_hchar, gl_wbox, gl_hbox;
int work_in[12],
    work_out[57],
    pxyarray[10];

/*
**  Initialisiere Applikation
**
*/
aes_init()
{
    int i;
    gl_apid = appl_init();
    gem_handle = graf_handle(&gl_wchar, &gl_hchar,
                           &gl_wbox, &gl_hbox );
}

/*
**  Terminiere Applikation
**
*/
aes_exit()
{
    gemdos(0x1);          /* Auf Taste warten */
    appl_exit();
}

/*
**  Öffne virtuelle Workstation.
**
*/
open_vwork()
{
    int i;

    for (i=0; i<10; work_in[i++]=1);
    work_in[10] = 2;
    handle = gem_handle;
    v_opnvwk(work_in, &handle, work_out);
}
```



```

/*****
/*      Schließe virtuelle Workstation      */
*****/

#define close_vwork()  v_clsvwk(handle);

/*****
/*      Hauptprogramm      */
*****/

main()
{
    int  xaupl,
        yaupl,
        x;
    int  d=5;                                /* Schrittweite      */
    aes_init();                               /* Initialisierung   */
    open_vwork();                             /* Auflösung feststellen */
    xaupl = work_out[0];
    yaupl = work_out[1];
    v_clrwk(handle);                          /* Bildschirm löschen */
    vswr_mode(handle, 1);                     /* Schreibmodus: Replace */
    vsf_color(handle, 1);                     /* Füllfarbindex = 1 */
    vsf_perimeter(handle, 1);                 /* Umrahmung ein */
    vsf_interior(handle, 2);                  /* Punktierung ein */
    vsf_style(handle, 20);                    /* Punktierung Nr. 20 */
    /* Zeichenschleife: */
    for (x=0; x<xaupl/5; x=x+d)
    {
        v_ellipse (handle, x, yaupl/2, x, yaupl/5);
        v_circle  (handle, xaupl/2, x/3, x/3);
        v_ellpie  (handle, xaupl-x, yaupl/2, x, yaupl/5,
                    1800-5*x, 1800+5*x);
        v_pieslice(handle, xaupl/2, yaupl-x, x,
                    900-2*x, 900+2*x);
    }
    close_vwork();                            /* Programmende      */
    aes_exit();
}

```

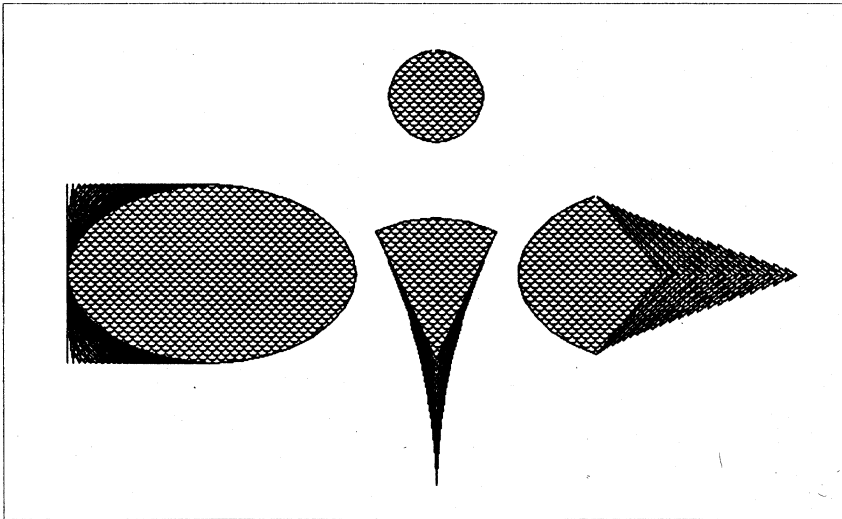


Abb. 15.26: Kreise, Ellipsen und anderes in C

15.7 Textdarstellung - gar nicht so einfach

Wir kommen zu einem außerordentlich interessanten Thema, das die Brücken schlägt zwischen der reinen grafischen Anwendung Ihres Computers zu allen anderen professionellen Sparten der Programmierung. Was sonst höchstens am Rande eines Grafikbuches dargestellt wurde, das bedarf nun peinlich genauer Erörterung.

Früher hatten die Computer aus Geschwindigkeitsgründen zumeist zwei getrennte Bildschirmspeicher, einen für hochauflösende Grafik, einen anderen für die Darstellung der Textoperationen. Dies ist im Zeitalter der schnellen Prozessoren, der Blitter und anderen Grafikprozessoren nicht mehr nötig. Text gehört nun genauso zur Grafik wie Linien, Kreise und andere Gebilde.

Somit wird das folgende Kapitel nicht nur für den fanatischen Grafikfreak essentiell, hier werden Programmierer von Datenbanken, Textverarbeitungssystemen, Kalkulationsprogrammen etc. notgedrungen hineinschauen, wenn sie auf den neuesten Stand des Know-hows Wert legen.

Doch Text - wir werden es sehen - nimmt doch noch eine gewisse Sonderstellung in der Reihe der Grafikobjekte ein. Hier gilt es, bestehende Normen zu

übernehmen (Satznormen etc.), aber auch die gute Lesbarkeit in jeder Auflösung zu gewährleisten. Verschiedene Schriftarten und Größen komplizieren das Ganze sehr zum Leidwesen des Programmierers.

15.7.1 Einfachste Textausgabe

Die einfachste Darstellung von Text knüpft noch an die alten Zeiten an, zu denen es nur und auch nur Buchstaben auf den Computer-Monitoren zu sehen gab. Sie hat in allen Programmiersprachen ihre Manifestation erhalten und ist nicht mehr wegzudenken.

Was in BASIC das PRINT, ist in C PRINTF, in Assembler das "Jump Subroutine to BASOUT" o.ä. Dies ist wohl so ziemlich der erste Befehl, den jeder Computerneuling vorgesetzt bekommt. Auch wir möchten hier mit ihm in dem folgenden kleinen BASIC-Programm beginnen:

```
' *****
' **                               **
' ** Der PRINT-Befehl             **
' **                               **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
a$="Sinuskurve"
z%=1
'
PRINT AT(20,1);"Die Sinuskurve"
'
FOR i%=1 TO LEN(a$)
  FOR x%=1 TO 80*xauf1%/639
    PRINT AT(x%,10*SIN(x%/12)+5+LEN(a$)/2+i%);MID$(a$,z%,1);
    z%=z%+1
    IF z%>LEN(a$)
      z%=1
    ENDIF
  NEXT x%
NEXT i%
'
VOID INP(2) !Auf Taste warten
EDIT
```

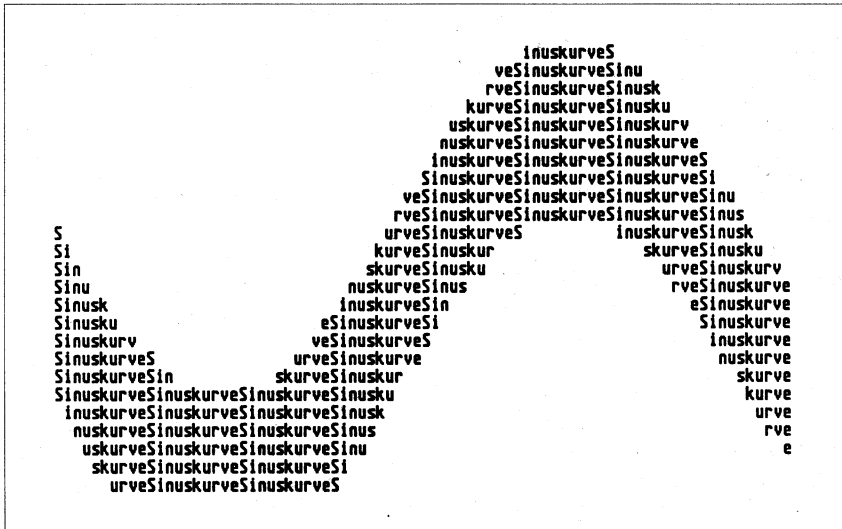


Abb. 15.27: Der PRINT-Befehl

In diesem Programm, das uns einen Text mehrmals sinusförmig auf den Bildschirm zaubert, erfährt der alte PRINT-Befehl mit AT(s,z) eine - man soll es kaum glauben - recht komfortable Ergänzung. s und z geben hier nämlich die Spalte und Zeile an, bei der mit der Ausgabe begonnen wird. Damit hätten wir so etwas ähnliches wie den PLOT-Befehl, nur auf Zeichen-Ebene.

Da unser Rechner in HGR eine Auflösung von 640x400, in MGR von 640x200 und in NGR von 320x200 besitzt, hieße das eigentlich, daß er in allen drei Auflösungen unterschiedlich viele Zeichen-Zeilen und -Spalten auf den Bildschirm bringen kann. Um dem abzuhelpen, verwendet er jedoch zwei verschiedene Standard-Zeichensätze mit unterschiedlicher Punktmatrix. In hochauflöster Grafik arbeitet er mit einem Zeichensatz, in dem jedes Zeichen aus einer Matrix von 8x16 Punkten aufgebaut ist. Damit ergeben sich $640/8=80$ Zeichen pro Zeile und $400/16=25$ Zeilen pro Bildschirm. In der mittleren Auflösung schaltet er auf einen Zeichensatz um, der für jedes Zeichen nur 8x8 Punkte verwendet. Die Folge ist, daß nicht, wie sonst denkbar, die Anzahl der darstellbaren Zeilen sinkt. Sie bleibt ebenso, wie unter HGR, bei 25. Dieser zweite, "kleinere" Zeichensatz ist übrigens ebenfalls in der hohen Auflösung darstellbar und ermöglicht so die Darstellung von 50 Zeilen auf dem normalen monochromen Bildschirm, aber davon später.

Lediglich bei der kleinen Auflösung, bei der sich die Zahl der Punkte in x-Richtung noch einmal halbiert, sinkt die Anzahl der Zeichen pro Zeile auf 40, da auch hier der 8x8-Zeichensatz unseres Atari ST verwendet wird.

Doch halten wir uns nicht lange bei diesem einfachen Beispiel auf und kommen gleich zu Atari-spezifischen Dingen, die wiederum mit dem VDI zusammenhängen. Unser Rechner besitzt nämlich, wie andere Rechner auch, die Möglichkeit, einen Textcursor darzustellen und zu steuern. Die hierzu notwendigen Befehle sind zwar im VDI implementiert, stammen aber eigentlich aus dem ganz normalen GEMDOS, das sich bekanntlich auf einer niedrigeren Ebene befindet. Unter GEMDOS sind diese Befehle auch unter dem Namen "VT52-Emulator" bekannt.

Wir werden hier nur die wichtigsten diesbezüglichen Kommandos vorstellen, also diejenigen, die auch im VDI vorkommen, um Ihnen nicht allzu lange die eigentlich interessanten grafischen Textbefehle vorzuenthalten. Es handelt sich dabei um eine Gruppe von VDI-Funktionen mit dem Namen ESCAPES. Sie besitzen alle, ähnlich den grafischen Grundfunktionen, dieselbe Funktionsnummer (5) und unterscheiden sich lediglich im Secundär-Opcode, der sogenannten Escape-ID. Sie stellen im VDI eigentlich eine jederzeit durch einen beliebigen Gerätetreiber erweiterbare Bibliothek von speziellen, gerätespezifischen Funktionen dar. Fest im ROM sind beim Atari ST nur die Bildschirm-spezifischen Befehle implementiert. Haben Sie jedoch noch einige weitere Gerätetreiber installiert, so werden Sie auch einige spezielle ESCAPES zum Betrieb dieser Geräte zur Verfügung gestellt bekommen. Wir werden also im Kapitel über Metafiles etc. noch einmal auf diese Funktionen zurückkommen. Im folgenden sei eine kurze, übersichtliche Zusammenstellung aller Bildschirm-spezifischen ESCAPES angeführt:

INQUIRE ADDRESSABLE ALPHA CHARACTER CELLS
(5/1 VDI)
EXIT ALPHA MODE (5/2 VDI)
ENTER ALPHA MODE (5/3 VDI)
ALPHA CURSOR UP (5/4 VDI)
ALPHA CURSOR DOWN (5/5 VDI)
ALPHA CURSOR RIGHT (5/6 VDI)
ALPHA CURSOR LEFT (5/7 VDI)
HOME ALPHA CURSOR (5/8 VDI)
ERASE TO END OF ALPHA SCREEN (5/9 VDI)
ERASE TO END OF ALPHA LINE (5/10 VDI)
DIRECT ALPHA CURSOR ADDRESS (5/11 VDI)
OUTPUT CURSOR ADDRESSABLE ALPHA TEXT (5/12 VDI)
REVERSE VIDEO ON (5/13 VDI)
REVERSE VIDEO OFF (5/14 VDI)
INQUIRE CURRENT ALPHA CURSOR ADDRESS (5/15 VDI)
INQUIRE TABLET STATUS (5/16 VDI)
HARDCOPY (5/17 VDI)

PLACE GRAPHIC CURSOR AT LOCATION (5/18 VDI)
REMOVE LAST GRAPHIC CURSOR (5/19 VDI)
SET VIDEO LINE OFFSET (5/101 VDI)
SET FONT (5/102 VDI)

Lassen Sie sich bitte nicht von der Menge der insgesamt 21 Funktionen beeindrucken. Wenn Sie das Ganze ein wenig genauer anschauen, dann werden Sie bemerken, wie wenig an den verschiedenen Befehlen jeweils neu zu verstehen ist. Sie können also einen Textcursor (ALPHA CUSOR genannt) in Form eines kleinen buchstabengroßen Rechteckes auf den Bildschirm bringen, ihn an eine beliebige Bildschirmposition setzen, spalten- und zeilenweise bewegen und Text normal oder revers ab dieser Stelle ausgeben. Sie können ihn frei für den Benutzer sichtbar auf dem Bildschirm plazieren oder im Verborgenen halten. Zudem existiert noch ein sogenannter Grafik-Cursor, der normalerweise die Form des Mauszeigers annimmt (ist allerdings nicht durch die Maus bewegbar, das kann aber natürlich quasi "zu Fuß" programmiert werden) und an jeder beliebigen Bildschirmkoordinate zu plazieren ist.

Eine Sonderstellung nehmen die beiden letzten ESCAPES mit den Nummern 101 und 102 ein. Wir werden später sehen, daß es möglich ist, eigene ESCAPES zu definieren, deren ESCAPE-ID dann bei 101 beginnt. Diese beiden ESCAPES sind solche nachträglich (allerdings vom Betriebssystem) implementierten ESCAPES. Sie übernehmen auch sehr spezielle Aufgaben.

Mit der Nummer 101 kann ein Bildschirm-Offset angegeben werden, also ein Wert in Bytes, der zu der aktuellen Bildschirmadresse addiert wird, bevor das Betriebssystem ein Zeichen ausgibt. Die Grafikausgabe ist davon allerdings nicht betroffen. Auf diese Weise können Sie z.B. einen oberen Bildschirmteil quasi als Textfenster reservieren. Ihre Zeile und Spalte 0 (Homeposition) beginnt dann erst ein Stück tiefer. Sie sollten aber bei der Verwendung dieses Befehles Vorsicht walten lassen, da für das Betriebssystem Ihr Monitor trotzdem noch 25 Zeilen besitzt und Ihr Cursor auf einmal nach unten hin verschwinden könnte.

Funktion Nummer 102 ist sehr nützlich bei der Implementierung von neuen Zeichensätzen, wenn man den (später vorgestellten) Befehl FONT LOAD umgehen will, der, wie wir im nächsten Kapitel sehen werden, einige Probleme mit sich bringt. Um Ihnen eine kleine Vorstellung über die Möglichkeiten der Bildschirm-ESCAPES zu geben, sei hier ein kompletter Bildschirmeditor vorgestellt, der einzig und allein mit diesen ESCAPE-Funktionen des VDI operiert. Da das Programm wieder in GFA-BASIC geschrieben wurde, mußten wir uns zunächst natürlich eine kleine Library schreiben, die im hinteren Teil des folgenden Programms zu finden ist. Die C-Freunde werden sicher stöhnen,

denn Sie haben es sichtlich einfacher. Sie brauchen praktisch nur den ersten Teil dieses Programms. Die Library entfällt, da sie direkt beim Link-Vorgang mit eingebunden wird.

```

' *****
' **
' ** Die Escapes **
' **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
' Workstation-GeräteKennziffer:
handle%=DPEEK(VDIBASE+40)
'
' Cursor einschalten:
@v_enter_cur(handle%)
'
' maximale Cursorposition erfragen:
@vq_chcells(handle%,*rows,*columns)
' Cursor auf Bildschirmmitte positionieren:
@vs_curaddress(handle%,rows/2,columns/2)
'
f11%=1 !RVS-Flag
f12%=1 !Modus-Flag
REPEAT
  a%=INP(2) !Auf Zeichen warten
  IF a%>127 !druckbares Zeichen?
    a%=a% AND 127 !nein
    IF a%=&H48 !Cursor auf?
      @v_curup(handle%)
    ENDIF
    IF a%=&H50 !Cursor ab?
      @v_curdown(handle%)
    ENDIF
    IF a%=&H4B !Cursor links?
      @v_curleft(handle%)
    ENDIF
    IF a%=&H4D !Cursor rechts?
      @v_curreight(handle%)
    ENDIF
    IF a%=&H47 !clr/home?
      @v_curhome(handle%) !Cursor in Home-Position
    ENDIF
    IF a%=&H52 !Insert?
      f11%=ABS(f11%-1) !Flag zwischen 0 und 1 wechseln
      IF f11%=1
        @v_rvoff(handle%) !Invers-Modus aus
      ELSE
        @v_rvon(handle%) !Invers-Modus ein
      
```

```
ENDIF
ENDIF
IF a%=&H61 !Undo?
    fl2%=ABS(fl2%-1) !Flag zwischen 0 und 1 wechseln
    IF fl2%=1
        @v_enter_cur(handle%)!Alpha-Cursor-Modus ein
    ELSE
        @v_exit_cur(handle%) !Grafik-Cursor-Modus ein
    ENDIF
ENDIF
ELSE
    @v_curtext(handle%,CHR$(a%)) !Zeichen an Cursorposition
ENDIF
UNTIL a%=&H1B !bis ESC
'
EDIT
'
' ESCAPE-FUNKTIONEN:
PROCEDURE v_escape(handle%,id%,sintin%,sptsin%)
    DPOKE CONTRL+2,sptsin%
    DPOKE CONTRL+6,sintin%
    DPOKE CONTRL+10,id%
    DPOKE CONTRL+12,handle%
    VDISYS 5
RETURN
'
' INQUIRE ADDRESSABLE ALPHA CHARACTER CELLS
PROCEDURE vq_chcells(handle%,p_rows%,p_columns%)
    @v_escape(handle%,1,0,0)
    *p_rows%=DPEEK(INTOUT)
    *p_columns%=DPEEK(INTOUT+2)
RETURN
'
' EXIT ALPHA MODE
PROCEDURE v_exit_cur(handle%)
    @v_escape(handle%,2,0,0)
RETURN
'
' ENTER ALPHA MODE
PROCEDURE v_enter_cur(handle%)
    @v_escape(handle%,3,0,0)
RETURN
'
' ALPHA CURSOR UP
PROCEDURE v_curup(handle%)
    @v_escape(handle%,4,0,0)
RETURN
'
' ALPHA CURSOR DOWN
PROCEDURE v_curdown(handle%)
    @v_escape(handle%,5,0,0)
RETURN
```



```
'  
' ALPHA CURSOR RIGHT  
PROCEDURE v_curright(handle%)  
    @v_escape(handle%,6,0,0)  
RETURN  
'  
' ALPHA CURSOR LEFT  
PROCEDURE v_curleft(handle%)  
    @v_escape(handle%,7,0,0)  
RETURN  
'  
' HOME ALPHA CURSOR  
PROCEDURE v_curhome(handle%)  
    @v_escape(handle%,8,0,0)  
RETURN  
'  
' ERASE TO END OF ALPHA SCREEN  
PROCEDURE v_eeos(handle%)  
    @v_escape(handle%,9,0,0)  
RETURN  
'  
' ERASE TO END OF ALPHA LINE  
PROCEDURE v_eeol(handle%)  
    @v_escape(handle%,10,0,0)  
RETURN  
'  
' DIRECT ALPHA CURSOR ADDRESS  
PROCEDURE vs_curaddress(handle%,row%,column%)  
    DPOKE INTIN,row%  
    DPOKE INTIN+2,column%  
    @v_escape(handle%,11,2,0)  
RETURN  
'  
' OUTPUT CURSOR ADDRESSABLE ALPHA TEXT  
PROCEDURE v_curtext(handle%,string$)  
    string$=string$+CHR$(0)  
    FOR xi%=0 TO LEN(string$)-1  
        DPOKE INTIN+2*xi%,ASC(MID$(string$,xi%+1,1))  
    NEXT xi%  
    @v_escape(handle%,12,LEN(string$),0)  
RETURN  
'  
' REVERSE VIDEO ON  
PROCEDURE v_rvon(handle%)  
    @v_escape(handle%,13,0,0)  
RETURN  
'  
' REVERSE VIDEO OFF  
PROCEDURE v_rvoff(handle%)  
    @v_escape(handle%,14,0,0)  
RETURN  
'  
' INQUIRE CURRENT ALPHA CURSOR ADDRESS
```

```
PROCEDURE vq_curaddress(handle%,p_row,p_column)
  @v_escape(handle%,15,0,0)
  *p_row=DPEEK(INTOUT)
  *p_column=DPEEK(INTOUT+2)
RETURN
'
' INQUIRE TABLET STATUS
PROCEDURE vq_tabstatus(handle%)
  @v_escape(handle%,16,0,0)
  status=DPEEK(INTOUT)
RETURN
'
' HARD COPY
PROCEDURE v_hardcopy(handle%)
  @v_escape(handle%,17,0,0)
RETURN
'
' PLACE GRAPHIC CURSOR AT LOCATION
PROCEDURE v_dspcur(handle%,x%,y%)
  DPOKE PTSIN,x%
  DPOKE PTSIN+2,y%
  @v_escape(handle%,18,0,1)
RETURN
'
' REMOVE LAST GRAPHIC CURSOR
PROCEDURE v_rmcure(handle%)
  @v_escape(handle%,19,0,0)
RETURN
'
' SET VIDEO LINE OFFSET
PROCEDURE v_vloffset(handle%,offset%)
  DPOKE INTIN,offset%
  @v_escape(handle%,101,1,0)
RETURN
'
' SET FONT
PROCEDURE v_sfont(handle%,fontaddress)
  xi%=INT(fontaddress/65536)
  DPOKE INTIN,xi%
  DPOKE INTIN+2,fontaddress-xi%*65536
  @v_escape(handle%,102,2,0)
RETURN
```

Sie sehen, wie einfach, ja geradezu primitiv dieses Programm wäre, wenn wir in GFA-BASIC die Library nicht hinzufügen müßten. In C nimmt sich das Programm fast als kleine "Spielerei" am Rande aus. Wenn Sie in C programmieren, dürfte es wohl kaum ein Problem darstellen, dieses BASIC-Programm kurz umzuschreiben. (Wie gesagt, den ganzen Bibliotheksanhang brauchen Sie nicht mehr.)

Der vorgestellte Editor beschränkt sich hier der Einfachheit halber auf den Bildschirm. Sie könnten natürlich sehr einfach an verschiedenen Stellen Ergänzungen vornehmen, und schon hätten Sie ein kleines Textverarbeitungsprogramm. Doch seien Sie nicht zu voreilig, wir werden Ihnen nämlich noch später einen sehr viel komfortableren Editor liefern, der den Vergleich mit einem Textverarbeitungsprogramm nicht zu scheuen braucht (Direktformatierung auf dem Bildschirm etc.). Doch kurz sollten wir das obige Programm einmal beschreiben.

Fangen wir bei den Bibliotheks-Funktionen an. Die verschiedenen Funktionen sind prinzipiell genauso implementiert, wie wir das von den vielen bisher abgedruckten Beispielprogrammen her kennen. Um allerdings die Schreibarbeit und die Programmlänge ein wenig einzudämmen, haben wir noch eine kleine Funktion namens "v_escape" definiert, die die Beschickung des Kontroll-Arrays und den Funktionsaufruf unternimmt. Das eigentliche Programm beginnt zunächst mit der Feststellung der Anzahl der Zeilen und Spalten. Diese Daten werden dann dafür verwendet, den Cursor in der Bildschirmmitte zu verhaften. Was dann kommt, ist nicht schwer zu verstehen. Durch die Zeile

```
a% = INP(2)
```

wartet das Programm zunächst auf einen Tastendruck. Dann wird überprüft, ob das eingegebene Zeichen ein druckbares Zeichen darstellt, oder ob es sich z.B. um Cursorfunktionen handelt. Im ersten Fall wird das Zeichen (hinter ELSE) mit der Funktion "v_curtex" an der momentanen Bildschirmposition ausgegeben. Im anderen Fall wird die Taste auf die verschiedenen möglichen Kontroll-Funktionen überprüft:

crsr auf	Cursor nach oben bewegen (v_curup)
crsr ab	Cursor nach unten bewegen (v_curdown)
crsr li	Cursor nach links bewegen (v_curleft)
crsr re	Cursor nach rechts bewegen (v_curright)
Clr/Home	Cursor nach Homeposition bewegen (v_curhome)
Insert	Wechsel zwischen Invers-Modus ein/aus (v_rvon/v_rvoff)
Undo	Wechsel zwischen Alpha Cursor ein/aus (v_enter_cur/v_exit_cur)
Esc	Programmende

Wie Sie sehen, sind alle Funktionen auch dann aktiv und wählbar, wenn der ALPHA-Cursor nicht eingeschaltet ist. Für die ESCAPE-Funktionen gibt es keine systematische Implementierung in GFA-BASIC (außer PRINT AT()).

Die oben beschriebenen VDI-Funktionen werden Sie - wie auch mich - nicht vollständig befriedigen. Unter "Text in Grafik" stelle auch ich mir weitaus mehr vor, z.B. eine freie Positionierung von Text an jeder beliebigen Bildschirmkoordinate. Bisher ist es nur möglich, Text zeilen- und spaltenweise auf den Bildschirm zu bringen. Mit dem VDI-Befehl TEXT (8 VDI) jedoch werden Sie dazu in der Lage sein.

Was es mit dem hier so geheimnisvollen Aktionspunkt und dem Zauberwort Alignment auf sich hat, werden wir später noch erfahren. Hier interessiert nur, daß der diesem Befehl übergebene Text ab den spezifizierten Koordinaten auf dem Bildschirm ausgegeben werden kann. Dabei werden alle später noch vorgestellten Text-Attribute (also keine Linien- oder Flächenattribute) sowie der eingestellte Zeichenmodus und ein eventuelles Clip-Fenster berücksichtigt.

Wichtig für BASIC- und Assembler-Programmierer ist die Übergabe des Textstrings. Genauso wie bei der entsprechenden ESCAPE-Funktion (s.o.) wird hier jedes Zeichen einzeln in ein INTIN-Array-Element gePOKEt. Damit werden für ein Zeichen aber genau 16 Bits (statt der üblichen acht) verbraucht. Ein String kann also nicht einfach 1:1 in das INTIN-Array übertragen werden, er muß vielmehr sozusagen "gedehnt" werden.

Auch die Farbe des Textes wollen wir an dieser Stelle schon ändern können. Sie ist unabhängig von der Farbe der Linien- oder Flächen-Objekte und wird mit dem folgenden Befehl SET GRAPHIC TEXT COLOR INDEX (22 VDI) eingestellt.

Natürlich sollten wir uns das Ganze einmal aus der Programmierersicht anschauen. GFA-BASIC bietet zwar auch eine Implementierung des Text-Befehles, sie umfaßt allerdings noch einige Dinge mehr, die wir erst noch kennenlernen, und ermöglicht nicht alle tatsächlich möglichen Textfunktionen. Aus diesen Gründen verschieben wir das noch ein wenig und realisieren unsere Vorhaben zunächst ausschließlich mit den VDI-Funktionen, was C- und Assembler-Programmierer erfreuen wird:

```
' *****  
' **  
' ** Grafiktexte **  
' **  
' *****  
'  
' Bildschirmauflösung:  
xauf1%=DPEEK(&H2686)  
yauf1%=DPEEK(&H2688)  
'  
handle%=DPEEK(VDIBASE+40)  
'
```

```

@vst_color(handle%,1)
,
y1%=0
FOR gm%=1 TO 4
    GRAPHMODE gm%           !Grafikmodus setzen
    y1%=y1%+30
    y%=y1%
    x%=0
    FOR i%=0 TO 40 STEP 2    !Abstandsschleife
        x%=x%+i%
        y%=y%+i%/3
        @v_gtext(handle%,x%,y%,"Ellie") !Textausgabe
    NEXT i%
NEXT gm%
,
VOID INP(2)
CLS
,
FOR gm%=1 TO 4
    GRAPHMODE gm%
    zx%=(gm%-1)*xauf1%/6
    zy%=(gm%-1)*yauf1%/6
    FOR i%=0 TO 200
        @v_gtext(handle%,RND*xauf1%/3+zx%,RND*yauf1%/3+zy%,"Pirelli")
    NEXT i%
NEXT gm%
,
VOID INP(2)
EDIT
,
,
' TEXT
PROCEDURE v_gtext(handle%,x%,y%,string$)
    string$=string$+CHR$(0)      !0 anhängen
    DPOKE CONTRL+2,1
    DPOKE CONTRL+6,LEN(string$)
    DPOKE CONTRL+12,handle%
    FOR xi%=0 TO LEN(string$)-1
        DPOKE INTIN+2*xi%,ASC(MID$(string$,xi%+1,1))
    NEXT xi%
    DPOKE PTSIN,x%
    DPOKE PTSIN+2,y%
    VDISYS 8
RETURN
,
' SET GRAPHIC TEXT COLOR INDEX
PROCEDURE vst_color(handle%,color_index%)
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,color_index%
    VDISYS 22
RETURN

```

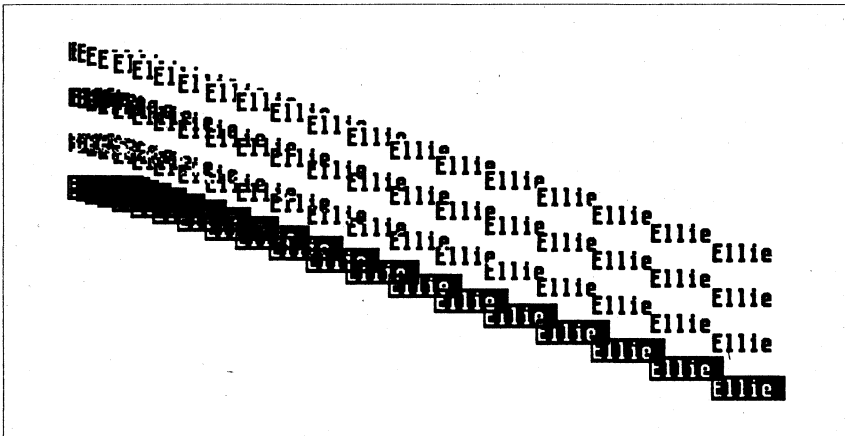


Abb. 15.28: Grafiktexte 1

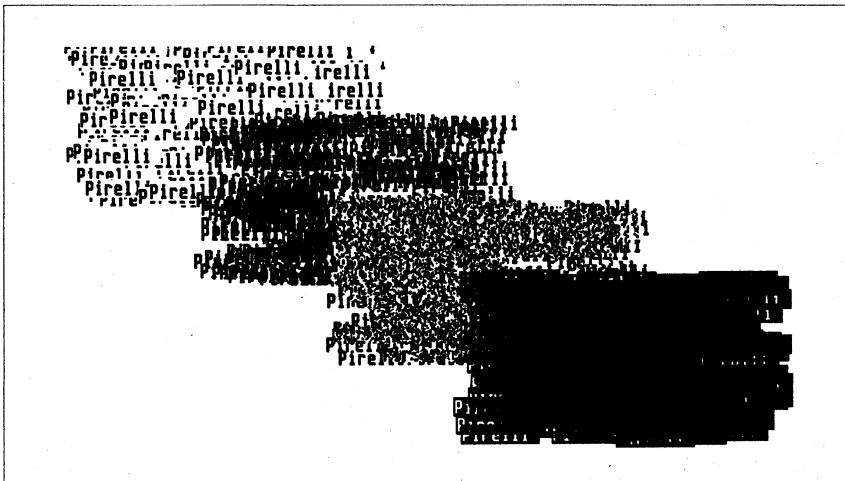


Abb. 15.29: Grafiktexte 2

Das zweigeteilte Programm soll Ihnen den Einfluß der verschiedenen Zeichenmodi auf die Textausgabe veranschaulichen. Wie das Programm dies unternimmt, ist an dieser Stelle nicht so wichtig. Wichtig sind die Erkenntnisse über die Beschaffenheit einzelner Buchstaben.

Jeder Buchstabe besteht nämlich aus einer oben bereits erwähnten rechteckigen Matrix (z.B. 16x8 oder 8x8 Punkte), in der Punkte gesetzt oder gelöscht sein können, um die Buchstabenform herauszuarbeiten. Im Replace-Modus wird nun die gesamte Matrix, also das ganze Rechteck mit gesetzten und auch ungesetzten, Punkten "ohne Rücksicht auf Verluste" auf den Bildschirm gebracht. Teile des Bildschirms werden also hierdurch auch mit nicht gesetzten Punkten überschrieben. Wir kennen das Prinzip ja bereits von den Füllmustern. Im Transparent-Modus jedoch werden nur die in der Matrix tatsächlich gesetzten Punkte auch geplottet, alle anderen werden ignoriert, die Matrix ist quasi durchsichtig.

Auch im XOR-Modus werden - bedingt durch die Verknüpfungsart - nur die gesetzten Punkte berücksichtigt. Gemäß den Eigenschaften dieses Modus' werden allerdings auch die Punkte gelöscht (bzw. erhalten eine andere Farbe), die vor Aufbringen des Textes bereits gesetzt waren und laut Zeichenmaske nun noch einmal gesetzt werden sollten. Im Invers-Transparent-Modus sehen Sie die gesamte Matrix eines Zeichens, da diese hier zunächst invertiert wird, um dann im Transparent-Modus auf den Bildschirm gebracht zu werden. Daß wir die Form und Größe der Matrix eines Buchstabens noch ändern können (Kursiv-Schrift, Schriftgröße), werden Sie im nächsten Kapitel miterleben.

15.7.2 Die Fonts und Schriftarten

Einfachen Text auf den Bildschirm (oder natürlich mit dem entsprechenden Treiber auch auf andere Geräte) zu bringen, ist für uns nun kein Brief mit sieben Siegeln mehr. Das allerdings konnten auch vor dem Atari ST bereits andere Computer. Jetzt wird es schon interessanter. Jetzt kommen Funktionen auf uns zu, die dem Ganzen einen professionellen Anstrich geben.

Der Atari ST kann jeden Buchstaben verkleinern und auch vergrößern. Das hört sich ja schon einmal gut an, oder? Für diesen Größenwechsel besitzt er zwei VDI-Funktionen. In der einen geben Sie einfach die Höhe der gewünschten Schrift in Bildschirmpixel an, und schon wird jeder Buchstabe, den Sie fortan z.B. mit TEXT, aber auch mit den ESCAPE-Funktionen ausgeben, diese gewünschte Höhe einnehmen. Beachten Sie auch hier, daß damit Text in den verschiedenen Auflösungen unterschiedliche Größe erhält. Zwar wählt Ihr Rechner zu Beginn seiner Arbeit - wie am Anfang des letzten Kapitels beschrieben - einen passenden Zeichensatz aus, diese Wahl setzen Sie mit der Größenangabe allerdings zwangsläufig wieder außer Kraft. Die mögliche Zahl der Zeichen pro Zeile und auch die Zahl der Zeilen pro Bildschirm ändert sich somit mit Ihrer Manipulation.

Es gibt aber noch einen weiteren Befehl, der es Ihnen ebenfalls erlaubt, die Höhe eines Zeichens festzulegen. Hierbei geben Sie diese dann jedoch in einer anderen Maßeinheit an: Punkt. Punkt ist eine Längeneinheit aus der Satztechnik:

$$1 \text{ Punkt} = 1/216 \text{ Inch} = 0,118 \text{ mm}$$

Ein Punkt sind also ca. 0,118 Millimeter, was dann exakt die Größe des Zeichens auf dem Bildschirm, Drucker o.ä. angibt. Gleichzeitig bezieht sich diese Höhenangabe nicht auf das Zeichen selbst, sondern auf die Höhe der oben bereits besprochenen Zeichenmatrix, die ja meist größer ist als das Zeichen selbst. Damit wäre diese Angabe eigentlich - wie bei den Satzmaschinen Konvention - ein Maß für den Zeilenabstand eines Textes.

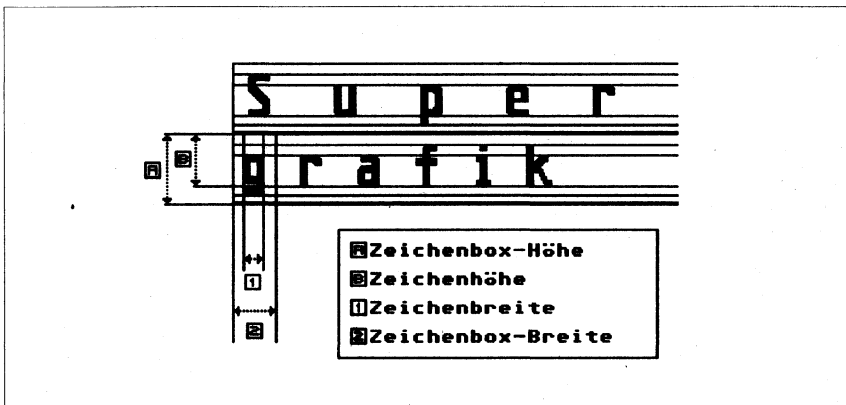


Abb. 15.30: Zeichen-/Cell-Höhe in Pixel/Punkt

Sie sollten Ihre Größenangaben möglichst immer in Punkt vornehmen, da sonst nicht garantiert werden kann, daß die Zeichen z.B. auf einem Drucker genauso groß ausfallen werden wie auf dem Bildschirm. Ein Nachteil bei dieser Höhenangabe ist allerdings die Tatsache, daß es pro Zeichensatz nur wenige ganz diskrete Zeichengrößen gibt, während bei der Pixel-Angabe bis zu einer bestimmten Obergrenze aus jedem Wert auch eine andere Zeichengröße resultiert.

Da Sie nur die gewünschte Höhe des Zeichens angeben können, andererseits aber aus der Zeichenhöhe auch die Zeichenbreite resultiert, die proportional zur Höhe zu- oder abnimmt, liefern Ihnen die beiden Funktionen, die wir nun

endlich vorstellen wollen, noch einmal sowohl die tatsächlich gewählten Zeichenhöhen und -breiten in Pixel als auch die tatsächlich gewählten Höhen und Breiten der Zeichenmatrix:

```
SET CHARACTER HEIGHT, ABSOLUTE MODE (12 VDI)
SET CHARACTER CELL HEIGHT, POINTS MODE (107 VDI)
```

Diese Angaben sind schließlich notwendig, um zu ermitteln, wann ein String noch in eine Zeile paßt etc. Geben Sie die Höhe in Punkt an, so wird Ihnen zusätzlich noch die tatsächlich gewählte Punkt-Höhe zurückgegeben, da ja - wie erwähnt - nicht alle Höhen zulässig sind.

Doch VDI beläst es nicht einfach bei diesen Zeichenmanipulationen. Es bietet Ihnen noch mehr. Ohne einen zusätzlichen Zeichensatz laden zu müssen o.ä., kann es Zeichen auf verschiedene Arten verzerren, punktieren und so weiter und so fort. Dies wird alles mit Hilfe der Funktion SET GRAPHIC TEXT SPECIAL EFFECTS (106 VDI) unternommen.

Die verschiedenen Schrifteffekte werden dabei aus einer einzigen Zeichenmatrix sozusagen errechnet. Fettschrift entsteht dabei sozusagen durch "Doppeldruck"; ein und dasselbe Zeichen wird praktisch noch einmal, nur etwas horizontal versetzt gezeichnet, Kursivschrift entsteht durch das Schrägstellen der gesamten Zeichenbox usw. Die angegebene schattierte Schrift wird allerdings vom momentanen Bildschirm-Treiber nicht unterstützt.

Alle fünf Schrifteffekte können miteinander kombiniert werden, Sie können also auch kursive, unterstrichene Fettschrift anwählen. Dies erreichen Sie durch das Setzen bzw. Löschen des jeweils zuständigen Bits im Effekt-Wert. Haben Sie kein Bit gesetzt, dann erscheint Normalschrift. Damit haben Sie die Möglichkeit, summa summarum 32 verschiedene Schriftarten von einem Zeichensatz anzuwählen. Zusammen mit den vielen verschiedenen Schriftgrößen haben Sie eine schier unübersehbare Fülle von Variationsmöglichkeiten eines einzigen Zeichensatzes. Um hier noch einigermaßen Überblick zu erlangen, starten Sie am besten das folgende Programm, mit dem Sie direkt am Bildschirm die verschiedenen Einstellungen vornehmen können:

```
' *****
' **                               **
' ** Unser erster Grafik-Text-Editor **
' **                               **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
```

```
handle%=DPEEK(VDIBASE+40)
,
,
a$="" !Textspeicher auf leer
a1$="" !Texthilfsspeicher
x1%=-1 !Koordinatenhilfsspeicher
y1%=-1
height%=10 !Zeichenhöhe
effect%=0 !Text-Effektnummer
farbe%=1 !Text-Farbe
,
REPEAT
  MOUSE x%,y%,k% !Mausabfrage
  c$=INKEY$ !Tastaturabfrage
  ,
  IF k%=1 !linke Maustaste?
    @v gtext(handle%,x%,y%,a$) !ja -> Text fest-schreiben
  ENDIF
  ,
  IF c$<>" !Tastendruck?
    IF ASC(c$)=0 !Sondertaste?
      c%=ASC(RIGHT$(c$,1)) !Scan-Code berechnen
      ,
      IF c%=&H47 !Clr/Home?
        a$="" !ja -> Textspeicher löschen
      ENDIF
      IF c%=&H48 !Cursor auf?
        height%=height%+1 !ja -> Zeichenhöhe erhöhen
      ENDIF
      IF c%=&H50 !Cursor ab?
        height%=height%-1 !ja -> Zeichenhöhe verringern
      ENDIF
      IF c%=&H4B !Cursor links?
        effect%=effect%-1 !ja -> Effektnummer verringern
      ENDIF
      IF c%=&H4D !Cursor rechts?
        effect%=effect%+1 !ja -> Effektnummer erhöhen
      ENDIF
      IF c%=&H61 !Undo?
        farbe%=farbe%+1 !ja -> Farbnummer erhöhen
      ENDIF
      IF c%=&H62 !Help?
        farbe%=farbe%-1 !ja -> Farbnummer verringern
      ENDIF
      ,
    ELSE
      IF ASC(c$)=&H8 !Backspace?
        a$=LEFT$(a$,LEN(a$)-1) !ja -> letztes Zeichen löschen
      ELSE
        a$=a$+c$ !nein -> Zeichen anhängen
      ENDIF
    ENDIF
  ENDIF
ENDIF
```

```

    @newtext(x%,y%,a$)          !Text erneuern
ENDIF
,
IF x%<>x1% OR y%<>y1%          !Koordinatenänderung?
    @newtext(x%,y%,a$)        !ja -> Text erneuern
ENDIF
,
UNTIL c$=CHR$(&H1B)            !bis ESC
,
EDIT                            !Ende
,
,
' Text neu zeichnen:
PROCEDURE newtext(x%,y%,text$)
    GRAPHMODE 3                !XOR-Modus ein
    @v_gtext(handle%,x1%,y1%,a1$) !Alten Text löschen
,
' Neue Textparameter setzen:
VSYNC
@vst_height(handle%,height%,*char_width,*char_height,*cell_width,*cell_height)
@vst_color(handle%,farbe%)
@vst_effects(handle%,effect%)
@v_gtext(handle%,x%,y%,a$)      !Neuen Text zeichnen
x1%=x%
y1%=y%
a1%=a$
,
' Informationszeilen ausgeben:
VSYNC
PRINT AT(1,23);SPC(39)
PRINT AT(1,23);"Farbe:      ";farbe%
VSYNC
PRINT AT(1,24);SPC(39)
PRINT AT(1,24);"x-Pos.:    ";x%;AT(17,24);" / y-Pos.: ";y%
VSYNC
PRINT AT(1,25);SPC(39);
PRINT AT(1,25);"Zeichenhöhe: ";height%;AT(17,25);" / Effekt: ";effect%;
PRINT " (";BIN$(effect%);");";
RETURN
,
,
' VDI-Library:
,
' TEXT
PROCEDURE v_gtext(handle%,x%,y%,string$)
    string$=string$+CHR$(0)      !0 anhängen
    DPOKE CONTRL+2,1
    DPOKE CONTRL+6,LEN(string$)
    DPOKE CONTRL+12,handle%
    FOR xi%=0 TO LEN(string$)-1
        DPOKE INTIN+2*xi%,ASC(MID$(string$,xi%+1,1))
    NEXT xi%

```

```
DPOKE PTSIN,x%
DPOKE PTSIN+2,y%
VDISYS 8
RETURN
'
' SET CHARACTER HEIGHT, ABSOLUTE MODE
PROCEDURE
vst_height(handle%,height%,p_char_width%,p_char_height%,p_cell_width%,p_cell_height%)
DPOKE CONTRL+2,1
DPOKE CONTRL+12,handle%
DPOKE PTSIN,0
DPOKE PTSIN+2,height%
VDISYS 12
*p_char_width%=DPEEK(PTSOUT)
*p_char_height%=DPEEK(PTSOUT+2)
*p_cell_width%=DPEEK(PTSOUT+4)
*p_cell_height%=DPEEK(PTSOUT+6)
RETURN
'
' SET CHARACTER CELL HEIGHT, POINTS MODE
PROCEDURE
vst_point(handle%,point%,p_char_width%,p_char_height%,p_cell_width%,p_cell_height%)
DPOKE CONTRL+6,1
DPOKE CONTRL+12,handle%
DPOKE INTIN,point%
VDISYS 107
*p_char_width%=DPEEK(PTSOUT)
*p_char_height%=DPEEK(PTSOUT+2)
*p_cell_width%=DPEEK(PTSOUT+4)
*p_cell_height%=DPEEK(PTSOUT+6)
RETURN
'
' SET GRAPHIC TEXT COLOR INDEX
PROCEDURE vst_color(handle%,color_index%)
DPOKE CONTRL+6,1
DPOKE CONTRL+12,handle%
DPOKE INTIN,color_index%
VDISYS 22
RETURN
'
' SET GRAPHIC TEXT SPECIAL EFFECTS
PROCEDURE vst_effects(handle%,effect%)
DPOKE CONTRL+6,1
DPOKE CONTRL+12,handle%
DPOKE INTIN,effect%
VDISYS 106
RETURN
```

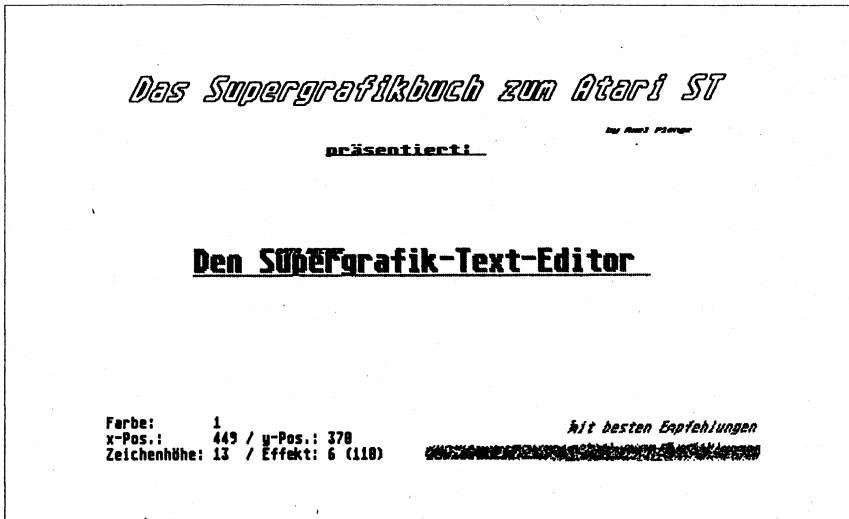


Abb. 15.31: Grafik-Text-Editor

Der hochtrabende Name "Grafik-Text-Editor" ist vielleicht ein wenig übertrieben. Man sollte ihn vielleicht für ein Programm reservieren, das Sie ein wenig weiter unten zu sehen bekommen.

Doch wollen wir uns kurz mit diesem BASIC-Programm beschäftigen. Auf dem Bildschirm erscheint zunächst einmal rein gar nichts, bis auf drei Statuszeilen am unteren Bildschirmrand. Erst wenn Sie ein paar Zeichen angeben, dann tut sich etwas. Der eingegebene Text nämlich wird auf dem Bildschirm ausgegeben. Daran allein ist eigentlich noch nicht viel zu staunen. Auch die Tatsache, daß Sie diesen Text mit der Maus auf dem Bildschirm frei positionieren können, macht noch keinen Staat.

Interessant wird es erst, wenn Sie eine der Steuertasten betätigen. Dann nämlich können Sie die aktuelle Schriftgröße und die verschiedenen Texteffekte manipulieren. Die gewählten Parameter werden dann sofort, stets auf dem neuesten Stand, in den drei Statuszeilen ausgegeben. Drücken Sie einmal die linke Maustaste, so wird der aktuelle Text fest auf den Bildschirm geschrieben. Auf diese Weise können Sie Ihr eigenes, ganz persönliches Text-Arrangement kreieren und z.B. per Hardcopy auf dem Drucker ausgeben.

Mit den folgenden Tasten können sie die verschiedenen Funktionen des Programms manipulieren:

Mausbewegung	- Freie Text-Positionierung auf dem Bildschirm
Mausknopf li	- Text an aktuelle Position schreiben
crsr auf	- Zeichenhöhe in Pixel steigern
crsr ab	- Zeichenhöhe in Pixel verringern
crsr rechts	- Text-Effektnummer erhöhen
crsr links	- Text-Effektnummer verringern
Undo	- Farbindex erhöhen
Help	- Farbindex verringern
Clr/Home	- Aktuellen Text löschen
Backspace	- Ein Zeichen im akt. Text löschen
Esc	- Programmende

Der Programmaufbau entspricht dem einiger, bereits vorgestellter Programme und sollte auch aufgrund der vollständigen Dokumentation nicht besonders schwer zu verstehen sein. Aus diesem Grunde verzichten wir hier auf eine ermüdende Programmbeschreibung. Der Befehl VSYNC allerdings könnte noch unbekannt sein. Er wartet auf einen sogenannten Vertical Blank (VBL), also einen senkrechten Strahlrücklauf, d.h. auf einen neuen Bildaufbau auf dem Monitor. Dies wird zur möglichst flackerfreien Darstellung verwendet.

Interessant ist auch hier, wie immer, die Realisierung der VDI-Funktionen in BASIC. Die hierzu definierten Library-Prozeduren stellen eine gute Ergänzung zu den Funktionsbeschreibungen dar und sollten unbedingt einmal betrachtet werden.

Wir haben bereits oben des öfteren von "Zeichensätzen" gesprochen, ohne den Begriff näher erläutert zu haben. Unter Zeichensatz oder in Zukunft auch Font genannt, verstehen wir die Gesamtheit aller Zeichenmatrizen für jeweils einen vollständigen ASCII-Wertebereich (Alphabet plus Ziffern plus Sonderzeichen etc.). In einem Zeichensatz sind also die Matrizen für alle Buchstaben eines Alphabets etc. gespeichert. Beim Atari ST können die Matrizen eines solchen Fonts nun beliebig groß und beliebig gestaltet sein, auch die einzelnen Zeichen eines Fonts können z.B. für Proportionalschrift beliebige Ausdehnung besitzen. Mehrere verschiedene Fonts können zu kombinierten Fonts erklärt werden. Dies findet vor allem dann Anwendung, wenn es von einer Schrift verschiedene Punkt-Größen geben soll. Das Betriebssystem wählt nämlich von solchen kombinierten Fonts automatisch stets denjenigen aus, der die richtige Größe für die gewählte Zeichenhöhe besitzt. Kombinierte Fonts sollten also nicht völlig verschiedene Schriftarten darstellen, sondern vielmehr unterschiedlich große Ausführungen ein und derselben Schrift. Will man dagegen

eine gänzlich verschiedene Schrift einführen, so empfiehlt sich die Schaffung eines neuen Fonts mit vielleicht wieder unterschiedlichen kombinierten Fonts.

Von Haus aus besitzt der Atari ST, wie bereits erwähnt, zwei verschiedene Zeichensätze, einen 16x8 und einen 8x8 Punkte pro Zeichen umfassenden Font. Diese Fonts gelten als kombinierte Fonts im obigen Sinne und werden automatisch für die verschiedenen Schriftgrößen herangezogen. Wie gesagt, schaltet Ihr Atari ST jeweils für die unterschiedlichen Auflösungen auch unterschiedliche Fonts ein.

Einen neuen Font (der Aufbau eines Fonts wird in einem späteren Kapitel erläutert) können Sie z.B. mit der VDI-ESCAPE Funktion Nummer 102 einbinden (s. ESCAPES).

Eine andere Möglichkeit bietet VDI mit der Funktion LOAD FONTS, die einen neuen Zeichensatz von Diskette lädt. Mit dieser Funktion hat es allerdings eine besondere Bewandnis. Sie funktioniert auf dem Atari ST nicht ohne weiteres. Um sie anwenden zu können, bedarf es eines kleinen Zusatzprogramms, des sogenannten GDOS. GDOS verwaltet ebenfalls die verschiedenen Gerätetreiber, falls vorhanden, die mit OPEN WORKSTATION geladen werden können. Ohne GDOS ist auch die Funktion OPEN WORKSTATION (nicht zu verwechseln mit OPEN VIRTUAL SCREEN WORKSTATION) nicht anwählbar. Sie führt dann unmittelbar zum Absturz.

Nun, wie kommt man zu diesem GDOS? Dieses kleine, etwa 9 KByte große Programm wird nicht offiziell vertrieben. Es existiert allerdings bereits mehrere tausendmal und sicher haben auch Sie eine Version irgendwo auf Ihren Disketten. Die Grafik-Programme GEM DRAW und GEM PAINT, die für den Atari eigentlich nur inoffiziell existieren, aber trotzdem in der Programmsammlung fast jedes Atari ST-Besitzers vorhanden sind, verwenden dieses GDOS. Inzwischen gibt es auch einige "legale" Software, zu deren Lieferumfang eine neuere Version des GDOS gehört. Ersatzweise kann man auch da "AMC-GDOS" benutzen oder auch "G PLUS+"

Um GDOS nutzen zu können, müssen Sie es nämlich schon beim Booten laden lassen. Bei diesem Boot-Vorgang werden grundsätzlich (nicht nur bei diesen Disketten) alle Programmdateien in einem eventuell vorhandenen AUTO-Ordner gestartet. Befindet sich nun GDOS im AUTO-Ordner, wird es ebenfalls gestartet und dadurch installiert.

Wollen Sie nun die sonst nicht funktionierenden VDI-Funktionen ebenfalls nutzen, müssen Sie GDOS in Ihren AUTO-Ordner bringen und booten, den Computer also ausschalten und wieder einschalten oder einen Reset machen.

Auf den Programmdisketten zu den beiden Grafikprogrammen befinden sich aber noch verschiedene andere, sehr nützliche Programme. Dies sind ein Druckertreiber namens FX80.SYS mit einem komfortablen Bedienerprogramm namens OUTPUT.PRG (bzw. OUTPUT.APP). Ferner existieren dort ein Metafile-Treiber mit dem Namen META.SYS sowie ein komplett neuer Zeichensatz für Bildschirm und einer für den Drucker (eigentlich mehrere kombinierte):

Bildschirmzeichensätze:

IBMHSS10.FNT
IBMHSS14.FNT
IBMHSS18.FNT
IBMHSS36.FNT

Druckerzeichensätze:

EPSHSS10.FNT
EPSHSS14.FNT
EPSHSS20.FNT
EPSHSS28.FNT
EPSHSS36.FNT

Ein ASCII-File mit dem Namen ASSIGN.SYS, das ebenfalls bei OPEN WORKSTATION geladen wird, liefert GDOS die Informationen, welche Treiber welchen Handle-Nummern mit welchen Zeichensätzen zugeordnet werden sollen. Wir werden diese Dinge später bei der Beschreibung der Ein- und Ausgabemöglichkeiten von Grafik noch beschreiben.

Leider existieren für den Atari ST kaum zusätzliche Zeichensätze. GEM-Implementierungen für den IBM jedoch liefern dort eine erstaunliche Vielfalt von neuen Fonts. Ein Font, der unter GEM auf dem IBM läuft, läuft auch, sofern die Font-Daten auf den Atari ST übertragen werden können, ohne irgendwelche Änderungen auf dem Atari-GEM. Unter Berücksichtigung der Urheberrechte können also IBM-Zeichensätze ohne Probleme auf den Atari übertragen werden.

Die folgenden Ausführungen lassen einen eventuellen Urheberrechtsschutz der aufgeführten Programme und Daten-Files unberücksichtigt. Bevor Sie also irgendetwas mit diesen Ausführungen anfangen wollen, sollten Sie sich zunächst einmal über die Copyright-Bedingungen der einzelnen Programmenteile informieren. Bedenken Sie dabei bitte, daß auch Fonts urheberrechtlich geschütztes Material darstellen.

Um nun einen solchen Zeichensatz auf den Bildschirm zu bringen, verwenden wir einmal die Files auf den Programmdisketten der beiden obigen Grafikprogramme. Hierzu sind notwendig:

- GDOS in einem AUTO-Ordner
- Booten der Diskette mit GDOS
- ASSIGN.SYS
- IBMHSS10.FNT
- IBMHSS14.FNT
- IBMHSS18.FNT
- IBMHSS36.FNT
- EPSHSS10.FNT
- EPSHSS14.FNT
- EPSHSS20.FNT
- EPSHSS28.FNT
- EPSHSS36.FNT
- META.SYS
- FX80.SYS

Die letzten sieben Files sind eigentlich nicht notwendig. Sie stehen aber in ASSIGN.SYS und sollten deshalb auch vorhanden sein. Sie können die entsprechenden Einträge in ASSIGN.SYS aber auch löschen. Im folgenden wird vorausgesetzt, daß diese Files alle auf der aktuellen Disk stehen.

In einem Programm, das nun einen neuen Zeichensatz laden und verwenden möchte, muß zunächst einmal die Arbeitsstation wie gewohnt geöffnet werden (OPEN VIRTUAL SCREEN WORKSTATION). Dann kann der Zeichensatz mit LOAD FONTS geladen und eingebunden werden. Mit der Funktion SET TEXT FACE wird dieser neue Zeichensatz angewählt. Bevor wir die Applikation beenden, müssen wir den Zeichensatz durch die Funktion UNLOAD FONTS wieder freigeben (sonst Systemabsturz!). So einfach ist das!

Die notwendigen VDI-Befehle dazu heißen:

```
LOAD FONTS (119 VDI)
UNLOAD FONTS (120 VDI)
SET TEXT FACE (21 VDI)
```

Leider funktionieren die nachträglich implementierten VDI-Befehle in GFA-BASIC nicht mehr, sobald GDOS gebootet wurde, das ja notwendig ist, um die Zeichensätze nachzuladen (Pointer verbogen?). Aus diesem Grunde steht an dieser Stelle ein Beispielprogramm in C, das einen neuen Zeichensatz von Diskette lädt und in verschiedenen Größen auf den Bildschirm bringt. Das Programm ist zwar in allen Auflösungen lauffähig, u.U. können in niedrigen

Auflösungen allerdings ein paar Zeichengrößen fehlen, was der Sache aber keinerlei Abbruch tut! Vergessen Sie bitte nicht, vorher die Diskette mit dem AUTO-Ordner zu booten und auch die notwendigen Files auf Diskette zu übertragen! (s.o.)

```
*****
**                                     **
** Neue Zeichensätze nachladen      **
**                                     **
**                                     **
*****

int contrl[12],
    intin[128],
    ptsin[128],
    intout[128],
    ptsout[128];
int gl_apid;
int gem_handle;
int handle;
int count;
int gl_wchar, gl_hchar, gl_wbox, gl_hbox;
int work_in[12],
    work_out[57],
    pxyarray[10];

/*****
/*      Initialisiere Applikation      */
*****/
aes_init()
{
    int i;
    gl_apid = appl_init();
    gem_handle = graf_handle(&gl_wchar, &gl_hchar,
                             &gl_wbox, &gl_hbox );
}

/*****
/*      Terminiere Applikation      */
*****/
aes_exit()
{
    gemdos(0x1);          /* Auf Taste warten */
    appl_exit();
}

/*****
/*      Öffne virtuelle Workstation    */
*****/
open_vwork()
{
    int i;
```

```

    for (i=0; i<10; work_in[i++]=1);
    work_in[10] = 2;
    handle = gem handle;
    v_opnvwk(work_in, &handle, work_out);
}

/*****
/*      SchlieÙe virtuelle Workstation      */
*****/

#define close_vwork()  v_clsvwk(handle);

/*****
/*      Hauptprogramm      */
*****/
int  set_font,
     set_baseline,
     char_width,
     char_height,
     cell_width,
     cell_height,
     xaupl,
     yaupl,
     hor_out,
     vert_out,
     additional;

main()
{
    int i;
    aes_init();
    open_vwork();           /* Initialisierung      */
    xaupl = work_out[0];    /* Auflösung feststellen */
    yaupl = work_out[1];
    v_clrwk(handle);        /* Bildschirm löschen   */
    vswr_mode(handle, 1);   /* Schreibmodus: Replace */
    /* Zusätzlichen FONT-Satz nachladen: */
    additional = vst_load_font(handle, 1);
    /* Text-Attribute einstellen:      */
    vst_point(handle, 5, &char_width, &char_height,
                &cell_width, &cell_height);
    set_baseline = vst_rotation(handle, 0);
    set_font = vst_font(handle, 2);
    vst_color(handle, 3);
    vst_effects(handle, 0x5);
    vst_alignment(handle, 0, 0, &hor_out, &vert_out);

    /* Text in verschiedener Größe ausgeben: */
    for (i=1; i<30; i=i+1)
    {
        vst_font(handle, 1);
        vst_point(handle, i, &char_width, &char_height,
                    &cell_width, &cell_height);
        v_gtext(handle, (i<15)? 10:150, ((i<15)? i:i-15 ) * 25 + 10, "Font 1");
    }
}

```

```
vst_font(handle, 2);
vst_point(handle, 1, &char_width, &char_height,
           &cell_width, &cell_height);
v_gtext(handle, (i<15)? 80:250, ( (i<15)? i:i-15 ) *25 + 10, "Font 2");
}
vst_effects(handle, 0x0);
v_justified(handle, 40, yauf1-4, "Das war's mal wieder!", xauf1-40, 1, 1);

/* Zusätzlichen FONT-Satz wieder freigeben: */
vst_unload_fonts(handle, 1);
close_vwork();          /* Programmende */
aes_exit();
}
```

Wie Sie sehen, können Sie natürlich sämtliche Effekte auch für den neuen Zeichensatz programmieren.

Die hier verwendeten VDI-Funktionen "vst_rotation()", "v_justified()" und "vst_alignment()" lernen Sie im nächsten Kapitel kennen. Sie sollen aber auch in C vorgestellt werden, was in diesem Programm hiermit geschieht.

In diesem kleinen Beispielpogramm werden die beiden Fonts, System-Font und nachgeladener Font, gegenübergestellt, so daß Sie einen direkten Vergleich haben. Die Zeichengröße wird mit der Punkt-Funktion eingestellt, um Ihnen die verschiedenen Größen der kombinierten Fonts zu demonstrieren.

15.7.3 Komfortabel: Textformatierung

Zum vorläufigen Höhepunkt unserer Ausführungen über die Textdarstellung auf dem Atari ST gelangen wir mit der Einführung der nun folgenden Befehle. Es sind dies drei weitere VDI-Funktionen, die das Gesamtbild in angenehmer Weise abrunden. Sie dienen der komfortablen Formatierung und Justierung von Text auf dem Bildschirm:

```
SET GRAPHIC TEXT ALIGNMENT (39 VDI)
SET CHARACTER BASELINE VEKTOR (13 VDI)
JUSTIFIED GRAPHICS TEXT (11/10 VDI)
```

Zu diesen VDI-Funktionen muß einiges gesagt werden. Es geht hier um die formatierte Ausgabe eines Textes bzw. eines Strings. In dem ersten hier angeführten Befehl kann Text, man höre und staune, gedreht werden. Dabei ändert sich entsprechend auch die Schreibrichtung. Anzugeben ist der Winkel der Grundlinie des Textes (s. Skizze). Er kann, genauso wie bei den Kreisfunktionen, in 1/10 Grad angegeben werden. Das ergibt einen Wertebereich von 0 bis 3599:

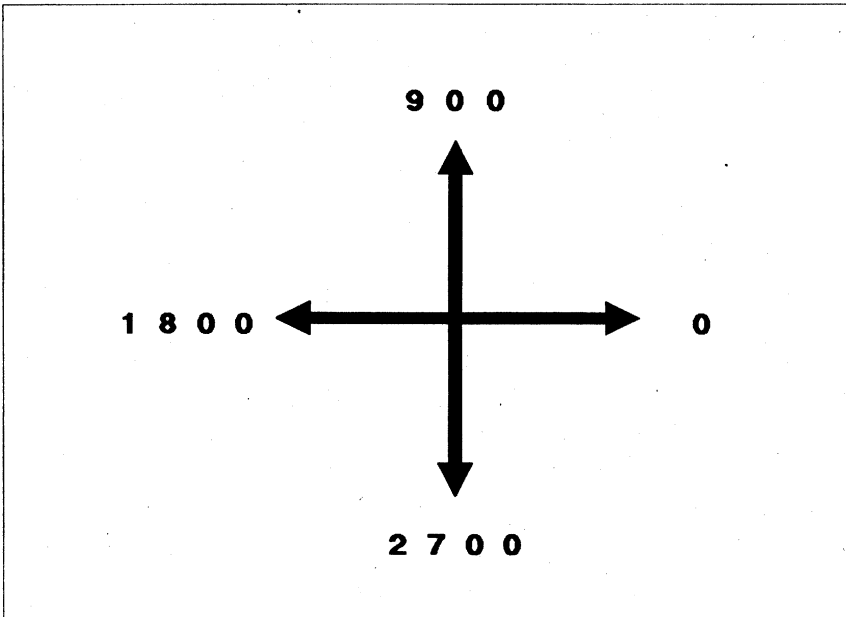


Abb. 15.32: Grad-Wertebereich Text-Rotation

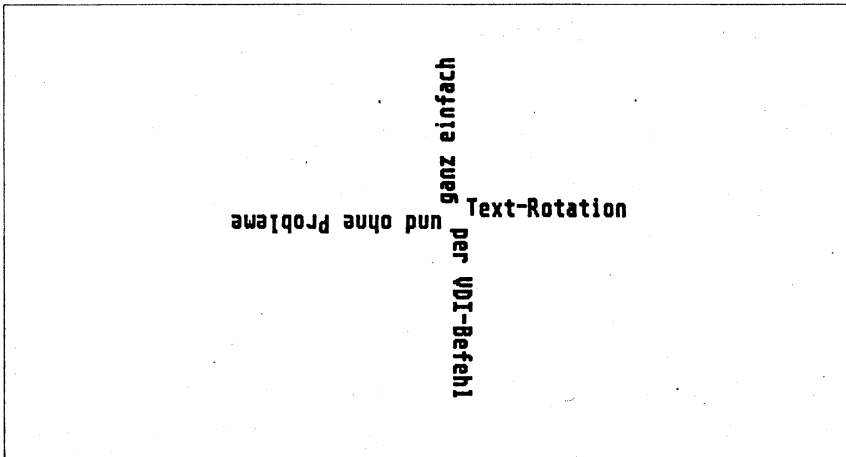


Abb. 15.33: Text in alle vier Richtungen

Unser Atari ST unterstützt allerdings nur insgesamt vier Schreibrichtungen. Damit haben lediglich die Werte 0, 900, 1800 und 2700 Effekt. Alle Zwischenwerte bewirken jeweils die nächstuntere Ausrichtung. Werte über 3599 sind - anders als bei den erwähnten Kreisfunktionen - nicht zugelassen.

Der zweite Befehl ist ein wenig komplizierter. Mit ihm wird die Position des sogenannten Aktionspunktes eines Strings festgelegt. Unter Aktionspunkt verstehen wir hier diejenige Stelle im String, die bei der Ausgabe genau auf dem Punkt zu liegen kommt, dessen x- und y-Koordinaten wir bei dem TEXT-Befehl (und wie wir sehen werden auch bei JUSTIFIED GRAPHICS TEXT) angeben. Es ist also sozusagen der Bezugspunkt im Text bei der String-Positionierung.

Bei der Angabe können Sie zwischen verschiedenen Aktionspunktlagen wählen. Die horizontale Ausrichtung bezieht sich dabei auf den Gesamt-String. Wenn Sie also für "hor_in" 0 übergeben, dann befindet sich der Aktionspunkt am äußersten linken Rand des gesamten Strings. Beim Wert 1 liegt er genau in der String-Mitte, bei 2 am rechten String-Rand.



Abb. 15.34: Horizontale Aktionspunktpositionen

Die vertikale Positionierung beschreibt die vertikale Lage des Aktionspunktes innerhalb des Textes. Sie können hier angeben, ob er sich am oberen, am unteren Rand oder an bestimmten anderen Positionen des Strings befinden soll. Dabei werden ganz bestimmte markante Punkte innerhalb einzelner Buchstaben zur Verfügung gestellt, die Sie am besten der Skizze entnehmen.

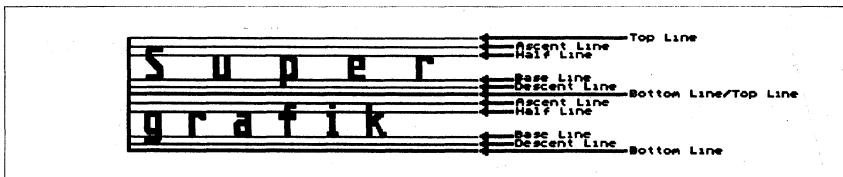


Abb. 15.35: Vertikale Aktionspunktpositionen

Bevor wir uns der Erläuterung des dritten VDI-Befehles widmen (JUSTIFIED GRAPHIC TEXT) soll an dieser Stelle ein kurzes Demo-Programm stehen, das Ihnen die Zusammenhänge zwischen Text-Rotation und den verschiedenen Möglichkeiten der Text-Ausrichtung deutlich macht. Wie in diesem Buch schon so oft praktiziert, können Sie auch hier direkt am Bildschirm wieder die verschiedenen Parameter per Tastatur verändern und so direkt erfahren, wo sich der Text befindet, wenn Sie z.B. die horizontale Ausrichtung verändern usw. Durch das alte Leid der GFA-Implementierung von VDI-Befehlen dehnt sich das Programm natürlich wieder. Haben Sie jedoch die Möglichkeit, in C zu programmieren, dann erübrigt sich das ja für Sie. Wenn Sie die Bibliotheksfunktionen einmal eingetippt bzw. von Disk geladen haben, dann können Sie sie ja zu den übrigen bereits bekannten Funktionen hinzufügen. Auf diese Weise schaffen Sie sich nach und nach eine komplette VDI-Library für GFA-BASIC.

```

' *****
' **                               **
' ** Alignment - einmal anschaulich **
' **                               **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)

handle%=DPEEK(VDIBASE+40)

DIM v_ausr$(5),h_ausr$(2)

' Alignment-Bezeichnungen aus DATA-Zeilen:
FOR i%=0 TO 5
  READ v_ausr$(i%)
NEXT i%
READ h_ausr$(0),h_ausr$(1),h_ausr$(2)

angle%=0           !Rotationswinkel
height%=13         !Zeichenhöhe
hor_in%=0          !horizontale Ausrichtung
vert_in%=0         !vertikale Ausrichtung
'

cl$="Guten Tag!"    !Text
xl%=-1
yl%=-1
'

GRAPHMODE 2        !Transparent-Modus
'                  !Versuchen Sie andere!

REPEAT
'
' Auf Mausänderung oder Taste warten:
REPEAT

```

```

MOUSE x%,y%,k%
c$=INKEY$
UNTIL c$<>" OR x%<>x1% OR y%<>y1% OR k%<>0
x1%=x%
y1%=y%
' Warten, bis wieder losgelassen:
REPEAT
UNTIL INKEY$="" AND MOUSEK=0
'
c%=ASC(RIGHT$(c$))      !Tasten-Scan-Code ermitteln
IF c%=&H48               !Cursor auf?
    height%=height%+1    !ja -> Zeichenhöhe inc
ENDIF
IF c%=&H50               !Cursor ab?
    height%=height%-1    !ja -> Zeichenhöhe dec
ENDIF
IF c%=&H4B               !Cursor links?
    angle%=angle%+900     !ja -> Rotationswinkel inc
    IF angle%>3599
        angle%=0
    ENDIF
ENDIF
ENDIF
'
IF k%=1                 !Linker Mausknopf?
    hor_in%=hor_in%+1    !ja -> hor. Ausrichtung inc
    IF hor_in%=3
        hor_in%=0
    ENDIF
ENDIF
IF k%=2                 !Rechter Mausknopf?
    vert_in%=vert_in%+1  !ja -> vert. Ausrichtung inc
    IF vert_in%=6
        vert_in%=0
    ENDIF
ENDIF
ENDIF
'
CLS                     !Bildschirm löschen
' Statuszeilen schreiben:
PRINT AT(1,22);"Rotationswinkel: ";angle%/10;" Grad"
PRINT AT(1,23);"Zeichenhöhe: ";height%
PRINT AT(1,24);"Horizontale Ausrichtung: ";h_ausr$(hor_in%)
PRINT AT(1,25);"Vertikale Ausrichtung: ";v_ausr$(vert_in%);
'
DRAW 0,y% TO xaufl%,y%  !Koordinatenlinien zeichnen
DRAW x%,0 TO x%,yaufl%
'
' Textparameter realisieren:
@vst_rotation(handle%,angle%)
@vst_height(handle%,height%,*ch_w%,*ch_h%,*ce_w%,*ce_h%)
@vst_alignment(handle%,hor_in%,vert_in%,*hor_out%,*vert_out%)
'
' Text auf Bildschirm schreiben:

```



```

    @v_gtext(handle%,x%,y%,c1$)
UNTIL c$=" " OR ASC(c$)=&H1B      !Space oder ESC = Ende
EDIT
'
' Alignment-Kennzeichnungen:
DATA Baseline,Halffline,Ascent line
DATA Bottom line,Descent line,Top line
'
DATA Linksbündig,Zentriert,Rechtsbündig
'
' VDI-Library:
'
' TEXT
PROCEDURE v_gtext(handle%,x%,y%,string$)
    string$=string$+CHR$(0)      !0 anhängen
    DPOKE CONTRL+2,1
    DPOKE CONTRL+6,LEN(string$)
    DPOKE CONTRL+12,handle%
    FOR xi%=0 TO LEN(string$)-1
        DPOKE INTIN+2*xi%,ASC(MID$(string$,xi%+1,1))
    NEXT xi%
    DPOKE PTSIN,x%
    DPOKE PTSIN+2,y%
    VDISYS 8
RETURN
'
' SET CHARACTER HEIGHT, ABSOLUTE MODE
PROCEDURE
vst_height(handle%,height%,p_char_width%,p_char_height%,p_cell_width%,p_cell_height%)
    DPOKE CONTRL+2,1
    DPOKE CONTRL+12,handle%
    DPOKE PTSIN,0
    DPOKE PTSIN+2,height%
    VDISYS 12
    *p_char_width%=DPEEK(PTSOUT)
    *p_char_height%=DPEEK(PTSOUT+2)
    *p_cell_width%=DPEEK(PTSOUT+4)
    *p_cell_height%=DPEEK(PTSOUT+6)
RETURN
'
' SET CHARACTER CELL HEIGHT, POINTS MODE
PROCEDURE
vst_point(handle%,point%,p_char_width%,p_char_height%,p_cell_width%,p_cell_height%)
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,point%
    VDISYS 107
    *p_char_width%=DPEEK(PTSOUT)
    *p_char_height%=DPEEK(PTSOUT+2)

```

```

    *p_cell_width%=DPEEK(PTSOUT+4)
    *p_cell_height%=DPEEK(PTSOUT+6)
RETURN
'
' SET CHARACTER BASELINE VEKTOR
PROCEDURE vst_rotation(handle%,angle%)
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,angle%
    VDISYS 13
    set_baseline%=DPEEK(INTOUT)
RETURN
'
' SET GRAPHIC TEXT COLOR INDEX
PROCEDURE vst_color(handle%,color_index%)
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,color_index%
    VDISYS 22
RETURN
'
' SET GRAPHIC TEXT ALIGNMENT
PROCEDURE vst_alignment(handle%,hor_in%,vert_in%,p_hor_out%,p_vert_out%)
    DPOKE CONTRL+6,2
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,hor_in%
    DPOKE INTIN+2,vert_in%
    VDISYS 39
    *p_hor_out%=DPEEK(INTOUT)
    *p_vert_out%=DPEEK(INTOUT+2)
RETURN

```

Das Programm ist gut dokumentiert und sollte von Ihnen einmal durchgesehen werden. Es zeichnet ein Fadenkreuz auf den Bildschirm, dessen Position Sie durch Mausbewegungen jederzeit verändern können. Der Schnittpunkt der beiden "Fäden" gibt dabei die jeweiligen Koordinaten an, die dem Befehl TEXT zur Textausgabe übergeben werden. Der vorgefertigte Text steht dabei in der String-Variablen c1\$ und kann von Ihnen natürlich beliebig verändert werden. Ausnahmsweise wird er im Transparent-Modus auf den Bildschirm gebracht, um das Fadenkreuz nicht zu löschen. Nach dem Löschen des gesamten Bildschirms wird dann nach jeder Änderung, die Sie über verschiedene Tasten vornehmen können, das Bild wieder neu aufgebaut.

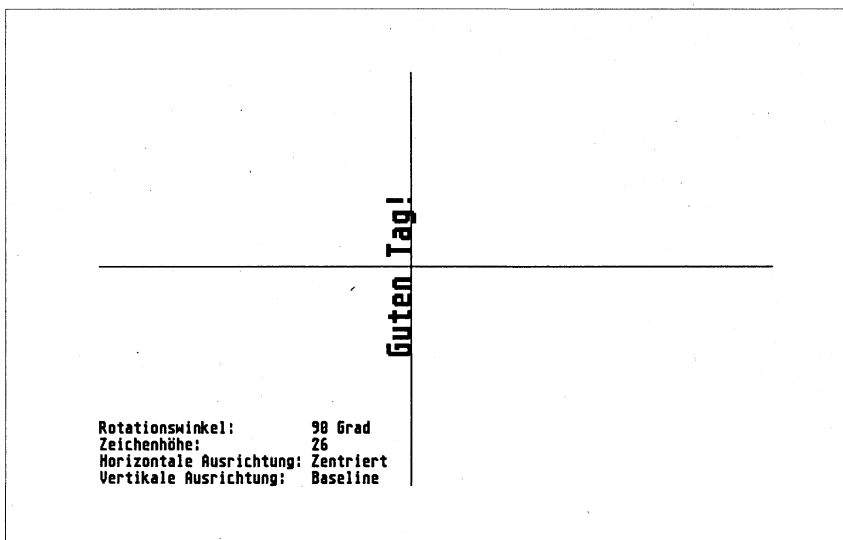


Abb. 15.36: Alignment - einmal anschaulich

In vier Statuszeilen erfahren Sie jeweils die aktuellen Werte für Rotationswinkel, Zeichenhöhe und Ausrichtung. Über die folgenden Benutzereingaben können Sie die verschiedenen Text-Attribute verändern:

- Mausbewegung - Textkoordinaten ändern
- Cursor auf - Zeichenhöhe steigern
- Cursor ab - Zeichenhöhe verringern
- Cursor links - Rotationswinkel um 900 (90 Grad) erhöhen
- Mausknopf li - Horizontale Ausrichtung erhöhen
- Mausknopf re - Vertikale Ausrichtung erhöhen
- Esc o. Space - Programmende

So, nun wissen Sie hoffentlich genau Bescheid, und wir können beim Text-Kommando weitermachen.

Wie der Name bereits andeutet, wird mit der VDI-Bibliotheksfunktion JUSTIFIED GRAPHICS TEXT ein String nicht nur einfach auf dem Bildschirm ausgegeben, wie wir das vom TEXT-Befehl her kennen. Vielmehr erfolgt eine komplette Text-Justierung. Was das für uns bedeutet, werden wir zu ergründen versuchen.

Zunächst einmal wird hier unter Justierung das Dehnen oder Stauchen eines Strings auf eine bestimmte Länge verstanden. Wenn wir mit der alten Funktion TEXT Strings auf dem Bildschirm ausgeben, dann wird das Ergebnis auf dem Bildschirm stets die Länge besitzen, die von der Zeichenzahl des Strings festgelegt wird. Hier nun haben wir die Möglichkeit anzugeben, wie viele Pixel der gesamte String lang sein darf. Die hier vorgestellte Ausgabefunktion dehnt oder staucht nun den auszugebenden Text auf diese von Ihnen eingestellte Länge.

Dies kann auf verschiedene Weise erfolgen. Da die einzelnen Zeichen selbst nicht einfach gedehnt bzw. gestaucht werden können, müssen die Zwischenräume zwischen ihnen erhalten. Dabei gibt es die Option der Dehnung der Zwischenräume zwischen den einzelnen Zeichen, wie die Möglichkeit der Dehnung der Zwischenräume zwischen Wörtern (also die Dehnung von Leerzeichen). Im ersten Fall werden die Wörter auseinandergezogen, im zweiten bleibt die Länge der einzelnen Wörter gleich, der Abstand untereinander ändert sich allerdings. Beide Justierungsmöglichkeiten können auch gleichzeitig eingestellt werden. Werden beide Justierungen ausgeschaltet, dann funktioniert dieser Befehl genauso wie der bekannte TEXT.

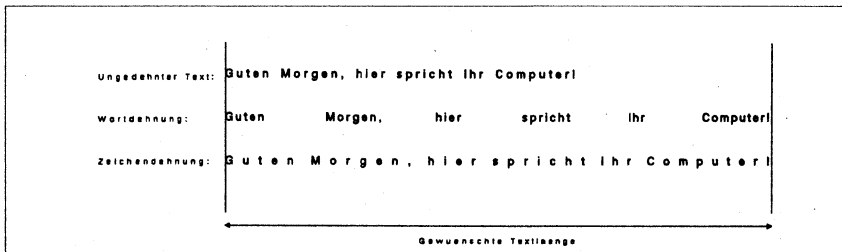


Abb. 15.37: Wort- und Zeichendehnung (Justified text)

Sie können sich vorstellen, welche Möglichkeiten sich Ihnen hier bieten! Zusammen mit der Textausrichtung können Sie ohne viele Probleme z.B. eine professionelle Textverarbeitung entwickeln mit Direktformatierung auf dem Bildschirm, Zentrierung und und und. Wenn Sie zusätzlich in Besitz eines Druckers sind, der ebenfalls selbsttätig Text formatieren kann, wie dies beispielsweise die EPSON-Drucker FX85 oder FX105 machen (bei ihnen kann eingegebener Text rechts-, linksbündig oder im Blocksatz ausgegeben werden), braucht sich Ihr Textverarbeitungsprogramm nicht darum zu kümmern.

Natürlich soll das alles nicht nur graue Theorie bleiben. Wie einfach die Programmierung eines solchen Textverarbeitungsprogramms unter VDI ist, soll Ihnen die nun folgende Anwendung zeigen, die die Grundzüge eines solchen

Projektes demonstriert. Es stellt tatsächlich einen BildschirmditorEditor dar, der Text direkt am Bildschirm formatiert, was so manches Textverarbeitungsprogramm noch nicht kann! In einem solchen Buch kann natürlich kein professionelles Textprogramm vorgestellt werden. Aus diesem Grunde sind gewisse Einschränkungen unvermeidbar. Aber sehen Sie selbst:

```

' *****
' **                               **
' ** Mini-Textomat **
' **                               **
' *****

' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)

handle%=DPEEK(VDIBASE+40)

IF xauf1%<639                                !Texthöhe in Abhängigkeit
  choehe%=6                                  !von Auflösung!
ELSE
  choehe%=13
ENDIF

rows%=80*xauf1%/639                          !Anzahl Bildschirmspalten
columns%=25                                   !Anzahl Bildschirmzeilen

' Höhe setzen und Zeichenformate erfragen:
@vst_height(handle%,choehe%,*char_width%,*char_height%,*cell_width%,*cell_height%
)

DIM zeile$(rows%)                            !Textspeicher

lr%=10,                                       !Linker Rand
rr%=70*xauf1%/639                           !Rechter Rand
or%=4                                         !Oberer Rand
ur%=20                                       !Unterer Rand

crsr_row%=0                                  !akt. Cursorzeile
crsr_col%=0                                  !akt. Cursorspalte

effect%=0                                     !Special Effects
word_space%=1                               !Justified-Parameter
char_space%=1

leerz$=SPACE$(rr%-lr%)                      !Leerzeile definieren

' Speicher löschen:
FOR ir%=0 TO rows%-1
  zeile$(ir%)=""
NEXT ir%

```



```

    zw1%=word_space%
    zw2%=char_space%
    word_space%=0
    char_space%=0      !keine Justierung!
    @crsr_cr           !CR ausführen
    @newzeile j(crsr_row%) !Und Zeile unjustiert schreiben
    word_space%=zw1%
    char_space%=zw2%
ENDIF
,
IF c_scan%=&H62      !Help-Taste?
    word_space%=ABS(word_space%-1) !zwischen 0 und 1 wechseln
    @statuszeile    !Statuszeile schreiben
    @formatiere     !Text neu formatieren
ENDIF
,
IF c_scan%=&H61      !Undo-Taste?
    char_space%=ABS(char_space%-1) !zwischen 0 und 1 wechseln
    @statuszeile    !Statuszeile schreiben
    @formatiere     !Text neu formatieren
ENDIF
,
IF c_scan%=&H4B      !Cursor links?
    effect%=effect%-1 !Effektnummer verringern
    @statuszeile
    @formatiere
ENDIF
,
IF c_scan%=&H4D      !Cursor rechts?
    effect%=effect%+1 !Effektnummer erhöhen
    @statuszeile
    @formatiere
ENDIF
,
ELSE
    , Zeichen in Zeile aufnehmen:
    ,
    wort$=wort$+c$      !Zunächst in Wortspeicher
    @newzeile uj(crsr_row%) !Zeile ausgeben
    , Bei Wortende oder Zeilenende Wort an Zeile anhängen:
    IF c$=" " OR (crsr_col%+1=rr%-lr% AND zeile$(crsr_row%)="")
        zeile$(crsr_row%)=zeile$(crsr_row%)+wort$
        wort$=""      !Und Wortspeicher löschen
    ENDIF
    crsr_col%=crsr_col%+1 !Cursorspalte erhöhen
    ,
    IF crsr_col%=rr%-lr% !Rechten Rand erreicht?
        @crsr_cr      !CR ausführen
        @newzeile uj(crsr_row%) !und neue Zeile ausgeben
    ENDIF
,
ENDIF
ENDIF

```

```

'
'
UNTIL c$=CHR$(&H1B)           !Bei ESC: Ende
'
EDIT
'
'
' Zeile justiert neu schreiben:
PROCEDURE newzeile j(reihe%)
  @vst_effects(handle%,effect%) !Effekte setzen
  '
  xx%=cell_width%*lr%           !x-Punkt-Koord. Linker Rand
  yy%=cell_height%*(or%+reihe%) !y-Punkt-Koord. Zeile
  zz%=cell_width%*(rr%-lr%)     !Zeilenbreite in Punkten
  '
  @v_gtext(handle%,xx%,yy%,leerz$) !Zeile löschen
  '
  ' Leerzeichen am Zeilenende löschen:
  REPEAT
    aa$=zeile$(reihe%)
    '
    IF RIGHT$(aa$,1)=" "
      zeile$(reihe%)=LEFT$(aa$,LEN(aa$)-1)
    ENDIF
  UNTIL RIGHT$(aa$,1)<>" "
  '
  ' Zeile justiert ausgeben:
  @v_justified(handle%,xx%,yy%,zeile$(reihe%),zz%,word_space%,char_space%)
RETURN
'
' Zeile unjustiert neu schreiben:
PROCEDURE newzeile uj(reihe%)
  xx%=cell_width%*lr%           !x-Koord. Linker Rand
  yy%=cell_height%*(or%+reihe%) !y-Koord. Zeile
  @v_gtext(handle%,xx%,yy%,leerz$) !Zeile löschen
  ' Und unjustiert ausgeben:
  @v_gtext(handle%,xx%,yy%,zeile$(reihe%)+wort$)
RETURN
'
' Gesamten Text neu formatieren:
PROCEDURE formatiere
  FOR zi%=0 TO crsr_row%
    @newzeile_j(zi%)           !Zeile für Zeile neu
  NEXT zi%
RETURN
'
' Cursor carriage return:
PROCEDURE crsr_cr
  @crsr_down                   !Cursor ab
  crsr_col%=LEN(wort$)         !auf Wortende
RETURN
'
' Cursor eine Zeile tiefer:

```



```

PROCEDURE crsr down
  IF crsr_row% < ur% - or%
    @newzeile_j(crsr_row%)           !Zeile justiert ausgeben
    crsr_row% = crsr_row% + 1         !Cursorzeile erhöhen
  ENDIF
RETURN
'
' Cursor eine Spalte nach links
PROCEDURE crsr left
  IF crsr_col% > 0
    crsr_col% = crsr_col% - 1         !Cursorspalte verringern
  ENDIF
RETURN
'
' Statuszeile schreiben:
PROCEDURE statuszeile
  PRINT AT(1,1); "word_space: "; word_space%
  PRINT AT(16,1); "char_space: "; char_space%
  PRINT AT(31,1); SPC(15)
  PRINT AT(31,1); "Effekt: "; effect%
RETURN
'
'
' VDI-Funktionen:
'
' TEXT
PROCEDURE v_gtext(handle%, x%, y%, string$)
  string$ = string$ + CHR$(0)         !0 anhängen
  DPOKE CONTRL+2, 1
  DPOKE CONTRL+6, LEN(string$)
  DPOKE CONTRL+12, handle%
  FOR xi% = 0 TO LEN(string$) - 1
    DPOKE INTIN+2*xi%, ASC(MID$(string$, xi%+1, 1))
  NEXT xi%
  DPOKE PTSIN, x%
  DPOKE PTSIN+2, y%
  VDISYS 8
RETURN
'
' JUSTIFIED GRAPHICS TEXT
PROCEDURE v_justified(handle%, x%, y%, string$, length%, word_space%, char_space%)
  string$ = string$ + CHR$(0)
  DPOKE CONTRL+2, 2
  DPOKE CONTRL+6, 2+LEN(string$)
  DPOKE CONTRL+10, 10                 !Sekundär-Opcode
  DPOKE CONTRL+12, handle%
  DPOKE INTIN, word_space%
  DPOKE INTIN+2, char_space%
  FOR xi% = 0 TO LEN(string$) - 1
    DPOKE INTIN+2*xi%+4, ASC(MID$(string$, xi%+1, 1))
  NEXT xi%
  DPOKE PTSIN, x%
  DPOKE PTSIN+2, y%

```

```
DPOKE PTSIN+4,length%
DPOKE PTSIN+6,0
VDISYS 11
RETURN
'
' SET CHARACTER HEIGHT, ABSOLUTE MODE
PROCEDURE
vst_height(handle%,height%,p_char_width%,p_char_height%,p_cell_width%,p_cell_height%)
    DPOKE CONTRL+2,1
    DPOKE CONTRL+12,handle%
    DPOKE PTSIN,0
    DPOKE PTSIN+2,height%
    VDISYS 12
    *p_char_width%=DPEEK(PTSOUT)
    *p_char_height%=DPEEK(PTSOUT+2)
    *p_cell_width%=DPEEK(PTSOUT+4)
    *p_cell_height%=DPEEK(PTSOUT+6)
RETURN
'
' SET CHARACTER CELL HEIGHT, POINTS MODE
PROCEDURE
vst_point(handle%,point%,p_char_width%,p_char_height%,p_cell_width%,p_cell_height%)
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,point%
    VDISYS 107
    *p_char_width%=DPEEK(PTSOUT)
    *p_char_height%=DPEEK(PTSOUT+2)
    *p_cell_width%=DPEEK(PTSOUT+4)
    *p_cell_height%=DPEEK(PTSOUT+6)
RETURN
'
' SET GRAPHIC TEXT COLOR INDEX
PROCEDURE vst_color(handle%,color_index%)
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,color_index%
    VDISYS 22
RETURN
'
' SET GRAPHIC TEXT SPECIAL EFFECTS
PROCEDURE vst_effects(handle%,effect%)
    DPOKE CONTRL+6,1
    DPOKE CONTRL+12,handle%
    DPOKE INTIN,effect%
    VDISYS 106
RETURN
```

word_space: 1 char_space: 1 Effekt: 5

*Wier erleben Sie, was Ihre und stanne,
die simple Realisierung von direkter
Formatierung eines Textes auf den
Bildschirm. Mit der hilfreichen
Unterstützung des Betriebssystems macht
es uns überhaupt kein Problem, solche
Werte, die sonst nur in komfortabelsten
Textverarbeitungsprogrammen vorkommen,
ohne besondere Umstände selbst in BASIC
zu lösen. Klingt nicht schlecht, oder?
§*

Abb. 15.38: Mini-Textomat

Wie in einem ganz normalen Textverarbeitungsprogramm können Sie, sobald Sie das Programm gestartet haben, beliebigen Text eingeben. Sobald Sie über die im Programm eingestellte Textbreite hinausschreiben (können Sie natürlich jederzeit im Programm ändern), wird die Zeile nach den aktuell eingestellten Justierungsmodi, die oben in einer Statuszeile angezeigt werden, formatiert, und Sie schreiben in der nächsten Zeile weiter. Das Ganze können Sie so ohne weiteres fortführen, bis Sie am Bildschirmende ankommen, dann geht es leider nicht weiter.

Sie sind in der Lage, nach der Eingabe eines Return jederzeit den gesamten Text neu zu formatieren, indem Sie die einstellbaren Justierungen ändern. Auch die Änderung der Schrifteffekte ist dann möglich. Die folgende Aufstellung gibt Ihnen eine Übersicht über Ihre möglichen Benutzerinterventionen:

Zeichen	Einfügen eines Zeichens in die aktuelle Zeile
Backspace	Löschen des Zeichens vor dem Cursor bis maximal zum Zeilenanfang
Return	vorzeitiger Abschluß einer Zeile
Help	Wortzwischenraumdehnung ein/aus
Undo	Zeichenzwischenraumdehnung ein/aus
Cursor li	Schrifteffektnummer verringern
Cursor re	Schrifteffektnummer erhöhen
Esc	Programmende

Wie Sie sehen, können Sie nach der Fertigstellung einer Zeile keine Veränderungen mehr an ihr vornehmen. Das würde den Programmieraufwand wieder um eine Stufe erhöhen.

Kommen wir nun zur Programmbeschreibung. Das sollten wir hier doch einmal wieder richtig ausführlich machen, da wir es mit einem relativ komplizierten Programm zu tun haben. Die Kommentare am Programmrand werden sicher ihren Teil dazu beisteuern, Ihnen alles so verständlich wie nur irgend möglich zu machen.

Nach der üblichen Feststellung der momentanen Bildschirmauflösung und der Einstellung der auflösungsabhängigen Zeichengröße werden einige wichtige Initialisierungen vorgenommen. Hier können Sie jederzeit selbst Werte wie linker, rechter, oberer und unterer Rand einstellen usw.

Interessant wird es, wenn die große, alles umfassende Programmschleife beginnt. Sie wird erst verlassen, wenn Sie `[Esc]` zum Programmende eingeben.

Um beim Warten auf Ihre Eingabe einen blinkenden Cursor (hier in Form eines Tiefstriches) an der richtigen Position darstellen zu können, wird diese zunächst aus der Zeichenbreite und -höhe (sie wurden anfangs mit dem Befehl "vst_height" ermittelt) sowie den Randwerten und der aktuellen Zeichenposition berechnet. Innerhalb der folgenden Warteschleife kann dann dieser Cursor stets gezeichnet und wieder gelöscht werden, woraus der Blinkeffekt resultiert.

Haben Sie tatsächlich eine Taste betätigt, so werden ASCII- und Scan-Werte dieser Taste ermittelt (der Scan-Wert ist die Tastennummer), um entsprechend der Eingabe zu verzweigen. Falls es sich um ein druckbares Zeichen handelt, muß es in den Text aufgenommen werden.

Hierzu wird hinter das ELSE der großen IF...ELSE-Anweisung gesprungen (Kommentar: "Zeichen in Zeile aufnehmen"). Wir wollen diesen Part zunächst besprechen, da wir aufgrund dessen die Struktur des Textspeichers kennenlernen.

Der gesamte Text wird zeilenweise in einem String-Array gespeichert (Name: zeile\$()). Wird ein neues Zeichen eingegeben, so gelangt es allerdings nicht sofort in diesen Zeilenspeicher. Es wird zunächst in einem Wortspeicher (Name: wort\$) gehalten, der erst dann in das Array übertragen wird, sobald ein Wort zu Ende geschrieben ist.

Damit jederzeit die aktuelle Zeile neu und unjustiert auf den Bildschirm geschrieben werden kann, wurde ein Unterprogramm angelegt namens "newzeile_uj", das diese Aufgabe übernimmt. Es wird direkt nach der Übernahme des Zeichens in den Wortspeicher aufgerufen (s. Listing).

In diesem Unterprogramm, dem die Nummer der auszugebenden Zeile übergeben wird, wird zunächst die neu zu schreibende Zeile gelöscht. Die notwendigen Koordinaten und die Zeilenlänge werden dabei aus den verschiedenen Parametern ähnlich den Koordinaten zur Berechnung der Cursorposition ermittelt. Dann wird der Inhalt des Zeilenspeichers plus dem Inhalt des Wortspeichers mittels "v_gtext()" unjustiert ausgegeben.

Zurück im Hauptprogramm wird getestet, ob es sich bei der letzten Zeicheneingabe um ein Wortende handelte. Dies ist der Fall, wenn ein Leerzeichen eingegeben wurde oder die gesamte Zeile bisher aus einem einzigen Wort besteht, das nun über die Zeilengrenze hinausgeschrieben wurde. In diesem Fall hängt das Programm den Inhalt des Wortregisters an den Zeilenspeicher hinten an.

Wurde nun der linke Rand erreicht, so wird der Cursor mittels der Routine "crsr_cr" eine Zeile tiefer und auf das Ende des noch nicht fertig geschriebenen Wortes gesetzt (das unfertige Wort muß schließlich in die nächste Zeile übernommen werden). Mit der Routine, die den Cursor eine Zeile tiefer setzt ("crsr_down") wird ebenfalls die gesamte Zeile (natürlich ohne Wortspeicher) diesmal justiert ausgegeben (Routine: "newzeile_j").

Das Unterprogramm zur justierten Ausgabe macht eigentlich das gleiche, wie dasjenige zur unjustierten Textdarstellung. Zusätzlich werden jedoch eventuell am Zeilenende befindliche Leerzeichen eliminiert. Dann wird endlich die heiß ersehnte Funktion "v_justified()" aufgerufen!

Doch kommen wir wieder zurück zum Hauptprogramm. Wir waren an der Stelle stehengeblieben, an der das Zeilenende erreicht wurde. Nachdem nun die gefüllte Zeile justiert (ohne das unfertige Wort am Ende der Zeile) ausgegeben wurde, muß nun nur noch die neue Zeile zu Bildschirm gebracht werden ("newzeile_uj").

Jetzt wissen wir, wie der Text aufgebaut ist, und wir können uns mit den verschiedenen Kontroll-Funktionen beschäftigen.

Da wäre als allererstes die Tasteneingabe Backspace. Sie ist die komplizierteste von allen, da sie ein Zeichen aus einer Zeile löscht. Solange noch Zeichen im Wortspeicher vorhanden sind, ist das Ganze kein Problem, das Wort wird einfach um ein Zeichen verkürzt. Ist der Wortspeicher allerdings leer, dann

gibt es da nicht mehr viel zu kürzen. In diesem Fall muß der gesamte Zeilenspeicher nach dem ersten Leerzeichen von hinten durchsucht werden, das den Start des letzten Wortes im Zeilenspeicher markiert. Ist dieser Wortanfang gefunden, dann wird das gesamte Wort in den Wortspeicher übertragen und aus dem Zeilenspeicher gelöscht.

Jetzt haben wir wieder etwas zum verkürzen. Natürlich darf nicht vergessen werden, die Zeile unjustiert neu auszugeben.

Ist der Speicher "c_asci%" gleich 13, so haben Sie die `[Return]`-Taste betätigt. In dem Fall wird die Zeile vorzeitig beendet, das aktuelle Wort in den Zeilenspeicher übertragen und die Zeile unjustiert ausgegeben. Im Falle der `[Help]`-Taste wird die Variable "word_space%" zwischen 0 und 1 gewechselt. Da ihr Wert aktuell in der Statuszeile erscheint, wird dies mit der Routine statuszeile neu ausgegeben. Anschließend wird der gesamte Text neu formatiert.

Auf diese Weise werden nach und nach alle Tasteneingaben abgearbeitet. Sie sehen also, wie im Grunde recht einfach mit VDI selbst komplizierte Programme bewältigt werden können.

Zum guten Schluß seien noch kurz die GFA-BASIC-Befehle vorgestellt, die einige der VDI-Funktionen bezüglich der Textausgabe übernehmen:

DEFTEXT fa,ef,wi,gr (GFA-BASIC-Befehl)

Definieren einiger Textattribute

fa: Textfarbindex
ef: Texteffekt (s.o.)
wi: Rotationswinkel (s.o.)
gr: Textgröße in Pixel (s.o.)

TEXT x,y,str\$ (GFA-BASIC-Befehl)

Schreiben evtl. justierten Textes gemäß Textattributen

x: x-Koordinate des Textes
y: y-Koordinate des Textes
l: Gewünschte Textlänge >0: Dehnung der Zeichenzwischenräume, <0: Dehnung der Wortzwischenräume
str\$: Auszugebender String oder String-Variable

15.8 Bildschirmfenster und Grafikausgabe

Ihr Atari ST kann nicht nur einfache Grafiken auf den Bildschirm zaubern. Bekannt und beliebt ist er vor allem durch sein GEM, das erlaubt, z.B. mit Bildschirmfenstern zu arbeiten.

Bildschirmfenster bedürfen aber besonderer Beachtung bei der Programmierung von Grafik. Schließlich darf kein Grafikteil über den Rand eines solchen Fensters hinausragen, da er sonst unerlaubte Teile überschreiben würde. Andererseits ist es natürlich auch schwierig, einen Buchstaben mittendrin einfach abzuschneiden oder nur die Teile eines Kreises zu zeichnen, die innerhalb dieses Fensters liegen, ohne die Unterstützung des Betriebssystems.

Auch sollten Inhalte von Fenstern verschoben, kopiert und abgespeichert werden können usw. Hierfür müssen ebenfalls besondere Funktionen existieren.

15.8.1 Ein Fenster zur Grafik - einfache Fenstertechnik

Wir haben das Problem bereits umrissen: Wie schaffen wir es, eine beliebige Grafik auszugeben und trotzdem nur innerhalb eines bestimmten Fensters zu zeichnen?

Nun, VDI stellt uns hierzu eine zentrale und besonders wichtige Routine zur Verfügung, die Sie unbedingt kennen sollten. Sie hat den Namen SET CLIPPING RECTANGLE und definiert ein einfaches Rechteck. Das hat den Effekt, daß sämtliche Grafikausgaben nur noch den Bereich dieses beliebig großen Rechtecks berücksichtigen. Alle Grafikausgaben, die nicht in diesem Rechteck stattfinden, werden einfach ignoriert. Das heißt aber nicht, daß der Mittelpunkt Ihres Kreises oder der Startpunkt einer Linie nicht außerhalb dieses Fensters liegen darf. Es werden jedoch nur diejenigen Teile dieser Grafikobjekte gezeichnet, die innerhalb des Clip-Rechtecks liegen, die anderen Teile werden einfach abgeschnitten (ge-clippt). Sie kennen den Effekt vielleicht bereits von den Kreisbefehlen. Lag z.B. der Mittelpunkt eines Kreises am Bildschirmrand, so wurde - natürlich - nur der Teil gezeichnet, der innerhalb des Bildschirmfensters lag.

Die Beschreibung finden Sie im Teil D unter:

SET CLIPPING RECTANGLE (129 VDI)

Wie Sie sich denken können, wird Sie dieser Befehl ständig begleiten, wenn Sie beabsichtigen, sich mit Bildschirmfenstern o.ä. herumzuschlagen.

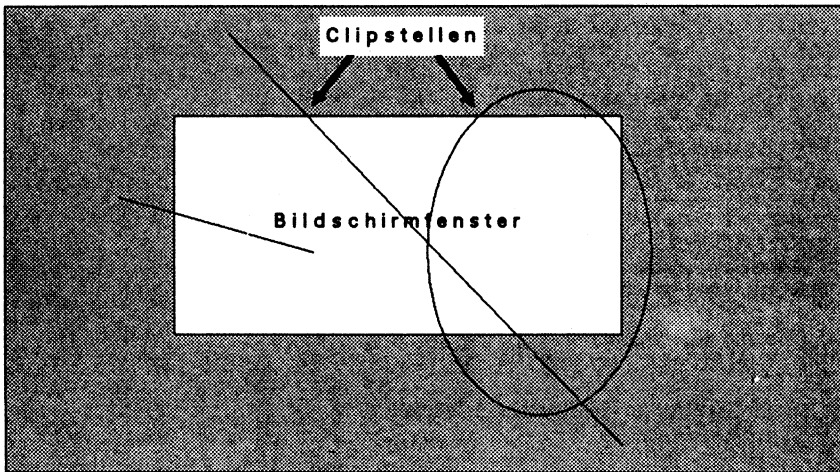


Abb. 15.39: Clipping von Linien, Kreisen etc.

Um Sie nicht allzu sehr auf dem Trockenen zu lassen, folgt selbstverständlich wieder ein kleines Beispielprogramm in GFA-BASIC. Das Programm arbeitet in der Version 2.x und 3.x. Zusätzlich existieren in der Version 3.x eigene Befehle für diesen Zweck.

```

' *****
' **
' ** Clipping **
' **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
handle%=DPEEK(VDIBASE+40)
'
DIM pxyarray%(3)
'
GRAPHMODE 1 !Replace-Modus
'
DEFFILL 1,2,2 !Punkt-Muster
PBOX 0,0,xauf1%,yauf1% !gesamten Bildschirm füllen
'
DEFFILL ,0 !Füllen mit Hintergrundfarbe

```



```

'
IF xaufl%>320
  PRINT AT(29,1);"Fenster in der Grafik"
ELSE
  PRINT AT(10,1);"Fenster in der Grafik"
ENDIF
'
' Schleife zur Fenstervergrößerung:
FOR c%=10 TO xaufl%/2 STEP 20
  @vs_clip(handle%,0,*pxyarray%()) !Clipping aus
  pxyarray%(0)=xaufl%/2-c% !Clipp-Rechteck definieren
  pxyarray%(1)=yaufl%/2-c%/4
  pxyarray%(2)=xaufl%/2+c%
  pxyarray%(3)=yaufl%/2+c%/4
  ' Rahmen um Fenster zeichnen:
  BOX pxyarray%(0)-1,pxyarray%(1)-1,pxyarray%(2)+1,pxyarray%(3)+1
  ' Clipping einschalten:
  @vs_clip(handle%,1,*pxyarray%())
  ' Durch Clipping nur Fensterinhalt löschen:
  PBOX 0,0,xaufl%,yaufl%
  ' Zeichenschleife:
  FOR x%=0 TO xaufl% STEP 2
    DRAW x%,x%/20*SIN(x%/50)+yaufl%/2 TO x%/2,60*SIN(x%/60)+yaufl%/2
  NEXT x%
NEXT c%
'
' Clipping wieder aus:
@vs_clip(handle%,0,*pxyarray%())
'
VOID INP(2)
EDIT
'
' VDI-Library:
'
' SET CLIPPING RECTANGLE
PROCEDURE vs_clip(handle%,clip_flag%,p_pxyarray%)
  DPOKE CONTRL+2,2
  DPOKE CONTRL+6,1
  DPOKE CONTRL+12,handle%
  DPOKE INTIN,clip_flag%
  SWAP *p_pxyarray%,xi%()
  DPOKE PTSIN,xi%(0)
  DPOKE PTSIN+2,xi%(1)
  DPOKE PTSIN+4,xi%(2)
  DPOKE PTSIN+6,xi%(3)
  SWAP *p_pxyarray%,xi%()
  VDISYS 129
RETURN

```

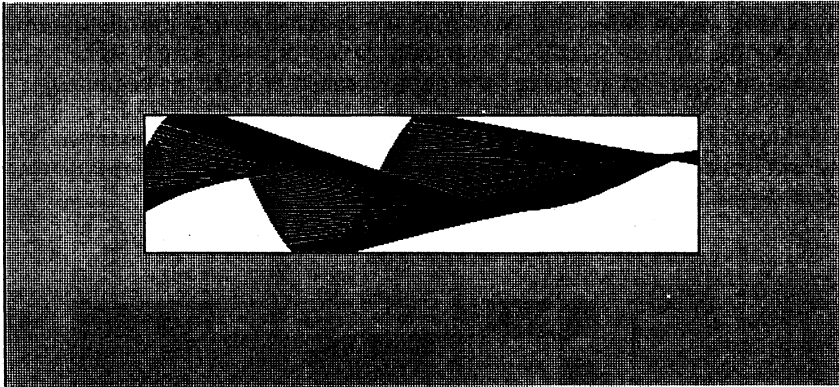


Abb. 15.40: Clipping

In diesem Programm wird in mehreren Durchläufen stets dieselbe Funktion oder besser dasselbe Grafikgebilde gezeichnet. In jedem Durchlauf wird jedoch das hierzu definierte Clip-Fenster ständig vergrößert, so daß ein immer größerer Ausschnitt dieses Gebildes zu sehen ist. Gelöscht wird die Figur durch Zeichnen eines mit Hintergrundfarbe gefüllten Rechtecks in Bildschirmgröße, das ebenfalls durch das Clipping auf das Clip-Fenster beschränkt wird. Wollen Sie mit der normalen Funktion `CLEAR WORKSTATION` bzw. `CLS` in GFA-BASIC arbeiten, so sollten Sie wissen, daß diese Funktion das Clipping nicht berücksichtigt. Gelöscht wird also in jedem Fall der gesamte Bildschirm!

15.8.2 Kopieren und Überlagern

Eine wesentliche Grundlage für den Umgang mit Bildschirmfenstern ist die Beherrschung der nun folgenden Befehle. Wir haben es hier mit den sogenannten Rasterfunktionen des VDI zu tun. Diese Befehle haben, wie der Name schon sagt, die Aufgabe, mit Rastern zu arbeiten.

Was dabei unter Rastern zu verstehen ist, ist nicht ganz einfach zu erklären. Ein uns sehr wohl bekanntes Raster ist der Speicher unseres Grafikbildschirms. Aber auch vor allem die speichermäßige Repräsentation rechteckiger Bildschirmausschnitte stellen hier ein eigenes Teilraster dar.

Das Problem hierbei ist die Tatsache, daß ein solcher rechteckiger Ausschnitt nicht unbedingt durch hintereinanderfolgende Bytes im Speicher repräsentiert wird. Vielmehr besteht zwischen jeder Grafik-Zeile eine Lücke von mehreren,

manchmal vielen Bytes. Erst dann folgt ein Speicherbereich, der die Information für die nächste Grafik-Zeile enthält. Dabei braucht so ein Raster nicht einmal ausgerechnet an einem Byte- oder Wortstart im Speicher beginnen.

Der Idealfall eines Teilrasters ist dann natürlich eines, das den gesamten Bildschirm umfaßt. In diesem Raster käme dann tatsächlich jedes Start-Byte einer Zeile direkt hinter dem End-Byte der vorherigen.

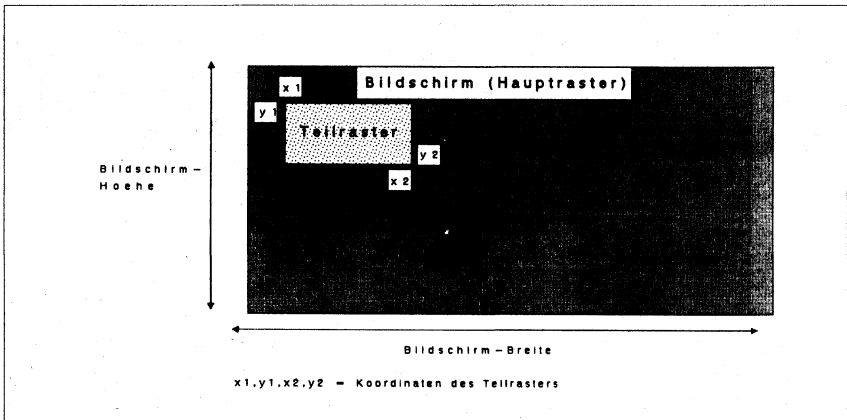


Abb. 15.41: Rasterbereich in Bildschirmraster

Die hier zu besprechenden Befehle können nun mit solchen Rastern arbeiten, sie kopieren, miteinander verknüpfen etc. Sie sind dabei in der Lage, nicht nur mit Teilrastern eines großen Rasters zu arbeiten, das so groß ist wie der Bildschirm. Sie können sich ebenso sozusagen einen kleinen Bildschirm mit beispielsweise 10 mal 20 Punkten oder auch einen Riesenschirm mit 2000 mal 2000 Punkten denken. Aus diesen gedachten Bildschirmen (oder eigentlich besser: Bildschirmspeichern) können Sie mit den Rasterfunktionen nun Teilraster auskopieren und in andere gedachte oder gar in den tatsächlichen Bildschirm übertragen, mit ihm verknüpfen und so weiter.

Die einzigen Voraussetzungen für einen solchen Bildschirm sind die folgenden: Jede Zeile des gedachten Bildschirms muß von links nach rechts linear im Speicher vorliegen. Nach dem End-Byte einer beliebigen Zeile muß direkt das Start-Byte der folgenden Zeile kommen. Sollen mehrfarbige Raster bearbeitet werden, dann müssen die Farbebenen genauso abgelegt sein wie im originalen Bildschirm. Wo dann das jeweilige Teilraster liegt, um das es dann im Endeffekt ja wirklich geht, und wie groß es ist, ist dann völlig gleichgültig.

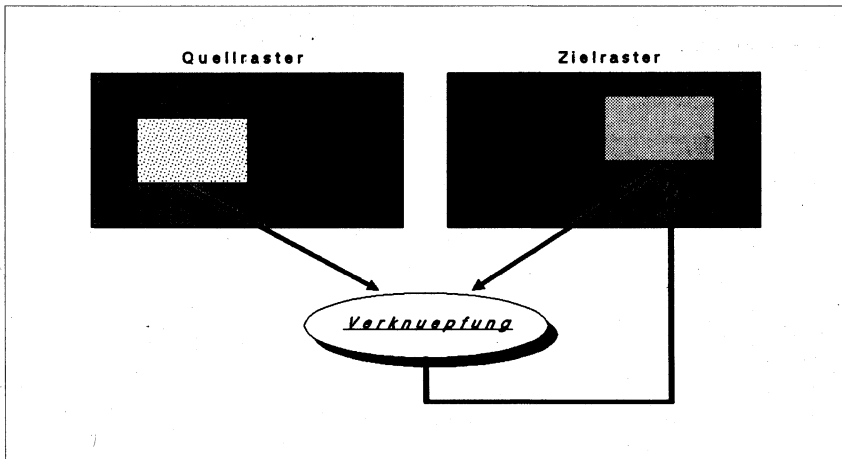


Abb. 15.42: Verknüpfung zweier Teilraster

Bevor wir mit den Befehlen arbeiten, ist es also notwendig, die Ausmaße und den Speicherbereich eines solchen gedachten Bildschirmspeichers zu definieren. Hierzu müssen zwei sogenannte Memory Form Definition Blocks (MFDB) angelegt werden, einer für das Quellraster, einer für das Zieleraster.

Dies sind insgesamt 20 Bytes große Blöcke mit folgendem Aufbau:

Byte \$00	Langwortadresse der Startadresse des gedachten Bildschirms (obere linke Ecke). Wird hier eine 0 eingetragen, so gilt das als Flag dafür, daß der aktuelle Bildschirm gemeint ist. Alle folgenden Parameter werden dann ignoriert.
Byte \$04	Breite des "Bildschirms" in Pixel (Wort)
Byte \$06	Höhe des "Bildschirms" in Pixel (Wort)
Byte \$08	Breite des "Bildschirms" in Worten (= Breite in Punkten durch 16) (Wort)
Byte \$0A	Hier wird ein Flag gesetzt: =0: Format ist gerätespezifisch, =1: Format ist Standard. Der Eintrag hat Wortlänge
Byte \$0C	Anzahl der Farbenen des "Bildschirms" (Wort)
Byte \$0E	Frei für zukünftige Funktionen (Wort)
Byte \$10	Frei für zukünftige Funktionen (Wort)
Byte \$12	Frei für zukünftige Funktionen (Wort)

In C existiert bereits eine Strukturdefinition im Headerfile GEMDEFS.H, das den folgenden Aufbau besitzt:

```
typedef struct fdbstr
{
    long fd_addr;
    int fd_w;
    int fd_h;
    int fd_wdwidth;
    int fd_stand;
    int fd_nplanes;
    int fd_r1;
    int fd_r2;
    int fd_r3;
} FDB;
```

Damit können Sie also ganz einfach mittels

```
FDB eigener_block;
```

einen eigenen Block definieren. Wollen sie GEMDEFS.H, das Sie ja mit #INCLUDE einbinden müssen, nicht verwenden, so müssen Sie die Strukturdefinition selbst vornehmen (s. C-Beispiel). In BASIC oder Assembler müssen Sie natürlich entsprechende Speicherbereiche reservieren und mit den notwendigen Daten belegen. Die drei folgenden Funktionen müssen Sie im Zusammenhang mit den Rastern kennen:

```
COPY RASTER, OPAQUE (109 VDI)
COPY RASTER, TRANSPARENT (121 VDI)
TRANSFORM FORM (110 VDI)
```

Erst wenn Sie für Ziel und für Quelle einen Block definiert haben, dann dürfen Sie die Rasterbefehle anspringen. (Unter bestimmten Umständen - die Teilraster dürfen sich nicht überschneiden - können Sie Quelle und Ziel auch identisch halten.)

Nachdem wir einen solchen Bildschirm definiert haben, geben wir in den beiden ersten Befehlen die Eckkoordinaten der Teilraster des Quell- und Zielspeicherbereichs an, mit denen wir operieren wollen. Neben diesen Koordinaten müssen wir der Funktion natürlich noch mitteilen, an welcher Speicherstelle unsere MFDBs liegen. Die Langwortadressen dieser beiden Speicherblöcke werden im Kontroll-Array angegeben.

Beachten Sie bitte hier und bei allen Ihren Programmen unbedingt den folgenden Hinweis, wenn Sie nicht unliebsame Programmabstürze produzieren wollen:

Die Raster-Funktionen unterliegen keinem Clip-Fenster. Weder der VDI-Befehl SET CLIPPING RECTANGLE wird berücksichtigt, noch werden die Rastergrenzen eingehalten, falls Sie beim Raster-Funktionsaufruf Koordinaten angeben, die eigentlich viel zu groß für Ihre Definition Ihres virtuellen Bildschirms sind. Dies kann im harmlosen Fall (zu große x-Koordinaten) dazu führen, daß das Raster auf der anderen Seite (rechts oder links) wieder im (virtuellen) Bildschirm erscheint. Geben Sie aber zu große y-Koordinaten (oder bei weitem zu große x-Koordinaten) beim Aufruf der verschiedenen Raster-Funktionen an, so wird der hinter dem Raster liegende Speicherbereich angesprochen und evtl. zerstört, Speicher also, der eigentlich gar nicht mehr zum Raster gehört und wichtige Daten enthalten kann! Auf kurz oder lang wird es dann sicher zu einem Rechnerabsturz kommen. Arbeiten Sie mit dem normalen Bildschirm, hinter dessen Speicher z.B. eine RAM-Disk liegt (ein beliebiger Platz), dann sind ruck zuck Ihre Daten auf dieser RAM-Disk unwiederbringbar verloren. Seien Sie also vorsichtig bei der Verwendung der Raster-Befehle, und speichern Sie vor Ihrem Test-Programmstart stets wichtige Daten auf Disk! Eine Lösung des Problems ist die Programmierung einer eigenen Clip-Routine für Raster-Funktionen.

Mit der Funktion COPY RASTER, OPAQUE können wir zwei Raster q und z miteinander verknüpfen und im Zielraster z ablegen. Dies geht allerdings nur mit Rastern, die das Format des aktuellen Gerätes besitzen (=gerätespezifisch), d.h. wenn Sie gerade im hochaufgelösten Modus arbeiten, so können auch nur hochaufgelöste Raster (eine Farbebene) verknüpft werden. Ist jedoch gerade der niedrig aufgelöste Modus eingestellt, so muß Ihr Raster auch den niedrig aufgelösten Bildschirmspeicheraufbau besitzen (vier Farbenen ineinander). Ist das nicht der Fall, so müssen Sie ein oder beide vollständige Raster (also jeweils den gesamten gedachten Bildschirm!) zunächst einmal mit der Funktion TRANSFORM FORM (s.u.) in das gerätespezifische Format umwandeln. Erst dann dürfen Sie COPY RASTER, OPAQUE anwenden.

Dann aber stehen Ihnen insgesamt 16 verschiedene Verknüpfungsarten zwischen den beiden Rastern zur Verfügung. Dabei kann es durchaus vorkommen (Modi 3, 5, 10 und 12), daß eigentlich nur ein Raster für das Ergebnistraster hinzugezogen wird. Dann wird z.B. das Quellraster nur in das Zielraster kopiert (Modus 3) oder das Zielraster ohne Beteiligung des Quellrasters invertiert (Modus 10) etc. Modi 0 und 15 ignorieren gar beide Raster. Modus 0 löscht lediglich das Zielraster, während Modus 15 alle Bits in ihm setzt.

Die Verknüpfungen finden dabei bit-weise statt. Im monochromen Modus sind natürlich gesetzte oder gelöschte Bits identisch mit gesetzten oder gelöschten Punkten. Eine Verknüpfung zweier Punkte wird also immer wieder nur einen gesetzten oder gelöschten Punkt hervorbringen. In den beiden

Farbmodi jedoch gehören ja jeweils zwei bzw. vier Bits zusammen, die die Farbe eines Punktes angeben. Wenn nun also solche z.B. vier Bits (für die Farbe eines Punktes) aus dem Quellraster mit den entsprechenden vier Bits aus dem Zielraster verknüpft werden, so kommt es nicht nur dazu, daß dieser Punkt entweder gesetzt oder gelöscht wird (monochromer Modus). Vielmehr findet hier dann ein Farbwechsel statt, da jedes einzelne der vier Bits mit dem entsprechenden Bit der vier anderen verknüpft wird. Dieses Phänomen kennen wir bereits vom Zeichenmodus 3 (XOR-Modus). Vielleicht sollten Sie dort noch einmal nachlesen. Hier wird allerdings nicht nur im XOR-Modus verknüpft (Raster-Modus 6). Hier haben Sie noch eine ganze Palette anderer Verknüpfungsarten.

Eine Möglichkeit, Raster verschiedener Formate zu bearbeiten, erschließt eine Variation des COPY RASTER-Befehls: COPY RASTER, TRANSPARENT. Hier kann als Quelle ausschließlich ein monochromes Raster verwendet werden. Das Zielraster hat dann stets das momentan aktuelle, also gerätespezifische Format (kann also durchaus auch monochrom sein!).

Dafür existieren allerdings nur vier verschiedene Verknüpfungsformen. Wir kennen sie bereits vom Zeichenmodus. Es sind die bekannten

- Replace
- Transparent
- XOR
- Invers Transparent

Während wir die beiden Formen Replace und XOR bei zwei identischen Rasterformaten auch mit dem Befehl COPY RASTER, OPAQUE realisieren können (Modi 3 und 6), stellen die beiden anderen Modi grundsätzlich unterschiedliche Verknüpfungsarten dar (außer im monochromen Modus, wo sie durch OPAQUE-Modi 7 und 11 nachgebildet werden können).

Da das Quellraster ja monochrom ist, müssen in der Funktion die Farbindizes für gesetzte (Bits=1) und gelöschte (Bits=0) Punkte angegeben werden. Das monochrome Raster wird mit diesen Informationen intern quasi auf das gerätespezifische Format "erweitert".

Die Funktion COPY RASTER, TRANSPARENT wird man wohl stets dann verwenden, wenn z.B. ein festgelegtes monochromes Raster in einem Programm, das in allen drei Auflösungsstufen funktionieren soll, beispielsweise in den Bildschirm übertragen werden soll (Sprites oder ähnliche feste Objekte).

Als eine letzte Rasterfunktion stellt die Funktion TRANSFORM FORM die Möglichkeit zur Verfügung, ganze "gedachte Bildschirme", also ganze Raster

von einem Bildschirmformat in ein anderes umzuwandeln. Sie können mit dieser Funktion beispielsweise monochrome in farbige Bilder und umgekehrt verwandeln. Selbstverständlich müssen Sie dabei die unterschiedlichen Punktezahlen berücksichtigen. So werden Sie niemals ein vollständiges monochromes Bild in Gänze auf einen farbigen Bildschirm bringen. Ein farbiges Bild wird stets nur einen Teil des monochromen Bildschirms einnehmen. Im Speicher erhalten Sie jedoch durch diese Funktion das vollständige Bild. Die Umwandlung eines niedrig aufgelösten Bildes in ein Bild mittlerer Auflösung ist allerdings nicht direkt möglich.

Damit dieser Befehl ordnungsgemäß arbeitet, müssen Sie in den MFDBs das Format-Flag richtig setzen. Unter "Standard" ist stets das monochrome Bild zu verstehen.

Um Ihnen den gesamten Zusammenhang etwas plastischer erscheinen zu lassen, fügen wir an dieser Stelle wieder ein Beispielprogramm an. Lassen Sie sich überraschen:

```
' *****
' **                               **
' ** Rasteroperationen           **
' **                               **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
handle%=DPEEK(VDIBASE+40)
'
RESERVE 150000                                !Reserviere 150000 Bytes
'                                              !für GFA-BASIC; aller anderer
'                                              !Speicher: freigegeben
psrcmfdb%=GEMDOS(&H48,L:20)                  !20 Bytes für Quellrasterblock
pdesmfdb%=GEMDOS(&H48,L:20)                  !20 Bytes für Zielrasterblock
screen%=GEMDOS(&H48,L:32000)                 !32000 Bytes für Bildzwischenspeicher
'
' Definiere Quellrasterblock (MFBD):
LPOKE psrcmfdb%,screen%                      !Adresse Bildzwischenspeicher
DPOKE psrcmfdb%+4,xauf1%+1                   !Gesamt-Rasterbreite in Punkten
DPOKE psrcmfdb%+6,yauf1%+1                   !Gesamt-Rasterhöhe in Punkten
DPOKE psrcmfdb%+8,(xauf1%+1)/16              !Gesamt-Rasterbreite in Wörtern
DPOKE psrcmfdb%+10,1                          !Flag für standard/gerätespezifisch
DPOKE psrcmfdb%+12,4/2^XBIOS(4)              !Anzahl der Farbebenen
DPOKE psrcmfdb%+14,1                          !zukünftige Belegungen:
DPOKE psrcmfdb%+16,1
DPOKE psrcmfdb%+18,1
'
' Definiere Zielrasterblock (MFDB):
```



```

LPOKE pdesmfdb%,0           !Flag für Bildschirmspeicher
' Alle anderen MFDB-Belegungen entfallen:
' DPOKE pdesmfdb%+4,xauf1%+1
' DPOKE pdesmfdb%+6,yauf1%+1
' DPOKE pdesmfdb%+8,(xauf1%+1)/16
' DPOKE pdesmfdb%+10,1
' DPOKE pdesmfdb%+12,4/2^XBIOS(4)
' DPOKE pdesmfdb%+14,1
' DPOKE pdesmfdb%+16,1
' DPOKE pdesmfdb%+18,1
,
DIM pxyarray%(7)
,
REPEAT
  GRAPHMODE 1               !Replace-Modus ein
  DEFLINE 1                 !Durchgezogene Linien
  ,
  ' Hand-Zeichenmodus:
  REPEAT
    MOUSE x1%,y1%,k%       !Mausdaten holen
    IF k%=1                 !falls linker Mausknopf
      DRAW x%,y% TO x1%,y1% !Zeichne Linie vom letzten Punkt
    ENDIF
    x%=x1%                 !merke Koordinaten
    y%=y1%
  UNTIL k%=2               !bis rechter Mausknopf
  ,
  ' Kopiere gesamten Bildschirm in Zwischenspeicher:
  pxyarray%(0)=0           !obere linke Koordinate
  pxyarray%(1)=0
  pxyarray%(2)=xauf1%      !untere rechte Koordinate
  pxyarray%(3)=yauf1%
  pxyarray%(4)=0
  pxyarray%(5)=0
  pxyarray%(6)=xauf1%
  pxyarray%(7)=yauf1%
  ' Kopiere "des" (Ziel) nach "scr" (Quelle):
  @vrcopyfm(handle%,3,*pxyarray%(),pdesmfdb%,psrcmfdb%)
  ,
  ' Alternativer BASIC-Befehl:
  ' (XBIOS(2) bringt akt. Bildschirmadresse)
  ' BMOVE XBIOS(2),screen%,32000
  ,
  ' zu kopierenden Bereich umrahmen:
  x1%=MOUSEX
  y1%=MOUSEY
  x2%=x1%
  y2%=y1%
  ,
  DEFLINE 4                 !Punkt-Strich-Linie
  GRAPHMODE 3              !XOR-Modus
  ,
  REPEAT

```

```

x%=x2%                                !Koordinaten merken
y%=y2%
BOX x1%,y1%,x2%,y2%                  !Bereich umrahmen
REPEAT
    MOUSE x2%,y2%,k%                 !Auf Mausereignis warten
    UNTIL x2%<>x% OR y2%<>y% OR k%<>2
    BOX x1%,y1%,x%,y%                 !Rechteck wieder löschen
UNTIL k%<>2                            !bis Mausknopf losgelassen
,
DEFINE 1
GRAPHMODE 1
,
' Umrahmten Block Mauszeiger folgen lassen:
pxyarray%(0)=x1%                      !Quellkoordinaten im
pxyarray%(1)=y1%                      !Bild-Zwischenspeicher
pxyarray%(2)=x2%
pxyarray%(3)=y2%

x3%=x1%
y3%=y1%
,
REPEAT
    x%=x3%
    y%=y3%
    pxyarray%(4)=x3%                  !Zielkoordinaten im
    pxyarray%(5)=y3%                  !Bildschirmspeicher
    pxyarray%(6)=x3%+ABS(x2%-x1%)
    pxyarray%(7)=y3%+ABS(y2%-y1%)
    ' Bereich in Bildschirm kopieren (im XOR-Modus 6):
    @vro cpyfm(handle%,6,*pxyarray%(),psrcmfdb%,pdesmfdb%)
    REPEAT
        MOUSE x3%,y3%,k%             !Auf Mausereignis warten
    UNTIL x3%<>x% OR y3%<>y% OR k%<>0
    ' Bereich wieder löschen (XOR):
    @vro cpyfm(handle%,6,*pxyarray%(),psrcmfdb%,pdesmfdb%)
UNTIL k%<>0                            !bis Mausknopf betätigt
,
' Bereich zeichnen:
REPEAT
    MOUSE x%,y%,k%                    !Mauskoordinaten holen
    pxyarray%(4)=x%                   !Zielkoordinaten im
    pxyarray%(5)=y%                   !Bildschirm setzen
    pxyarray%(6)=x%+ABS(x2%-x1%)
    pxyarray%(7)=y%+ABS(y2%-y1%)
    ' Bereich durch OR (7) zeichnen:
    @vro cpyfm(handle%,7,*pxyarray%(),psrcmfdb%,pdesmfdb%)
UNTIL k%=0                            !bis Mausknopf los
,
UNTIL INKEY$<>" "                     !Ende bei Taste
,
,
' Bild hochscrollen:
' Definiere Quellrasterblock (MFBD):

```

```

LPOKE psrcmfd%,0          !Flag für Bildschirmspeicher
'
' Definiere Zielrasterblock (MFDB):
LPOKE pdesmfd%,0          !Flag für Bildschirmspeicher
'
DEFINE 1
DRAW 0,yaufl% TO xaufl%,yaufl% !Unterste Zeile anfärben
'
' Kopiere gesamten Bildschirm eine Zeile höher:
' Kopiere (Replace):
FOR x%=0 TO yaufl%
  pxyarray%(0)=0          !Quelle: obere linke Koordinate
  pxyarray%(1)=1
  pxyarray%(2)=xaufl%     !Quelle: untere rechte Koordinate
  pxyarray%(3)=yaufl%-x%
  pxyarray%(4)=0          !Ziel: obere linke Koordinate
  pxyarray%(5)=0
  pxyarray%(6)=xaufl%     !Ziel: untere rechte Koordinate
  pxyarray%(7)=yaufl%-x%-1
  VSYNC
  @vro cpyfm(handle%,3,*pxyarray%(),psrcmfd%,pdesmfd%)
NEXT x%
'
'
' Reservierte Speicherbereiche wieder freigeben:
VOID GEMDOS(&H49,L:screen%) !Bild-Zwischenspeicher
VOID GEMDOS(&H49,L:psrcmfd%) !Quell-MFDB
VOID GEMDOS(&H49,L:pdesmfd%) !Ziel-MFDB
EDIT
'
'
' COPY RASTER, OPAQUE
PROCEDURE vro cpyfm(handle%,wr_mode%,p_pxyarray%,psrcmfd%,pdesmfd%)
  DPOKE CONTRL+2,4
  DPOKE CONTRL+6,1
  DPOKE CONTRL+12,handle%
  LPOKE CONTRL+14,psrcmfd%
  LPOKE CONTRL+18,pdesmfd%
  DPOKE INTIN,wr_mode%
  SWAP *p_pxyarray%,xi%()
  FOR ii%=0 TO 7
    DPOKE PTSIN+2*ii%,xi%(ii%)
  NEXT ii%
  SWAP *p_pxyarray%,xi%()
  VDISYS 109
RETURN

```

Hier haben wir eine schöne, typische Anwendung der Rasterbefehle. Nach dem Starten dieses Programms können Sie mit der Maus ein beliebiges Bild auf den Bildschirm zeichnen, solange Sie den linken Mausknopf gedrückt halten. Lassen Sie den Mausknopf wieder los, hört der Rechner auch auf zu zeichnen. Betätigen Sie ihn wieder, geht es weiter.

Sind Sie mit Ihrem Werk zufrieden, dann positionieren Sie den Mauszeiger auf einen beliebigen Punkt. Wenn Sie nämlich nun die rechte Maustaste gedrückt halten und die Maus bewegen, dann ziehen Sie ein Rechteck beliebiger Größe auf. Mit diesem Rechteck legen Sie einen Bereich fest, den Sie später duplizieren werden.

Lassen Sie die rechte Maustaste wieder los, dann erscheint sofort eine Kopie des von Ihnen eingegrenzten Bildschirmbereiches auf dem Bildschirm. Bewegen Sie die Maus, so folgt diese Kopie dem Mauszeiger. Sobald Sie nun auf den linken Mausknopf drücken, so wird besagter Bereich an den jeweils aktuellen Positionen, Ihrem Mauszeiger folgend, fest auf den Bildschirm gezeichnet.

Wenn Sie den linken Mausknopf nun wieder loslassen und nicht zwischendurch irgendwann einmal eine Taste gedrückt haben, dann fängt alles wieder von vorne an, Sie können also wieder mit Ihrer Maus zeichnen. Zum guten Schluß wird der gesamte Bildschirm nach oben wie ein Stück Papier fortgeschrollt.

Interessanter noch als die Programmbedienung ist die Realisierung auf Programmebene. Das Prinzip des Programms ist eigentlich recht einfach. Als Ziel- bzw. Quellraster dienen zum einen der aktuelle 32000 Byte große Bildschirmspeicher. Zum anderen wird ein imaginärer zweiter Bildschirmspeicher angelegt (ebenfalls 32000 Bytes groß). Für diesen imaginären Bildschirm muß im RAM Speicherplatz reserviert werden. Dies geschieht mit der Funktion "Malloc()" des GEMDOS (Funktionsnummer: \$48). Damit stellt uns das Betriebssystem diesen Speicher zur Verfügung und liefert uns die Startadresse des gewünschten Speicherbereiches zurück (Variable: screen%).

In GFA-BASIC muß dem Betriebssystem jedoch vorher überhaupt einmal Speicher zurückgegeben werden, da normalerweise sämtlicher zur Verfügung stehender Rechnerspeicher von GFA-BASIC besetzt ist. Dies wird mit dem GFA-Befehl RESERVE realisiert. RESERVE reserviert GFA-BASIC eine bestimmte Anzahl von Bytes, der Rest wird dem Betriebssystem zurückgegeben.

Da neben dem virtuellen Bildschirm noch die beiden MFDBs mit je 20 Bytes angelegt werden müssen, wird auch für sie - ebenfalls mit "Malloc()" - RAM reserviert. Dann übergibt das Programm diesen beiden Bereichen die notwendigen Informationen über die Raster. Da der virtuelle Bildschirm exakt so aufgebaut sein soll, wie der aktuelle Bildschirm, wird hierzu die aktuelle Bildschirmauflösung (xauf%, yauf%) usw. herangezogen. Die Funktion XBIOS(4) liefert einen Wert, der die jeweilige Auflösung bestimmt:

=0: Niedrige Auflösung

=1: Mittlere Auflösung

=2: Hohe Auflösung

Durch die Formel:

$$4 / 2^{XBIO(4)}$$

wird die Anzahl der daraus resultierenden Farbebenen berechnet. (Rechnen Sie einmal nach!)

Der Zielrasterblock soll der aktuelle Bildschirmspeicher sein. Also schreiben wir einfach für die Adresse das Flag 0, und schon wird stets der Bildschirm angesprochen, egal, wo er zur Zeit liegt.

Nach diesen Vorbereitungen geht es richtig ans Werk. Die Routine zum Handzeichnen sollte relativ einfach zu durchschauen sein. Es wird einfach ständig die Maus abgefragt und, falls der linke Knopf gedrückt sein sollte, vom letzten Mauspunkt zum jetzigen eine Linie gezogen. Das geschieht so lange, bis $k\%=2$ ist, also der rechte Mausknopf betätigt wurde.

Dann kopiert das Programm mit dem Befehl COPY RASTER, OPAQUE den gesamten Bildschirmspeicher in den virtuellen Screen. Damit haben wir zwei Ausfertigungen des Bildschirms im Speicher.

Jetzt kann der Benutzer seinen Kopierbereich mit dem oben beschriebenen Rechteck eingrenzen. Die hierzu verwendete Rechenvorschrift sollte Ihnen bekannt vorkommen, da wir sie vom Prinzip her bereits etliche Male in diesem Buch vorgestellt haben.

Jetzt wird es spannend. Nachdem der Benutzer nämlich sein Rechteck festgelegt hat, werden dessen Koordinaten für die Quellkoordinaten der COPY RASTER-Funktion verwendet. Die Zielkoordinaten stammen von der aktuellen Mausposition (und natürlich von der Rechteckgröße). Mit diesen Informationen kann nun der Bereich des eingegrenzten Rechteckes aus dem virtuellen Bildschirmspeicher an eine beliebige, hier von der Mausposition festgelegte Stelle auf dem richtigen Bildschirm erscheinen. Das Ganze wird in Modus 6 (XOR) bewerkstelligt, damit der Bereich jederzeit bei Mausbewegungen wieder gelöscht und woanders hinkopiert werden kann.

Wird nun die linke Maustaste betätigt, dann zeichnet das Programm in Modus 7 (ODER-Verknüpfung), wodurch das Ergebnis unlöschar auf dem Bildschirm erscheint.


```

/*****
/*      Terminiere Applikation      */
*****/
aes_exit()
{
    gemdos(0x1);           /* Auf Taste warten */
    appl_exit();
}

/*****
/*      öffne virtuelle Workstation    */
*****/
open_vwork()
{
    int i;

    for (i=0; i<10; work_in[i++]=1);
    work_in[10] = 2;
    handle = gem_handle;
    v_opnvwk(work_in, &handle, work_out);
}

/*****
/*      Schließe virtuelle Workstation */
*****/

#define close_vwork()  v_clsvwk(handle);

/*****
/*      Hauptprogramm      */
*****/
/* Typ- und Strukturdefinition für MFDB: */
typedef struct fdbstr {
    long  fd_addr;      /* Rasteradresse          */
    int   fd_w;         /* Rasterbreite in Punkten */
    int   fd_h;         /* Rasterhöhe in Punkten  */
    int   fd_wdwidth;   /* Rasterbreite in Wort   */
    int   fd_stand;     /* Standard-Flag          */
    int   fd_nplanes;   /* Anzahl der Farbebenen  */
    int   fd_r1;        /* zukünftige Erweiterungen */
    int   fd_r2;
    int   fd_r3;
    } FDB;

int  xauf1,           /* Neuer Typ: FDB          */
     yauf1,           /* x-Bildschirmauflösung  */
     quell_breite = 5*16, /* y-Bildschirmauflösung  */
     quell_hoehe = 18,   /* Quellrasterbreite      */
     wr_mode;           /* Quellrasterhöhe        */
/* 80x18 Quellmusterdefinition für COPY RASTER, TRANSPARENT: */
int  raster[18][5] = {
    {0xff00, 0xff00, 0xff00, 0xff00, 0xff00},
    {0xff00, 0xff00, 0xff00, 0xff00, 0xff00},
    {0xff00, 0xff00, 0xff00, 0xff00, 0xff00},

```

```

        {0xff00, 0xff00, 0xff00, 0xff00, 0xff00},
        {0xff00, 0xff00, 0xff00, 0xff00, 0xff00},
        {0xff00, 0xff00, 0xff00, 0xff00, 0xff00},
        {0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0},
        {0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0},
        {0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0},
        {0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0},
        {0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0},
        {0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0},
        {0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0, 0x0ff0},
        {0x00ff, 0x00ff, 0x00ff, 0x00ff, 0x00ff},
        {0x00ff, 0x00ff, 0x00ff, 0x00ff, 0x00ff},
        {0x00ff, 0x00ff, 0x00ff, 0x00ff, 0x00ff},
        {0x00ff, 0x00ff, 0x00ff, 0x00ff, 0x00ff},
        {0x00ff, 0x00ff, 0x00ff, 0x00ff, 0x00ff},
        {0x00ff, 0x00ff, 0x00ff, 0x00ff, 0x00ff},
    };

FDB  psrcMFDB;                /* Quellrasterstruktur */
FDB  pdesMFDB = {0,0,0,0,0,0,0,0}; /* Zielrasterstruktur: */
                                     /* Adresse=0 für Bildschirm*/
int  color_index[2] = {2,1};    /* Farbindizes für 0 und 1 */

main()
{
    int x,y;
    /* Quellrasterstruktur Wertebelegung: */
    psrcMFDB.fd_addr = (long)&raster[0][0];
    psrcMFDB.fd_w    = quell_breite;
    psrcMFDB.fd_h    = quell_hoehe;
    psrcMFDB.fd_wdwidth = quell_breite/16;
    psrcMFDB.fd_stand = 0;
    psrcMFDB.fd_nplanes = 1;
    psrcMFDB.fd_r1     = 1;
    psrcMFDB.fd_r2     = 1;
    psrcMFDB.fd_r3     = 1;
    aes_init();
    open vwork();          /* Initialisierung */
    xaupl = work_out[0];   /* Auflösung feststellen */
    yaupl = work_out[1];
    v_clrwk(handle);       /* Bildschirm löschen */
    vsf_color(handle, 3);  /* Füllmusterfarbe für BAR */
    vsf_interior(handle, 2); /* Füllmuster für BAR */
    vsf_style(handle, 9);
    /* Zeichenschleife: */
    for (wr mode=1, y=0; y<2; y++)
        for (x=0; x<2; x++)
        {
            /* Gefülltes Rechteck auf den Bildschirm: */
            pxyarray[0] = x*xaupl/3 + 20;
            pxyarray[1] = y*yaupl/4 + 20;
            pxyarray[2] = pxyarray[0] + xaupl/3-5;
            pxyarray[3] = pxyarray[1] + yaupl/4-5;
            v_bar(handle, pxyarray);
            /* Quellraster in Rechteck kopieren: */

```



```

/* Zielkoordinaten festlegen: */
pxyarray[4] = pxyarray[0];
pxyarray[5] = pxyarray[1];
pxyarray[6] = pxyarray[0] + quell_breite-1;
pxyarray[7] = pxyarray[1] + quell_hoehe-1;
/* Quellkoordinaten festlegen: */
pxyarray[0] = 0;
pxyarray[1] = 0;
pxyarray[2] = quell_breite-1;
pxyarray[3] = quell_hoehe-1;
/* Raster in Bildschirm kopieren */
/* und Schreibmodus hochzählen: */
vrt_cpyfm (handle, wr_mode++, pxyarray,
           &psrcMFDB, &pdscMFDB, color_index);
}
close_vwork(); /* Programmende */
aes_exit();
}

```

Beachten Sie bitte bei diesem C-Programm, daß Sie sich die Typ- und Struktur-Definition FDB sparen können, falls Sie mit

```
#include <gemdefs.h>
```

unter anderem diese Definition eingebunden haben.

Das Programm zeigt Ihnen die Wirkung der vier Raster-Modi der Funktion COPY RASTER, TRANSPARENT. Dazu werden vier mit einem Muster gefüllte Rechtecke auf den Bildschirm gebracht und von dem in dem Array raster[][] definierten monochromen Raster überlagert.

Hierzu werden nach den Variablendefinitionen die MFDBs eingestellt. Ziel-MFDB ist der Bildschirm (s. Deklaration). Der Quell-MFDB wird aus der Array-Adresse, der Rasterhöhe und Breite gespeist. Der übrige Programmverlauf ist recht einfach und direkt aus der Programmfunktion bzw. den eingebauten Kommentaren ersichtlich.

Obwohl die GFA-Implementierungen der Raster-Funktionen bei weitem nicht so flexibel sind wie die originalen VDI-Routinen, seien sie hier kurz angeführt:

SGET stringvar\$ (GFA-BASIC-Befehl)

SGET a\$

Kopieren des gesamten Bildschirms in eine String-Variable

stringvar\$: String-Variable, in die kopiert wird

SPUT stringvar\$ (GFA-BASIC-Befehl)

SPUT a\$

Kopieren einer String-Variable in den Bildschirmspeicher

stringvar\$: String-Variable, aus der kopiert wird

GET x1,y1,x2,y2,stringvar\$ (GFA-BASIC-Befehl)

Kopieren eines durch ein Rechteck umgrenzten Bildschirmbereichs in eine String-Variable

x1: x-Koord. Eckpunkt Rechteck oben links
y1: y-Koord. Eckpunkt Rechteck oben links
x2: x-Koord. Eckpunkt Rechteck unten rechts
y2: y-Koord. Eckpunkt Rechteck unten rechts
stringvar\$: Stringvariable, in die kopiert wird

PUT x1,y1,stringvar\$(modus) (GFA-BASIC-Befehl)

Verknüpfen eines durch GET gefüllten Strings mit einem Bildschirmteil

x1: x-Koord. Eckpunkt Bereich oben links
y1: y-Koord. Eckpunkt Bereich oben links
stringvar\$: Stringvariable, aus der kopiert wird
modus: Verknüpfungsmodus (0-15)
 Unter modus verstehen wir die 16 Modi, wie wir sie bei COPY
 RASTER, OPAQUE kennengelernt haben.

Wie Sie sehen, beziehen sich alle vier Befehle lediglich auf den Bildschirm. Dafür entfällt aber auch die Arbeit, MFDBs einzurichten.

15.9 Sprites auch auf dem Atari ST

Wir kommen zu einem höchst interessanten Kapitel, das Sie vielleicht bereits von anderen Computern her kennen und bisher auf dem Atari ST schmerzlich vermisst haben: Sprites. Sprites sind vordefinierte, auf dem Bildschirm frei bewegliche Figuren. Strenggenommen versteht man unter Sprites hardwaremäßig auf den Bildschirm gebrachte Objekte. Da unser Atari solche Objekte

von Haus aus nicht kennt, müssen wir sie programmieren. Softwaremäßige Sprites können auf verschiedene Art und Weise programmiert und verwaltet werden. Wir werden hier zwei vorstellen.

Das Problem beim Zeichnen von Objekten auf den Bildschirm, die jederzeit wieder fortbewegt werden sollen, ist die Hintergrundrettung, d.h. die Rettung desjenigen Teils der Grafik, der von diesem Sprite überdeckt wird. Wird das Sprite nämlich wieder an eine andere Stelle des Bildschirms gebracht, so muß der verbleibende Grafikteil restauriert werden.

Relativ unproblematisch ist das Ganze, wenn lediglich ein einziges Sprite gleichzeitig auf den Bildschirm gebracht werden soll. In diesem Fall kann der vom Sprite überdeckte Hintergrund (meist ein rechteckiger Bereich) mit Hilfe der bekannten Raster-Funktionen zwischengespeichert und später wieder an diese Stelle zurückkopiert werden.

Wir kennen diesen Vorgang vom normalen Mauszeiger, der ein solches Sprite darstellt. Probleme gibt es erst, wenn neue Grafik gezeichnet werden soll. Werden dann nämlich Punkte dorthin gezeichnet, wo zur Zeit das Sprite liegt, dann sind sie beim nächsten Restaurieren des Bildschirms wieder verloren. Aus diesem Grunde muß der Mauszeiger auch bei jeder Bildschirmoperation aus- und danach wieder eingeschaltet werden. Erkennen können Sie diesen Vorgang an dem Flackern des Mauszeigers z.B. bei Fensterbewegungen etc.

Wollen wir jedoch ein zweites Sprite gleichzeitig auf dem Bildschirm darstellen, so tauchen ernste Schwierigkeiten auf. Denn sollten sich einmal die beiden Sprites überlappen, dann werden nicht nur Hintergrundteile gerettet, sondern Teile des zweiten Sprites ebenfalls als Hintergrund interpretiert. Die Folge: Bei der nächsten Sprite-Bewegung bleiben Sprite-Teile am Bildschirm stehen.

Vermeiden kann man dies entweder durch Vermeidung von Sprite-Überschneidungen oder durch das Löschen und Neuzeichnen jedes einzelnen Sprites bei jeder Sprite-Bewegung (auch wenn sich nur ein einziges Sprite bewegt hat). Das geht natürlich (ohne Blitter) auf die Dauer auf die Rechenzeit. Dabei ist das Einhalten einer strengen Zeichenreihenfolge notwendig. Die Sprites müßten durchnummeriert und ihre Koordinaten zwischengespeichert werden. Sollen alle (oder nur manche) Sprites bewegt werden, so müssen zunächst die zwischengespeicherten Koordinaten aller Sprites geändert werden. Dann erst sollten die Sprites zunächst in einer festgelegten Reihenfolge gelöscht und in der umgekehrten Reihenfolge wieder gezeichnet werden. Sie erkennen den enormen Aufwand. Eine andere Möglichkeit bedient sich des XOR-Modus. Sofern alle Sprites im XOR-Modus gezeichnet werden, können Sie jederzeit durch die selbe Operation wieder vom Bildschirm gelöscht werden, ohne daß Sie sich darum kümmern müssen, ob sich noch weitere Sprites

auf dem Bildschirm befinden oder welche Hintergrundgrafik bereits existiert. Den Nachteil dieser Methode werden Sie bei Hintergründen mit feinen Mustern erkennen. Hier verschwimmen die Sprites bis hin zur Unkenntlichkeit (s. C-Beispielprogramm).

GFA-BASIC bietet ebenfalls die Möglichkeit der Sprite-Programmierung an. Da es sich bei diesen Sprites jedoch nur um den zweckentfremdeten Mauszeiger handelt, treten bei mehreren unabhängigen Objekten die oben beschriebenen Probleme auf. Ein Sprite hat stets und unveränderbar die Größe 16x16 Punkte. Zur Programmierung dieser Sprites steht der folgende Befehl zur Verfügung:

SPRITE str\$,x,y (GFA-BASIC-Befehl)

SPRITE str\$

- a) Setzen bzw. Bewegen eines Sprites
- b) Löschen des zuletzt gezeichneten Sprites

str\$: Definitionsstring:

Wort 1: x-Koord. des Aktionspunktes relativ zur oberen linken Sprite-Ecke (1-16)

Wort 2: y-Koord. des Aktionspunktes relativ zur oberen linken Sprite-Ecke (1-16)

Wort 3: Zeichenmodus: 0=Normal/1=XOR

Wort 4: Maskenfarbindex

Wort 5: Sprite-Form-Farbindex

Wörter 6-37: Abwechselnd eine Zeile der Maske und eine Zeile der Form, Start mit der ersten Maskenzeile

x: x-Koordinate Sprite

y: y-Koordinate Sprite

Wird dieser Befehl mit Koordinatenangaben eingegeben, so setzt er an die entsprechende Stelle ein Sprite, dessen Aussehen durch den Definitions-String festgelegt wird. Werden die Koordinaten fortgelassen, so wird das zuvor gezeichnete Sprite wieder gelöscht.

In dem Definitions-String geben Sie einige Informationen über das zu zeichnende Sprite an. Dabei legen Sie in den ersten beiden String-Worten (vier Bytes, also vier Zeichen) den sogenannten Aktionspunkt des Sprites fest. Dieser Aktionspunkt stellt - ähnlich wie beim ALIGNMENT-Befehl zur Textdarstellung - denjenigen Punkt im Sprite dar, für den die eigentlichen Sprite-Koordinaten gelten.

Im dritten Wort steht der Zeichenmodus für die Mausform, alsdann folgen die jeweiligen Farbindizes für Sprite-Maske und Sprite-Form. Um das zu verstehen, müssen wir uns ein wenig mit dem Sprite-Aufbau beschäftigen.

Wie gesagt ist ein Sprite 16x16 Punkte groß. Damit paßt seine Definition (Bit=0 für Punkt nicht gesetzt, Bit=1 für Punkt gesetzt) in 16 Wörter. Neben der so (ähnlich einem Füllmuster) anzugebenden Sprite-Form existiert noch eine sogenannte Sprite-Maske. Diese Maske hat noch einmal den gleichen Aufbau wie die eigentliche Form. Wenn Sie jedoch in dieser Maske ein Bit (= ein Punkt) setzen, so wird an dieser Stelle später ein Punkt in der Farbe der Maske gesetzt. Damit haben Sie die Möglichkeit, anzugeben, ob ein Punkt des Sprites durchsichtig, gesetzt oder gelöscht sein soll:

Bit in Form	Bit in Maske	Ergebnis
0	0	Punkt durchsichtig
0	1	Punkt in Maskenfarbe
1	0	Punkt in Formfarbe
1	1	Punkt in Formfarbe

Die Form überdeckt also die Sprite-Maske. Normalerweise (monochrom) hat die Maske die Farbe des Hintergrundes, während die Form die Zeichenfarbe annimmt. Damit wäre an jedem Maskenpunkt ein Punkt gelöscht, an jedem Formpunkt ein Punkt gesetzt. Das muß allerdings nicht unbedingt so sein. Jetzt können Sie sicher verstehen, was mit Farbindizes der Maske und Form gemeint ist. Die Definition der Maske und der Form werden in den Wörtern 6 bis 37 vorgenommen. Sie werden dabei wie folgt angegeben:

Wort 6 Erstes Maskenwort
Wort 7 Erstes Formwort
Wort 8 Zweites Maskenwort
Wort 9 Zweites Formwort
...
Wort 36 Sechzehntes Maskenwort
Wort 37 Sechzehntes Formwort

Um uns die Anwendung und die Programmierung von Sprites mit diesem Befehl in GFA-BASIC zu vergegenwärtigen, schauen wir uns einmal das folgende Beispiel-Programm an:

```

' *****
' **                **
' ** Atari-Sprites **
' **                **
' *****
'
' Bildschirmauflösung:
xauf1%=DPEEK(&H2686)
yauf1%=DPEEK(&H2688)
'
DIM spri$(6),xs$(6),ys$(6)
'
PRINT AT(15,1);"Die ST-Sprite-Show"
'
' Rechtecke zeichnen:
FOR x%=xauf1%/8 TO xauf1%/4 STEP 4
    RBOX x%,x%,xauf1%-x%,yauf1%-x%
NEXT x%
FOR x%=xauf1%/4 TO xauf1%/2
    RBOX x%,x%,xauf1%-x%,yauf1%-x%
NEXT x%
'
' Spriteattribute setzen:
schmet$=CHR$(0)+CHR$(0)      !x-Koord. Aktionspunkt = 0
schmet$=schmet$+CHR$(0)+CHR$(0) !y-Koord. Aktionspunkt = 0
schmet$=schmet$+CHR$(0)+CHR$(1) !XOR-Modus
schmet$=schmet$+CHR$(0)+CHR$(0) !Maskenfarbe = Hintergrund
schmet$=schmet$+CHR$(0)+CHR$(1) !Formfarbe = Zeichenfarbe
'
' Sprite-Definition aus DATAs lesen:
FOR ii%=0 TO 15
    READ m1%,m2%,f1%,f2%
    schmet$=schmet$+CHR$(m1%)+CHR$(m2%)+CHR$(f1%)+CHR$(f2%)
NEXT ii%
'
' Sprite-Definition an 7 Sprites verteilen
' und Startkoordinaten festlegen:
FOR num%=0 TO 6
    spri$(num%)=schmet$
    xs$(num%)=num%*xauf1%/10
    ys$(num%)=num%*yauf1%/10
NEXT num%
'
' Sprites zufällig bewegen:
REPEAT
    VSYNC                      !Bildsynchronisation
    FOR num%=0 TO 6
        xs$(num%)=ABS(xs$(num%)+RND(0)*5-2)
        ys$(num%)=ABS(ys$(num%)+RND(0)*5-2)
        IF xs$(num%)>xauf1%
            xs$(num%)=xauf1%
        ENDIF
    
```

```

IF ys%(num%)>yauf1%
  ys%(num%)=yauf1%
ENDIF
'
'VSYNC
SPRITE spri$(num%),xs%(num%),ys%(num%)
NEXT num%
UNTIL INKEY$<>"
'
EDIT
'
'
' Sprite-Definition mit Maske und Form:
DATA &X00111000,&X00011100, &X00010000,&X00001000
DATA &X00011100,&X00111000, &X00001000,&X00010000
DATA &X00001110,&X01110000, &X00000100,&X00100000
DATA &X00110111,&X11101100, &X00000010,&X01000000
DATA &X01111111,&X11111110, &X00111001,&X10011100
DATA &X11101111,&X11110111, &X01000101,&X10100010
DATA &X11010111,&X11101011, &X10000011,&X11000001
DATA &X11011111,&X11111011, &X10001001,&X10010001
DATA &X11111111,&X11111111, &X10011101,&X10111001
DATA &X11011111,&X11111011, &X10001001,&X10010001
DATA &X11010011,&X11010011, &X10000001,&X10000001
DATA &X11000011,&X11000011, &X10000001,&X10000001
DATA &X11100111,&X11100111, &X01000011,&X11000010
DATA &X01110111,&X11101110, &X00100011,&X11000100
DATA &X00111110,&X01111100, &X00011100,&X00111000
DATA &X00000000,&X00000000, &X00000000,&X00000000

```

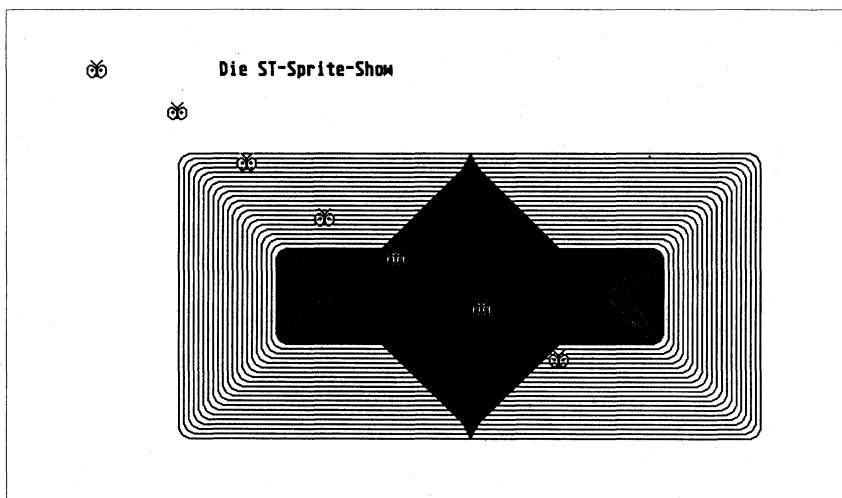


Abb. 15.43 Atari-Sprites

Dieses kleine GFA-BASIC-Programm bringt insgesamt sieben Sprites gleichen Aussehens auf den Bildschirm und bewegt sie zufallsgesteuert auf ihm hin und her. Um die Wechselwirkungen der Sprites mit dem Hintergrund zu demonstrieren, wurde er vorher mit einigen Rechtecken bemalt. Eine beliebige Taste beendet das Spektakel.

Der String schmet\$ enthält dabei die vollständige Sprite-Definition mit den Attributdaten und den aus den DATA-Zeilen eingelesenen Masken- und Form-Bytes. Interessant ist hierbei für Sie sicherlich die Handhabung und "Bestückung" dieses Strings mit den verschiedenen Werten.

In der großen REPEAT...UNTIL-Schleife werden dann die sieben Sprites nacheinander bewegt. Die Schrittweite und -richtung dieser Bewegung hängt dabei von dem Ergebnis der RND()-Funktion ab, die einen Zufallswert zwischen 0 und 1 liefert.

Lassen Sie das Programm doch einmal eine Zeitlang laufen. Sobald sich zwei Sprites treffen, werden Sie sehen, was es mit den Ausführungen vom Anfang auf sich hat.

Wenn Sie sich das obige Programm und sein Ergebnis anschauen, werden Sie sich sicher über die geringe Größe und Flexibilität der Sprites ärgern. Zudem passieren solche unschönen Malheurs beim Zusammentreffen zweier Sprites.

Wie Sie beliebig große und beliebig viele Sprites programmieren, die sich ruhig auch einmal überlappen können, zeigt Ihnen das folgende C-Programm. In ihm wird ein 32x38 Punkte großes Sprite definiert und - ähnlich wie beim obigen BASIC-Programm - diesmal in 8-facher Ausfertigung auf den Bildschirm gezaubert. Das Programm verwendet hierzu die Rasterbefehle im XOR-Modus. Daß die x-Größe des Sprites durch 16 teilbar ist, ist in diesem Programm nicht unbedingt notwendig. Zwar muß das Hauptraster (der "virtuelle Bildschirm") in der x-Auflösung durch 16 teilbar sein, das eigentliche Sprite aber kann ja auch nur einen Ausschnitt dieses Hauptrasters darstellen. Es sind aber auch Alternativen hierzu unter Verwendung von Bit-Feldern möglich. :

```
/*  
**  
** Sprites - beliebig groß **  
**  
*/
```

```
int contrl[12],  
    intin[128],  
    ptsin[128],  
    intout[128],  
    ptsout[128];
```



```

int gl_apid;
int gem_handle;
int handle;
int count;
int gl_wchar, gl_hchar, gl_wbox, gl_hbox;
int work_in[12],
    work_out[57],
    pxyarray[10];

/*****
/*      Initialisiere Applikation      */
*****/
aes_init()
{
    int i;
    gl_apid = appl_init();
    gem_handle = graf_handle(&gl_wchar, &gl_hchar,
                             &gl_wbox, &gl_hbox);
}

/*****
/*      Terminiere Applikation      */
*****/
aes_exit()
{
    gemdos(0x1);          /* Auf Taste warten */
    appl_exit();
}

/*****
/*      Öffne virtuelle Workstation    */
*****/
open_vwork()
{
    int i;

    for (i=0; i<10; work_in[i++]=1);
    work_in[10] = 2;
    handle = gem_handle;
    v_opnvwk(work_in, &handle, work_out);
}

/*****
/*      Schließe virtuelle Workstation */
*****/

#define close_vwork() v_clswwk(handle);

/*****
/*      Hauptprogramm      */
*****/
/* Typ- und Strukturdefinition für MFDB: */
typedef struct fdbstr {

```

```

        long fd_addr;    /* Rasteradresse */
        int fd_w;        /* Rasterbreite in Punkten */
        int fd_h;        /* Rasterhöhe in Punkten */
        int fd_wdwidth;  /* Rasterbreite in Wort */
        int fd_stand;    /* Standard-Flag */
        int fd_nplanes;  /* Anzahl der Farbebenen */
        int fd_r1;       /* zukünftige Erweiterungen */
        int fd_r2;
        int fd_r3;
    } FDB;                /* Neuer Typ: FDB */

int xaflu,                /* x-Bildschirmauflösung */
yaflu,                   /* y-Bildschirmauflösung */
i,                       /* Sprite-Nummer */
quell_breite = 2*16,     /* Quellrasterbreite */
quell_hoehe = 36;        /* Quellrasterhöhe */
/* Quellmusterdefinition für COPY RASTER, TRANSPARENT: */
int hase[38][2] = {
    {0x0004, 0x4000}, {0x000a, 0xa000},
    {0x000a, 0xa000}, {0x0012, 0x9000},
    {0x0012, 0xa000}, {0x000a, 0xa000},
    {0x000b, 0xe000}, {0x0004, 0x2000},
    {0x0008, 0x1000}, {0x000a, 0x5000},
    {0x0009, 0x9000}, {0x000a, 0x6000},
    {0x0009, 0xa000}, {0x0007, 0xc000},
    {0x0018, 0x3000}, {0x0020, 0x0800},
    {0x0040, 0x0400}, {0x01c0, 0x0700},
    {0x02a0, 0x0a80}, {0x0290, 0x1280},
    {0x0308, 0x2300}, {0x0184, 0x4d00},
    {0x01fc, 0x5100}, {0x0119, 0xc900},
    {0x0116, 0x3900}, {0x0110, 0x09f0},
    {0x0110, 0x0910}, {0x0090, 0x0910},
    {0x0088, 0x1210}, {0x0087, 0xe220},
    {0x00c0, 0x07c0}, {0x0130, 0x0e00},
    {0x021f, 0xf100}, {0x0410, 0x1100},
    {0x0410, 0x1100}, {0x0220, 0x1100},
    {0x01c0, 0x0e00}, {0x0000, 0x0000},
};

int xkoord[8], ykoord[8];    /* Sprite-Koordinaten */
FDB psrMFDB;                 /* Quellrasterstruktur */
FDB pdesMFDB = {0,0,0,0,0,0,0,0}; /* Zielrasterstruktur: */
/* Adresse=0 für Bildschirm */
int color_index[2] = {0,1}; /* Farbindizes für 0 und 1 */

main()
{
    int x,y;
    /* Quellrasterstruktur Wertebelegung: */
    psrMFDB.fd_addr = (long)&hase[0][0];
    psrMFDB.fd_w = quell_breite;
    psrMFDB.fd_h = quell_hoehe;
    psrMFDB.fd_wdwidth = quell_breite/16;
    psrMFDB.fd_stand = 0;
    psrMFDB.fd_nplanes = 1;

```

```

psrcMFDB.fd_r1      = 1;
psrcMFDB.fd_r2      = 1;
psrcMFDB.fd_r3      = 1;
aes_init();
open_vwork();        /* Initialisierung */
xaupl = work_out[0];  /* Auflösung feststellen */
yaupl = work_out[1];
v.clrwk(handle);      /* Bildschirm löschen */
vsf_color(handle, 3); /* Füllmusterfarbe für BAR */
vsf_interior(handle, 2); /* Füllmuster für BAR */
/* Zeichenschleife für 4 Rechtecke: */
for (y=0; y<2; y++)
  for (x=0; x<2; x++)
  {
    vsf_style(handle, x+y*8);
    /* Gefülltes Rechteck auf den Bildschirm: */
    pxyarray[0] = x*xaupl/3 + 20;
    pxyarray[1] = y*yaupl/4 + 20;
    pxyarray[2] = pxyarray[0] + xaupl/3-5;
    pxyarray[3] = pxyarray[1] + yaupl/4-5;
    v_bar(handle, pxyarray); /* Rechteck zeichnen */
  }
/* Initialisierung der Sprite-Koordinaten */
/* und erstes Zeichnen der Sprites */
for (i=0; i<8; i++)
{
  xkoord[i] = xaupl/8 * i; /* Startkoordinaten */
  ykoord[i] = yaupl/8 * i; /* berechnen */
  setsprite(xkoord[i], ykoord[i]); /* Sprite setzen */
}
while (gemdos(0x0b)==0) /* CONSTAT, Tastenpuffer testen */
{
  xbios(37); /* Vsync, auf Strahlrücklauf warten */
  for (i=0; i<8; i++)
  {
    /* Altes Sprite löschen: */
    setsprite(xkoord[i], ykoord[i]);
    /* Sprite-Koordinaten zufällig ändern: */
    /* xbios(17) ist die XBIOS-Funktion 'Random' */
    xkoord[i] = xkoord[i] + (xbios(17) & 3) - 1;
    ykoord[i] = ykoord[i] + (xbios(17) & 3) - 1;
    if (xkoord[i]<0 || xkoord[i]>xaupl)
      xkoord[i] = 0;
    if (ykoord[i]<0 || ykoord[i]>yaupl)
      ykoord[i] = 0;
    /* Neues Sprite setzen: */
    setsprite(xkoord[i], ykoord[i]);
  }
}
close_vwork(); /* Programmende */
aes_exit();
}

```

```

/*****
/**  Sprite in XOR-Modus nach x,y setzen (löschen)  **/
*****/
setsprite(x,y)
int x,y;
{
    int breite, hoehe;
    /* Rasterrechteck mit Bildschirm clippen: */
    breite = quell_breite;
    hoehe = quell_hoehe;
    if ( clipscreen(&x, &breite, &y, &hoehe) != 0);
    {
        /* Quellraster (Sprite) in Bildschirm kopieren: */
        /* Zielkoordinaten festlegen: */
        pxyarray[4] = x;
        pxyarray[5] = y;
        pxyarray[6] = x + breite;
        pxyarray[7] = y + hoehe;
        /* Quellkoordinaten festlegen: */
        pxyarray[0] = 0;
        pxyarray[1] = 0;
        pxyarray[2] = breite;
        pxyarray[3] = hoehe;

        /* Raster in Bildschirm kopieren (XOR-Modus): */
        vrt_cpyfm (handle, 3, pxyarray,
                  &psrcMFDB, &pdesMFDB, color_index);
    }
}

/*****
/**  Rechteck mit Bildschirm clippen  **/
*****/
/* Funktion = 0: Rechteck außerhalb Bildschirm! */
clipscreen(p_x, p_breite, p_y, p_hoehe)
int  *p_x,
     *p_breite,
     *p_y,
     *p_hoehe;
{
    if (*p_x > xaufl || *p_y > yauf1) /* Sprite vollständig außerhalb? */
        return(0); /* ja -> Rückgabewert: 0 */
    if (*p_x + *p_breite > xaufl) /* Sprite rechts halb außerhalb? */
        *p_breite = xaufl - *p_x; /* Clippen */
    if (*p_y + *p_hoehe > yauf1) /* Sprite links halb außerhalb? */
        *p_hoehe = yauf1 - *p_y; /* Clippen */
    return(1); /* Rückgabewert: 1 */
}

```

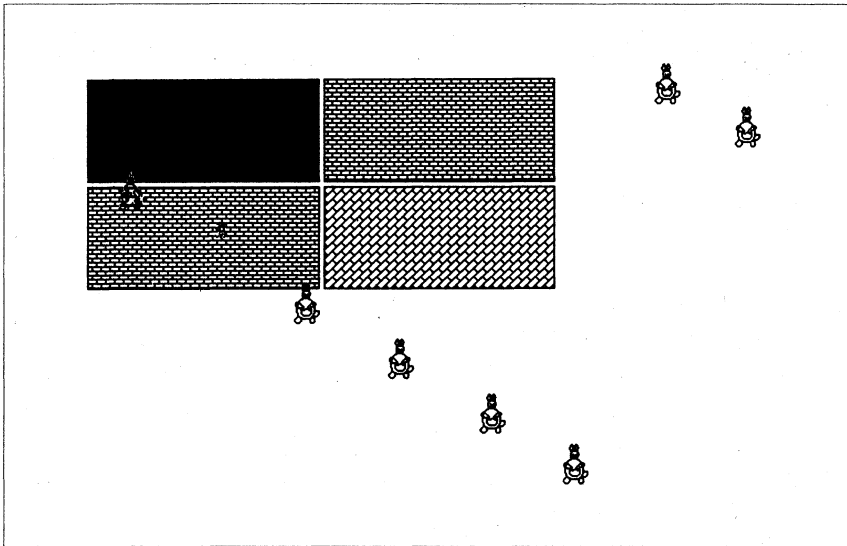


Abb. 15.44: Sprites - beliebig groß

Aus dem Kapitel über die Raster-Kommandos sollten Ihnen einige Passagen des Programms bereits bekannt vorkommen (MFDB-Definition, Musterdefinition etc.). Auch das Zeichnen der vier Rechtecke als Hintergrundgrafik wurde aus dem dort vorgeführten Beispiel übernommen.

Wie gesagt werden die Sprites mittels COPY RASTER im XOR-Modus an die jeweiligen Bildschirmstellen übertragen. Hierzu werden die Koordinaten der Sprites in einer FOR-Schleife initialisiert und die Sprites selbst zum ersten Mal auf den Bildschirm gebracht. Das Zeichnen der Sprites übernimmt dabei die Funktion "setsprite()", der die Position übergeben wird. Hier passiert nun folgendes:

Bevor die Raster-Funktion zur Übertragung der Sprite-Definition aufgerufen wird, muß zunächst geprüft werden, ob sich das Sprite noch vollständig in dem Bildschirmfenster befindet. Sollte das nicht der Fall sein, so muß das Sprite entweder so abgeschnitten werden, daß es noch auf die Mattscheibe paßt, oder es darf - sofern vollständig außerhalb - gar nicht gezeichnet werden (Clipping).

Dies übernimmt die Clip-Routine "clipscreen()", die exakt die erforderlichen Sprite-Umformungen vornimmt, die dazu nötig sind, das Sprite "passend" zu

machen. Falls kein Punkt des Sprites mehr auf dem Bildschirm liegt, liefert die Funktion den Rückgabewert 0, ansonsten 1.

Mit den so eventuell angepaßten Sprite-Breiten und -höhen wird nun in der Funktion "setsprite()" das "pxyarray[]" bestückt und die Funktion COPY RASTER, TRANSPARENT aufgerufen.

Beachten Sie bitte, daß das nochmalige Aufrufen dieser Funktion mit den gleichen Spritedaten (Form, Koordinaten) das an dieser Stelle befindliche Sprite wieder löscht.

Kehren wir zum Hauptprogramm zurück. Hier wird so lange gezeichnet, bis Sie eine Taste betätigen ("while-Schleife"). Der Zeichenvorgang an sich geht wie gewohnt vor sich: Sprite löschen, Koordinaten ändern (hier mit der XBIOS-Random-Funktion), und Sprite wieder setzen.

Mit diesem Programm ist es kein Problem, beispielsweise Sprites von den Ausmaßen eines ganzen Bildschirms zu zeichnen. Der Weg zur problemlosen Animation mit dem Atari ST ist geebnet.

16. Mehr zum Bildschirm

Grafik ist zwar ein interessantes Gebiet - doch es gibt noch ein paar andere Aspekte, die zum Bildschirm des Atari erwähnt werden sollten. Wie zum Beispiel die folgenden.

16.1 Das Video-RAM des Atari ST

Grafik mit den vielfältigen Funktionen des GEM ist recht komfortabel, aber manchmal kommt man einfach nicht drumherum, den Bildschirmspeicher direkt zu beackern - und sei es nur zum Abspeichern in eine Datei. Betrachten wir deshalb den Aufbau des Bildschirmspeichers in den verschiedenen Grafik-Modi des ST.

Unmittelbar nach dem Einschalten des ST wird ein Bereich von 32 KByte von der oberen Speichergrenze als Video-RAM initialisiert. Damit ergibt sich je nach Speicherausbau eine Adreßlage von \$78000 bis \$7FFFF oder \$F8000 bis \$FFFFF als Bildschirm-RAM. Dieses Video-RAM kann wie ein Fenster in den ST betrachtet werden. In der folgenden Beschreibung gehen wir zur Vereinfachung auf die Gegebenheiten im 260 ST mit "nur" 512 KByte ein. Bei den anderen Modellen verschiebt sich die Adresse - aber die kann ja mit der entsprechenden XBIOS-Funktion leicht ermittelt werden.

Beginnen wir mit dem einfachsten Modus, dem 640x400-Modus. In diesem Fall bilden je 80 Bytes oder besser gesagt 40 Worte (Doppelbytes) eine Bildschirmzeile. Das Wort mit der niedrigsten Adresse wird am linken Bildschirm-Rand dargestellt, die weiteren Worte werden kontinuierlich von links nach rechts dargestellt. Innerhalb eines Words liegt das oberste, höchstwertige Bit links, das niederwertigste Bit ganz rechts.

Mit diesen Daten kann nun jeder Punkt des Bildschirms recht einfach angesteuert oder ausgelesen werden. Um zum Beispiel den ersten Bildschirm-Punkt zu setzen, muß in die Speicherstelle \$78000 der Wert \$8000 geschrieben werden. Allerdings muß an dieser Stelle eine kleine Einschränkung gemacht werden. Die Lage des Bildschirm-RAMs läßt sich im ST recht einfach verschieben. Dann stimmt die absolute Adresse \$78000 natürlich nicht mehr. Aus diesem Grund ist es in den meisten Fällen günstiger, den Punkt über die Line-"A"-Funktion \$A001 zu setzen. Diese Routine geht von einem X-Y-Koordinatensystem mit Ursprung in der linken oberen Ecke aus und bestimmt selber die Lage des Video-RAMs, um den Punkt an der richtigen Stelle zu setzen. In diesem Auslösungsmodus wird also jeder Bildpunkt durch ein Bit repräsen-

tiert. Ist das Bit gesetzt, so erscheint der Punkt dunkel oder auch hell, wenn im Color-Palette-Register 0 der inverse Darstellungsmodus gewählt ist. Der Bildschirm besteht aus nur einer Bit-Ebene, Plane genannt. Unterschiedliche Farben jedoch lassen sich mit nur einer Bit-Ebene (einer Plane) nicht darstellen. Darum wird die Auflösung in den Farbmodi mit steigender Anzahl gleichzeitig darstellbarer Farben geringer.

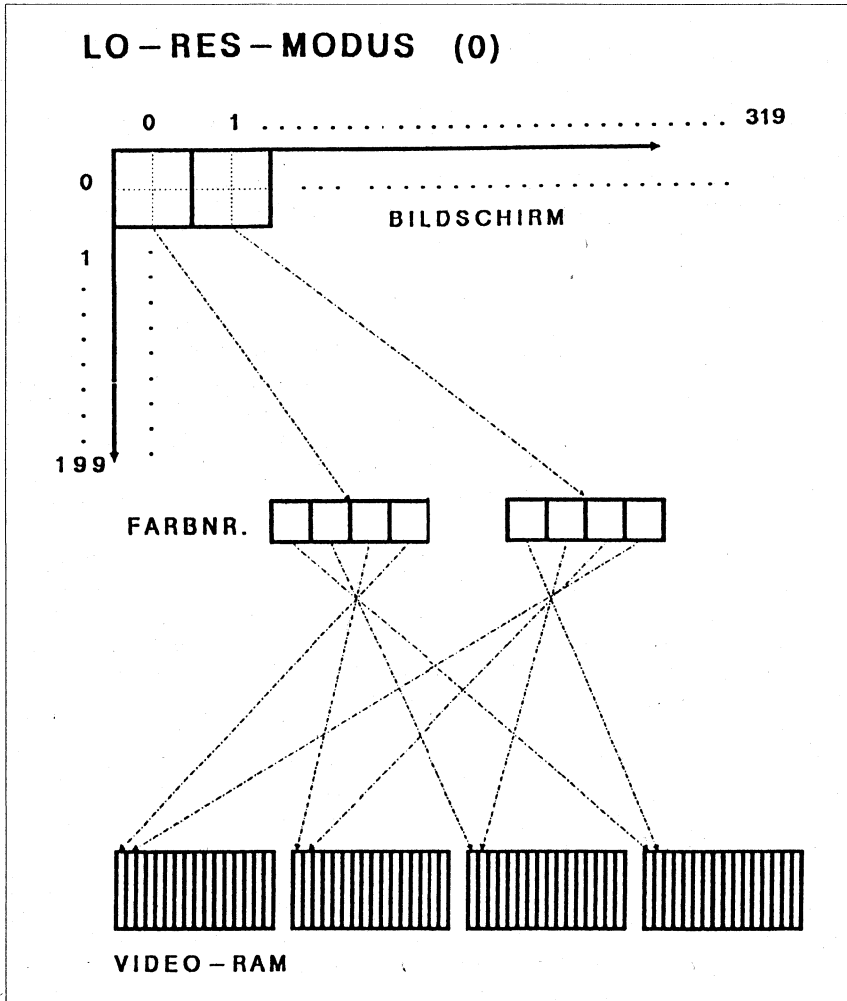


Abb. 16.1: Grafik im Low-Res-Modus

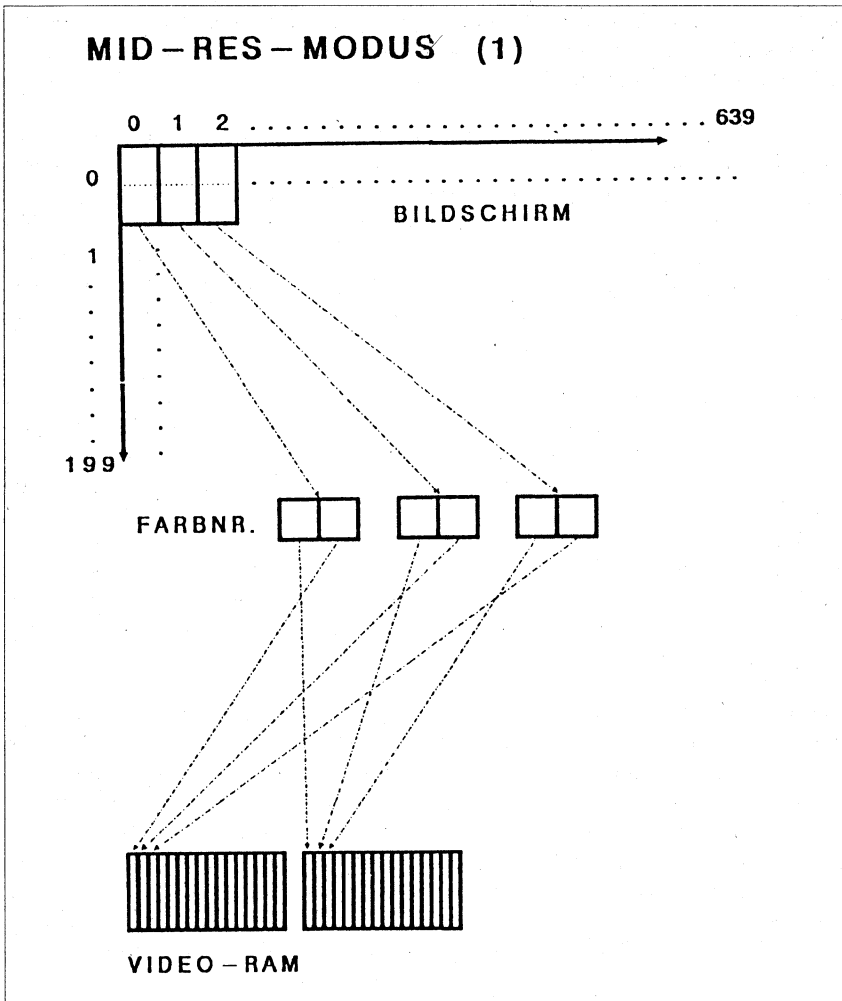


Abb. 16.2: Grafik im Med-Res-Modus

In der Auflösungsstufe 640x200 Punkte sind immerhin schon vier Farben möglich. In dieser Darstellungsart bilden immer zwei aufeinander folgende Speicherworte eine zusammengehörende Einheit. Die Farbe eines Bildpunkts wird bestimmt durch den Wert der beiden gleichwertigen Bits in den beiden Worten. Sind beide Bits Null, so ergibt sich die Hintergrundfarbe. Für die Pixel-Darstellung gehören also zwei aufeinander folgende Worte zusammen. Für

die Farben jedoch gehören alle ungeraden Worte zu einer Plane. Die zweite Plane wird aus den geraden Worten gebildet. In diesem Modus verfügt man also über zwei Planes (Farb-Bit-Ebenen).

Richtig bunt wird die Sache dann aber im Modus mit "nur" 320x200 Punkten. In dieser Betriebsart bilden dann sogar vier aufeinander folgende Speicher-Worte eine Einheit, die über die Farbe der 16 zugehörigen Pixel bestimmen. Um bei dem zuvor bereits einmal benutzten Beispiel zu bleiben: Um den Punkt in der linken oberen Ecke des Bildschirms zu setzen, müssen die vier Worte \$78000, \$78002, \$78004 und \$78006 in ihren obersten Bits manipuliert werden. Je nach Bit-Muster in den Worten ergibt sich dann die gewünschte Farbe.

Das erfordert natürlich schon einiges an Rechenaufwand, um einen Punkt Modus-unabhängig in der jeweils gewünschten Farbe zu setzen. All diese Arbeit wird einem jedoch von der \$A001-Routine abgenommen. Diese Routine setzt alle zusammengehörenden Bits der jeweils eingestellten Auflösung für die gewünschte Farbe.

In diesem Modus sind natürlich auch vier Planes vorhanden. Die erste Plane wird, um bei unserem Beispiel zu bleiben, von den Worten an den Adressen \$7F000, \$7F008, \$7F010 ..., die weiteren Planes von den entsprechend anderen Adressen gebildet.

Die drei eingebauten Fonts sind fixe Schriften, haben also eine fest definierte Größe in Pixeln und einen definierten Abstand. Die kleinste Schrift hat eine Matrix von 6x6 Bildpunkten. Bei einer Auflösung von 640x400 Punkten könnten so 106 Zeichen in 66 Zeilen untergebracht werden. Diese Schrift ist im ST nur bei der Ausgabe unter GEM, nicht jedoch zur Ausgabe unter TOS einsetzbar und wird z.B. bei der Ausgabe des Directory in Icon-Form eingesetzt. Die nächstgrößere Schrift setzt sich aus 8x8 Punkten zusammen. Diese Schrift wird verwendet, wenn ein Farbmonitor am ST angeschlossen ist, während die normale Schwarzweiß-Darstellung mit dem dritten, größten Font vorgenommen wird.

Dieser Font benutzt eine Matrix von 8x16 Punkten. Den genauen Aufbau des Font-Headers finden Sie beim Befehl \$A008, der eine recht komplexe, aber sehr vielseitige Textausgabe darstellt, die weit über das hinausgeht, was mit den Routinen des BIOS und des GEMDOS möglich ist.

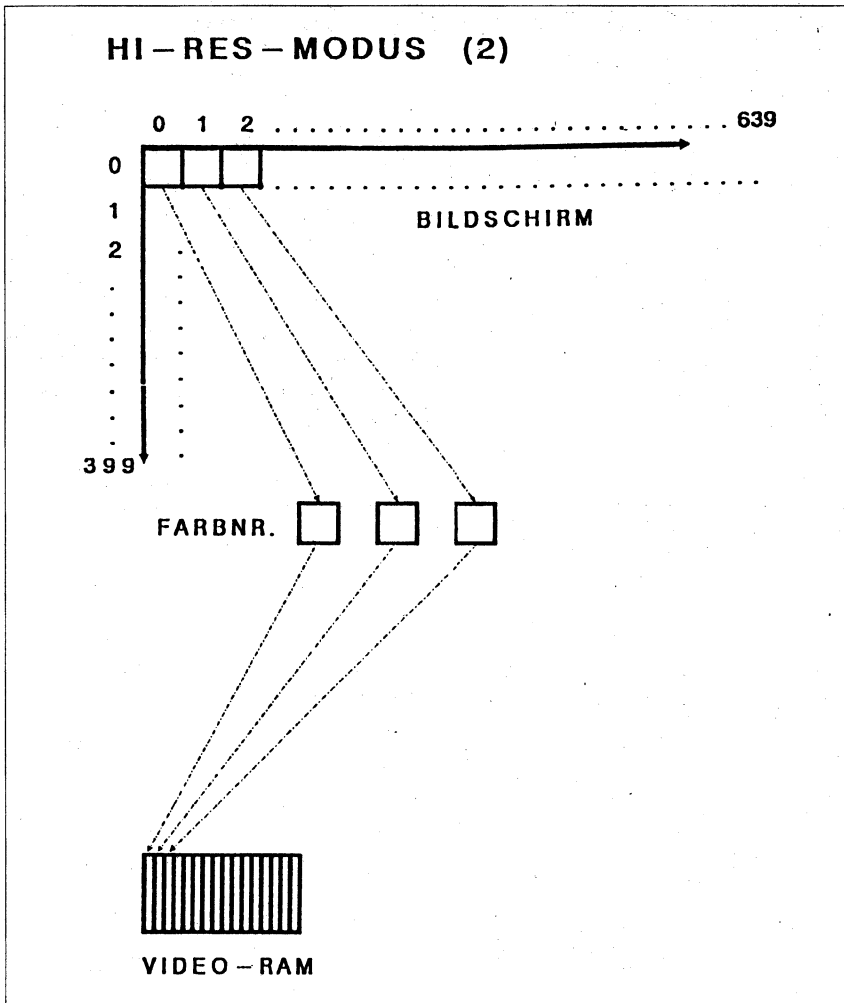


Abb. 16.3: Grafik im Hi-Res-Modus

16.2 Umschaltung von 50 auf 60 Hz für Farbmonitore

Sicherlich ist Ihnen das Flimmern Ihres Farbmonitors oder Fernsehers gegenüber dem Monochrommonitor aufgefallen. Dies liegt an der niedrigeren Bildwiederholfrequenz von nur 50 Hz gegenüber 71 Hz, d.h. beim Farbmonitor wird 50mal pro Sekunde ein Bild gezeigt, während es beim Monochrommonitor 71 Bilder sind. In Amerika aber ist es üblich, daß der Wechselstrom mit 60 Hz aus der Steckdose kommt, und so hat auch der Atari die Möglichkeit, für amerikanische Fernseher mit 60 Hz die Bilder zum Monitor zu schicken. Diese Frequenz wird von den meisten uns bekannten Monitoren und Fernsehern noch unterstützt. Sollte Ihr Monitor es nicht können, so erkennen Sie dies nach Ausführen des folgenden Programmes an einem Durchlaufen des Bildes. Er kann dann das Bild nicht synchronisieren.

Für die Bilddarstellung ist der "Shifter" im ST zuständig. Er sorgt sowohl für die Farb- als auch die Schwarzweiß-Darstellung. Und in einem seiner Register ist ein Bit verantwortlich für die Umschaltung von 50 auf 60 Hz:

Adresse: \$FF820A (16744970) (Byte)

An dieser Adresse stehen zwei relevante Bits, das eine, Bit 0, wird beim Bildschirm-Saver benutzt und schaltet wahlweise externe und interne Synchronisation um, das zweite, Bit 1, ist für uns hier interessant:

Bit 0	0 = interne Synchronisation
	1 = externe Synchronisation
Bit 1	0 = 60 Hz
	1 = 50 Hz

Wir wollen die Programmzeile so auslegen, daß bei jedem Starten die jeweils andere Frequenz eingestellt wird. Es könnte dann etwa so aussehen:

```
GFA: Spoke &HFF820A, (Peek(&HFF820A) Xor 2)
C:   char *zeiger;
     long save_sp;
     save_sp=gemdos(0x20,0L);
     zeiger = 0xFF820A;
     *zeiger = *zeiger ^ 2;
     gemdos(0x20,save_sp);
ASS: clr.l    -(sp)
     move.w  #$20,-(sp)
     trap   #1
     addq.l  #6,sp
```

```

bchg    #1,$FF820A
move.l   D0,-(sp)
move.w   #$20,-(sp)
trap     #1
addq.l   #6,sp

```

Wie Sie sehen, muß der Prozessor im Supervisor-Modus arbeiten, um auf Register des Shifters zugreifen zu können. Das ^-Zeichen im C-Programm steht nicht für die Potenzierung (die gibt es in C nicht), sondern für das Exklusiv-Oder.

```

*****
*      50-60Hz-Umschaltung      *
*****
clr.l    -(sp)      * Supervisormodus
move.w   #$20,-(sp)
trap     #1
addq.l   #6,sp
bchg     #1,$FF820A * Bit für 50/60Hz umdrehen
move.l   d0,-(sp)   * User-Modus
move.w   #$20,-(sp)
trap     #1
addq.l   #6,sp
clr.w    -(sp)      * Programm beenden
trap     #1

```

16.3 Bildschirmschoner

Wenn Sie viel programmieren oder oft lange Texte mit einer Textverarbeitung bearbeiten, dann haben Sie sicherlich in einer kleinen Pause, z.B. für das Mittagessen, den Computer eingeschaltet gelassen. Dies ist für den Rechner auch nicht weiter schädlich.

Doch für den Bildschirm sind sich nicht ändernde Standbilder eine Gefahr. Die Elektronen, die das Bild auf den Schirm bringen, prallen immer wieder an der gleichen Stelle auf die beschichtete Innenseite der Bildröhre auf. Diese Beschichtung wird aber durch fortlaufenden Beschuß beschädigt, das Bild brennt sich auf der Beschichtung ein. Auch nach dem Ausschalten des Monitors bleibt ein einmal eingebranntes Bild gut sichtbar erhalten. Man kann nun entweder den Monitor während der Pause am Helligkeitsregler dunkel stellen oder aber folgendes Programm benutzen.

```

*****
*      Bildschirmschoner      *
*      SAVER.PRG              *
*****
        move.l   4(sp),a0      * Speicherbedarf ausrechnen
        move.l   #$100,d6      *
        add.l    12(a0),d6      *
        add.l    20(a0),d6      *
        add.l    28(a0),d6      *
        pea      init          * Initialisierung im Supervisor
        move.w   #38,-(sp)      *
        trap     #14           *
        addq.l   #6,sp         *
        clr.w    -(sp)         * Programm resident halten
        move.l   d6,-(sp)      *
        move.w   #$31,-(sp)    *
        trap     #1           *
*                               *
init:                               *
        move.w   $454,d0        * Anzahl VBL-Routinen
        lsl.w    #2,d0         * mal 4
        move.l   $456,a0        * Liste der Startadressen
        moveq    #4,d1         * Suche zweiten Eintrag
*                               *
beginnensearch:                    *
        tst.l    (a0,d1)        * Wenn frei, dann steht hier eine 0
        beq.s    found          *
        addq.w   #4,d1          * sonst nächsten Eintrag untersuchen
        cmp.w    d0,d1          * schon alle Einträge durchsucht?
        bne.s    search        *
        illegal  * Ja, dann Bombendrohung
*                               *
found:                               *
        add      d1,a0          *
        move.l   a0,vektor      * merken für später
        move.l   #vbl1,(a0)     * Vektor auf VBL-Routine biegen
        move.l   $118,spr adr   * alten Tastatur-Interrupt merken
        move.l   #new_ir,$118   * und neuen eintragen
        rts                    *
*****
*      neue VBL-Routine wenn hell      *
*****
vbl1:                               *
        sub.w    #1,zaehler     * Zähler abziehen
        bne.s    vbl1_end       * schon Ende
        xor.b    #$FF,$FF8240   * Farbpalettenregister 0 XOR 255
        move.l   vektor,a0      * Vektor für VBL-Queue
        move.l   #vbl2,(a0)     * neue Routine eintragen
        move.w   #710,zaehler   * Sekunden * 71, Zähler hochsetzen
*                               *
vbl1_end:                          *
        rts                    *
*****
*      neue VBL-Routine wenn dunkel    *
*****

```

```

vbl2:
    sub.w    #1,zaehler          * Zähler abziehen
    bne.s    vbl2_end            * schon Ende
    xor.b     #$FF,$FF8240        * Farbpalettenregister 0 XOR 255
    move.l    vektor,a0          * Vektor für VBL-Queue
    move.l     #vbl1,(a0)         * alte Routine eintragen
    move.l     #710,zaehler       * Sekunden * 71, Zähler hochsetzen
vbl2_end:
    rts
*****
*   neue Tastaturinterrupt-Routine
*****
new_ir:
    move.w    #8520,zaehler       * zwei Minuten Verzögerung
    .dc.w     $4EF9              * Opcode für JMP
spr_adr:
    .dc.l     0                  * Platz für Sprungadresse in
                                * Originalroutine
*****
.data
zaehler:     .dc.w 8520          * Zeitzähler zwei Minuten
vektor:      .dc.l 0
*****

```

Das Programm wurde mit dem Assembler aus dem Atari-Entwicklungspaket entwickelt. Zunächst berechnet es den benötigten Speicherplatz und installiert im Supervisor-Modus zwei neue Interrupt-Routinen, einmal eine VBL-Routine im VBL-Queue, zum anderen einen neuen Tastatur- und Midi-Interrupt.

An der Adresse \$118 steht beim Atari der Interrupt-Vektor für die beiden ACIA-Bausteine, die für Tastatur und Midi zuständig sind. Diesen Vektor merken wir uns in der Variablen "spr_adr", die wir mit einem Long-Wert im Programm (Textsegment) freihalten. Vor dieser Variablen steht ein Pseudowert \$4EF9, der als Opcode für den direkten Jmp-Befehl steht. Zusammengesetzt ergibt sich daraus ein Jmp in die normale Interrupt-Routine. Durch diese Programmierung (Patches) konnte die Änderung ohne Benutzung eines Registers ablaufen, so daß diese nicht gerettet werden müssen.

Nachdem der Vektor gemerkt wurde, tragen wir unsere Routine ein. Bei jeder Mausbewegung, Maustaste, Tastaturbenutzung und jedem ankommenden Midi-Zeichen wird nun unsere Routine aufgerufen, die den Zähler auf den Wert für die Verzögerung setzt.

In der installierten VBL-Routine VBL1 wird nun je nach Bildfrequenz jeweils 71-, 60- oder 50mal pro Sekunde der Wert des Zählers herabgezählt. Wird der Wert 0 erreicht, so schaltet das Programm auf externe Synchronisation um, und da diese fehlt, wird der Bildschirm invertiert. Weiterhin wird eine neue VBL-Routine VBL2 installiert. Diese wartet nun so lange, bis wieder eine Taste

oder ähnliches gedrückt wurde und über den neuen Tastatur- und Midi-Interrupt der Zähler wieder gesetzt wird. Dann wird der Bildschirm wieder invertiert und er erscheint wieder wie gewohnt.

Hier noch die genaue Beschreibung der Speicherstelle, die für die Einstellung der Synchronisation verantwortlich ist. In dem Byte sind lediglich die unteren beiden Bits benutzt.

Während der Schonungsphase arbeitet der Rechner natürlich voll weiter, so daß Sie z.B. lange Programme compilieren können, ohne ständig die Maus zu bewegen.

17. Soundprogrammierung auf dem Atari ST

Oft muß der Atari-Fan hören, daß sein Rechner über völlig unzureichende Sound-Fähigkeiten verfügt. Mit diesem Kapitel wollen wir Ihnen das Gegenteil beweisen. Es ist nämlich oft so, daß der Soundchip gar nicht richtig ausgenutzt wird, was wahrscheinlich daran liegt, daß nur wenige Programmierer wissen, wie sie mit ihm umgehen müssen.

Wir wollen hier versuchen, Ihnen die Möglichkeiten und die Programmierung des YM-2149 von Yamaha (das ist der Soundchip) näherzubringen. Zu mehr ist ein Buch aber auch nicht in der Lage; die Resultate müssen Sie ganz einfach hören. Außerdem sollten Sie mit dem Chip spielen, also versuchen, immer neue und schönere Klänge und Geräusche zu erzeugen.

Wir werden auf zwei Ebenen mit dem Soundchip arbeiten: in GFA-BASIC und in C/Assembler. Im ersteren Fall ist die Programmierung einfach und schnell, was leider durch etwas eingeschränkte Möglichkeiten erkauft werden muß. Im zweiten Fall können Sie wirklich alle Register ziehen, doch ist die Programmierung alles andere als komfortabel.

17.1 Der Sound-Befehl in GFA-BASIC

Dies ist die einfachste Möglichkeit, dem Rechner einen Ton zu entlocken. Die Syntax des Befehls lautet:

Sound

Soundregisternummer, Lautstärke, Note, Oktave, Wartezeit

Der Soundchip verfügt über drei getrennte Sound-Register (1 bis 3), die einzeln angesprochen werden können. Damit ist auch klar, daß wir maximal dreistimmige Musik machen können. Die Lautstärke kann von 0 (nicht mehr zu hören) bis 15 (fortissimo) reichen. Der zu spielende Ton wird durch zwei Parameter gekennzeichnet: Note und Oktave. Note darf einen Wert zwischen 1 (C) und 12 (H) annehmen, womit eine ganze Oktave mitsamt Halbtönen abgedeckt wird:

generatoren (nicht der ganze Chip!) mit $1/8 \text{ MHz} = 125000 \text{ Hz}$ getaktet werden. Eine Tabelle mit den Frequenzen normaler Töne finden Sie etwas weiter unten. Erlaubt sind Periodendauern zwischen 0 (kein Ton) und 4095.

17.3 Neue Möglichkeiten mit dem Wave-Befehl

Die Töne, die wir mit dem "Sound"-Befehl erzeugen, können zwar ganz hübsche Melodien bilden. Sie klingen aber zu gleichmäßig, um den Zuhörer zu begeistern. Um das zu ändern, bietet der Soundchip die Möglichkeit, den Lautstärkeverlauf eines Tons durch eine sog. Hüllkurve festzulegen. Damit kann ein Ton z.B. hart angeschlagen werden und langsam verklingen, wie man es von einem Glockenspiel erwarten würde. Neun verschiedene Hüllkurven stehen zur Verfügung:

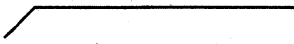
Hüllkurven		
Nummer	Grafik	Beschreibung
1-3/9		linear fallend
4-7		linear steigend
8		Sägezahn fallend
10		Dreieck
11		linear fallend, dann laut
12		Sägezahn steigend
13		linear steigend, dann laut
14		Dreieck
15		linear steigend

Abb. 17.2: Hüllkurven

Mit dem "Wave"-Befehl können Sie nun für einen bereits mit Sound aktivierten Ton eine Hüllkurve einstellen:

Wave Kontrollregister, Zuordnung, Hüllkurvenform, Dehnung, Wartezeit

Das Kontrollregister kann mit seinen sechs Bit die drei Tongeneratoren (Sound-Register) ein- und ausschalten und auf jeden Tonkanal den Rauschgenerator zuschalten. Die Bits 0 bis 2 des Registers kontrollieren die Tonkanäle 1 bis 3 (Bit gesetzt = Kanal an), und die Bits 3 bis 5 schalten das Rauschen auf die Kanäle 1 bis 3 (Bit gesetzt = Rauschen an). So bedeutet ein Wert von neun (binär 001001), daß der erste Tongenerator spielt und zusätzlich rauscht. Zuordnung bestimmt, auf welche Tongeneratoren die durch den Wave-Befehl zu aktivierende Hüllkurve angewendet werden soll. Die Zuordnung der Bits zu den Generatoren ist die gleiche wie im Kontrollregister.

Hüllkurvenform und Dehnung bestimmen schließlich die eigentliche Hüllkurve. Die möglichen Formen und deren Kennnummern entnehmen Sie bitte der obenstehenden Zeichnung. Dehnung bestimmt die zeitliche Länge einer Periode der Hüllkurve. Je größer der Wert für Dehnung ist, desto länger ist die Kurve. Erlaubt sind Werte von 0 bis 65535.

Das scheint alles etwas kompliziert und trocken zu sein, doch schauen (und hören!) Sie sich einmal die folgenden Beispiele an:

```

' Tonerzeugung in GFA-BASIC
' MP 28-06-88
'
' 1. Sound-Befehl über Note und Oktave
'
For O%=1 To 8
  For N%=1 To 12
    Sound 1,15,N%,O%,2
  Next N%
Next O%
'
'
' 2. Sound-Befehl mit Periodendauer
'
For P%=10 To 1000
  Sound 1,15,#P%,1
Next P%
'
'
' 3. Wave-Befehl
'
Sound 1,15,1,4 ! Ton einschalten
Wave 1,1,14,15,200 ! Hüllkurve: Dreieck Haltedauer: 15'
Wave 1,1,14,400,200 ! wie vorher, aber längere Haltedauer'Wave
1,1,14,3000,200 ! nochmals längere Haltedauer
'
Wave 1,1,8,3000,200 ! gleiche Haltedauer, aber Sägezahn fallend

```

```
,
Wave 1,1,1,40000,200    ! linear fallend, lange Haltedauer
,
,
4. Tonleiter mit linear fallender Hüllkurve, z.B. Glockenspiel
,
For N%=1 To 13           ! 13 Töne = chromatische Tonleiter
  Sound 1,15,N% Mod 12,5+N% Div 12
  Wave 1,1,1,8000,20     ! Hüllkurve, Haltedauer der Hüllkurve und
                        ! Wartezeit
Next N%
,
Sound 1,0,0,0,0          ! Sound-Generator abschalten
```

17.4 Sound in C und Assembler

Die direkte Programmierung des Soundchips in C und Assembler scheint zunächst einmal viel komplizierter zu sein als die Arbeit unter BASIC. Das liegt daran, daß nicht mehr mit einem oder zwei Befehlen ein beliebiger Ton erzeugt werden kann. Statt dessen gibt es im Soundchip 16 Register, in die verschiedene Daten eingetragen werden müssen. Bevor wir klären, wie man an diese Register herankommt, wollen wir Sie erst einmal erklären:

0,1 Periodendauer Kanal A. Register 0 enthält die unteren acht Bits der Periodendauer, Register 1 die oberen vier (!) Bits.

2,3 wie 0 und 1, aber für Kanal B

4,5 wie 0 und 1, aber für Kanal C

6 Die fünf unteren Bits steuern die Rauschfrequenz. Je kleiner der Wert, desto höher die Frequenz.

7 Multifunktionsregister:

- Bit 0 Kanal A ein- (gesetzt) oder ausschalten (gelöscht)
- Bit 1 wie 0, aber für Kanal B
- Bit 2 wie 0, aber für Kanal C
- Bit 3 Rauschen auf Kanal A zuschalten (wenn Bit gesetzt)
- Bit 4 wie 3, aber für Kanal B
- Bit 5 wie 3, aber für Kanal C
- Bit 6 für die Tonerzeugung nicht interessant, muß 1 sein
- Bit 7 wie 6

8 Lautstärke (Bits 0-3) Kanal A

Ist Bit 4 gesetzt, so wird die eingestellte Lautstärke ignoriert. Statt dessen wird die Lautstärke durch die aktuelle Hüllkurve bestimmt.

9 wie 8, aber für Kanal B

10 wie 8, aber für Kanal C

11,12 Bestimmen die Dehnung (Geschwindigkeit) der Hüllkurve. Je größer der Wert ist, desto langsamer verläuft die Kurve. Register 11 gibt das Low-Byte, Register 12 das High-Byte an.

13 Bestimmt die Hüllkurve. Die möglichen Werte wurden bereits im Zusammenhang mit dem Wave-Befehl in einer Grafik dargestellt.

14 Port A, für uns ohne Bedeutung

15 Port B, für uns ohne Bedeutung

So, diese Daten sollten Sie erst einmal in Ruhe verdauen. Wenn Sie damit fertig sind, können wir uns an das Beschreiben und Auslesen der Register machen. Der offizielle Weg führt über eine Routine des XBIOS. Sie heißt Giaccess und wird wie folgt aufgerufen:

```
return = Giaccess (daten, register);
In Assembler geht es so:
move.w #register, -(sp)
move.w #daten, -(sp)
move.w #28, -(sp)      ; Funktionsnummer
trap #14
addq.l #6, sp          ; Ergebnis steht dann in D0 bereit
```

Die Daten (acht Bit) werden in das Register register geschrieben. Register ist die Registernummer des Soundchips + 128, also mit gesetztem siebten Bit. Will man ein Register lesen, so entfällt das + 128; return bzw. D0 enthält dann den aktuellen Registerinhalt.

Der inoffizielle Weg ist nur in Assembler interessant. Zuerst wird die gewünschte Registernummer an die Adresse \$FFFF8800 (Byte!) geschrieben. Dann kann entweder aus der gleichen Adresse der aktuelle Registerinhalt gelesen oder in das Register über die Adresse \$FFFF8802 geschrieben werden. Sie sollten allerdings in der Zwischenzeit alle Interrupts abschalten, damit nicht zwischen der Auswahl des Registers und dem Zugriff auf selbiges z.B. ein Tastendruck einen Tastaturklick auslösen könnte.

Diese Art der Sound-Programmierung ist natürlich nicht sehr komfortabel; sie macht aber die Erstellung eigener Sound-Bibliotheken besonders interessant. Damit haben Sie wieder einmal die Möglichkeit, selber zu experimentieren - Übung macht den Meister!

17.5 Musik im Interrupt

Unser Betriebssystem TOS stellt einen ganz besonderen Leckerbissen zur Verfügung: Es ist in der Lage, ganze Musikstücke vollautomatisch in einem Interrupt abarbeiten zu lassen. Folgender Systemaufruf meldet die Sound-Verarbeitung an:

Dosound (Befehle)

Oder:

```
pea    Befehle
move.w #32,-(sp) ; Funktionsnummer
trap   #14
addq.l #6,sp
```

Befehle zeigt auf eine Liste mit Kommandos. Folgende Kommandos werden verstanden:

"\$00"-"\$0F"

Das Byte, das dem Kommando-Byte folgt, wird in das durch das Kommando-byte bestimmte Register geschrieben.

"\$80"

Das folgende Byte wird in das sog. temporäre Register geschrieben.

"\$81"

Dem Kommando folgen drei Parameter-Bytes. Das erste ist die Nummer eines Registers im Soundchip, in das der Inhalt des temporären Registers (siehe \$80) geschrieben wird. Das zweite Byte (-128 bis +127) wird dann zum temporären Register addiert. Nach einer 50stel Sekunde wiederholt sich der Vorgang. Er wird erst beendet, wenn das temporäre Register den Endwert erreicht hat, der im dritten Parameter-Byte steht. Damit kann man z.B. einen Ton automatisch weich erhöhen oder langsam lauter/leiser werden lassen.

"\$82" - "\$FF"

Ist das folgende Byte 0, so wird die Sound-Verarbeitung abgebrochen. Ansonsten stellt der Parameter eine Wartezeit dar, gemessen in 50stel Sekunden.

So praktisch diese Routine auf den ersten Blick auch scheint - es ist praktisch unmöglich, größere Stücke mit diesen Sound-Befehlen zu programmieren. Es gibt aber ein paar Musikprogramme, die eingegebene Noten nicht nur direkt spielen, sondern auch in Sound-Befehlen ausgeben können, mit der Sie die Interrupt-Routine bequem füttern können. Damit können Sie z.B. eine Titelmusik laufen lassen, während das Hauptprogramm noch von der Diskette geladen wird.

Zum Schluß geben wir Ihnen noch eine Tabelle, der Sie Frequenzen und Periodendauern von fünf Oktaven entnehmen können. Zu jeder Periodendauer werden auch gleich das Low- und das Highbyte mit angegeben:

Note	Frequenz	Perioden- dauer	High- Byte	Low- Byte
c	65.41	1911	7	119
cis=des	69.30	1804	7	12
d	73.42	1703	6	167
dis=es	77.78	1607	6	71
e	82.41	1517	5	237
f	87.31	1432	5	152
fis=ges	92.50	1351	5	71
g	98.00	1276	4	252
gis=as	103.83	1204	4	180
a	110.00	1136	4	112
ais=b	116.54	1073	4	49
h	123.47	1012	3	244
c	130.82	956	3	188
cis=des	138.59	902	3	134
d	146.84	851	3	83
dis=es	155.57	804	3	36
e	164.82	758	2	246
f	174.62	716	2	204
fis=ges	185.00	676	2	164
g	196.00	638	2	126
gis=as	207.66	602	2	90
a	220.00	568	2	56
ais=b	233.08	536	2	24
h	246.94	506	1	250
c	261.63	478	1	222
				...

cis=des	277.18	451	1	195
d	293.67	426	1	170
dis=es	311.13	402	1	146
e	329.63	379	1	123
f	349.23	358	1	102
fis=ges	369.99	338	1	82
g	392.00	319	1	63
gis=as	415.31	301	1	45
a	440.00	284	1	28
ais=b	466.16	268	1	12
h	493.88	253	0	253
c	523.26	239	0	239
cis=des	554.36	225	0	225
d	587.34	213	0	213
dis=es	622.26	201	0	201
e	659.26	190	0	190
f	698.46	179	0	179
fis=ges	739.98	169	0	169
g	784.00	159	0	159
gis=as	830.62	150	0	150
a	880.00	142	0	142
ais=b	932.32	134	0	134
h	987.76	127	0	127
c	1046.52	119	0	119
cis=des	1108.72	113	0	113
d	1174.68	106	0	106
dis=es	1244.52	100	0	100
e	1318.52	95	0	95
f	1396.92	89	0	89
fis=ges	1479.96	84	0	84
g	1568.00	80	0	80
gis=as	1661.24	75	0	75
a	1760.00	71	0	71
ais=b	1864.64	67	0	67
h	1975.52	63	0	63

Teil C

18. Die Komponenten des ST

In diesem Teil beschäftigen wir uns intensiv mit der Hardware. Von außen ist dem ST ja nicht anzusehen, was sich hinter dem Gehäuse verbirgt. Neben der elektrischen Beschreibung, die viele Neuigkeiten bringen dürfte, sind auch die vielen Register der ICs unser Thema. Ihre Kenntnis ist für den ausgefuchsten Programmierer mit hochgesteckten Ambitionen unerlässlich. Aber verschaffen wir uns doch erst einmal ein wenig Überblick.

Logisch - zunächst ist da ein Prozessor, meistens von 1 MByte RAM assistiert, der da herumwerkelt. In unserem Fall ist das der MC68000. Dazu kommt dann noch das ROM (192 KByte bei den älteren Modellen). Aber - und jetzt kommen die Hardwarefreaks erst richtig ins Schwärmen - da sind ja auch noch die vielen Schnittstellen, die ebenfalls durch entsprechende Chips versorgt sein wollen. Sehen Sie sich einmal Abbildung 18.1 an.

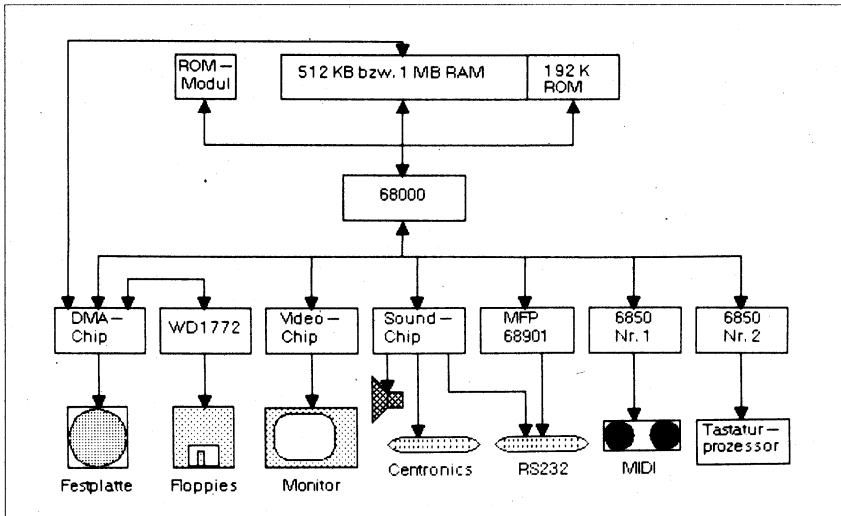


Abb. 18.1: Blockschaltbild des ST

Sie zeigt ein etwas vereinfachtes Blockschaltbild des ST. Wie Sie sehen, gibt es eine Reihe von Chips, die sich unter Umständen die Arbeit an den Schnittstellen teilen und quasi dem Prozessor unterstehen. Genauer gesagt: Der 68000

gibt seine Daten an diese Chips weiter, die dann den Rest erledigen - zum Beispiel ein Bild auf dem Monitor erzeugen, Daten auf der Diskette speichern usw. Umgekehrt funktioniert das natürlich auch. In diesem Fall holt der 68000 die vorher gewünschten Daten einfach an den Schnittstellenbausteinen ab.

Oben habe ich von einem etwas vereinfachten Blockschaltbild gesprochen. Aus gutem Grund, denn Atari hat dem 68000 drei weitere ICs zur Seite gestellt, die für Spezialaufgaben zur Verfügung stehen: GLUE, SHIFTER und MMU. Der GLUE sorgt zum Beispiel dafür, daß alle Speicherzugriffe auch tatsächlich bei dem Baustein landen, der die gewünschte Adresse beherbergt. Auch die Interrupts stammen von hier und vieles mehr. MMU steht für Memory Management Unit. Sie regelt den Zugriff auf den Speicher, da es vorkommen kann, daß mehrere Bausteine dies gleichzeitig tun möchten. Hinter dem Shifter verbirgt sich der oben erwähnte Video-Chip.

19. Die ICs des Atari ST

19.1 Der Prozessor 68000

Das Herz des gesamten Atari ST-Systems ist der Mikroprozessor MC68000. Dieser 16-Bit-Prozessor zählt zur absoluten Spitzenklasse, obwohl er langsam von seinen Nachfolgern (68020, 68030) verdrängt wird. Sowohl Programmierer als auch Hardware-Designer finden in ihm einen gleichermaßen einfach wie komfortabel handzuhabenden Prozessor. Obwohl die Entwicklung des 68000 bereits im Jahre 1977 bei der Firma MOTOROLA begann und erste Prototypen schon 1979 verfügbar waren, setzte sich der 68000 doch erst einige Jahre später erfolgreich gegen seinen schärfsten Konkurrenten, den 8086/8088 von INTEL durch. Maßgeblich für den Erfolg der INTEL-Prozessoren ist sicher die Tatsache, daß sie in den IBM-PCs und den diversen Nachbauten eingesetzt sind. Die Zahl der (preiswerten) mit einem 68000 ausgerüsteten Computer war jedoch bis zum Erscheinen der Atari ST-Serie gleich Null. Jetzt aber sind diese leistungsstarken 16-Bitter in eine Preisklasse gerutscht, die bislang vornehmlich 8-Bit-Rechnern vorbehalten war.

Was aber sind nun die hervorstechenden Merkmale des MC68000? In Schlagworten ausgedrückt läßt sich die folgende (sicher unvollständige) Liste aufstellen:

- 16 Daten-Bits
- 24 Adreßbits (16 Megabyte Adreßbereich!)
- alle Signale direkt ohne Multiplexer verfügbar
- problemloser Betrieb "alter" 8-Bit-Peripherie
- leistungsfähige Maschinenbefehle
- einfache und leicht zu lernende Assembler-Syntax
- 14 verschiedene Adressierungsarten
- 17 Register mit einer Breite von 32 Bit

Die genannten und weitere, vorläufig unerwähnte Leistungsmerkmale machen den 68000 zu einem hervorragenden Mikroprozessor für Personal- und Homecomputer. Nachdem auch die Preise für Halbleiterspeicher immer weiter sinken, die Kapazität der verfügbaren Speicher jedoch ständig steigt, bekommen die 8-Bit-Rechner mit 64 KByte Speicher zunehmend Konkurrenz durch den 68000. Bei diesen neuen Rechnern sind Speichergrößen üblich, die noch vor wenigen Jahren der mittleren Datentechnik vorbehalten waren.

19.1.1 Die Register des 68000

Betrachten wir zunächst die Register des 68000. Die Darstellung Bild 19.1 zeigt nicht nur die 17 vorhandenen 32-Bit-Register, sondern auch den Program-Counter und das Status-Register.

Die acht vollständig identischen Datenregister können zur Aufnahme und Berechnung von Daten wie auch von Adressen dienen. Sie stellen gewissermaßen die von den 8-Bit-Prozessoren bekannten Akkumulatoren dar. Man verfügt also insgesamt über acht Akkus, eine Erweiterung, die besonders Maschinenprogrammierer zu schätzen lernen werden. Der Zugriff auf die Daten in den Registern ist sehr flexibel. So können Daten mit 1-, 8-, 16- und 32-Bit-Größen verarbeitet werden. Auch eine (allerdings auf BCD-Zahlen beschränkte) 4-Bit-Verarbeitung ist möglich. Bei der Verarbeitung von Daten auf Bit-Ebene kann jedes der 32 Bits der Register gezielt beeinflusst werden. Bei Verarbeitung von 8- und 16-Bit-Daten kann allerdings nur auf die unteren 8 bzw. 16 Bit der Datenregister direkt zugegriffen werden.

Bei den Adreßregistern kann nicht ganz so flexibel auf die Daten zugegriffen werden, wie dieses bei den Datenregistern möglich ist. Allerdings werden sie in der Hauptsache zur Adressierung und nicht zur Berechnung von Daten genutzt. Eine Bearbeitung der Daten ist nur mit Wort- (16 Bit) und Langwort-Operationen (32 Bit) möglich.

Die Adreßregister müssen in zwei Gruppen unterschieden werden. Allgemein nutzbar sind die Adreßregister A0 bis A6. Die Register A7 und A7' erfüllen einen speziellen Zweck. Diese Register werden vom Prozessor als Stackpointer benutzt. Die Ursache für das Vorhandensein zweier Stackpointer ist durch den Umstand begründet, daß der 68000 in den zwei Betriebsarten USER MODE und SUPERVISOR MODE operieren kann. Je nach Betriebsart wird der User Stackpointer oder der Supervisor Stackpointer als Register A7 angesprochen. Beachten Sie, daß tatsächlich zwei unterschiedliche Register "unter" A7 arbeiten, jedoch nur das der jeweiligen Betriebsart zugeordnete als Registerinhalt verfügbar ist. Die Unterschiede zwischen den beiden Betriebsarten werden etwas später geklärt.

Auch der Program-Counter ist als 32-Bit-Register ausgeführt. Damit wäre theoretisch ein Adreßbereich von über 4 Gigabyte möglich. Beim 68000 werden jedoch die Adreßbits A24 bis A31 nicht benutzt. Dadurch ist der durchgängig anzusprechende Adreßbereich des 68000 auf 16 Megabyte "beschränkt".

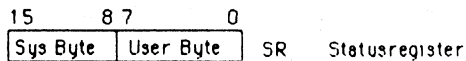
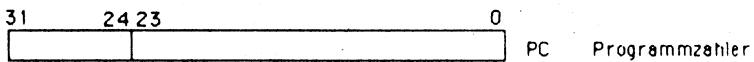
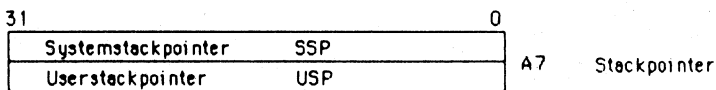
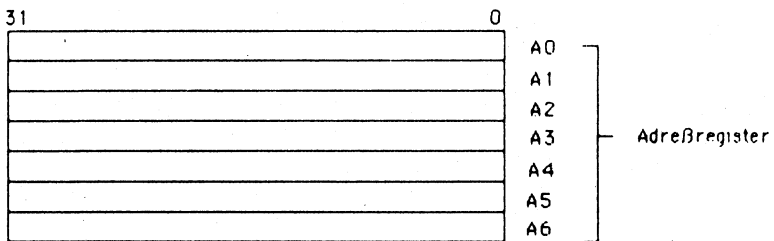
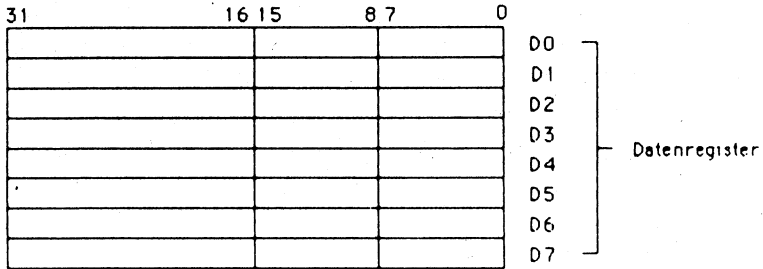
68000 Registersatz

Abb. 19.1: Die Register des 68000

Das Statusregister des 68000 umfaßt 16 Bits. Allerdings werden von diesen nur zehn Bits genutzt. Das Statusregister ist in zwei Hälften geteilt. Die unteren acht Bits (eigentlich nur die Bits 0 bis 4) stellen das sogenannte User Byte dar. In diesen Bits, oft als Flags bezeichnet, werden Ergebnisse von arithmetischen und Vergleichsoperationen angezeigt und können zu ergebnisabhängigen Programmverzweigungen genutzt werden. Die Erklärung der einzelnen Bits im User Byte würde an dieser Stelle zu weit führen. Wir wollen uns mit einer kurzen Aufstellung begnügen:

Bit 0	Carry-Flag
Bit 1	Overflow-Flag
Bit 2	Zero-Flag
Bit 3	Negativ-Flag
Bit 4	eXtend-Flag

Das System-Byte des Statusregisters belegt die Bits 8 bis 10, 13 und 15. Die restlichen im Statusregister nicht belegten Bits ergeben beim Lesen der Daten immer Null.

Das Bit 15 des Statusregisters wird als "Trace-Bit" bezeichnet. Dieses Bit erlaubt einen rein software-gesteuerten "Single Step" (Einzelschritt) durch ein beliebiges Programm.

Das Bit 13 ist das "Supervisor-Bit". Ist das S-Bit gesetzt, so befindet sich der 68000 im Supervisor Mode. Diese Betriebsart ist üblicherweise dem Betriebssystem vorbehalten. Nur in dieser Betriebsart kann der Prozessor alle Befehle ausführen. Im User Mode, in dem üblicherweise die Anwenderprogramme laufen, sind die sogenannten privilegierten Befehle nicht nutzbar. Durch einen speziellen Hardware-Aufbau kann der Zugriff auf Speicherbereiche (z.B. wichtige Systemvariablen, I/O-Bereich) im User Mode untersagt werden. Auch kann das System-Byte des Statusregisters nur im Supervisor Mode manipuliert werden. Damit kann auf einfache Weise vom Supervisor in den User Modus gewechselt werden. Umgekehrt ist diese Möglichkeit natürlich nicht gegeben, um das Konzept der strikten Trennung zwischen den beiden Bereichen nicht zu unterlaufen.

Die Bits 8 bis 10 werden als Interrupt-Maske bezeichnet und stehen in direktem Zusammenhang mit den Pins IPL0 bis IPL2.

Die Möglichkeiten, mit dem 68000 Interrupts zu verarbeiten, sind sehr leistungsstark. Insgesamt existieren sieben verschiedene Interrupt-Prioritäten. Die oberste Priorität besitzt der "Nicht Maskierbare Interrupt", NMI. Dieser Interrupt wird erkannt, wenn alle drei IPL-Anschlüsse gleichzeitig Low werden. Sind dagegen alle drei IPL-Anschlüsse auf High-Pegel, so wird dieses als

kein Interrupt, also normaler Betrieb interpretiert. Die verbleibenden sechs Möglichkeiten können mit Hilfe der drei Bits im System-Byte des Statusregisters maskiert werden. Der 68000 verfügt also über insgesamt sechs maskierbare Interrupt-Ebenen. Ist in der Interrupt-Maske z.B. das Bit I2 gesetzt, I0 und I1 dagegen gelöscht, so werden nur Interrupts der Ebenen 7, 6 und 5 (Kombinationen 000, 001 und 010) vom Prozessor erkannt und bearbeitet. Alle anderen Bit-Kombinationen an den Anschlüssen IPL0 bis IPL2 werden vom Prozessor ignoriert.

19.1.2 Die Exceptions des 68000

Haben wir bisher von Interrupts gesprochen, so nur aus dem Grunde, daß dieser Begriff von anderen Prozessoren her üblich ist. Dabei sind die durch äußere Signale ausgelösten Interrupts nach MOTOROLA-Nomenklatur eine spezielle Art von Exception, übersetzt etwa Ausnahme. Grundsätzlich gibt es beim Betrieb des 68000 nicht nur die Unterscheidung zwischen User- und Supervisormodus, sondern auch die Unterscheidung zwischen normalem Betrieb und Ausnahmebehandlung. Eine solche Ausnahme bilden die schon bekannten Interrupts. Weitere Ursachen für Exceptions sind unter anderem das Auftreten undefinierter Opcodes, Wort- oder Langwortzugriffe auf ungerade Adressen, aber auch das Ausführen spezieller Opcodes, die gewissermaßen Software-Interrupts darstellen. Um das Bearbeiten der vielen verschiedenen Exceptions einfach zu gestalten, sind beim 68000 die ersten 1024 Bytes des Speichers (\$000000 bis \$0003FF) reserviert. In diesem Bereich liegt die Exception-Tabelle. Jeder Exception wird ein aus vier Bytes bestehendes Langwort zugeordnet. Tritt eine Exception auf, so holt der 68000 selbsttätig den Inhalt der zugehörigen Speicherplätze und setzt die Programmausführung an der enthaltenen Adresse fort. Einzige Ausnahme ist der ebenfalls den Exceptions zugeordnete Reset, dem insgesamt acht Bytes, also zwei Langworte zugeordnet sind. Im ersten Langwort ist üblicherweise der Initialwert des Supervisor Stackpointers enthalten. Das zweite Langwort enthält die Adresse der eigentlichen Reset-Routine. Der Aufbau und die Belegung der Exception-Tabelle ist weiter unten ausgeführt.

19.2 Die Atari-Custom Chips

Insgesamt vier von Atari speziell entwickelte ICs versehen im ST ihren Dienst. Diese Chips mit den Namen GLUE, MMU, DMA und SHIFTER spielen eine enorme Rolle für die ausgesprochene Preiswürdigkeit des Systems. In diesen Chips sind viele hundert Gatter-Funktionen integriert. Die ersten Prototypen

des ST bestanden aus fünf ca. 50 mal 30 cm großen, fast vollständig mit TTL-ICs bestückten Leiterplatten. Erst durch die Integration dieser vielen TTL-Gatter in vier ICs wurde die Serienproduktion des ST möglich und rentabel. Jetzt gibt es nur noch acht TTL-ICs im Atari ST.

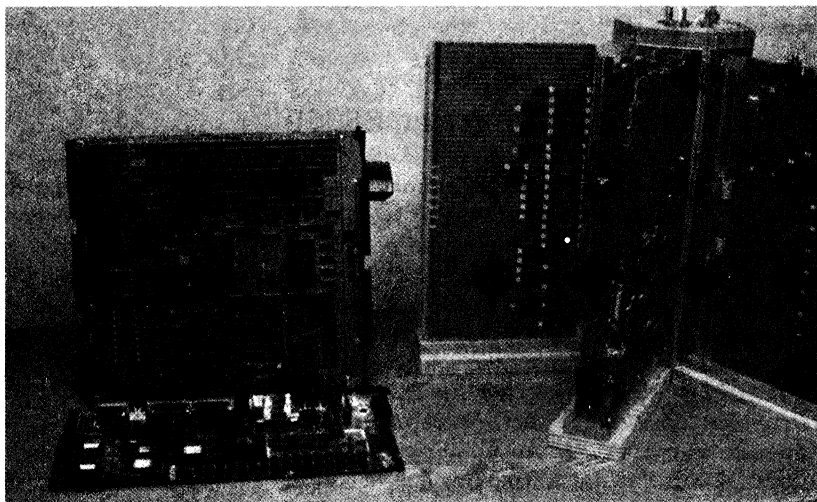


Abb. 19.2: Die drei Entwicklungsstufen des ST

Auf dem Photo sind die drei Entwicklungsstufen des ST zu sehen. Der große Turm zeigt den ersten real existierenden ST, der noch vollständig mit TTL-Bausteinen aufgebaut war. Dieser Rechner hat tatsächlich funktioniert und diente zum Optimieren und Erstellen der ersten Software. Die kleine Platine im Vordergrund ist ein ST, wie er in den ersten Monaten von Atari ausgeliefert wurde. Die größere Platine dahinter stellt eine Zwischenstufe der Entwicklung dar, in der verschiedene Funktionen schon in PLAs zusammengefaßt waren.

Da die ICs speziell für den Atari ST entwickelt wurden, ist natürlich kein Datenblatt von Atari erhältlich. Aber auch ohne Datenblätter können wir Ihnen eine Menge Informationen über die Funktion der Chips mitteilen.

Da die Chips nach einem funktionierenden Prototypen entwickelt wurden, ergeben sich einige interessante Fakten. So sind die ICs praktisch nur im Ver-

bund arbeitsfähig. Der DMA-Chip z.B. kann nicht allein in einem beliebigen anderen Computersystem als DMA-Controller eingesetzt werden, da ihm eine wesentliche Funktion fehlt. Er besitzt keinen eigenen Adreßzähler und kann somit auch nicht selbsttätig den Speicher adressieren. Diese Funktion wird von der MMU übernommen. Genauso ist es dem SHIFTER ergangen. Dieses für die Erzeugung des Video-Bildes und der Farben verantwortliche IC hat ebenfalls keine Möglichkeit, das Video-RAM zu adressieren. Auch hier springt die MMU hilfreich ein, indem sie die Adressierung übernimmt.

Letztendlich kann es dem Systemprogrammierer ziemlich egal sein, in welchem IC sich welches Register befindet. Wesentlich ist nur, daß die Adressen der Register bekannt sind und man weiß, was mit ihnen zu steuern ist. Darum wollen wir in diesem Kapitel nur kurz auf die Anschlüsse der einzelnen ICs eingehen. Sie sollen ja nicht wie ein "Ochs vorm Berg" sitzen und sich fragen, wie um alles in der Welt das Ganze funktioniert.

Das sicher wichtigste IC der "Vierer-Bande" ist der GLUE. Die Bezeichnung deutet schon auf die Funktion: "Glue" bedeutet etwa Leim oder Kleister. Dieses IC mit seinen insgesamt 68 Anschlüssen hält das gesamte System zusammen, indem es die annähernd vollständige Decodierung der Adreßbereiche für das RAM und die Peripherie-ICs vornimmt. Die Decodierung ist auch der Grund, weshalb dieses IC als einziges im gesamten System den vollständigen Adreß-Bus zur Verfügung hat.

Weiterhin werden die DMA-Handshake-Signale BR, BG und BGACK vom GLUE erzeugt bzw. ausgewertet. Die Zeitpunkte der DMA-Anforderung werden dem GLUE durch Signale vom DMA-Controller diktiert. Interessanterweise verfügt der GLUE neben dem BG ("Bus Grant")-Eingang auch noch über einen BGO, "Bus Grant Out". Das läßt erkennen, daß die ICs schon in Hinblick auf eine eventuelle spätere Erweiterung konzipiert wurden.

Auch die Interrupt-Signale werden vom GLUE erzeugt. Im ST werden nur die Anschlüsse IPL1 und IPL2 verwendet. Ohne zusätzliche Hardware kann also kein NMI (Interrupt-Level 7) ausgelöst werden. Die Anschlüsse MFPINT und IACK sind ebenfalls für die Interrupt-Steuerung zuständig.

Die Function Code-Anschlüsse sind an den GLUE geführt, da er auch die Bereichsüberprüfung und Zugriffsberechtigung bei Speicherzugriffen vornimmt. Dazu muß natürlich das BERR-Signal vom GLUE im Fehlerfall erzeugt werden. Aber auch DTACK und VMA/VPA können vom GLUE erzeugt werden. VPA ist besonders in Hinblick auf die Chip Select-Signale für die Peripherie-ICs nötig.

Als Takt-Frequenz erhält der GLUE eine Frequenz von 8 MHz. Durch Teilerketten werden die Frequenzen 2 MHz (Betriebsfrequenz des Soundchip) und 500 kHz (Taktfrequenz für Tastatur- und Midi-ACIA) erzeugt.

Auch die Signale HSYNC, VSYNC, BLANK und DE (Display Enable) für den Betrieb des Monitors werden durch Teilerketten im GLUE erzeugt. Allerdings kann die Erzeugung der Synchron-Takte im GLUE ausgeschaltet werden. In diesem Fall können externe Sync-Signale am Monitor-Anschluß eingespeist werden. Damit kann das Bild des ST mit dem Bild einer Video-Kamera synchronisiert werden.

Auch die MMU verfügt über insgesamt 68 Anschlüsse. Dieses IC übernimmt hauptsächlich drei wesentliche Aufgaben. Die wichtigste Aufgabe ist die Kopplung des gemultiplexten Adreß-Busses der dynamischen RAMs an den Bus des Prozessors. Dazu sind die Adreßleitungen A1 bis A21 an die MMU gelegt. Das ergibt einen Adreßbereich von insgesamt 4 Megabyte. Tatsächlich ist sowohl Hardware wie auch Betriebssystem des ST für solche Speichergrößen ausgelegt. Zum Betrieb der dynamischen RAMS werden die Signale RAS0, RAS1, CAS0L, CAS0H, CAS1L und CAS1H sowie der gemultiplexte Adreß-Bus aus der MMU herausgeführt. Natürlich sind auch die Signale DTACK, R/W, AS, LDS und UDS an die MMU geführt beziehungsweise aus ihr herausgeführt.

Eine weitere wesentliche Aufgabe der MMU wurde bereits erwähnt. Sie unterstützt den SHIFTER beim Erzeugen der Video-Signale, indem sie die Bildinformation im RAM des ST adressiert und dem SHIFTER übergibt. Zu diesem Zweck ist eine Zählerkette in der MMU integriert, die mit einem Startwert geladen wird und kontinuierlich alle 500 Nanosekunden ein Wort im Speicher adressiert und die enthaltene Information dem SHIFTER mit dem Signal DCYC übergibt. Der Startwert des Video-Zählers und damit die Lage des Bildschirmspeichers kann beliebig in 256-Byte-Schritten verschoben werden.

Eine weitere in der MMU integrierte Zählerkette ist, wie bereits kurz erwähnt, für die Adressierung des Speichers bei DMA zuständig. Dieser Zähler wird zu Beginn einer DMA-Operation (Disk oder Harddisk) mit der Adresse geladen, in den oder aus dem die Daten transportiert werden. Mit jedem übertragenen Wort wird der Zähler automatisch erhöht.

Der SHIFTER hat die Aufgabe, aus der im Video-RAM gespeicherten Information Impulse für den Monitor zu erzeugen. Wird der ST in der Auflösungsstufe 640 x 200 oder 320 x 200 betrieben, dann werden auch die Farbinformationen vom SHIFTER erzeugt.

Die Information aus dem RAM gelangt mit dem Signal LOAD über den Daten-Bus in den SHIFTER. Bei einer Auflösung von 640 x 400 Punkten wird das erzeugte Video-Signal am MONO-Anschluß ausgegeben. Da in dieser Auflösung keine Farben möglich sind, sind die jeweils dreifach vorhandenen RGB-Ausgänge inaktiv. In den beiden anderen Auflösungen ist dagegen der MONO-Ausgang inaktiv, jetzt wird die Bildinformation aus den Signalen an den RGB-Ausgängen gebildet.

Die drei zu einer Farbe gehörenden Ausgänge arbeiten zusammen mit externen Widerständen als Digital/Analog-Wandler. Je nach eingestellter Farbe ergeben sich an den Ausgängen unterschiedliche Pegel, die vom Monitor in unterschiedliche Farben umgesetzt werden.

Über die mit dem Adreß-Bus verbundenen Anschlüsse R1 bis R5 werden die sogenannten Palettenregister angesprochen. In den Palettenregistern sind die Werte der Farben enthalten, die den einzelnen Bit-Mustern zugeordnet sind. Insgesamt sind 16 Paletten-Register entsprechend den maximal 16 gleichzeitig darstellbaren Farben im 320 x 200-Modus vorhanden. Da jedoch alle 16 Farben aus den drei Grundfarben Rot, Grün und Blau in jeweils acht Stufen erzeugt werden können, beträgt die Gesamtzahl der darstellbaren Farben 512.

Der DMA-Controller ist wie der SHIFTER in einem handelsüblichen 40-poligen Gehäuse untergebracht. Seine Aufgaben sind die Ansteuerung des Floppy-Controllers, der Harddisk und eventueller weiterer Peripherie (z.B. Netzwerk).

Die Geschwindigkeit des Datentransfers beim Betrieb von Floppy-Laufwerken könnte noch ohne Probleme vom Prozessor übernommen werden. Bei Harddisks jedoch ist die Geschwindigkeit des Datentransfers so hoch, daß selbst einem mit 8 MHz getakteten 68000 die "Puste" ausgeht. Diese hohen Geschwindigkeiten sind nur noch mittels DMA erzielbar.

Da der DMA-Controller Daten transportiert, ist er mit dem Daten-Bus des Prozessors verbunden. Intern verfügt er über zwei Register, von denen eines quasi als bidirektionaler Buffer der Daten zum DMA-Port arbeitet. Die genaue Verwendung der Register werden wir später besprechen. Interessant ist jedoch die Tatsache, daß der 16-Bit-Daten-Bus des Prozessors auf einen 8-Bit-Daten-Bus auf der Floppy- und Harddisk-Seite reduziert wird. Beim Datentransfer werden automatisch je nach Richtung zwei Bytes zu einem Wort zusammengefügt oder ein Wort in zwei Bytes zerlegt und übertragen.

Zur Ansteuerung des Floppy-Controllers dienen die Signale CA1 und CA2, CR/W und FDCS sowie das vom Floppy-Controller gelieferte Signal FDRQ. CA1 und CA2 stellen Register-Auswahlsignale dar, mit denen die Register des

FDC (Floppy Disk Controller) selektiert werden können. CR/W bestimmt die Richtung der Datenübertragung von und zum FDC, aber auch zu weiterer am DMA-Port angeschlossener Peripherie.

Das Signal RDY dient zur Kommunikation mit dem GLUE (DMA-Anforderung) und der MMU (Adreßzähler). Mit diesem Signal zeigt der DMA an, daß ein Wort übertragen werden soll.

Wie man sieht, ist ein separater Betrieb der einzelnen ICs annähernd unmöglich. Alle Funktionen sind so ineinander verzahnt, daß sich bei fast allen Operationen die einzelnen ICs ergänzen.

19.2.1 Die Register der Atari-Custom-Chips

Wie bereits erwähnt, lassen sich die Register der Customchips keinen bestimmten ICs zuordnen. Diese Zuordnung ist aber auch nicht relevant. Wichtiger für den Systemprogrammierer sind die zur Verfügung stehenden Adresgruppen und Funktionen. Die einzelnen Register befinden sich nach Funktionsgruppen zusammengefaßt in verschiedenen, jeweils 512 Byte großen Adreßbereichen.

Auf den folgenden Seiten finden Sie eine Übersicht über alle derzeit bekannten Register mit detaillierter Beschreibung.

\$FF8000 Speicherkonfiguration

In diesem Bereich existiert ein einziges 8-Bit-Register auf der Adresse \$FF8001. In ihm wird in den unteren vier Bits die Speicherkonfiguration eingestellt. Bevor wir jedoch auf die möglichen Einstellungen eingehen, kurz noch ein paar Worte zu den von der Hardware des ST vorgegebenen Möglichkeiten.

Das MMU-IC wurde von den Atari-Entwicklern auf maximale Vielseitigkeit konstruiert. So lassen sich im ST je nach gewünschtem Speicherausbau drei verschiedene Speichertypen verwenden. Es sind dies die Typen 64-KBit-, 256-KBit- und die 1-Megabit-Chips.

Da alle diese ICs Bit-orientiert sind, werden für den Speicherausbau jeweils 16 Speicher-ICs benötigt. Die Bezeichnung für 16 solcher ICs (unabhängig von der Speicherkapazität) ist Bank. Damit ergeben sich die möglichen Ausbaustufen zu 128 KByte, 512 KByte oder 2 Megabyte.

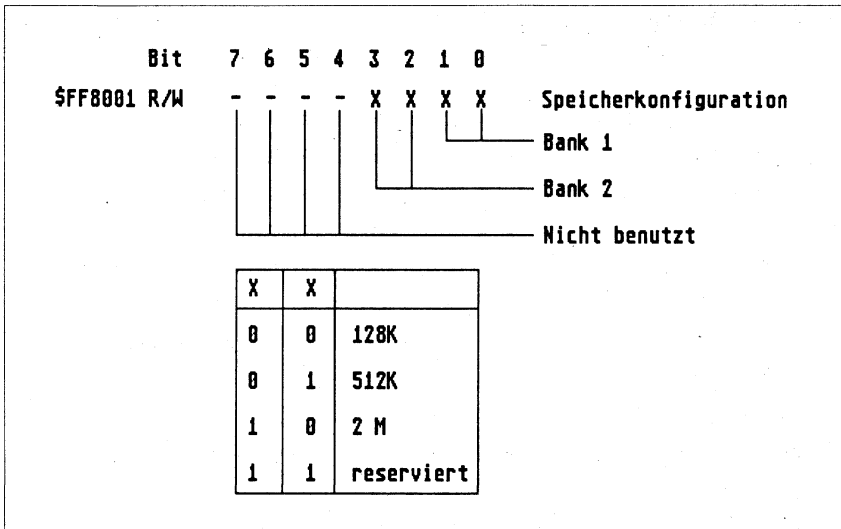


Abb. 19.3: Die möglichen Speicherkonfigurationen

Allerdings kann die MMU bei Bedarf auch zwei getrennte Banks ansteuern. Die für die zweite Bank benötigten RAS- und CAS-Signale sind an der MMU vorhanden. Da die Bestückungen in den Banks unterschiedlich sein dürfen, ergeben sich viele Kombinationsmöglichkeiten.

Die wesentliche Aufgabe des Konfigurationsregisters ist es, die vielen möglichen Speicherausbauten jeweils zu einem durchgehend adressierbaren Speicherbereich zu verbinden. Ohne diese Fähigkeit der MMU würde bei Bestückung mit zwei Banks zu je 512 KByte eine große Lücke zwischen den beiden Speicherbereichen entstehen. Das Speicherkonfigurationsregister kann gelesen und beschrieben werden.

\$FF8200 Register für Video-Anzeige

Bit	7	6	5	4	3	2	1	0	
\$FF8201 R/W	X	X	X	X	X	X	X	X	Videoram Highbyte
\$FF8203 R/W	X	X	X	X	X	X	X	X	Videoram Midbyte
\$FF8205 R/-	X	X	X	X	X	X	X	X	Videaddress Counter High
\$FF8207 R/-	X	X	X	X	X	X	X	X	Videaddress Counter Mid
\$FF8209 R/-	X	X	X	X	X	X	X	X	Videaddress Counter Low
\$FF820A R/W	-	-	-	-	-	-	X	X	Sync Modus
									0 = intern / 1 = externer Sync.
									0 = 60Hz / 1 = 50Hz

Abb. 19.4: Die Register für die Video-Anzeige

In den Registern dieses Adreßbereichs werden die Lage des Video-RAMs, die eingestellte Auflösung des Bildschirms und die Werte der Farbpalette gespeichert.

Die zwei Byte-Register an den Adressen "\$FF8201" und "\$FF8203" bestimmen den Anfang des 32-KByte-Video-RAMs im gesamten Speicher. Um das Video-RAM an jeder beliebigen Adresse beginnen zu lassen, müßte ein weiteres Register für das Lowbyte vorhanden sein. Ein solches Register fehlt jedoch, als Wert wird immer 0 angenommen. Dadurch kann das Video-RAM nur in Stufen von 256 Bytes in seiner Lage verschoben werden. Nach dem Einschalten wird das Video-RAM an die obere Speichergrenze gelegt. Dadurch ergeben sich je nach Speicherausbau die Anfangsadressen "\$78000", "\$F8000", "\$1F8000" oder "\$3F8000".

Die drei Byte-Register "\$FF8205" bis "\$FF8209" können nur ausgelesen werden. Sie geben Auskunft über die derzeit dargestellte Videoadresse. Im ST wird der Videospeicher vollkommen linear abgetastet. Alle zwei Mikrosekunden wird die durch die drei Register erzeugte Adresse um zwei erhöht.

Im Sync-Mode-Register ("FF820A") sind nur die Bits 0 und 1 belegt. Die restlichen Bits werden nicht genutzt. Ist das Bit 0 gesetzt, so werden die intern erzeugten HSync- und VSync-Impulse ausgeschaltet. Dann kann die Synchronisation des Bildes von außen über die Monitor-Buchse zugeführt werden. Diese Option schafft ganz neue Möglichkeiten, besonders im Bereich der Videofilmtechnik. Da sich auf einfache Weise die Bilder einer Video-Kamera und des ST

synchronisieren lassen, kann der ST zum Einblenden von Schrift oder Grafik in einen Film eingesetzt werden.

Bit 1 des Sync-Mode-Registers bestimmt die Bildfrequenz. Allerdings ist dieses Bit nur in den beiden "niedrigen" Auflösungsstufen wirksam. In der hochauflösenden Betriebsart arbeitet der ST immer mit ca. 70 Hertz Bildfrequenz, die vom Atari-Monitor vorgeschrieben werden, um die hohe Auflösung auch flimmer- und verzerrungsfrei darzustellen.

Das Sync-Mode-Register kann sowohl gelesen auch beschrieben werden.

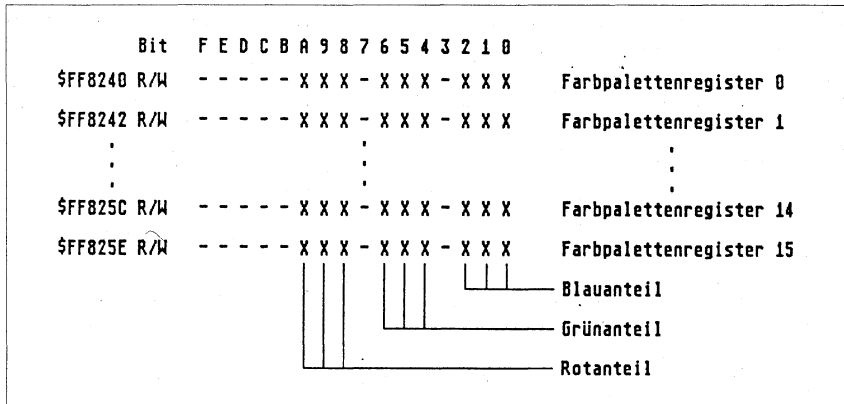


Abb. 19.5: Das Sync-Mode-Register

Insgesamt verfügt der ST über 512 darstellbare Farben, kann aber nur maximal 16 unterschiedliche Farben zu einer Zeit auf dem Bildschirm darstellen. Diese Beschränkung kann so erklärt werden, daß man sagt, der Anwender habe 16 Farbstifte zur Verfügung, und jeder Farbstift könne eine von 512 Farben haben. Die 16 Bit großen Farbpalettenregister im Adreßbereich "\$FF8240" bis "\$FF825E" stellen nun die 16 Stifte dar. Die Codierung der Farben geschieht bitweise nach einem festen Schema. Jedes der 16 Register enthält 9 die Farbe beeinflussende Bits, deren Organisation in der Abbildung 19.5 angegeben ist. Die durch ein x gekennzeichneten Stellen in dem Registerwort haben keine Auswirkung. Die Bits 0 bis 2 bestimmen den Blau-Anteil der gewünschten Farbe, Bit 4 bis 6 sind für den Grün-Anteil und Bit 8 bis A für den Rot-Anteil zuständig. Je höher der Wert in den drei Bits ist (0 bis 7 sind ja jeweils möglich), desto intensiver wird die resultierende Farbe.

In der mittleren Auflösungsstufe sind bei einer Grafik-Auflösung von 640 x 200 Punkten nur vier verschiedene Farben gleichzeitig darstellbar. Dann werden die Palettenregister 4 bis 15 ignoriert.

Wenn man von maximal 16 darstellbaren Farben spricht, so ist das zwar grundsätzlich richtig, der Inhalt des Palettenregisters 0 jedoch bestimmt in allen Auflösungsstufen die Farbe des Hintergrundes.

Bei der höchsten Auflösungsstufe mit 640 x 400 Punkten kann nur eine "Farbe" gewählt werden. In diesem Fall bestimmt das Bit 0 des Palettenregisters 0 die Hintergrundfarbe. Ist das Bit gesetzt, so erscheint die Schrift schwarz auf hellem Hintergrund, bei gelöschtem Bit 0 wird das Bild invertiert, d.h. es erscheinen helle Buchstaben auf dunklem Hintergrund.

Die Farbgregister können sowohl gelesen als auch beschrieben werden.

Bit	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
\$FF8260 R/W	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X
Videomodus																

X	X	Auflösung
0	0	320*200 Pixel , 4 Ebenen
0	1	640*200 Pixel , 2 Ebenen
1	0	640*400 Pixel , 1 Ebene
1	1	reserviert

Abbildung 19.6: Einstellung der Hardware auf die gewünschte Grafikauflösung

Das Register auf der Adresse "\$FF8260" stellt die Hardware auf die gewünschte Grafik-Auflösung ein.

Das Auflösungsregister kann sowohl gelesen als auch beschrieben werden.

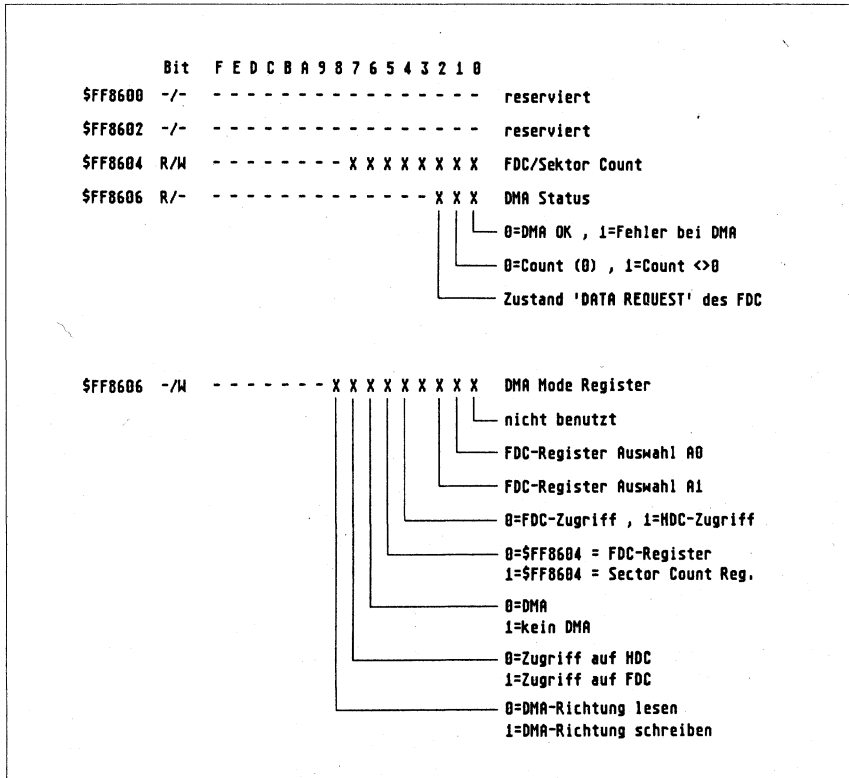
\$FF8600 DMA/Disk-Controller

Abb. 19.7.: Die Register des DMA-/Disk-Controllers

Der DMA-/Disk-Controller-Bereich beinhaltet alle Register, die in irgendeiner Weise mit DMA und Diskettenbetrieb zu tun haben. Dazu gehören Zugriff auf die Register des FDC, Status-Informationen über den Erfolg einer DMA-Operation und DMA-Basis-Adreßregister.

Über die unteren acht Bits der Adresse \$FF8604 kann der Prozessor wahlweise auf die Register des FDC oder auf das Sektor-Count-Register zugreifen. Die oberen acht Bit enthalten keine Information. Welches Register des FDC über "\$FF8604" angesprochen wird, hängt vom Zustand des Bit 5 im DMA-Mode Control-Register "\$FF8606" ab.

Der Inhalt dieser Adresse kann gleichermaßen beschrieben wie gelesen werden. Der Zugriff muß ein Wort-Zugriff sein.

	Bit	7	6	5	4	3	2	1	0	
\$FF8609	R/W	X	X	X	X	X	X	X	X	DMA Basis und Counter High
\$FF860B	R/W	X	X	X	X	X	X	X	X	DMA Basis und Counter Mid
\$FF860D	R/W	X	X	X	X	X	X	X	X	DMA Basis und Counter Low

Abb. 19.8.: Die DMA-Basis-Register

Wird das Register an der Adresse "\$FF8606" gelesen, so erhält man den DMA-Status in den unteren drei Bit des Registers. Schreibzugriffe auf diese Adresse beeinflussen das DMA-Mode-Register. Auch hier sind einzelnen Bits bestimmte Aufgaben zugeordnet.

Die genannte Bedeutung für Bit 7 des DMA-Mode Registers ist leider fragwürdig. Zwar bezieht sich die Funktionsangabe auf die von Atari gelieferten Informationen. In der Praxis ist dieses Bit jedoch bei jedem Zugriff auf den DMA-Port oder den FDC gesetzt.

Bei DMA-Transfers muß der Hardware mitgeteilt werden, ab welcher Speicheradresse die Daten übertragen werden sollen. Diese Basis-Adresse wird in den in Abbildung 19.8 dargestellten Registern eingetragen. Mit jedem übertragenen Wort wird der Inhalt der Register entsprechend erhöht. Die Initialisierung der drei Register muß mit dem Low-Byte beginnen. Danach sind (in dieser Reihenfolge) das Mid- und High-Byte zu initialisieren.

19.3 Der Floppy-Disk-Controller WD 1772

Obwohl der 1772 von WESTERN DIGITAL über nur 28 Pins verfügt, enthält der Chip doch einen vollständigen Floppy-Disk-Controller (FDC) mit eingebauten Möglichkeiten, die so manchen 40-poligen FDC in den Schatten stellen. Dieses IC ist zu den bekannteren Controllern der Serien 179x/279x weitgehend softwarekompatibel. Leser, die mit Controllern dieser Serien Erfahrung haben, werden die Ähnlichkeiten schnell bemerken.

Im Telegrammstil lassen sich die folgenden Möglichkeiten des 1772 aufzählen:

- Einfache 5-Volt-Versorgung
- Eingebauter Datenseparator
- Eingebaute Schreib-Vorkompensationslogik
- Single und Double Density
- Eingebaute Motorkontrolle
- DMA-fähig

Obwohl dem Anwender in der Wahl der benutzten Diskettenformate, also z.B. Sektorlänge, Anzahl der Sektoren pro Track und Anzahl der Tracks pro Diskette, vom Controller fast keine Einschränkungen gemacht werden, ist das verwendete Format wegen der festgelegten Datenübertragung doch spezifiziert. So können z.B. Apple- oder Commodore-Disketten mit diesem Controller nicht verarbeitet werden.

Bevor wir jedoch auf die Möglichkeiten des FDC eingehen, wollen wir einen Blick auf die 28 Anschlüsse des ICs werfen.

19.3.1 Die Anschlüsse des 1772

Die Anschlüsse des FDC 1772 lassen sich grob in drei Gruppen zusammenfassen. Die erste Gruppe umfaßt die Versorgungsanschlüsse.

Vcc Stromversorgung + 5 Volt

GND Masseanschluß

MR Master Reset. Mit einem Low an diesem Anschluß wird der FDC in einen definierten Grundzustand versetzt.

Das Prozessor-Interface des FDC bildet die zweite Gruppe von Anschlüssen. Über diese Pins läuft (in der Hauptsache) der Datentransfer zwischen FDC und Prozessor.

D0-D7 Über diesen acht Bit breiten bidirektionalen Bus werden Daten, Kommandos und Statusinformationen zwischen FDC und Computersystem ausgetauscht.

CS Auf die Register des FDC kann nur zugegriffen werden, wenn der CS-Anschluß Low ist.

R/W "Read/Write". Der Pegel an diesem Anschluß bestimmt die Datenrichtung. Bei einem High können Daten aus dem FDC gelesen, bei einem Low in den FDC geschrieben werden.

A0,A1 Mit diesen Bits wird (in Verbindung mit R/W) eingestellt, auf welches Register des 1772 zugegriffen wird. Insgesamt enthält der 1772 fünf Register, die teilweise gelesen und beschrieben werden können. Andere Register lassen sich nur beschreiben oder auslesen. Die folgende Tabelle zeigt die vom Hersteller getroffene Zuordnung:

A1	A0	R/W=1	R/W=0
0	0	Status Reg.	Command Reg.
0	1	Track Reg.	Track Reg.
1	0	Sector Reg.	Sector Reg.
1	1	Data Reg.	Data Reg.

DRQ "Data Request". Dieser Ausgang zeigt mit einem High-Pegel an, daß entweder beim Lesen das Datenregister voll ist, also ausgelesen werden muß, oder daß beim Schreiben das Datenregister leer ist, also neu versorgt werden muß. Mit Hilfe dieses Anschlusses läßt sich ein DMA-Betrieb des FDC erzielen.

INTRQ

"Interrupt Request". Dieser Ausgang wird High, wenn ein Kommando beendet wurde. Zurückgesetzt wird der Ausgang mit dem Lesen des Status-Registers.

CLK "Clock". Das Clock-Signal zählt nur bedingt zum Prozessor-Bus. An diesem Eingang muß eine Frequenz von 8 MHz anliegen, über die das interne Timing des FDC bestimmt wird.

Die dritte Gruppe von Signalen umfaßt das Floppy-Interface. Über diese Anschlüsse ist eine fast vollständige Steuerung der Laufwerke möglich.

STEP Am Step-Ausgang liefert der 1772 einen Impuls für jeden Schritt des Kopfmotors.

DIRC "Direction". Mit diesem Anschluß wird entschieden, in welche Richtung der Kopf zu bewegen ist. Bei einem High an diesem Pin bewegt sich der Kopf zur Mitte der Diskette.

- RD "Read Data". An RD wird die von der Diskette gelieferte Information angelegt. Diese Information enthält gleichermaßen Takt- wie Datenimpulse, die im internen Datenseparator getrennt und aufbereitet werden.
- MO "Motor On". Über diesen Anschluß werden die Motoren der Laufwerke gesteuert. Die Steuerung geschieht in der Form, daß bei Lese-, Schreib- und Such-Operationen die Motoren automatisch gestartet werden.
- WG "Write Gate". WG wird Low, bevor auf die Diskette geschrieben wird. Ohne dieses Signal verriegeln die Laufwerke die Schreiblogik, so daß ein Beschreiben der Diskette unmöglich wird.
- WD "Write Data". Dieser Ausgang liefert den seriellen Datenstrom der zu schreibenden Daten- und Taktimpulse.
- TR00 "Track 00". Mit Hilfe dieses Eingangs kann der Controller ermitteln, ob der Schreib-/Lesekopf des selektierten Laufwerks auf der Spur Null plaziert ist. In diesem Fall wird TR00 vom Laufwerk auf Low gelegt.
- IP "Index Pulse". Das Index-Loch in der Diskette markiert den physikalischen Beginn eines jeden Tracks. Beim Formatieren der Diskette wartet der FDC zunächst auf den Anfang der Spur, bevor er mit dem Formatieren beginnt.
- WPRT "Write Protect". Bei den im Atari benutzten 3,5-Zoll-Disketten wird die Diskette mit Hilfe eines kleinen Schiebers vor unerwünschtem Beschreiben geschützt. Ob die Diskette schreibgeschützt ist, wird dem FDC an diesem Eingang mitgeteilt.
- DDEN "Double Density Enable". Dieses Signal paßt nur bedingt in die Gruppe der Floppy-Steuer-Signale. Mit dem Pegel an diesem Anschluß wird zwischen den Aufzeichnungsformaten "Single Density" und "Double Density" umgeschaltet.

19.3.2 Die Register des WD 1772

Bei der Beschreibung der Anschlüsse A1 und A0 wurden die internen Register bereits kurz genannt. Jetzt soll eine genauere Beschreibung der Funktion dieser Register folgen.

CR (Command Register)

In dieses 8-Bit-Register werden die vom FDC auszuführenden Kommandos geschrieben. Allerdings dürfen Kommandos nur dann in das CR geschrieben werden, wenn kein anderes Kommando in Arbeit ist. Obwohl der FDC "nur" elf verschiedene Kommandos kennt, ist eine solche Anzahl verschiedener Möglichkeiten vorhanden, daß die Kommandos im Detail etwas später behandelt werden. Das CR kann nur beschrieben werden.

STR (Status-Register)

Das Status-Register gibt Aufschluß über verschiedene Zustände des FDC. Dabei wird die Information in einzelnen Bits codiert. Da die Bedeutungen der Bits vom derzeitigen Kommando abhängen, werden wir sie bei der Kommando-Beschreibung einfügen. Das Status-Register kann nur gelesen werden.

TR (Track-Register)

Im TR ist die momentane Position des Schreib-/Lese-Kopfes enthalten. Mit jeder Bewegung des Kopfes wird das TR entsprechend der Bewegungsrichtung um eins erhöht oder verringert. Bei einigen Kommandos wird der Inhalt des TR mit den von der Diskette gelesenen Informationen verglichen. Das Ergebnis beeinflußt dann das Status-Register. TR kann sowohl gelesen als auch beschrieben werden.

SR (Sector-Register)

Das SR enthält bei Lese- und Schreiboperationen die Nummer des gewünschten Sektors. Wie auch TR kann SR gelesen und beschrieben werden.

DR (Data-Register)

Über das Data-Register werden die auf die Diskette zu schreibenden Daten an den FDC übergeben. Beim Lesen dagegen kann die Disketteninformation aus dem DR gelesen werden.

19.3.3 Die Programmierung des FDC

Die Programmierung des FDC stellt für den Systemprogrammierer keine große Schwierigkeit dar. Im ST wird die direkte (und sicher in den meisten Fällen überflüssige) Programmierung des FDC durch den Anschluß an den DMA-Baustein teilweise etwas erschwert, teilweise aber auch drastisch vereinfacht.

Die elf möglichen Kommandos des FDC lassen sich in vier Gruppen oder Typen unterteilen.

Type	Kommando
1	Restore, Spur Null suchen
1	Seek, Spur suchen
1	Step, einen Track in die bisherige Richtung
1	Step In, Kopf nach innen (zur Disk-Mitte)
1	Step Out, Kopf nach außen (zum Disk-Rand)
2	Read Sector, Sektor lesen
2	Write Sector, Sektor schreiben
3	Read Address, ID lesen
3	Read Track, ganzen Track lesen
3	Write Track, Track schreiben (formatieren)
4	Force Interrupt, Interrupt auslösen

Die Typ-1-Kommandos

Die Kommandos dieser Gruppe sind für die Positionierung des Schreib-/Lesekopfes zuständig. Die Bit-Muster dieser fünf Befehle finden Sie in der folgenden Aufstellung:

Bit	7	6	5	4	3	2	1	0
Restore	0	0	0	0	H	V	R1	R0
Seek	0	0	0	1	H	V	R1	R0
Step	0	0	1	U	H	V	R1	R0
Step In	0	1	0	U	H	V	R1	R0
Step Out	0	1	1	U	H	V	R1	R0

Alle fünf Kommandos haben mehrere variable Bits. So wird in den Bits R0 und R1 die Zeit zwischen zwei Step-Impulsen angegeben. Die möglichen Kombinationen sind:

R1	R0	Steprate
0	0	2 Millisekunden
0	1	3 Millisekunden
1	0	5 Millisekunden
1	1	6 Millisekunden

Diese Bits müssen entsprechend den Erfordernissen des Laufwerks im Kommando-Byte gesetzt werden. Das V-Bit in den Kommandos der Gruppe 1 ist das sogenannte "Verify-Flag". Wird dieses Flag im Kommandowort gesetzt, so führt der FDC nach einer Kopfbewegung einen automatischen Verify durch, d.h. er vergleicht die im Track-Register gespeicherte Track-Nummer mit der im ersten zu lesenden ID-Feld gespeicherten Track-Nummer. Stellt der FDC Übereinstimmung fest, so wird das Kommando ohne Fehler beendet.

Das H-Bit bestimmt, ob die sogenannte "Spin-Up-Sequenz" eingehalten wird. Da in Systemen mit dem 1772 die Laufwerksmotoren direkt vom FDC gesteuert werden, muß nach dem Einschalten eine gewisse Zeit gewartet werden, bis der Motor seine Nennumdrehungszahl von 300 UPM erreicht hat. Ist im Kommandowort das "H-Bit" gelöscht, so überprüft der FDC vor Aktivierung des Kommandos den Zustand des Motor-On-Pins. Ist der Motor nicht eingeschaltet, so wird dieser Anschluß auf High gelegt (Motor On), und der FDC wartet insgesamt sechs Umdrehungen (Index-Impulse), bevor das Kommando gestartet wird. Läuft der Motor jedoch, so wird die Wartezeit nicht eingelegt, da in diesem Fall die Nenndrehzahl wahrscheinlich erreicht ist.

Bei den drei verschiedenen Step-Kommandos hat das Bit 4 im Kommandowort die Bezeichnung "U-Bit". Bei gesetztem "U-Bit" wird jeder Step, also jeder Wechsel der Kopfposition, im Track-Register mitgeführt. Ist das "U-Bit" gelöscht, so unterbleibt das Update.

Die Typ-2-Kommandos

Die Kommandos dieser Gruppe umfassen das Lesen und Schreiben von Sektoren. Auch bei diesen Kommandos haben einzelne Bits unterschiedliche Bedeutungen:

Bit	7	6	5	4	3	2	1	0
Read Sector	1	0	0	M	H	E	O	0
Write Sector	1	0	1	M	H	E	P	A0

Das "H-Bit" (Bit 3) in den beiden Befehlen ist das schon bekannte Spin-Up-Bit. Ist das "E-Bit" gesetzt, so wird vom FDC eine Wartezeit von 30 Millisekunden eingelegt, bevor das Kommando gestartet wird. Diese Wartezeit wird bei einigen Laufwerken benötigt, um ein Schwingen der Kopfmechanik nach einem Spurwechsel ausklingen zu lassen. Ist das "E-Bit" auf Null, so wird mit dem Kommando unmittelbar begonnen.

Das "M-Bit" bestimmt, ob ein oder mehrere Sektoren nacheinander gelesen werden sollen. Ist das Bit Null, so wird nur jeweils ein Sektor gelesen oder geschrieben. Multi-Sektor-Lesen mit gesetztem "M-Bit" läuft so ab, daß der FDC nach jedem Sektor automatisch den Inhalt des Sektorregisters erhöht und den neuen Sektor liest.

Beim Sektorlesen müssen die Bits 0 und 1 gelöscht sein. Beim Schreiben jedoch haben sie eine spezielle Bedeutung. Das als A0-Bit bezeichnete Bit 0 bestimmt, ob ein gelöscht oder normales Data Adress Mark geschrieben wird. Da die weitaus meisten Betriebssysteme diese Möglichkeit nicht nutzen, wird in der Regel ein normales Data Adress Mark geschrieben. Das "P-Bit" (Bit 1) bestimmt, ob die Vorkompensation beim Schreiben der Daten ein- oder ausgeschaltet ist. Die Prekompensation wird üblicherweise eingesetzt, um auf den inneren Tracks einer Diskette eine größere Datensicherheit zu erlangen.

Die Typ-3-Kommandos

In dieser Gruppe sind die Kommandos "Read Adress", "Read Track" und "Write Track" zusammengefaßt.

Read Adress liefert dem Programmierer Informationen aus dem nächsten zu lesenden ID-Feld auf der Diskette. Das ID-Feld beschreibt in vier Bytes die Daten Track, Sektor, Diskettenseite und Sektorlänge eines jeden Sektors.

Read Track liefert alle Bytes, die beim Formatieren auf die Diskette geschrieben wurden, also auch die Daten, die sonst üblicherweise nicht zugänglich sind, da sie zwischen den Sektoren stehen.

Write Track schließlich formatiert eine Spur zur späteren Datenspeicherung. Die benötigten Bit-Muster für die Kommandos lauten:

Bit	7	6	5	4	3	2	1	0
Read Adress	1	1	0	0	H	E	0	0
Read Track	1	1	1	0	H	E	0	0
Write Track	1	1	1	1	H	E	P	0

"H-" und "E-Bit" geben wie auch bei den Kommandos der Gruppe 2 an, ob die Spin-Up-Sequenz und die Head-Settle-Time von 30 Millisekunden gewartet werden muß. Das "P-Bit" beim Formatieren hat dieselbe Funktion wie beim Schreiben von Sektoren, bestimmt also über die Prekompensation beim Formatieren.

Das Typ-4-Kommando

Diese Gruppe besteht aus nur einem Befehl, dem "Force" "Interrupt". Dieser Befehl kann als einziger zu jedem Zeitpunkt, also auch während der Bearbeitung eines anderen Kommandos, dem FDC übermittelt werden. Sobald der FDC dieses Kommando in einer der möglichen Variationen erhält, wird der laufende Befehl beendet. Das Bit-Muster des "Force-Interrupt" lautet:

Bit	7	6	5	4	3	2	1	0
Force Intr.	1	1	0	1	3	2	1	0

Die Bits I0 bis I3 geben an, welche Bedingungen den Interrupt auslösen sollen. I0 und I1 haben jedoch beim 1772 keine Bedeutung und sollten Low sein. Ist I2 gesetzt, so wird mit jedem Indeximpuls ein Interrupt erzeugt. Mit dieser Option könnte softwaremäßig die Drehzahl der Diskette ermittelt werden. Wird I3 gesetzt, so wird unmittelbar ein Interrupt erzeugt und das laufende Kommando beendet. Sind alle I-Bits Null, so wird der laufende Befehl ohne Interrupt beendet.

19.4 Der MFP 68901

MFP ist die Abkürzung für "Multi Function Peripheral". Sie werden zustimmen, daß der Name für dieses IC nicht zu hoch gegriffen ist, wenn Sie mit den Möglichkeiten dieses ICs vertraut sind.

In kurzen Stichworten können die folgenden Leistungsmerkmale genannt werden:

- 8-Bit-Parallelport
- Datenrichtung jedes Portbits separat programmierbar
- Verwendung der Portbits als Interrupt-Eingang möglich
- 16 mögliche Interrupt-Quellen

- vier universelle Timer
- serielle Schnittstelle eingebaut

Um alle Möglichkeiten des Chips unterzubringen, wurde ein 48-poliges Gehäuse notwendig. Wir wollen zunächst kurz auf die Bedeutung der verschiedenen Pins eingehen, bevor der interne Registeraufbau besprochen wird.

19.4.1 Die Anschlüsse des 68901

Die Anschlüsse des MFP lassen sich in Funktionsgruppen zusammenfassen. Die erste Funktionsgruppe besteht aus den Versorgungsanschlüssen.

GND, Vcc Über Vcc und GND wird der MFP mit der Stromversorgung verbunden.

CLK ist der Clock-Eingang des MFP. Dieses CLK-Clock-Signal muß nicht notwendigerweise mit dem Systemtakt des Prozessors übereinstimmen. Im Atari ST wird der MFP mit einer Frequenz von 4 MHz versorgt.

Der Anschluß an den Daten-Bus des Prozessors wird mit den Pins D0 bis D7, CS, DS, DTACK, RS1 bis RS5 und RESET vorgenommen.

D0-D7 Diese bidirektionalen Anschlüsse des 68901 werden normalerweise mit den unteren acht Daten-Bits des 68000 verbunden. Zwar wäre auch ein Anschluß an den Datenleitungen D8 bis D15 möglich. Dann aber wäre eine wesentliche Möglichkeit, die Erzeugung von Non-Auto-Interrupts, unmöglich. Bei dieser Interrupt-Methode wird der Interrupt-Vektor auf die unteren acht Daten-Bits des Busses gelegt.

CS ("Chip Select"). CS wird benötigt, um die internen Register des MFP ansprechen zu können. CS ist Low aktiv.

DS ("Data Strobe"). Der Anschluß DS wird entweder mit dem Prozessorsignal LDS oder UDS verbunden. Welches Signal verwendet wird, hängt davon ab, ob der MFP auf der oberen oder unteren Hälfte des Daten-Busses betrieben wird.

DTACK ("Data Transfer Acknowledge"). Dieses Signal zeigt den Abschluß des Bus-Zyklus (lesen oder schreiben) des Prozessors an.

RS1-RS5 ("Register Select"). Die RS-Anschlüsse werden normalerweise mit den unteren fünf Adreßleitungen des Prozessors verbunden und dienen der Auswahl der 24 internen Register.

RESET Wenn der RESET-Anschluß für mindestens 2 Mikrosekunden Low ist, wird der MFP in einen definierten Grundzustand versetzt. Ein RESET wird beim Einschalten des Systems, aber auch bei der Ausführung des RESET-Befehls des Prozessors erzeugt.

Eine weitere wichtige Gruppe von Signalen stellen die Interrupt-Anschlüsse dar. Zu dieser Gruppe sind die Pins IRQ, IACK, IEI und IEO zu zählen.

IRQ ("Interrupt ReQuest"). IRQ wird dann Low, wenn ein Ereignis im MFP einen Interrupt ausgelöst hat. Mit diesem Anschluß wird dem Prozessor der Interrupt signalisiert.

IACK ("Interrupt ACKnowledge"). Im Falle eines Interrupts (Low an IRQ und IEI) legt der MFP bei Low an IACK und DS einen Interrupt-Vektor auf die Datenleitungen. Jede der 16 möglichen Interrupt-Quellen kann einen eigenen Vektor liefern, so daß die Behandlung des Interrupts deutlich vereinfacht und beschleunigt wird.

IEI, IEO ("Interrupt Enable In", "Out"). Diese beiden Anschlüsse erlauben die Verkettung von mehreren MFPs. Dabei wird die Priorität der einzelnen MFPs durch die Position in der Kette festgelegt. Bei zwei MFPs wird IEI des höher priorisierten MFP auf Masse gelegt. IEO wird mit dem IEI-Eingang des zweiten 68901 verbunden. Der IEO-Ausgang des zweiten MFP bleibt unbeschaltet. Im Falle eines Interrupts wird der Prozessor das Signal IACK erzeugen. Dieses Signal gelangt an beide MFPs. Wenn der erste MFP der Kette für den Interrupt verantwortlich ist, so wird er seinen IEO-Ausgang auf High legen. Bis zum Beenden der Interrupt-Routine kann keine neue Unterbrechung durch den zweiten oder weitere MFPs erzeugt werden.

Die nächste zu betrachtende Gruppe sind die acht I/O-Anschlüsse.

IO0-7 (Input Output). Diese Anschlüsse können zum einen als ganz normale Ein-/Ausgabe-Leitungen benutzt werden. Die Datenrichtung jedes Portbits ist in einem Datenrichtungsregister separat einstellbar. Zusätzlich jedoch kann jedes Portbit als Inter-

rupt-Eingang programmiert werden. Mit einem speziellen Register kann für jedes Bit getrennt die Flanke des auslösenden Signals eingestellt werden.

Eine weitere Gruppe von Anschlüssen stellen die Timer-Pins dar. Auch die Timer Clock-Anschlüsse werden unter dieser Gruppe aufgeführt.

XTAL1, XTAL2

("Timer Clock Crystal2). An diesen Anschlüssen kann ein Quarz angeschlossen werden, der die Arbeitsfrequenz der vier Timer liefert.

TAI, TBI

("Timer Input"). Die Timer A und B können anders als die Timer C und D nicht nur als reine Zähler der an XTAL1 und 2 angelegten Timer-Frequenz benutzt werden, sondern können auch als Ereigniszähler und zur Impulsbreitenmessung eingesetzt werden. In diesen beiden Fällen muß ein externes Signal an die entsprechenden Timer-Eingänge angelegt werden.

TAO, TBO, TCO, TDO

("Timer Output"). Jeder Timer verfügt über einen Ausgang, dessen Pegel wechselt, wenn der Timer seinen Zählerstand von 01 nach 00 wechselt. Unabhängig von der eingestellten Betriebsart arbeiten die Zähler grundsätzlich als Abwärtszähler. Ihr Inhalt wird mit jedem Impuls um eins verringert.

Die vorletzte Gruppe von Signalen beinhaltet die Anschlüsse der universellen seriellen Schnittstelle. Die im MFP eingebaute vollduplexfähige Schnittstelle kann wahlweise asynchron oder synchron und mit unterschiedlichen Send- und Empfangsbaudraten arbeiten.

SI

("Serial Input"). An den SI-Eingang wird ein zu empfangener serieller Bit-Strom angelegt.

SO

("Serial Output"). Entsprechend SI ist an diesem Ausgang der Bit-Strom der zu sendenden Daten verfügbar.

RC

("Receiver Clock"). Die Übertragungsgeschwindigkeit der Empfangsdaten wird durch die Frequenz an diesem Eingang bestimmt. Als Quelle dieses Signals kann z.B. einer der eingebauten vier Timer genutzt werden.

TC

("Transmitter Clock"). Entsprechend zu RC bestimmt die Frequenz dieses Signals die Baudrate der zu sendenden Daten.

Die letzte Gruppe von Signalen ist im Atari nicht verwendet. Diese Signale werden benötigt, wenn die serielle Schnittstelle mittels DMA betrieben wird.

RR ("Receiver Ready"). Dieser Anschluß gibt Aufschluß über den Zustand des Empfänger-Datenregisters. Ist ein Zeichen vollständig empfangen, so wird dieser Anschluß auf Masse gelegt. Dadurch wird dem DMA-Controller mitgeteilt, daß ein Zeichen vom MFP abgeholt werden muß.

TR ("Transmitter Ready"). Dieser Anschluß übernimmt eine ähnliche Funktion für den Sendeteil der seriellen Schnittstelle. Mit einem Low an diesem Pin wird dem DMA-Controller mitgeteilt, daß ein neues Zeichen in den MFP übertragen werden muß.

19.4.2 Die Register des MFP

Wie bereits erwähnt, enthält der 68901 insgesamt 24 verschiedene Register. Die hohe Anzahl und die logische Anordnung der Register erleichtert die Programmierung des MFP sehr.

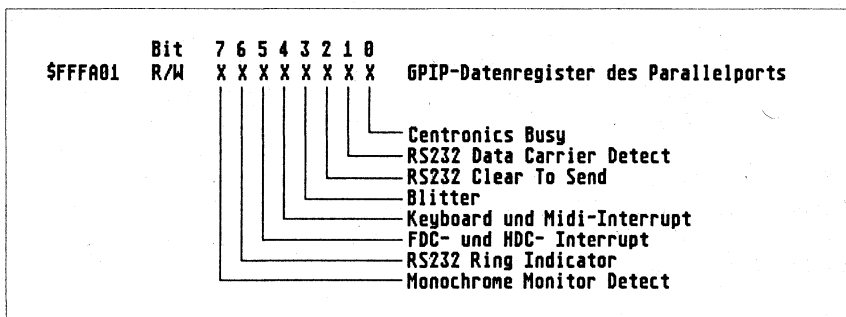


Abb. 19.9: GPIIP-Datenregister des Parallelports

Dieses Register ist das Datenregister des 8-Bit-Ports. Mittels dieser Adresse werden die Daten an die Portbits ausgegeben bzw. eingelesen.

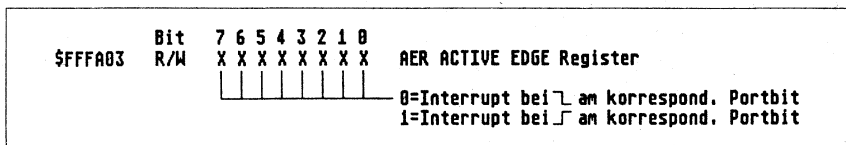


Abb. 19.10: AER Active Edge Register

Wenn Portbits als Interrupt-Eingang benutzt werden, so kann mit diesem Register festgelegt werden, ob der Interrupt bei einem Low-High- oder High-Low-Wechsel ausgelöst wird. Dabei bedeutet eine Null in diesem Register einen Interrupt bei High-Low-Wechsel, eine Eins dagegen Interrupt bei Low-High-Wechsel.

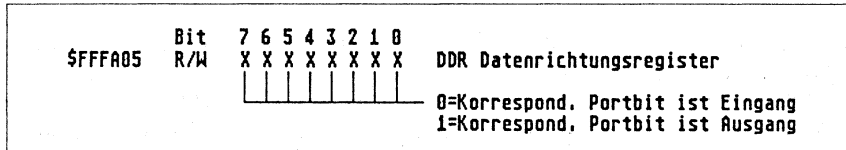


Abb. 19.11: DDR Datenrichtungsregister

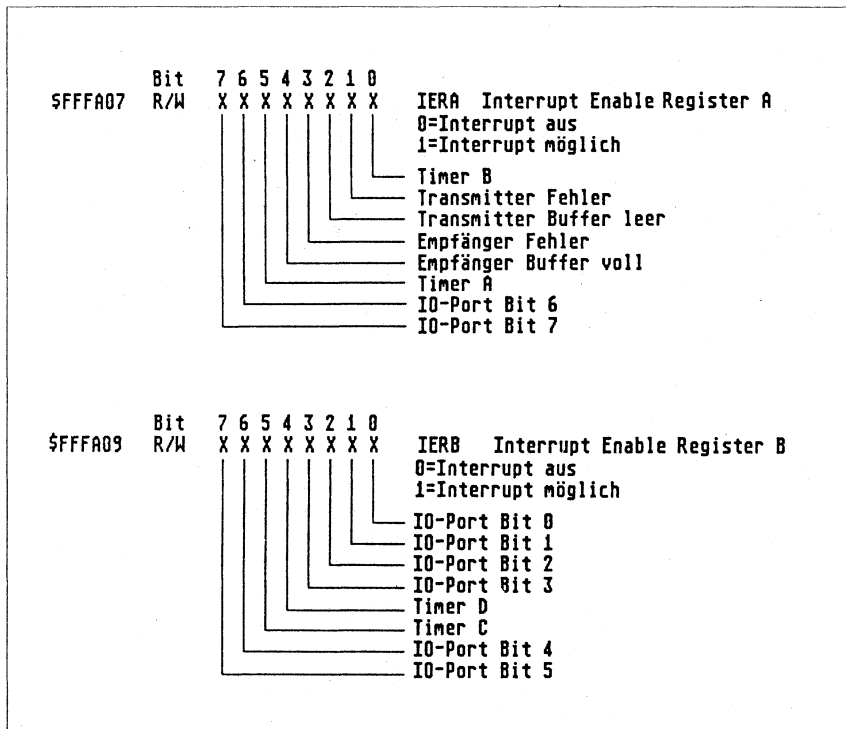


Abb. 19.12: IERB Interrupt Enable Register

Wie bereits erwähnt, kann die Datenrichtung eines jeden Portbits individuell eingestellt werden. Wird in ein DDR-Bit eine Null geschrieben, so arbeitet der entsprechende Anschluß als Eingang, eine Eins in dem DDR-Bit programmiert das Portbit als Ausgang.

Portbits werden durch AER- und DDR-Bit mit derselben Position beeinflusst.

Jede Interrupt-Quelle des MFP läßt sich separat ein- und ausschalten. Da insgesamt 16 verschiedene Interrupt-Quellen oder -Kanäle vorhanden sind, werden zwei 8-Bit-Register verwendet. Wird eine Eins in ein Bit von IERA oder IERB geschrieben, so ist der entsprechende Kanal "enabled", also eingeschaltet. Eine Null dagegen "disabled" den Kanal. Tritt an einem ausgeschalteten Kanal das den Interrupt auslösende Ereignis ein, so wird es vom MFP vollständig ignoriert.

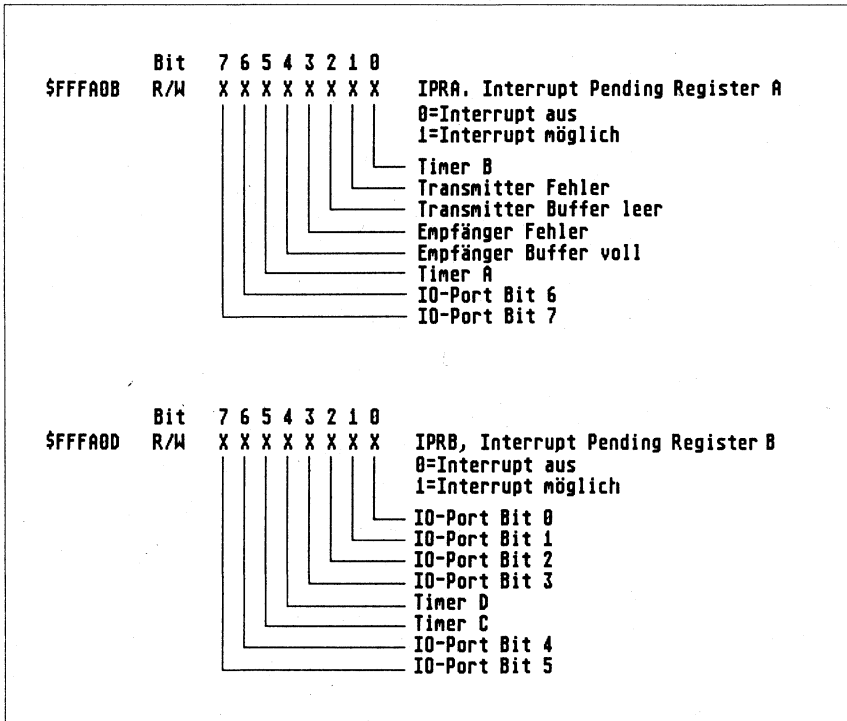


Abb. 19.13: IPRA Interrupt Pending Register A und B

Sobald ein Interrupt eines eingeschalteten Kanals auftritt, wird das dem Kanal zugeordnete Bit im Interrupt Pending Register auf Eins gesetzt. Wird im System von der Möglichkeit der Vektor-Erzeugung Gebrauch gemacht, so wird das Bit automatisch wieder gelöscht, wenn der MFP die Vektornummer auf den Daten-Bus legt. Wird von dieser Möglichkeit jedoch kein Gebrauch gemacht, so muß das IPR softwaremäßig gelöscht werden. Um ein Bit im IPR zu löschen, wird ein Byte an den MFP geschickt, in dem alle bis auf das zu löschende Bit gesetzt sind.

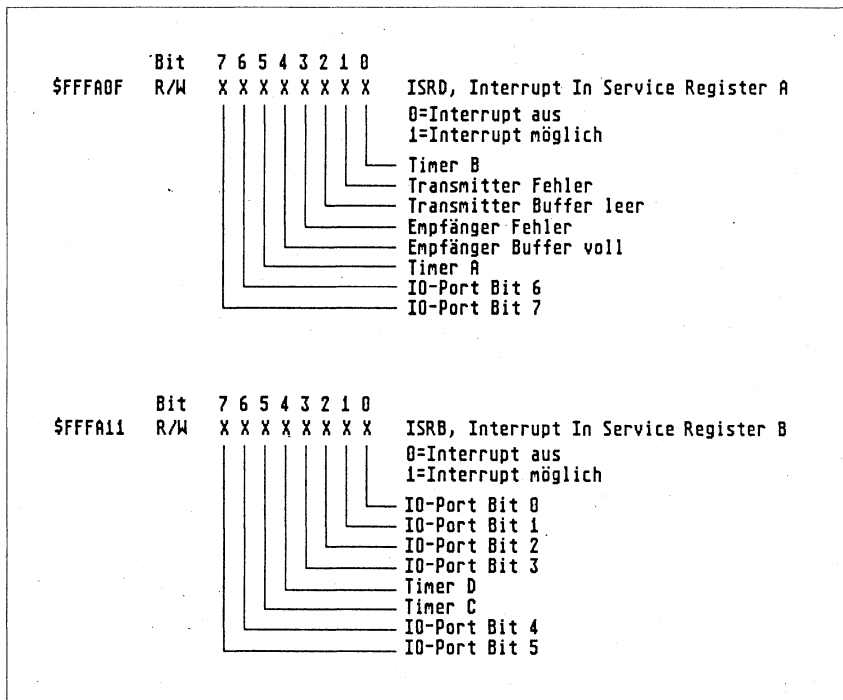


Abb. 19.14: ISRD Interrupt In Service Register A und B

Die Funktion dieser Register ist etwas komplex und hängt von Bit 3 des Registers 12 ab. Dieses als S-Bit bezeichnete Bit bestimmt, ob der 68901 im sogenannten "Software End-of-Interrupt"-Modus (SEI) oder im "Automatic End-of-Interrupt"-Modus (AEI) arbeitet. Im AEI-Modus wird das Interrupt Pending Bit (IPR) gelöscht, wenn der Prozessor während eines IACK-Zyklus die Vektornummer vom MFP holt. Auch das zugehörige Interrupt In-Service Bit wird zu diesem Zeitpunkt gelöscht. Danach können neu eintreffende Ereignisse

einen Interrupt auslösen, auch wenn die Interrupt-Routine des ersten Interrupts noch nicht beendet ist.

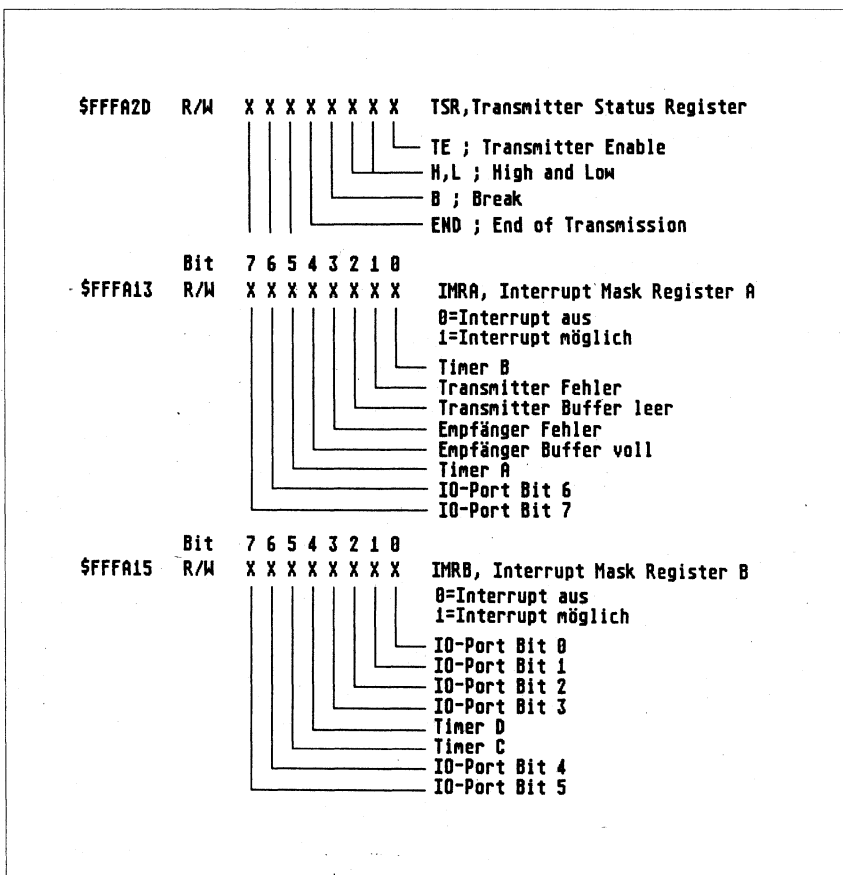


Abb. 19.15: TSR Transmitter Status Register, Interrupt Mask Register A und B

Im SEI-Modus dagegen wird das korrespondierende ISR-Bit gesetzt, wenn die Vektornummer des Interrupts vom Prozessor angefordert wird. Am Ende der Interrupt-Routine muß das Bit gelöscht werden, indem ein Byte an den MFP geschickt wird, in dem alle bis auf das zu löschende Bit gesetzt sind. Solange das Interrupt In-Service Bit gesetzt ist, sind alle Interrupts niedrigerer Priorität im MFP ausmaskiert. Da das Pending-Bit des aktiven Kanals mit Übergabe der Vektornummer gelöscht wurde, kann sogar ein gleichartiger Interrupt ein

zweites Mal auftreten. Allerdings werden dieser und eventuelle niedrigere Interrupts erst bearbeitet (lösen einen IRQ aus), wenn das gesetzte In-Service Bit gelöscht wird.

Mit Hilfe dieser Register können einzelne, in den IER eingeschaltete Interrupt-Quellen ausmaskiert werden. Das bedeutet, daß die Interrupt-Ereignisse zwar intern erkannt und im IPR signalisiert werden, die IRQ-Leitung jedoch auf High-Pegel bleibt.

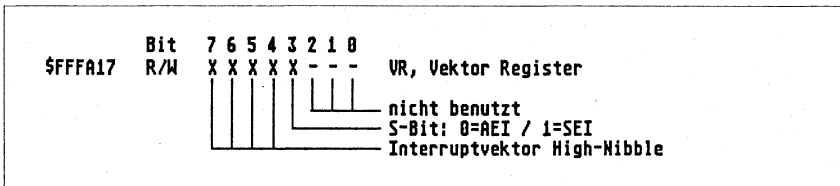


Abb. 19.16: VR Vector Register

Im Falle eines Interrupts kann der 68901 auf Anforderung des Prozessors im Interrupt Acknowledge Cycle eine der Interrupt-Quelle entsprechende Vektornummer generieren. Jeder der 16 Interrupt-Kanäle verfügt über seinen eigenen Vektor. Die 16 Kanäle werden entsprechend der Priorität in den unteren vier Bits der Vektornummer codiert, die oberen vier Bits des Vektors werden aus dem Vektor-Register kopiert. Dazu müssen diese Bits im VR entsprechend gesetzt werden.

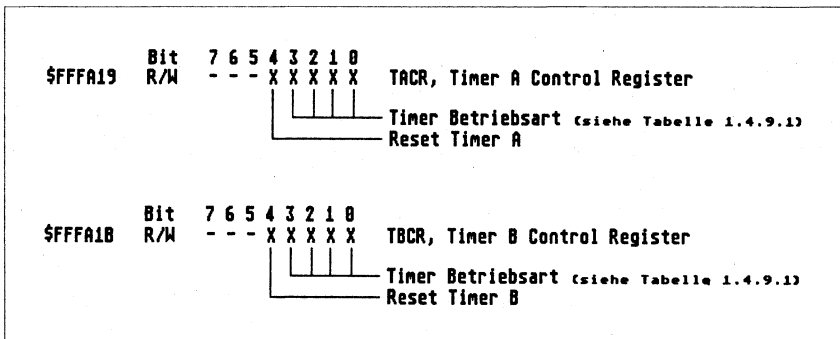


Abb. 19.17: TACR/TBCR Timer A und B Control Register

Das Bit 3 des VR ist das schon erwähnte S-Bit. Ist dieses Bit gesetzt (wie im ST), so arbeitet der MFP im "Software End-of-Interrupt"-Modus, bei gelöschtem Bit ist der Modus "Automatic End-of-Interrupt".

Bevor wir auf die Programmierung dieser Register eingehen, müssen wir einige Worte über die Möglichkeiten der Timer verlieren.

Die Timer A und B verfügen über identische Möglichkeiten. Jeder Timer besteht aus einem Datenregister, einem programmierbaren Vorteiler und einem 8-Bit-Abwärtszähler. Der Inhalt des Zählers wird bei jedem vom Vorteiler gelieferten Impuls um eins verringert. Steht der Zähler auf dem Wert 01, so wechselt beim nächsten Impuls der dem entsprechenden Timer zugeordnete Ausgang seinen Pegel. Gleichzeitig wird der Timer mit dem Wert des Timer-Datenregisters geladen. Ist bei diesem Kanal das IER-Bit gesetzt, so wird ein Interrupt ausgelöst.

Als Quelle des Zählertakts wird üblicherweise die an XTAL1, 2 anliegende Quarzfrequenz verwendet. Diese Betriebsart wird als Delay-Mode (Verzögerungsmodus) bezeichnet und steht auch bei den Timern C und D zur Verfügung.

Bei den Timern A und B jedoch können auch externe Impulse an den Timer-Eingängen TAI und TBI gezählt werden. In diesem Fall spricht man vom Event Count-Mode. Beim Zählen externer Impulse wird der Vorteiler ausgeschaltet, um jeden anliegenden Impuls zählen zu können. Die maximale Zählfrequenz an den Timer-Eingängen darf ein Viertel der Betriebsfrequenz des MFP, im ST also 1 MHz, nicht überschreiten.

Eine weitere Besonderheit dieser Betriebsart ist die Tatsache, daß die Timer-Eingänge den Interrupts der I/O-Anschlüsse I3 und I4 zugeordnet sind. Je nach Programmierung der entsprechenden Bits im Active Edge Register kann durch einen Pegelsprung an den Timer-Eingängen ein Interrupt ausgelöst werden. TAI ist dabei I4, TBI dem Anschluß I3 zugeordnet. I3 und I4 können jedoch weiterhin als I/O-Anschlüsse ohne Interrupt-Möglichkeit genutzt werden.

Timer A und B kennen noch eine dritte Betriebsart, die Pulsweitenmessung. Diese Betriebsart ähnelt dem Delay-Mode mit dem Unterschied, daß mittels der Timer-Eingänge TAI und TBI die Timer ein- und ausgeschaltet werden können. Auch bei dieser Betriebsart werden die AER-Bits von I3 und I4 benutzt, um festzulegen, ob die Timer bei einem High oder einem Low an den Timer-Eingängen arbeiten dürfen. Ist z.B. AER-Bit 4 gesetzt, so arbeitet der Zähler, wenn TAI High ist. Wechselt TAI von High nach Low, so wird ein Interrupt erzeugt, vorausgesetzt, der Interrupt des I4-Kanals ist zugelassen.

Doch kommen wir nun zur Programmierung von TACR und TBCR. In beiden Registern werden nur fünf der acht möglichen Bits genutzt. Die Bits 0 bis 3 bestimmen die Betriebsart des entsprechenden Timers.

Die Bits 4 der Timer Control Register haben eine besondere Funktion. Mit diesen Bits können die Timer-Ausgänge der zugehörigen Timer zu jedem Zeitpunkt auf Low gelegt werden. Im Anschluß an die Operation kann der Ausgang jedoch unmittelbar wieder High werden, wenn gerade zu diesem Zeitpunkt der Timer abläuft.

Die genaue Zuordnung zeigt die Abbildung 19.18:

TACR & TBCR	Timer A/B Control Register
Bit 3 2 1 0	Funktion
0 0 0 0	Timer Stop, keine Funktion ausführen
0 0 0 1	Delay Modus, Vorteiler teilt durch 4
0 0 1 0	Delay Modus, Vorteiler teilt durch 10
0 0 1 1	Delay Modus, Vorteiler teilt durch 16
0 1 0 0	Delay Modus, Vorteiler teilt durch 50
0 1 0 1	Delay Modus, Vorteiler teilt durch 64
0 1 1 0	Delay Modus, Vorteiler teilt durch 100
0 1 1 1	Delay Modus, Vorteiler teilt durch 200
1 0 0 0	Event Count Mode
1 0 0 1	Pulsweiten-Mode, Vorteiler teilt durch 4
1 0 1 0	Pulsweiten-Mode, Vorteiler teilt durch 10
1 0 1 1	Pulsweiten-Mode, Vorteiler teilt durch 16
1 1 0 0	Pulsweiten-Mode, Vorteiler teilt durch 50
1 1 0 1	Pulsweiten-Mode, Vorteiler teilt durch 64
1 1 1 0	Pulsweiten-Mode, Vorteiler teilt durch 100
1 1 1 1	Pulsweiten-Mode, Vorteiler teilt durch 200

Abb. 19.18: Die Bits 4 der Timer A/B Control/Register

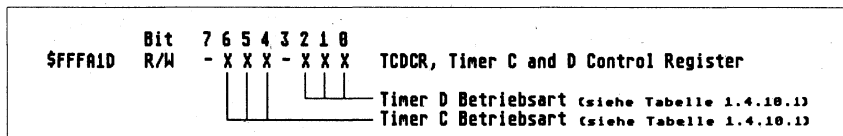


Abb. 19.19: TCDCCR Timer C und D Control Register

Die Timer C und D verfügen nur über den Delay-Mode. Deshalb reicht ein Byte zur Steuerung der beiden Timer. Die Steuerinformation wird in den unteren drei Bits der Nibbles (Vier-Bit-Hälften) programmiert. Die Bits 0 bis 2 sind dem Timer D zugeordnet, Timer C wird mit den Bits 4 bis 6 beeinflusst. Die Zuordnung zeigt Abb. 19.20:

TCD CR		Timer C and D Control Register
Bit	2 1 0	Funktion Timer D
Bit	6 5 4	Funktion Timer C
	0 0 0	Funktion Stop, keine Funktion ausführen
	0 0 1	Delay Modus, Vorteiler teilt durch 4
	0 1 0	Delay Modus, Vorteiler teilt durch 10
	0 1 1	Delay Modus, Vorteiler teilt durch 16
	1 0 0	Delay Modus, Vorteiler teilt durch 50
	1 0 1	Delay Modus, Vorteiler teilt durch 64
	1 1 0	Delay Modus, Vorteiler teilt durch 100
	1 1 1	Delay Modus, Vorteiler teilt durch 200

Abb. 19.20: Die Bits 4 des Timer C und D Control Registers

	Bit	7 6 5 4 3 2 1 0	
\$FFFA1F	R/W	X X X X X X X X	TADR, Timer A Data Register
\$FFFA21	R/W	X X X X X X X X	TBDR, Timer B Data Register
\$FFFA23	R/W	X X X X X X X X	TCDR, Timer C Data Register
\$FFFA25	R/W	X X X X X X X X	TDDR, Timer D Data Register

Abb. 19.21: Die vier Timer-Datenregister

Die vier Timer-Datenregister werden mit dem Wert programmiert, der in den Zähler geladen wird, wenn er nach Erreichen des Zählerstandes 01 einen Impuls bekommt. Von diesem Wert wird dann kontinuierlich abwärts gezählt.

	Bit	7 6 5 4 3 2 1 0	
\$FFFA27	R/W	X X X X X X X X	SCR, Synchronous Character Register

Abb. 19.22: SCR Synchronous Character Register

Bei der synchronen Datenübertragung wird in dieses Register ein Wert geschrieben, der beim Empfang den Beginn der Übertragung kennzeichnet. Ist der Synchron-Mode ausgewählt, so werden alle empfangenen Zeichen mit dem im SCR gespeicherten Wert verglichen. Erst bei Übereinstimmung werden die nachfolgenden Zeichen in den Empfänger-Buffer übertragen.

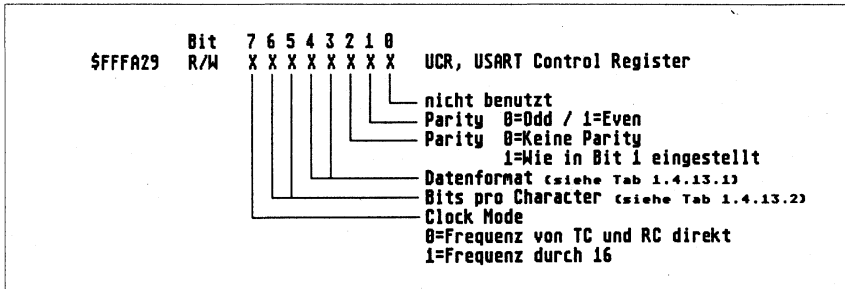


Abb. 19.23: UCR USART Control Register

USART ist die Abkürzung für Universal Synchronous/Asynchronous Receiver/Transmitter. Mit dem UCR werden bis auf die Baudrate alle Betriebsparameter der Schnittstelle eingestellt. Wie auch bei den Timern werden verschiedene Parameter in einzelnen Bits codiert. Die Belegung finden Sie in den Abbildungen 19.24 und 19.25:

UCR USART Control Register			
Bit	4	3	Funktionen
	0	0	Kein Startbit , kein Stopbit , Synchron
	0	1	1 Startbit , 1 Stopbit , Asynchron
	1	0	1 Startbit , 1.5 Stopbits , Asynchron
	1	1	1 Startbit , 2 Stopbits , Asynchron

Abb. 19.24: UCR USART Control Register-Funktionen

UCR	USART		Control Register
Bit	5	6	Wortlänge
	0	0	8 Bit
	0	1	7 Bit
	1	0	6 Bit
	1	1	5 Bit

Abb. 19.25: UCR USART Control Register-Wortlängen

Das RSR gibt Aufschluß über alle den Empfänger betreffenden Zustände. Wieder sind verschiedene Zustände in einzelnen Bits codiert.

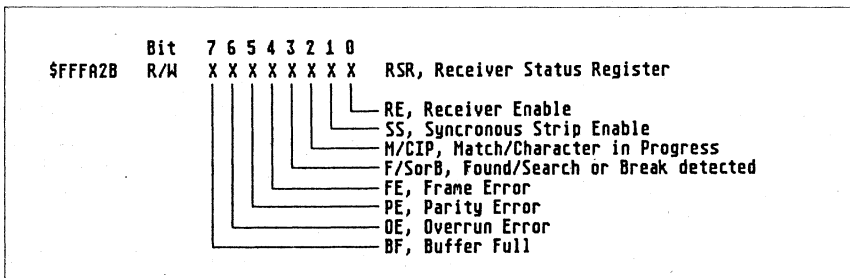


Abb. 19.26: RSR Receiver Status Register

- Bit 0: ("Receiver Enable Bit"). Wird dieses Bit gelöscht, so wird sofort der Empfangsteil ausgeschaltet. Alle Flags im RSR werden automatisch gelöscht. Bei gesetztem Bit arbeitet der Empfänger normal.
- Bit 1: ("Synchronous Strip Enable"). Mit diesem Bit kann bei synchronem Datentransfer bestimmt werden, ob ein mit dem im SCR gespeichertes identisches Zeichen in den Receiver Buffer übertragen wird oder nicht.
- Bit 2: ("Match/Character In Progress"). Im synchronen Übertragungsformat signalisiert dieses Bit, daß ein mit dem SCR-Byte identisches Zeichen empfangen wurde. In der asynchronen Betriebsart

wird dieses Bit gesetzt, sobald das Start-Bit erkannt wurde. Mit Erkennen des Stopp-Bits wird das Flag automatisch wieder gelöscht.

- Bit 3: ("Found - Search/Break Detected"). Auch Bit 3 des RSR hat je nach Format unterschiedliche Aufgaben. Im synchronen Übertragungsformat wird das Bit gesetzt, wenn ein empfangenes Zeichen mit dem im SCR gespeicherten übereinstimmt. Diese Bedingung kann über den Receiver Error-Kanal einen Interrupt auslösen. Im asynchronen Betrieb wird das Bit gesetzt, wenn ein BREAK empfangen wurde. Die Break-Bedingung wird erfüllt, wenn nach dem Start-Bit nur Nullen empfangen werden. Zur Unterscheidung des BREAK von einer "echten" Null muß der Pegel der Empfängerleitung auch zur Zeit des Stopp-Bits Low bleiben.
- Bit 4: ("Frame Error"). Ein Frame Error tritt auf, wenn ein Byte empfangen wurde, das nicht Null ist, dessen Stopp-Bit jedoch Null ist.
- Bit 5: ("Parity Error"). Der Zustand dieses Bits gibt Aufschluß darüber, ob das letzte empfangene Zeichen den Anforderungen an die Parität gerecht wurde. Ist die Paritätsprüfung beim Empfang ausgeschaltet, so wird das PE-Bit nicht beeinflußt.
- Bit 6: ("Overrun Error"). Dieses Bit wird gesetzt, wenn ein vollständiges Zeichen im Empfangsschieberegister zusammengesetzt worden ist, das zuvor empfangene Zeichen jedoch noch nicht aus dem Empfänger-Buffer ausgelesen wurde. Dieser Fehler kann einen Interrupt auslösen.
- Bit 7: ("Buffer Full"). Dieses Bit wird gesetzt, wenn ein vollständiges Zeichen in den Empfänger-Buffer aus dem Schieberegister übertragen wird. Sobald der Prozessor das Byte ausliest, wird das Bit gelöscht.

Wie das RSR Informationen über den Empfänger liefert, so ist das TSR für den Sender zuständig.

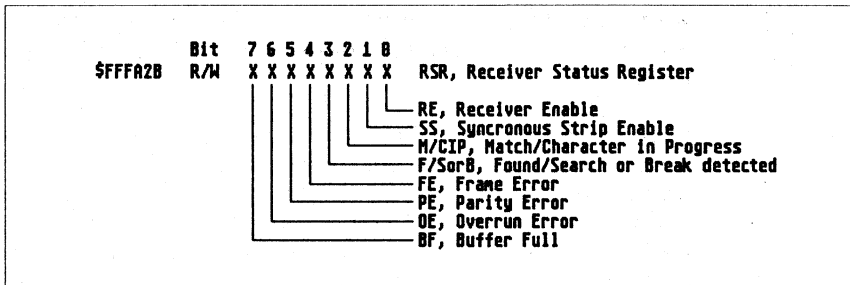


Abb. 19.27: TSR Transmitter Status Register

- Bit 0:** ("Transmitter Enable"). Wird dies Bit gelöscht, so ist der Sendeteil vollständig ausgeschaltet. Gleichzeitig wird das End-Bit gelöscht und das UE-Bit gesetzt (UE- und End-Bit siehe unten). Der Ausgang des Empfängers wird entsprechend H- und L-Bit eingestellt.
- Bit 1, Bit 2:** ("High- und Low-Bit"). Mit diesen Bits kann der Programmierer beeinflussen, welchen Zustand der Ausgang des ausgeschalteten Transmitters hat. Sind beide Bits gelöscht, so ist der Ausgang hochohmig. Ist nur das L-Bit gesetzt, so ist der Ausgang Low. Ist nur das High-Bit gesetzt, so ist der Ausgang High. Sind jedoch beide Bits gesetzt, so ist der sogenannte Loop-Back-Modus eingeschaltet. In diesem Zustand wird der Ausgang des Transmitters intern mit dem Eingang des Empfängers verbunden. Der Ausgang selbst liegt dann auf High-Pegel.
- Bit 3:** ("Break"). Bei der synchronen Datenübertragung hat das Break-Bit keine Funktion. In der asynchronen Betriebsart jedoch wird, sobald das Bit gesetzt ist, die Break-Bedingung ausgesendet.
- Bit 4:** ("End of Transmission"). Wird der Sendeteil während der laufenden Übertragung eines Zeichens ausgeschaltet, so wird das End-Bit gesetzt, sobald das Zeichen vollständig gesendet wurde. Wird jedoch zum Zeitpunkt des Abschaltens kein Zeichen übertragen, so wird das End-Bit unmittelbar gesetzt.
- Bit 5:** ("Auto Turnaround"). Ist dieses Bit gesetzt, so wird der Empfänger automatisch eingeschaltet, wenn der Sender ausgeschaltet und ein eventuelles letztes Zeichen vollständig gesendet wurde.

- Bit 6: ("Underrun Error"). Dieses Bit wird gesetzt, wenn das im Sendeschieberegister enthaltene Zeichen vollständig gesendet wurde, bevor ein neues Zeichen in den Sende-Buffer geschrieben wurde.
- Bit 7: ("Buffer Empty"). Dieses Bit wird gesetzt, wenn ein Zeichen aus dem Sende-Buffer in das Schieberegister übertragen wurde. Das Bit wird automatisch gelöscht, wenn neue Daten in den Sende-Buffer geschrieben werden.

\$FFFA2F	R/W	X X X X X X X X	UDR,USART Data Register
----------	-----	-----------------	-------------------------

Tabelle 1.4.16

Abb. 19.28: UDR USART Data Register

Über dieses Register werden die Sende- und Empfangsdaten übertragen. Beim Hineinschreiben gelangen die Daten in den Sende-Buffer, beim Lesen des Registers erhält man den Inhalt des Empfänger-Buffers.

19.5 Die ACIAs 6850

ACIA ist die Abkürzung für "Asynchronous Communications Interface Adapter". Dieses 24-polige IC beinhaltet alle zum Betrieb einer seriellen Schnittstelle benötigten Stufen. Auch Fehlererkennung und Datenformatierung werden von diesem Baustein übernommen. Ursprünglich für 6800-Computersysteme entworfen, kann dieses IC aber auch in 65xx- und 680xx-Systemen recht einfach eingesetzt werden. Im Atari ST übernehmen gleich zwei dieser Bausteine wichtige Aufgaben. Ein 6850 wird für die Kommunikation mit der Tastatur eingesetzt. Die Tastatur des ST verfügt über einen eigenen Prozessor, der die Tastatur-, Joystick- und Maus-Abfrage sowie das Führen der Uhrzeit übernimmt. Die Daten der Tastatur werden über eine serielle Schnittstelle in das eigentliche ST-System mit dem 68000 übertragen. Die zweite ACIA ist für den Betrieb der MIDI-Schnittstelle zuständig.

Bei der Tastatur-ACIA ist eine Änderung der Betriebsparameter nicht empfehlenswert. Zu schnell könnte man die Verbindung zwischen ST und Tastatur stören. Mit der MIDI-ACIA jedoch kann sicher noch die eine oder andere sinnvolle Applikation geschaffen werden. Schließlich steht ja nirgends geschrieben, daß die MIDI-Anschlüsse nicht auch für andere Zwecke gebraucht werden können. Eine Idee, die so bei Betrachtung der Schnittstelle aufkommt, ist die Vernetzung mehrerer STs zu einem Rechner-Verbund über die MIDI-Schnittstelle.

19.5.1 Die Anschlüsse des 6850

Lesern, die sich bisher nicht mit dem Prinzip der seriellen Datenübertragung beschäftigt haben, soll an dieser Stelle die Bedeutung der einzelnen Anschlüsse des 6850 erläutert werden.

Vss	Dieser Anschluß ist der Masseanschluß der Stromversorgung des ICs.
RX Data	Receive DATA. An diesen Anschluß wird der zu empfangene Bit-Strom angelegt. Dabei muß zuerst (nach dem Start-Bit) das niederwertigste Daten-Bit ankommen.
RX CLK	Receive Clock. Das Signal an diesem Anschluß bestimmt die Baudrate, also die Übertragungsgeschwindigkeit beim Empfang von Daten. Mit dem Signal dieses Anschlusses werden die hereinkommenden Daten synchronisiert. Die Frequenz von RX CLK richtet sich nach der gewünschten Übertragungsgeschwindigkeit und nach dem intern programmierten Teilverhältnis.
TX CLK	Transmitter Clock. Wie auch bei RX CLK bestimmt die Frequenz dieses Signals die Übertragungsgeschwindigkeit, hier jedoch beim Senden.
RTS	Request To Send. Dieser Ausgang des 6850 läßt sich gezielt vom Prozessor auf Low oder High setzen. Häufig wird dieser Anschluß zur Steuerung des Datentransfers benutzt. Mit einem Low an diesem Ausgang wird z.B. einem Modem signalisiert, daß der Rechner Daten senden will.
TX Data	Transmitter Data. Über diesen Anschluß werden zu übertragende Daten bitweise, also seriell, aus dem Computer "herausgeschoben".
IRQ	Interrupt ReQuest. Bei verschiedenen Zuständen der ACIA wird IRQ Low, um dem Prozessor zu signalisieren, daß seine Aufmerksamkeit benötigt wird. Mögliche Zustände sind vollständiger Empfang oder vollständiges Senden eines Zeichens.
CS 0,1,2	Chip Select. Diese drei Anschlüsse werden zur Selection der ACIA in einem Computersystem benötigt. Durch die relativ hohe Anzahl von drei CS-Signalen vereinfacht sich in kleineren Systemen oft die zur Adreßdecodierung benötigte Hardware.

RS	Register Select. Über diesen Anschluß wird in Verbindung mit dem Signal R/W das gewünschte interne Register angesprochen. Die Auswahl und Programmierung der Register wird weiter unten beschrieben.
Vcc	Dieser Anschluß ist der für alle ICs obligatorische Stromversorgungsanschluß. Die Betriebsspannung beträgt die üblichen 5 Volt Gleichspannung.
R/W	"Read/Write". Das Signal am R/W-Eingang beschreibt die vom Prozessor gewünschte Datenrichtung bei der Kommunikation mit der ACIA. Bei einem High-Signal liest der Prozessor Daten, bei Low-Pegel schreibt er Daten in den 6850.
E	"Enable". Das E-Signal bestimmt den Zeitpunkt des Lesens oder Schreibens der Daten. Da alle Schreib-/Lesevorgänge mit diesem Signal synchron ablaufen müssen, spricht man bei 68xxx vom synchronen Bus-Anschluß.
D0 - D7	Die Datenanschlüsse des 6850 werden mit dem Daten-Bus des Prozessors verbunden. Bis auf die Zeiten des Zugriffs auf die ACIA sind diese bidirektionalen Anschlüsse hochohmig.
DCD	"Data Carrier Detect". DCD stellt ein Signal zur Modemsteuerung dar. Über diesen Anschluß teilt das Modem dem Rechner mit, daß Daten empfangen werden sollen. Liegt der Eingang DCD auf High, so ist ein Empfang von seriellen Daten unmöglich.
CTS	"Clear To Send". Das Signal CTS antwortet dem Rechner auf das Signal RTS. Das Senden von Daten ist nur möglich, wenn dieser Anschluß auf Low-Pegel liegt.

19.5.2 Die Register des 6850

Insgesamt enthält der 6850 vier verschiedene Register. Zwei Register können nur gelesen, die anderen beiden nur beschrieben werden. Die Unterscheidung der Register erfolgt mit den beiden Signalen R/W und RS nach Abbildung 19.29.

R/W	RS	Register
0	0	Control Register
0	1	Sende Register
1	0	Status Register
1	1	Empfangs Register

Abb. 19.29: Die Register des 6850

Die Sende- und Empfangsregister (auch RX- und TX-Buffer genannt) sind für den Transfer der Daten zuständig. Wenn der Empfang von Daten möglich ist, so gelangen die seriell ankommenden Bits in ein Schieberegister. Sobald die spezifizierte Anzahl Bits angekommen ist, wird der Inhalt des Schieberegisters in den TX-Buffer übertragen. Ähnlich läuft der Vorgang beim Senden ab. Hier wird das zu übertragende Byte zunächst in das Senderegister geschrieben, aus dem es automatisch in das Sende-Schieberegister übertragen wird.

CR 1	CR 0	Funktionen
0	0	RXCLK/TXCLK ohne Teilung
0	1	RXCLK/TXCLK durch 16 (Midi)
1	0	RXCLK/TXCLK durch 64 (Tastatur)
1	1	Master Reset

Abb. 19.30: Senderegister

Das Control-Register mit seinen acht Bits steuert die internen Vorgänge. Um die Steuerung der diversen Funktionen mit einem Byte zu lösen, wurde einzelnen Bits eine bestimmte Bedeutung zugeordnet.

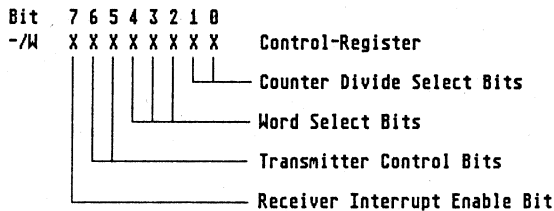


Abb. 19.31: Das Control-Register

CR 0,1

Diese Bits bestimmen, durch welchen Faktor die Transmitter- und Receiver-Clock-Frequenz geteilt wird. Zusätzlich ist mit diesen Bits eine Master Reset-Funktion verbunden. Da der 6850 über keinen separaten Reset-Anschluß verfügt, muß vor dem ersten Gebrauch der ACIA der Master Reset programmtechnisch durchgeführt werden.

CR 4	CR 3	CR 2	Funktion
0	0	0	7 Datenbits , 2 Stopbits , Parity even
0	0	1	7 Datenbits , 2 Stopbits , Parity odd
0	1	0	7 Datenbits , 1 Stopbit , Parity even
0	1	1	7 Datenbits , 1 Stopbit , Parity odd
1	0	0	8 Datenbits , 2 Stopbits , Keine Parity
1	0	1	8 Datenbits , 1 Stopbit , Keine Parity
1	1	0	8 Datenbits , 1 Stopbit , Parity even
1	1	1	8 Datenbits , 1 Stopbit , Parity odd

Abb. 19.32: Die Word Select-Bits

CR 2,3,4

Diese sogenannten "Word Select-Bits" geben an, ob sieben oder acht Daten-Bits und ein oder zwei Stopp-Bits übertragen wer-

den. Auch die Art der Paritätsprüfung wird in diesen Bits angegeben.

CR 6	CR 5	Funktion
0	0	RTS Low , Transmitter IRQ disabled
0	1	RTS Low , Transmitter IRQ enabled
1	0	RTS High , Transmitter IRQ disabled
1	1	RTS Low , Transmitter IRQ disabled ,Break senden

Abb. 19.33: Die Transmitter Control-Bits

CR 6,5

Diese Transmitter Control-Bits bestimmen den Pegel am RTS-Ausgang und erlauben oder verbieten einen Interrupt durch die ACIA, wenn das Sende-Register geleert ist. Zusätzlich kann das Senden eines sogenannten BREAK-Signals über den seriellen Ausgang veranlaßt werden. Das BREAK-Signal besteht aus einer langen Folge von Null-Bits.

CR 7	Funktion
0	Interrupts disabled (ausgeschaltet)
1	Interrupts enabled (eingeschaltet)

Abb. 19.34: Tabellarische Darstellung des Break-Signals

CR 7

Das Receiver Interrupt Enable-Bit bestimmt, ob durch den Empfänger Interrupts ausgelöst werden können. Ursache für einen Interrupt kann ein Wechsel des Pegels an der DCD-Leitung von Low nach High sein. Auch wenn der Empfänger-Daten-Buffer voll ist, kann ein IRQ ausgelöst werden. Außerdem kann ein Interrupt im Falle eines OVERRUN ausgelöst werden. Dieser Zustand tritt ein, wenn ein empfangenes Zeichen nicht rechtzei-

tig vom Prozessor gelesen wurde und ein vollständiges neues Zeichen von der ACIA in das interne Receiver-Datenregister übertragen wurde.

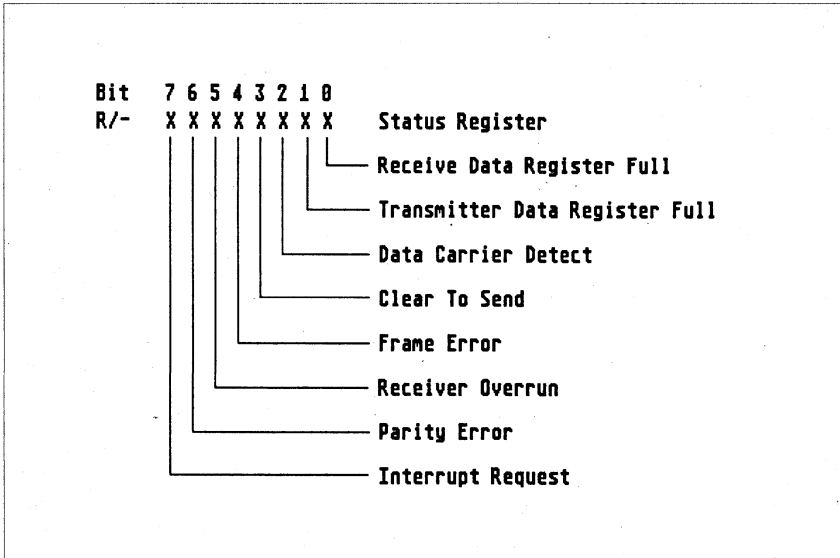


Abb. 19.35: Die acht Bit des Status Registers

Das Statusregister der ACIA gibt Aufschluß über bestimmte Zustände des Bausteins. Auch bei diesem Byte sind die Informationen in einzelnen Bits codiert.

- SR 0 Bei einem High dieses Bits ist das RX-Datenregister voll. Um einem OVERRUN zuvorzukommen, muß das Byte gelesen werden, bevor ein neues Zeichen vollständig empfangen wird.
- SR 1 Dieses Bit reflektiert den Zustand des TX-Daten-Buffers. Ist das Register leer, so ist das Bit gesetzt.
- SR 2 Mit einem Low-High-Wechsel am Pin DCD wird SR2 gesetzt. Ist der Receiver Interrupt zugelassen, so wird ein IRQ ausgelöst. Gelöscht wird das Bit, wenn zuerst das Statusregister, danach das Receiver-Register gelesen wird. Damit wird auch der IRQ aufgehoben. Bleibt nach dem Lesen der beiden Register SR2

weiterhin High, so ist dies ein Zeichen dafür, daß auch das Signal am DCD-Pin noch High ist. Sobald DCD auf Low geht, wird dann auch SR2 Low werden.

- SR 3 Dieser Anschluß zeigt direkt den Zustand des Pins CTS an. Dieses Signal kann nicht durch Master Reset oder Programmierung der ACIA beeinflusst werden.
- SR 4 SR4 zeigt sogenannte "Frame-Fehler". Frame- oder Rahmenfehler entstehen, wenn die Empfängerschaltung kein Stopp-Bit erkennen kann. Es wird mit jedem neuen Zeichen entsprechend gesetzt.
- SR 5 Dieses Bit zeigt die schon erwähnte "Overrun-Bedingung" an. Zurückgesetzt wird SR5 mit dem Lesen des RX-Buffers.
- SR 6 An Hand dieses Bits kann bei eingeschalteter Paritätsprüfung erkannt werden, ob die Parität des empfangenen Zeichens stimmt. Im Fehlerfall ist das Bit gesetzt.
- SR 7 Bit 7 des Statusregisters signalisiert den Zustand des IRQ-Pins. Dies Bit ermöglicht das Zusammenschalten von mehreren IRQ-Anschlüssen an einen Interrupt-Eingang. Im Falle eines Interrupts kann programmtechnisch ermittelt werden, welches IC den Interrupt ausgelöst hat.

Die ACIAs im Atari ST

Viele Möglichkeiten der ACIAs werden im ST nicht benötigt. Insbesondere sind die Anschlüsse CTS, DCD und RTS nicht belegt. Für den vorgesehenen Verwendungszweck ist das aber auch nicht erforderlich.

Die Tastatur-ACIA belegt die Adressen "\$FFFC00" und "\$FFFC02". Die eingestellten Parameter sind 8 Daten-Bits, 1 Stopp-Bit, keine Parität, 7812.5 Baud (500kHz/64).

Bis auf die Baudrate sind die Parameter der MIDI-Schnittstelle identisch. Sie beträgt 31250 Baud (500kHz/16). Dieser Wert ist durch die MIDI-Spezifikationen festgelegt.

19.6 Der Sound Generator YM-2149

Der YM-2149 von YAMAHA, ein mit dem recht verbreiteten General Instruments IC AY-3-8910 kompatibler Sound Generator (PSG) ist ein Soundchip der Spitzenklasse. Er wurde entwickelt für Telespiele, um diese mit besonders realistischem Sound zu versehen, nachdem die ersten Telespiele nur recht monotone Geräusche produzieren konnten.

Um möglichst universell einsetzbar zu sein, wurde der PSG mit einer Vielzahl von Möglichkeiten zur Klangbeeinflussung versehen. Zusätzlich sagte man sich bei der Entwicklung dieses ICs wohl, daß in fast allen Einsatzgebieten auch irgendwelche Tasten, Joysticks oder Schalter abgefragt werden müssen. So gab man diesem PSG auch noch zwei bidirektionale 8-Bit-Parallelports mit.

Die Leistungsdaten dieses ICs im Überblick lauten:

- drei unabhängig programmierbare Ton-Oszillatoren,
- ein programmierbarer Rausch-Generator,
- vollständig software-gesteuerte Analogausgänge,
- programmierbarer Mischer für Ton und Rauschen,
- 15 logarithmisch gestufte Lautstärkestufen,
- programmierbare Hüllkurven,
- zwei bidirektionale 8-Bit-Datenports,
- TTL-Kompatibel,
- einfache 5 Volt Betriebsspannung.

Insgesamt verfügt der YM-2149 über 16 Register. Über diese Register können alle Klangmöglichkeiten des Chips programmiert werden.

Die Schaltung des PSG kann in einzelne Funktionsblöcke unterteilt werden. Da ist zunächst der Block der Tongeneratoren. Die Tongeneratoren werden mit einem Taktsignal versorgt, das aus dem durch 16 geteilten Clock-Signal gewonnen wird. Die Tongeneratoren sind zuständig für die grundsätzliche Erzeugung der drei rechteckförmigen Tonfrequenzen.

Der Rauschgenerator erzeugt ein frequenzmoduliertes Rechtecksignal, dessen Pulsbreite von einem Pseudo-Rauschgenerator beeinflußt wird. Die Mischer koppeln die Ausgangssignale der drei Generatoren mit dem Rauschsignal. Die Kopplung kann für jeden Kanal getrennt programmiert werden. Der Funktionsblock der Amplitudenkontrolle bietet dem Anwender zwei Möglichkeiten.

Zum einen kann die Ausgangsamplitude (die Lautstärke) der drei Kanäle über die Programmierung des entsprechenden Lautstärke-Registers beeinflusst werden.

Alternativ besteht die Möglichkeit, sie vom PSG variabel zu beeinflussen. Dann wird der Ausgang des Hüllkurven-Registers genutzt, um die Lautstärke zu beeinflussen. Da die Hüllkurve (auch als Envelope, Umschlag oder Umhüllung bezeichnet) mit vier getrennten Parametern programmierbar ist, bestehen vielfältige Möglichkeiten der Tonbeeinflussung. Der Funktionsblock der D/A-Wandler ist zuständig für die Erzeugung der Lautstärke der Ausgangssignale. Da die Lautstärke- und Envelope-Informationen als digitale Werte vorliegen, werden sie im D/A-Wandler umgesetzt.

Der letzte Funktionsblock hat mit der Tonerzeugung nichts zu tun. In diesem Block sind die zwei I/O-Ports untergebracht.

19.6.1 Die Anschlüsse des Sound Chip

Da die Bezeichnungen der Anschlüsse des PSG nicht unbedingt selbsterklärend sind, hier die detaillierte Beschreibung der Funktion der Pins:

Vss Dies ist der Masse-Anschluß des PSG.

NC Dieser Anschluß wird nicht belegt.

ANALOG B Dies ist der Ausgang des Kanals B. Hier können die von Kanal B erzeugten Töne abgenommen werden. Die maximale Ausgangsspannung ist 1 Vss.

ANALOG A Funktion wie Pin 3 für den Kanal A.

IOB7 - 0 Die IOB-Anschlüsse stellen einen der zwei 8-Bit-Ports des PSG dar. Je nach Programmierung arbeiten die Anschlüsse als Ausgang oder Eingang. Dabei kann nur die Betriebsart für den ganzen Port eingestellt werden. Ein gemischter Betrieb (gleichzeitig Bits als Eingang, andere Bits als Ausgang) ist nicht möglich, wohl aber können die Datenrichtungen der beiden Ports getrennt voneinander bestimmt werden.

IOA7 - 0 Funktion wie IOB, nur für den Port A.

CLOCK Von der Frequenz dieses Signals werden alle Tonfrequenzen durch Teilung abgeleitet. Die Frequenz dieses Signals sollte zwischen 1 und 2 MHz liegen.

- RESET** Durch einen Low-Pegel an diesem Anschluß werden alle internen Register zurückgesetzt. Ohne Reset stehen nach dem Einschalten zufällige Werte in den verschiedenen Registern, die Folge wäre ein (wahrscheinlich) sehr unmusikalisches "Geräusch".
- A9** Dieser Anschluß kann als eine Art Chip Select-Signal verstanden werden. Um Register des PSG ansprechen zu können, muß dieses Signal Low sein.
- A8** Funktion wie A9, nur ist dieser Anschluß High-aktiv.
- TEST2** Test2 wird nur von der Herstellerfirma verwendet und muß im Betrieb unbeschaltet bleiben.

BDIR und BC1,2

Der Anschluß BDIR-Signal (Bus DIRection) und die Anschlüsse BC1 und BC2 (Bus Control) steuern den Registerzugriff auf den PSG. Auf den ersten Blick mag die in der Abbildung 19.36 gezeigte Zuordnung etwas seltsam erscheinen. Da dieses IC jedoch ursprünglich als Baustein zum Prozessor 1610, einem speziellen 16-Bit-Prozessor von General Instruments, entwickelt wurde, hat man beim Entwurf auf die speziellen Eigenschaften und Steueranschlüsse dieses Prozessors Rücksicht genommen.

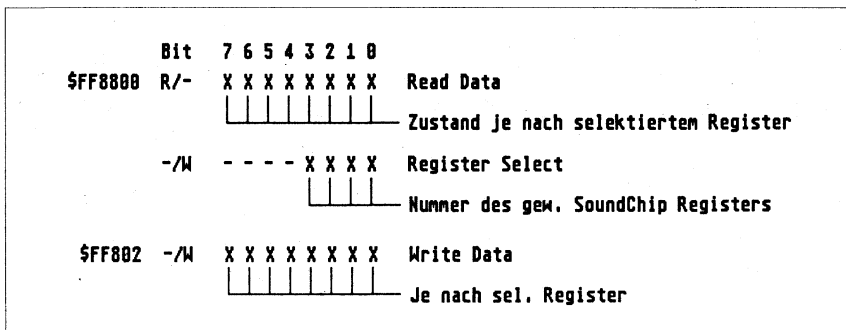


Abb. 19.36: Read Data und Write Data

Innerhalb der Tabelle gibt es nur vier tatsächliche Funktionen, die jedoch teilweise mehrfach vorhanden sind. Die Abbildung 19.37 zeigt darum noch einmal die wirklich benötigten Kombinationen. Dabei ist der Anschluß BC2 auf +5 Volt gelegt. Die verbleibende Tabelle wird also nur noch von den Signalen BDIR und BC1 bestimmt.

BDIR	BC1	Funktion
0	0	Inaktiv
0	1	Read, PSG-Register lesen
1	0	Write, PSG-Register schreiben
1	1	Latch, Registernummer schreiben

Abb. 19.37: Die Funktionen von BDIR und BC1

DA0 - 7 Diese Anschlüsse des Sound Chips werden mit dem Daten-Bus des Prozessors verbunden. Die Bezeichnung DA deutet an, daß sowohl Daten als auch (Register-)Adressen über diese Anschlüsse geführt werden.

ANALOG C Funktion wie Pin 3 für den Kanal C.

TEST1 Wie auch Test2 wird dieser Anschluß nur von der Hersteller-firma verwendet und muß im Betrieb unbeschaltet bleiben.

Vcc An diesem Anschluß wird die Betriebsspannung von +5 Volt angelegt.

19.6.2 Die Funktion der einzelnen Register des 2149

Da jetzt geklärt ist, wie über die Anschlüsse BDIR und BC1 die Register grundsätzlich angesprochen werden können, wollen wir sehen, was für Funktionen die Register ausführen. Dabei ist die Registernummer, die in der folgenden Aufstellung verwendet wird, gleich mit der Nummer, die im Adreßregister eingetragen werden muß, um das gewünschte Register anzusprechen.

Interessant ist noch die Tatsache, daß das Adreßregister seinen Inhalt bis zur nächsten Programmierung behält. Man kann also ohne Probleme mehrmals nacheinander auf ein Datenregister zugreifen, ohne jedes Mal das Adreßregister neu laden zu müssen.

BDIR	BC2	BC1	Funktion
0	0	0	Inactive
0	0	1	Latch Address
0	1	0	Inactive
0	1	1	Read from PSG
1	0	0	Latch Address
1	0	1	Inactive
1	1	0	Write to PSG
1	1	1	Latch Address

Abb. 19.38: BDIR, BC1 und BC2

Doch jetzt zur Registerbeschreibung.

	Bit	7	6	5	4	3	2	1	0	
Register 0	R/W	X	X	X	X	X	X	X	X	Periodendauer Kanal A Low-Byte
Register 1	R/W	-	-	-	-	X	X	X	X	Periodendauer Kanal A High-Byte

Abb. 19.39: Die Registerbeschreibung

Die Register 0 und 1 bestimmen die Periodendauer und damit die Frequenz des Tonsignals an ANALOG A. Allerdings sind nicht alle 16 Bit benutzt. Verwendung finden alle 8 Bit des Registers 0 und die vier unteren Bit des Registers 1. Dabei kann die Frequenz mit dem Register 0 in feinen, mit dem Register 1 in groben Stufen beeinflusst werden. Je kleiner der 12-Bit-Wert dieser Register wird, desto höher wird der Ton. Die Register 2,3 und 4,5 haben die gleiche Funktion wie die Register 0,1, aber für Kanal B bzw. C.

	Bit	7	6	5	4	3	2	1	0	
Register 6	R/W	-	-	-	X	X	X	X	X	Rauschgenerator

Abb. 19.40: Der Rauschgenerator

Das Register 6 beeinflusst mit seinen unteren fünf Bit den Rausch-Generator. Auch hier gilt: Je kleiner der Wert im Register, desto höher die Rauschfrequenz. Im Multi-Funktionsregister 7 kontrollieren einzelne Bits unterschiedliche Aufgaben. Die Verwendung der einzelnen Bits zeigt die Abbildung 19.41.

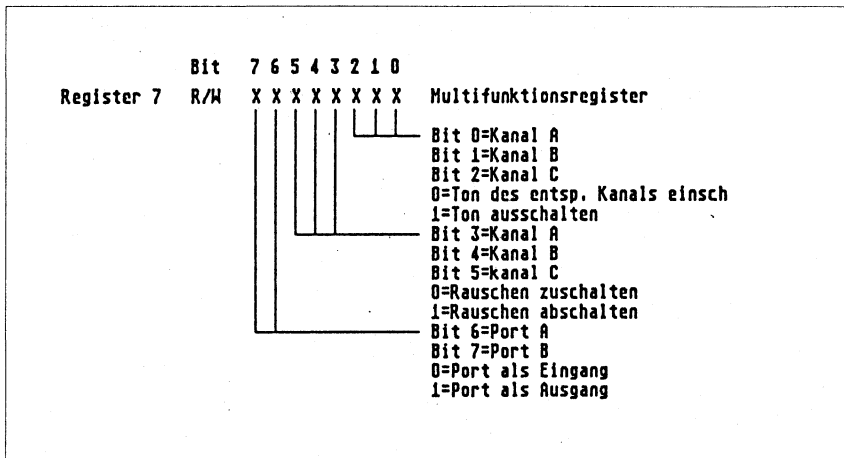


Abb. 19.41: Das Multifunktionsregister

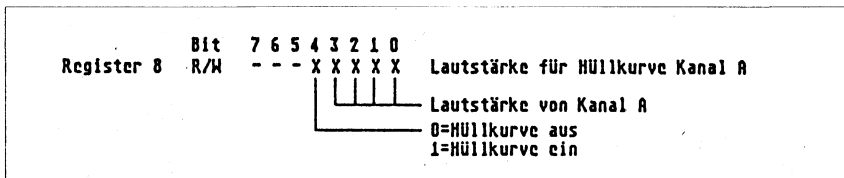


Abb. 19.42

Das Register 8 bestimmt die Lautstärke des Signals an Kanal A. Zur Lautstärke-Einstellung werden die vier unteren Bits verwendet.

Das Bit 4 hat eine besondere Bedeutung. Ist es gesetzt, dann wird die Lautstärke durch das Hüllkurven-Register bestimmt, der Inhalt der Bits 0 bis 3 wird dann ignoriert.

Die Register 9 und 10 wirken wie Register 8 für Kanal B bzw. C.

	Bit	7	6	5	4	3	2	1	0	
Register 11	R/W	X	X	X	X	X	X	X	X	Hüllkurve Low Byte
Register 12	R/W	X	X	X	X	X	X	X	X	Hüllkurve High Byte

Abb. 19.43: Hüllkurvenregister

Alle 16 Bit der beiden Register 11 und 12 beeinflussen die Periodendauer der Hüllkurve. Der Inhalt des Registers 11 wird als Low-Byte betrachtet, d.h. er beeinflusst die Periodendauer in feinen Schritten, das Register 12 ist das High-Byte des Hüllkurven-Generators.

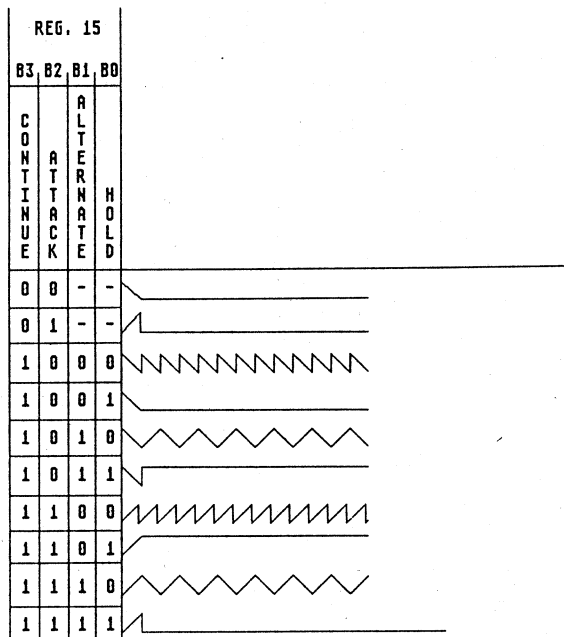


Abb. 19.44: Alle erzeugbaren Hüllkurven

Die Bits 0 bis 3 des Registers 13 bestimmen die Kurvenform des Hüllkurven-Generators. Die Zuordnung in Worten verständlich wiedergeben zu wollen, ist fast unmöglich. Die erzeugbaren Hüllkurven sind darum in der Grafik 19.44 gezeigt.

Über das Register 14 laufen die Zugriffe auf den Parallel-Port A. Dieses Register muß auf Grund seiner festen Funktion im ST als Ausgang programmiert sein (siehe Register 7).

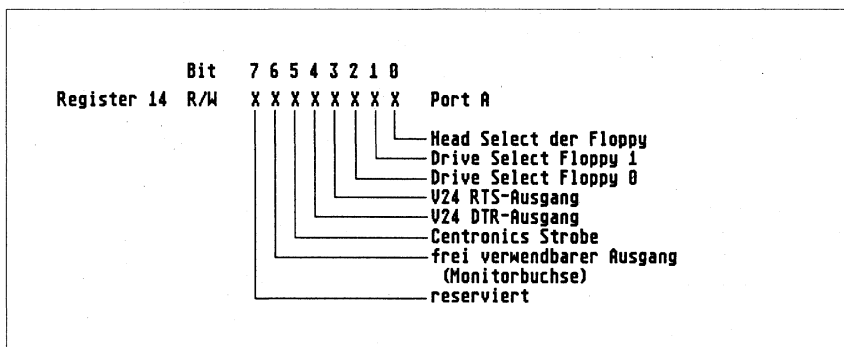


Abb. 19.45: Register 14 (Port A)

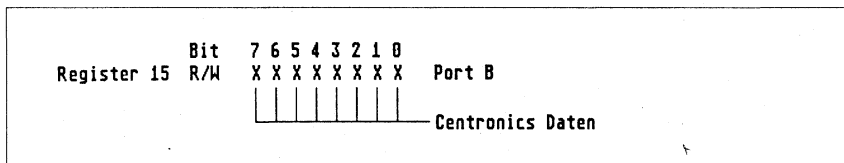


Abb. 19.46: Register 15 (Port B)

Über das Register 15 laufen die Zugriffe auf den Port B. Dieser Port wird zwar üblicherweise als Ausgang des Centronics-Ports benutzt, kann aber bei Bedarf auch als Eingang programmiert werden (siehe Register 7).

19.7 Der Atari-Blitter

Wer die Entwicklungsgeschichte des ST von den Anfängen an mitverfolgt hat, der wird sicher beim Wort BLITTER zusammensucken und sich insgeheim fragen, ob die Geschwindigkeit dieses Chips in Zusammenhang mit seiner

Entwicklungszeit steht. Wenn dem so sein sollte, dann wäre der Blitter kaum zu etwas nützlich. Tatsächlich zogen sich die Entwicklungszeiten trotz mehrfacher Ankündigung über mehr als zwei Jahre hin. Jeder ST-Besitzer wartete auf ihn, aber kaum einer hatte ihn tatsächlich gesehen.

Wie dem auch sei, die Wartezeit hat sich gelohnt. Der Atari-Blitter beschleunigt alle ihm zugedachten Aufgaben enorm. Selbst die optimiertesten in Assembler geschriebenen Grafik-Routinen können mit dem Blitter nicht mehr mithalten.

Der wesentliche Geschwindigkeitsgewinn wird vom Blitter dadurch erzeugt, daß er die Daten im DMA-Betrieb verarbeitet. Konkret bedeutet das, daß der Blitter nach erfolgter Programmierung ohne weiteres Zutun des Prozessors einen Speicherbereich mit Daten füllt oder einen Speicherbereich an eine andere Adresse verschiebt. Dabei sind so viel zusätzliche Möglichkeiten in den Blitter eingebaut, weil man für die softwaremäßige Realisation schon mehrere KByte Programme benötigt.

Die grundsätzlichen Grafik-Routinen des ST sind dem Programmierer über die sogenannten Line-A-Funktionen zugänglich. Der Blitter kann nun mehrere dieser teilweise sehr zeitaufwendigen Routinen übernehmen und mit wesentlich höherer Geschwindigkeit ausführen als dies dem 68000 möglich wäre. Da ist zunächst die Funktion "BitBlit" zu erwähnen, die einen bestimmten pixelorientierten Speicherbereich verschiebt. Aber auch das Füllen von beliebigen Speicherbereichen kann der Blitter übernehmen. Die detaillierten Angaben zu den Möglichkeiten des Blitters sollen etwas später folgen. Betrachten wir zunächst die vorhandene Hardware.

Da der Blitter ein DMA-Device ist, muß er die Möglichkeit besitzen, den Prozessor in einen scheintoten Zustand zu versetzen. Dazu werden beim 68000 die Anschlüsse BR ("Bus Request"), BG ("Bus Grant") und BGACK ("Bus Grant Acknowledge") benötigt. Über die BG-Leitung wird dem Prozessor mitgeteilt, daß jemand anderes den Adreß- und Daten-Bus für sich benötigt. Hat der Prozessor den "Bus Request" anerkannt, so meldet dieser an dem Anschluß BG dem anfordernden Device, daß er den Bus jetzt für das Bus-anfordernde DMA-Device zur Verfügung stellt. Jetzt muß noch eine kurze Wartezeit eingelegt werden, bis der 68000 tatsächlich seine Aktivitäten auf den verschiedenen Pins beendet hat (siehe Kapitel 19.2). Sobald die DMA-Einheit festgestellt hat, daß der Prozessor nicht mehr aktiv ist, kann sie mit Hilfe des Signals BGACK den Bus übernehmen. Nach beendetem Datentransfer wird die BGACK-Leitung vom DMA-Device wieder freigegeben, und der Prozessor erhält wieder die Kontrolle über den Bus.

Betrachten wir jetzt jedoch den Zeitpunkt, an dem der Blitter den Bus zugeteilt bekommt. An diesem Punkt kann er über den gesamten Adreßbereich des 68000 (das sind immerhin 16 Megabyte) verfügen. Um nun die Daten im Speicher in der zuvor programmierten Weise manipulieren zu können, muß er die nicht mehr vom Prozessor kommenden Steuersignale selbst erzeugen. Das ist zuallererst einmal die Read/Write-Leitung, über die bestimmt wird, ob Daten aus dem Speicher gelesen oder in ihn hinein geschrieben werden. Weitere wesentliche Signale für den Zugriff auf den Speicher sind AS ("Address Strobe"), LDS ("Lower Data Strobe") und UDS ("Upper Data Strobe").

Das Signal DTACK ("Data Transfer Acknowledge") erzeugt der Blitter nur, wenn auf seine Register zugegriffen wird, um damit dem Prozessor die Übernahme der Daten anzuzeigen. Beim eigentlichen DMA-Transfer wird es nicht gebraucht, da das Timing der RAM-Bausteine dem Blitter bekannt ist bzw. über das CLK-Signal festliegt.

Wie der andere vorhandene DMA-Kanal (Floppy und DMA-Port) und die ACIAs ist auch der Blitter interruptfähig. Das bedeutet konkret, daß er in der Lage ist, das Ende eines Datentransfers durch einen Interrupt anzuzeigen. Dazu wird das bis dahin noch freie Bit 3 des MFP Interrupt-Eingangs (GPIP) benutzt. Diese Möglichkeit wird zwar vom Betriebssystem des ST (wie auch beim Floppy-Controller und beim DMA-Port) nicht ausgenutzt, interrupt-orientierte Betriebssysteme wie RTOS, OS9 oder UNIX sollten jedoch vorzüglich mit diesem Blitter zusammenarbeiten können.

Die letzte noch nicht genannte Gruppe der Blitter-Anschlüsse stellen die Versorgungsanschlüsse dar. Neben der obligatorischen Betriebsspannung von 5 Volt und dem Masse-Anschluß benötigt der Blitter auch noch ein Taktsignal. Als Takt findet der Prozessortakt von 8 MHz Verwendung.

19.7.1 Die Register des Blitters

Der Blitter des Atari ST ist die hardwaremäßige Implementation des "Bitblt"-Algorithmus, wie er von den "Line-A"-Funktionen zur Verfügung gestellt wird. Zum besseren Verständnis der folgenden Erläuterungen sollten Sie evtl. die Beschreibungen der "Bitblt"-Funktion lesen.

In Abbildung 19.47 sehen Sie ein Blockdiagramm der Blitter-Funktionen. Grundsätzlich kann der Blitter einen Quellbereich, der noch mit einem eventuellen Raster verknüpft werden kann, mit einem Zielbereich über 16 verschiedene logische Operationen verknüpfen und das Ergebnis wieder im Zielbereich ablegen. Dabei können Quell- und Zielbereich an beliebigen Stellen im

RAM stehen. Im Gegensatz zum Prozessor, der nur byte- oder wortorientiert arbeiten kann, ist hier eine bitweise Orientierung möglich. Damit ist der Blitter zur Verarbeitung von pixelorientierter Grafik (bit mapped graphics) geeignet.

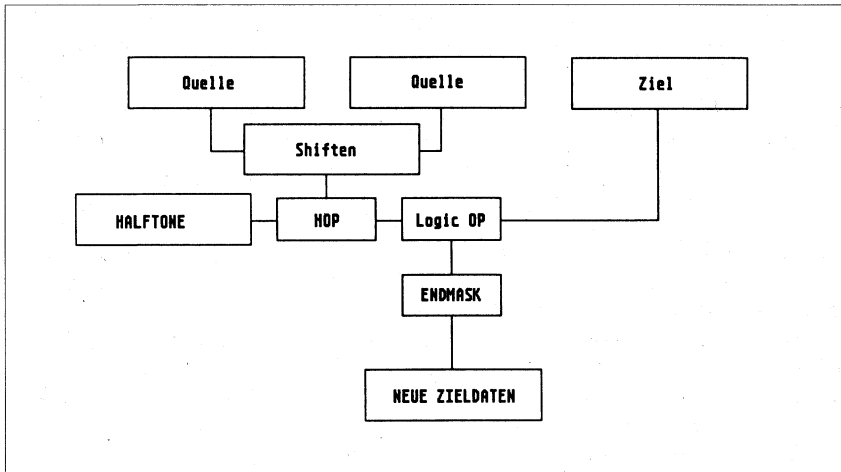


Abb. 19.47: Die einzelnen Register

Bit	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
\$FF8A00	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 0
\$FF8A02	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 1
\$FF8A04	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 2
\$FF8A06	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 3
\$FF8A08	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 4
\$FF8A0A	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 5
\$FF8A0C	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 6
\$FF8A0E	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 7
\$FF8A10	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 8
\$FF8A12	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 9
\$FF8A14	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 10
\$FF8A16	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 11
\$FF8A18	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 12
\$FF8A1A	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 13
\$FF8A1C	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 14
\$FF8A1E	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Halbton RAM 15

Abb. 19.48: Die Halbton-RAMs

Genausogut eignet er sich aber auch für allgemeine Kopier- und Transferaufgaben, z.B. ließe sich damit eine sehr schnelle RAM-Disk realisieren, die sogar noch wie eine Harddisk Interrupt-gesteuert laufen könnte. Im folgenden stellen wir Ihnen die einzelnen Register, über die der Blitter programmiert werden kann, im einzelnen vor. Die ersten 16 Register des Blitters werden als Halbton-RAM bezeichnet und enthalten ein Raster, das je nach der verwendeten Halbton-Operation verwendet wird. Die Register sind jeweils 16 Bit breit. Wenn das Raster benutzt wird, so wird jeweils ein Register für eine Zeile verwendet. Das Raster wiederholt sich also alle 16 Zeilen. Das später noch beschriebene "Line Number" Register bestimmt, welches Halbton-Register als nächstes benutzt wird.

Das nächste Register heißt "Source X Increment". Es ist ein vorzeichenbehaftetes 15-Bit-Register. Das unterste Bit wird ignoriert und als Null betrachtet. Dadurch sind nur gerade Werte möglich. Das Register gibt den Offset in Byte auf das nächste Source-Wort in der gleichen Zeile an. Normalerweise steht dort beim Atari im Monochrom-Modus eine Zwei. Dies ist auch der Fall, wenn im Farbmodus alle Ebenen gleichzeitig kopiert werden sollen. Soll im Midres- bzw. Lowres-Modus nur jeweils eine Ebene kopiert werden, so muß hier 4 bzw. 8 eingetragen werden.

Bit	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
\$FF8A20	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	Source X Increment
																	immer Null, nur gerade Increments

Abb. 19.49: Source X Increment

Das "Source Y Increment"-Register bestimmt, wieviel Bytes auf die aktuelle Source-Adresse addiert werden müssen, um vom Ende der aktuellen Zeile an den Anfang der nächsten Zeile zu gelangen. Im Monochrom-Modus ist eine Pixelreihe 80 Bytes lang; wenn nur ein Ausschnitt von 20 Bytes kopiert wird, so ergibt sich für "Source Y Increment" ein Wert von 60.

Bit	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
\$FF8A22	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	Source Y Increment
																	immer Null, nur gerade Increments

Abb. 19.50: Source Y Increment

Das "Source Address"-Register bestimmt die Anfangsadresse beim Beginn des Kopierens. Es kann mit einem Langwortzugriff gelesen oder beschrieben werden. Die Bits 0 sowie 24-31 sind immer Null, da nur gerade 24-Bit-Adressen verwendet werden. Der Inhalt dieses Registers wird im Laufe der Operation mit Hilfe der oben beschriebenen "Increment"-Register erhöht (oder erniedrigt, je nach Vorzeichen der "Increment"-Register). Beim Lesen des "Source Address"-Registers erhält man die Adresse des Source-Wortes, das als nächstes benutzt wird.

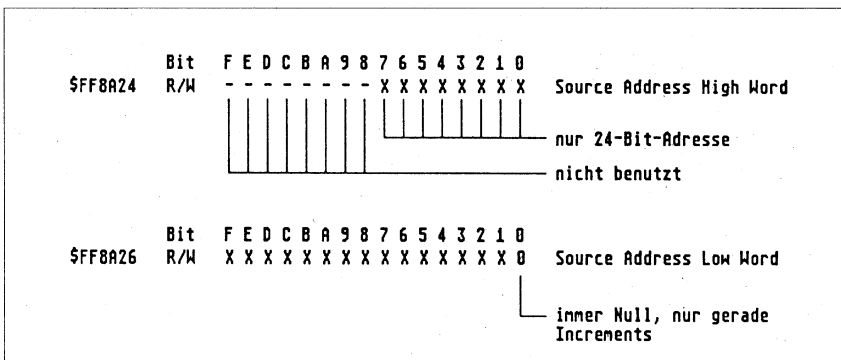


Abb. 19.51: Source Address Low- und High-Word

Die nächsten drei Register enthalten die sogenannte Endmask, mit der es folgende Bewandnis hat. Da der Blitter pixelorientiert arbeitet, die Bus-Zugriffe auf das RAM jedoch immer wortweise geschehen, können beim ersten sowie beim letzten gelesenen Wort jeweils Bits betroffen sein, die eigentlich nicht verändert werden dürfen. Die Endmasken geben nun an, welche Bits verändert und welche unverändert bleiben. Da über den Prozessor-Bus jeweils immer 16 Bits geschrieben werden, muß also das Zielwort zuerst gelesen werden, dann die erlaubten Bits verändert und das Ergebnis dann zurückgeschrieben werden (Read-Modify-Write-Zyklus). Die Endmask 1 gilt dabei für die linke Seite (Beginn einer Zeile), während Endmask 3 für die rechte Seite (Ende der Zeile) gilt. Endmask 2 wird bei allen anderen Worten benutzt und ist normalerweise \$FFFF, d.h. alle Bits werden verändert. Dabei ist auch ein vorheriges Lesen des Zielwortes nicht erforderlich. (Es sei denn, dies ist durch die logische Verknüpfung von Ziel und Quelle bedingt.) Das im Anschluß an die Registerbeschreibung stehende Beispiel dürfte die Bedeutung klarer werden lassen.

	Bit	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
\$FF8A28	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Endmask 1
\$FF8A2A	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Endmask 2
\$FF8A2C	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Endmask 3

Abb. 19.52: Endmask 1-3

Die nächsten drei Register sind "Destination X Increment", "Destination Y Increment" sowie "Destination Address". Sie haben die gleiche Bedeutung wie die zuvor besprochenen Register der Source, hier jedoch für das Ziel.

	Bit	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
\$FF8A2E	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	Destination X Increment
																		immer Null, nur gerade Increments

Abb. 19.53: Destination X Increment

	Bit	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
\$FF8A30	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	Destination Y Increment
																		immer Null, nur gerade Increments

Abb. 19.54: Destination Y Increment

	Bit	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
\$FF8A32	R/W	-	-	-	-	-	-	-	-	X	X	X	X	X	X	X	X	Destination Address High Word
																		nur 24-Bit-Adressen
																		nicht benutzt
\$FF8A34	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	Destination Address Low Word
																		immer Null, nur gerade Adressen

Abb. 19.55: Destination Address Low und High Word

Das "X Count"-Register gibt an, wieviel Worte in einer Zielzeile stehen. Der minimale Wert ist eins, der höchste ist 65536 (\$0000). Beim Lesen dieses Registers erhält man die Anzahl der noch in dieser Zeile zu übertragenen Worte. Wird das "X Count"-Register mit eins geladen, so werden die Werte in "Destination X Increment" sowie "Source X Increment" nicht benutzt, da die Zeile nach einem Wort schon zu Ende ist und das entsprechende "Y-Increment" direkt benutzt wird.

Das "Y Count"-Register schließlich bestimmt die Anzahl der Zeilen. Der kleinste Wert ist wieder eins, ein Wert von Null wird wieder als 65536 interpretiert. Beim Lesen dieses Registers erhält man die Anzahl Zeilen, die noch kopiert werden müssen. Nach jeder übertragenen Zeile wird der Wert dekrementiert, bis Null erreicht wird, womit die Übertragung beendet wird.

	Bit	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
\$FF8A36	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X-Count
\$FF8A38	R/W	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Y-Count

Abb. 19.56: X- und Y-Count

Alle bis jetzt beschriebenen Register können nur als Wort bzw. Langwort gelesen und geschrieben werden; ein Byte-Zugriff ist nicht erlaubt. Das HOP-Register bestimmt die Kombination von Quelle und Halbton-RAM. Die untersten beiden Bits haben folgende Bedeutung:

HOP	Kombination
0	alles 1-Bits
1	Halbton-RAM
2	Source
3	Source AND Halbton-RAM

Sie können also bestimmen, ob die Quelle unverändert benutzt wird (HOP = 2), ob sie mit dem Halbton-RAM logisch UND verknüpft wird (HOP = 3) oder ob nur das Halbton-RAM benutzt wird (HOP = 1). Dies ist z.B. nützlich, wenn eine Fläche mit einem Raster gefüllt werden soll. Weiterhin ist es noch möglich, das Ziel mit 1-Bits zu füllen (HOP = 0). Wenn das Halbton-RAM benutzt wird, so bestimmt ein weiteres Register, das noch beschrieben wird, welches der 16 Halbton-Register benutzt wird.

	Bit	7	6	5	4	3	2	1	0	
\$FF8A3A	R/W	-	-	-	-	-	X	X		HOP Halbton-Operation

Abb. 19.57: HOP (Halbton-Operation)

Das nächste Register bestimmt, nach welcher logischen Operation Ziel und Quelle verknüpft werden, um den neuen Zielwert zu erhalten. Hierbei gibt es 16 verschiedene Möglichkeiten, die in der folgenden Tabelle stehen.

(~s&~d)	(~s&d)	(s&~d)	(s&d)	Operation	neues Ziel
0	0	0	0	0	alles '0'-Bits
0	0	0	1	1	Quelle AND Ziel
0	0	1	0	2	Quelle AND NOT Ziel
0	0	1	1	3	Quelle
0	1	0	0	4	NOT Quelle AND Ziel
0	1	0	1	5	Ziel
0	1	1	0	6	Quelle XOR Ziel
0	1	1	1	7	Quelle OR Ziel
1	0	0	0	8	NOT Quelle AND NOT Ziel
1	0	0	1	9	NOT Quelle XOR Ziel
1	0	1	0	10	NOT Ziel
1	0	1	1	11	Quelle OR NOT Ziel
1	1	0	0	12	NOT Quelle
1	1	0	1	13	NOT Quelle OR Ziel
1	1	1	0	14	NOT Quelle OR NOT Ziel
1	1	1	1	15	alles '1'-Bits

	Bit	7	6	5	4	3	2	1	0	
\$FF8A3B	R/W	-	-	-	-	X	X	X	X	OP Logische Operation

Abb. 19.58: OP (Logische Operation)

Die wichtigsten Operationen sind 3 (Replace Mode, Quelle ersetzt Ziel) sowie 6 und 7 (XOR-Modus bzw. Überlagern von Ziel und Quelle, OR-Mode).

Im nächsten Register sind gleich mehrere Funktionen zusammengefaßt. Die untersten vier Bits bestimmen, welches der 16 Halbton-RAM-Register gerade benutzt wird. Der Wert wird jeweils nach einer Zeile inkrementiert oder dekrementiert, je nach Vorzeichen des Destination Y-Registers.

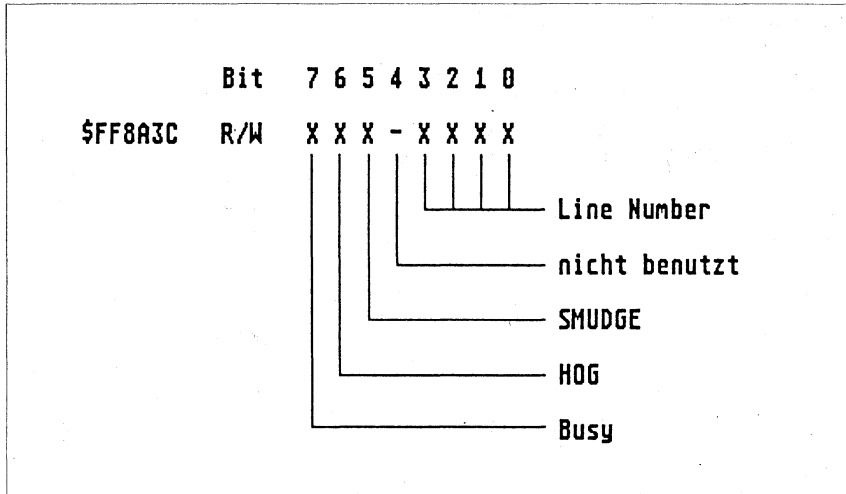


Abb. 19.59: Das Register FF8A3C

Wenn das SMUDGE-Bit gesetzt ist, wird die Nummer der Halbton-RAM-Register nicht wie zuvor beschrieben bestimmt, sondern aus den untersten vier Bits der (bereits verschobenen) Quelldaten. Die gewählte Halbton-Operation (HOP) bleibt weiterhin aktiv. Damit lassen sich spezielle Effekte erzielen.

Das nächste Bit in diesem Register bestimmt die Art und Weise der Bus-Zugriffe des Blitters. Wenn das HOG-Bit gelöscht ist, benutzen Blitter und Prozessor den Bus zu gleichen Teilen. Nach jeweils 64 Bus-Zyklen wird der Blitter angehalten, und der Prozessor erhält für die nächsten 64 Zyklen den Bus. Ist das HOG-Bit dagegen gesetzt, wird der Prozessor so lange angehalten, bis der Blitter seine Operation beendet hat. In beiden Fällen haben andere DMA-Geräte (Harddisk, Floppy) Priorität vor dem Blitter. Durch den Prefetch-Mechanismus des 68000-Prozessors kann man selbst im HOG-Modus nicht davon ausgehen, daß nach dem Start des Blitters die nächste Prozessor-Anweisung

erst dann ausgeführt wird, wenn der Blitter bereits fertig ist. Dazu ist das BUSY-Bit des Blitters abzufragen.

Das BUSY-Bit wird gesetzt, nachdem alle anderen Blitter-Register initialisiert wurden, um den Blitter zu starten. Es bleibt so lange gesetzt, bis der Blitter seine Operation beendet hat. Da der Interrupt-Ausgang des Blitters den Zustand dieses Bits widerspiegelt, kann das Ende der Blitter-Operation auch einen Interrupt auslösen, der auf Bit 3 des GPIF im MFP 68901 geführt ist (siehe Kapitel über die Exceptions).

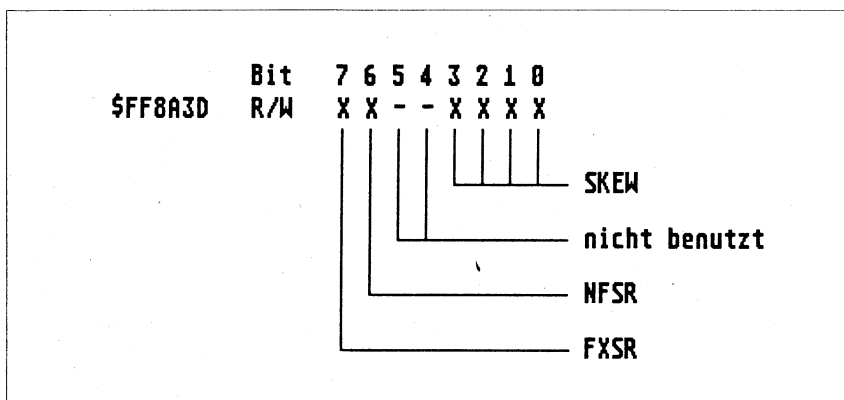


Abb. 19.60: Das Register FF8A3D

Das letzte Register des Blitters hat wieder mehrere Funktionen. Die untersten vier Bit bestimmen die Verschiebung des Quelloperanden, um ihn mit den Zieloperanden auf Deckung zu bringen. Da der Blitter zwar bitorientiert arbeitet, die Bus-Zugriffe jedoch wortweise geschehen, müssen die Quelldaten soweit verschoben werden, daß sie mit den Bit-Positionen von Halbton-Maske und Zieldaten übereinstimmen. Bei Bedarf müssen also zwei Worte der Quelldaten gelesen werden, um daraus durch Schieben die relevanten Bits in ein 16-Bit Quellregister zu holen (siehe auch Blockdiagramm, Abb. 19.47).

Im Zusammenhang damit stehen auch FXSR und NFSR. Die Bezeichnungen stehen für "Force eXtra Source Read" und "No Final Source Read". Wenn das FXSR-Bit gesetzt ist, wird am Beginn jeder Zeile ein zusätzliches Quellwort gelesen. Das NFSR-Bit wird dann gesetzt, wenn das letzte Wort jeder Quellzeile nicht gelesen zu werden braucht. Die Benutzung dieser Bits erfordert Änderungen von Source Y Increment und Source Address Register.

Normalerweise nimmt Ihnen das Betriebssystem die direkte Programmierung des Blitters ab. Wenn Sie die entsprechenden LINE-A oder VDI-Funktionen benutzen, entscheidet das Betriebssystem aufgrund des eingestellten Blitter-Modus (siehe XBIOS-Funktion 64), ob die Funktionen rein per Software oder aber mit dem Blitter erledigt werden. Das nachfolgend abgedruckte Beispielprogramm greift direkt auf den Blitter zu und muß deshalb im Supervisor-Modus des 68000 ablaufen, da sonst ein Bus-Error erzeugt wird.

Um mehr von dem Beispiel zu haben, sollten Sie vorher die Beschreibung der Funktion BITBLT (\$A007) sowie die dazugehörigen Beispiele lesen.

Das Beispielprogramm "blitter" einen Bereich des Bildschirms an eine andere Stelle. Die enthaltene Funktion "blit_it" ist jedoch universeller und kann beliebige RAM-Bereiche blitten. Betrachten Sie das Programm nur als Anregung. Die Hauptaufgabe dieses Beispiels ist es, Ihnen zu zeigen, wie man aus gegebenen Bildbereichen ("forms") und Pixel-Koordinaten die Werte für die einzelnen Register des Blitters bestimmt.

```
blitter    equ        $ff8a00
*          blitter register offsets
halftone   equ        0
src_xinc   equ        $20
src_yinc   equ        $22
src_addr   equ        $24
endmask1   equ        $28
endmask2   equ        $2a
endmask3   equ        $2c
dst_xinc   equ        $2e
dst_yinc   equ        $30
dst_addr   equ        $32
x_count    equ        $36
y_count    equ        $38
hop        equ        $3a
op         equ        $3b
line_num   equ        $3c
skew       equ        $3d
*          blitter register flags
flinebusy  equ        7          busy bit
*          blitter register bit masken
nhop_src   equ        $02        halbtone operation: source
mskewfxsr  equ        $80        fxsr maske
mskewnfsr  equ        $40        nfsr maske
mlinebusy  equ        $80        busy maske

physbase   equ        2          bildschirm adresse holen
xbios      equ        14

demo:
    lea     para,a4
    move    #physbase,-(sp)
```



```

trap      #xbios
addq.l    #2,sp          bildschirm adresse holen
move.l    d0,src_form(a4) quelle und
move.l    d0,dst_form(a4) ziel sind bildschirm
moveq     #2,d0          2 bytes offset
move      d0,src_nxwd(a4) auf naechstes wort
move      d0,dst_nxwd(a4) in gleicher farbebene
moveq     #80,d0         eine zeile ist 80 bytes lang
move      d0,src_nxln(a4) (monochrom modus)
move      d0,dst_nxln(a4)
moveq     #2,d0          offset auf naechste farbeebene
move      d0,src_nxpl(a4) wird nicht gebraucht
move      d0,dst_nxpl(a4) da monochrom
move      #25,src_xmin(a4) x1-koordinate source
move      #34,src_ymin(a4) y1-koordinate source
move      #220,dst_xmin(a4) x1-koordinate ziel
move      #234,dst_ymin(a4) y1-koordinate ziel
move      #77,width(a4)  breite in pixel
move      #50,height(a4) hoehe in pixel (anzahl zeilen)
move      #1,planes(a4)  monochrom
jsr       blit_it        und blitten !
rts       fertig
para      dc.w          17    platz fuer parameterblock
*         end masken
lf_endmaks:
          dc.w          $ffff
rt_endmask:
          dc.w          $7fff
          dc.w          $3fff
          dc.w          $1fff
          dc.w          $0fff
          dc.w          $07ff
          dc.w          $03ff
          dc.w          $01ff
          dc.w          $00ff
          dc.w          $007f
          dc.w          $003f
          dc.w          $001f
          dc.w          $000f
          dc.w          $0007
          dc.w          $0003
          dc.w          $0001
          dc.w          $0000
*
*         input: zeiger auf 34 byte parameter block in a4
*
*
src_form  equ      0        base address source memory form
src_nxwd  equ      4        offset next word in source
src_nxln  equ      6        source form width
src_nxpl  equ      8        offset between source planes
src_xmin  equ     10        source x1
src_ymin  equ     12        source y1

```

```

dst_form equ      14      base address dest memory form
dst_nxwd equ      18      offset next word in dest
dst_nxln equ      20      dest form width
dst_nxpl equ      22      offset between dst planes
dst_xmin equ      24      dest x1
dst_ymin equ      26      dest y1
width     equ      28      breite in pixel
height    equ      30      hoehe in pixel
planes    equ      32      anzahl planes

blit_it:
    lea        blitter,a5
*   brechnung von xmax aus xmin und width
    move       width(a4),d6
    subq       #1,d6          width -1
    move       src_xmin(a4),d0
    move       d0,d1
    add        d6,d1          src_xmax
    move       dst_xmin(a4),d2
    move       d2,d3
    add        d6,d3          dst_xmax
    moveq      #$f,d6         mod 16 maske
    move       d2,d4          dst_xmin
    and        d6,d4          dst_xmin mod 16
    add        d4,d4          zeiger auf left end mask table
    move       lf_endmask(pc,d4),d4 linke end mask
    move       d3,d5          dst_xmax
    and        d6,d5          dst_xmax mod 16
    add        d5,d5          zeiger auf right end mask
                                table
    move       rt_endmask(pc,d5),d5 invertierte rechte end mask
    not        d5             rechte end maske
*   berechnung skew
*   ((dst_xmin mod 16) - (src_xmin mod 16)) mod 16
*
*   bestimmung von FXSR und NFSR
*
*   3 bit index in tabelle
*
*   bit 0      0 src_xmin mod 16 >= dst_xmin mod 16
*               1 src_xmin mod 16 >  dst_xmin mod 16
*
*   bit 1      0 src_xmax/16 - src_xmin/16 <> dst_xmax/16 -
*               dst_xmin/16
*               0 src_xmax/16 - src_xmin/16 <> dst_xmax/16 -
*               dst_xmin/16
*
*   bit 2      0 dst_span gleich mehrere worte
*               1 dst_span gleich ein wort
*
    move       d2,d7          dst_xmin
    and        d6,d7          dst_xmin mod 16
    and        d0,d6          src_xmin mod 16
    sub        d6,d7          dst_xmin mod 16 - src_xmin mod 16

```

```

*
      clr      d6      > ? cy = 1 : cy = 0
      addx     d6,d6    cy nach bit 0
      lsr      #4,d0    src_xmin / 16
      lsr      #4,d1    src_xmax / 16
      sub      d0,d1    src_span - 1
      lsr      #4,d2    dst_xmin / 16
      lsr      #4,d3    dst_xmax / 16
      sub      d2,d3    dst_span - 1
      bne      set_endmask
*
      falls dst_span = ein wort, stehen beide endmask in endmask 1
*
      der blitter ignoriert endmask 2
      and      d5,d4
      addq     #4,d6     d6 bit 2 = 1 ein wort destination
set_endmask:
      move     d4,endmask1(a5)    linke endmask
      move     #$ffff,endmask2(a5) mittlere endmask
      move     d5,endmask3(a5)    rechte endmask
      cmp      d1,d3              anzahl source und dest worte
                                   gleich ?
      bne      set_count          nein
      addq     #2,d6              d6 bit 1 = 1 gleiche anzahl
                                   worte
set_count:
      move     d3,d4
      addq     #1,d4              anzahl worte in dest zeile
      move     d4,x_count(a5)
*
*
*      source start adresse bestimmen
*
*      src_form + (src_ymin * src_nxln) * (src_xmin/16 * src_nxwd)
*
      move.l   src_form(a4),a0    a0 -> start src form
      move     src_ymin(a4),d4    offset in lines to ymin
      move     src_nxln(a4),d5    lenght src line
      mulu     d5,d4
      add.l    d4,a0              a0 -> (0, ymin)
      move     src_nxwd(a4),d4    offset auf naechstes word
      move     d4,src_xinc(a5)
      mulu     d4,d0
      add.l    d0,a0              a0 -> erstes word (xmin, ymin)
      mulu     d4,d1              breite source line in bytes
      sub      d1,d5
      move     d5,src_yinc(a5)    offset ende zeile anfang
                                   naechste
*
      destination start adresse bestimmen
      move.l   dst_form(a4),a1    a1 -> start dst form
      move     dst_ymin(a4),d4
      move     dst_nxln(a4),d5
      mulu     d5,d4
      add.l    d4,a1
      move     dst_nxwd(a4),d4
      move     d4,dst_xinc(a5)

```

```

mulu      d4,d2
add.l     d2,d1
* dst yinc bestimmen
mulu      d4,d3
sub       d3,d5
move      d5,dst_yinc(a5)    destination y increment
and.b     #$f,d7
or.b      skew_flags(pc,d6),d7 skew-flags aus tabelle
move.b    d7,skew(a5)        in blitter
move.b    #mhopsrc,hops(a5)  halbtone operation: nur source
move.b    #3,ops(a5)         replace mode
lea       line_num(a5),a2    zeiger auf line number register
move.b    #flinebusy,d2     busy bit nach d2
move      planes(a4),d7      anzahl bildebene
bra       begin
skew_flags:
dc.b      mskewnfsr
dc.b      mskewfxsr
dc.b      0
dc.b      mskewnfsr+mskewfxsr
dc.b      0
dc.b      mskewfxsr
dc.b      0
dc.b      0
next_plane:
move.l    a0,src_addr(a5)    source adresse laden
move.l    a1,dst_addr(a5)    destination adresse laden
move      height(a4),y_count(a5) anzahl zeilen
move.b    #mlinebusy,(a2)    blitter starten
add       src_nxpl(a4),a0     start naechste src plane
add       dst_nxpl(a4),a1     start naechste dst plane
restart:
bset.b    d2,(a2)            restart blitter
nop
bne       restart            noch nicht fertig ?
begin     dbra      d7,next_plane naechste bildebene
rts

```

19.8 Die Echtzeituhr im Mega-ST

Bereits kurze Zeit nach dem Erscheinen des ST wurde klar, daß man bei diesem Rechner an der falschen Ecke gespart hatte. Zwar wird beim ST die Uhrzeit im Tastatur-Prozessor in Sekundenschritten mitgeführt, auch das GEMDOS verwaltet eine eigene softwaremäßig realisierte Uhr im 2-Sekunden-Takt, aber das ewige Setzen der Zeit nach dem Einschalten ist natürlich sehr lästig. Das gilt besonders, da ja der Rechner beim Speichern einer Datei das einge-

stellte Datum mit auf der Diskette vermerkt. Da aber in vielen Fällen das Setzen der Uhr vergessen wurde, konnte man sich kaum auf die Angaben verlassen. Um diesem Übel abzuhelpen, wurden von eifrigen Tüftlern verschiedene Lösungen ersonnen. Diese Lösungen reichen vom Puffern des Tastatur-Prozessors über Uhren-ICs, die am ROM-Port angeschlossen werden, bis zu Schaltungen, die einem Betriebssystem-ROM "untergeschoben" werden.

	Bit	7	6	5	4	3	2	1	0	
\$FFFC21	R/W	-	-	-	X	X	X	X		Sekunden Einer
\$FFFC23	R/W	-	-	-	X	X	X	X		Sekunden Zehner
\$FFFC25	R/W	-	-	-	X	X	X	X		Minuten Einer
\$FFFC27	R/W	-	-	-	X	X	X	X		Minuten Zehner
\$FFFC29	R/W	-	-	-	X	X	X	X		Stunden Einer
\$FFFC2B	R/W	-	-	-	X	X	X	X		Stunden Zehner
\$FFFC2D	R/W	-	-	-	X	X	X	X		Wochentag
\$FFFC2F	R/W	-	-	-	X	X	X	X		Tag Einer
\$FFFC31	R/W	-	-	-	X	X	X	X		Tag Zehner
\$FFFC33	R/W	-	-	-	X	X	X	X		Monat Einer
\$FFFC35	R/W	-	-	-	X	X	X	X		Monat Zehner
\$FFFC37	R/W	-	-	-	X	X	X	X		Jahr Einer
\$FFFC39	R/W	-	-	-	X	X	X	X		Jahr Zehner
\$FFFC3B	R/W	-	-	-	X	X	X	X		Steuerregister
\$FFFC3D	R/W	-	-	-	X	X	X	X		Steuerregister
\$FFFC3F	R/W	-	-	-	X	X	X	X		Steuerregister



Die Bits 4-7 werden grundsätzlich nicht benutzt.

Abb. 19.61: Die Register der internen Uhr

Der RP 5 C 15 scheint in mancher Hinsicht den auch sonst üblichen Uhren-ICs zu gleichen. Er besitzt einen je 4 Bit breiten Daten- und Adreß-Bus, über den sich insgesamt 16 Register adressieren lassen. Jedes der Register hat eine Datenbreite von 4 Bit und enthält eine Stelle der Uhrzeit oder des Datums im BCD-Format. Die drei zusätzlichen Register auf den Adressen \$FFFC3B bis \$FFFC3F sind uns leider nicht bekannt. Zwar werden diese Register beim Setzen der Uhr beschrieben, allerdings lassen sich aus dem Disassembler-Listing keine weiteren Schlußfolgerungen ziehen. Seinen Takt bezieht der Uhren-Chip von einem kleinen Quarz, der auf einer Frequenz von 32,768 kHz schwingt. Die Ansteuerung dieses eigentlich recht langsamen ICs wird über einen PAL, ein programmierbares Logik-Array vorgenommen. Da alle Register der Uhr im Adreßraum des Prozessors liegen, steht jetzt endlich eine (jedenfalls in Sekundenschritten) genaue und einfach auszulesende Uhr zur Verfügung. Beim

Anfordern der Uhrzeit über die entsprechenden XBIOS-Funktionen entscheidet das Betriebssystem des Mega ST selbständig, ob die Uhrzeit von der Hardware-Uhr oder vom Tastaturprozessor geholt wird, je nachdem, ob die Hardware-Uhr vorhanden ist oder nicht.

19.9 Die Belegung des I/O-Bereichs des Atari ST

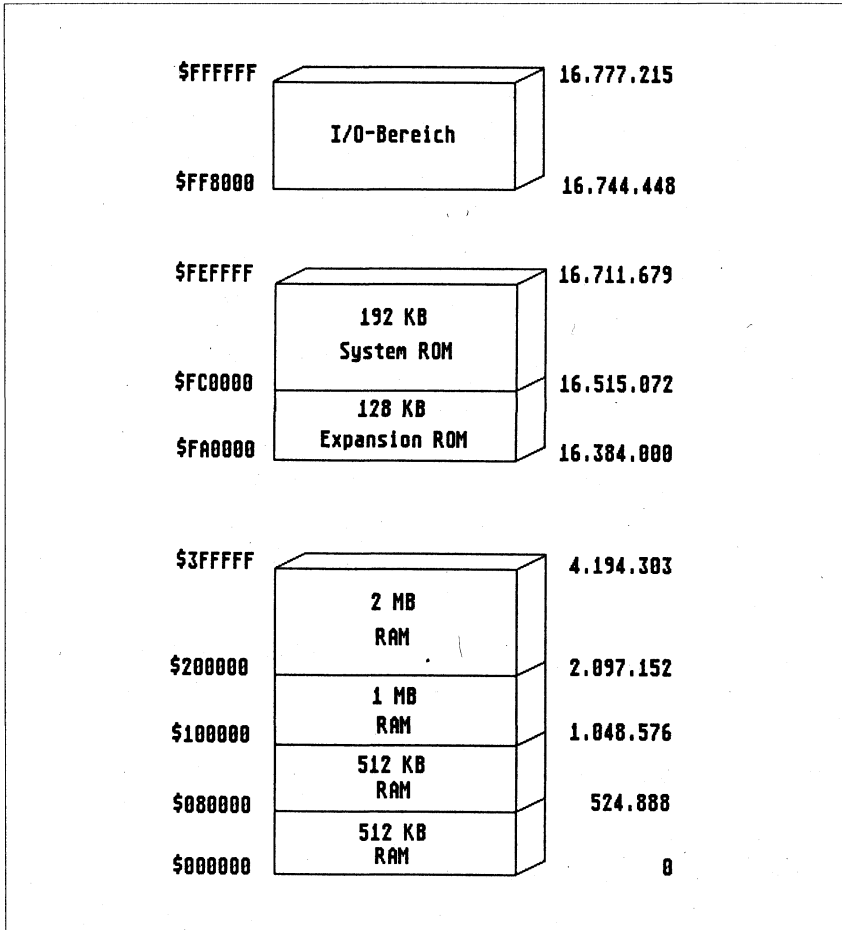


Abb. 19.62: Die Speicherbelegung des Atari ST

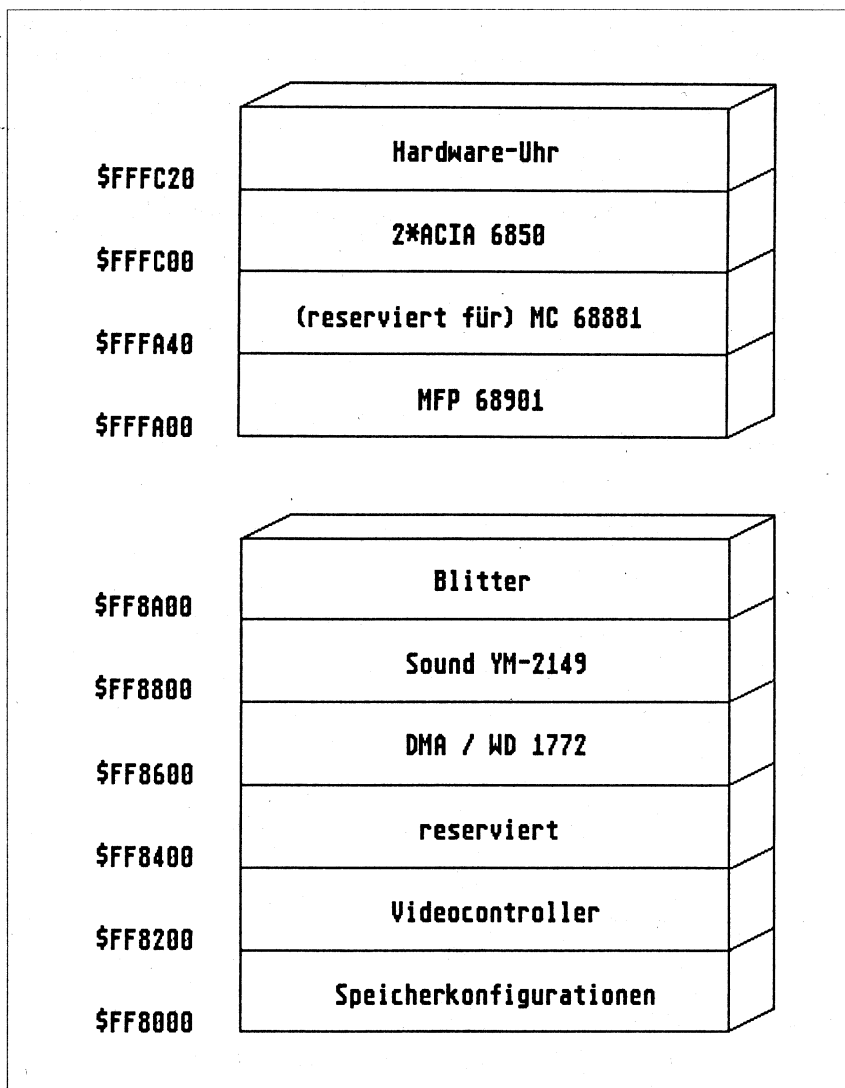


Abb. 19.63: Die Aufteilung des I/O-Bereichs

20. Die Schnittstellen

In Teil A dieses Buches haben wir die "gewöhnlichen" Schnittstellen unter die Lupe genommen, jene also, die bei der täglichen Arbeit mit dem ST gebraucht werden. Dieses Kapitel soll dazu dienen, die Interna weiter offenzulegen.

20.1 Die Tastatur

Glauben Sie, es sei müßig, über eine so triviale Sache wie die Tastatur ausführlich zu berichten, da Tastaturen ja doch alle gleich funktionieren? Weit gefehlt! Eigentlich müßte die Überschrift lauten: "Keyboard Subsystem" oder so ähnlich. Die Tastatur wird nämlich von einem eigenen leibhaftigen Prozessor gesteuert. Welche Konsequenzen das für Sie als Programmierer hat, werden Sie gleich sehen.

Als Prozessor dient ein Single Chip-Computer aus der Familie 6800, nämlich der 6301. Single Chip bedeutet, daß alles, was zum Betrieb benötigt wird, sich in einem einzigen IC befindet. In der Tat ist auf der Tastatur-Platine auch nur der 6301 mit ein paar passiven Bauelementen zu sehen. So verfügt der 6301 auf dem Chip über ROM, RAM, etliche I/O-Leitungen und sogar über eine serielle Schnittstelle. Über diese wird übrigens der Verkehr mit der Hauptplatine abgewickelt. Der Vorteil einer derartigen Anordnung liegt auf der Hand: Der Hauptrechner braucht sich nämlich nicht mehr mit der zeitraubenden Abfrage der Tastaturmatrix abzulassen. Statt dessen kann er sich voll der Abarbeitung Ihrer Programme widmen. Der Tastaturprozessor meldet sich selbsttätig, falls ein Ereignis eintreten sollte, von dem das Betriebssystem in Kenntnis zu setzen ist.

Dem 6301 obliegt aber nicht nur die relativ langweilige Bewachung der Tastatur. Er handelt auch souverän die wesentlich komplizierteren Aktionen ab, die im Zusammenhang mit der Maus notwendig sind. So bekommt der Hauptrechner bei evtl. Bewegungen des Tierchens die neuen X- und Y-Koordinaten gleich mundgerecht serviert. Überflüssig zu erwähnen, daß auch alle Vorgänge, die möglicherweise angeschlossene Joysticks verursachen, dem Boß gepetzt werden.

Außerdem ist noch eine Echtzeituhr enthalten, die in Schritten von 1 sec zählt. In der Abbildung 20.1 finden Sie eine Übersicht über die Schnittstellen zum 68000. Wie Sie sehen, wird der Hauptrechner so wenig wie möglich belastigt. Die ACIA 6850 sorgt dafür, daß er nur gestört wird, wenn tatsächlich ein Byte vollständig von der Tastatur empfangen wurde. Die ACIA ist übrigens unter

den Adressen \$FFFC00 (Control-Register) und \$FFFC02 (Datenregister) zu erreichen. Die einzige Verbindung zur Tastatur findet über die Leitungen K14 und K15 statt. K bezeichnet übrigens die Steckverbindung, über die die Tastatur mit der Hauptplatine verbunden ist.

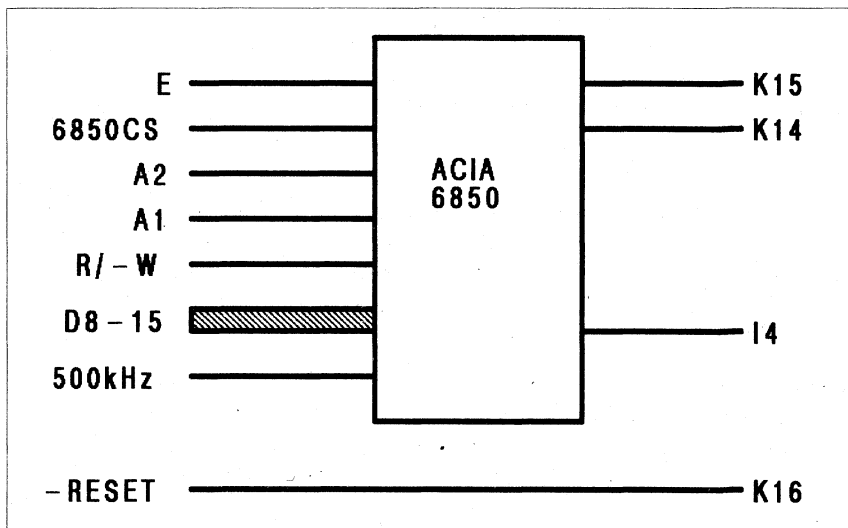


Abb. 20.1.: Die Schnittstellen des 68000

Die Meldung, daß die ACIA ein Byte empfangen hat, wird über die Leitung I4 zunächst an den MFP 68901 gemeldet, der dann beim 68000 einen Interrupt auslöst. Die Taktfrequenz von 500 KHz stammt vom GLUE. Daraus resultiert die "krumme" Übertragungsgeschwindigkeit von 7812.5 Baud. Falls es Sie wundert, daß auch Daten zum Tastaturprozessor übertragen werden können, finden Sie des Rätsels Lösung im Kapitel 20.1.2.

Aber Vorsicht: Abhängig vom Baujahr Ihres Atari ST kann es durchaus sein, daß Sie die in der Blockschaltung dargestellten Komponenten auf der Tastaturplatine nicht lokalisieren können. Der Grund dafür ist, daß die Platine im Laufe der Zeit mehrfach überarbeitet wurde. Hierdurch verlieren die folgenden Ausführungen jedoch nicht ihre prinzipielle Gültigkeit.

Sollten Sie die Hardware etwa modifizieren wollen, legen Sie die Tastatur mit den Tasten nach unten auf einen Tisch, indem Sie den Gehäuserahmen an den Rändern mit Klötzchen abstützen, so daß die Tasten nicht die Tischfläche be-

rühren. Nun können Sie die Platine abschrauben, um die Leiterbahnen verfolgen oder ausmessen zu können.

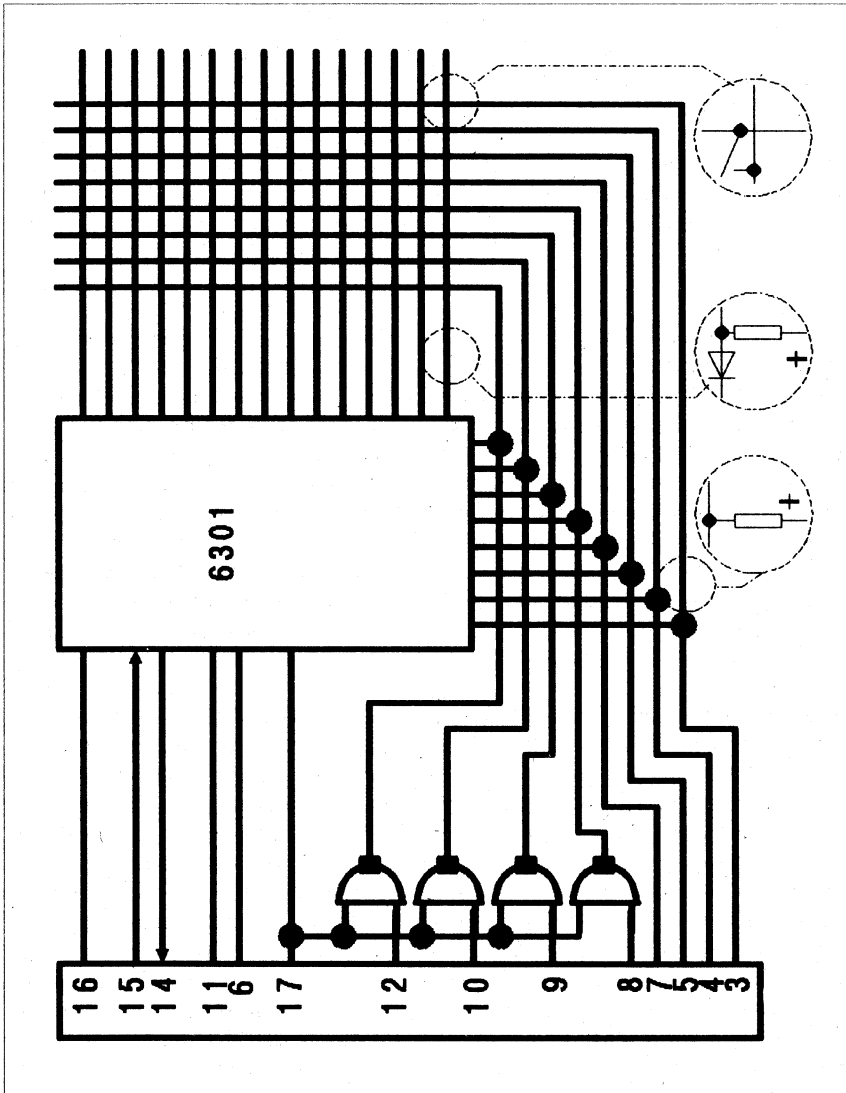


Abb. 20.2: Die Beschaltung des 6301

Das Blockschaltbild der Tastaturplatine finden Sie in Abbildung 20.2. Die Funktion ist ebenso einfach wie das Bild übersichtlich ist. Kein Wunder, denn je umfangreicher die Software, desto einfacher die Hardware.

So verfügt der Prozessor denn auch über stattliche 4 KByte ROM. Das RAM ist mit 128 Byte vergleichsweise klein, aber es dient ja auch nur als Puffer und zur Aufnahme einiger Pointer und Zähler.

Die mit K bezeichneten Leitungen sind wieder der Steckverbindung zur Hauptplatine zugeordnet. Mit wenigen Ausnahmen werden sie dort gleich zu den Anschlüssen für Maus und Joystick durchgeschleift. K16 ist die Reset-Leitung vom 68000. K15 führt die Sendedaten vom 6850, K14 die Sendedaten vom 6301.

Für die Abfrage der Tastaturmatrix sind die I/O-Ports 1(0-7), 3(1-7) und 4(0-7) zuständig. Über die Ports 3 und 4 wird zyklisch an je eine Leitung Low gelegt. Danach wird jedesmal der Zustand von Port 1 abgefragt. Ist eine Taste gedrückt, dringt der Low-Pegel zum Port 1 durch. Aus der Kombination der Werte an Port 3/4 und 1 läßt sich jede Taste identifizieren. Ist keine der Leitungen von Port 3/4 auf Low gelegt und dennoch ein Bit von Port 1 =0, so ist am Außenstecker 1 ein Joystick aktiv. Die Daten von Außenstecker 0, an den ja die Maus oder auch ein Joystick angeschlossen werden kann, dringen nicht zufällig durch, da sie erst mit Port 2 (Bit 0) durch die NAND-Gatter geschleust werden müssen. Die Drucktasten der Maus bzw. der Joysticks landen an Port 2 (1 u. 2). Die Zuordnungen der K-Leitungen zu den Signalnamen der Außenstecker finden Sie im nächsten Kapitel.

Zum Schluß noch ein kleines Kuriosum: Der Prozessor 6301 ist zwar völlig eigenständig, jedoch läßt er sich auch so konfigurieren, daß er mit einem externen ROM arbeitet. Dabei werden etliche Port-Leitungen zu Adreßleitungen umfunktioniert. Welche Konfiguration (von acht möglichen) der Prozessor annimmt, wird dadurch bestimmt, welche logischen Pegel während des Reset-Zyklus von außen an Port 2 (0-2) gelegt werden. Alle drei Leitungen High bringt den Prozessor in den Modus 7, eben den, der für die ihm hier zugedachte Aufgabe richtig ist. Nun hängen aber die Bits 1 und 2 an den Tasten der Maus. Läßt man diese während des Einschaltens in Ruhe, ist alles in Ordnung. Halten Sie jedoch beide gedrückt, geht der Prozessor in den Modus 1 und macht eine prächtige Bauchlandung, da die übrige Hardware für diese Betriebsart nicht vorgesehen ist. Sie merken das daran, daß sich der Maus-Cursor auf dem Bildschirm nicht bewegen läßt. Da hilft nur der Druck auf das Reset-Knöpfchen.

20.1.1 Die Maus

Dieses kleine Gertchen hat es in sich. Der Aufbau ist recht einfach, aber wirkungsvoll.

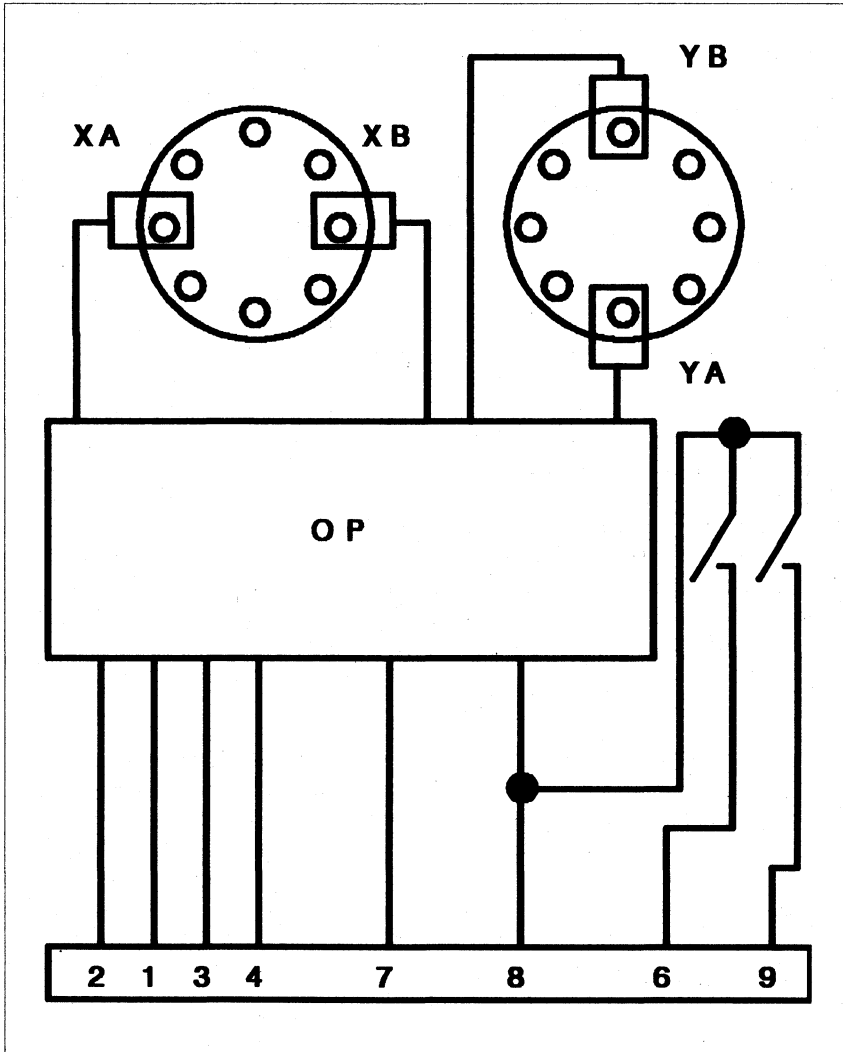


Abb. 20.3: Die Beschaltung des Maus-Ports

Es besteht im wesentlichen aus vier Lichtschranken, zwei Taktscheiben und dem Antriebsmechanismus. Die Aufgabe der Maus besteht darin, dem Computer Informationen über ihre Bewegungen zukommen zu lassen. Diese Information besteht aus den Komponenten: Richtung auf der X-Achse, Richtung auf der Y-Achse und zurückgelegter Weg auf jeder Achse.

Um das zu bewerkstelligen, treibt der von außen sichtbare Gummiball im Innern zwei Taktscheiben an, deren Antriebsachsen im Winkel von 90 Grad zueinander stehen. Je nachdem, in welche Richtung nun die Maus bewegt wird, dreht sich die eine oder andere Achse mehr oder weniger vor oder zurück.

Mausbewegung

Die absolute Bewegung auf jeder Achse festzustellen, ist überhaupt kein Problem. Von den Taktscheiben werden nämlich zugehörige Lichtschranken unterbrochen bzw. freigegeben. Man braucht nur noch die Impulse jeder Scheibe zu zählen, um über den zurückgelegten Weg auf jeder Achse informiert zu sein.

Schwieriger wird es, wenn auch noch die Bewegungsrichtung verlangt ist. Dazu haben jedoch die Konstrukteure zu einem genialen Trick gegriffen: Auf jeder Taktscheibe sitzt nämlich nicht nur eine, sondern zwei Lichtschranken, die sich auf dem Rad gegenüberstehen. Sie sind so angeordnet, daß sie durch die Scheibe nicht genau zur gleichen Zeit verdunkelt werden, sondern kurz nacheinander. Vielleicht wird dieser Umstand auf der Abbildung 20.3 nicht so deutlich; deshalb erklären wir es hier. Die Richtung läßt sich nun dadurch feststellen, daß man beobachtet, welche der beiden Lichtschranken zuerst unterbrochen wird. Deshalb sind auch die Impulse beider Lichtschranken nach außen geführt, insgesamt also vier. Entsprechend ihrer Bedeutung tragen sie denn auch die Namen XA, XB, YA und YB.

Die beiden Kontakte, die Sie auf dem Bild sehen, sollen die Drucktasten darstellen. Der große Kasten auf dem Bild ist ein vierfacher Operationsverstärker, der die recht buckeligen Lichtschrankenimpulse in schöne Rechtecksignale formt. Unten sehen Sie die Belegung der Control-Ports am Computer so abgebildet, wie Sie sie sehen, wenn Sie von außen darauf schauen. Hier die Bedeutung, hinter dem Schrägstrich die Bezeichnung, wenn ein Joystick angeschlossen ist, in Klammern die Pin-Nummer des Tastatur-Steckverbinders:

Port 0:		
1	XB/UP	(K12)
2	XA/DOWN	(K10)
3	YA/LEFT	(K9)
4	YB/RIGHT	(K8)
6	LEFT BUTTON/FIRE	(K11)
7	+5V	(K13)
8	GND	(K1)
9	RIGHT BUTTON	(K6)

Port 1:		
1	UP	(K7)
2	DOWN	(K5)
3	LEFT	(K4)
4	RIGHT	(K3)
5	Port 0 enable	K17)
6	FIRE	(K6)
7	+5V	K13)
8	GND	(K1)

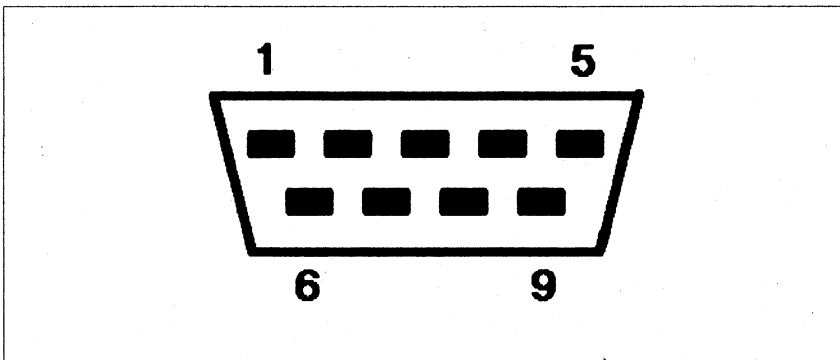


Abb. 20.4: Der Stecker für Maus und Joystick

20.1.2 Keyboard-Kommandos

Der Tastaturprozessor "versteht" etliche Kommandos, die sich darauf beziehen, wie z.B. die Maus zu behandeln ist. Man kann die Uhrzeit setzen, den internen Speicher auslesen usw. Ein Beispiel für die Anwendung finden Sie im Assembler-Listing am Ende des Kapitels.

Die "normale" Tätigkeit des Prozessors besteht ja darin, die Tastatur zu überwachen und jeden Tastendruck zu melden. Das geschieht in der Form, daß die Tastennummer beim Drücken der Taste ausgegeben wird. Beim Loslassen der Taste wird ebenfalls die Tastennummer gesendet, allerdings mit gesetztem Bit 7. Daraus folgt, daß es keine größere Tastennummer als 127 geben darf. Die Zuordnung der Tastennummern zu den Tasten finden Sie am Ende dieses Kapitels. In Wirklichkeit können diese Nummern aber nur bis 117 gehen, da nämlich Werte ab \$F6 für andere Zwecke reserviert sind. Es muß nämlich eine Möglichkeit geben, dem Hauptrechner auch andere Informationen als nur Tastennummern zu übergeben, z.B. die Uhrzeit oder die aktuelle Position der Maus. Da dies nicht in einem einzigen Byte abgehandelt werden kann, sondern nur in einem sogenannten Paket, kennzeichnen die Bytes ab \$F6 den Start eines solchen Pakets. Welches Kenn-Byte (sogenannter Header) welchem Paket voransteht, ist bei den einzelnen Kommandos beschrieben.

Ein Befehl an den Tastaturprozessor besteht aus dem Befehlscode (ein Byte) und evtl. notwendigen Parametern. Die folgende Beschreibung ist nach Befehls-Bytes sortiert.

\$07

Die Reaktion auf das Drücken einer der beiden Maus-Tasten wird eingestellt. Es wird ein Parameter-Byte mit folgendem Aufbau verlangt:

- Bit 0 = 1: Beim Drücken einer Maus-Taste wird die absolute Position gemeldet. Bit 2 muß =0 sein.
- Bit 1 = 1: Beim Loslassen einer Maus-Taste wird die absolute Position gemeldet. Bit 2 muß =0 sein.
- Bit 2 = 1: Die Maus-Tasten werden wie normale Tasten behandelt. Die linke Taste wird als Tastennummer \$74 gemeldet, die rechte als \$75.

Die Bits 3-7 müssen immer = 0 sein.

\$08

Es soll künftig die relative Maus-Position gemeldet werden. Dieser Befehl veranlaßt den Tastaturprozessor, automatisch die relative Position (die Distanz zur vorigen Position) zu melden, wenn eine Bewegung der Maus erfolgt ist. Eine Bewegung gilt dann als gegeben, wenn die Anzahl der Taktscheibenimpulse eine einstellbare Schwelle erreicht hat, die Maus also um ein bestimmtes Minimum bewegt wurde. Siehe auch \$0B. Ein relatives Maus-Paket sieht folgendermaßen aus:

- | | |
|--------|---|
| 1 Byte | Header im Bereich \$F8-\$FB. Die beiden untersten Bits des Headers geben den Zustand der beiden Maus-Tasten wieder. |
| 1 Byte | Relative X-Position (vorzeichenbehaftet!) |
| 1 Byte | Relative Y-Position (vorzeichenbehaftet!) |

Wenn sich zwischen zwei Paketen die relative Position einer Achse so stark verändert hat, daß die Distanz in einem Byte nicht mehr darstellbar wäre, wird automatisch ein weiteres Paket erzeugt, welches den Rest überträgt.

\$09

Es soll künftig die absolute Maus-Position gemeldet werden. Mit diesem Befehl werden auch die Koordinatenmaxima eingestellt, die höchstens zu melden sind. Gleichzeitig werden die internen Koordinatenzeiger auf Null gesetzt. Folgende Parameter sind verlangt:

- | | |
|--------|-----------------------|
| 1 Wort | Maximale X-Koordinate |
| 1 Wort | Maximale Y-Koordinate |

Mausbewegungen unter den Nullpunkt oder über die Maxima werden nicht gemeldet.

\$0A

Mit diesem Befehl ist es möglich, anstelle der Koordinaten die Tastennummern der Cursor-Tasten zu erhalten. Dem Betriebssystem erscheint dann eine Mausbewegung derart, als ob die entsprechenden Cursor-Tasten gedrückt worden wären. An Parametern sind notwendig:

- | | |
|--------|---|
| 1 Byte | Anzahl der Taktimpulse (X), nach denen jeweils einmal die Tastennummer für Cursor links (bzw. rechts) abgesetzt wird. |
| 1 Byte | Anzahl der Taktimpulse (Y), nach denen jeweils einmal die Tastennummer für Cursor auf (oder ab) abgesetzt wird. |

\$0B

Hiermit wird die Trigger-Schwelle gesetzt, oberhalb der Bewegungen erstmals gemeldet werden. Es verstreicht also eine bestimmte Anzahl Taktimpulse, bevor ein Paket abgesetzt wird. Das funktioniert nur in der relativen Betriebsart. Die Parameter sind:

1 Byte	Schwelle in X-Richtung
1 Byte	Schwelle in Y-Richtung

\$0C

Maus skalieren. Hiermit wird bestimmt, nach wieviel Taktimpulsen die Koordinatenzähler um 1 verändert werden. Dieser Befehl ist nur im absoluten Modus gültig. Folgende Parameter werden verlangt:

1 Byte	X-Skalierung
1 Byte	Y-Skalierung

\$0D

Absolute Mausposition abfragen. Parameter werden nicht verlangt, jedoch wird ein Paket folgenden Formats gesendet:

1 Byte	Header = \$F7
1 Byte	Tastenstatus
Bit 0 = 1:	Rechte Taste ist seit der letzten Abfrage gedrückt worden.
Bit 1 = 1:	Rechte Taste ist seit der letzten Abfrage nicht gedrückt worden.
Bit 2 = 1:	Linke Taste ist seit der letzten Abfrage gedrückt worden.
Bit 3 = 1:	Linke Taste ist seit der letzten Abfrage nicht gedrückt worden.

Aus dieser eigenartigen Zuordnung kann man schließen, daß eine Taste ihren Zustand seit der letzten Abfrage (evtl. mehrfach) verändert hat, wenn die beiden zugehörigen Bits =0 sind.

1 Wort	Absolute X-Koordinate
1 Wort	Absolute Y-Koordinate

\$0E

Interne Koordinatenzähler setzen. Es werden folgende Parameter verlangt:

1 Byte	=0 als Füllbyte
1 Wort	X-Koordinate
1 Wort	Y-Koordinate

\$0F

Der Ursprung für die Y-Koordinaten ist unten (dem Anwender am nächsten).

\$10

Der Ursprung für die Y-Koordinaten ist oben.

\$11

Die Datenübertragung zum Hauptrechner wird wieder aufgenommen (siehe \$13). Auch jeder andere Befehl (außer \$13) bringt den Transfer wieder in Gang.

\$12

Maus abschalten. Jeder Maus-Modus-Befehl (\$08, \$09, \$0A) schaltet die Maus wieder ein. Ist die Maus im Modus \$0A, hat dieser Befehl keine Auswirkung.

\$13

Datenübertragung zum Hauptrechner anhalten.



Soweit es der kleine Puffer des 6301 erlaubt, werden Mausbewegungen oder Tastendrucke noch gespeichert. Aktionen über die Pufferkapazität hinaus gehen verloren.

\$14

Jede Joystick-Bewegung wird automatisch gemeldet. Die gesendeten Pakete haben folgendes Format:

1 Byte	Header = \$FE oder \$FF für Joystick 0/1.
1 Byte	Bits 0-3 für die Stellung (jede Richtung ein Bit), Bit 7 für den Knopf.

\$15

Beendet den automatischen Meldemodus für den Joystick. Bei Bedarf muß ein Paket mit \$16 angefordert werden.

\$16

Joystick-Abfrage. Nach diesem Befehl sendet die Tastatur ein Paket wie oben beschrieben.

\$17

Joystick-Dauermeldung. Ein Parameter wird verlangt.

1 Byte Zeit zwischen zwei Meldungen in 1/100 sec.

Ab jetzt werden ständig (so lange kein anderer Modus eingestellt wird) Pakete folgenden Formats gesendet:

1 Byte Bit 0 für den Knopf von Joystick 1, Bit 1 für den von Joystick 0.

1 Byte Bits 0-3 für die Stellung von Joystick 1, Bits 4-7 für die Stellung des anderen Joystick.



Der Abtastabstand sollte nicht kürzer gewählt werden als der Übertragungskanal benötigt, die beiden Bytes des Paketes abzusetzen.

\$18

Feuerknopf-Dauermeldung. Es wird ständig der Zustand des Knopfes von Joystick 1(!) abgefragt und das Ergebnis, zu einem Byte zusammengepackt, gemeldet. Das heißt, daß in einem Melde-Byte acht Abfragen enthalten sind, wobei Bit 7 als zeitlich am ehesten eingetreten zu betrachten ist. Der Tastatur-rechner ermittelt die Zeit, die zwischen der Abholung der Bytes durch den Hauptrechner liegt. Diese Zeit wird in acht gleiche Intervalle eingeteilt, in denen der Knopf abgefragt wird. Deshalb sollte die Abholung so gleichmäßig wie möglich erfolgen. Dieser Modus bleibt bis zur Übermittlung eines anderen Befehls erhalten.

\$19

Cursor-Tasten-Simulationsmodus für Joystick 0(!). Die jeweilige Stellung des Joystick wird dem Hauptrechner so gemeldet, als ob die entsprechenden Cursor-Tasten (mehrfach) gedrückt worden wären. Um bei den folgenden Parametern nicht immer nahezu dasselbe erklären zu müssen, hier das Wesentli-

che: Alle Zeiten werden in 1/10 sec verstanden. R gibt die Zeit an, bis zu deren Erreichen Cursor-Klicks in Abständen von T gesendet werden. Danach beträgt der Abstand V. Wenn R = 0, ist nur V für den Abstand zuständig. Der Mechanismus tritt natürlich nur dann in Aktion, wenn der Joystick länger als T bzw. R in derselben Stellung ist.

1 Byte: RX
1 Byte: RY
1 Byte: TX
1 Byte: TY
1 Byte: VX
1 Byte: VY

\$1A

Joysticks abschalten. Jeder andere Joystick-Befehl setzt die Abfrage wieder in Gang.

\$1B

Uhrzeit setzen. Mit diesem Befehl wird die interne Echtzeituhr des Tastaturprozessors gestellt. Die zu übergebenden Werte sind sämtlich als gepackte BCD-Zahl zu verstehen, d.h. eine Ziffer (0-9) pro Halbbyte, was eine zweistellige Dezimalzahl pro Byte ergibt. Folgende Parameter sind notwendig:

1 Byte: Jahreszahl zweistellig (85, 86 usw.).
1 Byte: Monat zweistellig (12, 01 usw.).
1 Byte: Tag zweistellig (31, 01, 02 usw.).
1 Byte: Stunde zweistellig.
1 Byte: Minute zweistellig.
1 Byte: Sekunde zweistellig.

Jedes Halb-Byte, welches keine gültige BCD-Ziffer enthält (z.B. F), wird ignoriert. Auf diese Weise ist es möglich, auch nur Teile von Datum oder Uhrzeit zu ändern.

\$1C

Uhrzeit abfragen. Aufgrund dieses Befehls liefert die Tastatur ein Paket zurück, das im Aufbau dem oben beschriebenen Format entspricht. Lediglich ein Header ist natürlich noch vorangesetzt. Er hat den Wert \$FC.

\$20

Speicher laden. Der interne Speicher des Tastaturprozessors (natürlich ist nur der RAM im Bereich \$80 bis \$FF sinnvoll) kann mit diesem Befehl beschrieben werden. Uns ist unklar, welchen Sinn das ergeben soll, denn nach den bisherigen Untersuchungen (wir haben das Betriebssystem des 6301 bereits disassembliert) ist kein freier Raum verfügbar, den man nach Belieben manipulieren könnte. Vielleicht lassen sich aber auf diesem Wege bestimmte Parameter (nach genauer Kenntnis des Aufbaues) beeinflussen, die auf "legalem" Wege nicht zugänglich sind. Hier nun die Parameter:

- | | |
|--------|---|
| 1 Wort | gewünschte Startadresse |
| 1 Byte | Anzahl der Bytes (max. 128) Datenbytes. (entsprechend der Anzahl) |

Der Abstand, in dem die Daten-Bytes übergeben werden, muß kleiner als 20 msec sein.

\$21

Speicher lesen. Dieser Befehl bildet das Gegenstück zu \$20. An Parametern werden verlangt:

- | | |
|--------|--------------------------------------|
| 1 Wort | Adresse, ab der gelesen werden soll. |
|--------|--------------------------------------|

Es wird ein Paket folgenden Formates zurückgesendet:

- | | |
|---------|---|
| 1 Byte | Header 1 = \$F6. Es handelt sich hierbei um den sogenannten Status-Header, der allen Paketen vorangestellt wird, die irgendwelche Betriebszustände des Tastaturprozessors beinhalten. Zu den allgemeinen Status-Meldungen kommen wir jedoch noch. |
| 1 Byte | Header 2 = \$20 als Kennzeichen, daß dieses Status-Paket den Speicherinhalt transportiert. |
| 6 Bytes | Speicherinhalt, beginnend mit der im Befehl angegebenen Adresse. |

\$22

Routine ausführen. Mit diesem Befehl können Sie ein Unterprogramm im 6301 ausführen. Natürlich müssen Sie genau wissen, was es macht und wo es steht, sofern Sie es nicht selbst mittels \$20 in das RAM transferiert haben (wenn Sie

dort überhaupt noch ein freies Plätzchen gefunden haben). Als Parameter wird nur verlangt:

1 Wort Startadresse

Status-Meldungen

Sie können jederzeit die Betriebsparameter der Tastatur abfragen, indem Sie einfach zu dem Befehls-Byte, mit dem Sie die Betriebsart (deren Parameter Sie jetzt wissen möchten) einstellen würden, \$80 addieren. Sie erhalten dann ein Status-Paket zurück (Header = \$F6), dessen Format exakt den Parametern entspricht, die auch zur Einstellung der Betriebsart notwendig gewesen wären. Ein Beispiel macht das sicher deutlicher: Sie wollen wissen, wie die Maus skaliert ist. Also senden Sie als Kommando den Wert \$8C (da ja \$0C die Skalierung setzt). Sie erhalten folgendes zurück:

1 Byte Status-Header = \$F6
1 Byte X-Skalierung
1 Byte Y-Skalierung

also dasselbe Format, das beim Befehl \$0C notwendig gewesen wäre. Bei Befehlen, die keine Parameter verlangen, erhalten Sie den auslösenden Befehl als solchen zurück. Sie möchten z.B. wissen, in welcher Betriebsart sich der Joystick befindet (\$14 oder \$15). Sie schicken also den Wert \$94 (oder \$95, das spielt keine Rolle). Als Status-Paket erhalten Sie außer dem Header entweder \$14 oder \$15 zurück, je nachdem, in welcher Betriebsart sich der Joystick-Handler befand. Erlaubte Status-Abfragen sind:

\$87, \$88, \$89, \$8A, \$8B, \$8C, \$8F, \$90, \$92, \$94, \$95, \$99 und \$9A.

Hier schließlich noch das bereits angekündigte Programm, mit dem wir das ROM des 6301 ausgelesen und auf Drucker ausgegeben haben. Hieran sehen Sie auch, wie Sie überhaupt an die Status-Pakete der Tastatur herankommen. Vom Betriebssystem des 68000 werden diese nämlich normalerweise weggeworfen. Über die verwendeten GEMDOS- und XBIOS-Aufrufe informieren Sie sich bitte im Anhang.

prt	equ	0
chout	equ	3
gemdos	equ	1
bios	equ	13
xbios	equ	14
stvec	equ	12
rdm	equ	\$21
wrkbd	equ	25
kbvec	equ	34

```

term      equ      0
start:
move.w    #kbdvec, -(a7)
trap      #xbios
addq.l    #2, a7
move.l    d0, a0
lea       keyin, a1
move.l    d0, savea
move.l    stvec(a0), save
move.l    a1, stvec(a0)
move.w    #$f000, d4          Startadresse
loop:
move.w    d4, tbuf+1          aktuelle Adresse
bsr       keyout
wait:
tst.b     rbuf
beq       wait
moveq.w   #5, d6
bsr       bufout
addq.w    #6, d4              Endadresse ?
bmi       loop               nein >
bra       exit
bufout:
lea       rbuf+2, a4
bytout:
move.b    (a4)+, d0
bsr       hexout
dbra      d6, bytout
rts
hexout:
movea.w   d0, a1
lsr.b     #4, d0
andi.w    #15, d0
lea       table, a3
move.b    0(a3, d0), d2
lsl.w     #8, d2
move.w    a1, d0
andi.w    #15, d0
move.b    0(a3, d0), d2
move.w    d2, d0
move.w    d2, -(a7)
lsr.w     #8, d0
bsr       chrout
move.w    (a7)+, d0
bsr       chrout
move.b    #" ", d0
chrout:
move.w    d0, -(a7)
move.w    #prt, -(a7)
move.w    #chout, -(a7)
trap      #bios
addq.l    #6, a7
rts

```

```

exit:
movea    savea,a0
move.l   save,stvec(a0)
move.w   #term,-(a7)
trap     #gemdos
keyout:
clr.b    rbuf
pea      tbuf
move.w   #2,-(a7)
move.w   #wrkbd,-(a7)
trap     #xbios
addq.l   #8,a7
rts
keyin:
moveq    #7,d0
lea      rbuf,a1
repin:
move.b   (a0)+,(a1)+
dbra     d0,repin
rts
table:
dc.b     "0123456789ABCDEF"
rbuf     ds.b      8
save     ds.l      1
savea    ds.l      1
dummy    ds.b      1
tbuf     dc.b      rdm
ds.b     2
.end
    
```

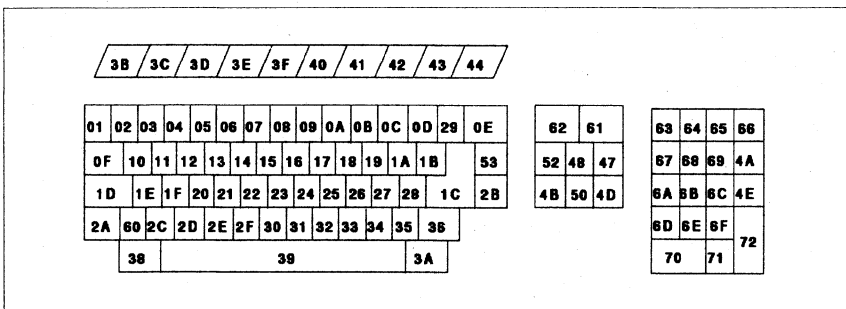


Abb. 20.5: Die Scan-Codes der Tastatur (hexadezimal)

20.2 Die Video-Buchse

Ohne sie geht nichts. Sie würden im wahrsten Sinne des Wortes im Dunkeln tappen. Auffallend sind die vielen Pins der Verbindung. Sicher werden zum Anschluß eines RGB-Monitors mehr Leitungen benötigt als für einen monochromen Bildschirm, aber sieben täten es auch. So hat es denn mit den übrigen Leitungen auch seine besondere Bewandnis. Auf Abbildung 20.6 finden Sie ein Blockschaltbild, auf dem Sie sehen können, wie der Video-Anschluß ins System eingebunden ist. Die Numerierung der Pins ist in der Abbildung auf der nächsten Seite so angegeben, wie Sie sie sehen, wenn Sie von außen in die Buchse hineinschauen. Hier die Pinbelegung:

- 1 (AUDIO OUT) Dieser Anschluß stammt von dem Verstärker, der hinter die Ausgänge des Sound-Chip geschaltet ist. Hier läßt sich notfalls auch ein hochohmiger (600 Ohm) Kopfhörer anschließen, falls Sie keinen Original-Monitor verwenden.
- 2 (COMP SYNC) Ist die Zusammenfassung von 9+12. Bei ATARI 520 ST nicht belegt!
- 3 (GPO) General Purpose Output. Dies ist ein Bit zu Ihrer freien Verfügung. Die Leitung führt TTL-Pegel und stammt von I/O-Port A Bit 6 des Sound-Chip.
- 4 (MONOCHROME DETECT) Ist diese Leitung, die zum I7-Eingang des MFP 68901 führt, =low, geht der Rechner in den hochauflösenden Monochrome-Modus. Wechselt während des Betriebes der Zustand der Leitung, wird ein Kaltstart ausgelöst.
- 5 (AUDIO IN) Führt zum Eingang des unter 1 beschriebenen Verstärkers und wird dort mit dem Ausgang des Sound-Chip gemischt.
- 6 (GREEN) Ist der analoge Grün-Ausgang des Video-Shifters.
- 7 (RED) Rot-Ausgang.
- 8 (+12V) Steuerspannung für Farbfernseher mit Video-Anschluß. Bei ATARI 520 ST = GROUND!
- 9 (HORIZONTAL SYNC) Ist für den horizontalen Strahlrücklauf des Monitors verantwortlich.
- 10 (BLUE) Ist der analoge Blau-Ausgang des Video-Shifters.

- 11 (MONOCHROME) Versorgt einen einfarbigen Monitor mit dem Helligkeitssignal.
- 12 (VERTICAL SYNC) Sorgt für den Strahlrücklauf am Bildende.
- 13 (GROUND)

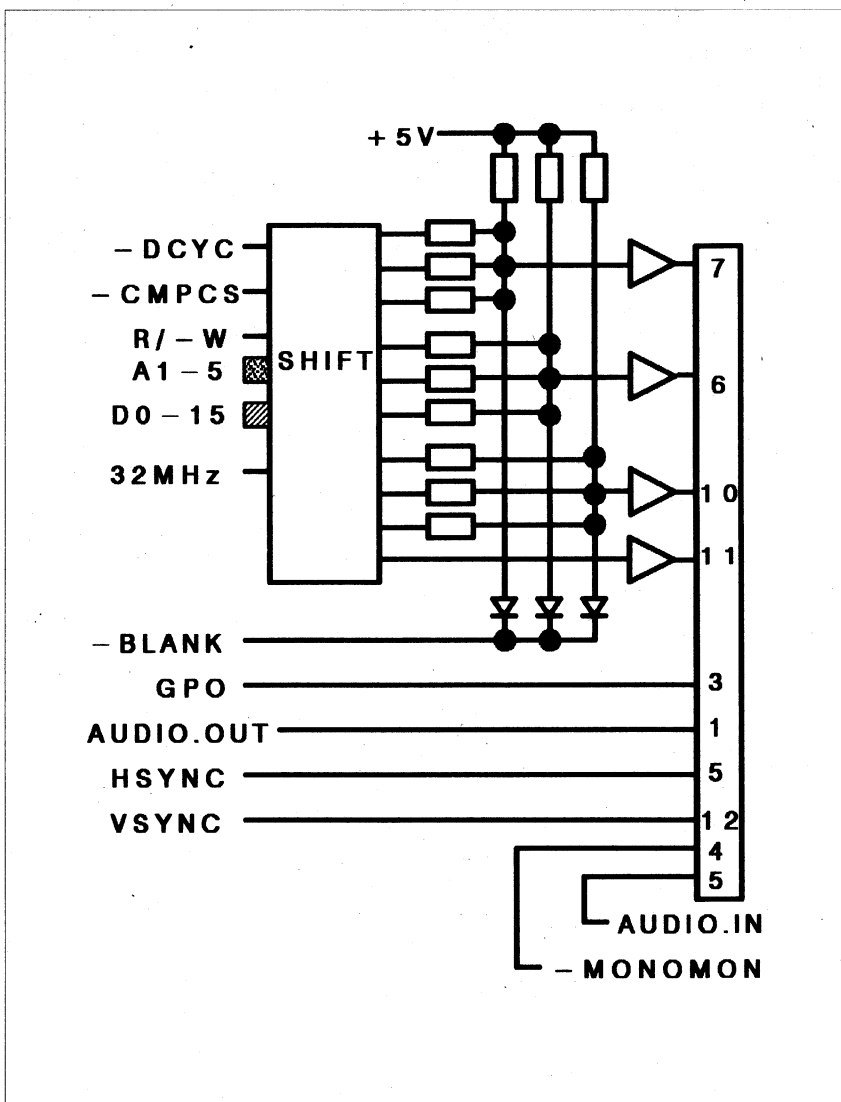


Abb. 20.6: Die Beschaltung der Monitorbuchse

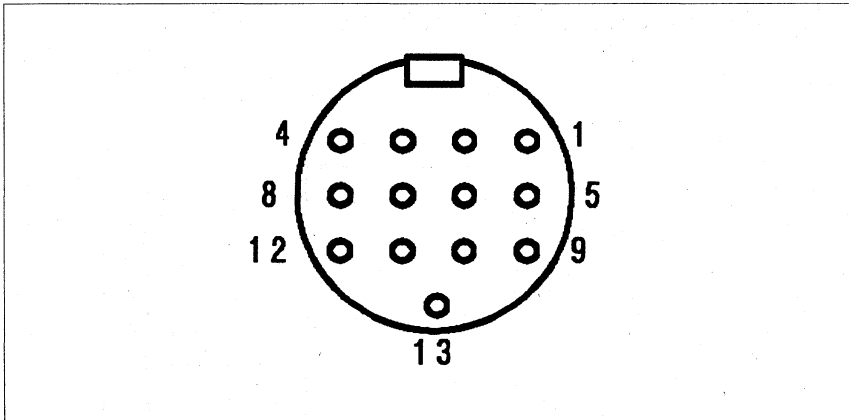


Abb. 20.7: Die Monitorbuchse

20.3 Der Centronics-Anschluß

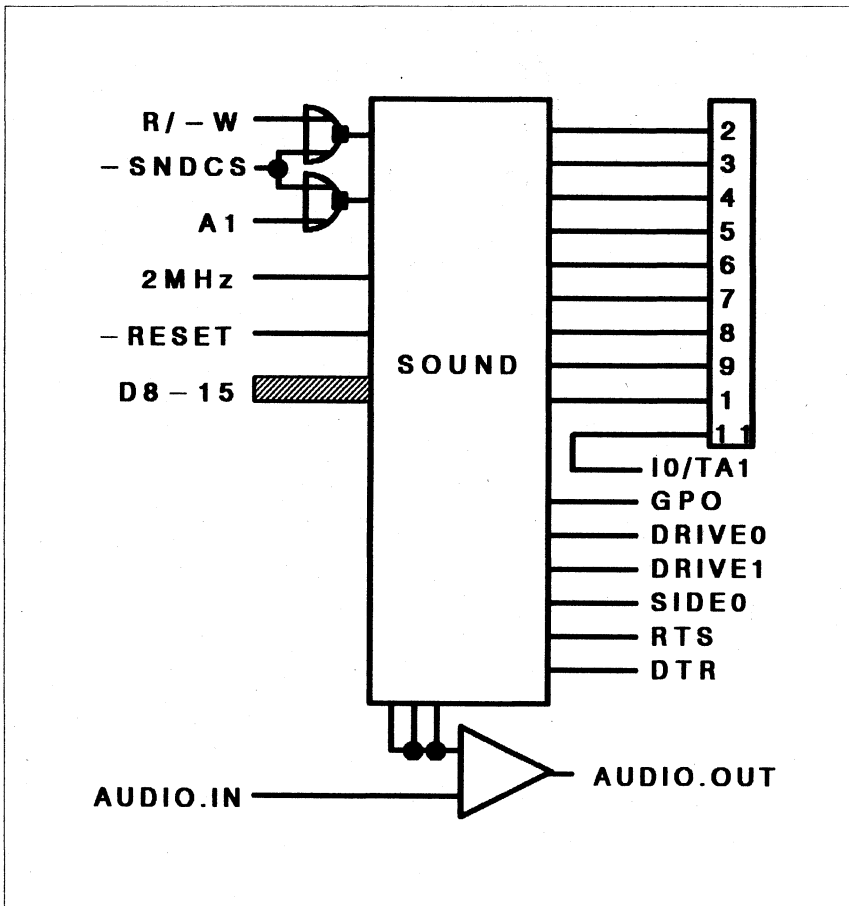


Abb. 20.8: Die Schaltung der Centronics-Schnittstelle

Nachdem die Centronics-Schnittstelle bereits im Teil A dieses Buches beschrieben wurde, soll uns hier nur noch die Einbindung in das System interessieren. Die ist etwas ungewöhnlich. Man hat nämlich zur Bedienung der Datenleitungen und des Strobes den Universal-Port des Sound-Chip herangezogen.

20.4 Die RS232-Schnittstelle

Auch diese Schnittstelle wurde bereits in Teil A beschrieben. Die Abbildung 20.9 zeigt die Einbindung ins Gesamtsystem. Normalerweise haben Sie mit der expliziten Handhabung dieser Schnittstelle nichts zu tun. Man läßt es vom Betriebssystem erledigen.

Dennoch fällt auf, daß hier nicht, wie sonst üblich, die Steuerung der Schnittstelle einem speziellen IC (UART) überlassen wird, sondern die Leitungen mehr oder weniger "zu Fuß" bedient werden. Sicher werden die Daten nicht "persönlich" bitweise verwurstet.

Dazu dient das Schieberegister des MFP. Die Handshake-Leitungen jedoch kommen aus den verschiedensten Ecken:

- 4 (RTS) Sendeanforderung stammt von I/O-Port A Bit 3 des Sound-Chip und wird immer dann high, wenn der Rechner in der Lage ist, ein Byte zu empfangen. Bei Atari wird das Signal nach Empfang eines Byte zunächst low gesetzt, bis das eben empfangene Byte verarbeitet ist.
- 5 (CTS) Die Empfangsbereitschaft eines angeschlossenen Gerätes wird an dem Interrupt-Eingang I2 des MFP abgefragt. Dieses Signal wird im Betriebssystem des Rechners (ab MEGA ST nicht mehr) falsch behandelt. So ist nur der Anschluß von Geräten möglich, die nach jedem empfangenen Byte (wie der 520 ST mit RTS) mit der Leitung "klappern". Sie geht nämlich auf den Eingang I2 des MFP und wird dort unglücklicherweise nur auf die Signalfanke untersucht.
- 8 (DCD) Trägersignal erkannt. Diese Leitung geht zum Interrupt-Eingang I1 des MFP.
- 20 (DTR) Gerät bereit. Sie stammt von I/O-Port A Bit 4 des Sound-Chip.
- 22 (RI) Dies ist eine recht wichtige Unterbrechung an I6 des MFP.

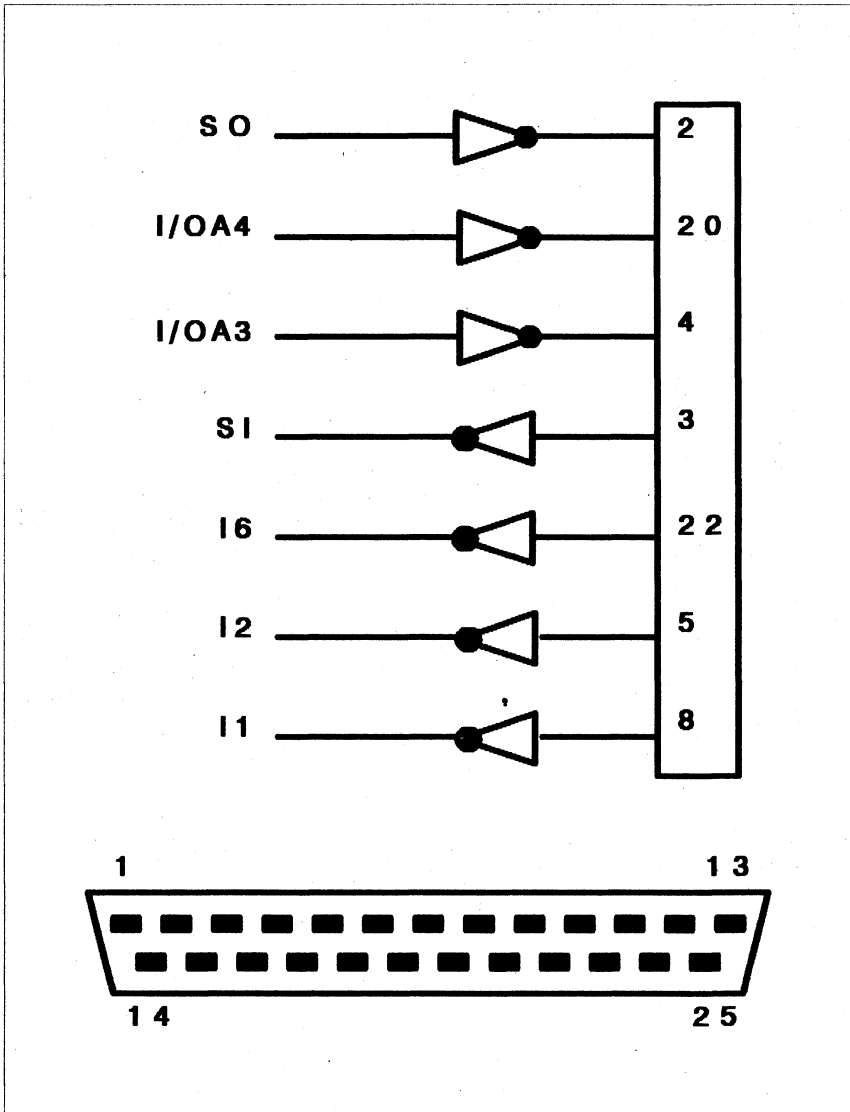


Abb. 20.9: Die Schaltung der RS232

20.5 Die Midi-Anschlüsse

Rein physikalisch gründet sich dieses Interface auf der Teletype-Basis, also dem Stromschleifenprinzip, welches sich durch eine hohe Störsicherheit auszeichnet. Die Übertragungsrate beträgt 31250 Baud, also ebenso viele Bits/s. Das Datenformat ist mit einem Start-Bit, acht Daten-Bits und einem Stopp-Bit festgelegt. So kann denn zur Steuerung auch ein IC dienen, welches sonst für RS232-Zwecke herangezogen wird. Die Einbindung ins System sehen Sie in Abbildung 20.10. Logisch ist MIDI ein Vielkanal-System, d.h. es können von einem Master 16 Geräte bedient werden oder ein Gerät mit 16 Stimmen, vereinfacht ausgedrückt. Diese Geräte "hängen" alle an derselben Leitung. Zur Identifikation, welches Gerät oder welche Stimme gemeint ist, wird jedem Datenpaket die Kanalnummer vorangestellt. Das Gerät, welches diese Nummer als seine eigene erkennt, wird die gewünschte Aktion ausführen.

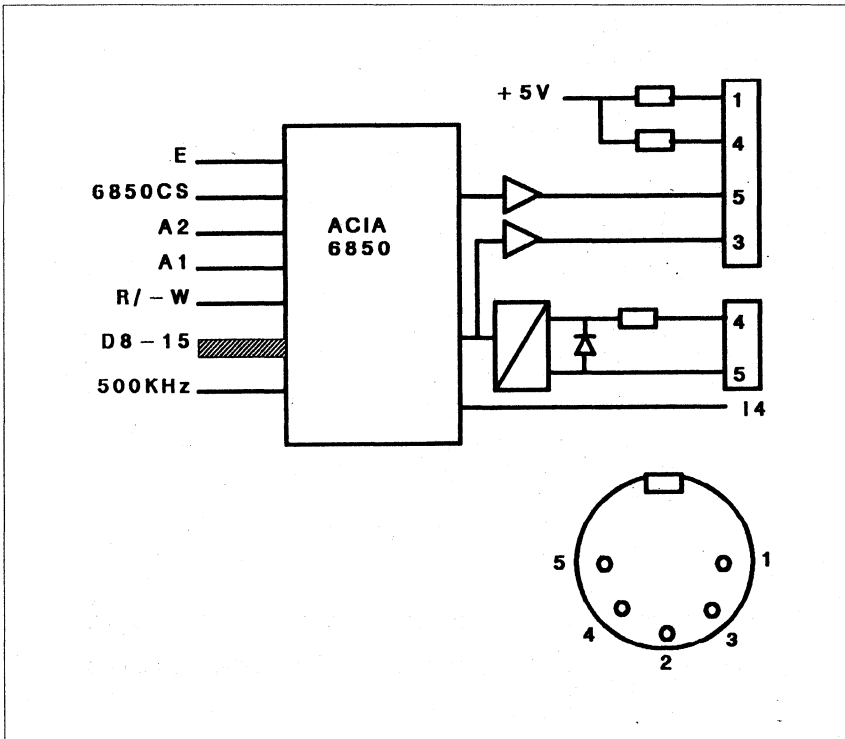


Abb. 20.10: Die MIDI-Schnittstelle

Sogar vom XBIOS werden diese Schnittstellen unterstützt, so daß Sie sich mit dem "Kleinkram" nicht herumprügeln müssen. Der Schleifenstrom geht über die Pins 4 und 5, und zwar bei MIDI-OUT bei 4 hinaus (+) und bei 5 herein, wenn ein Gerät angeschlossen ist. Bei MIDI-IN ist es umgekehrt, denn der Strom fließt bei 4 herein und bei 5 wieder hinaus. Dabei durchläuft er einen sogenannten Optokoppler, mit dem der Rechner galvanisch vom Sender getrennt wird. Die Empfangsdaten werden nach MIDI-OUT wieder durchgeschleift (Pins 1 und 3), womit auch die Funktion MIDI-THRU, wenn auch nicht ganz normgerecht, implementiert wäre.

20.6 Der Modulschacht

Der Modulschacht dient ausschließlich der Aufnahme sogenanntes ROM-Cartridges. Es lassen sich auf diesem Wege maximal 128 K (Bytes) im Adreßraum \$FA0000 bis \$FBFFFF ansprechen.

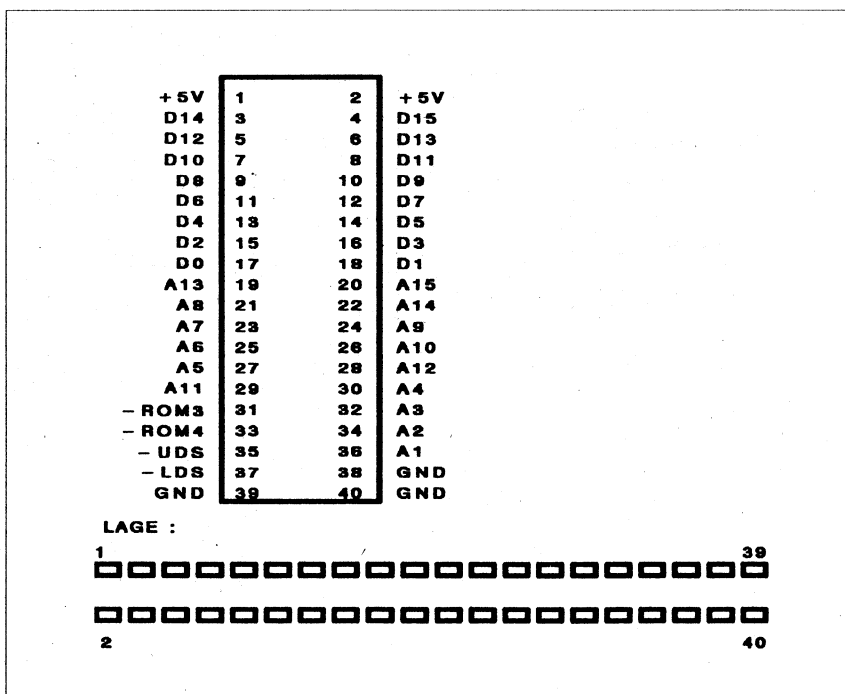


Abb. 20.11: Der Modulschacht

Warum wir die Ausschließlichkeit des Lesezugriffs so sehr betonen, hat folgenden Grund: Wir hätten es für praktisch gehalten, eine Cartridge mit RAM auszurüsten und in diese dann nach dem Systemstart ständig benötigte Programme hineinzuladen, die auch nach einem Reset noch erhalten gewesen wären. Um das zu probieren, haben wir das Signal R/-W nach außen gelegt. Die Erfahrung lehrte jedoch, daß ein Schreibzugriff auf die o.g. Adressen einen Bus-Error erzeugt. Dafür sorgt nämlich der GLUE. Wie Sie sehen, hat man bei Atari nichts dem Zufall überlassen.

20.6.1 ROM-Cartridges

In diesem Kapitel wollen wir Ihnen zeigen, wie ein Programm im ROM beschaffen sein muß, damit es auch vom Betriebssystem erkannt und geladen werden kann.

Das wichtigste ist das erste Langwort im ROM, welches eine sogenannte Magic Number enthalten muß, also eine Kennzahl. Diese wird beim Systemstart schon recht frühzeitig abgefragt, um festzustellen, ob es sich um eine reine Programm-Cartridge handelt oder um eine Diagnose-Cartridge. Die erstere muß die Kennzahl \$ABCDE42 enthalten, die letztere die Kennzahl \$FA52255F.

Mit den Diagnose-Cartridges wollen wir uns nicht weiter beschäftigen. Es soll Ihnen die Tatsache genügen, daß das Betriebssystem unmittelbar nach der Prüfung an die Adresse \$FA0004 springt, ohne zuvor das GEMDOS zu initialisieren. Ihnen steht also praktisch keinerlei Systemumgebung zur Verfügung.

Der interessantere Fall sind die Programm-Cartridges. In einem solchen ROM-Modul dürfen Sie auch mehrere Programme unterbringen. Damit das Betriebssystem diese auch auseinanderhalten kann, muß jedes Programm einen Vorspann besitzen, den sogenannten Application Header. Der erste muß gleich hinter der Magic Number beginnen, also an der Adresse \$FA0004, und ist folgendermaßen aufgebaut:

- 1 (Langwort) Adresse des nächsten Headers, falls sich mehrere Programme in dem Modul befinden. Der Header des letzten (oder einzigen) Programmes muß an dieser Stelle \$00000000 enthalten.
- 1 (Langwort) Initialisierungs-Code. Diesem entnimmt GEMDOS Informationen darüber, wie mit dem Programm (zunächst) zu verfahren ist. Im wesentlichen besteht dieses Langwort aus einer Adresse, die auf eine Initialisierungsroutine (falls vorhanden) zeigt. Zu

welchem Zeitpunkt diese Routine angesprungen wird, bestimmt das höchstwertige Byte in diesem Langwort. Dieses ist wie folgt verschlüsselt:

- Bit 0: Die Routine wird ausgeführt, bevor die Interrupt-Vektoren, Video-RAM usw. initialisiert werden.
- Bit 1: Die Routine wird ausgeführt, bevor GEMDOS initialisiert wird.
- Bit 3: Die Routine wird ausgeführt, bevor GEMDOS geladen wird. Achtung: Diese Funktion ist auf Rechnern, die das GEMDOS auf ROM haben, nicht verfügbar!
- Bit 5: Kennzeichen dafür, daß es sich bei dem Programm um ein Accessory handelt.
- Bit 6: Kennzeichen, daß das Programm vom Typ .TOS ist, also nicht die Systemumgebung von GEM benötigt.
- Bit 7: Kennzeichen dafür, daß das Programm vom Typ .TTP ist, also beim Start Parameter verlangt.
- 1 (Langwort) Startadresse des Programms, also wo es beim Anklicken loslaufen soll.
- 1 (Wort) Uhrzeit im DOS-Format. Für den Ablauf ohne Bedeutung.
- 1 (Wort) Datum im DOS-Format, sonst wie oben.
- 1 (Langwort) Länge des Programmes in Bytes. Für den Ablauf ohne Bedeutung.
- String Programmname in Klartext. Der Programmname unterliegt den üblichen Konventionen, also bis zu acht Zeichen vor dem Punkt und drei dahinter. Achtung: Der String muß unbedingt mit \$00 abgeschlossen sein.

So, das war's. Übrigens: Oben haben wir das Wort "Anklicken" fallenlassen. Was wir nämlich bisher verschwiegen haben, ist der Umstand, daß eine Programm-Cartridge ein eigenes Icon, so wie die Laufwerke, erhält. Klicken Sie dieses an, erscheinen die in dem Modul enthaltenen Programme auf dem Bildschirm, wonach Sie sie wie gewohnt starten können.

20.7 Die Floppy Disk-Schnittstelle

Zunächst einmal fällt die Schnittstelle für die Floppy-Laufwerke durch die ungewöhnliche Steckverbindung auf. Der 14-polige DIN-Stecker sieht aus wie ein etwas zu groß geratener sogenannter Diodenstecker. An ihm sind alle für den Betrieb von zwei Laufwerken erforderlichen Signale vorhanden.

Die meisten Signale sind Ihnen bereits von der Beschreibung des Floppy Controllers 1772 bekannt, da neun der vorhandenen Anschlüsse direkt oder über einen Buffer mit dem Controller verbunden sind.

Nur die Auswahl-Signale Drive Select 1 und Drive Select 2 sowie das Seitenauswahlsignal Side 0 Select werden nicht vom Floppy Controller, sondern vom Port A des Sound-Chip geliefert.

Belegung des Floppy-Steckers:

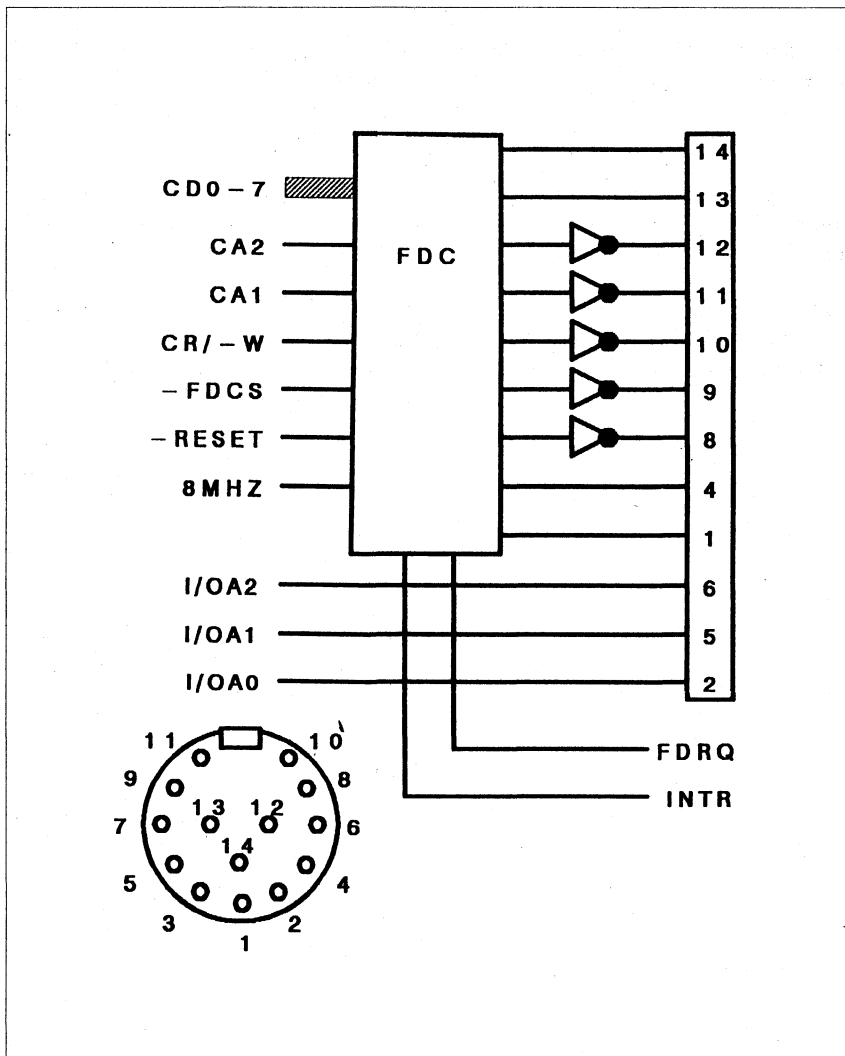


Abb. 20.12: Die Floppy-Schnittstelle

- 1 READ DATA
- 2 SIDE 0 SELECT
- 3 GND

- 4 INDEX
- 5 DRIVE 0 SELECT
- 6 DRIVE 1 SELECT
- 7 GND
- 8 MOTOR ON
- 9 DIRECTION IN
- 10 STEP
- 11 WRITE DATA
- 12 WRITE GATE
- 13 TRACK 00
- 14 WRITE PROTECT

20.8 Die DMA-Schnittstelle

Wohl jeder Atari-ST-Anwender hat schon einmal die Abkürzung DMA gehört oder gelesen und assoziiert mit diesem Begriff Geschwindigkeit, Festplatten oder Laserdrucker. Die beiden genannten Geräte sind ja die prominentesten `Benutzer` des DMA- oder Harddisk-Ports. Aber auch Netzwerk-Karten zur Verkopplung mehrerer STs sind in der letzten Zeit von Drittanbietern auf dem Markt erschienen, die sich des DMA-Ports bedienen. Was aber hat es nun mit der Geschwindigkeit und den weiteren Mysterien dieses 19-poligen Anschlusses des ST auf sich?

Leider war lange Zeit die verfügbare Information über diesen Anschluß nicht nur dürftig, sondern teilweise sogar falsch. Das bezieht sich sowohl auf die offizielle Atari-Dokumentation als auch auf die ersten Ausgaben des Intern. Doch in der Zwischenzeit haben sich einige der vermeintlichen Rätsel gelöst. Bevor wir jedoch auf den DMA-Bus und die Datenübertragung eingehen, wollen wir einige im weiteren häufig benutzte Begriffe festlegen und anschließend einen kurzen Blick auf die Belegung des DMA-Steckers (Abbildung 20.13) werfen.

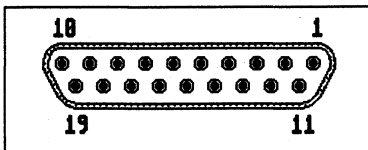


Abb. 20.13: Die Schaltung der DMA-Schnittstelle

In der von Atari benutzten Terminologie ist die Rede von INITIATOR und TARGET. Der Initiator ist immer der ST, da nur er eine Datenübertragung initiieren, also einleiten kann. Target bedeutet so viel wie Ziel und bezeichnet das angeschlossene Gerät. Maximal acht Targets lassen sich am DMA-Bus des ST anschließen. Zusätzlich gibt es noch die Devices. Jedes Target kann seinerseits acht Geräte ansteuern und kontrollieren. So könnte z.B. ein Festplattencontroller als Target bis zu acht angeschlossene Festplatten als Devices ansteuern. Damit ist eine maximale Grenze von 64 externen Geräten am ST vorgegeben.

Wir werden den Initiator weiter als ST oder Computer bezeichnen, die Targets nennen wir Controller, und die Devices sind bei uns Geräte. Bitte verwechseln Sie später nicht die Begriffe Controller und DMA-Controller. Letzterer ist der im ST eingebaute Chip, der für die Steuerung der Datenübertragung sowohl auf dem DMA-Bus als auch intern auf dem Daten-Bus des Prozessors zuständig ist.

- | | | |
|-----|-------|--|
| 1-8 | D0-D7 | Bidirektionale Datenleitung, über diese acht Anschlüsse werden die Daten von und zum ST übertragen. |
| 9 | CS | Chip Select, low-aktiv. Diese Leitung wird vom Rechner immer dann aktiviert, wenn entweder Befehle zum Controller gesendet oder Status-Bytes von dort gelesen werden. Während der DMA-Übertragung bleibt das Signal high. |
| 10 | IRQ | Interrupt Request, low-aktiv. Dieses Signal wird vom Controller bedient und zeigt an, daß eine Aktion, z.B. ein DMA-Transfer, beendet ist. |
| 11 | GND | Masseanschluß |
| 12 | RST | Reset, low-aktiv. Wird zum Rücksetzen der angeschlossenen Controller verwendet. |
| 13 | GND | wie Pin 11 |
| 14 | ACK | Acknowledge, low-aktiv. Dieses Signal hat nur während des DMA-Transfers eine Bedeutung. Je nach gewählter Datenrichtung hat dieses Signal unterschiedliche Bedeutung. Beim Senden von Daten an einen Controller zeigt ACK an, daß ein gültiges Daten-Byte an den Pins 1 bis 8 anliegt, beim Lesen von Daten wird die Übernahme eines Bytes durch einen Low-Impuls angezeigt. |
| 15 | GND | Wie Pin 11 |
| 16 | A1 | Address 1. Dieses vom ST gesteuerte Signal zeigt den angeschlossenen Controllern, ob das erste der insgesamt sechs Kommando-Bytes übertragen wird. In diesem Fall ist A1 low. |
| 17 | GND | Wie Pin 11 |
| 18 | R/W | Auch diese Leitung wird vom ST gesteuert und hat nur bei der Übertragung der Kommando-Bytes oder des Status einer DMA-Übertragung Bedeutung. Bei Low-Pegel 'schreibt' der ST Daten an einen angeschlossenen Controller, bei High liest er ein Byte. |

- 19 DRQ Data Request, low-aktiv. Dieses Signal wird während des DMA-Transfers vom externen Controller gesteuert. Über dieses Signal zeigt er dem ST, daß er ein Byte vom ST empfangen hat oder selber ein Byte auf die Anschlüsse 1 bis 8 gelegt hat.

Wie man bereits aus der Beschreibung der Schnittstellenanschlüsse sehen kann, existieren zwei vollständig unterschiedliche Möglichkeiten der Datenübertragung zwischen dem ST und einem Controller. Wir unterscheiden zwischen der Kommandophase und der eigentlichen DMA-übertragung. Während die Datenübertragung der Kommandophase mit den Signalen CS und IRQ (R/W und A1 werden nicht für den unmittelbaren Datentransfer benötigt) kontrolliert wird, haben beim DMA-Transfer nur die Signale DRQ und ACK sowie R/W für die Datenrichtung eine Bedeutung. Wo aber liegt der Unterschied zwischen den beiden Phasen und wozu wird die Unterscheidung nötig?

Obwohl der DMA-Transfer der Schnittstelle ihren Namen gegeben hat, beginnen wir mit der Kommando-Phase, denn ohne sie käme es zu gar keinem DMA.

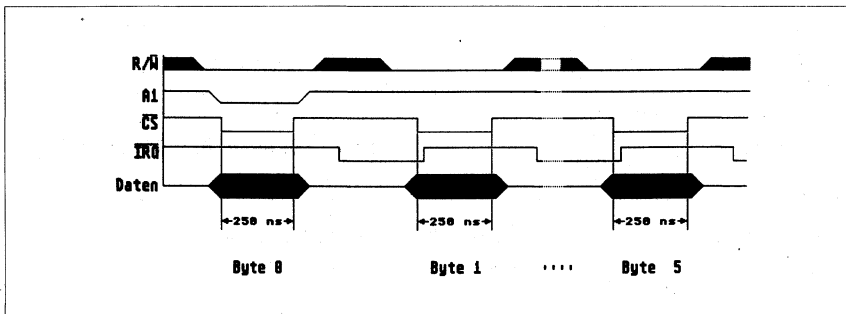


Abb. 20.14: Die Daten- und Kommandophase

Wie bereits erwähnt, werden bei der Kommando-Phase immer sechs Bytes übertragen. Der Aufbau eines Kommandos ist in Abbildung 20.14 dargestellt. Bei der Übertragung des ersten Bytes wird die Leitung A1 des DMA-Ports Low und signalisiert somit den Anfang des Kommandos. Wie in der Abbildung 20.15 gezeigt, wird im ersten Byte die Nummer des adressierten Gerätes in den Bits 5 bis 7 und das gewünschte Kommando in den Bits 0 bis 4 übertragen. Auch die R/W-Leitung geht während der Ausgabe auf Low, da der ST die Daten ausgibt. Unmittelbar, nachdem das Byte auf den Datenleitungen ausgegeben ist, wird der Anschluß CS für etwa 250 Nanosekunden Low und zeigt dem Controller an, daß er gültige Daten lesen kann. Hat er die Daten ge-

lesen, so signalisiert er dies seinerseits durch einen Low-Impuls auf der IRQ-Leitung. Nach dem gleichen Schema erfolgt jetzt die Übertragung der restlichen fünf Bytes. Hierbei muß jedoch die Leitung A1 High bleiben. Auch bei den folgenden Bytes stehen die Daten jeweils für 250 Nanosekunden zur Verfügung.

Die gesamte Übertragung erfolgt per Software über nur zwei Adressen im ST. Die Adresse 0xFF8604 stellt gewissermaßen das Datenregister für die Übertragung dar, und die Adresse \$FF8606 ist ein Steuerregister mit vielfältigen Möglichkeiten zur Datenübertragung. So kann hier zwischen Floppy-Controller und DMA-Port umgeschaltet werden. Der Pegel an der Leitung A1 wird hier beeinflußt, und die Datenrichtung (Pegel an R/W) kann mit einem Bit eingestellt werden. Beim DMA-Transfer ist es immer der Controller, der seine Bereitschaft zum Senden oder Empfangen von Daten mitteilen muß. Die Abbildungen 20.16 und 20.17 zeigen sowohl das Lesen als auch das Schreiben von Daten.

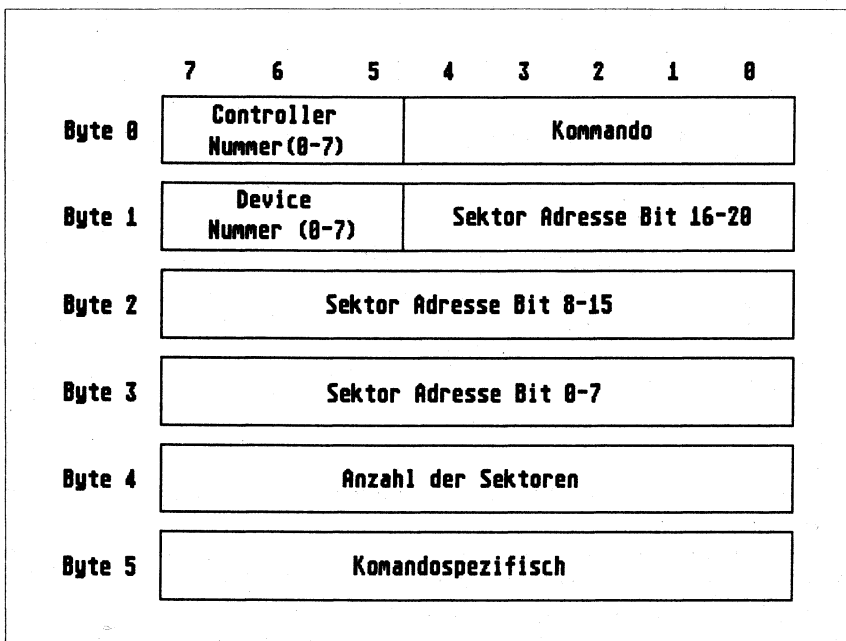


Abb. 20.15: Der Byte-Aufbau beim DMA-Transfer

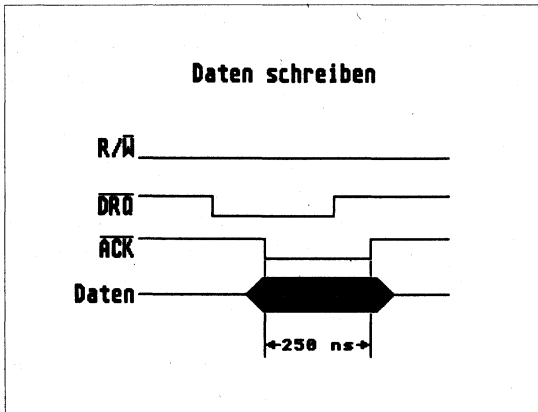


Abb. 20.16: Daten schreiben

Werden Daten vom ST ausgegeben, so geht zunächst einmal der R/W-Anschluß auf Low. Ist der Controller bereit und in der Lage, ein Byte zu lesen, so legt er seinerseits den Anschluß DRQ auf Low. Jetzt werden die Daten vom DMA-Controller-Chip des ST am Daten-Bus zur Verfügung gestellt und mit einem Low-Impuls von 250 Nanosekunden am ACK-Pin angezeigt. Mit der fallenden Flanke des ACK-Signals sollte der externe Controller das Datenbyte übernehmen und gleichzeitig die Übernahme dadurch quittieren, daß er den DRQ-Anschluß wieder auf High legt. Für die Übernahme des nächsten Bytes muß er DRQ wieder auf Low legen. Damit wird die Übertragung des nächsten Bytes eingeleitet.

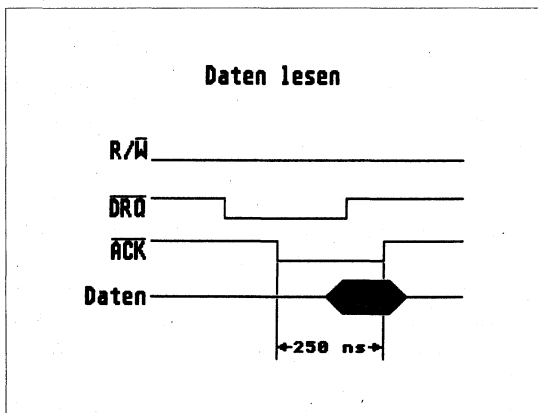


Abb. 20.17: Daten lesen

Liest der ST im DMA-Betrieb Daten, so verläuft die Übertragung ähnlich. Zunächst ist die R/W-Leitung High, da der ST lesen will. Hat der externe Controller Daten bereit, so legt er DRQ auf Masse. Der ST legt daraufhin das Signal ACK für 250 Nanosekunden auf Low. Dadurch wird das Datenbyte auf den Bus gelegt, und mit der steigenden Flanke des ACK übernimmt der ST das Byte. Ist das Byte vom Bus gelesen, so wird es nicht sofort in den Speicher des ST geschrieben. Statt dessen werden erst einmal 16 Bytes in einem von zwei identischen Zwischenspeichern des DMA-Chips untergebracht. Diese Zwischenspeicher sind als sogenannte Fifos (First In - First Out) organisiert. Das bedeutet, daß das zuerst in den Zwischenspeicher eingeschriebene Byte auch zuerst wieder ausgelesen wird.

Ist der benutzte Fifo voll, so versucht der DMA-Chip über den Anschluß Bus-Request den 68000 zu unterbrechen. Da sowohl der Prozessor für die Freigabe "seines" Daten- und Adreß-Busses als auch der eigentliche DMA-Transfer in das RAM des ST einige Zeit dauert, der externe Controller aber weiterhin seine Daten liefert, werden jetzt eintreffende Bytes im zweiten Fifo gespeichert. So ergibt sich jedesmal genug Zeit, um einen Fifo in das RAM zu übertragen. Bei dieser Übertragung hat der 68000 des ST überhaupt nichts zu melden. Er bekommt davon bis auf die DMA-Anforderungen nicht einmal etwas mit. Die geschilderten Abläufe werden ausschließlich durch die Logik des DMA-Chips kontrolliert. Aus diesem Grund sind Übertragungsgeschwindigkeiten von über 1,5 MByte möglich. Rechnerisch ergibt sich eine maximale Grenze der Übertragungsgeschwindigkeit irgendwo unterhalb von 4 Megabyte. Diese Obergrenze errechnet sich aus der festgelegten ACK-Zeit von 250 Nano-Sekunden. Da der Prozessor an diesen Vorgängen nicht beteiligt ist, muß die Erzeugung der Adressen separat erfolgen. Dafür ist der DMA-Chip mit seinen drei acht Bit breiten DMA-Adress-Register zuständig. Über diese Register wird der Speicher des ST adressiert, wenn Daten per DMA bewegt werden. Die Adressen dieser Register befinden sich auf den Adressen FF8609 (Highbyte), FF860B (Midbyte) und FF860D (Lowbyte). Hier wird die Adresse des ersten zu übertragenden Bytes eingeschrieben. Mit jedem übertragenen Byte wird der Inhalt dieser Register automatisch entsprechend erhöht.

Neben der Erzeugung der Adressen ist es erforderlich, auch die Größe des gewünschten Datenbereichs anzugeben. Dafür ist das Sektor-Count-Register auf der Adresse FF8604 vorhanden. Da jeder DMA-Transfer mindestens 512 Bytes bewegt, kleinere Speicherbereiche sind nicht möglich, reicht hier ein ein Byte großes Register, um immerhin $255 * 512 = 130560$ Bytes in einem Rutsch zu übertragen. Jedesmal, wenn ein 512 Byte großer Bereich übertragen worden ist, wird der Inhalt dieses Registers um eins vermindert. Erreicht der Zählerstand den Wert Null, so ist die DMA-Übertragung beendet. Für den Betrieb der Festplatte oder der Floppys, die ja auch per DMA betrieben werden, reicht

diese Größe fast immer aus. Für den Laserdrucker allerdings ist diese Größe nicht ausreichend. Hier muß der DMA-Transfer mehrfach mit neuen Adressen und Sektor-Counts gestartet werden. Der Begriff Sektor-Count übrigens bezieht sich auf die bei Atari übliche Größe eines Datensektors bei Floppy und Festplatte.

20.9 Der interne Erweiterungs-Bus der Mega-ST-Serie

Die Mega-ST-Serie unterscheidet sich im Grunde genommen nur recht wenig von den "alten" Geräten der ST-Serie. Die wohl wesentlichen Unterschiede sind der deutlich vergrößerte Arbeitsspeicher, der eingebaute Blitter und der interne Erweiterungs-Bus. Dieser Erweiterungs-Bus besteht aus einer 64-poligen Steckverbindung und ist für die Aufnahme einer Erweiterungskarte vorgesehen. Macht man sich die Mühe und zählt die Anschluß-Pins des Prozessors 68000 (ein kurzer Blick in das entsprechende Kapitel dieses Buches tut es jedoch auch), so stellt man mehr oder weniger schnell fest, daß auch der Prozessor über 64 Anschlußpins verfügt. Ja, mehr noch. Ein Vergleich der Abbildung 20.18 mit der Abbildung des Prozessors zeigt, daß die Belegung der 64-poligen Anschlußleiste (nicht aber die Numerierung!) fast vollständig der des Prozessors entspricht. Nur die Anschlüsse zur Spannungsversorgung des Prozessors und die Interrupt-Eingänge sind am Erweiterungs-Port unterschiedlich belegt. Die Pins 27 und 32 des Erweiterungs-Busses, die den Prozessoranschlüssen 14 und 49 entsprechen, sind nicht mit +5 Volt, sondern mit Masse verbunden. Über die Steckerleiste kann also eine Zusatzschaltung nicht versorgt werden. Dafür gibt es im Mega ST einen separaten Stromversorgungsstecker, den wir etwas später beschreiben werden.

Die Anschlüsse 23 bis 25 des Prozessors sind bekanntlich die Interrupt-Eingänge des 68000. Auf dem Erweiterungsstecker haben die entsprechenden Anschlüsse die Nummern 45, 47 und 49. Allerdings weicht die Belegung des Erweiterungs-Ports von der Prozessorbelegung ab. Der 68000 kann mit seinen drei Interrupt-Eingängen bis zu 7 unterschiedlich priorisierte Interrupts verarbeiten. Da jedoch ein Interrupt in der Regel nur als ein einzelnes Signal am Pin eines Peripherie-Bausteins auftritt, muß ein sogenannter 1-aus-8-Dekoder der CPU ein klein wenig helfen, um aus dem einen Signal eine Drei-Bit-Kombination zu erzeugen (Abbildung 20.19). Die Eingänge 1 bis 7 des Dekoders können jetzt direkt als Interrupt-Eingänge benutzt werden. Der Eingang 0 des Dekoders ist nicht nutzbar, da die drei Ausgänge A, B und C unabhängig vom Pegel an Eingang 0 High bleiben. Diese Kombination entspricht dem Prozessorzustand "kein Interrupt".

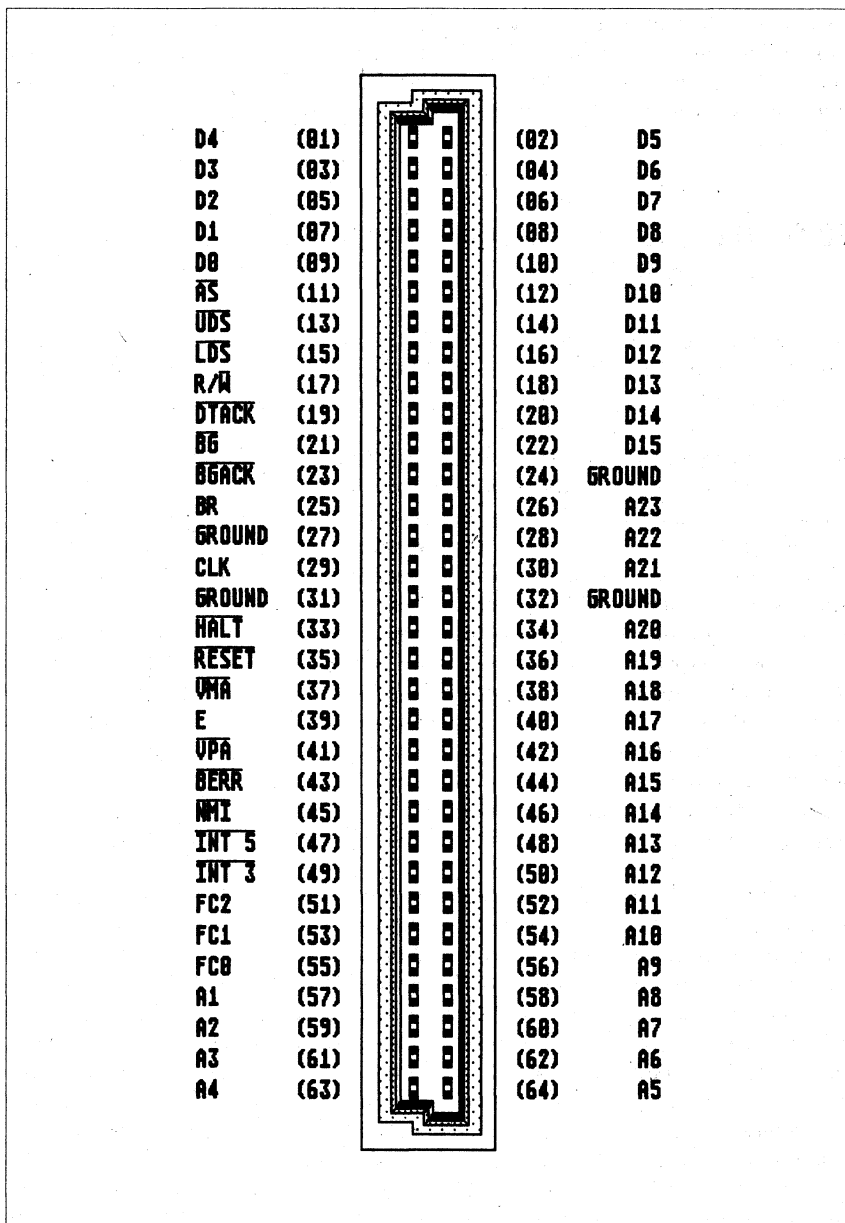


Abb. 20.18: Die Belegung der 64-poligen Anschlußleiste

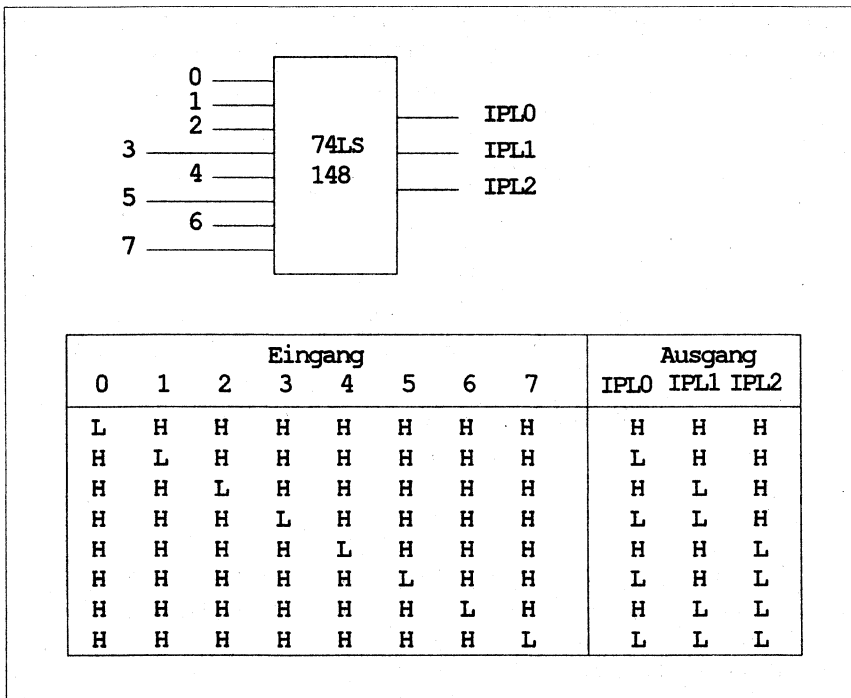


Abb. 20.19: Die Ein- und Ausgangszustände des 74LS148

Im ST werden von der vorhandenen Hardware nur die beiden Interrupt-Leitungen IPL1 und IPL2 genutzt. Damit sind drei verschiedene Interrupts möglich. Für den Erweiterungs-Port wird jetzt auch der Interrupt-Eingang IPL0 ins Spiel gebracht. Eine einfache Rechnung zeigt, daß, bei sieben möglichen und drei belegten Interrupts, jetzt noch vier Interrupts frei sind. Da aber nur die drei Pins 45, 47 und 48 auf dem Stecker des Erweiterungs-Ports zur Verfügung stehen, mußte ein möglicher Interrupt weiterhin ungenutzt bleiben. Auf dem Erweiterungs-Port stehen jetzt die Interrupts NMI, Int5 und Int3 zur Verfügung. Das Vorhandensein des NMI ist besonders interessant, da dieser im Gegensatz zu den anderen Interrupts nicht maskierbar ist und bei seinem Auftreten immer ausgeführt wird. Der NMI hat die höchste Priorität.

Über den sechspoligen Stromversorgungsstecker (Abbildung 20.20) kann eine Erweiterungsplatine mit +5 Volt und +12 Volt versorgt werden. Das Netzteil des Mega ST ist so ausreichend dimensioniert, daß die 5 Volt immerhin mit

750 Milliampere und der 12-Volt-Kreis mit 500 Milliampere belastet werden darf. Damit können schon recht umfangreiche Zusatzschaltungen versorgt werden.

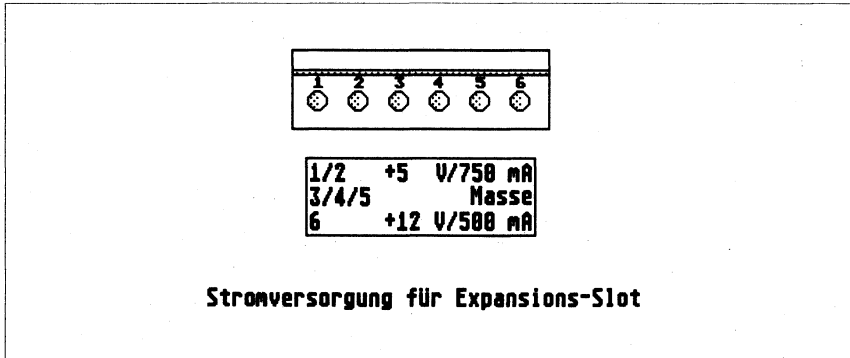


Abb. 20.20: Der sechspolige Stromversorgungsstecker

Mag zwar für manchen die Beschränkung auf nur einen Erweiterungs-Port etwas wenig erscheinen, so hat diese Beschränkung zumindest in einer Hinsicht auch ihre positiven Seiten. Der Entwickler einer Erweiterungskarte muß sich beim Entwurf nicht darum kümmern, daß "seine" Karte mit denen anderer Hersteller kompatibel ist. Jede Karte kann sich die freien Adreßbereiche nach eigenem Gutdünken aussuchen. Allerdings legt Atari den Entwicklern von Erweiterungskarten nahe, nur bestimmte Adreßbereiche zu benutzen. Diese Bereiche werden von Atari wie folgt angegeben:

```
$C00000 bis $CFFFFFFF
$FF0000 bis $FF7FFF
$FFFE00 bis $FFFFFFD
```

Teil D

21. Die Funktionen des GEMDOS

\$00 PTERM

C: void Pterm0()

Assembler: CLR.W -(SP)
 TRAP #1

GFA-BASIC: nicht anwendbar

OMIKRON: nicht anwendbar

Ein Aufruf des GEMDOS mit der Funktionsnummer 0 beendet das laufende Programm und kehrt an das Programm zurück, von dem es gestartet wurde. Bei Applikationen, also Programmen, die vom Desktop aus gestartet wurden, wird die Programmkontrolle wieder dem Desktop übergeben. Wurde das Programm von einem anderen Programm aus aufgerufen, so wird in das aufrufende Programm zurückgesprungen. Dieser Punkt ist vor allem für das Nachladen von Programmteilen wichtig.

Dieser Aufruf darf von GFA- oder OMIKRON-BASIC aus nicht ausgeführt werden, andernfalls "hängt" sich der ST auf.

Beispiel: CLR.W -(SP) 16 Null-Bits auf den Stack
 TRAP #1 und das Programm beenden.
 . Wenn es jetzt hier weitergeht,
 . dann ist etwas ganz falsch!

\$01 CCONIN

C: long Cconin()

Assembler: MOVE.W #1, -(SP)
 TRAP #1
 ADDQ.L #2, SP

GFA-BASIC: x% = GEMDOS(1)

OMIKRON: GEMDOS(x,1)

Mit CCONIN kann ein einzelnes Zeichen von der Tastatur geholt werden. Dabei wird so lange gewartet, bis ein Zeichen verfügbar ist. Das Ergebnis, also das von der Tastatur gelesene Zeichen, wird im Register D0 übergeben. Im Low-Byte des Low-Word steht der ASCII-Code der gedrückten Taste, im Low-Byte des High-Word des Registers wird der von der Tastatur gelieferte Scan-Code übergeben. Das ist wichtig bei der Abfrage der Tasten, denen kein ASCII-Code zugeordnet ist. Das trifft z.B. auf die zehn Funktionstasten oder die Tasten des Cursorblocks zu. Diese Tasten liefern bei Betätigung den ASCII-Wert Null.

Auch kann bei Bedarf anhand des Scan-Codes unterschieden werden, ob Tasten des Ziffernblocks oder der Haupttastatur gedrückt wurden, da diese Tasten zwar identische ASCII-Codes, jedoch unterschiedliche Scan-Codes liefern.

Zusätzlich kann in den obersten acht Bits (Bit 24 bis 31) der Shift-Status beim Cconin-Aufruf mitgeliefert werden. In diesem Fall entsprechen die Bits 24 bis 31 den Bits 0 bis 7 der BIOS-Funktion 11 "kbshift". Die Information wird allerdings nur dann beim Cconin-Aufruf geliefert, wenn zuvor das Bit 3 in der Speicherzelle "conterm" an der Adresse \$484 gesetzt wurde. Ist das Bit gelöscht, so sind die Shift-Bits nach dem Cconin gelöscht. Cconin erkennt kein **[Ctrl]-[C]**. Dieser Aufruf soll in OMIKRON-BASIC nicht benutzt werden.

Beispiel:

MOVE.W	#1,-(SP)	Funktionsnummer auf den Stack
TRAP	#1	GEMDOS aufrufen
ADDQ.L	#2,SP	Stack korrigieren
		Ergebnis in D0

\$02 CCONOUT

C:

void	Cconout(zeichen)
int	zeichen;

Assembler:

MOVE.W	#zeichen,-(SP)
MOVE.W	#2,-(SP)
TRAP	#1
ADDQ.L	#4,SP

GFA-BASIC: VOIDGEMDOS(2,zeichen%)

OMIKRON: GEMDOS(,2,zeichen)

CCONOUT stellt die einfachste (primitivste) Zeichenausgabe des GEMDOS dar. Mit dieser Funktion kann nur ein Zeichen auf dem Bildschirm ausgegeben werden. Dazu wird das Zeichen als erstes Wort auf den Stack gelegt. Der

ASCII-Wert des auszugebenden Zeichens muß im Low-Byte des Wortes stehen, das High-Byte sollte Null sein. Dieser Aufruf soll in OMIKRON-BASIC nicht benutzt werden.

Beispiel:

MOVE.W	#65,-(SP)	Ein A soll ausgegeben werden
MOVE.W	#2,-(SP)	Cconout
TRAP	#1	GEMDOS aufrufen, Zeichen ausgegen
ADDQ.L	#4,SP	Stack korrigieren

\$03 CAUXIN

C: int Cauxin()

Assembler:

MOVE.W	#3,-(SP)
TRAP	#1
ADDQ.L	#2,SP

GFA-BASIC: c% = GEMDOS(3)

OMIKRON: GEMDOS(c,3)

Unter der Bezeichnung Auxiliary Port verbirgt sich die RS232-Schnittstelle des Atari. Mit der Funktion Cauxin kann von der Schnittstelle ein Zeichen geholt werden. Die Funktion kehrt zurück, wenn ein Zeichen vollständig empfangen wurde. Das Zeichen befindet sich nach der Rückkehr in den unteren acht Bit des Registers D0.

Beispiel:

MOVE.W	#3,-(SP)	Cauxin
TRAP	#1	GEMDOS aufrufen, Zeichen ausgegen
ADDQ.L	#2,SP	Stack korrigieren
.		Zeichen in D0

\$04 CAUXOUT

C: void Cauxout(zeichen)
int zeichen;

Assembler:

MOVE.W	#zeichen,-(SP)
MOVE.W	#4,-(SP)
TRAP	#1
ADDQ.L	#4,SP

GFA-BASIC: VOIDGEMDOS(4,zeichen%)

OMIKRON: GEMDOS(,4,zeichen)

Analog der Eingabe von Zeichen über die serielle Schnittstelle kann mit dieser Funktion ein Zeichen gesendet werden. Auch bei dieser Funktion sollte der Programmierer die oberen acht Bits des Wortes löschen und den Code des zu sendenden Zeichens in den unteren acht Bits übergeben.

Beispiel:

MOVE.W	#\$41, -(SP)	Ein A soll ausgegeben werden
MOVE.W	#4, -(SP)	Cauxout
TRAP	#1	GEMDOS aufrufen, Zeichen ausgeben
ADDQ.L	#4, SP	Stack korrigieren

\$05 CPRNOUT

C:

```
void Cprnout(zeichen)
int          zeichen;
```

Assembler:

MOVE.W	#zeichen, -(SP)
MOVE.W	#5, -(SP)
TRAP	#1
ADDQ.L	#4, SP

GFA-BASIC: VOIDGEMDOS(5, zeichen%)

OMIKRON: GEMDOS(, 5, zeichen)

Dieser Aufruf ist die einfachste Methode, einen an der Centronics-Schnittstelle angeschlossenen Drucker zu betreiben. Mit jedem Aufruf wird dem Drucker ein Zeichen übergeben.

Wurde das Zeichen an den Drucker gesendet, so erhält man den Wert -1 (\$FFFFFFF) in D0 zurück. Ist jedoch der Drucker innerhalb von ca. 30 Sekunden nicht in der Lage, das Zeichen anzunehmen (nicht eingeschaltet, OFF LINE, kein Papier usw.), so ist in D0 eine Null enthalten.

Beispiel:

MOVE.W	#65, -(SP)	Ein A soll ausgegeben werden
MOVE.W	#5, -(SP)	Funktionsnummer
TRAP	#1	GEMDOS aufrufen, Zeichen ausgeben
ADDQ.L	#4, SP	Stack korrigieren
TST.W	D0	Flags beeinflussen
BEQ		printerfehler

\$06 CRAWIO

C:

```
long crawio(wrd)
int wrd;
```

Assembler: MOVE.W # wrd, -(SP)
 MOVE.W # 6, -(SP)
 TRAP # 1
 ADDQ.L # 4, SP

GFA-BASIC: zeichen% = GEMDOS(6, wrd%)

OMIKRON: GEMDOS(zeichen, 6, wrd)

CRAWIO ist eine etwas ungewöhnliche Mischung aus Tastatureingabe und Bildschirmausgabe und erhält einen Parameter auf dem Stack.

Bei einem Funktionsparameter \$FF wird die Tastatur überprüft. Wenn ein Zeichen vorhanden ist, so wird es, wie schon bei CCONIN besprochen, mit ASCII- und SCAN-Code in D0 übergeben. Ist jedoch kein Tastenwert vorhanden, so wird der Wert Null sowohl als ASCII-, als auch als SCAN-Code in D0 übergeben.

Wird ein von \$FF unterschiedlicher Wert an die Funktion übergeben, so wird der Wert als auszugebendes Zeichen interpretiert und an der momentanen Cursor-Position ausgegeben.

Dieser Aufruf soll in OMIKRON-BASIC nicht benutzt werden.

Beispiel:

ANFANG:

MOVE.W	#\$FF, -(SP)	Funktionswert Tastatur prüfen
MOVE.W	#6, -(SP)	Funktionsnummer
TRAP	#1	GEMDOS aufrufen, Tastatur prüfen
ADDQ.L	#4, SP	Stack korrigieren
TST.W	D0	Zeichen angekommen?
BEQ	ANFANG	noch nicht!
CMP.B	#3, D0	^C frei gewähltes Ende-Kennzeichen
BEQ	ENDE	
MOVE	D0, -(SP)	Zeichen für die Ausgabe auf den Stack
MOVE	#6, -(SP)	Funktionsnummer
TRAP	#1	GEMDOS aufrufen, Tastatur prüfen
ADDQ.L	#4, SP	Stack korrigieren
BRA	ANFANG	neues Zeichen holen

\$07 CRAWCIN

C: long Crawlcin()

Assembler: MOVE.W #7, -(SP)
TRAP #1
ADDQ.L #2, SP

GFA-BASIC: zeichen% = GEMDOS(7).

OMIKRON: GEMDOS(zeichen,7)

Die Funktion \$07 unterscheidet sich von Cconin \$01 nur dadurch, daß das von der Tastatur erhaltene Zeichen nicht auf dem Bildschirm ausgegeben wird. Es wird also genau wie bei Cconin auf einen Tastendruck gewartet. Allerdings darf man dieses Warten nicht zu wörtlich nehmen. Durch den Tastaturpuffer des ST kann es natürlich passieren, daß die entsprechenden Routinen wie Cconin oder Crawlcin mit einem Zeichen zurückkommen, auch wenn Sie zum Zeitpunkt des Aufrufes gar keine Taste gedrückt haben.

Dieser Aufruf soll in OMIKRON-BASIC nicht benutzt werden.

Beispiel: MOVE.W #7, -(SP) Funktionsnummer von Crawlcin
TRAP #1 GEMDOS aufrufen, Zeichen ausgegen
ADDQ.L #2, SP Stack korrigieren
Zeichen in D0

\$08 CNECIN

C: long Cnecin()

Assembler: MOVE.W #8, -(SP)
TRAP #1
ADDQ.L #2, SP

GFA-BASIC: zeichen% = GEMDOS(8)

OMIKRON: GEMDOS(zeichen,8)

Die Funktion \$08 unterscheidet sich nicht von der zuvor genannten Funktion \$07. Beide Funktionsaufrufe bewirken exakt dasselbe. Die Ursache für dieses scheinbar unsinnige Verhalten liegt in der schon erwähnten Kompatibilität mit MS-DOS begründet. Unter MS-DOS unterscheiden sich die beiden Funktionen

dadurch, daß mit \$08 bestimmte, beim Atari nicht vorhandene Tasten richtig ausgewertet werden, während diese Auswertung bei der Funktion \$07 unterbleibt.

Dieser Aufruf soll in OMIKRON-BASIC nicht benutzt werden.

Beispiel:

MOVE.W	#8, -(SP)	Cauxin
TRAP	#1	GEMDOS aufrufen, Zeichen ausgegen
ADDQ.L	#2, SP	Stack korrigieren
.		Zeichen in D0

\$09 CCONWS

C:

```
void Cconws(string)
char *string;
```

Assembler:

PEA	string
MOVE.W	#9, -(SP)
TRAP	#1
ADDQ.L	#6, SP

GFA-BASIC: VOID GEMDOS(9, L:string_address%)

OMIKRON: GEMDOS(,9,HIGH(string),LOW(string))

Mit Cconws, auch PRINT LINE genannt, kann eine ganze Zeichenkette an der momentanen Cursorposition ausgegeben werden. Dazu wird die Adresse der Zeichenkette als Parameter auf dem Stack übergeben. Die Zeichenkette selbst wird durch ein Null-Byte begrenzt.

Nach dem Aufruf erhalten Sie in D0 die Anzahl der Zeichen, die ausgegeben wurden. Die Definition VOID sollte also eventuell in WORD geändert werden.

Dieser Aufruf soll in OMIKRON-BASIC nicht benutzt werden.

Beispiel:

PEA	text	Adresse der Zeichenkette auf den Stack
MOVE	#\$09, -(SP)	Funktionsnummer Cconws
TRAP	#1	GEMDOS aufrufen
ADDQ.L	#6, SP	Stack wieder "aufräumen"
.		
text	.dc.b	'dies ist der auszugebende Text', \$0d, \$0a, 0

\$0A CCONRS

C: void Cconrs(buffer)
 char *buffer;

Assembler: PEA buffer
 MOVE.W #\$0A, -(SP)
 TRAP #1
 ADDQ.L #6, SP

GFA-BASIC: VOID GEMDOS(10, L:buffer_address%)

OMIKRON: GEMDOS(,10,HIGH(buffer),LOW(buffer))

Cconrs oder READLINE stellt eine recht komfortable Funktion zur Eingabe von Zeichen über die Tastatur dar. Im Gegensatz zu den "einfacheren" zeichenorientierten Eingabefunktionen kann mit READLINE eine ganze Eingabezeile von der Tastatur geholt werden. Die eingegebenen Zeichen werden gleichzeitig auf dem Bildschirm dargestellt.

Als Parameter wird der Funktion die Adresse eines Eingabe-Buffers übergeben. Der Wert des ersten Bytes des Eingabe-Buffers bestimmt die maximale Länge der Eingabezeile, muß also vor dem Aufruf initialisiert werden. Das zweite Byte des Buffers enthält nach dem Beenden der Eingabe die Anzahl der eingegebenen Zeichen. Die Zeichen selbst stehen im Buffer ab dem dritten Byte.

Die von READLINE benutzte Routine zur Tastatureingabe ist von den zeichenorientierten Console-Eingaben gänzlich unterschiedlich. Escape-Sequenzen werden bei der Ausgabe unterdrückt und nicht an den VT52-Emulator durchgereicht. Nur Control-Zeichen wie Control-H (Backspace) und Control-I (TAB) werden erkannt und entsprechend behandelt. Insgesamt sind die folgenden, teilweise sonst nicht wirksamen Control-Zeichen möglich:

- ^C - beendet Eingabe UND Programm (!)
- ^H - Backspace ein Zeichen
- ^I - TAB, Tabulatorsprung
- ^J - Line Feed, Eingabe beenden
- ^M - CR, Eingabe beenden
- ^R - eingegebene Zeile wird in neuer Zeile ausgegeben
- ^U - Zeile nicht werten, in neuer Zeile beginnen
- ^X - Zeile löschen, Cursor an Zeilenanfang.

Wenn man sicher ist, nicht die Control-Funktionen zu verwenden, so kann man READLINE recht einfach handhaben. Besonders die Funktion ^H, das Löschen eingegebener Zeichen, ist nützlich. Trotzdem sollte man bei umfangreicheren Programmen eine eigene Eingaberoutine schreiben, da ^C sehr "gefährlich" ist. Nicht nur am unmittelbaren Zeilenanfang (wie bei CP/M), sondern immer während der Eingabe wird mit ^C das Programm sofort beendet. Diese Möglichkeit ist nicht ausschaltbar.

Werden mehr Zeichen eingegeben als bei der Initialisierung im ersten Byte des Buffers angegeben wurden, so wird automatisch die Eingabe abgebrochen. Wird die Eingabe mit `Enter`, ^J oder ^M abgeschlossen, so steht das Abschlußzeichen nicht im Buffer (im Gegensatz zur offiziellen Atari-Dokumentation).

Nach der Eingabe enthält D0 die Anzahl der eingegebenen Zeichen ohne `Enter`, die ja auch im Buffer+1 gefunden werden kann. Der eingegebene String wird nicht unbedingt von einer Null abgeschlossen!

Beispiel:	PEA	buffer	Adresse des Eingabebuffers
	MOVE	#\$0a, -(SP)	Funktionsnummer
	TRAP	#1	
	ADDQ.L	#6, SP	Stack aufräumen
	.		
	.		
	buffer	DC.B 20	wir wollen max. 20 Zeichen haben
		DC.B 0	Anzahl der eingegebenen Zeichen
		DS.B 20	der eigentliche Eingabepuffer

\$0B CCONIS

C: int Cconis()

Assembler: MOVE.W #\$0B, -(SP)
 TRAP #1
 ADDQ.L #2, SP

GFA-BASIC: status! = GEMDOS(11)

OMIKRON: GEMDOS(status, 11)

Alle Tastenbetätigungen werden zunächst in einem Buffer des Betriebssystems zwischengespeichert. Dieser Speicher ist insgesamt 256 Bytes groß. Die zwischengespeicherten Tastenwerte werden erst beim Aufruf einer GEMDOS-Eingaberoutine aus dem Buffer an die Routine übergeben.

Mit CONSTAT oder Cconis kann geprüft werden, ob im Tastatur-Buffer Zeichen gespeichert sind. Nach dem Aufruf erhält man in D0 den Wert Null oder \$FFFF. Eine Null in D0 signalisiert, daß keine Zeichen verfügbar sind.

Diese Funktion in OMIKRON-BASIC bitte nicht verwenden.

Beispiel:

testloop:

MOVE	#\$0b, -(SP)	Funktionsnummer
TRAP	#1	
ADDQ.L	#2, SP	Stack aufräumen
TST.W	D0	Zeichen verfügbar?
BEQ	testloop	nein, noch mal nachsehen

\$0E DSETDRV

C: long Dsetdrv(drive)
 int drive;

Assembler: MOVE.W #drive, -(SP)
 MOVE.W #\$0E, -(SP)
 TRAP #1
 ADDQ.L #4, SP

GFA-BASIC: VOID GEMDOS(14, drive%)

OMIKRON: GEMDOS(,14,drive)

Mit der Funktion SETDRV kann das aktuelle Laufwerk bestimmt werden. Dazu wird dem Aufruf ein Parameter (16 Bit) übergeben, in dem die Laufwerksbezeichnung enthalten ist. Das Laufwerk A wird mit der Nummer 0, das Laufwerk B mit der Nummer 1 angesprochen.

Nach dem Aufruf erhält man in D0 die Nummer des vor dem Aufruf aktiven Laufwerkes.

Beispiel:

MOVE	#\$2, -(SP)	Drive C, z.B. Randisk
MOVE	#\$0e, -(SP)	Funktionsnummer
TRAP	#1	
ADDQ.L	#4, SP	Stack aufräumen vormals aktuelles Drive in D0

\$10 CCONOS

C: int Cconos()

Assembler: MOVE.W #\$10,-(SP)
 TRAP #1
 ADDQ.L #2,SP

GFA-BASIC: status! = GEMDOS(16)

OMIKRON: GEMDOS(status,16)

Cconos oder "CONOUT STAT" liefert nach dem Aufruf in D0 den Status der Console. Bei Rückgabe des Wertes \$FFFF kann ein Zeichen auf dem Bildschirm dargestellt werden. Ist jedoch der Rückgabewert Null, so ist derzeit keine Zeichenausgabe auf dem Bildschirm möglich. Wer jedoch hofft, einmal den "not ready"-Status zu erwischen, der wird durch einen Blick in das BIOS-Listing eines Besseren belehrt. Die Funktion liefert immer den Wert \$FFFF gleich -1 zurück. Die Funktion ist somit äußerst überflüssig.

Beispiel: MOVE #\$10,-(SP) Funktionsnummer
 TRAP #1
 ADDQ.L #2,SP Stack aufräumen
 immer \$FFFF in D0

\$11 CPRNOS

C: int Cprnos()

Assembler: MOVE.W #\$11,-(SP)
 TRAP #1
 ADDQ.L #2,SP

GFA-BASIC: status! = GEMDOS(17)

OMIKRON: GEMDOS(status,17)

Diese Funktion liefert den Status, also den Zustand der Centronics-Schnittstelle. Ist kein Drucker angeschlossen (nicht eingeschaltet, OFF LINE usw.), so erhält man nach dem Aufruf in D0 den Wert Null als Kennzeichen "Drucker nicht verfügbar". Ist der Drucker jedoch empfangsbereit, so erhält man in D0 den Wert \$FFFF zurück.

Beispiel:

MOVE	#\$11, -(SP)	wie geht's dem Drucker?
TRAP	#1	
ADDQ.L	#2, SP	Stack aufräumen
TST	D0	
BEQ	printererror	da stimmt was nicht!

\$12 CAUXIS

C: int Cauxis()

Assembler:

MOVE.W	#\$12, -(SP)
TRAP	#1
ADDQ.L	#2, SP

GFA-BASIC: status! = GEMDOS(18)

OMIKRON: GEMDOS(status,18)

Durch den Aufruf von Cauxis oder "AUXIN STAT" erfährt man, ob ein Zeichen vom Empfänger der seriellen Schnittstelle verfügbar ist (\$FFFF) oder nicht (Null). Wie bei allen anderen Funktionen wird der Wert in D0 zurückgegeben.

Beispiel:

warteloop:

MOVE	#\$12, -(SP)	wir warten auf ein zeichen
TRAP	#1	von der V24
ADDQ.L	#2, SP	Stack aufräumen
TST	D0	ist ein Zeichen da?
BNE	warteloop	nein, leider noch nicht

\$13 CAUXOS

C: int Cauxos()

Assembler:

MOVE.W	#\$13, -(SP)
TRAP	#1
ADDQ.L	#2, SP

GFA-BASIC: status! = GEMDOS(19)

OMIKRON: GEMDOS(status,19)

Cauxos oder "AUXOUT STAT" gibt Auskunft über den Zustand des Senders der seriellen Schnittstelle. Ein Wert von \$FFFF signalisiert, daß die serielle Schnittstelle ein Zeichen senden kann, Null dagegen signalisiert, daß derzeit kein Zeichen zum Transmitter geschickt werden sollte.

Beispiel:

warteloop:

MOVE	#\$13,-(SP)	wir warten, bis wir ein Zeichen
TRAP	#1	über die V24 senden dürfen
ADDQ.L	#2,SP	Stack aufräumen
TST	D0	dürfen wir jetzt?
BNE	warteloop	nein, leider noch nicht

\$19 DGETDRV

C: int Dgetdrv()

Assembler:

MOVE.W	#\$19,-(SP)
TRAP	#1
ADDQ.L	#2,SP

GFA-BASIC: drive% = GEMDOS(25)

OMIKRON: GEMDOS(drive,25)

Für manche Anwendungen ist es erforderlich zu wissen, welches Laufwerk derzeit angemeldet ist. Mit der Funktion \$19 kann das aktuelle Laufwerk einfach ermittelt werden. Nach dem Aufruf erhält man die Nummer des Laufwerks in D0 zurück. Die Bedeutung der Nummern ist identisch mit den Laufwerksnummern bei \$0E, Dsetdrv(), also 0 = A, 1 = B usw.

Beispiel:

MOVE	#\$19,-(SP)	welches Drive ist wohl aktiv?
TRAP	#1	
ADDQ.L	#2,SP	Stack aufräumen
ADD.B	#'A',D0	jetzt steht in D0 ein Zeichen zwischen "A" und "P"

\$1A FSETDTA

C:

```
void Fsetdta(buf)
char *buf;
```

Assembler:

PEA	buf
MOVE.W	#\$1A,-(SP)

```

TRAP          #1
ADDQ.L        #6,SP

```

GFA-BASIC: VOID GEMDOS(26, buf_address%)

OMIKRON: GEMDOS(,26,HIGH(buf_address),LOW(buf_address))

Die Disk-Transfer-Adresse ist die Adresse eines 44-Byte-Buffers, der für verschiedene Diskettenoperationen (besonders Directory-Operationen) benötigt wird. Beispiele für die Verwendung der DTA finden Sie bei den GEMDOS-Funktionen SEARCH FIRST und SEARCH NEXT.

Beispiel:

PEA	dtabuf	Adresse des 44-Byte DTA-Buffers
MOVE.W	#\$1A,-(SP)	Funktionsnummer SET DTA
TRAP	#1	DTA setzen
ADDQ.L	#6,SP	Stack aufräumen
.		
.		
dtabuf	ds.b 44	Platz für DTA-Buffer

\$20 SUPER

C: long Super(modus)
long modus;

Assembler: MOVE.L #modus,-(SP)
MOVE.W #\$20,-(SP)
TRAP #1
ADDQ.L #6,SP

GFA-BASIC: nicht verfügbar

OMIKRON: nicht anwendbar

Diese Funktion ist besonders für die Programmierer interessant, die während eines im User-Modus laufenden Programms auf die nur im Supervisor-Modus zugänglichen Peripherie-Bausteine oder die Systemvariablen zugreifen wollen oder müssen. Nach Aufruf dieser Funktion aus dem User-Modus heraus befindet sich der MC68000 im Supervisor-Mode. Im Gegensatz zur XBIOS-Routine zum Einschalten des Supervisor-Modus können nach erfolgtem SUPER-Aufruf weitere GEMDOS-, BIOS- und XBIOS-Aufrufe erfolgen. Der Aufruf von AES-Funktionen aus dem Supervisor-Mode heraus ist jedoch verboten. Zuwiderhandlungen werden durch Abwurf mehrerer Bomben bestraft.

Wird die Super-Funktion mit einem Wert von 1L (\$00000001) aufgerufen, dann wird uns mitgeteilt, in welcher Betriebsart sich der Prozessor befindet. Ist das Ergebnis in D0 nach dem Aufruf eine 0, so läuft der Prozessor im User-Mode. Ein Wert von -1L dagegen zeigt an, daß der Prozessor im Supervisor-Mode läuft. Eine Umschaltung wird mit diesem Parameter jedoch noch nicht vorgenommen.

Darum wollen wir jetzt den Fall betrachten, daß sich ein Programm im User-Mode befindet und die SUPER-Funktion mit einem Wert von Null auf dem Stack aufruft. In diesem Fall wird durch den Aufruf der Supervisor-Mode eingeschaltet. Der Supervisor-Stackpointer wird auf den derzeitigen Wert des User-Stacks gebracht, und in D0 erhält man den ursprünglichen Wert des Supervisor-Stacks zurück. Diesen Wert muß man sich im Programm unbedingt merken, um später wieder in den UserModus zu gelangen. Unterläßt man dies vor dem Beenden des Programms, so stehen die Chancen für einen Systemabsturz sehr gut.

Wird jedoch als Parameter beim erstmaligen Aufruf ein von Null unterschiedlicher Wert der SUPER-Funktion übergeben, so wird dieser Wert als gewünschter neuer Supervisor-Stack interpretiert. Auch in diesem Fall erhalten Sie in D0 den alten Supervisor-Stackpointer zurück, dessen Wert das Programm auf jeden Fall speichern sollte.

Wie bereits erwähnt muß spätestens vor der Beendigung des Programms der User-Mode wieder eingeschaltet werden. Dieser erneute Wechsel der Betriebsart benötigt die beim ersten Aufruf erhaltene Adresse, um den Supervisor-Stack auf seinen ursprünglichen Wert zu setzen.

Die Superfunktion unterscheidet sich in einem ganz wesentlichen Punkt von den bisherigen und allen anderen GEMDOS-Funktionen. Dieser Aufruf kann unter Umständen auch die Inhalte von A1 und D1 verändern. Wenn Sie also wichtige Werte in diesen Registern halten, so müssen Sie die Werte vor einem Aufruf der SUPER-Funktion irgendwo ablegen.

Beispiel:

CLR.L	-(SP)	der 68000 befindet sich im User-Modus User-Stack soll Supervisor-Stack werden
MOVE.W	#\$20,-(SP)	SUPER aufrufen
TRAP	#1	nach dem Trap ist Supervisor-Modus aktiv
ADD.L	#6,SP	D0 = alter Supervisor-Stack
MOVE.L	D0,_SAVE_SSP	Wert merken

hier kann im Supervisor-Mode
gearbeitet werden
also z.B. Zugriffe auf die System-
Variablen oder die
Register der Peripherie-Bausteine.

```
MOVE.L    _SAVE_SSP, -(SP)  alten Supervisor-Stackpointer
MOVE.W    #$20, -(SP)      SUPER aufrufen
TRAP      #1                jetzt sind wir wieder im User Mode
ADD.L     #6, SP
```

\$2A TGETDATE

C: int Tgetdate()

Assembler: MOVE.W #\$2A, -(SP)
 TRAP #1
 ADDQ.L #2, SP

GFA-BASIC: datum% = GEMDOS(42)

OMIKRON: GEMDOS(datum, 42)

Sicher haben Sie bereits einmal mit dem Kontrollfeld experimentiert. Neben verschiedenen anderen Funktionen ist im Kontrollfeld eine Uhr mit Uhrzeit und Datum enthalten. Für verschiedene Anwendungen kann es recht nützlich sein, das eingestellte Datum zur Verfügung zu haben. Das Datum läßt sich problemlos mit der Funktion GET DATE ermitteln. Dieser Funktionsaufruf benötigt keine Parameter und stellt nach dem Aufruf das Datum im Low-Word des Registers D0 zur Verfügung. Allerdings ist es recht aufwendig codiert, so daß das Ergebnis in D0 noch aufbereitet werden muß, um ein "richtiges" Datum zu erhalten.

In den unteren fünf Bits wird der Tag im Bereich von 1 bis 31 codiert. Die Bits 5 bis 8 enthalten den Monat im Wertebereich von 1 bis 12, und das Jahr ist in den Bits 9 bis 15 enthalten. Der Wertebereich in diesen "Jahresbits" geht von 0 bis 119. Der Wert dieser Bits muß zum Wert 1980 addiert werden, um die tatsächlich eingestellte Jahreszahl zu bekommen. Das Datum 12.12.1992 z.B. ergäbe \$198C in D0. Binär läßt sich das als %0001100.1100.01100 darstellen. Die Längen der drei Felder haben wir durch einen Punkt markiert.

Beispiel:

MOVE	#\$2A, -(SP)	wir wollen uns das Datum holen
TRAP	#1	
ADDQ.L	#2, SP	
MOVE	D0, D1	Ergebnis in D1 zwischenspeichern
AND	#\$11111, D0	die Tages-Bits ausmaskieren
MOVE	D0, TAG	und abspeichern
LSR	#5, D1	die 5 Tages-Bits rausschieben
MOVE	D1, D0	
AND	#\$1111, D0	und die Monats-Bits maskieren
MOVE	D0, MONAT	die Monatsnummer speichern
LSR	#4, D1	und noch mal schieben
MOVE	D1, JAHR	das Jahr steht jetzt in D1
.		
TAG	DS.W	1
MONAT	DS.W	1
JAHR	DS.W	1

\$2B TSETDATE

C: Tsetdate(datum)
 int datum;

Assembler: MOVE.W #datum, -(SP)
 MOVE.W #\$2B, -(SP)
 TRAP #1
 ADDQ.L #4, SP

GFA-BASIC: VOID GEMDOS(43, datum%)

OMIKRON: GEMDOS(, 43, datum)

Auch aus Anwendungsprogrammen können Uhrzeit und Datum gestellt werden. Das ist besonders für solche Programme interessant, die Datum und/oder Uhrzeit verwenden. Ein Beispiel wäre die Rechnungsbearbeitung, in der das aktuelle Datum in den Rechnungsausdruck eingefügt wird. Solche Programme können dann selbsttätig den Benutzer zur Eingabe des Datums auffordern. Damit kann vermieden werden, daß der Anwender zu Beginn der Arbeit das Setzen von Datum und Uhrzeit im Kontrollfeld vergißt und mit dem nach dem Einschalten vorhandenen Wert arbeiten.

Das Datum muß der Funktion SET DATE im selben Format übergeben werden wie es von GET DATE geliefert wird, also Bit 0-4 = Tag, Bit 5-8 = Monat, Bit 9-15 =Jahr-1980.

Wenn Sie in Ihren Programmen von der GEMDOS-Funktion zum Setzen des Datums Gebrauch machen, so sollten Sie anschließend nur mit der GEMDOS-Funktion Tgetdate und nicht über die entsprechende XBIOS-Funktion das Datum ermitteln. In der ROM-Version 1.0 des TOS werden Änderungen der Zeit oder des Datums über die GEMDOS-Aufrufe nicht an das BIOS gemeldet. In der Mega-TOS-Version (ROM-Version 1.2) hat sich das zwar geändert, aber wer nicht nur für sich selbst programmiert, sollte sich immer auf dem kleinsten gemeinsamen Level bewegen, um allen ST-Benutzern eine korrekte Funktion des Programms zu garantieren.

Übrigens nimmt es das GEMDOS mit unserer Vorstellung von korrekten Daten nicht so genau. Der 31. April ist ihm ganz recht. Nur wirklich übertriebene Werte wie 32.15. werden durch einen Fehlercode \$FFFFFFF in D0 ange-meckert. Sonst ist D0 normalerweise 0.

Beispiel:

MOVE.W	#101101011001,-(SP)	Datum soll 25.10.1985 sein
MOVE.W	#\$2B,-(SP)	Funktionsnummer von SET DATE
TRAP	#1	Datum setzen
ADDQ.L	#4,SP	Stack reparieren

\$2C TGETTIME

C: int Tgettime()

Assembler:

MOVE.W	#\$2C,-(SP)
TRAP	#1
ADDQ.L	#2,SP

GFA-BASIC: time% = GEMDOS(44)

OMIKRON: GEMDOS(time,44)

Die Funktion GET TIME liefert die Zeit der vom GEMDOS mitgeführten Uhr. Ähnlich wie beim Datum wird die Uhrzeit, die in D0 nach dem Aufruf zur Verfügung steht, in einzelnen Bits des Registers nach einem speziellen Muster codiert. In den Bits 0 bis 4 werden die Sekunden dargestellt. Da aber mit fünf Bit nur der Bereich von 0 bis 31 dargestellt werden kann, läuft die interne Uhr in 2-Sekunden-Schritten. Um das richtige Sekunden-Ergebnis zu erhalten, muß also der Inhalt dieser untersten fünf Bits mit zwei multipliziert werden. In den Bits 5 bis 10 wird die Anzahl der Minuten festgehalten, während die restlichen Bits 11 bis 15 Auskunft darüber geben, welche Stunde (im 24-Stunden-Format) dem GEMDOS geschlagen hat.

Beispiel:

MOVE	#\$2C,-(SP)	Ob wohl schon Mittag ist?
------	-------------	---------------------------

TRAP	#1	dazu holen wir die Uhrzeit vom GEMDOS
ADDQ.L	#2,SP	
MOVE	D0,D1	Ergebnis in D1 zwischenspeichern
AND	##1111,D0	die Sekunden in Zweier-Schritten
MOVE	D0,SEK	und abspeichern
LSR	#4,D1	die 4 Sekunden-Bits rausschieben
MOVE	D1,D0	
AND	##111111,D0	und die Minuten-Bits maskieren
MOVE	D0,MIN	die Minuten speichern
LSR	#6,D1	und noch mal schieben
MOVE	D1,STD	die Stunde steht jetzt in D1
.		
SEK	DS.W	1
MIN	DS.W	1
STD	DS.W	1

\$2D TSETTIME

C: int Tsettime(zeit)
 int zeit;

Assembler: MOVE.W #zeit,-(SP)
 MOVE.W #\$2D,-(SP)
 TRAP #1
 ADDQ.L #4,SP

GFA-BASIC: VOID GEMDOS(45, zeit%)

OMIKRON: GEMDOS(,45,zeit)

Auch das Setzen der Uhrzeit ist unter GEMDOS einfach möglich. Die Funktion TSETTIME erwartet dazu auf dem Stack einen 16-Bit-Wert (Wort), in dem die Zeit in derselben Form codiert ist wie auch TGETTIME die Uhrzeit liefert. Wenn GEMDOS die übergebene Zeit mag, so erhält man in D0 den Wert 0, sonst den Wert \$FFFFFFF zurück. Im übrigen verhält es sich mit der GEMDOS-Zeit wie mit dem Datum. Änderungen der Zeit mittels GEMDOS werden in der ROM-Version 1.0 dem XBIOS nicht mitgeteilt. Aus diesem Grund sollte man bei der einmal gewählten Methode XBIOS oder GEMDOS treu bleiben, sonst erhält man leicht unschöne Komplikationen.

Beispiel: MOVE.W ##1000101010111101,-(SP) Uhrzeit 17:21:58
 MOVE.W #\$2D,-(SP) Funktionsnummer von SET TIME
 TRAP #1 Datum setzen
 ADDQ.L #4,SP Stack reparieren

\$2F FGETDTA

C: long Fgetdta()

Assembler: MOVE.W #\$2F, -(SP)
 TRAP #1
 ADDQ.L #2, SP

GFA-BASIC: dta_pointer% = GEMDOS(47)

OMIKRON: GEMDOS(dta, 47)

Ein Aufruf dieser Funktion ergibt die Adresse des derzeitigen DTA-Buffers in D0. Eine genauere Erläuterung dieses Buffers finden Sie weiter unten bei den Funktionen SEARCH FIRST und SEARCH NEXT.

Beispiel: MOVE #\$2F, -(SP) Funktionsnummer Fgetdta
 TRAP #1 DTA holen
 ADDQ.L #2, SP
 MOVE.L D0, DTAPOINTER und evtl. für später merken

\$30 SVERSION

C: int Sversion()

Assembler: MOVE.W #\$30, -(SP)
 TRAP #1
 ADDQ.L #2, SP

GFA-BASIC: version% = GEMDOS(48)

OMIKRON: GEMDOS(version, 48)

Ein Aufruf dieser Funktion übergibt in D0 die Versionsnummer des GEMDOS. Die erste Version des ROM-TOS sowie ältere TOS-Versionen auf Diskette liefern den Wert \$1300, das GEMDOS der MEGA-STs den Wert \$1400.

Beispiel: MOVE #\$30, -(SP) Mal sehen, welche Version wir
 TRAP #1 haben
 ADDQ.L #2, SP
 CMP #\$1300, D0 die allseits bekannte Version?
 BNE mega_tos

\$31 PTERMRES

C: void Ptermres(keepcnt, retcode)
 long keepcnt;
 int retcode;

Assembler: MOVE.W #retcode, -(SP)
 MOVE.L #keepcnt, -(SP)
 MOVE.W #\$31, -(SP)
 TRAP #1

GFA-BASIC: nicht anwendbar

OMIKRON: nicht anwendbar

Diese Funktion ist vergleichbar mit der GEMDOS-Funktion PTERM \$00. Auch nach Aufruf dieser Funktion wird das Programm beendet. Allerdings unterscheidet sich \$31 Ptermres() in einigen wesentlichen Punkten von \$00.

Nach Abarbeiten des TRAP #1 wird wie bei TERM die Kontrolle an das Programm zurückgegeben, welches das soeben beendete Programm gestartet hat. Im Unterschied zu TERM jedoch kann dem Aufrufer eine Abschluß-Bedingung mitgeteilt werden. Liefert TERM grundsätzlich den Abschlußwert Null (kein Fehler) so kann bei \$31 ein beliebiger Wert zwischen Null und \$FFFF als Abschlußwert gewählt werden. Ein von Null unterschiedlicher Wert signalisiert üblicherweise "Fehler bei der Programmabarbeitung".

Ein weiterer wesentlicher Punkt liegt in der Speicherverwaltung des GEMDOS begründet. Wird ein Programm (nennen wir es einmal PRG1) gestartet, so wird ihm der gesamte verfügbare Speicherplatz zur Verfügung gestellt. Wird PRG1 mit TERM beendet, so wird der Speicherplatz wieder freigegeben und GEMDOS zur Verfügung gestellt. Wird danach ein neues Programm (PRG2) gestartet, so wird unmittelbar nach dem Laden der gesamte noch freie Speicherbereich gelöscht, sprich mit Nullen gefüllt. PRG1 ist damit tatsächlich physikalisch aus dem Speicher verschwunden. Mit der Funktion \$31 jedoch kann ein beliebiger Speicherbereich ab der Start-Adresse des Programms (inklusive Basepage, siehe später) geschützt werden. Dieser Speicherbereich wird nach dem Beenden des Programms nicht freigegeben und auch nicht gelöscht. Das Programm könnte wieder gestartet werden, ohne es neu zu laden. Wesentliche Anwendungen für Ptermres() sind z.B. Spooler, RAM-Disk und andere Hilfen, die, einmal installiert, immer im Speicher verfügbar sein müssen, um die Daten in der entsprechenden Weise zu speichern oder zu verarbeiten. Gleichzeitig müssen aber solche Programme nach der Installation korrekt beendet werden, um andere Programme laden und starten zu können.

In diesem Zusammenhang muß auf eine Besonderheit des GEMDOS hingewiesen werden. Ist ein Programm erst einmal als residentes Programm installiert, so sollte man beim Anfordern von Speicherbereichen per Malloc oder beim Öffnen von Dateien (Fopen, Fcreate) besondere Vorsicht walten lassen. Diese Vorsicht ist auch für Accessories geboten. KEEP PROCESS wird mit zwei Parametern aufgerufen. Das Beispielprogramm zeigt die Parameterübergabe. Wichtig ist auch, daß zusätzlich vom Programm per Malloc reservierter Speicher nicht freigegeben wird. Sollten jedoch zum Zeitpunkt des Ptermres() Dateien geöffnet sein, so werden diese vom GEMDOS geschlossen.

Beispiel:

MOVE.W	#0,-(SP)	Fehlercode kein Fehler, sonst 1
MOVE.L	#\$1000,-(SP)	\$1000 Bytes ab Base Page Anfang schützen
MOVE.W	#\$31,-(SP)	Programm endet ...
TRAP	#1	... jetzt!
.		nicht mal den Stack aufräumen!

\$36 DFREE

C:

void	Dfree(buffer,drive)
long	*buffer
int	drive

Assembler:

MOVE.W	#drive,-(SP)
PEA	buffer
MOVE.W	#\$36,-(SP)
TRAP	#1
ADDQ.L	#8,SP

GFA-BASIC: status% = GEMDOS(54, L:buffer_address%, drive%)

OMIKRON: GEMDOS(status,54,HIGH(buffer),LOW(buffer),drive)

Für diskettenorientierte Programme kann es sehr wichtig sein, die Größe des freien Platzes auf der Diskette zu ermitteln. Damit hat man die Möglichkeit, den Anwender zum rechtzeitigen Diskettenwechsel aufzufordern. "Disk Full"-Fehlermeldungen oder gar Datenverluste können dann sicher verhindert werden.

Genau diese Information liefert die Funktion \$36 Dfree(). Dazu muß die Nummer des gewünschten Laufwerks und die Adresse eines 16 Byte großen Buffers an die Funktion übergeben werden. Wird als Laufwerksnummer der Wert 0 übergeben, so wird die Information vom aktiven Laufwerk geholt, bei einer 1 von Drive A, bei 2 von Drive B. Die im Buffer übergebene Information teilt sich in 4 Langwörter auf. Das erste Langwort enthält die Anzahl noch freier

sogenannter Allocation Units oder Cluster. Jede Datei, und sei sie auch nur acht Bytes groß, benötigt mindestens eine solche Allocation Unit.

Das zweite Langwort gibt Aufschluß über die Anzahl der auf der Diskette vorhandenen Cluster, unabhängig, ob sie bereits belegt oder noch frei sind. Bei der "kleinen" einseitigen Diskette beträgt dieser Wert \$15F oder 351, die doppelseitigen Disketten haben \$2C7 = 711 Cluster. In der Zwischenzeit gibt es verschiedene spezielle Formaterroutinen, die bis zu elf Sektoren pro Track und bis zu 86 Tracks pro Diskette schaffen. Dadurch erhöht sich natürlich auch die Anzahl der Cluster pro Diskette. Die angegebenen Werte beziehen sich nur auf die "Standard-Formatierung".

Das dritte Langwort enthält die Größe eines Diskettensektors in Byte. Im letzten Langwort schließlich wird die Anzahl der zu einem Cluster gehörenden physikalischen Sektoren vermerkt. Hier ist normalerweise eine 2 eingetragen. Zwei Sektoren mit je 512 Bytes bilden also einen Cluster (Allocation Unit) von insgesamt 1024 Bytes. Aus den so gewonnenen Informationen kann nun ohne Schwierigkeiten die Anzahl der noch verfügbaren Bytes berechnet werden. Diese Funktion ist unglaublich langsam, auch wenn das bei Benutzung eines Diskettenlaufwerkes nicht so ins Auge fällt. Auf einer 20-MByte-Harddisk aber kann man bis zu 20 Sekunden warten, bis das Ergebnis geliefert wird.

Beispiel:

MOVE.W	#0, -(SP)	Information vom aktiven Drive
PEA	BUFFER	Adresse des 16-Byte Buffers
MOVE	#\$36, -(SP)	Funktionsnummer
TRAP	#1	
ADDQ.L	#8, SP	Stack aufräumen
.		
.		
.		

BUFFER:

freal: .ds.l	1	Free Clusters
total: .ds.l	1	Total Clusters
bps: .ds.l	1	Bytes/physical Sector
pspal: .ds.l	1	phys. Sectors/Cluster

\$39 DCREATE

C:

int	Dcreate(path)
char	*path;

Assembler:

PEA	path
MOVE.W	#\$39, -(SP)
TRAP	#1
ADDQ.L	#6, SP

GFA-BASIC: error% = GEMDOS(57, L:path_address%)

OMIKRON: GEMDOS(error,57,HIGH(path),LOW(path))

Vom Desktop aus kann ein Unter-Directory mit dem Punkt "NEUER ORDNER" angelegt werden. Mit dem Aufruf Dcreate \$39 kann ein solches Unter-Directory auch von Anwenderprogrammen heraus angelegt werden. Mit Hilfe solcher Unter-Directories oder Ordner kann man bei ein wenig Planung die Diskette recht übersichtlich organisieren.

Um einen neuen Ordner zu schaffen, wird der Funktion \$39 die Adresse des Ordner-Namens, auch als Pfadname (engl. Pathname) bezeichnet, übergeben. Dieser Name darf aus acht Zeichen und drei Zeichen Extension bestehen. Damit gilt für den Pfadnamen die gleiche Einschränkung wie für File-Namen. Er muß beim Aufruf von DCREATE mit einem Null-Byte abgeschlossen sein.

Nach dem Aufruf wird in D0 mitgeteilt, ob die Operation erfolgreich abgeschlossen wurde. Ist in D0 eine Null enthalten, so war der Aufruf erfolgreich. Fehler werden durch eine negative Nummer in D0 gekennzeichnet. Eine Übersicht über alle im Zusammenhang mit GEMDOS-Funktionen auftretenden Fehlermeldungen finden Sie zum Schluß dieses Kapitels.

Beispiel:	PEA	pathname	Adresse des Pfadnamens
	MOVE	#\$39, -(SP)	Funktionsnummer
	TRAP	#1	
	ADDQ.L	#6, SP	Stack reparieren
	TST.W	D0	Fehler aufgetreten?
	BNE	Fehler	es scheint so

```
pathname:
        .dc.b      'privat.dat',0
```

\$3A DDELETE

C:	int	Ddelete(path)
	char	*path;

Assembler:	PEA	path
	MOVE.W	#\$3A, -(SP)
	TRAP	#1
	ADDQ.L	#6, SP

GFA-BASIC: error% = GEMDOS(58, L:path_address%)

OMIKRON: GEMDOS(error,58,HIGH(path),LOW(path))

Ein mit MKDIR erzeugtes Subdirectory kann mit \$3A auch wieder gelöscht werden. Wie zuvor wird dazu der mit Null abgeschlossene Pfadname an RMDIR übergeben. Und auch die Fehlermeldungen sind entsprechend MKDIR Null bei Erfolg oder ein negativer Wert bei Fehlern. Eine wichtige Fehlermeldung soll an dieser Stelle schon einmal erwähnt werden. Es ist dies die Meldung -36 (\$FFFFFFCA). Diese Fehlermeldung erhalten Sie, wenn das zu löschende Subdirectory Files enthält.

Man kann mit RMDIR also nur leere Subdirectories löschen. Im Falle der beschriebenen Fehlermeldung muß man dann zunächst mit UNLINK (\$41) alle Files im Subdirectory löschen und danach einen neuen Anlauf nehmen.

Beispiel:	PEA	pathname	Adresse des Pfadnamens
	MOVE	#\$3A,-(SP)	Funktionsnummer
	TRAP	#1	
	ADDQ.L	#6,SP	Stack reparieren
	TST.W	D0	Fehler aufgetreten?
	BNE	era_sub_dir	wahrscheinlich erst mal putzen

```
pathname:
        .dc.b      'tmpfiles.a_z',0
```

\$3B DSETPATH

C:	int	Dsetpath(path)
	char	*path;

Assembler:	PEA	path
	MOVE.W	#\$3B,-(SP)
	TRAP	#1
	AQQD.L	#6,SP

GFA-BASIC: error% = GEMDOS(59, L:path_address%)

OMIKRON: GEMDOS(error,59,HIGH(path),LOW(path))

Das unter GEMDOS verfügbare System der Subdirectories ist in genau derselben Form auch unter UNIX verfügbar.

Man kann das durch den Pfad spezifizierte Subdirectory zum aktuellen Directory machen, indem man die Funktion DSETPATH aufruft. Danach kann

auf alle Files im ausgewählten Subdirectory nur über den File-Namen zugegriffen werden. Der Pfad ist durch die Funktion festgelegt.

Beispiel:

	PEA	path	Adresse des Pfads
MOVE.W	#\$3B, -(SP)		Funktionsnummer
TRAP	#1		
ADDQ.L	#6, SP		wie immer Stack reparieren
TST.W	D0		Fehler aufgetreten?
BNE	Fehler		es scheint so

path:

.dc.b 'c:\hugos.dat\privat\geheim\secret.top\' ,0

\$3C FCREATE

C:

int	Fcreate(fname,attr)
char	*fname;
int	attr;

Assembler:

MOVE.W	#ATTRIB, -(SP)
PEA	fname
MOVE.W	#\$3C, -(SP)
TRAP	#1
ADDQ.L	#8, SP

GFA-BASIC: handle% = GEMDOS(60, L:fname_adr%, attr%)

OMIKRON: GEMDOS(handle,60,HIGH(fname),LOW(fname),attr)

In allen Betriebssystemen läuft der Zugriff auf Dateien über die Sequenz Eröffnen der Datei, Zugriff auf die Daten, also Lesen oder Schreiben von Daten und abschließendem Schließen der Datei. Diese "Dreieinigkeit" existiert auch unter GEMDOS, allerdings gibt es eine Ausnahme. Unter CP/M z.B. kann auch eine nicht vorhandene Datei eröffnet werden. Mit dem Eröffnen wird dann die bisher nicht vorhandene Datei eingerichtet. Unter GEMDOS jedoch muß eine Datei zuerst einmal kreiert, also geschaffen werden. Dazu dient der Aufruf \$3C, Fcreate(). Diesem GEMDOS-Aufruf werden zwei Parameter übergeben, die Adresse des gewünschten File-Namens und ein Attribut-Wort.

Wird als Attribut-Wort Null übergeben, so wird eine ganz normale Datei eingerichtet, die sowohl beschrieben als auch gelesen werden kann. Wird als Attribut der Wert 1 übergeben, so kann die Datei nach dem Schließen nur noch

gelesen werden. Sie ist softwaremäßig schreibgeschützt, was natürlich nicht verhindern kann, daß die Datei nach einem Formatieren der Diskette verschwunden ist.

Weitere mögliche Attribute sind \$02, \$04 und \$08. Das Attribut \$02 erzeugt eine "versteckte" Datei und das Attribut \$04 eine "versteckte" Systemdatei. Das Attribut \$08 schließlich erzeugt eine Datei mit einem "Volume-Label". Das Volume-Label ist der (optionale) Name, der einer Diskette beim Formatieren gegeben werden kann. Aus den maximal acht Bytes des Namens und der Extension wird dann der Diskettenname erzeugt. Dateien mit einem der letzten drei Attribute sind von der normalen Directory-Suche im DESKTOP ausgenommen. Sie erscheinen jedoch im Directory z.B. des Programms COM-MAND.PRG.

Wenn die Funktion FCREATE beendet ist, so wird im Register D0 ein File-Descriptor, auch File-Handle genannt, zurückgegeben. Über diese File-Handle (ein numerischer Wert zwischen 6 und 45) gehen alle weiteren Zugriffe auf die Datei. Beim Schreiben, Lesen und Schließen von Dateien muß das Handle angegeben werden. Insgesamt können also maximal $28 = 40$ Dateien gleichzeitig geöffnet werden. Wenn beim FCREATE-Aufruf bereits eine Datei mit diesem Namen existiert, so wird diese auf Null-Länge abgeschnitten. Das ist gleichbedeutend mit der Sequenz Löschen der alten Datei und Erstellen einer neuen, gleichnamigen Datei, geht aber sehr viel schneller. Erhalten Sie nach dem FCREATE-Aufruf eine Handle-Nummer in D0 zurück, so muß die Datei nicht noch zusätzlich mit \$3D FOPEN eröffnet werden.

Beispiel:	MOVE.W	#\$0,-(SP)	Datei soll R/W-Status erhalten
	PEA	File-Name	Adresse des File-Namens auf den Stack
	MOVE.W	#\$3C,-(SP)	Fcreate-Funktionsnummer
	TRAP	#1	GEMDOS aufrufen
	ADDQ.L	#8,SP	das übliche
	TST	D0	Fehler aufgetreten?
	BMI	error	scheint so
	MOVE	D0,handle	File-Handle für spätere Zugriffe merken.

File-Name:		Null-Byte nicht vergessen !
	.dc.b	'myfile.dat',0
handle:		
	.ds.w	1

\$3D FOPEN

C: int Fopen(fname,mode)
 char *fname;
 int mode;

Assembler: MOVE.W #mode,-(SP)
 PEA fname
 TRAP #1
 ADDQ.L #6,SP

GFA-BASIC: handle% = GEMDOS(61, L:Varptr(fname\$), mode%)

OMIKRON: GEMDOS(handle,61,HIGH(fname\$),LOW(fname),mode)

Mit FCREATE kann man nur neue Dateien erstellen oder bestehende auf Null-Länge kürzen. Bestehende Dateien müssen aber auch weiter verarbeitet werden können. Dazu müssen solche Dateien mit der FOPEN-Funktion eröffnet werden.

Der erste Parameter der FOPEN-Funktion ist das Modus-Wort. Bei einer Null im Modus-Wort kann die geöffnete Datei nur gelesen, bei einer 1 nur beschrieben werden. Mit einem Wert von 2 kann die Datei gleichermaßen gelesen wie beschrieben werden. Als zweiter Parameter wird in der schon bekannten Weise die Adresse des durch ein Null-Byte abgeschlossenen Dateinamens übergeben. Als Ergebnis der FOPEN-Funktion wird in D0 die Handle-Nummer übergeben, wenn die Datei vorhanden und die gewünschte Zugriffsart möglich ist. Sonst erhält man in D0 eine Fehlernummer. Auf die Aufstellung der möglichen Fehlernummern am Ende des Kapitels wird noch einmal hingewiesen. Haben wir bisher in Verbindung mit den Datei-Funktionen immer nur von Dateien gesprochen, so war das zwar korrekt, aber eben nur die halbe Wahrheit. Neben den Dateien können auch noch drei sogenannte Devices eröffnet werden. Diese Devices oder Geräte sind die Konsole, also Tastatur und Monitor, die serielle Schnittstelle und der Druckeranschluß. Wir wollen hier nur zeigen, wie ein Device geöffnet wird und welche Handle Sie erhalten. Diese Information ist insofern wichtig, da sich die Device-Handles von den File-Handles unterscheiden.

Um ein Device zu eröffnen, wird der entsprechende Device-Name als File-Name angegeben. Die Device-Namen lauten CON: für die Console, AUX: für die serielle Schnittstelle und PRN: für die Druckerschnittstelle. Nach einem FOPEN mit einem der angegebenen Namen erhalten Sie eine Word-negative Handle zurück. \$FFFF (-1) wird zurückgegeben für CON:, \$FFFE (-2) erhalten Sie bei Angabe von AUX: und \$FFFD (-3) ist die Handle für den Printerport.

Beispiel:

MOVE.W	#\$2,-(SP)	Datei soll gelesen und beschrieben werden
PEA	File-Name	Adresse des File-Namens auf den Stack
MOVE.W	#\$3D,-(SP)	Funktionsnummer
TRAP	#1	GEMDOS aufrufen
ADDQ.L	#8,SP	Stack aufräumen
TST.L	D0	Fehler aufgetreten?
BMI	fopnerr	scheint so
MOVE	D0,fhandle	File-Handle für spätere Zugriffe merken.

File-Name: Null-Byte nicht vergessen !

.dc.b 'myfile.dat',0

fhandle:

.ds.w 1

\$3E FCLOSE

C:

int	Fclose(handle)
int	handle;

Assembler:

MOVE.W	#handle,-(SP)
MOVE.W	#\$3E,-(SP)
TRAP	#1
ADDQ.L	#4,SP

GFA-BASIC: error% = GEMDOS(62, handle%)

OMIKRON: GEMDOS(error,62,handle)

Jede geöffnete Datei sollte geschlossen werden, wenn innerhalb des Programms nicht mehr auf die Datei zugegriffen werden muß oder wenn das Programm selbst beendet wird. Besonders zum Schreiben eröffnete Dateien sollten Sie vor Programmende schließen. Zwar werden normalerweise automatisch alle noch offenen Dateien beim Programmende geschlossen. Allerdings ist es immer besser, sich nicht auf diesen Mechanismus zu verlassen.

Geschlossen werden Dateien mit dem Aufruf FCLOSE, dem die Handle-Nummer als Parameter übergeben wird. Als Return-Wert erhält man den Wert Null, wenn die Datei korrekt geschlossen worden ist.

Beispiel:

MOVE.W	handle,-(SP)	hier haben wir die Handle-Nummer abgelegt
MOVE.W	#\$3E,-(SP)	Funktionsnummer

TRAP	#1	GEMDOS aufrufen
ADDQ.L	#4,SP	Stack aufräumen
TST.W	D0	Fehler aufgetreten?
BMI	error	scheint so

handle:

.ds.w 1

\$3F FREAD

C:

long	Fread(handle, count, buff)
int	handle;
long	count;
char	*buff;

Assembler:

PEA	buff
MOVE.L	#count, -(SP)
MOVE.W	handle, (-SP)
MOVE.W	#3F, -(SP)
TRAP	#1
ADDQ.L	#12, SP

GFA-BASIC: rcnt% = GEMDOS(63, handle%, L:count%, L:Varptr(buff))

OMIKRON: GEMDOS(rcnt, 63, HIGH(count), LOW(count), HIGH(buffer), LOW(buffer))

Das Öffnen und Schließen von Dateien ist natürlich nur die halbe Miete, es soll ja auch mit Daten gearbeitet werden. So müssen z.B. Briefe, Lohn- und Gehaltslisten, Meßwerte und Artikelnummern gespeichert und später wieder gelesen werden können. Das Lesen solcher Daten kann in sehr eleganter Weise mit der Funktion FREAD geschehen. FREAD bekommt drei Parameter übermittelt: zuerst die Adresse eines Buffers, in den die Daten gelesen werden sollen, dann die Anzahl der aus der Datei zu lesenden Bytes und zum Schluß die Handle-Nummer der Datei. Diese Nummer haben Sie (hoffentlich) beim zuvor erfolgten FOPEN erhalten.

Als Rückgabewert erhalten Sie in D0 entweder eine Fehlernummer oder die Anzahl der fehlerfrei gelesenen Bytes. Wird versucht, über das Dateiende hinaus zu lesen, so ist der Wert in D0, die tatsächlich gelesene Anzahl Bytes, von der gewünschten Anzahl unterschiedlich. Danach erfolgende Lesezugriffe liefern als Rückgabewert eine Null. Dann ist das Ende der Datei erreicht.

Eine weitere Möglichkeit wäre es, sich vor dem Lesen die Größe der Datei in Bytes mittels SEARCH FIRST/SEARCH NEXT zu holen und den angegebenen

Betrag zu lesen. Das Lesen und Schreiben von Daten geschieht im GEMDOS übrigens völlig ungepuffert. Das bedeutet, daß beim Lesen von einzelnen Bytes tatsächlich für jedes der 512 Bytes eines Sektors der komplette Sektor gelesen wird. Besser und vor allem wesentlich schneller ist es, einen kompletten Sektor zunächst in einen Buffer zu lesen und sich die Daten anschließend byteweise aus dem Puffer zu holen. Der Geschwindigkeitszuwachs ist teilweise verblüffend!

Beispiel:

PEA	buffer	Adresse des Daten-Buffers
MOVE.L	#\$200,-(SP)	512 Bytes (ein Sektor) lesen
MOVE.W	handle,-(SP)	Platz für die Handle-Nummer
MOVE.W	#\$3F,-(SP)	Funktionsnummer
TRAP	#1	
ADD.L	#12,SP	
TST.L	D0	ist ein Fehler aufgetreten?
BMI	error	scheint so
CMP.L	#\$200,D0	tatsächlich 512 Bytes gelesen?
BNE	end_of_file	waren nicht mehr genug Daten im File

handle:		
	.ds.w	1 Platz für die Handle-Nummer
buffer:		
	.ds.w	\$100 256 Words = 512 Byte reicht in unserem Beispiel

\$40 FWRITE

C:

long	Fwrite(handle, count, buff)
int	handle;
long	count;
char	*buff;

Assembler:

PEA	buf
MOVE.L	#count,-(SP)
MOVE.W	handle,-(SP)
MOVE.W	#\$40,-(SP)
TRAP	#1
ADDQ.L	12,SP

GFA-BASIC: wcnt% = GEMDOS(64, handle%, L:count, L:Varptr(buf\$))

OMIKRON: GEMDOS(wcnt,64,handle,HIGH(count),LOW(count),HIGH(buffer),LOW(buffer))

Genau so einfach wie das Lesen von Dateien ist das Schreiben. Auch die übergebenen Parameter sind mit den beim Lesen benötigten Parametern identisch. Als Handle müssen von FOPEN- und FCREATE-Aufrufen gelieferte Handle-Nummern benutzt werden.

Beispiel:

PEA	buffer	Adresse des Daten-Buffers
MOVE.L	#\$200,-(SP)	jetzt schreiben wir einmal 512 Bytes
MOVE.W	handle,-(SP)	Platz für die Handle-Nummer
MOVE.W	#\$40,-(SP)	jetzt ist WRITE gefragt
TRAP	#1	
ADD.L	#12,SP	
TST.L	D0	ist ein Fehler aufgetreten?
BMI	error	das ist ärgerlich
.		
.		
handle:		
	.ds.w	1 Platz für die Handle-Nummer
.		
buffer:		
	.ds.b	\$200 reicht in unserem Beispiel

\$41 FDELETE

C:

int	Fdelete(fname)
char	*fname;

Assembler:

PEA	fname
MOVE.W	#\$41,-(SP)
TRAP	#1
ADDQ.L	#6,SP

GFA-BASIC: error% = GEMDOS(65, L:Varptr(fname\$))

OMIKRON: GEMDOS(error,65,HIGH(fname),LOW(fname))

Mit Fdelete, auch als Unlink bezeichnet, können nicht mehr benötigte Dateien gelöscht werden. Dazu muß dem Aufruf die Adresse des File-Namens oder bei Bedarf auch des vollständigen Pfadnamens übergeben werden. Enthält das Register D0 nach dem Aufruf eine Null, so ist die Datei gelöscht worden. Sonst erhält man eine Fehlernummer in D0 zurück.

Beispiel:

PEA	fname	Name des zu löschenden Files
MOVE.W	#\$41,-(SP)	Funktionsnummer Fdelete()
TRAP	#1	
ADD.L	#6,SP	
TST.L	D0	ist ein Fehler aufgetreten?

```

        BMI            error            das File gibt's nicht oder
        .              .                schlimmeres
fname:
        .dc.b          'b:\rolli\privat\pakman.prg',0

```

\$42 FSEEK

```

C:      long      Fseek(offset, handle, seekmode)
        long      offset;
        int       handle;
        int       seekmode;

```

```

Assembler: MOVE.W      #seekmode, -(SP)
               MOVE.W      handle, -(SP)
               MOVE.L      offset, -(SP)
               MOVE.W      #$42, -(SP)
               TRAP         #1
               ADDQ.L      10, SP

```

GFA-BASIC: error% = GEMDOS(66, L:offset%, handle%, seekmode%)

OMIKRON: GEMDOS(error, 66, HIGH(offset), LOW(offset), handle, seekmode)

FSEEK bietet eine außerordentlich komfortable Möglichkeit, den Datei-Pointer auf jedes beliebige Byte innerhalb der Datei zu stellen und ab dieser Stelle zu lesen oder zu schreiben. Diese UNIX-kompatible Positionierung innerhalb von Dateien ist um vieles komfortabler als die z.B. von CP/M bekannten Möglichkeiten zur block-orientierten Dateiverwaltung.

Insgesamt werden der FSEEK-Funktion drei Parameter übergeben. Der erste Parameter gibt die Anzahl der Bytes an, um die der Pointer bewegt werden soll. Ein weiterer Parameter ist die Handle-Nummer der Datei. Der letzte Parameter ist ein Modus-Wort, das beschreibt, wie der Datei-Pointer zu bewegen ist. Eine Null als Modus bewegt den Zeiger an den Anfang und von da die angegebene Zahl Bytes zum Ende der Datei. Als Anzahl dürfen nur positive Werte gewählt werden. Bei einem Modus-Wert von 1 wird der Zeiger um den gewünschten positiven oder auch negativen Betrag ab der aktuellen Position bewegt, und bei einer 2 gilt die Entfernungsangabe ab dem Ende der Datei. Bei einer 2 als Moduswert sind nur negative Werte zugelassen. Nach dem Aufruf erhält man in D0 die absolute Position des Zeigers ab Dateianfang oder eine Fehlermeldung.

Beispiel:

MOVE.W	#1,-(SP)	relativ vom derzeitigen Filepointer
MOVE.W	handle,-(SP)	File-Handle
MOVE.L	#\$-20,-(SP)	32 Bytes zurück
MOVE.W	#\$42,-(SP)	Funktionsnummer
TRAP	#1	
ADD.L	#10,SP	
TST.W	D0	ist ein Fehler aufgetreten?
BMI	error	scheint so

handle:

.ds.w	1	Platz für die Handle-Nummer
-------	---	-----------------------------

\$43 FATTRIB

C:

int	Fattrib(fname, flag, attrib)
char	*fname;
int	flag;
int	attrib;

Assembler:

MOVE.W	attrib,-(SP)
MOVE.W	flag,-(SP)
PEA	fname
MOVE.W	#\$43,-(SP)
TRAP	#1
ADDQ.L	#10,SP

GFA-BASIC: error% = GEMDOS(67, L:Varptr(fnam\$), flag%, attrib%)

OMIKRON: GEMDOS(error,67,HIGH(fname),LOW(fname),flag,attrib)

Mit der Funktion FCREATE kann man einer Datei ein bestimmtes Attribut zuordnen. Dieses Attribut kann nur mit der Funktion FATTRIB oder CHANGE MODE (kurz CHMOD) ermittelt und gegebenenfalls geändert werden. Dazu muß der Name oder ggf. der Pfadname der Datei als Parameter an CHMOD übergeben werden. Ein weiteres Parameter-Wort gibt an, ob die Datei-Attribute ermittelt oder gesetzt werden sollen. Außerdem muß ein Wort übergeben werden, in dem die neu zu setzenden Attribute enthalten sind. Beim Ermitteln der Attribute wird dieses Wort zwar nicht benötigt, sollte aber trotzdem übergeben werden. Die möglichen Datei-Attribute haben wir zum Teil bereits bei der Funktion FCREATE gezeigt, sie sollen trotzdem noch einmal in tabellarischer Form aufgeführt werden.

\$00	normaler Datei-Status, lesen/schreiben möglich
\$01	READ-ONLY, Datei kann nur gelesen werden

- \$02 HIDDEN, "verborgenes" File
- \$04 System-File
- \$08 Volume-Label, enthält den Diskettenamen
- \$10 Subdirectory
- \$20 Archiv-Bit, Datei wurde beschrieben und korrekt geschlossen.

Die Attribute \$10 und \$20 können beim Anlegen einer Datei nicht angegeben werden. Das Attribut \$20 wird vom Betriebssystem vergeben, während für das Erstellen von Subdirectories vorteilhaft die GEMDOS-Funktion DCREATE verwendet wird. Die DCREATE-Funktion erzeugt nicht nur den Directory-Eintrag mit dem entsprechenden Attribut, sondern richtet das Subdirectory physikalisch auf der Diskette ein.

Grundsätzlich erhält man nach dem Aufruf in D0 den aktuellen Attributwert zurück, nach dem Setzen des Attributs also auch den neuen Wert, oder aber, wie bei allen anderen Funktionsaufrufen, eine negative Fehlermeldung.

1. Beispiel

MOVE.W	#1,-(SP)	Datei soll READ-ONLY-Attribut erhalten
MOVE.W	#1,-(SP)	Kennzeichen-Attribut setzen.
PEA	pathname	den Pfadnamen benötigen wir auch
MOVE.W	#\$43,-(SP)	Funktionsnummer
TRAP	#1	
ADD.L	#10,SP	
TST.W	D0	ist ein Fehler aufgetreten?
BMI	error	scheint so
.		
.		
.		
pathname:		Null-Byte am Ende nicht vergessen!
.dc.b		'killme.not',0

2. Beispiel

MOVE.W	#0,-(SP)	Dummy-Wert, wird nicht zwingend benötigt
MOVE.W	#0,-(SP)	Kennzeichen-Attribut ermitteln
PEA	pathname	und den Pfadnamen
MOVE.W	#\$43,-(SP)	Funktionsnummer
TRAP	#1	
ADD.L	#10,SP	
TST.W	D0	ist ein Fehler aufgetreten?
BMI	error	scheint so
.		
.		
.		
pathname:		Null-Byte am Ende nicht vergessen!
.dc.b		'was_bin.ich',0

\$45 FDUP

```
C:      int      Fdup(handle)
        int      handle;
```

```
Assembler:  MOVE.W      handle,-(SP)
              MOVE.W      #$45,-(SP)
              TRAP         #1
              ADDQ.L       #4,SP
```

GFA-BASIC: error% = GEMDOS(69, handle%)

OMIKRON: GEMDOS(error,69,handle)

Wie bei den Funktionen FREAD und FWRITE erwähnt stehen dem Programmierer auch die Devices Console, Lineprinter und RS232 wie Dateien zur Verfügung. Dadurch lassen sich Ein- und Ausgaben von Dateien auch auf diese Devices umleiten. Mit der DUP-Funktion kann einem der Standard-Handles eine File-Handle-Nummer zugewiesen werden. Nach dem Aufruf erhält man die nächste freie Handle-Nummer zurück.

Beispiel:	MOVE.W	stdh,-(SP)	Parameter ist ein Standard-Handle 0 - 5
	MOVE.W	#\$45,-(SP)	DUP ausführen
	TRAP	#1	
	ADDQ.L	#4,SP	
	TST.L	D0	möglich sind -37, -35 oder 0
	BNE	duperr	
	MOVE	D0,nstdh	Ergebnis ist Non-Standard-Handle (6 - 45)

\$46 FFORCE

```
C:      int      Fforce(stdh,nonstdh)
        int      stdh;
        int      nonstdh;
```

```
Assembler:  MOVE.W      nonstdh, -(SP)
              MOVE.W      stdh, -(SP)
              MOVE.W      #$46, -(SP)
              TRAP         #1
              ADDQ.L       #6, SP
```

GFA-BASIC: error% = GEMDOS(70, stdh%, nonstdh%)

OMIKRON: GEMDOS(error;70,stdh,nonstdh)

Die FFORCE-Funktion erlaubt eine noch weitergehende Beeinflussung der Handle-Nummern. Wird in einem Programm die Ein-/Ausgabe der Konsole ausschließlich über die FREAD- und FWRITE-Funktionen mit den Handle-Nummern 0 und 1 vorgenommen, so kann mit diesem Aufruf die Ein- bzw. Ausgabe umgeleitet werden. Bildschirmausgaben werden in eine Datei geschrieben, Eingaben werden nicht von der Tastatur, sondern aus einer zuvor eröffneten Datei geholt.

Beispiel:

MOVE.W	nstdh,-(SP)	Parameter ist Non-Standard-Handle
MOVE.W	stdh,-(SP)	Standard-Handle (0 - 5)
MOVE.W	#\$46,-(SP)	FORCE ausführen
TRAP	#1	
ADDQ.L	#6,SP	
TST.L	D0	möglich sind -37 oder 0
BNE	force_err	
.		
.		

\$47 DGETPATH

C:

void	Dgetpath(buf, drive)
char	*buf;
int	drive;

Assembler:

MOVE.W	drive,-(SP)
PEA	buf
MOVE.W	#\$47,-(SP)
TRAP	#1
ADDQ.L	#8,SP

GFA-BASIC: error% = GEMDOS(71, L:Varptr(buf\$), drive%)

OMIKRON: GEMDOS(error,71,HIGH(buf),LOW(buf),drive)

Mit der Funktion \$3B kann ein beliebiges Subdirectory zum aktuellen Directory ernannt werden. Alle Dateizugriffe ohne Pfadnamen laufen dann nur im eingestellten Subdirectory ab. Allerdings kann es unter bestimmten Voraussetzungen wesentlich sein, den derzeitigen Pfadnamen zum aktuellen Subdirectory zu ermitteln. Diese Arbeit erledigt der Funktionsaufruf DGETPATH \$47. Als Parameter benötigt dieser Aufruf die Bezeichnung des gewünschten Laufwerks (0=aktuelles Drive, 1=Drive A, 2=Drive B usw.) und die Adresse

eines Buffers, der mindestens 64 Bytes groß sein muß. In diesem Buffer wird der komplette Pfadname zum aktuellen Directory abgelegt. Der Pfadname wird durch ein Null-Byte abgeschlossen. Wird die Funktion aufgerufen, wenn das Haupt-Directory aktiv ist, so wird kein Pfadname übergeben. In diesem Fall enthält gleich das erste Byte des Buffers Null. Nach dem Aufruf muß D0 den Wert Null enthalten. Bei negativen Werten ist ein Fehler aufgetreten, z.B. dann, wenn eine falsche Laufwerksnummer angegeben wurde.

Beispiel:

MOVE.W	#0,-(SP)	Pfadnamen vom aktuellen Drive holen
PEA	buffer	Adresse des 64-Byte-Buffers
MOVE.W	#\$47,-(SP)	Funktionsnummer
TRAP	#1	
ADDQ.L	#8,SP	
.		
.		
buffer:		
.ds.b	128	sicher ist sicher

\$48 MALLOC

C:

long	Malloc(anzahl)
long	anzahl;

Assembler:

MOVE.L	anzahl,-(SP)
MOVE.W	#\$48,-(SP)
TRAP	#1
ADDQ.L	#6,SP

GFA-BASIC: address% = GEMDOS(72, L:anzahl%)

OMIKRON: GEMDOS(address,72,HIGH(anzahl),LOW(anzahl))

Die MALLOC-Funktion und die zwei folgenden Aufrufe MFREE und SET-BLOCK befassen sich mit der Speicherorganisation des GEMDOS. Wie bereits bei der Funktion \$31, KEEP PROCESS, erwähnt, wird einem Programm nach dem Laden der gesamte verfügbare Speicherbereich zugeordnet. Das ist in vielen Fällen unkritisch, da nur ein einziges Programm laufen soll.

Es ist aber gerade unter GEM wesentlich, zumindest einen Teil des Speichers nach dem Start des Programms wieder freizugeben, da verschiedene GEM-Funktionen Speicher mit der MALLOC-Funktion anfordern. Ein gutes Beispiel ist der Fileselector, der auch bei korrekter Programmierung nicht erscheint, wenn kein freier Speicher mehr zur Verfügung steht.

Andere Anwendungen sind z.B. Programme, die mit Overlays arbeiten. Um ein Overlay von der Diskette laden zu können, muß dem GEMDOS freier Speicher zur Verfügung stehen. Aus diesem Grunde sollte jedes Programm immer nur den Platz reservieren, den es tatsächlich als Programm- und Datenbereich benötigt. Der nicht benötigte Speicher kann dann an GEMDOS mittels SETBLOCK zurückgegeben werden.

Sollte ein Programm jedoch nach der Freigabe des nicht benötigten Speichers doch weiteren Speicher benötigen, so kann es den Speicher vom GEMDOS über die Funktion MALLOC (Memory Allocate) anfordern. Dazu wird mit der Funktion MALLOC die Anzahl der benötigten Bytes mitgeteilt. Als Return-Wert in D0 erhält man die Startadresse des durch den Aufruf reservierten Speicherbereichs oder eine Fehlermeldung, wenn versucht wird, mehr Speicher zu reservieren als überhaupt noch frei ist. Wird als Anzahl der zu allozierenden Bytes -1L übergeben, so erhält man die Größe des größten zusammenhängenden freien Speicherbereiches in Bytes in D0 zurück.

1. Beispiel	MOVE.L	#-1,-(SP)	Anzahl freier Bytes ermitteln
	MOVE.W	#\$48,-(SP)	Funktionsnummer
	TRAP	#1	
	ADDQ.L	#6,SP	Anzahl freier Bytes in D0
	.		
	.		
2. Beispiel	MOVE.L	#\$1000,-(SP)	Hex 1000 Bytes dem Programm zuteilen
	MOVE.W	#\$48,-(SP)	Funktionsnummer
	TRAP	#1	
	ADDQ.L	#6,SP	
	TST.W	D0	Fehler oder Adresse des Speichers?
	BMI	error	negative Long-Werte = Fehler!
	MOVE.L	D0,mstart	sonst Start-Adresse des reservierten Bereichs
	.		
	.		

mstart:

.ds.l 1

\$49 MFREE

C:	long	Mfree(addr)
	long	addr;

Assembler:	MOVE.L	addr,-(SP)
	MOVE.W	#\$49,-(SP)

```

TRAP          #1
ADDQ.L        #6,SP

```

GFA-BASIC: error% = GEMDOS(73, L:addr%)

OMIKRON: GEMDOS(error,73,HIGH(address),LOW(address))

Ein mit MALLOC reservierter Speicher kann jederzeit mit MFREE wieder an das GEMDOS zurückgegeben werden. Dazu wird dem GEMDOS die Adresse übermittelt, ab der der Speicher freigegeben werden soll. Der Wert wird üblicherweise die von MALLOC gelieferte Adresse sein. Bei einem Rückgabewert von Null in D0 wurde der Speicher ohne Fehler vom GEMDOS freigegeben, negative Werte signalisieren in der üblichen Weise Fehler.

Beispiel:

```

MOVE.L        #mstart,-(SP)  Adresse eines zuvor allokierten
                             Bereichs
MOVE.W        #$49,-(SP)    Funktionsnummer
TRAP          #1
ADDQ.L        #6,SP          Anzahl freier Bytes in D0
TST.L         D0             Fehler?
BNE           error          D0 <> 0 ist Fehler!

```

```

mstart:
        .ds.l        1

```

\$4A MSHRINK

C:

```

int      Mshrink(dummy, block, newsize)
word     dummy = 0;
long     block;
long     newsize;

```

Assembler:

```

MOVE.L    newsize,-(SP)
MOVE.L    block,-(SP)
MOVE.W    #0,-(SP)
MOVE.W    #$4A,-(SP)
TRAP      #1
ADDQ.L    #12,SP

```

GFA-BASIC: error% = GEMDOS(74, 0, L:block%, L:newsize%)

OMIKRON: GEMDOS(error,74,0,HIGH(block),LOW(block),HIGH(newsize),LOW(newsize))

Mit der MSHRINK-Funktion kann im Gegensatz zur MALLOC-Funktion ein genau bestimmter Speicherbereich reserviert werden. Der darüber liegende Speicher wird an GEMDOS zurückgegeben, auch wenn er zuvor reserviert war. Mit dieser Funktion kann z.B. ein Programm seinen tatsächlichen Speicherbedarf reservieren und den verbleibenden Bereich freistellen.

Als Parameter werden der Funktion die Anfangsadresse und die Länge des zu reservierenden Bereichs übergeben. Der mit diesen Parametern spezifizierte Bereich wird dann vom GEMDOS reserviert und erst mit Beenden des Programms oder nach Aufruf der MFREE-Funktion wieder freigegeben. Der zusätzliche Dummy-Parameter muß immer Null sein. Üblicherweise beginnen Programme mit der folgenden oder einer ähnlichen Befehlssequenz. Nach dem Aufruf muß in D0 eine Null stehen, sonst ist ein Fehler aufgetreten. Diese Funktion darf unter OMIKRON-BASIC nicht benutzt werden.

Beispiel:	MOVE.L	A7,A5	Stackpointer nach A5 retten
	MOVE.L	#USTCK,A7	Stack für das Programm einrichten
	MOVE.L	4(A5),A5	A5 zeigt jetzt auf den Base-Page-Anfang, genau \$100 Bytes unter dem Programm-anfang
	MOVE.L	\$C(A5),D0	\$C(A5) enthält Länge des Programm-Bereichs
	ADD.L	\$14(A5),D0	\$14(A5) enthält Länge des initialisierten Datenbereichs
	ADD.L	\$1C(A5),D0	\$1C(A5) enthält Länge des uninitialisierten Datenbereichs
	ADD.L	#\$100,D0	\$100 Bytes Base-Page reservieren
	MOVE.L	D0,-(SP)	D0 enthält die Länge des zu reserv. Bereichs
	MOVE.L	A5,-(SP)	A5 enthält Beginn des zu reserv. Speichers
	MOVE.W	#0,-(SP)	ein Wort ohne Bedeutung, trotzdem nötig!
	MOVE.W	#\$4A,-(SP)	Funktionsnummer
	TRAP	#1	
	ADD.L	#12,SP	die übliche Aufräumarbeit auf dem Stack
	TST.L	D0	Fehler aufgetreten?
	BNE	error	besser abbrechen, das ging schief
			hier gehts weiter, wenn alles geklappt hat

\$4B PEXEC

C:

long	Pexec(mode, prg, cmdl, envp)
int	mode;
char	*prg;
char	*cmdl;
char	*envp;

Assembler:

PEA	envp
PEA	cmdl
PEA	prg
MOVE.W	mode, -(SP)
MOVE.W	#\$4B, -(SP)
TRAP	#1
ADDQ.L	#16, SP

GFA-BASIC: error% = GEMDOS(75, mode%, L:Vprg\$, LV:cmdl\$, V:envp\$)

OMIKRON: GEMDOS(error, 75, mode, HIGH(prg), LOW(prg), HIGH(cmdl), LOW(cmdl), HIGH(envp), LOW(envp))

Die Pexec()-Funktion erlaubt das Laden und Nachladen von Programmen. Wahlweise kann das geladene Programm automatisch gestartet werden. Auf dem Stack werden neben der Funktionsnummer die Adressen von drei Strings und ein Moduswort erwartet.

Betrachten wir zunächst das Modus-Wort. Dieses Wort darf die Werte 0, 3, 4 oder 5 enthalten. Mode = 0 stellt die sogenannte LOAD'N GO-Option dar. In diesem Fall wird die Datei von Diskette geladen, deren File- und Path-Name in PTR1 enthalten ist. PTR2 enthält bei dieser Option den Command Tail, vergleichbar den Angaben, die Sie einem .TTP-Programm per Dialogbox übergeben können. PTR3 schließlich zeigt auf den sogenannten Environment-String. Sollen die Möglichkeiten von Command Tail oder Environment-String nicht genutzt werden, so sollten beide auf ein Null-Byte (NULL-Pointer) zeigen.

Nach dem Laden des auf diese Weise angegebenen Programms wird das Programm automatisch gestartet. Das aufrufende Programm, also das, welches den Pexec-Aufruf gestartet hat, bleibt im Speicher. Eventuell geöffnete Dateien werden an das neu gestartete Programm (den neu gestarteten Prozeß) weitergereicht. Dieses neue Programm wird auch als "Child Process" bezeichnet. Nach der Beendigung des Child Process mittels Pterm0 o.ä. wird die Kontrolle an das aufrufende Programm, den Parent Process, zurück übergeben.

Ist das Modus-Wort eine Drei, so werden die Parameter in PTR1 bis PTR3 in derselben Form wie auch beim Mode = 0 behandelt. Allerdings wird das spezifizierte File nur in den Speicher geladen, es wird nicht ausgeführt. Nach dem Aufruf von Pexec mit Mode = 3 erhält man die Adresse der Base Page des nachgeladenen Programms in D0 zurück.

Das ergibt vielleicht auf den ersten Blick keinen besonderen Sinn. Brauchbar ist diese Funktion jedoch zumindest in einem guten Debugger. Will man mit Hilfe des Debuggers ein fehlerhaftes Programm untersuchen, so darf es ja nicht unmittelbar nach dem Laden ausgeführt werden, da sonst dem Debugger die Kontrolle entzogen würde. Das fehlerhafte Programm würde stattdessen die Kontrolle erhalten und genauso abstürzen wie ohne Debugger. Die LOAD-Option des Pexec schafft hier Abhilfe.

Ist das Mode-Wort = 4, so wird ein im Speicher befindliches Programm gestartet, als sei es soeben geladen worden. Dazu wird in PTR2 die Adresse der benötigten Base Page erwartet. PTR1 und PTR3 werden nicht genutzt. Auf diese Weise kann ein Programm gestartet werden, das zuvor mit Pexec, Mode = 3, geladen wurde.

Als letzte Möglichkeit gibt es für das Mode-Wort den Wert Fünf. Mit dieser Option wird eine Base Page im Speicher angelegt. Außerdem wird der größte freie Speicherblock allokiert. Natürlich sind wesentliche Daten der Base Page nach diesem Aufruf nicht ausgefüllt. Insbesondere sind das die Angaben über Text-, Data- und BSS-Bereiche. Diese Angaben muß der Programmierer selbst eintragen.

Die folgende Tabelle zeigt noch einmal die verfügbaren Kombinationen:

Modus	ptr1	ptr2	ptr3	Funktion	Ergebnis
0	Path/File-Name	Kommandozeile	Environment	Prog. laden/starten	Return-Wert des Programms
3	Path/File-Name	Kommandozeile	Environment	Prog. laden	Basepage des Programms
4	0L	Basepage	0L	Prog. starten	Return-Wert des Programms
5	0L	Kommandozeile	Environment	Basepage allokieren	Basepage-Adresse

Beispiel:

PEA	env	Environment
PEA	com	Kommandozeile
PEA	fil	File-Name
MOVE.W	#0,-(SP)	Laden und Starten bitte
MOVE.W	#\$4B,-(SP)	Funktionsnummer

```

TRAP          #1
ADD.L         #14,SP      hier kommen wir erst nach Ende
                          des nachgeladenen Programms an
.
.
.
fil:          .dc.b       'qsort.prg',0      Sortierroutine nachladen
com:          .dc.b       'up daten.asc',0    Kommando an unsere Sortierroutine
env:          .dc.w       0                  kein Environment
.
.

```

\$4C PTERM

C: void Pterm(retcode)
 int retcode;

Assembler: MOVE.W retcode,-(SP)
 MOVE.W #\$4C,-(SP)
 TRAP #1

GFA-BASIC: nicht anwendbar

OMIKRON: nicht anwendbar

PTERM \$4C stellt nach Pterm0(), Funktionsnummer \$00, und Ptermres(), Nummer \$31, die dritte Methode dar, ein Programm zu beenden. Pterm() stellt den vom Programm belegten und auch evtl. zusätzlich allokierten Speicher automatisch wieder GEMDOS zur Verfügung. Anders aber als Pterm0() kann mit dieser Funktion dem Aufrufer ein von Null unterschiedlicher und selbst definierbarer Rückgabewert übermittelt werden. Damit hat man die Möglichkeit, dem aufrufenden Programm eine wenn auch kurze Nachricht mitzuteilen.

Alle in diesem Prozeß eröffneten Dateien werden von PTERM automatisch geschlossen.

Beispiel: MOVE.W #37,-(SP) nach Bedarf jeder 2-Byte-Wert
 MOVE.W #\$4C,-(SP) Programm jetzt
 TRAP #1 beenden
 . hier kommen wir nie hin

\$4E FSFIRST

C: int Ffirst(fnam,attr)
 char *fnam;
 int attr;

Assembler: MOVE.W attr, -(SP)
 PEA fnam
 MOVE.W #\$4E, -(SP)
 TRAP #1
 ADDQ.L #8, SP

GFA-BASIC: error% = GEMDOS(78, L:Varptr(fnam\$), attr%)

OMIKRON: GEMDOS(error, 78, HIGH(fname), LOW(fname), attr)

Mit der FSFIRST-Funktion kann überprüft werden, ob eine Datei mit dem angegebenen Namen im Directory vorhanden ist. Wird eine gleichnamige Datei gefunden, so erhält man den File-Namen, das File-Attribut, Datum und Uhrzeit der Datei-Erstellung und die Größe der Datei in Bytes zurück. Die gelieferte Information wird von GEMDOS im DTA-Buffer abgelegt, dessen Adresse ja mit der FSETDTA-Funktion bestimmt werden kann.

Besonders angenehm ist die Tatsache, daß der File-Name nicht vollständig eingegeben werden muß. Einzelne Zeichen im File-Namen können gegen ein Fragezeichen "?" ausgetauscht werden, aber auch ganze Buchstabengruppen können durch einen "*" ersetzt werden. In der extremen Form würde ein File-Name auf die Zeichenkette *.* reduziert. In diesem Fall würde die erste Datei im Directory den Anforderungen genügen und der File-Name mit den weiteren Informationen im DTA-Buffer erscheinen.

Außer dem File-Namen muß dem FSFIRST-Aufruf noch ein Such-Attribut mitgegeben werden. Die möglichen Parameter des Attributs entsprechen denen, die bei der CHMOD-Funktion einer Datei gegeben werden können:

\$00 = Normaler Zugriff, lesen/schreiben
 \$01 = Normaler Zugriff, schreibgeschützt
 \$02 = Verborgener Eintrag
 \$04 = Verborgener System-Eintrag
 \$08 = Volume Label, Disketten-Name
 \$10 = Subdirectory
 \$20 = Datei wurde beschrieben und geschlossen

Beim Suchen nach Dateien gelten die folgenden Regeln:

Ist Attribut-Wort Null, so werden nur normale Dateien erkannt. System-Dateien oder Subdirectories werden nicht erkannt. System-Files, verborgene Dateien und Subdirectories werden gefunden, wenn die entsprechenden Attribut-Bits gesetzt sind. Volume-Labels werden aber dann weiterhin nicht erkannt. Um das Volume-Label zu erhalten, muß diese Option ausdrücklich im Attribut gesetzt werden. Alle anderen Dateien werden dann nicht durchsucht. Nach dem Aufruf erhält man in D0 den Wert Null, wenn ein entsprechendes File gefunden wurde. In diesem Fall wird der 44 Bytes große DTA-Buffer wie folgt aufgebaut:

Byte 0 - 20	für GEMDOS reserviert (bitte ernst nehmen!).
Byte 21	File-Attribut
Byte 22 - 23	Uhrzeit der Datei-Erstellung
Byte 24 - 25	Datum der Datei-Erstellung
Byte 26 - 29	Dateigröße in Bytes (Langwort)
Byte 30 - 43	Name und Extension der gefundenen Datei

Wurde jedoch keine dem angegebenen Such-String entsprechende Datei gefunden, so erhält man die Fehlermeldung -33, File Not Found.

Beispiel:	PEA	dta	DTA-Buffer einrichten
	MOVE.W	#1A, -(SP)	Funktionsnummer SETDTA
	TRAP	#1	
	ADDQ.L	#6, SP	
	MOVE.W	#attrib, -(SP)	Attribut-Wert
	MOVE.L	#filnam, -(SP)	File-Name, nach dem gesucht wird
	MOVE.W	#\$4E, -(SP)	Funktionsnummer Fsfirst
	TRAP	#1	
	ADDQ.L	#8, SP	
	TST	D0	Datei gefunden?
	BNE	notfound	scheinbar nicht
	.		
	.		
attrib:	.dc.b	0	nur normale Files suchen
filnam:	.dc.b	'*.*', 0	die erste mögliche Datei suchen
	.		
	.		
dta:	.ds.b	44	Platz für den DTA-Buffer

\$4F FSNEXT

C: int Fsnext()

Assembler: MOVE.W #\$4F, -(SP)
 TRAP #1
 ADDQ.L #2, SP

GFA-BASIC: error% = GEMDOS(79)

OMIKRON: GEMDOS(error, 79)

Mit der FSNEXT-Funktion (Search Next) kann geprüft werden, ob sich möglicherweise andere Dateien auf der Diskette befinden, die mit dem angegebenen File-Namen übereinstimmen. Dazu muß nur die Funktionsnummer übergeben werden, Parameter werden von FSNEXT nicht benötigt. Allerdings sind alle Parameter bereits vom FSFIRST-Aufruf richtig gesetzt.

Wird der Suchstring wie im vorigen Beispiel sehr global gehalten, so können mit FSFIRST und FSNEXT alle Dateien einer Diskette nacheinander bestimmt und gegebenenfalls angezeigt werden. Damit hat man in Programmen die Möglichkeit, mit nur wenig Aufwand ein Directory anzuzeigen.

Beispiel: MOVE.W #\$4F, -(SP) Search next
 TRAP #1 ist noch was da?
 ADDQ.L #2, SP
 TST.L D0 bei negativen Werten nichts mehr
 .
 .

\$56 FRENAME

C: int Frename(dummy, oldname, newname)
 int dummy = 0;
 char *oldname;
 char *newname;

Assembler: PEA newname
 PEA oldname
 MOVE.W #0, -(SP)
 MOVE.W #\$56, -(SP)
 TRAP #1
 ADDQ.L #12, SP

GFA-BASIC: error% = GEMDOS(86, 0, L:Varptr(old\$), L:Varptr(new\$))

OMIKRON: GEMDOS(error,86,0,HIGH(old),LOW(old),HIGH(new),LOW(new))

Unter GEMDOS reicht zum Umbenennen einer Datei der Aufruf der Funktion FRENAME, der zwei Pointer auf File- oder Pfadnamen übergeben werden. Der erste Pointer zeigt auf den neuen Namen, bei Bedarf auch mit Angabe eines Pfadnamens der Datei, der zweite Pointer zeigt auf den bisherigen Namen. Neben den zwei Pointern wird noch ein 2-Byte-Parameter benötigt. Dieser Parameter enthält zwar keine zusätzliche Information, ist jedoch zwingend erforderlich.

FRENAME bezieht sich jedoch nicht ausschließlich auf den eigentlichen Namen der Datei. Man kann auch den Pfad-Namen "umbenennen". Im Endeffekt bedeutet das, daß die Datei in ein anderes Subdirectory kopiert und im bisherigen Directory gelöscht wird. Allerdings werden nicht die Daten, sondern nur der DIR-Eintrag von einem in ein anderes Directory kopiert. Darum funktioniert dieses FRENAME auch nur auf einem Laufwerk. Von Laufwerk A: nach Laufwerk B: kann eine Datei nicht umbenannt werden. Dazu muß ja auch der Datei-Inhalt kopiert werden.

In den ROM-Versionen 1.0 und 1.02 kann leider nicht der Name eines Subdirectories geändert werden. Als Rückgabewert erhält man in D0 entweder Null, dann ist der Name korrekt geändert worden, oder man erhält einen Fehlercode zurück.

Beispiel:

PEA	newnam	neuer Dateiname
PEA	oldnam	umzubenennende Datei
MOVE.W	#0,-(SP)	Dummy
MOVE.W	#\$56,-(SP)	Funktionsnummer
TRAP	#1	
ADD.L	#12,SP	
TST.L	D0	auf Fehler prüfen
.		
.		
oldnam:		Null-Byte am Ende nicht vergessen!
.dc.b	'oldfile.dat',0	
newnam:		
.dc.b	'newname.dat',0	
.		
.		

\$57 FDATEIME

C: void Fdateime(timeptr, handle, flag)
 int handle;

```
char      *timeptr;
int       flag;
```

```
Assembler:  MOVE.W    flag, -(SP)
               MOVE.W    handle, -(SP)
               PEA        timeptr
               MOVE.W    #$57, -(SP)
               TRAP       #1
               ADDQ.L     #10, SP
```

```
GFA-BASIC: GEMDOS(87, L:timeptr%, handle%, flag%)
```

```
OMIKRON:   GEMDOS(, 87, HIGH(time), LOW(time), handle, flag)
```

Wenn im Desktop das Directory nicht als Icons, sondern als Text dargestellt wird, so sieht man neben der Dateigröße in Bytes auch das Datum der Datei-Erstellung und die Uhrzeit. Diese beiden letzten Angaben können mit der Funktion \$57 wahlweise gesetzt oder ermittelt werden. Dazu ist es allerdings erforderlich, daß die Datei bereits mit FOPEN oder FCREATE eröffnet wurde. Die bei der Eröffnung erhaltene Handle-Nummer muß als Parameter übergeben werden. Weitere Parameter sind ein Wort, das als Flag signalisiert, ob Datum und Zeit (Null) gesetzt oder ermittelt (Eins) werden sollen, sowie ein Pointer auf einen vier Byte großen Buffer, der entweder die Ergebnis-Daten erhält oder der vor dem Aufruf mit den benötigten Daten versorgt wird.

Dieser Datum-Buffer enthält in den beiden ersten Bytes die Zeit, das Datum steht in den letzten Bytes. Das Format der Daten ist identisch mit denen der Funktionen zum Setzen/Ermitteln von Datum und Zeit.

Ein Wort zur Warnung an dieser Stelle. Programmierer, die diese Funktion sowohl in C als auch in Assembler aufrufen, müssen genau aufpassen. In der Include-Datei OSBIND.H einiger C-Compiler sind die Parameter "timeptr" und "handle" vertauscht. Überprüfen Sie bitte im Einzelfall, in welcher Reihenfolge Ihr Compiler die Parameter erwartet.

```
Beispiel:  MOVE.W    #0, -(SP)      Datum und Zeit setzen
             MOVE.W    handle, -(SP) Datei muß zuerst eröffnet werden
             PEA        buff        4 Bytes zum Speichern
             MOVE.W    #$57, -(SP) Funktionsnummer
             TRAP       #1
             ADDQ.L     #10, SP
```

```
handle:
```

```
    .ds.b      2
```


buff:

.ds.b

4

.

.

22. Die BIOS-Funktionen des Atari ST

0Getmpb

C: void Getmpb(zeiger)
 long zeiger;

Assembler: move.l zeiger, -(sp)
 move.w #0, -(sp)
 trap #13
 addq.l #6, sp

GFA: VOID BIOS(L:zeiger)

OMIKRON: BIOS(,0,HIGH(zeiger),LOW(zeiger))

Diese Funktion füllt einen Block von zwölf Bytes Länge, dessen Adresse in "zeiger" enthalten ist, mit dem sogenannten Memory-Parameter-Block. Dieser Block enthält selbst wieder drei Zeiger:

long	md_mfl	Memory free list
long	md_mal	Memory allocated list
long	md_rover	roving pointer

Die Strukturen, auf die jeder Zeiger zeigt, sind folgendermaßen aufgebaut:

long	m_link	Zeiger auf nächsten Block
long	m_start	Startadresse des Blocks
long	m_lenght	Länge des Blocks in Bytes
long	m_own	Process Descriptor

Beispiel: move.l #puffer, -(sp) Puffer für MPB
 move.w #0, -(sp) getmpb
 trap #13
 addq.l #6, sp

Wir erhalten die Werte \$48E, 0 und \$48E. Ab Adresse \$48E stehen folgende Daten (bei 1 MByte RAM bzw. mit Blitter-TOS und 4 MByte RAM):

m_link	0	kein weiterer Block
m_start	\$A100/\$39FF0/	\$CA00 Startadresse des freien Speichers
m_lenght	\$EDF00/\$BE010/\$3EB600	Länge des freien Speichers
m_own	0	kein Process Descriptor

1 Bconstat

C: int Bconstat(dev)
 int dev;

Assembler: move.w dev, -(sp)
 move.w #1, -(sp)
 trap #13
 addq.l #4, sp

GFA: status% = BIOS(1, device%)

OMIKRON: BIOS(status, 1, device)

Diese Funktion liefert den Status eines Eingabegeräts, der folgendermaßen definiert ist:

Status 0 kein Zeichen bereit

Status -1 (mindestens) ein Zeichen bereit

Der Parameter "dev" gibt das Eingabegerät an:

dev	Eingabegerät
0	PRT:, Centronics-Schnittstelle
1	AUX:, RS232-Schnittstelle
2	CON:, Tastatur und Bildschirm
3	MIDI, MIDI-Schnittstelle
4	IKBD, Tastatur-Port

Die folgende Tabelle gibt die erlaubten Zugriffe auf diese Geräte an:

Operation	PRT:	AUX:	CON:	MIDI	IKBD
Eingabe-Status	nein	ja	ja	ja	nein
Eingabe	ja	ja	ja	ja	nein
Ausgabe-Status	ja	ja	ja	ja	ja
Ausgabe	ja	ja	ja	ja	ja

Beispiel:

```

wait      move.w    #1,-(sp)    RS232
          move.w    #1,-(sp)    bconstat
          trap      #13
          addq.l    #4,sp
          tst       d0           Zeichen vorhanden ?
          beg       wait        nein, warten

```

Das Beispielprogramm wartet so lange, bis ein Zeichen von der RS232-Schnittstelle bereitsteht.

2 Bconin

```

C:        long      Bconin(dev)
          int        dev;

```

```

Assembler: move.w    dev,-(sp)
          move.w    #2,-(sp)
          trap      #13
          addq.l    #4,sp

```

```

GFA:      zeichen = BIOS(2,device%)

```

```

OMIKRON:  BIOS(zeichen,2,device)

```

Diese Funktion holt ein Zeichen von einem Eingabegerät ab. Der Parameter "dev" hat dabei die gleiche Bedeutung wie bei der vorherigen Funktion. Die Funktion kehrt erst dann zurück, wenn ein Zeichen bereitsteht.

Das erhaltene Zeichen steht im untersten Byte des Ergebnisses. War das Eingabegerät die Tastatur ("con", 2), so wird zusätzlich noch der Tasten-Scancode im unteren Byte des oberen Wortes des Ergebnisses geliefert (siehe Beschreibung des Tastatur-Prozessors).

```

Beispiel: move.w    #2,-(sp)    con
          move.w    #2,-(sp)    bconin
          trap      #13
          addq.l    #4,sp

```

3 Bconout

```

C:        void      Bconout(dev, c)
          int        dev, c;

```

Assembler: move.w c, -(sp)
 move.w dev, -(sp)
 move.w #3, -(sp)
 trap #13
 addq.l #6, sp

GFA: VOID BIOS(3, device%, zeichen%)

OMIKRON: BIOS(, 3, device, zeichen)

Diese Funktion dient zur Ausgabe eines Zeichens "c" an das Gerät "dev" (Bedeutung wie bei den obigen Funktionen). Die Funktion kehrt erst dann zurück, wenn das Zeichen ausgegeben wurde.

Beispiel: move.w #'A', -(sp)
 move.w #0, -(sp) PRN:
 move.w #3, -(sp) bconout
 trap #13
 addq.l #6, sp

Das Beispiel gibt den Buchstaben "A" an den Drucker aus.

4 Rwabs

C: long Rwabs(rwflag, puffer, anzahl, recnr, dev)
 long puffer;
 int , rwflag, anzahl, recnr, dev;

Assembler: move.w dev, -(sp)
 move.w recnr, -(sp)
 move.w anzahl, -(sp)
 move.l puffer, -(sp)
 move.w rwflag, -(sp)
 move.w #4, -(sp)
 trap #13
 add.w #14, sp

GFA: status = BIOS(4, rwflag%, L:puffer, anzahl%, recnr%, device%)

OMIKRON: BIOS(status, 4, rwflag, HIGH(puffer), LOW(puffer), anzahl, recnr, device)

Diese Funktion dient zum Schreiben und Lesen von Sektoren auf Diskette. Die Parameter haben dabei folgende Bedeutung:

rwflag	Bedeutung
0	Sektoren lesen
1	Sektoren schreiben
2	Sektoren lesen, Diskettenwechsel ignorieren
3	Sektoren schreiben, Diskettenwechsel ignorieren

"puffer" ist dabei die Adresse eines Puffers, in den Daten von Diskette gelesen bzw. vom dem die Daten auf Diskette geschrieben werden. Der Puffer sollte an einer geraden Adresse beginnen, da sonst die Übertragung sehr langsam verläuft.

"anzahl" gibt an, wie viele Sektoren bei einem Aufruf gelesen oder geschrieben werden sollen. "recno" gibt an, bei welchem logischen Sektor dabei begonnen wird.

"dev" schließlich bestimmt, welches Floppy-Laufwerk benutzt wird:

dev	Laufwerk
0	Drive A
1	Drive B
2+	Harddisk, RAM-Disk, Netzwerk

Die Funktion liefert als Ergebnis einen Fehlercode. Lautet dieser Wert Null, so ist die Operation fehlerfrei verlaufen. Bei einem Fehler steht der negative Fehlercode dort.

Beispiel:

```

move.w    #0,-(sp)    Laufwerk A
move.w    #10,-(sp)   Beginn bei logischem Sektor 10
move.w    #2,-(sp)    2 Sektoren lesen
move.l    #puffer,-(sp) Pufferadresse
move.w    #0,-(sp)    Sektoren lesen
move.w    #4,-(sp)    rwabs
trap      #13
add.w     #14,sp
...
puffer    ds.b        2*512

```

5 Setexec

C: long Setexec(nummer, vector)
 int nummer;
 long vector;

Assembler: move.l vector, -(sp)
 move.w nummer, -(sp)
 move.w #5, -(sp)
 trap #13
 addq.l #8, sp

GFA: oldvector = BIOS(5, nummer%, L:newvector)

OMIKRON: BIOS(oldvector, 5, nummer, HIGH(newvector), LOW(newvector))

Die Funktion setexec erlaubt es, einen Exception-Vektor des 68000-Prozessors zu verändern. Dazu müssen die Nummer des Vektors in "nummer", die Adresse der zugehörigen Routine in "vector" enthalten sein. Die Funktion liefert als Ergebnis den alten Vektor zurück. Will man den Vektor nur lesen, so übergibt man als neuen Wert -1. Mit dieser Funktion lassen sich die 256 Vektoren des Prozessors sowie acht Vektoren für GEM setzen, die die Nummern \$100 bis \$107 haben (Adressen \$400 bis \$41C).

Beispiel: move.l #buserror, -(sp)
 move.w #2, -(sp)
 move.w #5, -(sp)
 trap #13
 addq.l #8, sp
 ...
 buserror ...

6 Tickcal

C: long Tickcal()

Assembler: move.w #6, -(sp)
 trap #13
 addq.l #2, sp

GFA: millisek = BIOS(6)

OMIKRON: BIOS(millisek, 6)

Diese Funktion liefert als Ergebnis die Anzahl der Millisekunden zwischen zwei System-Timer-Aufrufen.

Beispiel:

```

move.w    #6, -(sp)
trap      #13
addq.l    #2, sp

```

Ergebnis: 20 ms

7 Getbpb

C:

```

long      Getbpb(dev)
int        dev;

```

Assembler:

```

move.w    dev, -(sp)
move.w    #7, -(sp)
trap      #13
addq.l    #4, sp

```

GFA: zeiger = BIOS(7, device%)

OMIKRON: BIOS(zeiger, 7, device)

Diese Funktion liefert einen Zeiger auf den BIOS-Parameter-Block des Drives "dev" (0 = Laufwerk A, 1 = Laufwerk B).

Der BPB ist folgendermaßen aufgebaut:

int	recsiz	Sektorgröße in Bytes
int	clsiz	Cluster-Größe in Sektoren
int	clsizb	Cluster-Größe in Bytes
int	rdlen	Directory-Länge in Sektoren
int	fsiz	FAT-Größe in Sektoren
int	fatrec	Start-Sektor der zweiten FAT
int	datrec	Sektornummer des ersten Daten-Clusters
int	numcl	Anzahl der Daten-Cluster auf der Diskette
int	bflags	Bit 0 = 0: 12-Bit FAT-Einträge = 1: 16-Bit FAT-Einträge

Eine Bemerkung zu rdlen: Da ein Directory-Eintrag 32 Bytes lang ist, errechnet sich die Gesamtzahl der möglichen Root-Directory-Einträge zu $rdlen * reccsize / 32 = rdlen * 16$.

Zu der Größe der FAT-Einträge ist zu bemerken, daß mit 12-Bit-Einträgen maximal 2000 Cluster = 4 MByte verwaltet werden können, während 16-Bit-Einträge bis zu 32000 Cluster = 64 MByte erlauben.

Die Funktion liefert für Laufwerk A die Adresse \$4DCE, für Laufwerk B die Adresse \$4DEE. Eine Adresse von Null kennzeichnet einen Fehler.

Beispiel:

```

move.w    #0,-(sp)    Laufwerk A
move.w    #7,-(sp)    getbpb
trap      #13
addq.l    #4,sp

```

Hier die Daten des BPB für 80 Track "single sided"- und "double sided"-Laufwerke:

Parameter	80 Track SS	80 Track DS
recsiz	512	512
clsiz	2	2
clsizb	1024	1024
rdlen	7	7
fsiz	5	5
fatrec	6	6
datrec	18	18
numcl	351	711
bflags	0	0

8 Bcostat

C:

```

long Bcostat(dev)
int      dev;

```

Assembler:

```

move.w    dev,-(sp)
move.w    #8,-(sp)
trap      #13
addq.l    #4,sp

```

GFA: status = BIOS(8,device%)

OMIKRON: BIOS(status,8,device)

Diese Funktion prüft, ob das mit "dev" angegebene Ausgabegerät bereit ist, das nächste Zeichen auszugeben. "dev" kann die Werte annehmen, die in

Funktion Eins beschrieben wurden. Das Ergebnis dieser Funktion lautet entweder -1, falls das Ausgabegerät bereit ist, oder Null, falls gewartet werden muß.

Beispiel:

move.w	#0, -(sp)	Drucker bereit?
move.w	#8, -(sp)	bcostat
trap	#13	
addq.l	#4, sp	

9 Mediach

C:

```
long Mediach(dev)
int dev;
```

Assembler:

move.w	dev, -(sp)
move.w	#9, -(sp)
trap	#13
addq.l	#4, sp

GFA: change = BIOS(9,device%)

OMIKRON: BIOS(change,9,device)

Mit dieser Funktion läßt sich ermitteln, ob zwischenzeitlich eine Diskette gewechselt wurde. Dazu muß dem Parameter "dev" die Laufwerksnummer übergeben werden (0 = Laufwerk A, 1 = Laufwerk B). Als Ergebnis können drei Werte auftreten:

- 0 Diskette wurde definitiv nicht gewechselt
- 1 Diskette kann gewechselt worden sein
- 2 Diskette wurde definitiv gewechselt

Beispiel:

move.w	#1, -(sp)	Laufwerk B
move.w	#9, -(sp)	mediach
trap	#13	
addq.l	#4, sp	

10 Drvmap

C:

```
long Drvmap()
```

Assembler:

move.w	#10, -(sp)
trap	#13
addq.l	#2, sp

GFA: `drvmap = BIOS(10)`

OMIKRON: `BIOS(drvmap,10)`

Diese Funktion liefert einen Bit-Vektor, der die angeschlossenen Laufwerke enthält. Das n-te Bit ist dabei gesetzt, wenn Laufwerk n verfügbar ist (0 bedeutet wieder A usw.). Auch wenn nur ein Laufwerk angeschlossen ist, wird %11 geliefert, da von zwei logischen Laufwerken ausgegangen wird.

Beispiel: `move.w        #10, -(sp)        drvmap`
 `trap            #13`
 `addq.l        #2, sp`

11 Kbshift

C: `long Kbshift(modus)`
 `int                modus;`

Assembler: `move.w        modus, -(sp)`
 `move.w        #11, -(sp)`
 `trap            #13`
 `addq.l        #4, sp`

GFA: `status = BIOS(11,modus%)`

OMIKRON: `BIOS(status,11,modus)`

Mit dieser Funktion können Sie den Status der Sondertasten der Tastatur erfahren oder verändern. Ist "modus" -1, so erhalten Sie den Status, ein positiver Wert wird als Status übernommen. Der Status ist ein Bit-Vektor, der folgendermaßen aufgebaut ist:

Bit	Bedeutung
0	rechte Shift -Taste
1	linke Shift -Taste
2	Ctrl -Taste
3	Alt -Taste
4	Caps Lock eingeschaltet
5	(Clr/Home), Ersatz für rechte Maus-Taste
6	(Insert), Ersatz für linke Maus-Taste
7	unbenutzt

Beispiel:

move.w	#-1,-(sp)	Shift-Status lesen
move.w	#11,-(sp)	kbshift
trap	#13	
addq.l	#4,sp	

Eine schnellere Möglichkeit, den Shift-Status zu erfahren oder zu ändern, besteht darin, die Adresse dieser Variablen aus dem TOS-Header zu entnehmen und direkt zu verwenden. Dies ist besonders dann angeraten, wenn der Aufruf innerhalb einer Interrupt-Routine geschehen soll. Die Vorgehensweise kann dann so sein:

move.l	\$4f2,a0	_sysbase, Adresse des TOS-Headers
cmp.w	#102,2(a0)	mindestens TOS-Version 1.02
blt	sorry	hier geht's leider nicht
move.l	\$24(a0),kbshift	Adresse des Shift-Status
...		
move.l	kbshift,a0	
btst.b	#3,(a0)	ALT-Taste gedrückt?
bne	is_alt	ja

Wie Sie aus dem Beispiel ersehen können, steht die Adresse des Shift-Status im ROM-Header mit einem Offset von \$24. Diese Möglichkeit besteht jedoch erst ab TOS-Version 1.02 (Blitter-TOS). Die Version steht ab ROM-Header plus 2 (siehe auch Beschreibung der Variablen `_sysbase` im Kapitel "Systemvariablen").

23. Die XBIOS-Funktionen des Atari ST

0 Initmous

C: void Initmous(typ, parameter, vector)
 int typ;
 long parameter, vector;

Assembler: move.l vector, -(sp)
 move.l parameter, -(sp)
 move.w typ, -(sp)
 move.w #0, -(sp)
 trap #14
 add.w #12, sp

GFA: VOID XBIOS(0, typ%, L:parameter%, L:vector%)

OMIKRON: XBIOS(, 0, typ, HIGH(parameter), LOW(parameter), HIGH(vector),
 LOW(vector))

Mit Hilfe dieser XBIOS-Funktion werden die Routinen zur Maus-Bearbeitung initialisiert. "vector" ist dabei die Adresse einer Routine, die bei einem Maus-Report vom Tastaturprozessor angesprungen wird. "typ" wählt unter folgenden Alternativen aus:

typ	Funktion
0	Disable mouse
1	Enable mouse, relativer Modus
2	Enable mouse, absoluter Modus
3	unbenutzt
4	enable mouse, Tastatur-Modus

Damit können Sie also auswählen, ob Mausbewegungen gemeldet werden und auf welche Weise dies geschieht.

"parameter" zeigt auf einen Parameterblock, der folgendermaßen aufgebaut ist:

```
char topmode
char buttons
char xparam
char yparam
```

"topmode" entscheidet über die Anordnung des Koordinatensystems. 0 bedeutet Y=0 liegt in der unteren Ecke, 1 bedeutet, daß die Y-Achse von oben aus zählt. "buttons" ist ein Parameter für den Befehl "set mouse buttons" des Tastatur-Prozessors (siehe Beschreibung des IKBD, intelligent keyboard). "xparam" und "yparam" sind Skalierungsfaktoren für die Maus-Bewegung. Wenn Sie "typ" zu 2 gewählt haben, also mit absolut mouse arbeiten, so schließen sich an den Parameterblock noch weitere vier Parameter an:

```
int    xmax
int    ymax
int    xstart
int    ystart
```

Dabei handelt es sich um die X- und Y-Koordinaten des Maximalwerts, den die Maus-Position annehmen kann, sowie den Startwert, auf den die Maus gesetzt wird.

Beispiel:

```

move.l    #vector, -(sp)  Adresse der Mouse-Routine
move.l    #parameter, -(sp)  Adresse des Parameterblocks
move.w    #1, -(sp)      enable relative mouse
move.w    #0, -(sp)      init mouse
trap      #14
add.w     #12, sp
...
parameter  dc.b          .....
...
vector     ...           Mouse-Interrupt-Routine
```

1 Ssbrk

C:

```

long      Ssbrk(anzahl)
int       anzahl;
```

Assembler:

```

move.w    anzahl, -(sp)
move.w    #1, -(sp)
trap      #14
addq.l    #4, sp
```

GFA: adresse = XBIOS(1, anzahl%)

OMIKRON: XBIOS(adresse, 1, anzahl)

Diese Funktion reserviert Speicherplatz. Die Anzahl der Bytes muß dabei in "anzahl" übergeben werden. Der Speicherplatz wird am oberen Ende des Speichers bereitgestellt. Die Funktion liefert als Ergebnis die Adresse des reser-

vierten Speicherbereichs. Diese Funktion muß vor der Initialisierung des Betriebssystems aufgerufen werden, d.h. sie muß noch vom Boot-ROM ausgeführt werden, ehe das Betriebssystem geladen wird.

Beispiel:

move.w	#\$400,-(sp)	1 KByte reservieren
move.w	#1,-(sp)	ssbrk
trap	#14	
addq.l	#4,sp	

2 Physbase

C: long Physbase()

Assembler:

move	#2,-(sp)
trap	#14
addq.l	#2,sp

GFA: base = XBIOS(2)

OMIKRON: XBIOS(base,2)

Diese Funktion liefert die Basis des physikalischen Bildschirm-RAMs. Unter der physikalischen Bildschirmadresse versteht man den Speicherbereich, der vom Video-Shifter dargestellt wird. Das Ergebnis ist ein Langwort.

Beispiel: \$F8000, Basisadresse des Bildschirms bei 1 MByte RAM

3 Logbase

C: long Logbase()

Assembler:

move	#3,-(sp)
trap	#14
addq.l	#2,sp

GFA: adresse = XBIOS(3)

OMIKRON: XBIOS(adresse,3)

Die logische Bildschirmbasis ist die Adresse, die bei allen Ausgabefunktionen als Bildschirmbasis benutzt wird. Sind physikalische und logische Bildschirmbasis verschieden, so wird ein Bildschirm dargestellt, während in einem ande-

ren RAM-Bereich bereits ein anderes Bild aufgebaut werden kann, welches dann später angezeigt wird. Das Ergebnis dieses Funktionsaufrufs ist wieder ein Langwort.

Beispiel: \$78000, Basisadresse des Bildschirms bei 512 KByte RAM

4 Getrez

C: int Getrez()

Assembler: move.w #4, -(sp)
 trap #14
 addq.l #2, sp

GFA: rez% = XBIOS(4)

OMIKRON: XBIOS(rez, 4)

Dieser Funktionsaufruf liefert die Bildschirmauflösung zurück:

0 := Low Resolution, 320*200 Punkte, 16 Farben
1 := Medium Resolution, 640*200 Punkte, 4 Farben
2 := High Resolution, 640*400 Punkte, monochrom

Beispiel: 2, monochrom

5 Setscreen

C: void Setscreen(logadr, physadr, rez)
 long logadr, physadr;
 int rez;

Assembler: move.w res, -(sp)
 move.l physadr, -(sp)
 move.l logadr, -(sp)
 move.w #5, -(sp)
 trap #14
 add.w #12, sp

GFA: VOID XBIOS(5, L:logadr%, L:physadr%, rez%)

OMIKRON: XBIOS(, 5, HIGH(logadr), LOW(logadr), HIGH(physadr), LOW(physadr), rez)

Diese Funktion dient zum Ändern der Bildschirmparameter, die mit den vorherigen drei Funktionen gelesen werden konnten. Soll ein Parameter nicht ge-

setzt werden, so muß ein negativer Wert übergeben werden. Die Parameter werden in der nächsten VBL-Routine gesetzt, damit keine Störungen auf dem Bildschirm auftreten.

Beispiel:

Die physikalische und die logische Bildschirmadresse soll auf \$70000 gesetzt, die Auflösung soll beibehalten werden.

```

move.w    #-1,-(sp)    Auflösung erhalten
move.l    #$70000,-(sp) physikalische Basis
move.l    #$70000,-(sp) logische Basis
move.w    #5,-(sp)     setscreen
trap      #14
add.w     #12,sp

```

6 Setpalette

```

C:         void        Setpalette(paletteptr)
           long        paletteptr;

```

```

Assembler: move.l      paletteptr,-(sp)
           move.w      #6,-(sp)
           trap        #14
           addq.l      #6,sp

```

```

GFA:       VOID XBIOS(6,L:palette%)

```

```

OMIKRON:   XBIOS(,6,HIGH(palette),LOW(palette))

```

Mit dieser Funktion kann eine neue Farbpalette geladen werden. Dazu muß paletteptr ein Zeiger auf eine Tabelle mit 16 Farben (je ein Wort) sein. Die Adresse der Tabelle muß gerade sein. Die Farben werden zu Beginn des nächsten VBL geladen.

```

Beispiel:  move.l      #palette,-(sp) Adresse der neuen Farbpalette
           move.w      #6,-(sp)     setpalette
           trap        #14
           addq.l      #6,sp
           ....
palette    dc.w        $777,$700,$070,$007,$111,$222,$333,$444
           dc.w        $555,$000,$001,$010,$100,$200,$020,$002

```


GFA: status = XBIOS(8,L:puffer%,L:filler%,dev%,sector%,track%,seite%,count%)

OMIKRON: XBIOS(status,8,HIGH(puffer),LOW(puffer),0,0,dev,sector,track,seite,count)

Mit dieser XBIOS-Funktion lassen sich ein oder mehrere Sektoren von Diskette lesen. Die Parameter haben dabei folgende Bedeutung:

- count:** Gibt an, wie viele Sektoren nacheinander gelesen werden. Dabei sind Werte zwischen eins und neun (Anzahl der Sektoren pro Track) möglich.
- seite:** Wählt die Diskettenseite aus, null bei einseitigen Laufwerken und null oder eins bei doppelseitigen Laufwerken.
- track:** Damit wird die Tracknummer bestimmt (0 -79 bei 80-Spur-Laufwerken oder 0-39 bei 40-Spur-Laufwerken).
- sector:** Die Sektornummer des ersten zu lesenden Sektors (1 - 9).
- dev:** Bestimmung der Laufwerksnummer, 0 für Laufwerk A und 1 für Laufwerk B.
- filler:** unbenutztes Langwort.
- puffer:** Puffer, in den die Diskettendaten geschrieben werden sollen. Der Puffer muß an einer Wortgrenze beginnen und groß genug für die zu lesenden Daten sein (512 Bytes mal Anzahl der Sektoren).

Die Funktion liefert als Ergebnis entweder Null, wenn kein Fehler aufgetreten ist, oder einen negativen Fehlercode, dessen Bedeutung Sie in der Tabelle im Anhang nachschlagen können.

Beispiel:

move.w	#1,-(sp)	einen Sektor lesen
move.w	#0,-(sp)	Seite Null
move.w	#0,-(sp)	Track Null
move.w	#1,-(sp)	Sektor Eins
move.w	#1,-(sp)	Laufwerk B
clr.l	-(sp)	
move.l	#puffer,-(sp)	
move.w	#8,-(sp)	flopdr
trap	#14	
add.w	#20,sp	

	tst	d0	Fehler aufgetreten?
	bmi	error	ja
	...		
puffer	ds.b 512		Puffer für einen Sektor

9 Flopwr

C:

```
int Flopwr(puffer, filler, dev, sector, track, seite, count)
long      puffer, filler;
int       dev, sector, track, seite, count;
```

Assembler:

```
move.w    count, -(sp)
move.w    seite, -(sp)
move.w    track, -(sp)
move.w    sector, -(sp)
move.w    dev, -(sp)
clr.l     -(sp)
move.l    puffer, -(sp)
move.w    #9, -(sp)
trap      #14
add.w     #20, sp
```

GFA:

```
status = XBIOS(9, L:puffer%, L:filler%, dev%, sector%, track%, seite%,
count%)
```

OMIKRON: XBIOS(status, 8, HIGH(puffer), LOW(puffer), 0, 0, dev, sector, track, seite, count)

Mit dieser XBIOS-Funktion lassen sich ein oder mehrere Sektoren auf Diskette schreiben. Die Parameter haben dabei die gleiche Bedeutung wie bei der Funktion 8 floprd. Die Funktion liefert als Ergebnis wieder einen Fehlercode, der die gleiche Bedeutung wie beim Sektorlesen hat.

Beispiel:

move.w	#3, -(sp)	drei Sektoren schreiben
move.w	#0, -(sp)	Seite n Null
move.w	#7, -(sp)	Track sieben
move.w	#1, -(sp)	Sektor eins
move.w	#0, -(sp)	Laufwerk A
clr.l	-(sp)	
move.l	#puffer, -(sp)	Adresse des Puffers
move.w	#9, -(sp)	flopwr
trap	#14	
add.w	#20, sp	
tst	d0	Fehler aufgetreten?

	bmi	error	ja
	...		
puffer	ds.b 3*512		Puffer für drei Sektoren

10 Flopfmt

C:

int	Flopfmt(puffer, filler, dev, spt, track, seite, interleave, magic, virgin)
long	puffer, filler, magic;
int	dev, spt, track, seite, interleave, virgin;

Assembler:

move.w	virgin, -(sp)
move.l	magic, -(sp)
move.w	interleave, -(sp)
move.w	seite, -(sp)
move.w	track, -(sp)
move.w	spt, -(sp)
move.w	dev, -(sp)
clr.l	-(sp)
move.l	puffer, -(sp)
move.w	#10, -(sp)
trap	#14
add.w	#26, sp

GFA: status=XBIOS(10,L:puffer%,L:filler%,dev%,spt%,track%,seite%,inter%,L:magic%,virgin%)

OMIKRON: XBIOS(status,10,HIGH(puffer),LOW(puffer),0,0,dev,spt,track,seite,inter,\$-789B,\$4321, virgin)

Diese Routine dient zum Formatieren eines Tracks auf der Diskette. Die Parameter haben dabei folgende Bedeutung:

virgin:	Mit diesem Wert werden die Sektoren beim Formatieren beschrieben. Als Standardwert hierzu empfiehlt sich \$E5E5. Das High-Nibble jedes Bytes darf dabei nicht den Wert \$F enthalten.
magic:	Als magic muß die Konstante \$87654321 benutzt werden, sonst wird die Formatierung nicht durchgeführt.
interleave:	Bestimmt, in welcher Reihenfolge die Sektoren auf Diskette geschrieben werden, normalerweise eins.
seite:	Damit wird die Diskettenseite ausgewählt (0 oder 1).

track:	Die Nummer des Tracks, der formatiert werden soll (0 - 79).
spt:	Sektoren pro Track, normalerweise 9.
dev:	Das Laufwerk, 0 für A und 1 für B.
filler:	unbenutztes Langwort
puffer:	Puffer für die kompletten Trackdaten; bei neun Sektoren pro Track muß der Puffer mindestens acht KByte groß sein.

Die Funktion gibt als Ergebnis einen Fehlercode zurück. Der Wert -16, bad sectors, bedeutet, daß bei einigen Sektoren die Daten nicht korrekt zurückgelesen werden konnten. In diesem Fall steht im Puffer eine Liste von schlechten Sektoren (Wort-Daten, durch Null abgeschlossen). Sie können daraufhin nochmal formatieren oder die Sektoren als schlecht markieren.

```

Beispiel:  move.w    #$E5E5, -(sp)    Initialdaten
              move.l    #$87654321, -(sp) magic
              move.w    #1, -(sp)    interleave
              move.w    #0, -(sp)    Seite 0
              move.w    #79, -(sp)   Track 79
              move.w    #9, -(sp)    neun Sektoren pro Track
              move.w    #0, -(sp)    Laufwerk A
              clr.l      -(sp)
              move.l     #puffer, -(sp)
              move.w     #10, -(sp)   flopfmt
              trap       #14
              add.w      #26, sp
              tst        d0
              bmi        error
puffer        ds.b       $2000        8K Puffer

```

11 unbenutzt

12 Midiws

```

C:         void      Midiws (count, ptr)
              int       count;
              long ptr;

```

```

Assembler: move.l    ptr, -(sp)
              move.w    count, -(sp)
              move.w    #12, -(sp)

```

```
trap          #14
addq.l        #8,sp
```

GFA: VOID XBIOS(12,count%,L:ptr%);

OMIKRON: XBIOS(,12,count,HIGH(ptr).LOW(ptr))

Mittels dieser Funktion ist es möglich, einen String an die MIDI-Schnittstelle auszugeben (MIDI OUT). ptr muß dabei auf den String zeigen, count muß die Anzahl der Zeichen minus 1 enthalten, die gesendet werden.

Beispiel:

```
move.l        #string,-(sp)  Adresse des Strings
move.w        #stringend-string-1,-(sp) Länge
move.w        #12,-(sp)      midiws
trap          #14
addq.l        #8,sp

....
string        dc.b          'MIDI-Daten'
stringend     equ          *
```

13 Mfpint

C:

```
void          Mfpint (nummer, vector)
int           nummer;
long          vector;
```

Assembler:

```
move.l        vector,-(sp)
move.w        nummer,-(sp)
move.w        #13,-(sp)
trap          #14
addq.l        #8,sp
```

GFA: VOID XBIOS(13,nummer%,L:vector%)

OMIKRON: XBIOS(,13,nummer,HIGH(vector),LOW(vector))

Damit kann eine Interrupt-Routine im MFP initialisiert werden. In "nummer" muß dabei die Nummer des MFP-Interrupts stehen (0-15), während "vector" die Adresse der zugehörigen Interrupt-Routine enthält. Der alte Interrupt-Vektor wird dabei überschrieben.

Beispiel:

```
move.l        #busy,-(sp)    Busy-Interrupt-Routine
move.w        #0,-(sp)       Vektor-Nummer 0
move.w        #13,-(sp)      mfpint
trap          #14
```



```
addq.l      #8,sp
....
busy:
```

14 lorec

C: long Iorec(dev)
 int dev;

Assembler: move.w dev, -(sp)
 move.w #14, -(sp)
 trap #14
 addq.l #4, sp

GFA: record = XBIOS(14, device%)

OMIKRON: XBIOS(record, 14, device)

Mit dieser Funktion wird ein Zeiger auf einen Puffer-Datensatz für ein Eingabegerät geholt. Folgende Eingabegeräte können angegeben werden:

dev	Eingabegerät
0	RS232
1	Tastatur
2	MIDI

Der Pufferdatensatz für ein Eingabegerät hat folgenden Aufbau:

long	ibuf	Zeiger auf Eingabepuffer
int	ibufsize	Größe des Eingabepuffers
int	ibufhd	Head Index
int	ibuftl	Tail Index
int	ibuflow	Low water mark
int	ibufhi	High water mark

Head-Index

Bei dem Eingabepuffer handelt es sich um einen zirkulären Puffer, der Head-Index gibt die nächste Schreibposition an (der Puffer wird von einer Interrupt-Routine gefüllt), und der Tail-Index gibt an, von wo aus dem Puffer gelesen werden kann. Sind Head- und Tail-Index gleich, so ist der Puffer leer. Low und High water mark werden im Zusammenhang mit der Kommunikationskontrolle bei RS232 benutzt (XON/XOFF oder RTS/CTS). Ist der Eingabepuf-

fer bis zur High water mark gefüllt, so wird dem Sender über XOFF bzw. CTS mitgeteilt, daß der Rechner keine weiteren Daten mehr empfangen kann. Werden die empfangenen Daten dann vom Rechner weiterverarbeitet, so daß der Pufferinhalt unter die Low water mark sinkt, so wird die Übertragung wieder freigegeben.

Für die RS232-Ausgabe existiert ein gleich aufgebauter Puffer-Datensatz, der sich direkt an den Eingabe-Datensatz anschließt.

Beispiel:

```

move.w    #1,-(sp)    Pufferdatensatz für Tastatur
move.w    #14,-(sp)   iorec
trap      #14
addq.l    #4,sp
....

```

Ergebnis: \$DB0

Die folgende Tabelle enthält die Daten für alle Geräte.

	RS232 Eingabe	RS232 Ausgabe	Tastatur	MIDI
Adresse	\$D8E	(\$D9C)	\$DB0	\$DBE
Pufferadresse	\$A0E	\$B0E	\$C0E	\$D0E
Pufferlänge	\$100	\$100	\$100	\$80
Head Index	0	0	0	0
Tail Index	0	0	0	0
Low water mark	\$40	\$40	\$40	\$20
High water mark	\$C0	\$C0	\$C0	\$60

Head- und Tail-Index sind natürlich vom augenblicklichen Betriebszustand abhängig. High und Low water mark sind jeweils auf 3/4 bzw. 1/4 der Puffergröße gesetzt. Sie sind nur bei eingeschaltetem XON/XOFF bzw. RTS/CTS bei RS232 von Bedeutung.

15 Rsconf

C:

```

void      Rsconf(baud, ctrl, ucr, rsr, tsr, scr)
int       baud, ctrl, ucr, rsr, tsr, scr;

```

Assembler:

```

move.w    scr,-(sp)
move.w    tsr,-(sp)
move.w    rsr,-(sp)
move.w    ucr,-(sp)
move.w    ctrl,-(sp)

```

```

move.w    baud, -(sp)
move.w    #15, -(sp)
trap      #14
add.w     #14, sp

```

GFA: VOID XBIOS(15,baud%,ctrl%,ucr%,rsr%,tsr%,scr%)

OMIKRON: XBIOS(,15,baud,ctrl,ucr,rsr,tsr,scr)

Diese XBIOS-Funktion dient zum Konfigurieren der RS232-Schnittstelle. Die Parameter haben dabei folgende Bedeutung:

scr: Synchronus-Character-Register im MFP
tsr: Transmitter-Status-Register im MFP
rsr: Receiver-Status-Register im MFP
ucr: USART-Control-Register im MFP
ctrl: Kommunikationsparameter
baud: Baudrate

Zur Bedeutung der MFP-Register schlagen Sie bitte im Kapitel über den MFP 68901 nach. Ist einer der Parameter -1, so wird der vorherige Wert beibehalten. Mit dem ctrl-Parameter können Sie den Handshake-Modus wählen:

ctrl	Bedeutung
0	kein Handshake, default nach dem Einschalten
1	XON/XOFF
2	RTS/CTS
3	XON/XOFF und RTS/CTS (nicht sinnvoll)

Der baud-Parameter enthält eine Kennziffer für die Baudrate.

baud	Baudrate
0	19200
1	9600
2	4800
3	3600
4	2400
5	2000
6	1800
7	1200
...	

...	
8	600
9	300
10	200
11	150
12	134
13	110
14	75
15	50

Beispiel:

```

move.w    #-1,-(sp)
move.w    #-1,-(sp)      MFP-Register nicht verändern
move.w    #-1,-(sp)
move.w    #-1,-(sp)
move.w    #1,-(sp)       XON/XOFF
move.w    #9,-(sp)       300 Baud
move.w    #15,-(sp)      rsconf
trap      #14
add.w     #14,sp

```

16 Keytbl

C:

```

long      Keytbl (unshift, shift, capslock)
long      unshift, shift, capslock;

```

Assembler:

```

move.l    capslock,-(sp)
move.l    shift,-(sp)
move.l    unshift,-(sp)
move.w    #16,-(sp)
trap      #14
add.w     #14,sp

```

GFA: oldtab = XBIOS(16,L:unshift%,L:shift%,L:capslock%)

OMIKRON: XBIOS(oldtab,16,HIGH(unshift),LOW(unshift),HIGH(shift),LOW(shift),HIGH(cpaslock),LOW(capslock))

Mit dieser Funktion ist es möglich, eine neue Tastaturbelegung zu erzeugen. Dazu müssen Sie die Adressen der neuen Tabellen übergeben, die die Tastencodes bei normalen Tasten (ohne **Shift**), bei gedrückter **Shift**-Taste und bei Capitals Lock enthalten. Die Funktion liefert als Ergebnis die Adresse der Vektortabelle, in der die drei Tastaturtabellenzeiger stehen. Soll eine Tabelle unverändert bleiben, so muß als Adresse -1 übergeben werden. Eine Tastatur-

tabelle muß 128 Bytes lang sein. Sie wird über den Tasten-Scancode adressiert und liefert als Ergebnis den ASCII-Code der jeweiligen Taste.

Beispiel:

move.l	#-1,-(sp)	Caps Lock nicht ändern
move.l	#shift,-(sp)	Shift-Tabelle
move.l	#unshift,-(sp)	Tabelle ohne Shift
move.w	#16,-(sp)	
trap	#14	
add.w	#14,sp	
....		
shift:	...	
unshift	...	

17 Random

C: long Random()

Assembler:

move.w	#17,-(sp)
trap	#14
addq.l	#2,sp

GFA: random = XBIOS(17)

OMIKRON: XBIOS(random,17)

Diese Funktion liefert eine 24-Bit-Zufallszahl. Die Bits 24-31 sind dabei Null. Bei jedem Aufruf erhalten Sie ein anderes Ergebnis. Nach dem Einschalten des Rechners wird jedesmal ein anderer Startwert erzeugt.

Beispiel:

move.w	#17,-(sp)	random
trap	#14	
addq.l	#2,sp	

18 Protobt

C:

void	Protobt(puffer, serialnr, disktyp, execflag)
long	puffer, serialnr;
int	disktyp, execflag;

Assembler:

move.w	execflag,-(sp)
move.w	disktyp,-(sp)
move.l	serialnr,-(sp)
move.l	puffer,-(sp)
move.w	#18,-(sp)

trap #14
add.w #14,sp

GFA: VOID XBIOS(18,L:puffer%,L:serial%,disktyp%,execflag%)

OMIKRON: XBIOS(,18,HIGH(puffer),LOW(puffer),HIGH(serial),LOW(serial),
 disktyp,execflag)

Diese Funktion dient dazu, einen Boot-Sektor zu erzeugen. Ein Boot-Sektor steht auf Track 0, Sektor 1 auf Seite 0 einer Diskette und gibt dem DOS Auskunft über den Diskettentyp. Ist der Boot-Sektor ausführbar, kann damit das Betriebssystem geladen werden. Mit dieser Funktion können Sie einen neuen Boot-Sektor erzeugen, etwa für ein anderes Diskettenformat, oder einen vorhandenen Boot-Sektor verändern. Die Parameter haben folgende Bedeutung:

execflag: Damit wird entschieden, ob der Boot-Sektor ausführbar ist:
0 nicht ausführbar
1 ausführbar
-1 Boot-Sektor bleibt so, wie er war

Der Disktyp kann folgende Werte annehmen:

0 40 Track, single sided (180 K)
1 40 Track, double sided (360 K)
2 80 Track, single sided (360 K)
3 80 Track, double sided (720 K)
-1 Disktyp bleibt unverändert

Serialnr: Ist eine 24-Bit-Seriennummer, die in den Boot-Sektor geschrieben wird. Ist die Seriennummer größer als 24 Bits (\$01000000), so wird eine Zufalls-Seriennummer erzeugt (mit obiger Funktion). Ein Wert von -1 bedeutet, daß die Seriennummer nicht verändert wird.

Puffer: Ist die Adresse eines 512-Byte-Puffers, der den Boot-Sektor enthält bzw. in dem der Boot-Sektor erzeugt wird.

Ein Boot-Sektor hat folgenden Aufbau:

Adresse	Track SS	40 Track DS	80 Track SS	80 Track DS	80 Track HD
0 -1	Branch-Befehl auf Boot-Programm, sofern ausführbar				
2 -7	'Loader'				
8 -10	24-Bit-Seriennummer				
11-12	BPS	512	512	512	512
13	SPC	1	2	2	2
14-15	RES	1	1	1	1
16	FAT	2	2	2	2
17-18	DIR	64	112	112	112
19-20	SEC	360	720	720	1440
21	MEDIA	252	253	248	249
22-23	SPF	2	2	5	5
24-25	SPT	9	9	9	9
26-27	SIDE	1	2	1	2
28-29	HID	0	0	0	0
510-511	CHECKSUM				

Die Abkürzungen haben dabei folgende Bedeutung:

- BPS:** Bytes pro Sektor. Die Sektorgröße ist bei allen Formaten einheitlich 512 Bytes
- SPC:** Sektoren pro Cluster. Die Anzahl der Sektoren, die vom DOS zu einem Block zusammengefaßt werden, 2 Sektoren gleich 1 KByte.
- RES:** Anzahl der reservierten Sektoren am Beginn der Diskette einschließlich dem Boot-Sektor.
- FAT:** Die Anzahl der File Allocation Tables auf der Diskette.
- DIR:** Die maximale Anzahl der Directory-Einträge.
- SEC:** Die Gesamtzahl der Sektoren auf der Diskette.
- MEDIA:** Media Descriptor Byte, wird vom ST-BIOS nicht benutzt.
- SPF:** Anzahl der Sektoren in jedem FAT.
- SPT:** Anzahl Sektoren pro Track.
- SIDE:** Anzahl der Seiten auf der Diskette.
- HID:** Anzahl der versteckten Sektoren auf der Diskette.

Der Boot-Sektor ist weitgehend kompatibel zu MS-DOS 2.x. Deshalb sind alle 16-Bit-Worte im 8086-Format abgespeichert (erst Low-Byte, dann High-Byte).

Ist die Checksumme des gesamten Boot-Sektors \$1234, so ist der Sektor ausführbar. In diesem Falle steht ab Adresse 30 das Boot-Programm.

Beispiel:

```

move.w    #-1,-(sp)    Ausführbarkeit nicht ändern
move.w    #3,-(sp)     80 Tracks, DS
move.l    #-1,-(sp)     Seriennummer nicht ändern
move.l    #puffer,-(sp)
move.w    #18,-(sp)     protobt
trap      #14
add.w     #14,sp
puffer    ds.b 512
    
```

Mit diesem Beispielprogramm kann z.B. ein vorhandener Boot-Sektor für 80 Tracks, double sided, angepaßt werden.

19 Flopver

C:

```

int        Flopver(puffer, filler, dev, sector, track, seite,
                  count)
long       puffer, filler;
int        dev, sector, track, seite, count;
    
```

Assembler:

```

move.w     count,-(sp)
move.w     seite,-(sp)
move.w     track,-(sp)
move.w     sector,-(sp)
move.w     dev,-(sp)
clr.l      -(sp)
move.l     puffer,-(sp)
move.w     #19,-(sp)
trap      #14
add.w     #16,sp
    
```

GFA:

```

status=XBIOS(19,L:puffer%,L:filler%,dev%,sector%,track%,seite%,
count%)
    
```

OMIKRON:

```

XBIOS(status,19,HIGH(puffer),LOW(puffer),0,0,dev,sector,track,
seite,count)
    
```

Diese Funktion dient zum Verifizieren eines oder mehrerer Sektoren auf der Diskette. Dabei werden die Sektoren von Diskette gelesen und mit dem Pufferinhalt im Speicher verglichen. Die Parameter haben die gleiche Bedeutung wie bei den Funktionen zum Lesen und Schreiben von Sektoren. Stimmen Sektor-

und Pufferinhalt überein, so wird als Ergebnis der Funktion Null zurückgegeben. Bei einem Fehler steht die negative Fehlernummer in D0. Die Bedeutung dieses Fehlercodes steht bei der Funktion "Sektor lesen". Im Fehlerfalle steht im Puffer eine Liste der fehlerhaften Sektoren (16-Bit-Werte), die durch ein Nullwort abgeschlossen ist. Benutzt man zum Schreiben von Sektoren die BIOS-Funktion 4 rwabs und ist die Variable fverify (\$444) gesetzt, so werden die Sektoren nach dem Schreiben automatisch verifiziert.

Beispiel:

move.w	#1,-(sp)	ein Sektor
move.w	#0,-(sp)	Seite Null
move.w	#39,-(sp)	Track 39
move.w	#1,-(sp)	Sektor 1
move.w	#0,-(sp)	Laufwerk A
clr.l	-(sp)	
move.l	#puffer,-(sp)	Pufferadresse
move.w	#19,-(sp)	flopver
trap	#14	
add.w	#16,sp	
tst	d0	Fehler?
bmi	error	

20 Scrdmp

C: void Scrdmp()

Assembler:

move.w	#20,-(sp)
trap	#14
addq.l	#2,sp

GFA: VOID XBIOS(20)

OMIKRON: XBIOS(,20)

Mit dieser Funktion wird eine Bildschirm-Hardcopy auf einen angeschlossenen Drucker ausgegeben. Dabei werden die voreingestellten Druckerparameter ("Drucker Anpassung") benutzt. Diese Funktion können Sie jederzeit auch durch gleichzeitiges Drücken der Tasten **Alternate** und **Help** auslösen oder vom Desktop aus durch "Bildschirm Drucken".

Beispiel:

move.w	#20,-(sp)	Hardcopy
trap	#14	XBIOS aufrufen
addq.l	#2,sp	

21 Cursconf

C: int Cursconf(funktion, rate)
 int funktion, rate;

Assembler: move.w rate, -(sp)
 move.w funktion, -(sp)
 move.w #21, -(sp)
 trap #14
 addq.l #6, sp

GFA: newrate% = XBIOS(21, funktion%, rate%)

OMIKRON: XBIOS(newrate, 21, funktion, rate)

Diese XBIOS-Funktion dient zur Bestimmung der Cursor-Funktion. "funktion" kann dabei Werte von 0-5 haben, die folgende Bedeutung haben:

Funktion	Bedeutung
0	Cursor abschalten
1	Cursor einschalten
2	Cursor blinkend
3	Cursor nicht blinkend
4	Cursor Blinkrate setzen
5	Cursor Blinkrate holen

Sie können mit dieser Funktion also bestimmen, ob der Cursor sichtbar ist, ob er feststehend ist oder blinkt. Ein Ergebnis liefert diese XBIOS-Funktion nur, wenn Sie die alte Blinkrate holen. Die Einheit der Blinkfrequenz ist abhängig von der Bildfrequenz: Bei einem Monochrommonitor beträgt Sie 70 Hz; ein Farbmonitor wird mit 50 Hz betrieben. Mit Funktionsnummer 5 können Sie eine neue Blinkrate einstellen. Den Parameter "rate" brauchen Sie nur dann zu benutzen, wenn Sie eine neue Blinkrate übergeben wollen.

Beispiel: move.w #20, -(sp) 20/70 Sekunden
 move.w #4, -(sp) Blinkrate setzen
 move.w #21, -(sp) cursconf
 trap #14
 addq.l #6, sp

22 Settime

C: void Settime(time)
 long time;

Assembler: move.l time, -(sp)
 move.w #22, -(sp)
 trap #14
 addq.l #6, sp

GFA: VOID XBIOS(22,L:time%)

OMIKRON: XBIOS(,22,HIGH(time),LOW(time))

Mit dieser Funktion können Sie Uhrzeit und Datum neu setzen. Die Zeit wird dabei im unteren Wort von time übergeben, das Datum im oberen Wort. Zeit und Datum sind folgendermaßen kodiert:

Bit 0-4 Sekunden in Zweierschritten
Bit 5-10 Minuten
Bit 11-15 Stunden
Bit 16-20 Tag 1-31
Bit 21-24 Monat 1-12
Bit 25-31 Jahr 0-119 (minus Offset 1980)

Beispiel: move.l #%1011001100000100000000000000, -(sp)
 move.w #22, -(sp) settime
 trap #14
 addq.l #6, sp

Dieser Aufruf setzt das Datum auf den 16. September 1985 und die Uhrzeit auf 8 Uhr.

23 Gettime

C: long Gettime()

Assembler: move.w #23, -(sp)
 trap #14
 addq.l #2, sp

GFA: time = XBOIS(23)

OMIKRON: XBIOS(time,23)

Diese Funktion liefert das aktuelle Datum und die Uhrzeit im oben beschriebenen Format.

Beispiel:

	move.w	#23,-(sp)	gettime
	trap	#14	
	addq.l	#2,sp	
	move.l	d0,time	Zeit und Datum merken

24 Bioskeys

C: void Bioskeys()

Assembler:

move.w	#24,-(sp)
trap	#14
addq.l	#2,sp

GFA: VOID XBIOS(24)

OMIKRON: XBIOS(,24)

Wenn Sie mit der XBIOS-Funktion 16, keytbl, eine neue Tastaturbelegung gewählt haben, stellt diese Funktion wieder die Standardtastaturbelegung des BIOS her. Sie können diese Funktion z.B. vor dem Verlassen eines eigenen Programms aufrufen, das die Tastaturbelegung verändert hat.

Beispiel:

move.w	#24,-(sp)	bioskeys
trap	#14	
addq.l	#2,sp	

25 lkbdws

C:

void	lkbdws(anzahl, zeiger)
int	anzahl;
long	zeiger;

Assembler:

move.l	zeiger,-(sp)
move.w	anzahl,-(sp)
move.w	#25,-(sp)
trap	#14
addq.l	#8,sp

GFA: VOID XBIOS(25,anzahl%,L:zeiger%)

OMIKRON: XBIOS(,25,anzahl,HIGH(zeiger),LOW(zeiger))

Zum Übermitteln von Befehlen an den Tastaturprozessor (intelligent keyboard) dient diese XBIOS-Funktion. "zeiger" ist dabei die Adresse eines Strings, der gesendet werden soll, "anzahl" ist die Länge des Strings minus 1.

Beispiel:

```

move.l    #string,-(sp)  Adresse des Strings
move.w    #strend-string-1,-(sp)  Länge minus 1
move.w    #25,-(sp)      ikbdws
trap      #14
addq.l    #8,sp
...
string    dc.b            $80,1
strend    equ             *
```

26 Jdisint

C:

```

void      Jdisint(nummer)
int        nummer;
```

Assembler:

```

move.w    nummer,-(sp)
move.w    #26,-(sp)
trap      #14
addq.l    #4,sp
```

GFA: VOID XBIOS(26,nummer%)

OMIKRON: XBIOS(,26,nummer)

Diese Funktion ermöglicht es, einen Interrupt des MFP 68901 selektiv zu sperren. Als Parameter wird dabei die Interrupt-Nummer des MFP übergeben (0 - 15). Die Bedeutung der einzelnen Interrupts ist im Kapitel "Interrupt" beschrieben.

Beispiel:

```

move.w    #10,-(sp)    RS232-Transmitter-Interrupt sperren
move.w    #26,-(sp)    disable Interrupt
trap      #14
addq.l    #4,sp
```

27 Jenabint

C:

```

void      Jenabint(nummer)
int        nummer;
```

Assembler: move.w nummer, -(sp)
 move.w #27, -(sp)
 trap #14
 addq.l #4, sp

GFA: VOID XBIOS(27, nummer%)

OMIKRON: XBIOS(, 27, nummer)

Soll ein Interrupt des MFP wieder freigegeben werden, so können Sie dazu diese Funktion benutzen. Übergeben wird dabei wieder die Nummer des Interrupts von 0 - 15.

Beispiel: move.w #12, -(sp) RS232-Receiver-Interrupt freigeben
 move.w #27, -(sp) enable Interrupt
 trap #14
 addq.l #4, sp

28 Giaccess

C: char Giaccess(daten, register)
 char daten;
 int register;

Assembler: move.w #register, -(sp)
 move.w #daten, -(sp)
 move.w #28, -(sp)
 trap #14
 addq.l #6, sp

GFA: status%=XBIOS(28, daten%, register%)

OMIKRON: XBIOS(status, 28, daten, register)

Diese Funktion erlaubt den Zugriff auf die Register des GI-Soundchips. "register" muß dabei die Registernummer des Soundchips enthalten (0-15). Die Bedeutung der einzelnen Register entnehmen Sie bitte der Hardware-Beschreibung des Soundchips. Ob das angegebene Register gelesen werden soll oder ob in das Register geschrieben wird, wird durch Bit 7 der Registernummer entschieden:

Bit 7 0: lesen
 1: schreiben

Beim Schreiben wird in "daten" der zu schreibende 8-Bit-Wert übergeben; beim Lesen liefert die Funktion den Inhalt des entsprechenden Registers.

Beispiel:

```

move.w    #$80+3,-(sp)   Register 3 schreiben
move.w    #$50,-(sp)     zu schreibender Wert
move.w    #28,-(sp)      giaccess
trap      #14
addq.l    #6,sp

```

29 Offgibit

C:

```

void      Offgibit(bitmaske)
int        bitmaske;

```

Assembler:

```

move.w    #bitmaske,-(sp)
move.w    #29,-(sp)
trap      #14
addq.l    #4,sp

```

GFA: VOID XBIOS(29,bitmaske%)

OMIKRON: XBIOS(,29,bitmaske)

Mit diesem Funktionsaufruf lassen sich selektiv Bits im Port A des Soundchips löschen. In der Bitmaske müssen die zu löschenden Bits auf Null, alle anderen auf Eins sein. Port A ist ein 8-Bit-Ausgabe-Port, in dem die einzelnen Bits folgende Bedeutung haben:

```

Bit 0:      Floppy Seite 0/ Seite 1 selektieren
Bit 1:      Floppy-Laufwerk A selektieren
Bit 2:      Floppy-Laufwerk B selektieren
Bit 3:      RS232 RTS (Request to Send)
Bit 4:      RS232 DTR (Data Terminal Ready)
Bit 5:      Centronics Strobe
Bit 6:      General Purpose Output
Bit 7:      unbenutzt

```

Beispiel:

```

move.w    #$EF,-(sp)     DTR-Bit löschen, Bit 4
move.w    #29,-(sp)      offgibit
trap      #14
addq.l    #4,sp

```

30 Ongibit

C: void Ongibit(bitmaske)
 int bitmaske;

Assembler: move.w #bitmaske, -(sp)
 move.w #30, -(sp)
 trap #14
 addq.l #4, sp

GFA: VOID XBIOS(30, bitmaske%)

OMIKRON: XBIOS(, 30, bitmaske%)

Das Gegenstück zu der obigen Funktion bildet diese. Damit ist es möglich, ein oder mehrere Bits im Port A des Soundchips zu setzen.

Beispiel: move.w #\$10, -(sp) DTR-Bit setzen, Bit 4
 move.w #30, -(sp) ongibit
 trap #14
 addq.l #4, sp

31 Xbtimer

C: void Xbtimer(timer, control, data, vector)
 int timer, control, data;
 long vector;

Assembler: move.l vector, -(sp)
 move.w data, -(sp)
 move.w control, -(sp)
 move.w timer, -(sp)
 move.w #31, -(sp)
 trap #14
 add.w #12, sp

GFA: VOID BIOS(31, timer%, control%, data%, L:vector%)

OMIKRON: XBIOS(, 32, timer, control, data, HIGH(vector), LOW(vector))

Mit dieser Funktion können Sie einen Timer im MFP 68901 starten und ihm eine Interrupt-Routine zuordnen. "timer" ist die Nummer des Timers im MFP:

Timer A : 0
Timer B : 1

Timer C : 2
 Timer D : 3

"data" und "control" sind die Werte, die in das entsprechende Kontroll- und Datenregister des Timers übertragen werden. Beachten Sie dazu die Hardware-Beschreibung des MFP 68901. "vector" ist die Adresse der zugehörigen Interrupt-Routine, die nach Ablauf des Timers angesprungen wird. Die vier Timer des MFP werden teilweise schon vom Betriebssystem benutzt:

Timer A: Für den Endbenutzer reserviert
 Timer B: Horizontal-Blank-Zähler
 Timer C: 200 Hz System Timer
 Timer D: RS232 Baudrate-Generator

(der zugehörige Interruptvektor ist frei)

Beispiel:

```

move.l    #vector, -(sp)  Interrupt-Routine
move.w    data, -(sp)    Daten-
move.w    control, -(sp) und Kontroll-Register
move.w    #0, -(sp)      Timer A
move.w    #31, -(sp)     xbtimer
trap      #14
add.w     #12, sp
```

32 Dosound

C:

```

void      Dosound(zeiger)
long      zeiger;
```

Assembler:

```

move.l    zeiger, -(sp)
move.w    #32, -(sp)
trap      #14
addq.l    #6, sp
```

GFA: VOID XBIOS(32, L:zeiger%)

OMIKRON: XBIOS(, 32, HIGH(zeiger), LOW(zeiger))

Diese Funktion erlaubt eine automatische komfortable Abarbeitung eines Sounds. Dazu muß "zeiger" auf einen String aus Sound-Befehlen zeigen. Folgende Befehle können benutzt werden:

Befehle \$00 - \$0F

Diese Befehle werden als Registernummer des Soundchips interpretiert. Ihnen folgt jeweils ein Byte, das in das entsprechende Register geladen wird.

Befehl \$80

Diesem Befehl folgt ein Argument, das in ein temporäres Register geladen wird.

Befehl \$81

Diesem Befehl müssen drei Argumente folgen. Das erste Argument ist die Nummer des Registers des Soundchips, in das der Inhalt des temporären Registers geladen wird. Das zweite Argument ist ein Zweierkomplementwert, der zu dem temporären Register addiert wird. Das dritte Argument enthält ein Endekriterium. Das Ende ist erreicht, wenn der Inhalt des temporären Registers gleich dem Endekriterium ist.

Befehle \$82 - \$FF

Diesen Befehlen folgt jeweils ein weiteres Argument. Lautet dieses Argument null, so wird die Sound-Verarbeitung abgebrochen. Ansonsten gibt dieses Argument an, wie viele Timerticks (20ms, 50 Hz) bis zur nächsten Verarbeitung des Sounds vergehen.

Beispiel:

	move.l	#zeiger, -(sp)	Zeiger auf Sound-Befehle
	move.w	#32, -(sp)	dosound
	trap	#14	
	addq.l	#6, sp	
			
zeiger	dc.b	0,10,1,50,...	

33 Setprt

C:

int	Setprt (config)
int	config;

Assembler:

move.w	config, -(sp)
move.w	#33, -(sp)
trap	#14
addq.l	#4, sp

GFA:

set%	= XBIOS(33, config%)
------	----------------------

OMIKRON: XBIOS(set,33,config)

Mittels dieser Funktion kann die eingestellte Druckerkonfiguration gelesen oder verändert werden. Enthält "config" den Wert -1, so wird nur der augenblickliche Wert geliefert, ansonsten wird der Wert als neue Druckerkonfiguration übernommen.

Die Druckerkonfiguration ist ein Bit-Vektor mit folgender Bedeutung:

Bit-Nummer	0	1
0	Matrixdrucker	Typenraddrucker
1	Monochromdrucker	Farbdrucker
2	Atari-Drucker	Epson-Drucker
3	Test-Modus	Qualitäts-Modus
4	Centronics-Port	RS232-Port
5	Endlospapier	Einzelblatt
6 - 14	reserviert	
15	immer 0	

Beispiel:

```

move.w    #000100, -(sp) Epson-Drucker
move.w    #33, -(sp)    setprt
trap      #14
addq.l    #4, sp
    
```

34 Kbdvbase

C: long Kbdvbase()

Assembler:

```

move.w    #34, -(sp)
trap      #14
addq.l    #2, sp
    
```

GFA: base = XBIOS(34)

OMIKRON: XBIOS(base,34)

Diese XBIOS-Funktion liefert einen Zeiger auf eine Vektortabelle, die die Adressen von Routinen enthält, die Tastatur- und MIDI-Daten verarbeiten. Die Tabelle ist folgendermaßen aufgebaut:

long	midivec	MIDI-Eingabe
long	vkbderr	Tastatur-Error
long	vmiderr	MIDI-Error

long	statvec	IKBD-Status
long	mousevec	Mouse-Routinen
long	clockvec	Uhrzeit-Routine
long	joyvec	Joystick-Routinen
long	midisys	MIDI-Systemvektor
long	ikbdsys	IKBD-Systemvektor

"midivec" zeigt auf eine Routine, die die empfangenen Daten vom MIDI-Eingang (Byte in D0) in den MIDI-Puffer schreibt.

"vkbderr" und "vmiderr" werden aufgerufen, wenn ein Überlauf von der Tastatur- oder MIDI-ACIA gemeldet wird.

Die restlichen vier Routinen "statvec", "mousevec", "clockvec" und "joyvec" verarbeiten die entsprechenden Datenpakete, die von der Tastatur-ACIA kommen. Diesen Routinen wird ein Zeiger auf das empfangene Paket in A0 und auf dem Stack übergeben. Der Mouse-Vektor wird von GEM benutzt. Wenn Sie eigene Routinen benutzen wollen, so müssen sie mit RTS abgeschlossen werden und dürfen nicht länger als eine Millisekunde Verarbeitungszeit benötigen.

Die Routinen "midisys" und "ikbdsys" werden aufgerufen, wenn ein Zeichen an der betreffenden ACIA zur Verfügung steht. "midisys" holt das Zeichen und springt dann über "midivec"; "ikbdsys" holt ein Datenpaket von der ACIA und verzweigt dann in die oben beschriebenen Unterrouinen.

Beispiel:

```

move.w    #34,-(sp)    kbdvbase
trap      #14
addq.l    #2,sp

```

Als Ergebnis erhalten wir \$DCC. Das Vektorfeld enthält folgende Werte:

midivec	\$FC2CE2/\$8B70
vkbderr	\$FC288E/\$871C (rts)
vmiderr	\$FC288E/\$871C (rts)
statvec	\$FC230A/\$8198 (rts)
mousevec	\$FD02C2/\$16150
clockvec	\$FC1D12/\$7BA0
joyvec	\$FC230A/\$8198 (rts)
midisys	\$FC284A/\$86D8
ikbdsys	\$FC285A/\$86E8

35 Kbrate

C: int Kbrate(verzoeg, wiederhol)
 int verzoeg, wiederhol;

Assembler: move.w wiederhol, -(sp)
 move.w verzoeg, -(sp)
 move.w #35, -(sp)
 trap #14
 addq.l #6, sp

GFA: erg% = XBIOS(35, verzoeg%, wiederhol%)

OMIKRON: XBIOS(erg, 35, verzoeg, wiederhol)

Mit dieser Funktion kann der Tastatur-Repeat kontrolliert werden. Der Parameter "verzoeg" gibt die Verzögerungszeit nach dem Drücken einer Taste an, nach der die Taste automatisch wiederholt wird. Der Parameter "wiederhol" bestimmt die Zeitspanne, nach der der Tastendruck jeweils wiederholt wird. Diese Werte können Sie vom Desktop aus mittels des Kontrollfeldes mit den beiden Schiebereglern ändern. Die Zeiten basieren auf dem 50-Hz-Systemtakt. Wird für einen der Parameter -1 angegeben, so wird der entsprechende Wert nicht gesetzt. Die Funktion liefert die vorherigen Werte als Ergebnis; Bit 0-7 enthalten den "wiederhol"-Wert und Bit 8-15 den Wert von "verzoeg".

Beispiel: move.w #-1, -(sp) alte Werte lesen
 move.w #-1, -(sp)
 move.w #35, -(sp) kbrate
 trap #14
 addq.l #6, sp
 Ergebnis: D0 = \$0B03

36 Prtblk

C: void Prtblk(parameter)
 long parameter;

Assembler: move.l parameter, -(sp)
 move.w #36, -(sp)
 trap #14
 addq.l #6, sp

GFA: VOID XBIOS(36, L:parameter%)

OMIKRON: XBIOS(, 36, HIGH(parameter), LOW(parameter))

Diese Funktion ähnelt der Funktion `scrdmp` (20) und wird von dieser benutzt. Die Funktion erwartet jedoch eine Parameterliste, deren Adresse übergeben wird. Diese Liste ist folgendermaßen aufgebaut:

long	blkprt	Adresse des Bildschirm-RAMs
int	offset	
int	width	Bildschirmbreite
int	height	Bildschirmhöhe
int	left	
int	right	
int	scrres	Bildschirmauflösung (0, 1 oder 2)
int	dstres	Druckerauflösung (0 oder 1)
long	colpal	Adresse der Farbpalette
int	type	Druckertyp (0-3)
int	port	Druckerport (0=Centronics, 1=RS232)
long	masks	Zeiger auf Halbtontmaske

Beispiel:

```

move.l    #parameter, -(sp)  Adresse des Parameterblocks
move.w    #36, -(sp)         prtblk
trap      #14
addq.l    #6, sp
....
parameter dc.l    ...

```

37 Vsync

C: void Vsync()

Assembler:

```

move.w    #36, -(sp)
trap      #14
addq.l    #2, sp

```

GFA: VOID XBIOS(36)

OMIKRON: XBIOS(,36)

Diese Funktion wartet auf den nächsten Bildrücklauf. Man kann sie z.B. benutzen, um Grafikausgaben mit dem Strahlrücklauf zu synchronisieren.

Beispiel:

```

move.w    #36, -(sp)    vsync abwarten
trap      #14
addq.l    #2, sp

```

38 Supexec

C: void Supexec(adresse)
 long adresse;

Assembler: move.l adresse, -(sp)
 move.w #38, -(sp)
 trap #14
 addq.l #6, sp

GFA: VOID XBIOS(38, L:adresse%)

OMIKRON: XBIOS(, 38, HIGH(adresse), LOW(adresse))

Soll eine Routine im Supervisor-Modus des 68000-Prozessors ausgeführt werden, so können Sie dies mit dieser Funktion erreichen. Dazu übergibt man einfach die Adresse der Routine an diese Funktion. Die aufzurufende Routine muß mit rts abgeschlossen sein.

Beispiel: move.l #adresse, -(sp)
 move.w #38, -(sp)
 trap #14
 addq.l #6, sp
 ...
adresse move.l \$400, d0
 ...
 rts

39 Puntaes

C: void Puntaes()

Assembler: move.w #39, -(sp)
 trap #14
 addq.l #2, sp

GFA: XBIOS(39)

OMIKRON: XBIOS(, 39)

Mit dieser Funktion kann das AES abgeschaltet werden, sofern es sich nicht im ROM befindet.

Beispiel: move.w #39, -(sp) AES abschalten
 trap #14
 addq.l #2, sp

64 Blitmode

C: int Blitmode(flag)
 int flag;

Assembler: move.w flag, -(sp)
 move.w #64, -(sp)
 trap #14
 addq.l #4, sp

GFA: mode%=XBIOS(64, flag%)

OMIKRON: XBIOS(mode, 64, flag)

Mit dieser XBIOS-Funktion läßt sich die Konfiguration eines evtl. eingebauten Blitters lesen und verändern. Wenn der Parameter "flag" den Wert -1 hat (0xFFFF), so wird die augenblickliche Konfiguration gelesen. Das Ergebnis ist eine Bitmaske, deren einzelne Bits folgende Bedeutung haben:

Bit	0	1
0	Blit-Operation per Software	Blit-Operationen per Hardware (Blitter)
1	kein Blitter vorhanden	Blitter ist vorhanden
2-14	undefiniert, reserviert	
15	immer 0	

Mit diesem Funktionsaufruf läßt sich also feststellen, ob im System ein Blitter vorhanden ist (Bit 1) und ob dieser benutzt wird (Bit 0).

Wenn ein Blitter vorhanden ist, so kann man durch Setzen oder Löschen von Bit 0 bestimmen, ob Blit-Operationen per Software oder mittels des Blitters durchgeführt werden sollen.

Bit	0	1
0	Blit-Operation per Software	Blit-Operationen per Hardware (Blitter)
1-14	undefiniert, reserviert	
15	muß 0 sein	

Beispiel:

move	#-1,-(sp)	konfiguration erfahren
move	#64,-(sp)	blitmode
trap	#xbios	
addq.l	#4,sp	
btst	#1,d0	ist blitter vorhanden?
beq	no_blit	-> nein
bset	#0,d0	
move	d0,-(sp)	blit-operationen per hardware
move	#64,-(sp)	blit-mode
trap	#14	
addq.l	#4,sp	

no_blit:
 rts

In obigem Beispielprogramm wird geprüft, ob ein Blitter vorhanden ist. Ist dies der Fall, so wird mit gesetztem Bit 0 dem System angezeigt, Blit-Operationen per Hardware, also mit dem Blitter durchzuführen. Der Versuch, den Blit-Mode auf "Hardware" zu setzen, wird bei nicht vorhandenem Blitter vom System ignoriert.

Beim Setzen des Blit-Modes sollte immer erst die Konfiguration geholt und die Bits 1-14 sollten übernommen werden. Sie sind für weitere Graphik-Funktionen oder -Chips reserviert.

Diese XBIOS-Funktion kann auf allen Rechnern mit ROM-TOS benutzt werden; unabhängig davon, ob sie einen Blitter enthalten oder nicht.

24. Der VT52-Emulator des Atari ST

Bei der Textausgabe auf den Bildschirm haben Sie beim Atari ST zwei Möglichkeiten. Zum einen können Sie sich der GEMDOS-Funktionen mittels TRAP#1 oder eines direkten BIOS-Aufrufs mit TRAP#13 bedienen. Die andere Möglichkeit besteht in der Nutzung der VDI-Funktionen.

Bei beiden Varianten haben Sie spezielle Möglichkeiten zur Bildschirmsteuerung. Betrachten wir zunächst die Ausgabe über die normalen DOS- bzw. BIOS-Aufrufe. Hier wurde bei der Bildschirmausgabe ein Terminal vom Typ VT52 emuliert, das eine ganze Anzahl von Steuerfunktionen zur Verfügung stellt. Diese Steuerfunktionen werden durch ein spezielles Sonderzeichen eingeleitet, den Escape-Code. Escape oder abgekürzt ESC hat den ASCII-Code 27. Dem ESC-Code folgen noch ein Buchstabe, der die Funktion bestimmt, sowie bei Bedarf einige weitere Parameter. In der folgenden Aufstellung sind sämtliche Steuercodes und ihre Bedeutung aufgelistet.

ESC A Cursor hoch

Damit wird der Cursor um eine Zeile nach oben bewegt. Befand sich der Cursor schon in der obersten Zeile, so passiert nichts.

ESC B Cursor runter

Diese ESC-Sequenz positioniert den Cursor um eine Zeile nach unten. Befindet sich der Cursor schon in der untersten Zeile, so geschieht wiederum nichts.

ESC C Cursor rechts

Um eine Spalte nach rechts wird der Cursor durch diese Sequenz bewegt.

ESC D Cursor links

Damit wird der Cursor um eine Position nach links bewegt.

ESC E Clear Home

Diese Steuersequenz löscht den kompletten Bildschirm und positioniert den Cursor in die linke obere Bildschirmecke (Home-Position).

ESC H Cursor Home

Damit können Sie den Cursor in die linke obere Bildschirmecke bringen, ohne jedoch den Bildschirminhalt zu löschen.

ESC I Cursor nach oben

Diese Sequenz bewegt den Cursor um eine Zeile nach oben. Im Gegensatz zu ESC A wird jedoch dann, wenn der Cursor sich schon in der obersten Zeile befand, eine Leerzeile eingefügt und der Rest des Bildschirms entsprechend um eine Zeile nach unten gescrollt. Die Spaltenposition des Cursors bleibt dabei unverändert.

ESC J Rest des Bildschirms löschen

Mittels dieser Funktion wird der Rest des Bildschirms ab der augenblicklichen Cursorposition einschließlich gelöscht. Der Cursor bleibt dabei an derselben Stelle.

ESC K Rest der Zeile löschen

Diese ESC-Sequenz löscht den Rest der Zeile, in der sich der Cursor befindet. Gelöscht wird ab Cursorposition einschließlich; die Cursorposition selbst wird nicht verändert.

ESC L Zeile einfügen

Damit ist es möglich, an der augenblicklichen Cursorposition eine Leerzeile einzufügen. Der Rest des Bildschirms wird nach unten geschoben; die unterste Zeile geht dabei verloren. Nach dem Einfügen steht der Cursor am Beginn der neuen Zeile.

ESC M Zeile löschen

Diese Funktion löscht die Zeile, in der sich der Cursor befindet, und rückt den Rest des Bildschirms nach oben. Die unterste Bildschirmzeile wird dadurch frei. Der Cursor befindet sich nach dem Löschen in der ersten Spalte der nachgerückten Zeile.

ESC Y Cursor positionieren

Die Funktion erlaubt es, den Cursor beliebig auf dem Bildschirm zu positionieren. Dazu braucht sie noch die Cursorzeile und -spalte als Parameter, die in dieser Reihenfolge mit einem Offset von 32 erwartet werden. Wollen Sie den

Cursor in Zeile 7, Spalte 40 setzen, so müssen Sie die Sequenz ESC Y CHR\$(32+7) CHR\$(32+ 40) ausgeben. Zeilen und Spalten werden dabei von Null an gezählt; bei einem 25 * 80 Bildschirm sind die Zeilen von 0 bis 24 und die Spalten von 0 bis 79 numeriert.

ESC b Schriftfarbe wählen

Mit dieser Funktion können Sie die Schriftfarbe für die weitere Ausgabe wählen. Bei einem Monochrommonitor haben Sie nur die Wahl zwischen 0 = weiß und 1 = schwarz. Bei Farbdarstellung können Sie je nach Modus aus vier oder 16 Farben auswählen. Bei dem Parameter werden nur die untersten vier Bits ausgewertet (mod 16). Sie können für die Farbe eins sowohl die Ziffer "1" als auch den Buchstaben "A" oder "a" verwenden.

ESC c Hintergrundfarbe wählen

Diese Funktion dient analog zur Wahl der Hintergrundfarbe. Wählen Sie für Schrift- und Hintergrund die gleiche Farbe, so können Sie die nachfolgende Textausgabe natürlich nicht mehr sehen.

ESC d Bildschirm bis Cursorposition löschen

Durch diese Sequenz wird der Bildschirm vom Anfang bis zur augenblicklichen Cursorposition einschließlich gelöscht. Die Position des Cursors wird dabei nicht verändert.

ESC e Cursor einschalten

Durch diese Escape-Sequenz wird der Cursor sichtbar, z.B. kann der Cursor eingeschaltet werden, wenn vom Benutzer eine Eingabe erwartet wird.

ESC f Cursor ausschalten

Hierdurch wird der Cursor wieder abgeschaltet.

ESC j Cursorposition speichern

Wollen Sie die augenblickliche Position des Cursors speichern, so bedienen Sie sich dazu dieser Sequenz. Leider wird diese Funktion intern auch von anderen ESC-Sequenzen benutzt, so daß danach die gespeicherte Position nicht mehr zur Verfügung steht.

ESC k Cursor auf gespeicherte Position setzen

Das Gegenstück zu der obigen Funktion ist diese. Damit wird der Cursor wieder auf die Position gesetzt, die vorher mit ESC j gespeichert wurde. War keine Cursorposition gespeichert, wird der Cursor in die Home-Position gebracht.

ESC l Zeile löschen

Damit wird die Zeile gelöscht, in der sich der Cursor befindet. Die übrigen Zeilen bleiben davon unbeeinträchtigt. Nach dem Löschen befindet sich der Cursor in der ersten Spalte der gelöschten Zeile.

ESC o Zeilenanfang löschen

Hiermit wird der Anfang der Cursorzeile bis zur Cursorposition einschließlich gelöscht. Die Position des Cursors bleibt unverändert.

ESC p Reverse ein

Durch diese Sequenz wird auf invertierte Ausgabe geschaltet. Dabei werden für die nachfolgende Ausgabe Schrift- und Hintergrundfarbe vertauscht. Bei einem Monochrommonitor erhalten Sie weiße Schrift auf schwarzem Grund.

ESC q Reverse aus

Zum Zurückschalten auf Normaldarstellung dient diese Sequenz.

ESC v Automatischer Überlauf ein

Nach Ausführen dieser Sequenz wird bei einer versuchten Ausgabe über das Zeilenende hinaus automatisch eine neue Zeile begonnen.

ESC w Automatischer Überlauf aus

Damit wird die obige Sequenz wieder aufgehoben. Beim Versuch, über das Zeilenende zu schreiben, werden alle nachfolgenden Zeichen in die letzte Spalte dieser Zeile geschrieben.

25. Die Systemvariablen des Atari ST

Der Atari ST benutzt eine Reihe von Systemvariablen, deren Bedeutung und Adressen sich auch in künftigen Versionen des Betriebssystems nicht ändern werden. Sollten Sie andere Variablen benutzen, die hier nicht aufgeführt sind, so sollten Sie sich immer im klaren darüber sein, daß diese bei einer neuen Version des Betriebssystems eine andere Bedeutung haben können. Die Systemvariablen sind im unteren RAM-Bereich direkt oberhalb der 68000-Exception-Vektoren untergebracht, also ab Adresse \$400 bzw. 1024. Auf den Adreßbereich von 0 bis \$7FF (2047) kann nur im Supervisormodus zugegriffen werden. Ein Zugriff im User-Modus des 68000 führt zu einem Bus-Error.

In der folgenden Auflistung wird jeweils der Originalname von Atari benutzt. Neben der Adresse der jeweiligen Variablen wird ein typischer Inhalt angegeben sowie die Bedeutung näher beschrieben. Sind beim Inhalt drei Werte angegeben, so bezieht sich der erste auf die ROM-Version des Betriebssystems vom 6.2.1986, der zweite gilt für die Disketten-TOS-Version vom gleichen Datum. Der dritte Wert schließlich ist für das deutsche Blitter-TOS vom 22.4.1987 (mit 4 MByte RAM) relevant.

Adresse	Länge	Name	Beispielinhalt
\$400	L	etv_timer	\$FCA62A/\$104B8/\$FCA7E4

Dies ist der Event-Timer-Vektor des GEM. Damit werden periodische Aufgaben des GEM erledigt.

\$404	L	etv_critic	\$FC0744/\$65D2/\$FC07EE
-------	---	------------	--------------------------

Critical Error Handler. Unter GEM zeigt dieser Vektor auf Adresse \$FE3226/\$294DE/\$FE3928. Dort wird der Versuch unternommen, Diskettenfehler zu korrigieren, z.B. wird bei Single-Floppy-Systemen aufgefordert, eine andere Diskette einzulegen.

\$408	L	etv_term	\$FC05C0/\$644E/\$FC0670
-------	---	----------	--------------------------

Dort steht die Adresse einer Routine, die vom GEMDOS vor Beendigung eines Programms aufgerufen wird.

\$40C 5L etv_xtra

Hier ist Platz für fünf weitere GEM-Vektoren, die z.Z. jedoch noch nicht genutzt werden.

\$420 L memvalid \$752019F3

Enthält die Speicherzelle den angegebenen Wert, so ist die Konfiguration des Memory-Controllers gültig.

\$424 B memctrl \$05

Hierbei handelt es sich um eine Kopie des Konfigurationswertes im Memory-Controller. Der angegebene Wert gilt für eine 1-MByte-Maschine.

\$426 L resvalid \$31415926

Steht hier der angegebene Wert, so wird bei einem Reset über den Resetvektor in Adresse \$42A gesprungen.

\$42A L resvector \$FC0008

Siehe oben.

\$42E L phystop \$100000/\$100000/\$400000
--

Dies ist das physikalische Ende des RAM-Speichers; \$80000 bei einer 512-KByte-Maschine, \$100000 bei einer 1-MByte-Maschine sowie \$400000 bei 4 MByte.

\$432 L _membot \$A100/\$39FF0/\$CA00

Hier beginnt der Benutzerspeicher (TPA, transient program area).

\$436 L _mentop \$F8000/\$F8000/\$3F8000
--

Dies ist das obere Ende des Benutzerspeichers.

\$43A	L	memval2	\$237698AA
-------	---	---------	------------

Dieser "magische" Wert erklärt zusammen mit "memvalid" die Speicherkonfiguration für gültig.

\$43E	W	flock	0
-------	---	-------	---

Enthält diese Variable einen Wert ungleich null, so findet gerade ein Floppy-Zugriff statt, und die VBL-Floppy-Routine ist gesperrt.

\$440	W	seekrate	3
-------	---	----------	---

Damit wird die Seek-Rate (Zeit, um den Schreib-/Lesekopf der Floppy zum nächsten Track zu bewegen) bestimmt nach folgender Tabelle:

Seek-Rate	Zeit
0	6 ms
1	12 ms
2	2 ms
3	3 ms

\$442	W	_timer_ms	\$14, 20 ms
-------	---	-----------	-------------

Der Zeitabstand zwischen zwei Timer-Aufrufen, 20 ms entspricht 50 Hz.

\$444	W	_fverify	\$FF
-------	---	----------	------

Enthält diese Speicherstelle einen Wert ungleich null, so wird nach jedem Floppy-Schreibzugriff über die BIOS-Funktion 4 ein Verify durchgeführt.

\$446	W	_bootdev	0
-------	---	----------	---

Enthält die Gerätenummer des Laufwerks, von dem das Betriebssystem geladen wurde.

\$448	W	palmode	0
-------	---	---------	---

Enthält diese Variable einen Wert ungleich null, so befindet sich das System im PAL-Modus (50 Hz); bei null bedeutet dies NTSC-Modus (60 Hz).

\$44A	W	defshiftmod	0
-------	---	-------------	---

Wenn der Atari von Monochrombetrieb auf Farbe umschaltet, so holt er sich von hier die neue Auflösung (0 = low resolution, 1 = medium resolution).

\$44C	B	sshiftmd	2
-------	---	----------	---

Hier steht eine Kopie des Registerinhalts für die Bildschirmauflösung.

0 - 320 * 200, low resolution

1 - 640 * 200, medium resolution

2 - 640 * 400, high resolution

\$44E	L	_v_bas_ad	\$F8000/\$F8000/\$3F8000
-------	---	-----------	--------------------------

Diese Variable enthält einen Zeiger auf das Video-RAM (logical screenbase). Die Bildschirmadresse muß immer an einer 256-Byte-Grenze beginnen.

\$452	W	vblsem	1
-------	---	--------	---

Ist diese Variable auf Null, so wird die Vertical-Blank-Routine nicht ausgeführt.

\$454	W	nvbls	8
-------	---	-------	---

Anzahl der Vertical-Blank-Routinen.

\$456	L	_vblqueue	\$4CE
-------	---	-----------	-------

Zeiger auf eine Liste von "nvbls"-Routinen, die während des VBL ausgeführt werden.

\$45A	L	colorptr	0
-------	---	----------	---

Ist dieser Wert ungleich null, so wird er als Zeiger auf eine Farbpalette interpretiert, die beim nächsten VBL geladen wird.

\$45E	L	screenpt	0
-------	---	----------	---

Dies ist ein Zeiger auf den Beginn des Video-RAMs, der beim nächsten VBL gesetzt wird. (Null, falls keine neue Adresse gesetzt werden soll.)

\$462	L	_vbclock	\$2D26A
-------	---	----------	---------

Zähler für die Anzahl der VBL-Interrupts

\$466	L	_frclock	\$2D267
-------	---	----------	---------

Anzahl der ausgeführten VBL-Routinen (nicht durch vblsem gesperrt).

\$46A	L	hdv_init	\$FC0D60/\$6BEE/\$FC0F44
-------	---	----------	--------------------------

Vektor zur Laufwerk-Initialisierung.

\$46E	L	swv_vec	\$FC0020/\$6120/\$FC0030
-------	---	---------	--------------------------

Vektor für Monitorwechsel. Über diesen Vektor wird gesprungen, wenn ein anderer Monitor (Monochrom/Farbe) angeschlossen wird (Default ist Reset).

\$472	L	hdv_bpb	\$FC0DE6/\$6C74/\$FC0FCA
-------	---	---------	--------------------------

Vektor zum Holen des BIOS-Parameter-Blocks (BIOS Funktion 7).

\$476	L	hdv_rw	\$FC10D2/\$6F60/\$FC12B6
-------	---	--------	--------------------------

Vektor für die Schreib-/Lese-Routine (BIOS-Funktion 4).

\$47A	L	hdv_boot	\$FC137C/\$720A/\$FC1560
-------	---	----------	--------------------------

Vektor zum Laden des Boot-Sektors.

\$47E	L	hdv_mediach	\$FC0F96/\$6E24/\$FC117A
-------	---	-------------	--------------------------

Vektor für Media-Change-Routine (BIOS Funktion 9).

\$482	W	_cmdload	0
-------	---	----------	---

Wenn diese Variable vom Boot-Programm ungleich null gesetzt wird, so wird nach dem Laden des Betriebssystems versucht, ein Programm namens "COMMAND.PRG" zu laden (z.B. eine Anwendung anstelle des Desktop).

\$484	B	conterm	6
-------	---	---------	---

Attribut-Vektor für Consol-Ausgabe.

Bit	Bedeutung
0	Tastatur-Klick ein/aus
1	Tastatur-Repeat ein/aus
2	Ton nach «Ctrl» «G» ein/aus
3	Bei der BIOS-Funktion "conin" wird in Bit 24-31 "kbshift" ftg zurückgeliefert.

\$48E	4L	themd	0
-------	----	-------	---

Memory-Descriptor, wird durch die BIOS-Funktion 0 "getmpb" ausgefüllt.

\$49E	2W	----_md	0
-------	----	---------	---

Platz für weitere Memory-Descriptoren.

\$4A2	L	savptr	\$90C
-------	---	--------	-------

Zeiger auf einen Save-Bereich für die Prozessor-Register nach einem BIOS-Aufruf.

\$4A6	W	_nfllops	2
-------	---	----------	---

Anzahl der angeschlossenen Floppies.

\$4A8	L	con_state	\$FC41BC/\$A04A/\$FC4362
-------	---	-----------	--------------------------

Vektor für Bildschirmausgabe; wird z.B. bei ESC-Funktionen auf entsprechende Routine gerichtet.

\$4AC	W	save_row	0
-------	---	----------	---

Zwischenspeicher für Cursor-Zeile beim Cursor-Positionieren mit ESC Y.

\$4AE	L	sav_context	0
-------	---	-------------	---

Zeiger auf einen Zwischenspeicher bei einer Exception-Behandlung.

\$4B2	2L	_buf1	\$60A4,\$60CC/\$60A4,\$60CC/\$8846,\$886E
-------	----	-------	---

Zeiger auf zwei Buffer-List-Header des GEMDOS. Der erste Header ist für Datensektoren zuständig, der zweite für FAT (file allocation table) und Directory. Jeder Buffer-Control-Block (BCB) ist folgendermaßen aufgebaut:

```

long BCB    $60B8, Zeiger auf nächsten BCB
int  drive  -1,   Drive-Nummer oder -1
int  typ     2,   Puffertyp
int  rec     $41C Record-Nummer in diesem Puffer
int  dirty   0,   dirty-Flag (Pufferinhalt verändert)
long DMD     $2B04 Zeiger auf Drive-Media-Descriptor
long buffer  $4E22 Zeiger auf den Puffer selbst
    
```

\$4BA	L	_hz_200	\$71280
-------	---	---------	---------

Zähler für 200-Hz-Systemtakt.

\$4BE	4B	the_env	0
-------	----	---------	---

Default Environment String, vier Null-Bytes.

\$4C2	L	_drvbits	3
-------	---	----------	---

32-Bit-Vektor für angeschlossene Drives. Bit 0 steht für Drive A, Bit 1 für Drive B usw.

\$4C6	L	_dskbufp	\$167A/\$167A/\$16DA
-------	---	----------	----------------------

Zeiger auf einen 1024-Byte-Disk-Puffer. Der Puffer wird auch für Grafik-Operationen benutzt und sollte nicht von Interrupt-Routinen benutzt werden.

\$4CA	L	_autopath	0
\$4CE	8L	_vbl_list	\$FD03C4,0,0.../\$16252,0,0.../\$FCFF84,0,0...

Liste der 8 Standard-VBL-Routinen.

\$4EE	W	_dumpflg	\$FFFF
-------	---	----------	--------

Dieses Flag wird durch gleichzeitiges Drücken von und um eins erhöht. Ein Wert von Null löst eine Bildschirm-Hardcopy auf dem Drucker aus. Durch nochmaliges Drücken von wird eine Hardcopy wieder abgebrochen.

\$4F2	L	_sysbase	\$FC0000/\$6100/FC0000
-------	---	----------	------------------------

Zeiger auf Beginn des Betriebssystems. In dem dort befindlichen Header stehen folgende weitere Systemvariablen zur Verfügung:

Offset	Variable	Bedeutung
2 W	Version	Version des Betriebssystems, z.B. \$102entspricht Version 1.02
4 L	reset	Reset-Adresse, z.B. \$FC0030
8 L	os_start	Zeiger auf Beginn des Betriebssystems
12 L	os_end	Zeiger auf Beginn des freien Rams
16 L	shell	Zeiger auf Default-Shell (Reset)
20 L	gem_magic	Adresse GEM-Magic. Ist der Inhalt der Long-Variablen, auf die diese Adresse zeigt, gleich \$87654321, so wird beim Booten das GEM-Desktop gestartet.
24 L	date	Erstellungsdatum des TOS im Format Monat/Tag/Jahr
28 W	pal_flag	Flag für PAL-Version
30 W	sys_date	Erstellungsdatum im GEMDOS-Format. Die folgenden Systemvariablen im Header findet man erst im TOS ab 1987
32 L	_root	Zeiger auf den internen GEMDOS-Memory-Pool. Mit Hilfe dieser Variablen wird mit dem FOLDR100.PRG der Memory-Pool erweitert.
		...

...		
36 L	kbshift	Adresse der Variablen, die den Shift-Status der Tastatur enthält (Siehe BIOS-Funktion 11, Kbshift).
40 L	_run	Zeiger auf die Adresse der Basepage des gerade laufenden Programms. Diese Variablen können nur gelesen werden (auch wenn sich das TOS im RAM befindet (Disketten-TOS)).

\$4F6 L	_shell_p;	0
\$4FA L	end_os	\$A100/\$3A4A0/\$CA00

Zeiger auf Ende des Betriebssystems im RAM, Beginn der TPA.

\$4FE L	exec_os	\$FD8E98/\$1F600/\$FD9B5C
---------	---------	---------------------------

Zeiger auf Beginn des AES. Wird normalerweise nach der Initialisierung des BIOS angesprungen.

\$502 L	dump_vec	\$FC0C2C/\$6ABA/FC0D62
---------	----------	------------------------

Über diesen Vektor wird gesprungen, wenn eine Hardcopy gedruckt wird (XBIOS-Funktion 20).

\$506 L	prt_stat	\$FC1F34/\$7DC2/\$FC2124
---------	----------	--------------------------

Vektor zum Holen des Druckerstatus bei Hardcopy-Funktion.

\$50A L	prt_vec	\$FC1EA0/\$7D2E/\$FC2090
---------	---------	--------------------------

Vektor zur Druckerausgabe bei Hardcopy-Funktion.

\$50E L	aux_stat	\$FC1F6E/\$7DFC/\$FC219A
---------	----------	--------------------------

Vektor zum Holen des V24-Ausgabestatus bei Hardcopy-Funktion.

\$512 L	aux_vec	\$FC1F86/\$7E14/\$FC21B4
---------	---------	--------------------------

Vektor für V24-Ausgabe bei Hardcopy-Funktion.

```
$516 L   pun_ptr           $D3A2
```

Ist eine Harddisk angeschlossen und das Treiberprogramm AHDI geladen, so zeigt dieser Vektor auf eine Struktur, die folgendermaßen aufgebaut ist:

```
int   puns           1
```

Anzahl der angeschlossenen Festplatten (physical units).

```
char v_p_un[16]   -1, -1, 0, 0, 0, -1, -1, ...
```

Physikalische Units der logischen Drives A bis P. Im Beispiel sind die Drives C bis E der Festplatte mit der Gerätenummer Null zugeordnet (-1 bedeutet nicht zugeordnet).

```
long pstart[16]   0, 0, 1, $3A91, $6291, 0, 0, ..
```

Startsektoren der einzelnen Partitions.

```
BPB   bpbs[16]
```

Hier stehen die Bios Parameter Blocks (BPBs) der angeschlossenen Partitions.

Die folgenden Systemvariablen finden sich erst bei Betriebssystemen, die ein Erstellungsdatum ab 1987 haben.

```
$51A L   memval3           $5555AAAA
```

Enthält die Variable den obigen "magischen" Wert, so gilt die Speicherkonfiguration nach einem Reset (zusammen mit memvalid und memval2) als gültig.

```
$51E 8L   bconstat_vec     $FC0670, $FC2138, $FC2226, $FC2044
                                $FC0670, $FC0670, $FC0670, $FC0670
```

Hier stehen acht Zeiger auf Routinen zum Holen des Eingabestatus (BIOS-Funktion 1, Bconstat). Der erste Wert gilt für Gerätenummer 0, der nächste für Gerät 1 usw. bis Gerät 7. Die Adresse \$FC0670 zeigt direkt auf einen rts-Befehl.

```
$53E 8L   bconin_vec       $FC2104, $FC2150, $FC223C, $FC2060
                                $FC0670, $FC0670, $FC0670, $FC0670
```

Diese Vektortabelle hat eine analoge Funktion wie die davor stehende. Hierbei handelt es sich jedoch um die Adressen für die BIOS-Funktion 2, Bconin.

\$55E 8L	bccstat_vec	\$FC2124, \$FC219A, \$FC226C, \$FC21DC \$FC2004, \$FC0670, \$FC0670, \$FC0670
----------	-------------	--

Hier stehen die Adressen zum Holen des Ausgabestatus' für die Gerätenummern Null bis sieben. Darüber wird beim Aufruf der BIOS-Funktion 8, Bccstat, gesprungen.

\$57E 8L	bconout_vec	\$FC2090, \$FC21B4, \$FC434C, \$FC2016 \$FC21EE, \$FC4340, \$FC0670, \$FC0670
----------	-------------	--

Als letztes in dieser Reihe sind hier die Adressen zur Zeichenausgabe abgelegt. Dies entspricht der BIOS-Funktion 3, Bconout.

26. Die LINE-A-Funktionen

Die LINE-A-Funktionen nutzen einen Teil der Übergabe-Arrays, die sonst für die GEM-Funktionen benutzt werden. Das Contrl-Array enthält für GEM-Funktionen im ersten Element (Contrl(0)) immer die Nummer der gewünschten Funktion. Dieser Parameter wird jedoch von den Line As nicht benutzt. Contrl(1) enthält bei jedem Aufruf des GEM die Anzahl der für die Funktion erforderlichen XY-Koordinaten. Diese Koordinaten müssen vor dem Aufruf im Ptsin-Array abgelegt werden. Das Element Contrl(2) wird vor dem Aufruf nicht versorgt. Es enthält nach dem Aufruf die Anzahl der XY-Koordinaten im Ptsout-Array. Contrl(3) gibt vor dem Aufruf einer GEM-Funktion an, wie viele Parameter der Funktion im Intin-Array übergeben werden, während Contrl(4) die Anzahl der Parameter im Intout-Array nach dem Aufruf bereithält. Die weiteren Parameter des Contrl-Arrays sind für den Anwender der Line A nicht relevant. Dazu kommen noch die LINE-A-Variablen (siehe unten).

Übrigens müssen Sie damit rechnen, daß die Register D0 bis D2 und A0 bis A2 von den Aufrufen der Funktionen geändert werden. Teilweise erhalten Sie Ergebnisse in den Registern zurück. Wichtige Werte in den genannten Registern sollten also besser vor einem Aufruf gerettet werden.

\$A000 Initialisierung

Initialisierung ist für diese Funktion eigentlich der falsche Ausdruck. Nach dem Aufruf erhält der Programmierer in den Registern D0 und A0 bis A2 die Adressen wichtiger Datenbereiche Eingabe-Parameter benötigt diese Funktion nicht.

In D0 und A0 wird dem Programm die Startadresse der Line-A-Variablen mitgeteilt. A1 zeigt nach dem Aufruf auf eine Tabelle mit drei Adressen. Diese drei Adressen sind die Start-Adressen der drei System-Font-Header. Das Register A2 schließlich zeigt auf eine Tabelle mit den Start-Adressen der 15 Line-A-Routinen.

\$A001 PUT PIXEL

Dieser Opcode setzt einen Punkt auf den durch die Koordinaten in Ptsin(0) und Ptsin(1) angegebenen Punkt. Die Farbe wird in Intin(0) übergeben. Ptsin(0) enthält die X-Koordinate, Ptsin(1) die Y-Koordinate.

Das verwendete Koordinatensystem hat seinen Ursprung in der linken oberen Ecke. Der mögliche Bereich der X- und Y-Koordinaten richtet sich natürlich nach den Möglichkeiten des eingestellten Modus. Allerdings werden Über-

schreitungen im X-Bereich nicht als Fehler behandelt. Statt dessen wird einfach bei einer Überschreitung die Y-Koordinate um den entsprechenden Betrag erhöht. Bei Überschreitungen des Y-Bereichs erfolgt keine Ausgabe.

Die Farbe in Intin(0) ist vom verwendeten Modus abhängig. Beim Betrieb des Monochrom-Monitors wird nur das Bit Null des Werts in Intin(0) ausgewertet.

\$A002 GET PIXEL

Mit diesem Opcode kann die Farbe eines Pixels bestimmt werden. Wie bei \$A001 wird die XY-Koordinate in Ptsin(0) und Ptsin(1) angegeben, den Farbwert des Pixels erhält man im Register D0 zurück.

\$A003 LINE

Mit dem Line-Opcode kann eine Linie zwischen den Punkten mit den Koordinaten x1,y1 und x2,y2 gezogen werden. Die Parameter werden bei dieser Funktion nicht über die Parameter-Arrays übergeben, sondern müssen vor dem Aufruf in die Line-A-Variablen übertragen werden. Die benutzen Variablen sind:

<u>_X1</u>	x1-Koordinate
<u>_Y1</u>	y1-Koordinate
<u>_X2</u>	x2-Koordinate
<u>_Y2</u>	y2-Koordinate
<u>_FG_BP_1</u>	Plane 1 (alle drei Modi)
<u>_FG_BP_2</u>	Plane 2 (640x200, 320x200)
<u>_FG_BP_3</u>	Plane 3 (nur 320x200)
<u>_FG_BP_4</u>	Plane 4 (nur 320x200)
<u>_LN_MASK</u>	Bit-Muster der Linie
	z.B. \$FFFF = durchgezogen
	\$CCCC = durchbrochen
<u>_WRT_MOD</u>	bestimmt den Schreibmodus
<u>_LSTLIN</u>	diese Variable sollte auf -1 (\$FFFF) gesetzt werden.

Ein Punkt, den es bei manchen Anwendungen zu beachten gilt, ist die Tatsache, daß beim Ziehen der Linie das höchste Bit des Linien-Bit-Musters immer an den linken Bildschirmrand gesetzt wird. Die Linie wird also immer von links nach rechts und von oben nach unten, nicht jedoch von x1,y1 nach x2,y2 gezogen.

Bereichsüberschreitungen werden wie bei PUT PIXEL behandelt. Soll also eine Linie von 0,0 nach 650,50 gezogen werden, so wird tatsächlich eine Linie von 0,0 bis 639,48 gezogen. Der "Rest" ergibt eine zusätzliche Linie von 0,49 bis 10,50.

Für das Linienziehen stehen insgesamt vier verschiedene Schreibmodi mit den Werten 0 bis 3 zur Verfügung. Beim Schreibmodus Null wird das ursprünglich vorhandene Bit-Muster "unter" der Linie gelöscht und das durch `_LN_MASK` bestimmte Bit-Muster eingetragen (Replace Mode). Im Transparent-Mode (`_WRT_MOD = 1`) werden Hintergrund, also altes Bit-Muster, und neues Linienmuster verODERt, also nur zusätzlich Punkte gesetzt. Im XOR-Mode (`_WRT_MOD = 2`) werden Hintergrund und Linienmuster Exklusiv-verODERt. Der letzte Modus (`_WRT_MOD = 3`) schließlich ist der sogenannte "Inverse Transparent-Mode". Wie auch beim Transparent Mode handelt es sich um eine ODER-Verknüpfung von Vordergrund und Hintergrunddaten, wobei jedoch die Vordergrunddaten, also das durch `_LN_MASK` festgelegte Bit-Muster, vor der VerODERung invertiert wird.

\$A004 HORIZONTAL LINE

Diese Funktion zieht eine Linie von `x1,y1` nach `x2,y1`. Das Ziehen einer horizontalen Linie ist bedeutend schneller als wenn eine Linie diagonal gezeichnet werden muß. Im Grunde werden auch diagonale Linien über diese Funktion erzeugt, indem die Linie in mehrere horizontale Einzelstücke zerlegt wird. Die Parameter werden direkt in die benötigten Variablen eingetragen.

<code>_X1</code>	x1-Koordinate
<code>_Y1</code>	y1-Koordinate
<code>_X2</code>	x2-Koordinate
<code>_FG_BP_1</code>	Plane 1 (alle drei Modi)
<code>_FG_BP_2</code>	Plane 2 (640x200, 320x200)
<code>_FG_BP_3</code>	Plane 3 (nur 320x200)
<code>_FG_BP_4</code>	Plane 4 (nur 320x200)
<code>_WRT_MOD</code>	bestimmt den Schreibmodus.
<code>_patptr</code>	Pointer auf das zu verwendende Linienmuster
<code>_patmsk</code>	"Maske" für das Linienmuster

Die in `_WRT_MOD` zulässigen Werte liegen auch bei diesem Aufruf zwischen 0 und 3. Der Inhalt der Variablen `_patptr` ist die Adresse, an der das gewünschte Linienmuster oder besser Füllmuster zu finden ist. Die H-Line-Funktion eignet sich nämlich hervorragend zum Erzeugen beliebig gefüllter Flächen. Eine wesentliche Rolle spielt dabei die Variable `_patmsk`. Von ihrem Wert ist die Anzahl der 16-Bit-Werte an der Adresse in `_patptr` abhängig. Ent-

hält z.B. `_patmsk` den Wert 5, so sollten ab der Adresse in `_patptr` sechs 16-Bit-Werte als Linienmuster im Speicher stehen. Soll nun eine horizontale Linie mit der Y-Koordinate Null gezogen werden, so wird das erste Bit-Muster als Linienmuster genommen. Für eine Linie mit der Y-Koordinate 1 wird das zweite Wort als Muster genommen usw. Das Muster für eine Linie mit der Y-Koordinate 6 wird wieder vom ersten Wert in der Bit-Tabelle bestimmt. Auf diese Weise können sehr komplexe Füllmuster ohne großen Aufwand erzeugt werden.

\$A005 FILLED RECTANGLE

Der Opcode \$A005 stellt eine Erweiterung oder besser spezielle Nutzung des Opcodes \$A004 dar. Mit dieser Funktion können beliebig gefüllte Rechtecke erzeugt werden. Als wesentliche Parameter werden die linke obere und die rechte untere Ecke als Koordinaten angegeben.

<code>_X1</code>	x1-Koordinate, links oben
<code>_Y1</code>	y1-Koordinate
<code>_X2</code>	x2-Koordinate, rechts unten
<code>_Y2</code>	y2-Koordinate
<code>_FG_BP_1</code>	Plane 1 (alle drei Modi)
<code>_FG_BP_2</code>	Plane 2 (640x200, 320x200)
<code>_FG_BP_3</code>	Plane 3 (nur 320x200)
<code>_FG_BP_4</code>	Plane 4 (nur 320x200)
<code>_WRT_MOD</code>	bestimmt den Schreibmodus.
<code>_patptr</code>	Pointer auf das zu verwendende Füllmuster
<code>_patmsk</code>	"Maske" für das Füllmuster
<code>_CLIP</code>	Clipping Flag
<code>_XMN_CLIP</code>	x Minimum für Clipping
<code>_XMX_CLIP</code>	x Maximum für Clipping
<code>_YMN_CLIP</code>	y Minimum für Clipping
<code>_YMX_CLIP</code>	y Maximum für Clipping

Bis auf die "Clip"-Variablen sind die verwendeten Variablen bereits bei den vorigen Aufrufen erläutert. Das Clipping erzeugt Ausschnitte aus dem gesamten Bild. Wird das Clipping-Flag auf Eins (oder jeden Wert ungleich Null) gesetzt, dann wird das Rechteck, das mit \$A005 gezeichnet wird, nur in dem durch die Clip-Bereichsvariablen angegebenen Ausschnitt dargestellt.

\$A006 FILLED POLYGON

Auch \$A006 ist eine Erweiterung des Aufrufs \$A004. Mit diesem Opcode können beliebig geformte Flächen mit einem Muster gefüllt werden. Allerdings

wird mit dem Aufruf nicht die gesamte Fläche gefüllt, sondern nur eine Rasterzeile, also eine einen Punkt breite horizontale Linie, gezeichnet. Dadurch ergeben sich natürlich wesentlich größere Beeinflussungsmöglichkeiten auf das Füllmuster. Die benötigten Variablen sind:

PTSIN	Array mit den XY-Koordinaten
CONTRL(1)	Anzahl der Koordinatenpaare
_Y1	Y-Koordinate der zu füllenden Zeile
_FG_BP_1	Bit-Wert für Plane 1
_FG_BP_2	Bit-Wert für Plane 2
_FG_BP_3	Bit-Wert für Plane 3
_FG_BP_4	Bit-Wert für Plane 4
_WRT_MOD	Schreibmodus (0-3)
_patptr	Pointer auf die Füllmuster
_patmsk	Anzahl der Füllmuster-Worte
_CLIP	Clipping-Flag
_XMN_CLIP	x Minimum für Clipping
_XMX_CLIP	x Maximum für Clipping
_YMN_CLIP	y Minimum für Clipping
_YMX_CLIP	y Maximum für Clipping

Im Grunde haben Sie hier alle Parameter genau so zu setzen, wie dies auch für einen Aufruf von \$A005 erforderlich ist. Nur die drei ersten Koordinaten sind unterschiedlich. Im PTSIN-Array werden die XY-Koordinaten gespeichert. Wichtig ist, daß Sie die Startkoordinaten auch noch einmal als letzte Koordinaten angeben. Um also ein Dreieck zu definieren, müssen Sie z.B. die Koordinaten (320,100), (120,300), (520,300) und (320,100) eingeben. Die Anzahl der effektiven Koordinatenpaare, in unserem Beispiel also drei, müssen Sie in CONTRL (1), dem zweiten Element des Arrays hinterlegen. Mit einem Aufruf der \$A006-Funktion müssen Sie noch die Y-Koordinate der zu ziehenden Linie angeben. Natürlich können Sie nacheinander alle Y-Koordinaten von 0 bis 399 (0 bis 199 in den Farbmodi) füllen lassen. Schneller ist es jedoch, sich aus den XY-Werten den kleinsten und größten verwendeten Y-Wert zu suchen und nur für diesen Bereich die Funktion aufzurufen.

\$A007 BITBLT

Die Funktion BITBLT kann einen rechteckigen Quellbereich in einen Zielbereich kopieren. Dabei kann der Quellbereich noch mit einem Raster verknüpft werden. Quell- und Zielbereich können dabei durch 16 verschiedene logische Operationen verknüpft werden. Sie können an beliebigen Adressen stehen. Im allgemeinen ist mindestens der Zielbereich Teil des Video-RAMs; es kann jedoch auch innerhalb des Bildschirms oder vom einem nicht angezeigten

Speicherbereich in einen anderen kopiert werden. Die Funktion wird von anderen LINE-A-Funktionen benutzt; das sind TEXTBLT und COPY RASTER FORM sowie die darauf basierenden VDI-Funktionen Copy Raster Opaque (vro_cpyfm) und Copy Raster Transparent (vrt_cpyfm). Ist in den Atari ST ein Blitter eingebaut, so erledigt er den BITBLT-Aufruf per Hardware. Entsprechend der Vielseitigkeit der Funktion müssen eine ganze Reihe von Parametern vor Aufruf der Funktion versorgt werden. Dazu gehören Angaben über den Quellbereich (Source), den Zielbereich (Destination) sowie ein evtl. verwendetes Raster (Pattern). Weiterhin werden allgemeine Informationen über die Anzahl der Bildebenen (Farbe/Schwarzweiß) sowie die logische Operation zur Verknüpfung von Ziel und Quelle benötigt. Die Daten stehen in einem 76 Byte großem Parameterblock, dessen Adresse der Funktion im Register A6 übergeben werden muß. Der Parameterblock sieht so aus:

Offset	Länge	Name
0 W	s_width	Breite des zu verarbeitenden Bereichs in Pixel
2 W	s_height	Höhe des zu verarbeitenden Bereichs in Pixel
4 W	planes	Anzahl der Bildebenen
6 W	fg_col	Vordergrund-Farbe
8 W	bg_col	Hintergrund-Farbe
10 L	op_tab	Logische Operation
14 W	s_xmin	X-Koordinate linke obere Ecke in Source
16 W	s_ymin	Y-Koordinate linke obere Ecke in Source
18 L	s_form	Startadresse der Source
22 W	s_nxwd	Offset auf nächstes Word in der Source in Byte
24 W	s_nxln	Offset auf nächste Zeile in der Source in Byte
26 W	s_nxpl	Offset auf nächste Farb-Ebene in der Source

Offset	Länge	Name
28 W	d_xmin	X-Koordinate linke obere Ecke in Destination
30 W	d_ymin	Y-Koordinate linke obere Ecke in Destination
32 L	d_form	Startadresse der Destination
36 W	d_nxwd	Offset auf nächstes Word in der Destination in Byte
38 W	d_nxln	Offset auf nächste Zeile in der Destination in Byte
40 W	d_nxpl	Offset auf nächste Farb-Ebene in der Destination
42 L	p_addr	Startadresse des Rasters (Pattern)
46 W	p_nxln	Offset auf nächste Zeile im Raster in Byte
48 W	p_nxpl	Offset auf nächste Farb-Ebene im Raster
50 W	p_mask	Raster-Index-Maske (Rasterhöhe in Pixel)
52 12W	filler	Wird von BITBLT intern benutzt

Wenn Ziel- und/oder Quellbereich der Bildschirm sind, so sind folgende Werte zu benutzen:

Auflösung	320 * 200	640 * 200	640 * 400
Farbebenen	4	2	1
d_form/s_form	Bildschirmadresse		
d_nxwd/s_nxwd	8	4	2
d_nxln/s_nxln	160	160	80
d_nxpl/s_nxpl	2	2	2

Schauen wir uns jetzt die 16 verschiedenen logischen Operationen an, die Ziel und Quelle folgendermaßen verknüpfen:

Operation	Funktion	
0	D' = 0	Ziel wird auf Hintergrundfarbe gesetzt
1	D' = S & D	
2	D' = S & ~D	
3	D' = S	Replace Mode
4	D' = ~S & D	Erase Mode
5	D' = D	
6	D' = S ^ D	Xor Mode
7	D' = S D	
8	D' = ~ (S D)	
9	D' = ~ (S ^ D)	
10	D' = ~D	
11	D' = S ~D	
12	D' = ~S	
13	D' = ~S D	
14	D' = ~ (S & D)	
15	D' = 1	Ziel wird auf Vordergrundfarbe gesetzt

Dabei bedeuten S Quellbereich (Source), D Destination-Bereich (Ziel) vor der Operation und D' Destination-Bereich nach der Operation. Bei den logischen Operationen bedeuten "&" UND-Verknüpfung, "|" ODER-Verknüpfung und "^" XOR-Verknüpfung (Exklusiv Oder). Der Operator "~" bedeutet Invertierung.

Für die BITBLT-Funktion sind vier solcher logischer Operationen anzugeben, die folgendermaßen adressiert werden:

$$op = 2 * fg + bg$$

Dabei ist op die benutzte logische Operation (0 - 3 relativ zu op_tab). fg ist die Vordergrundfarbe, und bg ist die Hintergrundfarbe.

\$A008 TEXTBLT

Mit der Text-Block-Transfer-Funktion kann ein beliebiges Zeichen aus einem beliebigen Textfont an jeder Grafik-Position ausgegeben werden. Zusätzlich kann das Zeichen in seiner Form verändert werden. So kann das Zeichen z.B. kursiv, fett, "outlined", vergrößert oder gedreht dargestellt werden. Diese Dinge lassen sich mit den "normalen" Zeichenausgaben über das BIOS oder GEMDOS nicht erzielen. TEXTBLT stellt vielmehr das Grundgerüst aller Textausgaben unter VDI (v_gtext etc.) dar.

Für die richtige Nutzung dieser Funktion muß eine Vielzahl von Parametern gesetzt und kontrolliert werden. Es muß schon ein etwas umfangreicheres Programm geschrieben werden, um Texte mit dieser Funktion auszugeben. Wenn die zusätzlichen Möglichkeiten nicht unbedingt erforderlich sind, kann eigentlich nur von der Benutzung dieser Funktion abgeraten und statt dessen auf die in diesem Fall sicher einfacher zu handhabende v_gtext verwiesen werden. Aber entscheiden Sie doch bitte selbst.

Bevor wir jedoch konkret werden und ein Zeichen auf den Bildschirm zaubern, müssen wir uns zunächst ein wenig mit der Organisation der Fonts beschäftigen. Besonders der Fonthead muß unter die Lupe genommen werden, da mit den darin enthaltenen Daten der Font in allen Einzelheiten beschrieben wird.

Grundsätzlich besteht ein Font aus den vier Datenbereichen Fonthead, Font-Daten, Character-Offset-Tabelle und Horizontal-Offset-Tabelle. Im Fonthead sind allgemeine Daten über den Font wie z.B. der Name des Fonts, seine Größe, die Anzahl der enthaltenen Zeichen und verschiedene andere Aspekte enthalten. Diese Information belegt insgesamt 88 Bytes. Die Font-Daten enthalten die Bit-Muster der vorhandenen darstellbaren Zeichen. Diese Daten sind äußerst platzsparend organisiert.

Um die Organisation besser beschreiben zu können, denken wir uns einfach einen Font mit nur zwei Zeichen, z.B. einem "A" und einem "B". Diese Zeichen sollen in einer 9x9-Matrix dargestellt werden. Die Font-Daten stehen jetzt so im Speicher, daß das Bit-Muster der obersten Scanline des "A" an einer Wort-Grenze beginnend gespeichert ist.

Da unser Font 9 Pixel = 9 Bit breit ist, wird ein Byte vollständig, das folgende Byte aber nur mit dem obersten Bit benutzt. Jetzt müßten eigentlich sieben Bits verschenkt werden, wenn die oberste Scanline des "B" auch wieder auf einer

Wort-Grenze beginnen soll. Dem ist aber nicht so, statt dessen beginnt die erste Scanline des "B" im Bit 6 des zweiten Bytes der Font-Daten. Nur die Daten der zweiten und weiteren Scanlines des "A" beginnen immer auf einer Wort-Grenze. Auf diese Weise wird im Font fast kein Bit verschenkt. Nur die Anfänge der Scanlines des ersten Zeichens beginnen tatsächlich auf einer Wort-Grenze, alle anderen Scanlines können auf jeder Bit-Position beginnen.

Durch diese platzsparende Speicherung muß die Position jedes Zeichens innerhalb des Fonts berechnet werden. Die Berechnung der Scanline-Positionen wird möglich durch die Character-Offset-Tabelle. In dieser Tabelle ist für jedes darstellbare Zeichen ein Eintrag enthalten, der die Bit-Position der ersten Scanline des Zeichens relativ zum Anfang der Font-Daten enthält. Für unser Beispiel enthält eine solche Tabelle die Einträge \$0000, \$0009 und \$0012. Durch die Einführung dieser Tabelle ist es möglich, echte Proportionalschrift auf dem Bildschirm zu erzeugen, da die Breite des auszugebenden Zeichens errechnet werden kann. Man subtrahiert dazu den Eintrag des auszugebenden Zeichens vom Eintrag des nächsten Zeichens. Um auch die Breite des letzten Zeichens bestimmen zu können, ist der letzte Eintrag vorhanden, dem aber kein Zeichen mehr zugeordnet ist.

Neben der Character-Offset-Tabelle gibt es noch die Horizontal-Offset-Tabelle. Diese Tabelle wird jedoch von den meisten Fonts nicht genutzt. Auch die im ST vorhandenen Fonts nutzen die Möglichkeiten dieser Tabelle nicht. Wäre die Tabelle vorhanden, so enthielte sie für jedes Zeichen einen positiven oder negativen Offset-Wert, um den das Zeichen bei der Ausgabe nach links oder rechts verschoben würde.

Zum Ende der Beschreibung des Font-Aufbaus folgt die Bedeutungen der Variablen im Fonthead.

Byte 0-1	Font Identifier. Eine Nummer, die den Font beschreibt. 1 = System-Font.
Byte 2-3	Font-Größe in Punkt (Punkt ist ein Maß aus der Satztechnik).
Byte 4-35	Der Name des Fonts als ASCII-Zeichenkette.
Byte 36-37	Der niedrigste ASCII-Wert der darstellbaren Zeichen.
Byte 38-39	Der höchste ASCII-Wert der darstellbaren Zeichen.
Byte 40-49	Relative Abstände von Top, Ascent, Half, Decent und Bottom Line zur Base Line.

- Byte 50-51 Breite des breitesten Zeichens im Font.
- Byte 52-53 Breite der breitesten Character-Zelle. Die Zelle ist immer mindestens ein Pixel breiter als das eigentliche Zeichen, damit zwei nebeneinander stehende Zeichen voneinander getrennt sind.
- Byte 54-55 Linker Offset.
- Byte 56-57 Rechter Offset. Die beiden Offset-Werte werden nur beim Kursiv-Stellen von Zeichen (Skewing) verwendet.
- Byte 58-59 Thickening. Soll ein Zeichen fatter dargestellt werden, so sollte der Wert in dieser Variablen benutzt werden.
- Byte 60-61 Underline. Enthält die Höhe des Unterstreichungsstrichs in Pixeln.
- Byte 62-63 Lightening Mask. "Light" dargestellte Zeichen finden Sie im Desktop, wenn in einem Pull-Down-Menü eine Option nicht verfügbar ist. Diese hellgrau erscheinenden Zeichen entstehen durch Ausmaskieren von Bits mit der Lightening Mask. Üblich ist der Wert \$5555.
- Byte 64-65 Skewing Mask. Wie zuvor, nur für die Kursiv-Stellung von Zeichen.
- Byte 66-67 Flag. Bit 0 wird gesetzt, wenn der Font der System-Font ist.
Bit 1 muß gesetzt werden, wenn die Horizontal-Offset-Tabelle vorhanden ist.
Bit 2 ist das sogenannte Byte-Swap-Flag. Ist es gesetzt, so stehen die Bytes im 68000-Format (Highbyte-Lowbyte) im Speicher. Ein gelöschtes Byte-Swap-Flag signalisiert, das die Daten im INTEL-Format, also umgekehrt, im Speicher stehen. Mit diesem Bit können Fonts vom IBM mit GEM auch auf dem ST und umgekehrt verwendet werden.
Bit 3 ist gesetzt, wenn die Breite aller Zeichen im Font gleich ist.
- Byte 68-71 Pointer auf die Horizontal-Offset-Tabelle oder Null.
- Byte 72-75 Pointer auf die Character-Offset-Tabelle.
- Byte 76-79 Pointer auf die Font-Daten.

- Byte 80-81 Form Width. In dieser Variablen ist die Summe der Breiten aller Zeichen enthalten. Der Wert gibt somit die Länge der Scanlines aller Zeichen und damit den Anfang der nächsten Line an.
- Byte 82-83 Form Height. In dieser Variable ist die Anzahl der Scanlines dieses Fonts enthalten.
- Byte 84-87 Enthält einen Pointer auf den nächsten Font.

Nach so viel Vorrede sollen jetzt die Parameter genannt werden, die beim \$A008-Opcode zu beachten bzw. zu versorgen sind.

<code>_WRT_MODE</code>	Schreibmodus
<code>_TEXT_FG</code>	Text Vordergrund-Farbe
<code>_TEXT_BG</code>	Text Hintergrund-Farbe
<code>_FBASE</code>	Pointer auf den Anfang der Font-Daten
<code>_FWIDTH</code>	Breite des Fonts
<code>_SOURCEX</code>	X-Koordinate des Zeichens im Font
<code>_SOURCEY</code>	Y-Koordinate des Zeichens im Font
<code>_DESTX</code>	X-Koordinate des Zeichens auf dem Bildschirm
<code>_DESTY</code>	Y-Koordinate des Zeichens auf dem Bildschirm
<code>_DELX</code>	Breite des Zeichens in Pixeln
<code>_DELY</code>	Höhe des Zeichens in Pixeln
<code>_STYLE</code>	Bitweise codierte Flags für spez. Effekte
<code>_LITEMASK</code>	Bit-Muster wird beim "Lightning" verwendet
<code>_SKEWMASK</code>	Bit-Muster wird beim "Skewing" verwendet
<code>_WEIGHT</code>	Faktor bei Zeichenverbreiterung
<code>_R_OFF</code>	Rechter Offset des Zeichens beim "Skewing"
<code>_L_OFF</code>	Linker Offset des Zeichens beim "Skewing"
<code>_SCALE</code>	Flag für Scaling (Vergrößerung/Verkleinerung)
<code>_XACC_DDA</code>	Akkumulator für Scaling
<code>_DDA_INC</code>	Faktor der Vergrößerung oder Verkleinerung
<code>_T_SCLSTS</code>	"Scaling"-Richtungs-Flag
<code>_CHUP</code>	Zeichenrotationsvektor
<code>_MONO_STATUS</code>	Flag fixe Schrift (Monospaced)
<code>_scrtp</code>	Pointer auf Arbeits-Buffer für Effekte
<code>_scrpt2</code>	Offset Scaling-Buffer in <code>_scrtp</code>

Weiterhin werden die fünf Clip-Variablen ausgewertet.

Wie man sieht, sind es eine enorme Vielzahl von Variablen, die bei der Ausgabe von Grafiktext ausgewertet werden. Hier können wir nur auf die wesentlichen (und von uns erforschten) Variablen eingehen. Die nicht besprochenen Variablen mögen Ihnen als fruchtbares Experimentierfeld dienen.

Der Schreibmodus erlaubt die Ausgabe von Zeichen in den bisher bekannten vier Modi Replace, Or, Xor und Inverse Or. Im Zusammenhang mit den vier Schreibarten wird die Variable `_TEXT_FG` benutzt. Sie bildet die zum Darstellen verwendete Vordergrundfarbe. Die Hintergrundfarbe `_TEXT_BG` spielt nur bei 16 zusätzlichen Modi eine Rolle, bei denen `_TEXT_FG` und `_TEXT_BG` nach verschiedenen Regeln verknüpft werden. Somit ist klar, daß die zusätzlichen Modi nur in Verbindung mit einem Farbbildschirm relevant sind. Die Variablen `_FBASE` und `_FWIDTH` werden entsprechend dem gewünschten Font gesetzt. Den Anfang der Font-Daten können Sie dem Header des gewünschten Fonts entnehmen (Byte 76-79 im Header). `_FWIDTH` muß mit dem Inhalt der Bytes 80 und 81 des Header beschrieben werden. `_SOURCEX` bestimmt, welches Zeichen Sie ausgeben. Hier ist der ASCII-Wert des gewünschten Zeichens einzutragen. `_SOURCEY` ist üblicherweise Null, da ja das Zeichen von der obersten bis zu untersten Scanline abgebildet werden soll.

`_DELX` als Breite des Zeichens kann berechnet werden, indem der Eintrag der Character-Offset-Table des gewünschten Zeichens vom nächsten Eintrag abgezogen wird. Das Ergebnis ist die Breite des Zeichens in Pixeln. `_DELY` muß mit dem Wert der Bytes 82-83 des Headers geladen werden.

Etwas Besonderes hat es mit der `_STYLE`-Variablen auf sich. Hier kann angegeben werden, ob Zeichen original oder verändert dargestellt werden. Die möglichen Veränderungen sind Hervorheben (thicken, Bit 0), Schattieren (lighten, Bit 1), Kursivstellen (Bit 2), und Outlined (Bit 4). Die jeweilige Veränderung wird durch Setzen der Bits eingeschaltet.

Eine weitere Veränderung ist das Scaling. Mit Scaling kann ein Zeichen in seiner Größe verändert werden. Leider klappt beim ST scheinbar nur das Vergrößern der Zeichen, Verkleinerungen konnten nicht erzielt werden. Ist das Scaling-Flag gelöscht (Null), so wird das Zeichen in seiner Originalgröße dargestellt. Ob vergrößert oder verkleinert werden soll, bestimmt das `_T_SCLSTS`-Flag. Hier muß für Vergrößerung ein Wert ungleich Null eingetragen werden. `_DDA_INC` soll den Wert der Vergrößerung oder Verkleinerung enthalten. Allerdings konnte nur bei einem Wert von \$FFFF eine Vergrößerung festgestellt werden.

Eine weitere interessante Variable ist CHUP. Mit Hilfe dieser Variablen können die Zeichen auf dem Bildschirm gedreht werden. Dazu muß der Winkel im Bereich von 0 bis 360 Grad in 1/10 Grad angegeben werden. Aber auch bei dieser Funktion muß eine Einschränkung gemacht werden. Es sind nur Drehungen in Schritten von 90 Grad möglich. Die anzugebenden Werte sind also 0 für Normal-Lage, 900 für 90 Grad Drehung, 1800 (Schrift steht auf dem Kopf) und 2700 für 270 Grad.

Soll mit den Effekten gearbeitet werden, so muß in `_scrchp` ein Pointer auf einen Buffer übergeben werden, in dem TEXTBLT Zwischenwerte speichern kann. Die genaue Größe dieses Buffers ist leider nicht bekannt, allerdings hat sich ein Buffer von 256 Byte immer als ausreichend erwiesen. Bei Vergrößerung muß ein weiterer Buffer (`_scrpt2`) angegeben werden. Als Parameter wird ein Offset angegeben, der sich auf den Anfang des `_scrchp`-Buffers bezieht. Ein Wert von \$40 hat sich hier als ausreichend erwiesen.

\$A009 SHOW MOUSE

Ein Aufruf dieses Opcodes schaltet die Darstellung des Maus-Cursors ein. Der Cursor folgt der Maus, wenn diese bewegt wird. Damit kann die Maus auch in Programmen genutzt werden, wenn auf die Benutzeroberfläche GEM verzichtet wird. Besonders für Spiele ist diese Möglichkeit reizvoll.

Benötigte Parameter werden im INTIN- und CONTRL-Array übergeben. CONTRL(1) sollte vor dem Aufruf gelöscht werden, und CONTRL(3) wird auf Eins gesetzt. INTIN(0) hat eine spezielle Bedeutung. Die Routine zur Verwaltung des Maus-Cursors zählt die Anzahl der Aufrufe zum Entfernen und Einschalten des Cursors. Wurde zweimal der Cursor ausgeschaltet, so muß auch zweimal der Aufruf zum Einschalten erfolgen, bevor er tatsächlich auf dem Bildschirm erscheint. Dieses Verhalten läßt sich aber beeinflussen, indem in INTIN(0) gelöscht wird. Mit diesem Parameter wird der Cursor unabhängig von der Anzahl vorangegangener HIDE-CURSOR-Aufrufe sofort gesetzt. Ist der Wert in INTIN(0) jedoch ungleich Null, so muß die tatsächlich benötigte Anzahl von \$A009-Aufrufen erfolgen, um den Cursor sichtbar zu machen.

\$A00A HIDE CURSOR

Diese Funktion stellt die schon erwähnte Möglichkeit zum Entfernen des Cursors dar. Wird diese Funktion wiederholt aufgerufen, so wird die Anzahl vom Betriebssystem mitgeführt und bestimmt die Anzahl der benötigten Aufrufe SHOW CURSOR, bis der Cursor tatsächlich erscheint.

\$A00B TRANSFORM MOUSE

Ist der Pfeil als Maus-Cursor für Spiele ungeeignet? Machen Sie sich doch einfach Ihren eigenen Cursor. Wie wär's mit einem kleinen Auto, das an Stelle des Pfeils über den Bildschirm flitzt? Mit dem Opcode \$A00B wird Ihrer Fantasie, jedenfalls was die Cursor-Form angeht, freien Lauf gelassen.

Die Parameter müssen im INTIN-Array übergeben werden. Insgesamt werden 34 Worte benötigt. Die folgende Tabelle gibt Aufschluß über Verwendung und mögliche Werte:

INTIN (3)	Maskenfarbindex, normalerweise 0
INTIN (4)	Datenfarbindex, normalerweise 1
INTIN (5) bis (20)	enthalten 16 Worte Cursor-Maske
INTIN (21) bis (36)	enthalten 16 Worte Cursor-Daten

Die Form des Cursors wird bestimmt durch die Cursor-Daten. Jede 1 in den Daten erzeugt einen Punkt auf dem Bildschirm. Wenn jedoch der Cursor über einen Buchstaben oder ein Muster gelegt wird, so kann die Grenze zwischen Hintergrund und Cursor nicht ausgemacht werden. An diesem Punkt tritt die Maske in Aktion. Jedes in der Maske gesetzte Bit löscht den Hintergrund an der angegebenen Stelle. Auf diese Weise läßt sich ein heller Rahmen um die Cursor-Form ziehen. Betrachten Sie einmal den normalen Pfeil-Cursor, wenn er einen Buchstaben oder ein Muster überdeckt, um die Auswirkungen der Maske zu sehen.

\$A00C UNDRAW SPRITE

Dieser Opcode ist in Verbindung mit \$A00D, Draw Sprite, zu sehen. Tatsächlich kennt der ST keine Hardware-Sprites in der Form, wie sie z.B. vom COMMODORE C64 bekannt sind. Die Sprites des ST werden rein softwaremäßig organisiert. Jedes Sprite ist 16x16 Pixel groß. Ein Beispiel für ein ST-Sprite ist z.B. der Maus-Cursor. Er wird über diese Funktion erzeugt.

Um ein zuvor gezeichnetes Sprite zu löschen, muß im Register A2 die Adresse eines Buffers angegeben werden, in dem beim Zeichnen des Sprites der Hintergrund gerettet wird. Der Opcode überträgt nur den Inhalt des Hintergrund-Buffers an die richtige Stelle des Bildschirms. Der Buffer selbst muß für jede Plane 64 Bytes groß sein. Zusätzlich werden zehn Bytes unabhängig von der Anzahl der Planes benutzt. Bei monochromer Darstellung ist der Buffer somit 74 Bytes, in der 320x200 Pixel-Auflösung (vier Planes) $4 \times 64 + 10 = 266$ Bytes groß.

\$A00D DRAW SPRITE

Mit dieser Funktion wird das gewünschte Sprite auf den Bildschirm gezeichnet. Dazu müssen Parameter in den Registern D0, D1, A0 und A2 übergeben werden.

D0 und D1 enthalten die X- und Y-Koordinate der gewünschten Position des Sprite auf dem Bildschirm, kurz Hot Spot genannt. A0 ist ein Pointer auf den sogenannten Sprite-Definition-Block und A2 enthält die Adresse des Sprite-Save-Buffers, in dem der Hintergrund für das Löschen des Sprites übertragen wird.

Der Sprite-Definition-Block muß folgenden Aufbau haben:

Wort 1 : X-Offset zum Hot-Spot
 Wort 2 : Y-Offset zum Hot-Spot
 Wort 3 : Format-Flag 0=VDI-Format, 1=XOR-Format
 Wort 4 : Hintergrund-Farbe (bg, Background)
 Wort 5 : Vordergrund-Farbe (fg, Foreground)

Daran schließen sich 32 Worte an, die das Muster des Sprites enthalten. Dabei muß das Muster in der folgenden Reihenfolge im Speicher stehen:

Wort 6 : Hintergrund-Muster oberste Zeile
 Wort 7 : Vordergrund-Muster oberste Zeile
 Wort 8 : Hintergrund-Muster zweite Zeile
 Wort 9 : Vordergrund-Muster zweite Zeile
 u.s.w.

Die Information im Format-Flag hat die folgende Bedeutung:

VDI-Format

fg	bg	Ergebnis
0	0	Der Hintergrund erscheint
0	1	Die in Wort 4 hinterlegte Farbe erscheint
1	0	Die in Wort 5 hinterlegte Farbe erscheint
1	1	Die in Wort 5 hinterlegte Farbe erscheint

XOR-Format

fg	bg	Ergebnis
0	0	Der Hintergrund erscheint
0	1	Die in Wort 4 hinterlegte Farbe erscheint
1	0	Pixel auf dem Screen wird mit fb-Bit XORed
1	1	Die in Wort 5 hinterlegte Farbe erscheint

\$A00E COPY RASTER FORM

Mit dem \$A00E-Opcode können beliebige Bereiche des Bildschirms kopiert werden. Dabei kann nicht nur innerhalb des Bildschirms kopiert werden, sondern auch aus dem Bildschirm in das freie RAM und, was noch wichtiger ist,

aus dem RAM in den Bildschirm. Aber auch komplette Bildschirmseiten lassen sich mit dem Copy-Raster-Opcode sehr schnell übertragen. Der Name Rasterform drückt jedoch auch eine Einschränkung der Funktion aus. Jede zu übertragende Rasterform muß auf einer Wortgrenze beginnen und aus einem Vielfachen von Worten bestehen.

Die Parameter sind recht zahlreich und werden in den CONTRL-, PTSIN- und INTIN-Arrays übergeben. Zusätzlich müssen zum Copy Raster zwei "Memory Form Definition"-Blöcke im Speicher stehen. Beginnen wir mit den MFD-Blöcken. Da ein Kopiervorgang immer eine Quelle und ein Ziel hat, beschreibt der eine Block den Quell-Speicherbereich, der zweite beschreibt das Ziel. Jeder Block besteht aus zehn Worten. In den ersten beiden Worten ist die Adresse des Speichers enthalten, der durch den Block beschrieben wird. Das dritte Wort gibt die Breite der Form in Pixeln an. Wort 4 bestimmt die Höhe der Form in Pixeln. Das fünfte Wort beschreibt die Breite der Form in Worten. Wort 6 sollte auf 0 gesetzt werden, und Wort 7 gibt an, aus wie vielen Planes die Form besteht. Die restlichen Worte sollten auf Null gesetzt werden, da sie für spätere Erweiterungen reserviert sind.

Sind Ziel und/oder Quelle der Bildschirm, so nimmt uns die Routine die Erstellung des MFDB schon ab: Tragen Sie in einem solchen Falle einfach Null als Adresse ein, und Sie brauchen sich um die weiteren Parameter des MFDB nicht mehr zu kümmern.

Benötigte Parameter für COPY RASTER

INTIN[0]	Bit 0-3	Bei opaque: Logical Operation Bei transparent: Writing Mode (Siehe \$A007, BITBLT)
	Bit 4	= 0 kein Pattern (Raster) benutzen = 1 Pattern benutzen
INTIN[1]		nur bei transparent: Farbindex für "1"-Bits
INTIN[2]		nur bei transparent: Farbindex für "0"-Bits
PTSIN[0]		x-Koordinate der linken oberen Ecke in der Source
PTSIN[1]		y-Koordinate der linken oberen Ecke in der Source
PTSIN[2]		x-Koordinate der rechten unteren Ecke in der Source
PTSIN[3]		y-Koordinate der rechten unteren Ecke in der Source
PTSIN[4]		x-Koordinate der linken oberen Ecke in der Destination
PTSIN[5]		y-Koordinate der linken oberen Ecke in der Destination
PTSIN[6]		x-Koordinate der rechten unteren Ecke in der Destination
PTSIN[7]		y-Koordinate der rechten unteren Ecke in der Destination
CONTRL[7+8]		Adresse Source MFDB

CONTRL[9+10]

	Adresse Destination MFDB	
_patptr	Zeiger auf Pattern (falls benutzt)	
_multifill	0 =	Pattern hat eine Ebene
	1 =	Pattern hat mehrere Ebenen
_COPYTRAN	0 =	opaque, n-Ebenen Quelle und n-Ebenen Ziel
	1 =	transparent Quelle mit einer Ebene wird durch alle Zielebenen ("transparent") kopiert.

Aufbau des Memory Form Definition Blocks

Offset	Größe	Bedeutung
0	L	Zeiger auf Raster-Image
4	W	Breite des Rasters in Pixel
6	W	Höhe des Rasters in Pixel
8	W	Breite des Rasters in Worten
10	W	Format Flag 0 = gerätespezifisch 1 = Standard-Format
12	W	Anzahl der Bildebenen
14	3W	reserviert

Soll die Funktion COPY RASTER benutzt werden, so muß das Rasterimage im gerätespezifischen Format vorliegen. (Beim Standardformat sind die einzelnen Bildebenen nicht wortweise verschachtelt, sondern als Ganzes hintereinander angeordnet.) Hier noch ein paar Anmerkungen zu den Begriffen "opaque" und "transparent". Beim "opaque"-Kopieren werden einfach die entsprechenden Farbebenen von Quelle und Ziel sowie ein eventuelles Raster über eine logische Operation mit einem Wert von 0 bis 15 verknüpft (siehe auch \$A007, BITBLT). Hierbei müssen die Anzahl der Farbebenen bei Ziel und Quelle übereinstimmen, sonst bricht die Funktion ab. Beim Opaque-Kopieren werden auch die Werte in INTIN[1] und [2] nicht benötigt.

Etwas Besonderes hat es dagegen beim Copy-Transparent auf sich. Hier wird der Quellbereich, der nur aus einer Farbebene bestehen darf, in einen mehrfarbigen Zielbereich kopiert. Der Quellbereich besteht also aus nur zwei verschiedenen Farben, repräsentiert durch die Bits "0" und "1". Sie können nun bestimmen, in welcher Farbe diese Pixel im Zielbereich erscheinen. Geben Sie

dazu einfach die entsprechenden Farbnummern in INTIN[1] und INTIN[2] an. In INTIN[0] wird hierbei nicht die logische Operation, sondern der Writing Mode angegeben:

INTIN[0]	Writing Mode
1	Replace-Mode
2	Transparent-Mode
3	XOR-Mode
4	Reverse-Transparent-Mode

Dieses Verfahren bietet sich immer dann an, wenn ein Quellbereich nur zwei-farbig ist und er sowohl beim Monochrom- als auch beim Farbbildschirm verwendet wird. Beim Kopieren auf monochrom erscheint das Bild naturgemäß nur in schwarzweiß, beim Farbbildschirm können Sie die beiden Farben aus der zur Verfügung stehenden Palette frei auswählen. Die Disketten-Icons auf dem Desktop werden z.B. nach diesem Verfahren kopiert.

Copy Raster Opaque ist im übrigen identisch mit der VDI-Funktion 109, vro_cpyfm, während Copy Raster Transparent der VDI-Funktion 121, vrt_cpyfm, entspricht.

\$A00F CONTOUR FILL (FLOOD FILL)

Der Line-A-Opcode \$A00F ist von Atari derzeit nicht dokumentiert. Wenn man sich jedoch einmal mit Hilfe eines Disassemblers das Programm anschaut, das im Fall eines \$A00x-Opcodes ausgeführt wird, so stellt man schnell fest, daß da "noch was sein" muß. Schwieriger ist es jedoch, dann herauszubekommen, welche Funktion der \$A00F-Opcode ausführt. Nun, uns ist dieses Ent-wirr-Spiel gelungen. \$A00F stellt eine weitere Fill-Routine zur Verfügung. Dieser Fill ist identisch zur VDI-Funktion 103 Contour Fill.

Contour Fill benötigt die Angabe einer XY-Koordinate und ein Mode-Wort. Die Koordinaten werden in PTSIN(0) und PTSIN(1) erwartet, das Mode-Wort in INTIN(0). Das Mode-Wort hat die folgende Bedeutung: Wird hier ein positiver Wert eingetragen, so wird dieser Wert als Farbindex betrachtet. Gefüllt wird dann eine Fläche, die entweder durch den Bildschirmrand oder die angegebene Farbe bestimmt wird. Ist der Wert jedoch negativ, so wird die Begrenzung durch jede andere als die Farbe des Startpunktes bestimmt.

Weiterhin verlangt diese Funktion in _FILL_ABORT die Adresse einer Routine, die in bestimmten Abständen aufgerufen wird. Damit ist es möglich, die Funktion abubrechen. Liefert die Funktion den Wert 0 in D0 zurück, wird die

Fill-Funktion fortgeführt, ansonsten wird das Füllen abgebrochen. Die von dieser Funktion ausgewerteten Variablen sind das Clipping, der Schreibmodus, Patternpointer und Patternmask sowie Multifill.

Die "Line A"-Variablen im Überblick

Nach der Initialisierung \$A000 erhält man in D0 und A0 die Adresse eines Variablenbereichs zurück, in dem über 50 Line-A-Variablen enthalten sind. Wesentliche Variablen wurden bereits bei den verschiedenen Aufrufen erklärt, nicht jedoch die Lage der Variablen innerhalb des Variablenblocks. Diese Aufstellung soll jetzt erfolgen. Bei der Benennung der Variablen haben wir uns an die Namen gehalten, die in der offiziellen Atari-Dokumentation genannt werden.

Offset ist der Wert, der angegeben werden muß, wenn auf die Variable registerrelativ zugegriffen wird.

Offs.	Name	Größe	Funktion
0	v_planes	Word	Anzahl der Planes
2	v_lin_wr	Word	Bytes pro Scanline
4	CTRL	Long	Pointer auf das CTRL-Array
8	INTIN	Long	Pointer auf das INTIN-Array
12	PTSIN	Long	Pointer auf das PTSIN-Array
16	INTOUT	Long	Pointer auf das INTOUT-Array
20	PTSOUT	Long	Pointer auf das PTSOUT-Array
24	_FG_BP_1	Word	Plane 0 Farbwert
26	_FG_BP_2	Word	Plane 1 Farbwert
28	_FG_BP_3	Word	Plane 2 Farbwert
30	_FG_BP_4	Word	Plane 3 Farbwert
32	_LSTLIN	Word	sollte -1 (\$FFFF) sein
34	_LN_MASK	Word	bei \$A003 Linienmuster
36	_WRT_MODE	Word	Schreibmodus (0 = Replace-Mode 1 = Transparent-Mode 2 = Xor-Mode 3 = Invers-Transparent-Mode)
38	_X1	Word	X1-Koordinate
40	_Y1	Word	Y1-Koordinate
42	_X2	Word	X2-Koordinate
44	_Y2	Word	Y2-Koordinate
46	_patptr	Long	Pointer auf das Füllmuster (siehe \$A004)
50	_patmsk	Word	Füllmuster-"Maske" (siehe \$A004)
52	_multifill	Word	0 = Füllmuster ist nur für eine Plane 1 = Füllmuster ist für Multi-Plane ausgelegt
54	_CLIP	Word	0 = kein Clipping eingeschaltet (siehe \$A005) ungleich 0 = Clipping

56	_XMN_CLIP	Word	und
58	_YMN_CLIP	Word	definieren linke obere Ecke des sichtbaren Bereichs bei Clipping
60	_XMX_CLIP	Word	und
62	_YMX_CLIP	Word	definieren rechte untere Ecke des sichtbaren Bereichs bei Clipping
64	_XACC_DDA	Word	sollte vor jedem Aufruf von TXTBLT auf \$8000 gesetzt werden
66	_DDA_INC	Word	Vergrößerungs-/Verkleinerungsfaktor \$FFFF für Vergrößerung, Verkleinerung funktioniert nicht (?)
68	_T_SCLSTS	Word	0 = Verkleinerung (?) 1 = Vergrößerung
70	_MONO_STAT	Word	1 = kein Proportional-Font 0 = Proportionalschrift oder Breite der Zeichen ändert sich bei Fettschrift oder Kursivstellen
72	_SOURCEX	Word	X-Koordinate des Zeichens im Font
74	_SOURCEY	Word	Y-Koordinate des Zeichens im Font (0)

Bemerkung:

SOURCEX ist der Wert des Zeichens aus der Horizontal-Offset-Tabelle (HOT) und läßt sich nach folgender Formel berechnen:

$SOURCEX = HOT\text{-}Element (ASCII\text{-}Wert \text{ minus } FIRST\ ADE)$

Die Variable FIRST ADE ist in den Bytes 36,37 des Font Header enthalten (siehe Beispiel).

76	_DESTX	Word	X-Position Zeichen auf den Bildschirm
78	_DESTY	Word	Y-Position Zeichen auf den Bildschirm
80	_DELX	Word	Breite des Zeichens
82	_DELY	Word	Höhe des Zeichens

Bemerkung:

DELX kann berechnet werden nach folgender Formel:

$DELX = SOURCEX + 1 \text{ minus } SOURCEX$

(siehe \$A008)

DELY ist der Wert FORM Height aus den Bytes 82,83 des Font Header

84	_FBASE	Long	Pointer auf Start der Font-Daten
88	_FWIDTH	Word	Breite der Font-Form
90	_STYLE	Word	Flags spezielle Effekte (siehe \$A008)
92	_LITEMASK	Word	Maske für das Schattieren
94	_SKEWMASK	Word	Maske für Kursiv-Schrift
96	_WEIGHT	Word	Anzahl der Bits, um die das Zeichen verbreitert werden soll
98	_R_OFF	Word	Offset für Kursiv-Schrift
100	_L_OFF	Word	Offset für Kursiv-Schrift

Bemerkung:

Die vorgenannten fünf Variablen sollten mit den entsprechenden Werten aus dem Font-Header versorgt werden.

102	<code>_SCALE</code>	Word	0 = kein Scaling 1 = Scaling (vergrößern/verkleinern)
104	<code>_CHUP</code>	Word	Winkel für Zeichenrotation 0 = normale Zeichendarstellung 900 = 90 Grad gedreht 1800 = 180 Grad gedreht 2700 = 270 Grad gedreht
106	<code>_TEXT_FG</code>	Word	Vordergrundfarbe für Textdarstellung
108	<code>_schrtp</code>	Long	Adresse des für Erzeugung der spez. Text-Effekte benötigten Buffers
112	<code>_schrtp2</code>	Word	Offset des Vergrößerungsbuffers im <code>_schrtp</code> -Buffer (\$40)
114	<code>_TEXT_BG</code>	Word	Hintergrundfarbe für Textdarstellung
116	<code>_COPYTRAN</code>	Word	0 = Copy opaque, n Ebenen Quelle nach n Ebenen Ziel 1 = Copy transparent, Quelle hat eine Ebene nach n Ebenen Ziel
118	<code>_FILL_ABORT</code>	Long	Zeiger auf eine Routine zum Test auf Abbruch der Fill-Funktion Return-Wert 0 = kein Abbruch, 1 = Abbruch

Es gibt noch eine Reihe weiterer Line-A-Variablen, die einen negativen Offset haben und von der BIOS-BildschirmAusgabe (VT52 Emulator, zugehörige Variablen beginnen mit `V_`) sowie vom VDI genutzt werden.

Dadurch ist es möglich, auf "legale" Weise z.B. an die Cursor-Position, Mauskoordinaten, Status der Maustasten oder die Form die Mauszeigers zu kommen. Weiterhin stehen hier noch Variablen, die die Textausgabe des VDI betreffen. Dadurch ist es z.B. möglich, den System-Font durch einen eigenen Font zu ersetzen, der dann bei allen GEM- und VDI-Funktionen benutzt wird.

Offs.	Name	Größe	Funktion
-906	<code>CUR_FONT</code>	Long	Zeiger auf den aktuellen Font-Header
-902 - 856	reserviert		
-856	<code>M_POS_HX</code>	Word	Mouse Hot Spot, X-Koordinate
-854	<code>M_POS_HY</code>	Word	Mouse Hot Spot, Y-Koordinate
-852	<code>M_PLANES</code>	Word	Writing-Mode für den Maus-Cursor 1 bedeutet "normal", -1 bedeutet "XOR"

Die Maus-Definition besteht aus 2 Ebenen, dem Vordergrund (V) und dem Hintergrund (H).

Die Tabelle zeigt das Ergebnis der vier möglichen Kombinationen. Siehe auch Line A \$A00B, Transform Mouse.

V	H	Normal	XOR
0	0	Ziel	Ziel
0	1	Hintergr	Hintergrund
1	0	Vordergr	Not Ziel
1	1	Vordergr	Vordergrund
-850	M_CDB_BG	Word	
-848	M_CDB_FG	Word	
-846	MASK_FORM	32 Word	
-782	INQ_TAB	45 Word	
-692	DEV_TAB	45 Word	
-602	G_CURX	Word	
-600	GCURY	Word	
-598	M_HID_CT	Word	
-596	MOUSE_BT	Word	
-594	REQ_COL	48 Word	
-498	SIZ_TAB	15 Word	

Die Zielfarbe (Bildschirm) ist unverändert
Die Farbe des Maus-"Hintergrunds" ist zu sehen

Vordergrundfarbe bzw. Bildschirmfarbe wird invertiert

Die Farbe des Maus-"Vordergrunds" ist zu sehen

Pixelfarbe für Mouse-Hintergrund

Pixelfarbe für Mouse-Vordergrund

Mouse-Daten alternierend je 16mal Hintergrund und Vordergrund

Ergebnis des VDI-Aufrufs vq_extend, 45 Worte

Ergebnis des VDI-Aufrufs v_opnwk, 45 Worte

Aktuelle X-Position der Maus

Aktuelle Y-Position der Maus

Mouse Hide Count, Schachteltiefe für Mouse Hide-Aufrufe

Aktueller Mouse Button-Status

Bit 0 = 1 linke Taste gedrückt

Bit 1 = 1 rechte Taste gedrückt

3mal 16 Worte für RGB-Anteil der Farben, siehe VDI- Aufruf vq_color

VDI-Daten für Text- Linien- und Marker-Ausgabe

0 Minimale Zeichenbreite

1 Minimale Zeichenhöhe

2 Maximale Zeichenbreite

3 Maximale Zeichenhöhe

4 Minimale Linienbreite

5 Reserviert

6 Maximale Linienbreite

7 Reserviert

8 Minimale Markerbreite

9 Minimale Markerhöhe

10 Maximale Markerbreite

11 Maximale Markerhöhe

12 - 14 Reserviert

-468	reserviert	Word	
-466	reserviert	Word	
-464	CUR_WORK	Long	Zeiger auf die Attribut-Struktur der aktuellen virtuellen Workstation
-460	DEF_FONT	Long	Zeiger auf Default-Fontheadere
-456	FONT_RING	4 Long	Diese Variable ist ein Feld mit vier Zeigern, die jeweils auf eine verkettete Liste von Fontheadern zeigen. Der erste Eintrag zeigt auf die Liste der System-Fonts, der zweite und dritte Eintrag dienen zur Erweiterung. Der vierte Eintrag beendet die Kette und muß immer Null sein. Wenn das VDI einen Font sucht, beginnt es mit dem ersten Eintrag in FONT_RING und durchsucht alle Fonts, die in dieser Liste verkettet sind. Ist die Liste zu Ende, so werden die beiden Erweiterungslisten durchsucht. Der Null-Eintrag im vierten Feld beendet die Suche. Die ersten beiden Einträge zeigen auf residente Fonts, der dritte Eintrag wird von GDOS benutzt. Wenn das VDI einen Font mit bestimmter Größe und Typ sucht, so werden die Listen durchsucht, bis zum ersten Mal ein Font vom gewünschten Typ gefunden wird. Dann sucht das VDI weiter nach der korrekten Font-Höhe. Die Suche wird abgebrochen, wenn entweder ein Font eines anderen Types oder der Nulleintrag im FONT_RING gefunden wird. Damit das VDI die Fonts finden kann, müssen Fonts gleichen Typs nach aufsteigender Höhe verkettet sein. Wenn vom Anwender residente Fonts eingebunden werden sollen, so sollte FONT_RING[0] auf diese Font-Liste zeigen. Der letzte Verkettungszeiger des letzten Fonts sollte dann auf den ursprünglichen Inhalt von FONT_RING[0] zeigen.
-440	FONT_COUNT	Word	Anzahl der Fonts in der FONT_RING-Liste
-438 - -348	reserviert		
-348	CUR_MS_STAT	Byte	Aktueller Mouse-Status
		Bit 0	linke Mouse-Taste
		Bit 1	rechte Mouse-Taste
		Bit 2 - 4	reserviert
		Bit 5	Maus wurde bewegt

			Bit 6	rechte Mouse-Taste geändert
			Bit 7	linke Mouse-Taste geändert
-347	reserviert	Byte		
-346	V_HID_CNT	Word		Hide Count (Schachteltiefe) für Textcursor
-344	CUR_X	Word		X-Position der Maus für nächstes Mouse Zeichnen
-342	CUR_Y	Word		Y-Position der Maus für nächstes Mouse Zeichnen
-340	CUR_FLAG	Byte		<> Null: Maus wird in der VBLANK-Routine neu gezeichnet
-339	MOUSE_FLAG	Byte		<> Null wenn Mouse-Interrupt-Verarbeitung erlaubt ist
-338	reserviert	Long		
-334	V_SAV_X	Word		gerettete X-Position des Textcursors
-332	V_SAV_Y	Word		gerettete Y-Position des Textcursors
-330	SAVE_LEN	Word		Höhe der geretteten Mouse-Form (Scanlinien)
-328	SAVE_ADR	Long		Bildschirmadresse des ersten geretteten Words unter der Maus
-324	SAVE_STAT	Word		Information über gerettete Maus
			Bit 0 = 1	Buffer-Inhalt gültig
			Bit 1 = 1	Breite der geretteten Daten ist ein Langwort
			Bit 1 = 0	Breite der geretteten Daten ist ein Wort
-322	SAVE_AREA	128 Word		geretteter Bildschirminhalt unter der Maus bis zu 4 Ebenen und 16 Langworten pro Ebene
-66	USER_TIM	Long		USER-TIM zeigt auf eine Routine des Benutzers, die bei jedem
-62	NEXT_TIM	Long		System-Timer-Tick aufgerufen wird. Am Ende der Routine sollte nach NEXT-TIM gesprungen werden. Siehe
				Auf VDI-Aufruf vex_time
-58	USER_BUT	Long		Zeiger auf Mouse-Button-Routine des Benutzers
-54	USER_CUR	Long		Zeiger auf Mouse-Routine des Benutzers
-50	USER_MOT	Long		Zeiger auf Benutzer-Routine für Mouse-Bewegung
-46	V_CEL_HT	Word		Höhe der Alphatext-Zelle in Pixel
-44	V_CEL_MX	Word		Anzahl Textzeichen pro Zeile minus 1
-42	V_CEL_MY	Word		Anzahl Textzeilen minus 1
-40	V_CEL_WR	Word		Byte-Offset zur nächsten Textzeile
-38	V_COL_BG	Word		Farbindex für Hintergrundfarbe
-36	V_COL_FG	Word		Farbindex für Vordergrundfarbe
-34	V_CUR_AD	Long		Bildschirmadresse des Text-Cursors
-30	V_CUR_OF	Word		Byte-Offset vom Bildschirmanfang zur ersten Textzeile
-28	V_CUR_X	Word		Text-Cursor, X-Position
-26	V_CUR_Y	Word		Text-Cursor, Y-Position
-24	V_PERIOD	Byte		Blinkfrequenz des Text-Cursors (Anzahl Bildwechsel)
-23	V_CUR_CT	Byte		Count-Down-Zähler für Cursor-Blinken

-22	V_FNTAD	Long	Zeiger auf aktuelle Font-Daten (nicht proportional)
-18	V_FNT_ND	Word	Höchster ASCII-Code im Font
-16	V_FNT_ST	Word	Kleinster ASCII-Code im Font
-14	V_FNT_WD	Word	Breite des Fonts in Byte (Form width)
-12	V_REZ_HZ	Word	Horizontale Auflösung in Pixel
-10	V_OFF_AD	Long	Zeiger auf Font Offset-Tabelle
-6	reserviert	Word	
-4	V_REZ_VT	Word	Vertikale Auflösung in Pixel
-2	BYTES_LN	Word	MFDB-Breite des Bildschirms

27. GEM-VDI

Die nun folgenden Funktionen des GEM-VDI sind mit all ihren Parametern für die Programmierung in C vorgegeben. Assembler-Programmierer müssen jedoch die selben Parameter in die gleichen Felder (INTIN usw.) schreiben. Dabei bedeutet die C-Definition INT Wortbreite und LONG Langwortbreite des jeweiligen Parameters. Danach wird mit der Adresse des VDI-Parameterblocks im Register D1 und dem Wert \$73 in D0 über den Befehl TRAP #2 das VDI aufgerufen. Der in Contrl(0) einzutragende Opcode wird jeweils neben dem Funktionsnamen angegeben (OP:n).

Bei einigen Funktionen werden immer die gleichen Parameter übergeben bzw. übernommen. Dies ist durch die Angabe der Parameter als Standard gekennzeichnet. Diese Standard-Parameter sind:

für die Eingabe	für die Ausgabe		
contrl(0)	Operationskode (OP)	contrl(2)	0
contrl(1)	0	contrl(4)	0
contrl(3)	0		
contrl(6)	Handle		

Kontroll-Funktionen

OPEN WORKSTATION OP: 1

Laden eines Gerätetreibers für ein Ein- oder Ausgabegerät. ATARI ST hat keine solchen Treiber, deshalb ist die Funktion (augenblicklich) noch nicht anzuwenden.

Eingabeparameter:

intin(0)	Zu ladender Treiber
intin(1)	Linientyp
intin(2)	Farbe Linienoperation
intin(3)	Markierungstyp
intin(4)	Markierungsfarbe
intin(5)	Zeichensatz
intin(6)	Schriftfarbe
intin(7)	Fülltyp
intin(8)	Index des Musters

intin(9)	Füllfarbe
intin(10)	Koordinatenflag
	0 = Normal
	1 = res.
	2 = Raster

Ausgabeparameter:

intout(0)	Rasterbreite in Pixel
intout(1)	Rasterhöhe in Pixel
intout(3)	Pixelbreite in 1/1000mm
intout(4)	Pixelhöhe in 1/1000mm
intout(5)	Schriftgrößen, Anz.
intout(6)	Linientypen, Anz.
intout(7)	Linienbreiten, Anz.
intout(8)	Markierungsarten, Anz.
intout(9)	Markierungsgrößen, Anz.
	0 = >32767
	1 = 2 >
	2 = Anzahl
intout(10)	Geräte-Zeichensätze, Anz.
intout(11)	Muster, Anz.
	1 = Tastatur
	2 = Maus + 1
intout(12)	Schraffiertypen, Anz.
intout(13)	Farben, Anz.
	1 = Tastatur
	2 = andere
intout(14)	Grafik-Grundfunktionen, Anz.
intout(15)	bis
intout(24)	Unterstützende Grafikfunkt.
	1 = Funktions-
	2 = andere
intout(34)	Grundfunktionen-Attribute
intout(35)	Farbdarstellung 0/1
intout(36)	Textrotation 0/1
intout(37)	ausgef. Bereiche 0/1
intout(38)	cell_Array-Funkt. 0/1
intout(39)	verfügbare Farben 0/n
intout(40)	Grafik-Cursor 1/2
intout(41)	wertverändernde Eingab.
intout(42)	Auswahltasten
intout(43)	alphanum. Eingabe
intout(44)	Geräteart

0 = Ausgabe
 1 = Eingabe
 2 = Ein-/Ausgabe
 3 = reserviert
 4 = Metafile
 ptsout(0) min. Zeichenbreite
 ptsout(1) min. Zeichenhöhe
 ptsout(2) max. Zeichenbreite
 ptsout(3) max. Zeichenhöhe
 ptsout(4) min. Zeilenbreite
 ptsout(5) 0
 ptsout(6) max. Zeilenbreite
 ptsout(7) 0
 ptsout(8) min. Markierungsbreite
 ptsout(9) min. Markierungshöhe
 ptsout(10) max. Markierungsbreite
 ptsout(11) max. Markierungshöhe

Zugriff in C

```

int work_in[12];
int work_out[57];
int handle;
v_opnwk(work_in, &handle, work_out);
  
```

CLOSE WORKSTATION	OP: 2
-------------------	-------

Schließen eines mit OPEN WORKSTATION geöffneten Arbeitsgerätes. Auf dem ST zur Zeit noch nicht einsetzbar.

Eingabeparameter:

Standard

Ausgabeparameter:

contrl(2) ptsin-Punkte, Anz.
 contrl(4) intout-Array-Länge

Zugriff in C

```

int handle;
v_clswk(handle)
  
```

OPEN VIRTUAL SCREEN WORKSTATION	OP: 100
---------------------------------	---------

Öffnen des Bildschirms als Workstation. Für alle Bildschirm-Anwendungen erforderlich.

Eingabeparameter:

intin(0)	Zu ladender Treiber
intin(1)	Linientyp
intin(2)	Farbe Linienoperation
intin(3)	Markierungstyp
intin(4)	Markierungsfarbe
intin(5)	Zeichensatz
intin(6)	Schriftfarbe
intin(7)	Fülltyp
intin(8)	Index des Musters
intin(9)	Füllfarbe
intin(10)	Koordinatenflag
	0 = Normal
	1 = res.
	2 = Raster

Ausgabeparameter:

intout(0..44)	entsprechend Ihrer Monitorversion
ptsout(0..11)	entsprechend Ihrer Monitorversion

Zugriff in C `int work_in[12]`
 `int work_out[57]`
 `int handle;`
 `appl_init()`
 `v_openvwk(work_in,&handle,work_out)`

Bemerkungen: Siehe `appl_init`.

Es wird automatisch CLEAR WORKSTATION durchgeführt.

CLOSE VIRTUAL SCREEN WORKSTATION OP: 101

Das Ausgabegerät wird geschlossen. Weitere Ausgaben sind danach nicht mehr möglich.

Eingabeparameter:

Standard

Ausgabeparameter:

Standard

Zugriff in C `int handle;`
 `appl_exit()`
 `v_clsvwk(handle)`

CLEAR WORKSTATION OP: 3

Löscht den Bildschirm bzw. führt ein FORM FEED auf Drucker/Plotter aus. Bei einem Metafile wird der Opcode gespeichert.

Eingabeparameter:

Standard

Ausgabeparameter:

Standard

Zugriff in C `int handle;`
 `v_clrwk(handle)`

Bemerkungen:

Wird nach Open Virtual Screen Workstation automatisch durchgeführt.

UPDATE WORKSTATION OP: 4
--

Grafikbefehle (für Drucker und Plotter) werden nicht direkt ausgeführt, sondern in einem speziellen Puffer gespeichert und mit dieser Funktion ausgeführt.

Eingabeparameter:

Standard

Ausgabeparameter:

Standard

Zugriff in C `int handle;`
 `v_updwk(handle)`

LOAD FONTS OP:119

Anfrage, wieviel Zeichensätze auf einem Gerät zur Verfügung stehen und Laden dieser Zeichensätze.

Eingabeparameter:

intin(0) reserviert (1)

Ausgabeparameter:

intout(0) Zusätzliche Zeichensätze, Anzahl

Zugriff in C int handle;
 int additional;
 int select;
 additional=vst_load_fonts(handle, select);

Bemerkungen:

Beim ST stehen keine weiteren Zeichensätze für den Bildschirm zur Verfügung.

UNLOAD FONTS OP: 120

Löscht die geladenen Zeichensätze und stellt somit Speicherplatz zur Verfügung. (Die Standardzeichensätze bleiben erhalten.)

Eingabeparameter:

intin(0) reserviert

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int select; (Eingabeparameter)
 vst_unload_fonts(handle, select);

SET CLIPPING RECTANGLE OP: 129

Abschalten des Ausschnittes, welcher für begrenzte Ausgaben definiert wurde.

Eingabeparameter:

intin(0) Ausschnitt

0 = aus

1 = ein

ptsin(0) x-Ordinate obere linke Ecke

ptsin(1) y-Ordinate obere linke Ecke
 ptsin(2) x-Ordinate untere rechte Ecke
 ptsin(3) y-Ordinate untere rechte Ecke

Ausgabeparameter:

keine

Zugriff in C int handle;
 int clip_flag; (für intin)
 int pxyarray[4]; (für ptsin)
 vs_clip(handle, clip_flag, pxyarray)

Bemerkungen:

Nach Öffnen des Arbeitsgerätes clip_flag=0.

Ausgabe-Funktionen

POLYLINE OP: 6

Verbinden beliebiger Bildschirmkoordinaten durch eine Linie.

Eingabeparameter:

ptsin(0) x-Ordinate des 1. Pixels
 ptsin(1) y-Ordinate des 1. Pixels
 : : : :
 ptsin(n) x-Ordinate des n. Pixels
 ptsin(n+1) y-Ordinate des n+1. Pixels

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int count; (Anzahl der Koordinaten)
 int pxyarray[2*count] (für ptsin)
 v_pline(handle, count, pxyarray);

POLYMARKEROP: 7

Es werden Markierungen auf dem Bildschirm ausgegeben, die durch die Attribut-Funktionen festgelegt sind.

Eingabeparameter:

ptsin(0)	x-Ordinate des 1. Pixels
ptsin(1)	y-Ordinate des 1. Pixels
:	:
ptsin(n)	x-Ordinate des n. Pixels
ptsin(n+1)	y-Ordinate des n+1. Pixels

Ausgabeparameter:

Standard

TEXT OP: 8

Ausgabe von Text auf dem Grafikbildschirm. Läßt sich in C gut anstelle von PRINTF verwenden. Der Text erscheint linksbündig an der spezifizierten Position. Mit den Attribut-Funktionen können andere Ausgabeformen angegeben werden. Der Ausgabe-String muß mit ASCII(0) enden (in C Standard).

Eingabeparameter:

intin(0-n)	Enthält den String in den niederwertigen Bytes.
ptsin(0)	x-Ordinate
ptsin(1)	y-Ordinate

Ausgabeparameter:

Standard

Zugriff in C

```
int handle;
int x;           (ptsin 0)
int y;           (ptsin 1)
char string[länge] (intin)
v_gtext(handle, x, y, string)
```

Bemerkungen:

Automatische Übertragung String -> intin in C

FILLED AREA OP: 9

Ausfüllen der Fläche im Vieleck. Die Definition des Vielecks erfolgt durch Übergabe von Koordinaten. Die Art des Füllens wird durch Attribut-Funktionen bestimmt.

Eingabeparameter:

ptsin(0) x-Ordinate des 1. Pixels
 ptsin(1) y-Ordinate des 1. Pixels
 : : :
 ptsin(n) x-Ordinate des n. Pixels
 ptsin(n+1) y-Ordinate des n+1. Pixels

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int count; (Anzahl der Koordinaten)
 int pxyarray[2*count] (ptsin)
 v_fillarea(handle, count, pxyarray)

CELL ARRAY OP: 10

Monochrom: Es wird ein Rechteck in der aktuellen Linienfarbe und Breite gezeichnet.

Farbe: Es wird ein Rechteck gezeichnet, dieses unterteilt in Tabellenform und die Grafikpunkte entsprechend ihrer Lage in der Tabelle eingefärbt. Jedem Tabellenelement wird eine Farbe zugeordnet.

Eingabeparameter:

contrl(7) Länge der Zeilen in intin
 contrl(8) Farbarray-Zonen, Anzahl
 contrl(9) Farbarray-Zeilen, Anzahl
 contrl(10) Zeichenmodus (-> Attributf.)
 intin(0-n) Farbe jeder Tabellenzone
 ptsin(0) x-Ordinate untere linke Ecke
 ptsin(1) y-Ordinate untere linke Ecke
 ptsin(2) x-Ordinate obere rechte Ecke
 ptsin(3) y-Ordinate obere rechte Ecke

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int pxyarray[4] (ptsin)
 int row_lenght; (contrl 7)

```

int el_used; (contrl 8)
int num_rows; (contrl 9)
int wrt_mode; (contrl 10)
int colarray[num_rows*el_used) (intin)
v_cellarray(handle, pxyarray, row_lenght, el_used, num_rows,
wrt_mode, colarray)

```

CONTOUR FILL

OP: 103

Ausfüllen eines Bereichs, der von Punkten gleicher Farbe bzw. vom Bildschirmrand begrenzt wird.

Eingabeparameter:

intin(0) Füllfarbe
 ptsin(0) x-Ordinate des Füll-Startpixels
 ptsin(1) y-Ordinate des Füll- Startpixels

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int index; (intin)
 int x; (ptsin 0)
 int y; (ptsin 1)
 v_contourfill(handle, x, y, index)

Bemerkungen:

Funktion wird nicht von allen Ausgabegeräten unterstützt.

FILL RECTANGLE

OP: 114

Ausfüllen eines durch gegenüberliegende Koordinaten definierten Rechtecks.

Eingabeparameter:

ptsin(0) x-Ordinate obere linke Ecke
 ptsin(1) y-Ordinate obere linke Ecke
 ptsin(2) x-Ordinate untere rechte Ecke
 ptsin(3) y-Ordinate untere rechte Ecke

Ausgabeparameter:

Standard

Zugriff in C `int handle;`
 `int pxyarray; (ptsin)`
 `vr_recfl(handle, pxyarray)`

Grafische Grundfunktionen OP: 11

Die grafischen Grundfunktionen (Linien, Rechtecke, Kreise usw.) werden alle unter dem Opcode 11 zusammengefaßt. Die Unterscheidung der verschiedenen Funktionen wird durch die Funktionskennung (FK) getroffen, welche in Contrl(5), also "contrl + 10", eingetragen werden muß. Diese FK wird bei den folgenden Funktionen mit angegeben:

BAR OP: 11, FK: 1

Die Funktion zeichnet entsprechend den Attribut-Funktionen ausgefüllte Balken.

Eingabeparameter:

`ptsin(0)` x-Ordinate obere linke Ecke
`ptsin(1)` y-Ordinate obere linke Ecke
`ptsin(2)` x-Ordinate untere rechte Ecke
`ptsin(3)` y-Ordinate untere rechte Ecke

Ausgabeparameter:

Standard

Zugriff in C `int handle;`
 `int pxyarray[4] (ptsin)`
 `v_bar(handle, pxyarray)`

ARC OP: 11, FK: 2

Zeichnet einen Kreisausschnitt gemäß der spezifizierten Anfangs- und Endwinkel.

Eingabeparameter:

`intin(0)` Anfangswinkel (0-3600 = 1/10tel Grad)
`intin(1)` Endwinkel (0-3600 = 1/10tel Grad)
`ptsin(0)` x-Ordinate Mittelpunkt
`ptsin(1)` y-Ordinate Mittelpunkt
`ptsin(2) bis`

ptsin(5) Enthält 0 (in Assembler zwingend)
 ptsin(6) Radius
 ptsin(7) Enthält 0 (in Assembler zwingend)

Ausgabeparameter:

Standard

Zugriff in C int handle; int x; (ptsin 0)
 int y; (ptsin 1)
 int radius; (ptsin 6)
 int begang; (intin 0)
 int endang; (intin 1)
 v_arc(handle, x, y, radius, begang, endang);

Bemerkungen:

Funktion wird nicht von jedem Gerät unterstützt.

PIE	OP: 11, FK: 3
-----	---------------

Zeichnet einen entsprechend den Attribut-Funktionen ausgefüllten Kreisabschnitt, der durch Anfangs- und Endwinkel definiert wird.

Eingabeparameter:

intin(0) Anfangswinkel (0-3600 = 1/10tel Grad)
 intin(1) Endwinkel (0-3600 = 1/10tel Grad)
 ptsin(0) x-Ordinate Mittelpunkt
 ptsin(1) y-Ordinate Mittelpunkt
 ptsin(2) bis
 ptsin(5) Enthält 0 (in Assembler zwingend)
 ptsin(6) Radius
 ptsin(7) Enthält 0 (in Assembler zwingend)

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int x; (ptsin 0)
 int y; (ptsin 1)
 int radius; (ptsin 6)
 int begang; (intin 0)
 int endang; (intin 1)
 v_pieslice(handle, x, y, radius, begang, endang);

Bemerkungen:

Funktion wird nicht von jedem Gerät unterstützt.

CIRCLE	OP: 11, FK: 4
--------	---------------

Zeichnet einen ausgefüllten Kreis entsprechend der Einstellung durch die Attribut-Funktionen.

Eingabeparameter:

ptsin(0)	x-Ordinate Mittelpunkt
ptsin(1)	y-Ordinate Mittelpunkt
ptsin(2) bis	
ptsin(5)	Enthält 0 (in Assembler zwingend)
ptsin(6)	Radius
ptsin(7)	Enthält 0 (in Assembler zwingend)

Ausgabeparameter:

Standard

Zugriff in C

```

int handle;
int x;(ptsin 0)
int y;(ptsin 1)
int radius;(ptsin 6)
v_circle(handle, x, y, radius);

```

Bemerkungen:

Funktion wird nicht von jedem Gerät unterstützt.

ELLIPTICAL ARC	OP: 11, FK: 6
----------------	---------------

Zeichnet einen elliptischen Ausschnitt entsprechend den eingegebenen Anfangs- und Endwinkel und den Attributen, welche durch die Attribut-Funktionen spezifiziert werden.

Eingabeparameter:

intin(0)	Anfangswinkel (0-3600 = 1/10tel Grad)
intin(1)	Endwinkel (0-3600 = 1/10tel Grad)
ptsin(0)	x-Ordinate Mittelpunkt
ptsin(1)	y-Ordinate Mittelpunkt
ptsin(2)	x-Radius
ptsin(3)	y-Radius

Ausgabeparameter:

Standard

Zugriff in C

```

int handle;
int x;(ptsin 0)
int y;(ptsin 1)
int xradius;(ptsin 2)
int yradius;(ptsin 3)
int begang;(intin 0), Anfangswinkel
int endang;(intin 1), Endwinkel
v_ellarc(handle, x, y, xradius, yradius, begang, endang);

```

ELLIPTICAL PIE

OP: 11, FK: 7

Zeichnet einen dem Anfangs- und Endwinkel entsprechenden elliptischen, ausgefüllten Ausschnitt unter Berücksichtigung der Zeichen-Attribute.

Eingabeparameter:

```

intin(0)    Anfangswinkel (0-3600 = 1/10tel Grad)
intin(1)    Endwinkel   (0-3600 = 1/10tel Grad)
ptsin(0)    x-Ordinate Mittelpunkt
ptsin(1)    y-Ordinate Mittelpunkt
ptsin(2)    x-Radius
ptsin(3)    y-Radius

```

Ausgabeparameter:

Standard

Zugriff in C

```

int handle;
int x;(ptsin 0)
int y;(ptsin 1)
int xradius;(ptsin 2)
int yradius;(ptsin 3)
int begang;(intin 0), Anfangswinkel
int endang;(intin 1), Endwinkel
v_ellipse(handle, x, y, xradius, yradius, begang, endang);

```

ELLIPSE

OP: 11, FK: 5

Die Funktion zeichnet eine Ellipse. Die Zeichenattribute sind zuvor zu bestimmen.

Eingabeparameter:

ptsin(0) x-Ordinate Mittelpunkt
 ptsin(1) y-Ordinate Mittelpunkt
 ptsin(2) x-Radius
 ptsin(3) y-Radius

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int x; (ptsin 0)
 int y; (ptsin 1)
 int xradius; (ptsin 2)
 int yradius; (ptsin 3)
 v_ellipse(handle, x, y, xradius, yradius, begang, endang);

ROUNDED RECTANGLE

OP: 11, FK: 8

Zeichnet ein Rechteck mit abgerundeten Ecken. Die Koordinaten sind die Punkte, die sich bei Verlängerung der Seitenkanten als Schnittpunkte ergeben würden.

Eingabeparameter:

ptsin(0) x-Ordinate obere linke Ecke
 ptsin(1) y-Ordinate obere linke Ecke
 ptsin(2) x-Ordinate untere rechte Ecke
 ptsin(3) y-Ordinate untere rechte Ecke

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int pxyarray[4] (ptsin)
 v_rbox(handle, pxyarray)

FILLED ROUNDEDRECTANGLE

OP: 11, FK: 9

Zeichnet ein ausgefülltes Rechteck mit abgerundeten Ecken. Die Koordinaten sind dabei die Punkte, die sich aus der Verlängerung der Schnittpunkte ergeben.

Eingabeparameter:

ptsin(0) x-Ordinate obere linke Ecke
 ptsin(1) y-Ordinate obere linke Ecke
 ptsin(2) x-Ordinate untere rechte Ecke
 ptsin(3) y-Ordinate untere rechte Ecke

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int pxyarray[4] (ptsin)
 v_rfbbox(handle, pxyarray)

JUSTIFIED GRAPHICS TEXT OP: 11, FK: 10

Ausgabe von links- und rechtsbündigen Texten durch Dehnung der Zwischenräume (Einfügen von Space). Es ist auch möglich, die einzelnen Zeichen zu dehnen. Der Text wird an intin übergeben und muß mit ASCII(0) enden.

Eingabeparameter:

intin(0) Wortdehnung 0=aus 1=ein
 intin(1) Zeichendehnung 0=aus 1=ein
 intin(2-n) enthält den String
 ptsin(0) x-Ordinate Textausrichtung
 ptsin(1) y-Ordinate Textausrichtung
 ptsin(2) Soll-Textlänge in x-Richtung

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int x; (ptsin 0)
 int y; (ptsin 1)
 int lenght; (ptsin 2)
 int word_space; (intin 0)
 int char_space; (intin 1)
 int string[n]; (intin 2..n)
 v_justified(handle, x, y, string, lenght, word_space, char_space);

Die Attribut-Funktionen

SET WRITING MODE OP: 32

Grafikausgabe-Modus setzen. Der Modus steht in `intin(0)`:

- `replace` (Normal, alle Punkte werden gesetzt)
- `transparent` (setzt alle Punkte außer solche, an die Punkte ohne Farbe gesetzt werden sollen)
- `XOR` (setzt Punkte, wo keine sind und löscht Punkte, die mit der neuen Grafik übereinstimmen).
- `reverse transparent` (setzt alle sich überschneidenden Punkte einer Farbe)

Eingabeparameter:

`intin(0)` Zeichenmodus

Ausgabeparameter:

`intout(0)` Rückgabe des ausgewählten Modus

Zugriff in C

```
int handle;
int mode;
int set_mode;
set_mode = vswr_mode(handle, mode);
```

Bemerkungen:

Wahrheitswerte der Zeichenmodi:

- Replace:** Punktfarbe = (Farbmaske Objekt) AND (Grafikobjekt)
- Transparent:** Punktfarbe = ((Farbmaske Objekt) AND (Grafikobjekt)) OR ((schon gesetzte Farbe) AND NOT (Grafikobjekt))
- XOR-Modus:** Punktfarbe = (Grafikobjekt) XOR (schon gesetzte Farbe)
- Reverse-Transparent:** Punktfarbe = ((schon gesetzte Farbe) AND (Grafikobjekt)) OR ((Farbmaske Objekt) AND NOT (Grafikobjekt))

SET COLORREPRESENTATION OP: 14

Mischen der Farben aus den Grundfarben Rot, Grün und Blau durch Festlegung der Intensität. Die Funktion ist nur bei Farbmonitoren sinnvoll.

Eingabeparameter:

intin(0)	Farbindex 0..255
intin(1)	Rot-Anteil (0-1000)
intin(2)	Grün-Anteil (0-1000)
intin(3)	Blau-Anteil (0-1000)

Ausgabeparameter:

Standard

Zugriff in C

```
int handle;  
int index;(intin 0)  
int rgb_in[3];(intin 1..3)  
vs_color(handle, index, rgb_in)
```

Bemerkungen:

Die verfügbaren Farbmöglichkeiten werden beim Öffnen des Gerätes ermittelt. Siehe "Open Virtual Screen Workstation".

SET POLYLINE COLOR INDEX OP: 17

Auswahl einer Farbe aus der gemischten Farbtabelle für das Zeichnen mit grafischen Grundfunktionen. Bei Monochrom-Monitoren ist nur 0 und 1 möglich - diese Farben müssen nicht gemischt werden.

Eingabeparameter:

intin(0)	Linienfarbe
----------	-------------

Ausgabeparameter:

intout(0)	gewählte Linienfarbe
-----------	----------------------

Zugriff in C

```
int handle;  
int color_index;  
int set_color;  
set_color = vs1_color(handle, color_index)
```

SET POLYLINE LINE TYPE	OP: 15
------------------------	--------

Festlegen eines Linientyps für Polyline:

- | | | |
|-----------------------|--|--|
| 1 = durchgehend | | |
| 2 = unterbrochen | | |
| 3 = Punkte | | |
| 4 = Strichpunkt | | |
| 5 = gestrichelt (1/2) | | |
| 6 = Strichpunkt-Punkt | | |
| 7 = User-definiert | | mit SET-USER DEFINED LINE STYLE PATTERN. |

Eingabeparameter:

intin(0= Linienstil

Ausgabeparameter:

intout(0) gewählter Linienstil

Zugriff in C int handle;
 int style;
 int set_type
 settype=vsl_type(handle, style);

SET USER-DEFINED LINE STYLE PATTERN	OP: 113
-------------------------------------	---------

Legt das Aussehen für die Linie 7 in der Polyline-Funktion fest. siehe auch SET POLYLINE LINE TYPE.

Eingabeparameter:

intin(0) Liniengestaltung als 16-Bit-Wort

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int pattern;
 vsl_udsty (handle, pattern);

SET POLYLINE LINE WIDTH OP: 16

Definition der Linienbreite mit ungeraden Zahlen größer 2.

Eingabeparameter:

ptsin(0) Linienbreite
ptsin(1) Enthält 0 (in Assembler zwingend)

Ausgabeparameter:

ptsout(0) gewählte Linienbreite
ptsout(1) Enthält 0

Zugriff in C int handle;
 int width;(ptsin 0)
 int set_width;(ptsout 0)
 set_width = vsl_width(handle, width);

SET POLYLINE END STYLES OP: 108

Beendet Linien der Polyline-Funktion durch bestimmte Formen:

1 = normal (einfache Kanten)
2 = pfeilförmig
3 = runde Ecken

Eingabeparameter:

intin(0) Stil des Linienanfangs
intin(1) Stil des Linienendes

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int beg_style;(intin 0)
 int end_style;(intin 1)
 vsl_ends(handle, beg_style, end_style)

SET POLYMARKER TYPE

OP: 18

Setzt Markierungsart für Polymarker-Funktion:

Eingabeparameter:

intin(0) Markierungssymbol
 1 = Punkt
 2 = Pluszeichen
 3 = Stern (Standard, wenn Auswahl >7, also ungültig)
 4 = Quadrat
 5 = Diagonal-Kreuz
 6 = Diamant

Ausgabeparameter:

intout(0) gewähltes Markierungssymbol

Zugriff in C int handle;
 int symbol;
 int set_type;
 set_type = vsm_type(handle, symbol);

SET POLYMARKER HEIGHT

OP: 19

Bestimmt die Höhe der Polymarker-Symbole.

Eingabeparameter:

ptsin(0) Enthält 0 (in Assembler zwingend)
ptsin(1) Markierungshöhe (falls ungültig, nächste kleinere!)

Ausgabeparameter:

ptsout(0) erhält Markierungsbreite
ptsout(1) erhält Markierungshöhe

Zugriff in C int handle;
 int height;
 int set_handle;
 set_height = vsm_height(handle, height);

Bemerkungen:

Die C-Funktion erhält keine Markierungsbreite.

SET POLYMARKER COLOR INDEX

OP: 20

Setzt die Farbe der mit Polymarker vorgenommenen Markierungen.

Eingabeparameter:

intin(0) Farbindex

Ausgabeparameter:

intout(0) gewählter Farbindex

Zugriff in C int handle;
 int color_index;
 int set_color;
 set_color=vsm_color(handle, color_index)

SET CHARACTER HEIGHT, ABSOLUTE MODE OP: 12

Legt die Größe (Character-Höhe) des Grafiktextes fest. Die Größen ergeben sich aus einem gedachten Rechteck mit der Grundlinie=Textboden, Seitenlinien=Textanfang und -ende sowie Dachlinie=oberes Ende der Großbuchstaben.

Eingabeparameter:

ptsin(0) Enthält 0 (in Assembler zwingend)
 ptsin(1) Zeichenhöhe

Ausgabeparameter:

ptsout(0) Zeichenbreite
 ptsout(1) gewählte Zeichenhöhe
 ptsout(2) Breite der gedachten Box
 ptsout(3) Höhe der gedachten Box

Zugriff in C int handle;
 int height;(ptsin 1)
 int char_width;(ptsout 0)
 int char_height;(ptsout 1)
 int cell_width;(ptsout 2)
 int cell_height;(ptsout 3)
 vst_height(handle, height, &char_width, &char_height, &cell_width,
 &cell_height)

SET CHARACTER CELL HEIGHT, POINT MODE OP: 107

Setzt die Höhe einer Box, welche der äußeren Größe eines Zeichens entspricht, in Schritten von 1/72 Inch (engl. Maßeinheit: 1 Inch = 2.54 cm).

Eingabeparameter:

intin(0) Zeichenboxhöhe (Zeilenabstand)

Ausgabeparameter:

intout(0) gesetzte Zeichenboxhöhe
ptsout(0) Zeichenbreite
ptsout(1) Zeichenhöhe
ptsout(2) Aktuelle Box-Breite
ptsout(3) Aktuelle Box-Höhe

Zugriff in C int handle;
 int point;
 int set_point; (intout)
 int char_width; (ptsout 0)
 int char_height; (ptsout 1)
 int cell_width; (ptsout 2)
 int cell_height; (ptsout 3)
 set_point=vst_point(handle, point, &char_width, &char_height,
 &cell_width, &cell_height)

Bemerkungen:

In Proportionalchrift wird das größte Zeichen als Bemessungsgrundlage genommen.

SET CHARACTER BASELINE VECTOR OP: 13

Dreht die Character-Grundlinie in Schritten zu 1/10tel Grad (ermöglicht Schrift in alle Richtungen). Null Grad ist dabei rechts, die Drehung erfolgt entgegen dem Uhrzeiger.

Eingabeparameter:

intout(0) Drehwinkel (0-3600)

Ausgabeparameter:

intout(0) gewählter Drehwinkel

Zugriff in C `int handle;`
 `int angle;`
 `int set_baseline;`
 `set_baseline=vstvst_rotation(handle, angle)`

Bemerkungen:

Es sind nur Winkel in 90 Grad-Schritten möglich.

SET TEXT FACE	OP: 21
---------------	--------

Auswahl eines Zeichensatzes, der mit "Load Fonts" geladen worden ist.

Eingabeparameter:

`intin(0)` Zeichensatz

Ausgabeparameter:

`intout(0)` gewählter Zeichensatz

Zugriff in C `int handle;`
 `int font;`
 `int set_font;`
 `set_font=vst_font(handle, font)`

Bemerkungen:

Für Atari ST existiert kein Gerätetreiber, der mehrere Zeichensätze verfügbar hat. Die Funktion ist daher nicht sinnvoll einzusetzen. Zeichensatz 1 ist Standard.

SET GRAPHIC TEXT COLOR INDEX	OP: 21
------------------------------	--------

Auswahl der Textfarbe aus den gemischten Farben.

Eingabeparameter:

`intin(0)` Farbindex

Ausgabeparameter:

`intout(0)` gesetzter Farbindex

Zugriff in C `int handle;`
 `int color_index`
 `int set_color`
 `set_color=vst_color(handle,color_index)`

SET GRAPHIC TEXT SPECIAL EFFECTS OP: 106

Auswahl des Schrifttyps.

Eingabeparameter:

`intin(0)` Schrifttyp in den sechs niederwertigen Bits.

Bit	Wert	Schriftart
0	1	fett
1	2	hell
2	4	italic
3	8	Unterstreichen
4	16	Breitschrift
5	32	schattiert

Ausgabeparameter:

`intout(0)` gewählter Schrifttyp

Zugriff in C `int handle;`
 `int effect;`
 `int set_effect;`
 `set_effect=vst_effects(handle, effect);`

Bemerkungen:

Ist die Funktion auf dem Ausgabegerät nicht verfügbar, ist `set_effect` 0 (kein besonderer Schrifttyp gesetzt).

SET GRAPHIC TEXT ALIGNMENT OP: 39

Ausrichten des mit der Justify-Funktion ausgegebenen Textes horizontal:

Eingabeparameter:

`intin(0)` Horizontale Ausrichtung
 0 = links (linksbündig)

1 = zentriert (mittelständig)
 2 = rechts (rechtsbündig)

intin(1) Vertikale Ausrichtung
 0 = baseline (Untergrenze der Zeichen)
 1 = halfline (Obergrenze der Kleinbuchstaben)
 2 = ascent line (Obergrenze der Zeichen)
 3 = bottom line (an der Untergrenze der gedachten Box*)
 4 = descent line (Untergrenze der Unterlängen)
 5 = top line (an der Obergrenze der gedachten Box*)
 (*) siehe Justified Graphics Text.

intout(0) gesetzte horizontale Ausrichtung

intout(1=) gesetzte vertikale Ausrichtung

Zugriff: in C

```
int handle;
int hor_in;(intin 0)
int hor_out;(intout 0)
int vert_in;(intin 1)
int vert_out;(intout 1)
vst_alignment(handle, hor_in, vert_in, &hor_out, vert_out);
```

SET FILL INTERIOR STYLE	OP: 23
-------------------------	--------

Auswahl der Füllfunktion. Im Anhang finden Sie eine Tabelle mit allen möglichen Fülltypen.

0 = nicht ausfüllen
 1 = ganz ausfüllen (mit der Füllfarbe)
 2 = punktiert (siehe Set Fill Style Index, Punkte=Füllfarbe)
 3 = schraffiert (siehe Set Fill Style Index, Linien=Füllfarbe)
 4 = user-definiert (siehe Set User Defined Fill Pattern)

Eingabeparameter:

intin(0) Füllstil (0-4)
intout(0) gewählter Füllstil

Zugriff in C

```
int handle;
int style;
int set_interior;
set_interior=vsf_interior(handle, style);
```

SET FILL STYLEINDEX**OP: 24**

Füllmuster auswählen, abhängig von der mit Set Fill Interior Style eingestellten Füllart (intin 0 = 2 oder 3).

Eingabeparameter:

intin(0) Füllmuster (1-24 punktiert, 1-12 schraffiert)

Ausgabeparameter:

intout(0) gewähltes Füllmuster

Zugriff in C int handle;
 int style_index;
 int set_style;
 set_style=vsf_style(handle, style_index)

SET FILL COLORINDEX**OP: 25**

Setzt die Farbe für das Füllmuster entsprechend der mit Set Color Representation festgelegten Mischfarben (bei Monochrom-Monitoren ist nur 0 (schwarz) und 1 (weiß) möglich.)

Eingabeparameter:

intin(0) Füllfarbe

Ausgabeparameter:

intout(0) gewählte Füllfarbe

Zugriff in C int handle;
 int color_index;
 int set_color;
 set_color=vsf_color(handle, color_index);

SET FILL PERIMETER VISIBILITY**OP: 104**

Füllflächenumrahmung-Sichtbarkeit festlegen.

Eingabeparameter:

intin(0) Sichtbarkeit 0=ausschalten 1=einschalten

Ausgabeparameter:

intout(0) gewählte Sichtbarkeit

Zugriff in C int handle;
 int per_vis;
 int set_perimeter;
 set_perimeter=vsf_perimeter(handle, per_vis)

SET USER DEFINED FILL PATTERN OP: 112

Benutzereigenes Füllmuster definieren. Für das Füllmuster werden 16 2-Byte-Worte benötigt, wobei jedes Wort eine Zeile des Musters bestimmt. Für mehrfarbige Muster müssen mehrere Muster-Arrays hintereinander definiert werden.

contrl(3) Anzahl der Muster-Worte (Farben*16)

Eingabeparameter:

intin(0) bis
 intin(15) 16-Wort-Gruppe für erste Füllfarbe (Muster)
 intin(16) bis
 intin(31) 16-Wort-Gruppe für zweite Füllfarbe (Muster)
 u.s.w.

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int planes; /* Füllfarben-Anzahl */(Länge(intin)+1)/16
 int pfill_pat[planes*16] (Länge(intin)+1)
 vsf_updat(handle, pfill_pat, planes);

Raster-Operationen

COPY RASTER,OPAQUE OP: 109

Logische Verknüpfung zwischen Quell- und Zielraster. Besitzen Quell- und Zielraster unterschiedliche Größen, so wird in der Größe des Quellrasters verknüpft. Verknüpfungsmodi sind (QR = Quellraster, ZR = Zielraster, R = Verknüpfungsergebnis):

- 0 $R = 0$
- 1 $R = QR \text{ AND } ZR$
- 2 $R = QR \text{ AND } (\text{NOT } ZR)$
- 3 $R = QR$
- 4 $R = (\text{NOT } QR) \text{ AND } ZR$
- 5 $R = ZR$
- 6 $R = QR \text{ XOR } ZR$
- 7 $R = QR \text{ OR } ZR$
- 8 $R = \text{NOT } (QR \text{ OR } ZR)$
- 9 $R = \text{NOT } (QR \text{ XOR } ZR)$
- 10 $R = \text{NOT } ZR$
- 11 $R = QR \text{ OR } (\text{NOT } ZR)$
- 12 $R = \text{NOT } QR$
- 13 $R = (\text{NOT } QR) \text{ OR } ZR$
- 14 $R = \text{NOT } (QR \text{ AND } ZR)$
- 15 $R = 1$

Eingabeparameter:

- contrl(7-8) Adresse des Quellrasters (MFBD)
- contrl(9-10) Adresse des Zielrasters (MFDB)
- intin(0) Verknüpfungsmodus
- ptsin(0) X-Ordinate linke obere Ecke Quellraster
- ptsin(1) Y-Ordinate linke obere Ecke Quellraster
- ptsin(2) X-Ordinate rechte untere Ecke Quellraster
- ptsin(3) Y-Ordinate rechte untere Ecke Quellraster
- ptsin(4) X-Ordinate linke obere Ecke Zielraster
- ptsin(5) Y-Ordinate linke obere Ecke Zielraster
- ptsin(6) X-Ordinate rechte untere Ecke Zielraster
- ptsin(7) Y-Ordinate rechte untere Ecke Zielraster

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int wr_mode; (intin)
 int pxyarray[8] (ptsin)
 int *psrcMFDB; (contrl 7-8)
 int *pdesMFDB; (contrl 9-10)
 vro_cpyform(handle, wr_mode, pxyarray, psrcMFDB, pdesMFDB);

Bemerkungen:

In Maschinensprache muß ein MFDB aufgebaut werden, während dies in C automatisch geschieht.

COPY RASTER, TRANSPARENT

OP: 121

Verknüpfung eines einfarbigen mit einem mehrfarbigen Raster.

- 1 replace (Normal, alle Punkte werden gesetzt)
- 2 transparent (setzt alle Punkte außer solche, an die Punkte ohne Farbe gesetzt werden sollen)
- 3 XOR (setzt Punkte, wo keine sind und löscht Punkte, die mit der neuen Grafik übereinstimmen).
- 4 reverse transparent (setzt alle sich überschneidenden Punkte einer Farbe)

Eingabeparameter:

- contrl(7-8) Adresse des Quellrasters (MFBD)
- contrl(9-10) Adresse des Zielrasters (MFDB)
- intin(0) Zeichenmodus (-> SET WRITE MODE + s.o.)
- intin(1) Farbindex gesetzte Punkte
- intin(2) Farbindex nicht gesetzte Punkte
- ptsin(0) X-Ordinate linke obere Ecke Quellraster
- ptsin(1) Y-Ordinate linke obere Ecke Quellraster
- ptsin(2) X-Ordinate rechte untere Ecke Quellraster
- ptsin(3) Y-Ordinate rechte untere Ecke Quellraster
- ptsin(4) X-Ordinate linke obere Ecke Zielraster
- ptsin(5) Y-Ordinate linke obere Ecke Zielraster
- ptsin(6) X-Ordinate rechte untere Ecke Zielraster
- ptsin(7) Y-Ordinate rechte untere Ecke Zielraster

Ausgabeparameter:

Standard

Zugriff in C

```
int handle;
int wr_mode; (intin 0)
int color_index[2]; (intin 1..2)
int pxyarray[8] (ptsin)
int *psrcMFDB; (contrl 7-8)
int *pdesMFDB; (contrl 9-10)
vr_trnfm()
vro_cpyfm(handle, wr_mode, pxyarray, psrcMFDB, pdesMFDB);
```

Bemerkungen:

In Maschinensprache muß ein MFDB aufgebaut werden, während dies in C automatisch geschieht.

TRANSFORM FORM OP: 110

Umwandlung von Rasterformaten in geräteeigene und umgekehrt. Aufruf u.U. vor vro_cpyfm notwendig.

Eingabeparameter:

contrl(7-8) Adresse des Quellrasters (MFBD)
contrl(9-10) Adresse des Zielrasters (MFDB)

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int *psrcMFDB; (contrl 7-8)
 int *pdesMFDB; (contrl 9-10)
 vs_trnfm(handle, psrcMFDB, pdesMFDB);

Bemerkungen:

In Maschinensprache muß ein MFDB aufgebaut werden, während dies in C automatisch geschieht.

GET PIXEL OP: 105

Stellt Zustand (gesetzt/nicht gesetzt) und Farbe eines Punktes auf dem Bildschirm fest.

Eingabeparameter:

ptsin(0) x-Ordinate des Pixels
ptsin(1) y-Ordinate des Pixels

Ausgabeparameter:

intout(0) Zustand 0=nicht gesetzt 1=gesetzt
intout(1) Pixelfarbe

Zugriff in C int handle;
 int x; (ptsin 0)

```

int y;(ptsin 1)
int *pel;(intout 0)
int *index;(intout 1)
v_get_pixel(handle, x, y, pel, index);

```

Eingabe-Funktionen

SET INPUT MODE OP: 33

Setzt den Modus der logischen Eingabeeinheiten. Diese sind:

- 1 = Positions-Eingaben (Maus oder Grafik-Cursor)
- 2 = Wert-Eingaben (Cursor-Tasten "nach oben" und "nach unten")
- 3 = Logische Auswahl (ST-Funktionstasten F1 bis F10)
- 4 = String-Eingabe (Tastatur)

Die logische Eingabeeinheiten arbeiten in folgenden Modi:

- 1 = request mode (Nachfrage-Modus; es wird auf ein Eingabeereignis gewartet)
- 2 = sample mode (Erfrage-Modus; es wird der Zustand des Eingabegeräts ermittelt, aber nicht auf eine Eingabe gewartet)

Eingabeparameter:

intin(0) Eingabegerät (1..4)
intin(1) Eingabemodus (1..2)

Ausgabeparameter:

intout(0) gewählter Eingabemodus

Zugriff in C int handle;
 int set_mode;(intout 0)
 int dev_type;(intin 0)
 int mode;(intin 1)
 set_mode=vsin_mode(handle, dev_type, mode);

INPUT LOCATOR, REQUEST MODE	OP: 28
-----------------------------	--------

Ausgabe der Grafik-Cursor-Position nach einem Tastendruck. Der Cursor wird während des Aufrufs dieser Funktion unabhängig von seinem sonstigen Zustand sichtbar.

Eingabeparameter:

ptsin(0) x-Ordinate Initialisierung Grafik-Cursor
ptsin(1) y-Ordinate Initialisierung Grafik-Cursor

Ausgabeparameter:

intout(0) ASCII-Code der Taste, welche die Funktion beendet
ptsout(0) festgestellte x-Ordinate vom Grafik-Cursor
ptsout(1) festgestellte y-Ordinate vom Grafik-Cursor

Zugriff in C

```
int handle;
int x;(ptsin 0)
int y;(ptsin 1)
int xout;(ptsout 0)
int yout;(ptsout 1)
int term;(intout 0)
set_mode=vsin_mode(handle, 1, 1);
vrq_locator(handle, x, y, &xout, &yout, &term);
```

INPUT LOCATOR, SAMPLE MODE	OP: 28
----------------------------	--------

Ausgabe der Grafik-Cursor-Position, ohne daß eine Taste gedrückt werden mußte und Rückgabe des ASCII-Wertes einer evtl. gedrückten Taste.

Eingabeparameter:

ptsin(0) x-Ordinate (Initialisierung) Grafikkursor
ptsin(1) y-Ordinate (Initialisierung) Grafikkursor

Ausgabeparameter:

ptsout(0) x-Ordinate der festgestellten Grafik-Cursor-Position
ptsout(1) y-Ordinate der festgestellten Grafik-Cursor-Position
contrl(2) Koordinaten-Status (0=nicht geändert, 1=geändert)
contrl(4) Tastatur-Status (0=nicht geändert, 1=geändert)
intout(0) ASCII-Wert einer evtl. gedrückten Taste (Low-Byte)

Zugriff in C

```

int handle;
/*status = contrl(2) | (contrl(4) <<1) */
int x;(ptsin 0)
int y;(ptsin 1)
int xout;(ptsout 0)
int yout;(ptsout 1)
int term;(intout 0)
set_mode=vsin_mode(handle, 1, 2);
status=vsm_locator(handle, x, y, &xout, &yout, &term);
status : contrl(4) OR contrl(2) (Bitweise)

```

INPUT VALULATOR, REQUEST MODE OP: 29

Rückgabe der betätigten Taste oder Tastenkombination zur Wertveränderung. Der Rückgabewert liegt zwischen 0 und 100. Dabei gilt:

↑	= Wert n+10,
↓	= Wert n-10,
Shift + ↑	= Wert n+1,
Shift + ↓	= Wert n-1.

Eingabeparameter:

intin(0) Wert-Initialisierung

Ausgabeparameter:

intout(0) Wert-Rückgabe
 intout(1) gedrückte Taste als ASCII

Zugriff in C

```

int handle;
int valulator_in;(intin)
int valulator_out;(intout 0)
int terminator;(intout 1)
set_mode=vsin_mode(handle, 2, 1);
vrq_valulator(handle, valulator_in, &valulator_out, &terminator);

```

INSERT VALULATOR, SAMPLE MODE OP: 29

Stellt fest, ob und welche Aktion die Eingabeeinheit beeinflusst hat. Der Rückgabewert liegt zwischen 0 und 100. Dabei gilt:

↑	= Wert n+10,
↓	= Wert n-10,

+ = Wert n+1,
 + = Wert n-1.

Eingabeparameter:

intin(0) Wert-Initialisierung

Ausgabeparameter:

intout(0) ermittelter Wert
 intout(1) ASCII-Code der gedrückten Taste/Kombination.
 contrl(4) 0 (keine Aktion)
 1 (Wertveränderung)
 2 (Taste gedrückt)

Zugriff in C

```

int handle;
status;
int val_in (intin 0)
int val_out;(intout 0)
int term;(intout 1)
set_mode=vsin_mode(handle, 2, 2);
vsm_valulator(handle, val_in, &val_out, &term, &status);
  
```

INPUT CHOICE, REQUEST MODE	OP: 30
----------------------------	--------

Wartet auf Drücken einer Funktionstaste und gibt den Wert zurück. Bei ungültigen Eingaben wird der ASCII-Code der falschen Eingabetaste zurückgegeben.

Eingabeparameter:

intin(0) Auswahlnr.-Initialisierung (F1-F10)

Ausgabeparameter:

intout(0) Gedrückte Funktionstaste (1..10)

Zugriff in C

```

int handle;
int ch_in;
int *ch_out;
set_mode=vsin_mode(handle, 3, 1);
vrq_choice(handle, ch_in, ch_out);
  
```

INPUT CHOICE, SAMPLE MODE

OP: 30

Gibt den Wert der zuletzt gedrückten Funktionstaste bzw. den ASCII-Code einer falschen gedrückten Taste aus.

Eingabeparameter:

Standard

Ausgabeparameter:

intout(0) gedrückte Funktionstaste (1..10) (bzw. ASCII ungültige Taste)
 contrl(4) status
 0 (kein Tastendruck)
 1 (Tastendruck)

Zugriff in C int handle;
 int status;(s.u.)
 int choice;(Ausgabeparameter)
 set_mode=vsin_mode(handle, 3, 2);
 status=vsm_choice(handle, &choice);

INPUT STRING, REQUEST MODE

OP: 31

Liest Zeichen(-ketten) von der Tastatur und ermöglicht die gleichzeitige Bildschirmausgabe. Die Eingabe gilt als beendet, wenn die maximale (spezifizierte) String-Länge erreicht oder Return gedrückt wurde.

Eingabeparameter:

intin(0) max. String-Länge
 intin(1) Bildschirmdarstellung 0=nein 1=ja
 ptsin(0) x-Ordinate obere linke Ecke Bildschirmdarstellung
 ptsin(1) y-Ordinate obere linke Ecke Bildschirmdarstellung

Ausgabeparameter:

intout(0-n) erhält den String in den niederwertigen Bytes.

Zugriff in C int handle;
 int max_lenght;(intin 0)
 int echo_mode;(intin 1)
 int echo_xy[2];(ptsin 0..1)
 char string[max_lenght]; (intout)
 set_mode=vsin_mode(handle, 4, 1);
 vrq_string(handle, max_lenght, echo_mode, echo_xy, &string);

Bemerkungen:

Wenn `max_lenght < 0`, wird der 2-Byte-Code (VDI-Standard-Tastatur) benutzt.

INPUT STRING, SAMPLE MODE	OP: 31
---------------------------	--------

Liest Zeichen(-ketten) von der Tastatur und ermöglicht die gleichzeitige Bildschirmausgabe. Die Eingabe gilt als beendet, wenn die maximale (spezifizierte) String-Länge erreicht oder `Return` gedrückt wurde.

Eingabeparameter:

<code>intin(0)</code>	max. String-Länge
<code>intin(1)</code>	Bildschirmdarstellung 0=nein 1=ja
<code>ptsin(0)</code>	x-Ordinate obere linke Ecke Bildschirmdarstellung
<code>ptsin(1)</code>	y-Ordinate obere linke Ecke Bildschirmdarstellung

Ausgabeparameter:

<code>intout(0-n)</code>	erhält den String in den niederwertigen Bytes.
<code>contrl(4)</code>	(status) 0 (ungültiges Zeichen gesendet, Funktion abgebrochen) >1 (String-Länge der gültigen Eingabe)

Zugriff in C

```
int handle;
int max_lenght; (intin 0)
int echo_mode; (intin 1)
int echo_xy[2]; (ptsin 0..1)
char string[max_lenght]; (intout)
set_mode=vsin_mode(handle, 4, 1);
status=vrq_string(handle,max_lenght,echo_mode,echo_xy, &string);
```

Bemerkungen:

Wenn `max_lenght < 0`, wird der 2-Byte-Code (VDI-Standard-Tastatur) benutzt.

SET MOUSE FORM	OP: 111
----------------	---------

Definiert das Aussehen des Mauszeigers in einem 16x16-Raster.

Eingabeparameter:

<code>intin(0)</code>	x-Ordinate des Aktionspixels (z.B. Pfeilspitze)
<code>intin(1)</code>	y-Ordinate des Aktionspixels (jeweils relativ zur oberen linken Ecke des Cursor-Rasters)
<code>intin(2)</code>	reserviert, muß 1 sein

intin(3) Farbindex der Cursor-Maske
intin(4) Farbindex der Cursorform
intin(5) bis
intin(20) Maske (16 Wort-Gruppe)
intin(21) bis
intin(36) Form (16-Wort-Gruppe, Punkte in Vordergrundfarbe=1)

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int pcur_form[37];(intin)
 vsc_form(handle, pcur_form);

EXCHANGE TIMERINTERRUPT VECTOR	OP: 118
--------------------------------	---------

Setzt eine Benutzer-Routinen-Adresse auf den Systemtimer-Interrupt.

Eingabeparameter:

contrl(7) bis
contrl(8) 32-Bit-Adresse, neuer Interrupt-Vektor

Ausgabeparameter:

contrl(9) bis
contrl(10) 32-Bit-Adresse, alter Interrupt-Vektor
intout(0) Abstände zwischen den Interrupts in 1/1000 Sekunden.

Zugriff in C int handle;
 int *tim_addr; (contrl 7..8)
 int *otim_addr; (contrl 9..10)
 int tim_conv; (intout)
 vex_time(handle, tim_addr, otim_addr, &tim_conv);

SHOW CURSOR	OP: 122
-------------	---------

Macht den Grafikcursor auf dem Bildschirm sichtbar. Wenn das Flag in intin(0) nicht ausgeschaltet wird, ist zum Sichtbarmachen des Grafikcursors die gleiche Anzahl Show Cursor-Aufrufe erforderlich, wie zuvor Hide Cursor aufgerufen wurde.

Eingabeparameter:

intin(0) Flag (0 = Hide Cursor Aufrufanzahl wird ignoriert,
1 = Für jeden Hide Cursor ist ein Show Cursor-Aufruf
erforderlich).

Ausgabeparameter:

Standard

Zugriff in C int handle;
 int reset;
 v_show_c(handle, reser);

HIDE CURSOR OP: 123

Macht den Grafikcursor auf dem Bildschirm unsichtbar; sichtbar machen mit Show Cursor.

Eingabeparameter:

Standard

Ausgabeparameter:

Standard

Zugriff in C int handle;
 v_hide_c(handle);

SAMPLE MOUSE BUTTON STATE OP: 124
--

Gibt die Position des Grafikcursors und den Status der Maustasten aus.

Eingabeparameter:

Standard

Ausgabeparameter:

intout(0) Tastenstatus
 0 = nicht gedrückt
 1 = Taste gedrückt)
ptsout(0) x-Ordinate des Grafikcursors
 1 = y-Ordinate des Grafikcursors

Zugriff in C

```
int handle;  
int pstatus; (intout)  
int x; (ptsout 0)  
int y; (ptsout 1)  
vq_mouse(handle, &pstatus, &x, &y);
```

EXCHANGE BUTTON CHANGE VECTOR	OP: 125
-------------------------------	---------

Bewirkt, daß bei Betätigung der Maustaste zu einer definierten Routine gesprungen wird.

Eingabeparameter:

contrl(7) bis

contrl(8) 32-Bit-Adresse, neue Maustastenreaktionsadresse

Ausgabeparameter:

contrl(9) bis

contrl(10) 32-Bit-Adresse (alte Adresse)

Zugriff in C

```
int handle;  
int *pusrcode; (Eingabeparameter)  
int *psavcode; (Ausgabeparameter)  
vex_butv(handle, pusrcode, psavcode);
```

Bemerkungen:

Der gedrückte Mausknopf ist in einem Prozessorregister enthalten.

EXCHANGE MOUSEMOVEMENT VECTOR	OP: 126
-------------------------------	---------

Maus-Bewegungs-Handling: bei Bewegung der Maus wird zu einer definierten Routine gesprungen und die neue Cursor-Position berechnet, ohne die alte zu aktualisieren.

Eingabeparameter:

contrl(7) bis

contrl(8) 32-Bit-Adresse, neue Mausbewegungsreaktionsadresse

Ausgabeparameter:

contrl(9) bis

contrl(10) 32-Bit-Adresse (alte Adresse)

Zugriff in C `int handle;`
 `int *pusrcode; (Eingabeparameter)`
 `int *psavcode; (Ausgabeparameter)`
 `vex_motv(handle, pusrcode, psavcode);`

Bemerkungen:

Die neuen Koordinaten des Grafikcursors sind in einem Prozessorregisterpaar enthalten.

EXCHANGE CURSOR CHANGE VECTOR	OP: 127
-------------------------------	---------

Bewirkt, daß bei Bewegung des Grafikcursors zu einer definierten Routine gesprungen wird.

Eingabeparameter:

contrl(7) bis
 contrl(8) 32-Bit-Adresse, neue Cursor-Veränderungs-Reaktionsadresse

Ausgabeparameter:

contrl(9) bis
 contrl(10) 32-Bit-Adresse (alte Adresse) sichern

Zugriff in C `int handle;`
 `int *pusrcode; (Eingabeparameter)`
 `int *psavcode; (Ausgabeparameter)`
 `vex_curv(handle, pusrcode, psavcode);`

Bemerkungen:

Die neuen Koordinaten des Grafikcursors sind in einem Prozessorregisterpaar enthalten.

SAMPLE KEYBOARD KEY INFORMATION	OP: 128
---------------------------------	---------

Tastaturstatus der Sondertasten ermitteln.

Eingabeparameter:

Standard

Ausgabeparameter:

intout(0) Status (0..15)

Bit	Wert	Taste
0	1	Shift rechts
1	2	Shift links
2	4	Ctrl
3	8	Alternate

Zugriff in C `int handle;`
 `int pstatus;`
 `vq_key_s(handle, &pstatus);`

Nachfrage-Funktionen

EXTENDED INQUIRE FUNCTION	OP: 102
---------------------------	---------

Ermittlung der 57 Workstation-Parameter oder 19 weiterer gerätespezifischer Informationen:

Eingabeparameter:

`intin(0)` Typ der Information
 0 = Open Workstation
 1 = Extended)

Ausgabeparameter:

`intout(0)` Bildschirmart (0..4)
`intout(1)` Hintergrundfarben, Anzahl
`intout(2)` Texteffekt-Unterstützung
`intout(3)` Vergrößerung
 0 = unmöglich
 1 = möglich
`intout(4)` Farbebenen für Raster, Anzahl
`intout(5)` Unterstützung Übersichts-Table
 0 = ja
 1 = nein
`intout(6)` Rasteroperationen je Sekunde
`intout(7)` Konturen Füllen Unterstützung
 0 = nein
 1 = ja

intout(8)	Text Rotation Unterstützung 0 = nein 1 = 90-Grad-Schritte 2 = beliebige Winkel zur Basis
intout(9)	Zeichenmodi, Anzahl
intout(10)	Verfügbare Eingabe-Modi 0 = keiner 1 = request 2 = sample
intout(11)	Textausrichtung Unterstützung 0 = nein 1 = ja
intout(12)	Farbwechsel Plotter 0 = möglich 1 = keine Einrichtung
intout(13)	Farbwechsel Matrixdrucker 0 = möglich 1 = keine Einrichtung
intout(14=	Verfügbare Punktzahl in Zeichenfunktion -1 = unbegrenzt
intout(15)	Intin-Array, max. Länge -1 = unbegrenzt
intout(16)	Maustasten, Anzahl
intout(17)	verbreiterte Linien, Unterstützung 0 = nein 1 = ja
intout(18)	Zeichenmodi verbreiterte Linien, Anzahl

Zugriff in C int handle;
 int owflag;
 int work_out[57];
 vq_extnd(handle, owflag, work_out)

INQUIRE COLOR REPRESENTATION OP: 26
--

Ermittlung der Farbmischung für den spezifizierten Index.

Eingabeparameter:

intin(0)	Farbindex
intin(1)	Index-Flag (0=übergebener 1=aktueller Farbindex)

Ausgabeparameter:

intout(0) Rot-Anteil (0-1000)
intout(1) Grün-Anteil (0-1000)
intout(2) Blau-Anteil (0-1000)

Zugriff in C int handle;
 int color_index; (intin 0)
 int set_flag; (intin 1)
 int rgb[3]; (intout 0..2)
 vq_color(handle, color_index, set_flag, rgb);

INQUIRE CURRENT POLYLINE ATTRIBUTES OP: 35

Ermittlung der Linienattribute.

Eingabeparameter:

Standard

Ausgabeparameter:

intout(0) Typ-Attribut
intout(1) Farb-Attribut
intout(2) Zeichenmodus
intout(3) Form des Linienanfangs (Kante, Pfeil, Rund)
intout(4) Form des Linienendes (Kante, Pfeil, Rund)
ptsout(0) Breite-Attribut

Zugriff in C int handle;
 int attrib[6];
 vql_attributes(handle, attrib);

INQUIRE CURRENT POLIMARKER ATTRIBUTES OP: 36

Ermittelt die definierten Markierungstypen.

Eingabeparameter:

Standard

Ausgabeparameter:

intout(0) Typ der Markierung (Punkt, Stern,...)
intout(1) Farbe der Markierung
intout(2) Zeichenmodus

ptsout(0) Breite der Markierung
ptsout(1) Höhe der Markierung

Zugriff in C int handle;
 int attrib[5];
 vqm_attributes(handle, attrib);

INQUIRE CURRENT FILL AREA ATTRIBUTES

OP: 37

Ermittelt die eingestellten Füllattribute.

Eingabeparameter:

Standard

Ausgabeparameter:

intout(0) Fülltyp
intout(1) Farbe
intout(2) Muster
intout(3) Zeichenmodus
intout(4) Umrahmungsstatus

Zugriff in C int handle;
 int attrib[5];
 vqf_attributes(handle, attrib);

INQUIRE CURRENT GRAPHIC TEXT ATTRIBUTES

OP: 38

Ermittelt die eingestellten Textattribute.

Eingabeparameter:

Standard

Ausgabeparameter:

intout(0) Zeichensatz
intout(1) Farbe des Textes
intout(2) Drehwinkel
intout(3) Justifikation horizontal
intout(4) Justifikation vertikal
intout(5) Zeichenmodus
ptsout(0) Zeichenbreite
ptsout(1) Zeichenhöhe

ptsout(2) Breite der gedachten Box (-> ...)
ptsout(3) Höhe der gedachten Box (-> ...)

Zugriff in C int handle;
 int attrib[10];
 vqt_attributes(handle, attrib);

INQUIRE TEXT EXTEND

OP: 116

Berechnet die Ausdehnung eines Strings aufgrund seiner ihn bestimmenden Attribute.

Eingabeparameter:

intin(0-n) enthält den String in den Low-Bytes

Ausgabeparameter:

ptsout(0) x-Ordinate Punkt 1 (relativ)
ptsout(1) y-Ordinate Punkt 1 (relativ)
ptsout(2) x-Ordinate Punkt 2 (relativ)
ptsout(3) y-Ordinate Punkt 2 (relativ)
ptsout(4) x-Ordinate Punkt 3 (relativ)
ptsout(5) y-Ordinate Punkt 3 (relativ)
ptsout(6) x-Ordinate Punkt 4 (relativ)
ptsout(7) y-Ordinate Punkt 4 (relativ)

Zugriff in C int handle;
 int string[]; (Eingabeparameter)
 int extend[8]; (Ausgabeparameter)
 vqu_extent(handle, string, extent);

Bemerkungen:

In C wird der String automatisch an intin übergeben.

INQUIRE CHARACTER CELL WIDTH OP: 117
--

Ermittelt die Ausmaße eines Zeichens und der Zeichenbox (Abstand des Characters zum Rand der ihn umgebenden Box).

Eingabeparameter:

intin(0) Zeichen, dessen Maße genommen werden sollen.

Ausgabeparameter:

intout(0)	gewähltes Zeichen (-1, wenn ungültig)
ptsout(0)	Boxbreite
ptsout(1)	enthält 0
ptsout(2)	Abstand

GEM-AES

Die Programmierung des AES ist sehr ähnlich wie die des VDI. Durch den Maschinenbefehl TRAP #2 wird das GEM aufgerufen, in dem Datenregister D0 muß nun aber der Wert \$C8 für die Wahl des AES liegen. Außerdem wird in D1 ein Zeiger auf eine Tabelle übergeben, in welcher die Startadressen der Parameterfelder des AES liegen. Diese Felder sind folgende:

Control:	Hier werden die Funktionsnummer sowie die Längen der anderen Felder eingetragen.
Global:	Dieses Feld enthält diverse Informationen über den Prozeß, aus dem der Funktionsaufruf stammt. Die Belegung dieses Feldes ist jedoch für den Programmierer nicht wichtig.
Intin:	Ähnlich wie beim VDI werden hierin die verschiedenen Parameter an die aufgerufene Funktion übergeben.
Intout:	In diesem Array werden Rückgabeparameter vom AES eingetragen.
Addrin:	Dieses Array besteht aus 32-Bit-Langworten. Es werden dort eventuell nötige Zeiger an das AES übergeben.
Addrout:	Aus der AES-Funktion zurückgegebene Zeiger finden sich in diesem Array.

Die Felder haben grundsätzlich festgelegte Längen. Dennoch werden sie nie alle in voller Länge benötigt, so daß man nur die notwendigen Einträge vornehmen muß. Die Anzahl der in den einzelnen Feldern eingesetzten Parameter wird jeweils im Control-Feld eingetragen.

Die nun folgenden Funktionen des GEM-AES sind mit all ihren Parametern für die Programmierung in C vorgegeben. Assembler-Programmierer müssen

jedoch dieselben Parameter in die gleichen Felder (Intin usw.) schreiben. Dabei bedeutet die C-Definition INT Wortbreite und LONG Langwortbreite des jeweiligen Parameters.

WINDOW-Behandlung

WIND_GET OP: 104

Abfrage diverser Daten über ein Fenster.

Eingabeparameter:

control (Standard 0-4) 104,2,5,0,0

intin(0) Geräte-Identifikationscode

intin(1) Art der zu übernehmenden Daten (Typ), wie folgt:

4 = Arbeitsbereich d. Fensters

5 = Gesamtgröße des Fensters

6 = Gesamtgröße vorangegangenes Fenster

7 = Größtmögliche Gesamtgröße (= Wind_Create)

8 = relative Position Schieber Horizontal (1..1000)

9 = relative Position Schieber Vertikal (1..1000)

10 = Window-Handle des aktiven Fensters

11 = Erstes Rechteck der Fenster-Rechteckliste

12 = Nächstes Rechteck der Fenster-Rechteckliste (xy-Koordinaten Ecke sowie Breite,Höhe)

13 = reserviert

14 = reserviert

15 = Größe des Schiebers Horizontal relativ zur umrandeten Kasten-größe (1-1000, -1=min. Quadrat)

16 = Größe des Schiebers Vertikal relativ zur umrandeten Kasten-größe (1-1000, -1=min. Quadrat)

Ausgabeparameter:

intout(0) Fehlerüberprüfung (0=Fehler, >0=Kein Fehler)

intout(1) Rückmeldung (Einzelwertrückgaben) oder x-Ordinate

intout(2) y-Ordinate (bei mehreren Rückgabewerten)

intout(3) Breite (" " ")

intout(4) Höhe (" " ")

Zugriff in C int wi_ghandle; (intin 0)
 int wi_gfield; (intin 1)
 int wi_greturn; (intout 0)

```
int wi_gw1, wi_gw2, wi_gw3, wi_gw4; (intout 1..4)
wi_greturn=wind_get(wi_ghandle, wi_gfield, &wi_gw1, &wi_gw2,
&wi_gw3, &wi_gw4);
```

WIND_CREATE

OP: 100

Aufbau eines Fensters einstellen. Dabei können in einem Fenster die folgenden Symbole bzw. Funktionen eingebaut werden (Übergabe in einem 12-Bit-Feld):

Bit		
0	Fenstername	(Titelzeile des Fensters)
1	Fenster schließen	(Löschfeld obere linke Ecke)
2	Volle Ausdehnung	(Full Window obere rechte Ecke)
3	Verschieben	(Schieberahmen, oberer Balken)
4	Information	(Informationsbalken)
5	Format	(Fensterformat, rechte untere Ecke)
6	Pfeil nach oben	(oben rechts, ggf. unter FULL W.)
7	Pfeil nach unten	(unten rechts, ggf. über Format)
8	vertikal schieben	(Schieber im waagerechten Rand)
9	Pfeil nach links	(linke untere Ecke)
A	Pfeil nach rechts	(Rechte untere Ecke, neben Format)
B	horizont. schieben	(Schieber im senkrechten Rand)

Eingabeparameter:

control(0-4) 100,5,1,0,0
 intin(0) Bitmuster für die Fensterkomponenten (Lowbyte)
 intin(1) x-Ordinate (größtmögliches Fenstermaß)
 intin(2) y-Ordinate (größtmögliches Fenstermaß)
 intin(3) Breite (größtmögliche)
 intin(4) Höhe (größtmögliche)

Ausgabeparameter:

intout(0) Kennziffer (wenn <0 keine Fenster verfügbar)

Zugriff in C int wind_crkind; (intin 0)
 int wi_crx; (intin 1)
 int wi_crwy; (intin 2)
 int wi_crrw; (intin 3)
 int wi_crwh; (intin 4)
 int wi_crreturn; (intout 0)
 wi_crreturn=wind_create(wi_crkind, wi_crx, wi_crwy, wi_crrw,
 wi_crwh);

WIND_OPEN	OP: 101
-----------	---------

Öffnen eines Fensters mit den angegebenen aktiven Fensterbereichen und Funktionen.

Eingabeparameter:

control(0..4) 101,5,1,0,0
intin(0) Window-Handle (Kennziffer des geöffneten Fensters)
intin(1) x-Ordinate
intin(2) y-Ordinate
intin(3) Breite
intin(4) Höhe

Ausgabeparameter:

intout(0) Fehlerüberprüfung
0 = Fehler
>0 = Kein Fehler

Zugriff in C int wi_ohandle; (intin 0)
int wi_owx; (intin 1)
int wi_owy; (intin 2)
int wi_oww; (intin 3)
int wi_owh; (intin 4)
int wi_oreturn; (intout 0)
wi_oreturn=wind_create(wi_crkind, wi_owx, wi_owy, wi_oww, wi_owh);

WIND_CLOSE	OP: 102
------------	---------

Schließen eines Fensters mit den angegebenen aktiven Fensterbereichen und Funktionen. Das Fenster ist nicht zerstört, es kann mit WIND_OPEN wieder reaktiviert werden.

Eingabeparameter:

control(0..4) 102,1,1,0,0
intin(0) Window-Handle (Kennziffer des Fensters)

Ausgabeparameter:

intout(0) Fehlerüberprüfung
0 = Fehler
>0 = Kein Fehler

Zugriff in C `int wi_clhandle; (Eingabeparameter)`
 `int wi_clreturn; (Ausgabeparameter)`
 `wi_clreturn=wind_close(wi_clhandle);`

WIND_DELETE	OP: 103
-------------	---------

Löschen eines Fensters. Der belegte Speicherplatz wird wieder freigegeben. Vor dem Löschen muß ein Fenster unbedingt geschlossen werden!

Eingabeparameter:

`control(0..4)` 103,1,1,0,0
`intin(0)` Window Handle (Kennziffer des Fensters)

Ausgabeparameter:

`intout(0)` Fehlerüberprüfung
 0 = Fehler
 >0 = Kein Fehler

Zugriff in C `int wi_dhandle; (Eingabeparameter)`
 `int wi_dreturn; (Ausgabeparameter)`
 `wi_dreturn=wind_delete(wi_dhandle);`

Bemerkungen:

Vor dem Löschen eines Fensters ist es unbedingt zu schließen!

WIND_SET	OP: 105
----------	---------

Titelzeile und Fensterrandbereiche können in WIND_CREATE gestaltet werden. Diese Funktion ermöglicht eine nachträgliche Änderung der Gestaltungs-komponenten.

Eingabeparameter:

`control(0..4)` 105,6,1,0,0
`intin(0)` Window Handle (Kennziffer des Fensters)
`intin(1)` Änderungsart, und zwar:
 1 = `intin 2` enthält d. Komponenten-Bit-Feld (Create)
 2 = `intin 2+3` bilden einen Zeiger, der auf den Namen des Fensters (String im Speicher) verweist.
 3 = wie vor, jedoch für die Informationszeile.
 4 = reserviert
 5 = Festlegung des aktuellen Fensters

6-7 = reserviert

8 = Position Horizontaler Schieber ändern (relativ)

9 = Position Vertikaler Schieber ändern (relativ)

10 = Augenblicklich aktives Fenster festlegen

11-13 = reserviert

14 = Neue Desktop-Zeichnungs-Adresse übergeben intin 2 + 3 =
Adresse Struktur, 4 = Erstes Objekt

15 = Größe des horizontalen Schiebers ändern.

16 = Größe des vertikalen Schiebers ändern.

intin(2-5) enthält die Änderungsdaten (s.o.)

Ausgabeparameter:

intout(0) Fehlerüberprüfung

0 = Fehler

>0 = Kein Fehler

Zugriff in C int wi_shandle; (intin 0)

int wi_sfield; (intin 1)

int wi_sw1; (intin 2)

int wi_sw2; (intin 3)

int wi_sw3; (intin 4)

int wi_sw4; (intin 5)

int wi_sreturn; (intout)

wi_sreturn=wind_set(wi_shandle, wi_sfield, wi_sw1, wi_sw2, wi_sw3,
wi_sw4);

WIND_FIND OP: 106

Ausgabe des "Window Handle" des Fensters, auf das der Grafik-Cursor (Maus) zeigt.

Eingabeparameter:

control(0..4) 106,2,1,0,0

intin(0) x-Ordinate Mauszeiger

intin(1) y-Ordinate Mauszeiger

Ausgabeparameter:

intout(0) Window Handle (ermittelte Kennziffer)

Zugriff in C int wi_fmx; (intin 0)

int wi_fmy; (intin 1)

int wi_freturn; (intout)

wi_freturn=wind_find(wi_fmx, wi_fmy);

Bemerkungen:

Wenn der Mauszeiger nicht auf ein Fenster zeigt, wird 0 zurückgegeben.

WIND_UPDATE	OP: 107
-------------	---------

Neuaufbau eines Bildschirmbereichs bzw. Übernahme der Mauskontrolle durch die Applikation. Im ersten Fall ist kein Benutzerzugriff mehr gestattet. Im zweiten Fall sind die Dropdown-Menüs inaktiv und Größe und Positionsänderungen werden der Applikation nicht mehr übermittelt.

Eingabeparameter:

control(0..4) 107,1,1,0,0

intin(0) Statusübermittlung:

1 = Bildaufbau beendet

2 = Bildaufbau begonnen

3 = Mauskontrolle nicht mehr beim Benutzer

4 = Mauskontrolle wieder beim Benutzer

Ausgabeparameter:

intout(0) Fehlerüberprüfung (0=Fehler, >0=Kein Fehler)

Zugriff in C int wi_ubegend; (Eingabeparameter)
 int wi_ureturn; (Ausgabeparameter)
 wi_ureturn=wind_update(wi_ubegende);

WIND_CALC	OP: 108
-----------	---------

Berechnet anhand der bekannten Größe die fehlende (Fensterumriß oder Außenmaße des Arbeitsbereichs).

Eingabeparameter:

control(0..4) 108,6,5,0,0

intin(0) Art der ausgeführten Funktion

1 = Fensterumrißmaße berechnen (# 0)

2 = Fensterarbeitsbereich berechnen (# 1)

intin(1) Bit-Feld der Fensterkomponenten (-> WIND_CREATE)

intin(2) x-Ordinate des Arbeitsbereichs bzw. Fensterumrisses

intin(3) y-Ordinate des Arbeitsbereichs bzw. Fensterumrisses

intin(4) Breite des Arbeitsbereichs bzw. Fensterumrisses

intin(5) Höhe des Arbeitsbereichs bzw. Fensterumriss Ausgabeparameter:

intout(0) Fehlerüberprüfung (0=Fehler, >0=Kein Fehler)
 intout(1) x-Ordinate des Fensterumrisses bzw. Arbeitsbereichs
 intout(2) y-Ordinate des Fensterumrisses bzw. Arbeitsbereichs
 intout(3) Breite des Fensterumrisses bzw. Arbeitsbereichs
 intout(4) Höhe des Fensterumrisses bzw. Arbeitsbereichs

Zutritt in C

```

int wi_ctype; (intin 0)
int wi_ckind; (intin 1)
int wi_cinx; (intin 2)
int wi_ciny; (intin 3)
int wi_cinw; (intin 4)
int wi_cinh; (intin 5)
int wi_creturn; (intout 0)
int wi_coutx; (intout 1)
int wi_couty; (intout 2)
int wi_coutw; (intout 3)
int wi_couth; (intout 4)
wi_creturn=win_calc(wi_ctype, wi_ckind, wi_cinx, wi_ciny, wi_cinw,
wi_cinh, wi_coutx, wi_couty, wi_coutw, wi_couth);
  
```

Ereignis-Behandlung

EVNT_BUTTON OP: 21

Wartet auf einen Maustastendruck. Der Tastencode und die Mausposition werden festgestellt. Außerdem werden die Nicht-ASCII-Tasten (**Shift**, **Ctrl**, **Alternate**) abgefragt.

Eingabeparameter:

control(0..4) 21,3,5,0,0
 intin(0) Maustastenbetätigungen (Clicks), die zu einem Resultat führen sollen (Anzahl).
 intin(1) Berücksichtigungs-Maske
 1 = links
 2 = rechts
 3 = beide
 intin(2) Erforderlicher Reaktionszustand
 0 = nicht gedrückt
 1 = gedrückt (norm.)

Ausgabeparameter:

- intout(0) Anzahl der Tastenbetätigungen (Clicks) innerhalb des Bemessungszeitraumes (Sekundenbruchteil).
- intout(1) x-Ordinate Mauszeiger beim Klicken.
- intout(2) y-Ordinate Mauszeiger beim Klicken
- intout(3) Betätigte Maustasten entsprechend der AND-Bitmaske zu intin 1.
- intout(4) Tastenstatus von Nicht-ASCII-Tasten
 0 = keine Taste
 1 = rechts
 2 = links (oder beliebige Kombinationen daraus)
 4 =
 5 =

Zugriff in C

```
int ev_bclicks;(intin 0)
int ev_bmask;(intin 1)
int ev_bstate;(intin 2)
int ev_breturn;(intout 0)
int ev_bmx;(intout 1)
int ev_bmy;(intout 2)
int ev_bbutton;(intout 3)
int ev_bkstate;(intout 4)
ev_breturn=evnt_button(ev_bclicks, ev_bmask, ev_bstate, &ev_bmx,
&ev_bmy, &ev_bbutton, &ev_bkstate);
```

EVNT_DCLICK

OP: 26

Doppelklick-Geschwindigkeit setzen/abfragen.

Eingabeparameter:

- control(0..4) 26,2,1,0,0
- intin(0) Geschwindigkeit von 0 (langsam) bis 4 (schnell)
- intin (1) Flag: 1=neuen Wert setzen, 0=alten Wert ermitteln

Ausgabeparameter:

- intout(0) aktuelle Geschwindigkeit

EVNT_KEYBD OP: 20

Es wird auf einen Tastendruck gewartet. Der Code der gedrückten Taste wird an intout übergeben (ASCII im Low-Byte, Scan-Code im High-Byte).

Eingabeparameter:

control(0..4) 20,0,1,0,0

Ausgabeparameter:

intout(0) Codierung der gedrückten Taste.

Zugriff in C int ev_kreturn; (Ausgabeparameter)
ev_kreturn=evnt_keybd();

EVNT_MOUSE	OP: 22
------------	--------

Wartet, bis sich der Mauszeiger innerhalb eines definierten Bereichs (Rechteck) befindet oder diesen Bereich verläßt.

Eingabeparameter:

control(0..4) 22,5,5,0,0

intin(0) Prüfen auf Betreten (0) oder Verlassen (1) (Flag)

intin(1) x-Ordinate Rechteck

intin(2) y-Ordinate Rechteck

intin(3) Breite

intin(4) Höhe

Ausgabeparameter:

intout(0) reserviert (=1)

intout(1) x-Ordinate Mauszeiger

intout(2) y-Ordinate Mauszeiger

intout(3) Maustastenstatus (0=keine 1=links 2=rechts)

intout(4) Tastenstatus von Nicht-ASCII-Tasten:

0 = keine Taste

1 = rechts

2 = links (oder beliebige Kombinationen daraus)

4 =

5 =

Zugriff in C int ev_moflags (intin 0)
int ev_mox; (intin 1)
int ev_moy; (intin 2)
int ev_mowidth; (intin 3)
int ev_moheight; (intin 4)
int ev_moresvd; (intout 0)
int ev_momx; (intout 1)
int ev_momy; (intout 2)

```
int ev_mobutton;(intout 3)
int ev_mokstate;(intout 4)
ev_moresvd=evnt_mouse(ev_moflags, ev_mox, ev_moy, ev_mowidth,
ev_moheight, &ev_momx, &ev_momy, &ev_mobutton, &ev_mokstate);
```

EVNT_TIMER	OP: 24
------------	--------

Pause (warten) für eine bestimmte Zeit.

Eingabeparameter:

control(0..4) 24,2,1,0,0

intin(0-1) 64-Bit-Wert (Pause in 1/1000 Sekunden Dauer)

Ausgabeparameter:

intout(0) reserviert (=1)

Zugriff in C ev_tlocount;(intin 0)
ev_thicount;(intin 1)
ev_tresvd;(intout)
ev_tresvd=evnt_timer(ev_tlocount, ev_thicount);

EVNT_MESAG	OP: 23
------------	--------

Warten auf das Vorliegen einer Nachricht im Ereignis-Buffer.

Eingabeparameter:

control(0..4) 23,0,1,1,0

addrin(0) Adresse, an welcher der 16-Byte Nachrichten-Buffer beginnt.

Ausgabeparameter:

intout(0) reserviert (=1)

Zugriff in C int ev_mgpbuff;(addrin)
int ev_mgresvd;(intout)

EVNT_MULTI	OP: 25
------------	--------

Warten auf eines oder mehrere Ereignisse. Auf welche Ereignisse geprüft werden soll, wird in einem Bit-Muster festgelegt:

Bit	
0	Tastatur
1	Maustasten
2	Mausbereich I (definiertes Rechteck)
3	Mausbereich II (definiertes Rechteck)
4	Das Vorhandensein einer Nachricht im Ereignis-Buffer
5	Timer

Eingabeparameter:

control(0..4) 25,16,7,1,0

intin(0) Bitmaske für Ereignisüberprüfung, s.o. von EVNT_BUTTON:

intin(1) Maustastenbetätigungen (Klicks), die zu einem Resultat führen sollen (Anzahl)

intin(2) Berücksichtigungs-Maske

1 = links

2 = rechts

3 = beide

intin(3) Erforderlicher Reaktionszustand

0 = nicht gedrückt

1 = gedrückt (norm.)

intin(4) Prüfen auf Betreten (0) oder Verlassen (1) (Flag) (von EVNT_MOUSE 1. Rechteck)

intin(5) x-Ordinate Rechteck

intin(6) y-Ordinate Rechteck

intin(7) Breite

intin(8) Höhe

intin(9) Prüfen auf Betreten (0) oder Verlassen (1) (Flag) (von EVNT_MOUSE 2. Rechteck)

intin(10) x-Ordinate Rechteck

intin(11) y-Ordinate Rechteck

intin(12) Breite

intin(13) Höhe

intin(14) bis

intin(15) 64-Bit-Wert (Pause in 1/1000 Sekunden Dauer) (von EVNT_TIMER)

addrin(0) Adresse, an welcher der 16-Byte Nachrichten-Buffer beginnt. (von EVNT_MESAG:)

Ausgabeparameter:

- intout(0) Aufgetretene Ereignisse (intin 0 AND stattgefundenere Ereignisse)
von EVNT_BUTTON:
- intout(1) x-Ordinate Mauszeiger
- intout(2) y-Ordinate Mauszeiger
- intout(3) Maustastenstatus (0=keine 1=links 2=rechts)
- intout(4) Tastenstatus von Nicht-ASCII-Tasten:
0 = keine Taste
1 = rechts
2 = links (oder beliebige Kombinationen daraus)
4 =
5 =
- intout(5) Gedrückte Taste
- intout(6) Anzahl der zum Ereignis führenden Maustastenbetätigungen.

Zugriff in C

```

ev_mflags; (intin 0)
ev_mbclicks; (intin 1)
ev_mbmask; (intin 2)
ev_mbstate; (intin 3)
ev_mm1flags; (intin 4)
ev_mm1x; (intin 5)
ev_mm1y; (intin 6)
ev_mm1width; (intin 7)
ev_mm1height; (intin 8)
ev_mm2flags; (intin 9)
ev_mm2x; (intin 10)
ev_mm2y; (intin 11)
ev_mm2width; (intin 12)
ev_mm2height; (intin 13)
ev_mtlocount; (intin 14)
ev_mthicount; (intin 15)
ev_mmgpbuff; (addrin)
ev_mwhich; (intout 0)
ev_mmoz; (intout 1)
ev_mmoxy; (intout 2)
ev_mmobutton; (intout 3)
ev_mmokstate; (intout 4)
ev_mkreturn; (intout 5)
ev_mbreturn; (intout 6)
ev_mwhich=evnt_multi(ev_flags, ev_mbclicks, ev_mbmask, ev_mbstate,
ev_mm1flags, ev_mm1x, ev_mm1y, ev_mm1width, ev_mm1height,
ev_mm2flags, ev_mm2x, ev_mm2y, ev_mm2width, ev_mm2height,
ev_mmgpbuff, ev_mtlocount, ev_mthicount, &ev_mmoz, &ev_mmoxy,
&ev_mmobutton, &ev_mmokstate, &ev_mkreturn, &ev_mbreturn;

```

Applikations-Behandlung

APPL_EXIT OP: 19

Abmelden der Applikation, Freigabe der Application-ID.

Eingabeparameter:

control(0..4) 19,0,1,0,0

Ausgabeparameter:

intout(0) Status: 0=Fehler, >0=OK

Zugriff in C int ap_xreturn;
 ap_xreturn=appl_exit();

APPL_FIND OP: 13

Identifikationsnummer (ID) einer parallel laufenden Applikation ermitteln.

Eingabeparameter:

control(0..4) 13,0,1,1,0

addrin(0) Adresse des File-Namens der zu suchenden Applikation

Ausgabeparameter:

intout(0) Applikations-Identifikationscode

Zugriff in C int ap_fid;
 ap_fid=appl_find(ap_File-Name);

APPL_INIT OP: 10

Initialisierung der AES-Bibliotheken. Erst nach Durchführung dieser Routine sind die anderen für den Benutzer verfügbar.

Eingabeparameter:

control(0..4) 10,0,1,0,0

Ausgabeparameter:

intout(0) Applikations-Identifikationscode

Zugriff in C `int ap_id;
ap_id=appl_init();`

Bemerkungen:

Der Applikations-Identifikationscode ist z.Z. noch ohne Bedeutung, da nur eine Applikation geladen ist. Bei der vorliegenden Applikation wird er auf -1 (\$FFFF) gesetzt.

APPL_READ OP: 11

Lesen einer Anzahl von Bytes aus einem Ereignispuffer.

Eingabeparameter:

control(0..4) 11,2,1,1,0
 intin(0) Applikations-Identifikationscode (hFFFF)
 intin(1) Anzahl der zu lesenden Bytes
 addrin(0) Buffer-Adresse mit den zu lesenden Bytes

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C `int ap_rid;(intin 0)
int ap_rlenght;(intin 1)
int ap_rpbuff;(addrin)
int ap_rreturn;(intout)
ap_rreturn=appl_read(ap_rid,ap_rlenght,ap_rpbuff);`

APPL_TRECORD OP: 15

Speichern aller Ereignisse (Events), die auftreten. Mit APPL_TPLAY können diese Ereignisse wieder "abgespielt" werden.

Eingabeparameter:

control(0..4) 15,1,1,1,0
 intin(0) Anzahl der zu speichernden Events
 addrin(0) Adresse des Speichers für die Events

Ausgabeparameter:

intout(0) Anzahl der aufgezeichneten Ereignisse

Zugriff in C `int ap_trcount, ap_trreturn;
 ap_trreturn=appl_trecord(ap_trmem, ap_trcount);`

APPL_TPLAY OP: 14

Abspielen der mit APPL_TRECORD aufgenommenen Folge von Ereignissen.

Eingabeparameter:

`control(0..4)` 14,2,1,1,0
`intin(0)` Anzahl der abzuspielenden Ereignisse
`intin(1)` Abspiel-Geschwindigkeit (0-1000)
`addrin(0)` Adresse der gespeicherten Ereignisse

Ausgabeparameter:

`intout(0)` immer 1

Zugriff in C `int ap_tpnum, ap_tpscale;
 ap_tpretreturn=appl_tplay(ap_tpmem, ap_tpnum, ap_tpscale);`

APPL_WRITE OP: 12

Schreiben einer Anzahl von Bytes in einen Ereignis-Buffer.

Eingabeparameter:

`control(0..4)` 12,2,1,1,0
`intin(0)` Applikations-Identifikationscode (hFFFF)
`intin(1)` Anzahl der zu schreibenden Bytes
`addrin(0)` Buffer-Adresse, an welche die Bytes geschrieben werden sollen.

Ausgabeparameter:

`intout(0)` Fehlerkontrolle (0=Fehler 1=kein Fehler)

Zugriff in C `int ap_wid;(intin 0)
 int ap_wlenght;(intin 1)
 int ap_wpbuff;(addrin)
 int ap_wreturn;(intout)
 ap_wreturn=appl_write(ap_wid,ap_wlenght,ap_wpbuff);`

Objektbehandlung

OBJC_ADD OP: 40

Einfügen eines Objekts in einen Objektbaum.

Eingabeparameter:

control(0..4) 40,2,1,1,0

intin(0) Index des Objekts, an das angefügt werden soll

intin(1) Index des neuen Objektes

addrin(0) Adresse des Objektbaumes

Ausgabeparameter:

intout(0) Status: 0=Fehler, 1=OK

Zugriff in C int ob_aparent, ob_achild;
ob_areturn=objc_add(ob_atree, ob_aparent, ob_achild);

OBJC_CHANGE OP: 47

Ändern des Aussehens des Objektes in der Begrenzung.

Eingabeparameter:

control(0..4) 47,8,1,1,0

intin(0) Objekt (Index), das geändert werden soll

intin(1) reserviert (muß 0 sein)

intin(2) x-Ordinate Begrenzungskasten

intin(3) y-Ordinate Begrenzungskasten

intin(4) Begrenzungskasten-Breite

intin(5) Begrenzungskasten-Höhe

intin(6) Neuer Objektstatus

intin(7) Zeichnungsflag (0=nicht neu zeichnen 1=neu zeichnen)

addrin(0) Objektbaum (Adresse)

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C int ob_cobject; (intin 0)
int ob_cresvd; (intin 1)
int ob_cxclip; (intin 2)
int ob_cyclip; (intin 3)

```

int ob_cwclip;(intin 4)
int ob_chclip;(intin 5)
int ob_cnewstate;(intin 6)
int ob_credraw;(intin 7)
int ob_ctree;(addrin)
int ob_creturn;(intout)
ob_creturn=objc_change(ob_ctree, ob_cobject, ob_crescvd,
ob_cxclip, ob_cyclip, ob_cwclip, ob_chclip, ob_cnewstate,
ob_credraw);

```

OBJC_DELETE OP: 41

Nimmt ein Objekt aus einer Baumstruktur heraus (löscht es).

Eingabeparameter:

control(0..4) 41,1,1,1,0
intin(0) Index des zu löschenden Objekts
addrin(0) Adresse des Objektbaumes

Ausgabeparameter:

intout(0) Status: 0=Fehler, 1=OK

Zugriff in C int ob_dobject, ob_dlreturn;
 ob_dlreturn=objc_delete(ob_dltree, ob_dobject);

OBJC_DRAW OP: 42

Zeichnen eines Objektbaumes innerhalb von festzulegenden Zeichengrenzen und einer Objektbaumstrukturebene.

Eingabeparameter:

control(0..4) 42,6,1,1,0
intin(0) Index des zuerst zu zeichnenden Objektes
intin(1) Anzahl der Strukturebenen
intin(2) x-Ordinate Rechteckbegrenzung
intin(3) y-Ordinate Rechteckbegrenzung
intin(4) Rechteckbreite
intin(5) Rechteckhöhe
addrin(0) Objektbaum-Adresse

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler 1=kein Fehler)

Zugriff in C ob_drstartob; (intin 0)
 ob_drdepth; (intin 1)
 ob_drxcclip; (intin 2)
 ob_dryclip; (intin 3)
 ob_drwclip; (intin 4)
 ob_drhclip; (intin 5)
 ob_drtree; (addrin),
 ob_drreturn; (intout)
 ob_drreturn=objc_draw(ob_drtree, ob_drstartob, ob_drdepth,
 ob_drxcclip, ob_dryclip, ob_drwclip, ob_drhclip);

OBJC_EDIT	OP: 46
-----------	--------

Eingabe eines Textes in ein Objekt vom Typ G_TEXT oder G_BOXTEXT.

Eingabeparameter:

control(0..4) 46,4,2,1,0
 intin(0) Objekt (Index) mit dem einzugebenden Text
 intin(1) Zeicheneingabe
 intin(2) Nächste Zeichenposition im Text
 intin(3) Editiermodus (0=reserviert, 1..3)
 addrin(0) Objektbaum (Adresse)

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)
 intout(1) Nächste Zeichenposition nach dem Aufruf (Nummer)

Zugriff in C int ob_edobject; (intin 0)
 int ob_edchar; (intin 1)
 int ob_edidx; (intin 2)
 int ob_edkind; (intin 3)
 int ob_edtree; (addrin)
 int ob_ed_return; (intout 0)
 int ob_ednewidx; (intout 1)
 ob_edreturn=objc_edit(ob_edtree, ob_edobject, ob_edchar, ob_edidx,
 ob_edkind, ob_ednewidx);

Bemerkungen:

Die Funktion muß dreimal hintereinander mit unterschiedlichen Werten für den Editiermodus aufgerufen werden.

OBJC_FIND OP: 43

Ermittlung des Objektes unter dem Mauszeiger. Der Startpunkt wird angegeben.

Eingabeparameter:

control(0..4) 43,4,1,1,0
 intin(0) Objekt (Index); bei dem die Suche beginnt
 intin(1) Anzahl der abzusuchenden Strukturebenen
 intin(2) x-Ordinate Mauspointer
 intin(3) y-Ordinate Mauspointer
 addrin(0) Baum (Adresse), in dem gesucht werden soll

Ausgabeparameter:

intout(0) Objekt (Index); auf dem sich der Mauszeiger befindet (wenn kein Objekt, dann -1)

Zugriff in C int ob_fstartob; (intin 0)
 int ob_fdepth; (intin 1)
 int ob_fm; (intin 2)
 int ob_fmy; (intin 3)
 int ob_ftree; (addrin)
 int ob_fobnum; (intout)
 ob_fobnum=objc_find(ob_ftree, ob_fstartob, ob_fdepth, ob_fm, ob_fmy);

OBJC_OFFSET OP: 44

Ermittelt die Objektkoordinaten relativ zum Bildschirm-Nullpunkt.

Eingabeparameter:

control(0..4) 44,1,3,1,0
 intin(0) Objekt (Index), dessen Position gefunden werden soll
 addrin(0) Objektbaum (Adresse)

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)
 intout(1) relative x-Ordinate zum Bildschirmnullpunkt
 intout(2) relative y-Ordinate zum Bildschirmnullpunkt

Zugriff in C int ob_ofobject; (intin)
 int ob_ofree; (addrin)

```

int ob_ofreturn;(intout 0)
int ob_ofxoff;(intout 1)
int ob_ofyoff;(intout 2)
ob_ofreturn=objc_offset(ob_oftree,ob_ofobject,&ob_ofxoff,&ob_ofyoff);

```

OBJC_ORDER OP: 45

Verschieben eines Objekts innerhalb der logischen Baumstruktur.

Eingabeparameter:

control(0..4) 45,2,1,1,0
 intin(0) Index des zu verschiebenden Objekts
 intin(1) neue Position des Objekts
 addrin(0) Adresse des Objektbaumes

Ausgabeparameter:

intout(0) Status: 0=Fehler, 1=OK

Objektbaumstruktur-Behandlung

RSRC_FREE OP: 111

Der mit der Objektbaumstruktur besetzte Speicher (RSRC_LOAD) wird wieder freigegeben. Der Objektbaum ist danach nicht mehr zugänglich.

Eingabeparameter:

control(0..4) 111,0,1,0,0

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler 1=kein Fehler)

Zugriff in C int re_freturn;(Ausgabeparameter)
 re_freturn=rsrc_free();

RSRC_GADDR OP: 112

Ermittlung der Datenstrukturadresse von Objektbäumen oder objektspezifischen Informationen.

Eingabeparameter:

control(0..4) 112,2,1,0,1
intin(0) Art der gesuchten Datenstruktur(-adresse):
0 = mit RSRC_LOAD geladene Baumstruktur
1 = Objekt
2 = Text
3 = Piktogramm
4 = Bit-Mustergrafik
5 = String-Text
6 = Bit-Mustergrafik-Zeiger
7 = Objektspezifizierung
8 = Text-Zeiger
9 = Textschablone-Zeiger
10 = Textabgabebegrenzungs-String-Zeiger
11 = Piktogramm Darstellungsmaske
12 = Piktogramm Bit-Muster
13 = Piktogramm Text
14 = Bit-Mustergrafik-Zeiger
15 = Freier String-Zeiger
16 = Freie Bit-Mustergrafik-Zeiger
intin(1) Objekt (Index)

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)
addrout(0) Datenstrukturadresse

Zugriff in C int re_gtype; (intin 0)
int re_gindex; (intin 1)
int re_greturn; (intout)
int re_gaddr; (addrout)
re_greturn=rsrc_gaddr(re_gtype, re_gindex, re_gaddr);

RSRC_LOAD OP: 110

Es wird Speicherplatz freigegeben und in diesen eine Objektbaumstruktur geladen. Der für den Eingabeparameter notwendige String muß mit h00 abgeschlossen werden.

Eingabeparameter:

control(0..4) 110,0,1,1,0
addrin(0) Pointer auf den Dateinamen-String.

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C int re_lpfname;(addrin)
 int re_lreturn;(intout)
 re_lreturn=rsrc_load(re_lpfname);

RSRC_OBFIX OP: 114

Umwandeln der Koordinaten eines Objektes von Zeichen- auf Pixel-Format (geschieht automatisch bei RSRC_LOAD).

Eingabeparameter:

control(0..4) 114,1,1,1,0
 intin(0) Index des betreffenden Objekts
 addrin(0) Adresse des Objektbaumes

Ausgabeparameter:

intout(0) reserviert

Zugriff in C int re_oresvd, re_oobject;
 re_oresvd=rsrc_obfix(re_otree, re_oobject);

RSRC_SADDR OP: 113

Speicherung einer Datenstrukturadresse in einer objektspezifischen Datenstruktur.

Eingabeparameter:

control(0..4) 113,2,1,1,0
 intin(0) Art der zu speichernden Datenstruktur(-adresse):
 0 = mit RSRC_LOAD geladene Baumstruktur
 1 = Objekt
 2 = Text
 3 = Piktogramm
 4 = Bit-Mustergrafik
 5 = String-Text
 6 = Bit-Mustergrafik-Zeiger
 7 = Objektspezifizierung
 8 = Text-Zeiger
 9 = Textschablone-Zeiger

10 = Textabgabebegrenzungs-String-Zeiger
11 = Piktogramm Darstellungsmaske
12 = Piktogramm Bit-Muster
13 = Piktogramm Text
14 = Bit-Mustergrafik-Zeiger
15 = Freier String-Zeiger
16 = Freie Bit-Mustergrafik-Zeiger

intin(1) Objekt (Index)
addrin(0) Datenstrukturadresse

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=Kein Fehler)

Zugriff in C int re_stype; (intin 0)
int re_sindex; (intin 1)
int re_saddr; (addrin) int re_sreturn; (intout)
re_sreturn=rsrc_saddr(re_stype, re_sindex, re_saddr);

Verwaltung von Formularen

FSEL_INPUT OP: 90

Darstellung und Verwaltung des File-Selectors.

Eingabeparameter:

control(0..4) 90,0,2,2,0
addrin(0) Adresse des Pfadnamens (Abschluß mit 0)
addrin(1) Adresse des File-Namens (Abschluß mit 0)

Ausgabeparameter:

intout(0) Status: 0=Fehler, 1=OK
intout(1) Betätigte Rückgabetaste (1=OK, 0=Abbruch)

Zugriff in C int fs_ireturn, fs_iexbutton;
fs_ireturn=fsel_input(fs_iinpath, fs_iinsel, &fs_iexbutton);

FORM_ALERT OP: 52

Eine Warnmeldung wird ausgegeben. FORM_ALERT ruft gleichzeitig OBJC_DRAW und FORM_DO auf. Bis zu drei "Knöpfe", die dann bei Betäti-

gung eine bestimmte Reaktion verursachen, können dargestellt werden. Es wird auf eine Benutzereingabe gewartet.

Eingabeparameter:

control(0..4) 52,1,1,1,0
 intin(0) Return-Taste simuliert:
 0 = keinen Knopf
 1 = ersten Knopf
 2 = zweiten Knopf
 3 = dritten Knopf
 addrin(0) Adresse zum Warnmeldungs-String

Ausgabeparameter:

intout(0) Betätigter Bestätigungsknopf (1..3)

Zugriff in C int fo_adeftbtn; (intin)
 int fo_astring; (addrin)
 int fo_aexbbtn; (intout)
 fo_aexbbtn=form_alert(fo_adeftbtn, fo_astring);

FORM_BUTTON	OP: 56
-------------	--------

Überwachung der Maustasten, ähnlich wie FORM_KEYBD.

Eingabeparameter:

control(0..4) 56,2,1,1,0
 intin(0) Objekt, das sich unter dem Mauszeiger befindet
 intin(1) Anzahl der Mausklicks
 addrin(0) Adresse der Baumstruktur

Ausgabeparameter:

intout(0) Status:
 0 = es wurde EXIT gewählt
 >0 = noch nicht fertig

Zugriff in C int fo_breturn, fo_bob, fo_bclks;
 fo_breturn=form_button(fo_btrees, fo_bob, fo_bclks, &fo_bobnxt);

FORM_CENTER**OP: 54**

Berechnung der Koordinaten eines bestimmten Objektbaums. Im Fall einer Bildschirmdarstellung erscheint dieser in der Mitte.

Eingabeparameter:

control(0..4) 54,0,5,1,0

addrin(0) Baumstruktur-Adresse, dessen Koordinaten berechnet werden sollen.

Ausgabeparameter:

intout(0) reserviert (=1)

intout(1) Zentrierte x-Ordinate

intout(2) Zentrierte y-Ordinate

intout(3) Breite

intout(4) Höhe

Zugriff in C `int fo_ctree;(addrin)`
 `int fo_cresvd;(intin 0)`
 `int fo_cx;(intin 1)`
 `int fo_cy;(intin 2)`
 `int fo_cw;(intin 3)`
 `int fo_ch;(intin 4)`
 `fo_cresvd=form_center(fo_tree, &fo_cx, &fo_cy, &fo_cw, &fo_ch);`

FORM_DIAL OP: 51

Für die Verwaltung von Formularen sind die folgenden Funktionen (in der angegebenen Reihenfolge) erforderlich. Welche Funktion ausgeführt werden soll, muß an intin(0) übergeben werden.

0 = Bildschirmspeicherbereich reservieren

1 = sich ausdehnendes Rechteck zeichnen

2 = sich verkleinerndes Rechteck zeichnen

3 = reservierten Speicherbereich freigeben.

Eingabeparameter:

control(0..4) 51,9,1,1,0

intin(0) Auszuführende Funktion, s.o.

intin(1) x-Ordinate des am kleinsten ausgedehnten Rechtecks

intin(2)	y-Ordinate des am kleinsten ausgedehnten Rechtecks
intin(3)	Breite des am kleinsten ausgedehnten Rechtecks
intin(4)	Höhe des am kleinsten ausgedehnten Rechtecks
intin(5)	x-Ordinate des am größten ausgedehnten Rechtecks
intin(6)	y-Ordinate des am größten ausgedehnten Rechtecks
intin(7)	Breite des am größten ausgedehnten Rechtecks
intin(8)	Höhe des am größten ausgedehnten Rechtecks

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler, >0=kein Fehler)

Zugriff in C

```

int fo_diflag; (intin 0)
int fo_dilittlx; (intin 1)
int fo_dilittly; (intin 2)
int fo_dilittlw; (intin 3)
int fo_dilittlh; (intin 4)
int fo_dibigx; (intin 5)
int fo_dibigy; (intin 6)
int fo_dibigw; (intin 7)
int fo_dibigh; (intin 8)
fo_direturn=form_dial(fo_diflag, Anfangs-x, Anfangs-y, Anfangs-
breite, Anfangshöhe);

```

FORM_DO	OP: 50
---------	--------

Übergabe der Programmkontrolle an AES zur Überwachung der Benutzereingaben im Formular.

Eingabeparameter:

control(0..4) 50,1,1,1,0

intin(0) Formularaufbau (0=keine Textfelder im Formular n=Anzahl der Textfelder im Formular)

addrin(0) Objektbaumadresse

Ausgabeparameter:

intout(0) Vom Benutzer angewähltes Objekt (Index)

Zugriff in C

```

int fo_dostartob; (intin)
int fo_dotree; (addrin)
int fo_doreturn; (intout)
fo_doreturn=form_do(fo_dotree, fo_dostartob);

```

FORM_ERROR OP: 53

Eine Fehlermeldung wird ausgegeben. Es wird ein Alarmfenster mit der Fehlernummer ausgegeben. Als Bestätigungstaste steht ein CANCEL-Feld zur Verfügung. Es wird auf eine Benutzereingabe gewartet.

Eingabeparameter:

control(0..4) 53,1,1,0,0
intin(0) Aufgetretener Fehler (nur TOS-Fehler)

Ausgabeparameter:

intout(0) Betätigte Rückgabetaste (nur 1 ist möglich)

Zugriff in C int fo_enum; (intin)
 int fo_eexbbtn; (intout)
 fo_eexbbtn=form_error(fo_enum);

FORM_KEYBD OP: 55

Kontrolle von Formulareingaben, Unteroutine von FORM_DO.

Eingabeparameter:

control(0..4) 55,3,2,1,0
intin(0) Edit-Objekt, in dem sich der Cursor befindet
intin(1) nächstes Objekt
intin(2) eingegebenes Zeichen
addrin(0) Adresse der Baumstruktur

Ausgabeparameter:

intout(0) Status:
 0 = es wurde EXIT gewählt
 >0 = noch nicht fertig
intout(1) neues Edit-Objekt
intout(2) Text-Zeichen (=0, wenn Steuerzeichen)

Zugriff in C int fo_kreturn, fo_kob, fo_kobnxt, fo_kchar;
 int fo_koboutnxt, fo_koutchar;
 fo_kreturn=form_keybd(fo_ktree, fo_kob, fo_kobnxt, fo_kchar,
 &fo_koboutnxt, &fo_koutchar);

Pull-Down-Menü-Handhabung

MENU_BAR OP: 30

Darstellen oder Löschen der Menüleiste eines Menüobjektbaums.

Eingabeparameter:

control(0..4) 30,1,1,1,0
 intin(0) Aktionsflag (0=löschen 1=darstellen)
 addrin(0) Menüobjektbaum-Adresse

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C int me_bshow;(intin)
 int me_btree;(addrin)
 int me_breturn;(intout)
 me_breturn=menu_bar(me_btree, me_bshow);

MENU_ICHECK OP: 31

Löschen oder Darstellen eines Prüfsymbols (der OK-Haken, der neben einem Menüpunkt zum Zeichen seiner Aktivierung erscheinen kann).

Eingabeparameter:

control(0..4) 31,2,1,1,0
 intin(0) Menü-Eintrag-Nummer
 intin(1) Darstellungs-Flag (0=nicht darstellen 1=darstellen)
 addrin(0) Menüobjektbaum-Adresse

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C int me_citem;(intin 0)
 int me_ccheck;(intin 1)
 int me_ctree;(addrin)
 int me_creturn;(intout)
 me_creturn=menu_ichkck(me_ctree, me_citem, me_ccheck);

MENU_IENABLE OP: 32

Einzelne Menüpunkte können dem Benutzerzugriff entzogen werden. Der Menüpunkt erscheint dann in LIGHT-Schrift.

Eingabeparameter:

control(0..4) 32,2,1,1,0
intin(0) Menü-Eintrag-Nummer
intin(1) Aktivitäts-Flag (0=LIGHT-Schrift 1=aktiviert)
addrin(0) Menüobjektbaum-Adresse

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C int me_eitem;(intin 0)
 int me_echeck;(intin 1)
 int me_etree;(addrin)
 int me_ereurn;(intout)
 me_ereurn=menu_ienable(me_etree, me_eitem, me_echeck);

MENU_TNORMAL OP: 33

Menütitel können auch in umgekehrter Schriftart (bei monochrom weiße Schrift auf schwarzem Balken) dargestellt werden, um die Auswahl kenntlich zu machen.

Eingabeparameter:

control(0..4) 33,2,1,1,0
intin(0) Menütitel-Nummer
intin(1) Darstellungsflag (0=revers 1=normal)
addrin(0) Objektbaummenü-Adresse

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C int me_ntitle;(intin 0)
 int me_nnormal;(intin 1)
 int me_ntree;(addrin)
 int me_nreturn;(intout)
 me_nreturn=menu_tnormal(me_ntree, me_ntitle, me_nnormal);

MENU_TEXT OP: 34

Änderung von Texten im Menüeintrag. Der neue Text darf nicht länger als der alte Text sein (sonst Zerstörung der Menübaumstruktur), und der String muß mit 0 abgeschlossen werden.

Eingabeparameter:

control(0..4) 34,1,1,2,0
intin(0) Menüeintrag-Nummer
addrin(0) Menüobjektbaum-Adresse
addrin(1) Adresse des neuen Eintragtextes.

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C int me_titem;(intin)
int me_ttree;(addrin 0)
int me_ttext;(addrin 1)
int me_treturn;(intout)
me_treturn=menu_text(me_ttree, me_titem, me_ttext);

MENU_REGISTER OP: 35

Aktivierung eines Accessory-Namens im ersten Menükasten.

Eingabeparameter:

control(0..4) 35,1,1,1,0
intin(0) Applikations-Identifikationscode (-> APPL_INIT) (hier: Prozess-identifizierung)
addrin(1) Zeiger auf den Desk-Accessory-Namen (Ende 0)

Ausgabeparameter:

intout(0) Desk-Accessory-Nummer (0..5, -1=keine vorhanden)

Zugriff in C int me_rapid;(intin)
int me_rpstring;(addrin)
int me_rmenuid;(intout)
me_rmenuid=menu_register(me_rapid, me_rpstring);

Grafik-Behandlung**GRAF_HANDLE** **OP: 77**

Grafik-Handle ermitteln. Diese Funktion ist vor der Durchführung von Grafikfunktionen erforderlich.

Eingabeparameter:

control(0..4) 77,0,5,0,0

Ausgabeparameter:

intout(0)	Handle
intout(1)	Buchstabenbreite
intout(2)	Buchstabenhöhe
intout(3)	Breite der gedachten Box um den Buchstaben
intout(4)	Höhe der gedachten Box um den Buchstaben

Zugriff in C

```
int gr_handle; (intout 0)
int gr_hwchar; (intout 1)
int gr_hhchar; (intout 2)
int gr_hwbox; (intout 3)
int gr_hhbox; (intout 4)
gr_handle=graf_handle(&gr_hwchar, &gr_hhchar, &gr_hwbox,
&gr_hhbox);
```

Bemerkungen:

Intout(1)-(4) sind systemspezifische Informationen. Das Handle in intin(0) hingegen ist wichtig und sollte für spätere Benutzung gespeichert werden.

GRAF_RUBBERBOX **OP: 70**

Markiert die linke obere Ecke eines Rechtecks und folgt (wobei die linke obere Ecke stehen bleibt) den Bewegungen der Maus. Nach dem Loslassen der linken Maustaste ergibt die Mauszeigerposition die rechte untere Ecke, und die Größe des Rechtecks wird übergeben.

Eingabeparameter:

control(0..4) 70,4,3,0,0

intin(0)	x-Ordinate linke obere Ecke
intin(1)	y-Ordinate linke obere Ecke

intin(2) Kleinstmögliche Rechteckbreite
 intin(3) Kleinstmögliche Rechteckhöhe

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)
 intout(1) Rechteckbreite nach Loslassen der Maustaste
 intout(2) Rechteckhöhe nach Loslassen der Maustaste

Zugriff in C int gr_rx; (intin 0)
 int gr_ry; (intin 1)
 int gr_rminwidth; (intin 2)
 int gr_rminheight; (intin 3)
 int gr_rreturn; (intout 0)
 int gr_rlastwidth; (intout 1)
 int gr_rlastheight; (intout 2)
 gr_rreturn=graf_rubberbox(gr_rx, gr_ry, gr_rminwidth,
 gr_rminheight, &gr_rlastwidth, &gr_rlastheight);

GRAF_DRAGBOX OP: 71

Bewegen eines Rechtecks innerhalb eines zweiten mittels des Mauszeigers. Nach dem Loslassen der linken Maustaste steht das verschiebbare Rechteck wieder fest.

Eingabeparameter:

control(0..4) 71,8,3,0,0
 intin(0) Rechteckbreite (verschiebbar)
 intin(1) Rechteckhöhe (verschiebbar)
 intin(2) x-Ordinate verschiebbares Rechteck
 intin(3) y-Ordinate verschiebbares Rechteck
 intin(4) x-Ordinate feststehendes Rechteck
 intin(5) y-Ordinate feststehendes Rechteck
 intin(6) Rechteckbreite (feststehend)
 intin(7) Rechteckhöhe (feststehend)

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)
 intout(1) x-Ordinate (neu) des verschiebbaren Rechtecks
 intout(2) y-Ordinate (neu) des verschiebbaren Rechtecks

Zugriff in C int gr_dwidth; (intin 0)
 int gr_dheight; (intin 1)
 int gr_dstartx; (intin 2)

```

int gr_dstarty;(intin 3)
int gr_dboundx;(intin 4)
int gr_dboundy;(intin 5)
int gr_dboundw;(intin 6)
int gr_dboundh;(intin 7)
int gr_dreturn;(intout 0)
int gr_dfinishx;(intout 1)
int gr_dfinishy;(intout 2)
gr_dreturn=graf_dragbox(gr_dwidth, gr_dheight, gr_dstartx,
gr_dstarty, gr_dboundx, gr_dboundy, gr_dboundw, gr_dboundh,
&gr_dfinishx, &gr_dfinishy);

```

GRAF_MOVEBOX	OP: 72
---------------------	---------------

Zeichnen eines Rechtecks, welches sich von einer Position zu einer anderen fortbewegt und dort stehenbleibt.

Eingabeparameter:

control(0..4) 72,6,1,0,0
intin(0) Rechteckbreite
intin(1) Rechteckhöhe
intin(2) x-Ordinate Startposition
intin(3) y-Ordinate Startposition
intin(4) x-Ordinate Endposition
intin(5) y-Ordinate Endposition

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C

```

int gr_mwidth; (intin 0)
int gr_mheight;(intin 1)
int gr_msourcex;(intin 2)
int gr_msourcey;(intin 3)
int gr_destx;(intin 4)
int gr_desty;(intin 5)
int gr_mreturn;(intout)
gr_mreturn=graf_movebox(gr_mwidth, gr_mheight, gr_msourcex,
gr_msourcey, gr_mdestx, gr_mdesty);

```

GRAF_GROWBOX	OP: 73
---------------------	---------------

Zeichnen eines sich ausdehnenden Rechtecks. Angegeben werden die Maße für die maximale und minimale Größe.

Eingabeparameter:

control(0..4) 73,8,1,0,0
 intin(0) x-Ordinate Rechteck (klein)
 intin(1) y-Ordinate Rechteck (klein)
 intin(2) Breite Rechteck (klein)
 intin(3) Höhe Rechteck (klein)
 intin(4) x-Ordinate Rechteck (groß)
 intin(5) y-Ordinate Rechteck (groß)
 intin(6) Breite Rechteck (groß)
 intin(7) Höhe Rechteck (groß)

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C int gr_gstx; (intin 0)
 int gr_gsty; (intin 1)
 int gr_gstwidth; (intin 2)
 int gr_gstheight; (intin 3)
 int gr_gfinx; (intin 4)
 int gr_gfiny; (intin 5)
 int gr_gfinwidth; (intin 6)
 int gr_gfinheight; (intin 7)
 int gr_greturn; (intout)
 gr_greturn=graf_growbox(gr_gstx, gr_gsty, gr_gstwidth,
 gr_gstheight, gr_gfinx, gr_gfiny, gr_gfinwidth, gr_gfinheight);

GRAF_SHRINKBOX	OP: 74
----------------	--------

Zeichnen eines kleiner werdenden Rechtecks. Angegeben werden die Maße für minimale und maximale Größe.

Eingabeparameter:

control(0..4)f 74,8,1,0,0
 intin(0) x-Ordinate Rechteck (klein)
 intin(1) y-Ordinate Rechteck (klein)
 intin(2) Breite Rechteck (klein)
 intin(3) Höhe Rechteck (klein)
 intin(4) x-Ordinate Rechteck (groß)
 intin(5) y-Ordinate Rechteck (groß)
 intin(6) Breite Rechteck (groß)
 intin(7) Höhe Rechteck (groß)

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C int gr_sfinx; (intin 0)
 int gr_sfiny; (intin 1)
 int gr_sfinwidth; (intin 2)
 int gr_sfinheight; (intin 3)
 int gr_sstx; (intin 4)
 int gr_ssty; (intin 5)
 int gr_sstwidth; (intin 6)
 int gr_sstheight; (intin 7)
 int gr_sreturn; (intout)
 gr_sreturn=graf_shrinkbox(gr_sfinx, gr_sfiny, gr_sfinwidth,
 gr_sfinheight, gr_sstx, gr_ssty, gr_sstwidth, gr_sstheight);

GRAF_WATCHBOX OP: 75

Kontrolle, ob der Mauszeiger ein Rechteck (dieses muß Teil eines Objektbaums sein) betritt oder verläßt in dem Augenblick, wenn die linke Maustaste losgelassen wird.

Eingabeparameter:

control(0..4) 75,4,1,1,0
 intin(0) reserviert
 intin(1) Objekt (Index)
 intin(2) Objektstatus beim Betreten:
 0 = Normal
 1 = Selektiert
 2 = Gekreuzt
 4 = Geprüft
 8 = Nicht zugänglich
 16 = outlined
 32 = schattiert
 intin(3) Objektstatus beim Verlassen:
 0 = Normal
 1 = Selektiert
 2 = Gekreuzt
 4 = Geprüft
 8 = Nicht zugänglich
 16 = outlined
 32 = schattiert
 addrin(0) Objektbaum-Adresse

Ausgabeparameter:

intout(0) Mauszeigerstatus (0=nicht im Rechteck 1=im Rechteck)

Zugriff in C

```
int gr_wobject; (intin 1)
int gr_winstate; (intin 2)
int gr_woustate; (intin 3)
int gr_wptree; (addrin)
int gr_wreturn; (intout)
gr_wreturn=graf_watchbox(gr_wptree, gr_wobject, gr_winstate,
gr_woustate);
```

GRAF_SLIDEBOX

OP: 76

Verschieben eines Rechtecks in einem anderen horizontal oder vertikal entsprechend dem Maus-Pointer, wobei das umfassende Rechteck ein übergeordnetes Objekt des beweglichen Rechtecks sein muß. Die neue Position ist an der Stelle, an der die linke Maustaste losgelassen wird.

Eingabeparameter:

control(0..4) 76,3,1,1,0
 intin(0) Rechteck (Index) im übergeordneten Objekt(baum)
 intin(1) Rechteck (Index) des beweglichen Objektes
 intin(2) Bewegungsrichtung (0=waagerecht 1=senkrecht)
 addrin(0) Objektbaum-Adresse

Ausgabeparameter:

intout(0) Mittelpunkt des beweglichen Objekts relativ zum umfassenden Rechteck.

Zugriff in C

```
int gr_slparent; (intin 0)
int gr_slobject; (intin 1)
int gr_slvh; (intin 2)
int gr_slptree; (addrin)
int gr_slreturn; (intout)
gr_slreturn=graf_slidebox(gr_slptree, gr_slparent, gr_slobject,
gr_slvh);
```

GRAF_MOUSE OP: 78

Änderung des Aussehens des Maus-Pointers. Es kann unter acht Standardformen und einer vom Benutzer definierbaren Form gewählt werden.

Eingabeparameter:

control(0..4) 78,1,1,1,0
intin(0) Mausform:
 0 = Pfeil (normal, Muß für Menüleisten etc.)
 1 = Cursor
 2 = Biene
 3 = Hand mit ausgestrecktem Zeigefinger
 4 = Hand, flach
 5 = Fadenkreuz (schmal)
 6 = Fadenkreuz (fett)
 7 = Fadenkreuz (Umriß)
 255 = freie Definition
 256 = Mausform aus
 257 = Mausform ein
addrin(0) Adresse des benutzereigenen Mausform-Definitionsblocks.

Ausgabeparameter:

intout(0) Fehlerkontrolle (0=Fehler >0=kein Fehler)

Zugriff in C gr_monumber; (intin)
 gr_mofaddr; (addrin)
 gr_moreturn; (intout)
 gr_moreturn=graf_mouse(gr_monumber, gr_mofaddr);

GRAF_MKSTATE OP: 79

Ermitteln der Mausposition, des Status der Maustasten und der Sondertasten

, und .

Eingabeparameter:

control(0..4) 79,0,5,0,0

Ausgabeparameter:

intout(0) reserviert (=1)
intout(1) x-Ordinate Mauszeiger
intout(2) y-Ordinate Mauszeiger
intout(3) Maustastenstatus (0=keine 1=links 2=rechts 3=beide)
intout(4) Nicht-ASCII-Tastenstatus
 Bit
 0 (rechts)
 1 (links)

2
 3

Zugriff in C

```
int gr_mkresvd; (intin 0)
int gr_mkmx; (intin 1)
int gr_mkmy; (intin 2)
int gr_mkmstate; (intin 3)
int gr_mkkstate; (intin 4)
gr_mkresvd=graf_mkstate(&gr_mkmx, &gr_mkmy, &gr_mkmstate,
&gr_mkkstate);
```

Scrap-Funktionen

SCRP_READ OP: 80

Wiederholen eines mit SCR_P_WRITE in einen globalen Puffer des AES geschriebenen String.

Eingabeparameter:

control(0..4) 80,0,1,1,0
 addrin(0) Adresse, in die der globale String kopiert werden soll

Ausgabeparameter:

intout(0) Status: 0=Fehler, >0=OK

Zugriff in C

```
int sc_rreturn;
sc_rreturn=scrp_read(sc_rpscrap);
```

SCRP_WRITE OP: 81

Schreiben eines Strings in den globalen Puffer des AES. Der String kann von einem anderen Programm mittels SCR_P_READ wiedergeholt werden.

Eingabeparameter:

control(0..4) 81,0,1,1,0
 addrin(0) Adresse, aus dem der String zu holen ist

Ausgabeparameter:

intout(0) Status: 0=Fehler, >0=OK

Shell-Funktionen**SHEL_FIND** OP: 124

Durchsuchen einer Diskette nach einem angegebenen File-Namen. Wird er nicht auf der vorgegebenen Diskette gefunden, so wird noch auf Laufwerk A gesucht.

Eingabeparameter:

control(0..4) 124,0,1,1,0

addrin(0) des File-Namens, der gesucht werden soll. An diese Stelle wird bei gefundenen File der Pfadname kopiert.

Ausgabeparameter:

intout(0) Status: 0=Fehler, >0=OK

SHEL_GET OP: 122

Lesen einer Anzahl Bytes aus dem Environment-Speicher des GEM (z.B. Desktop.Inf-Einträge).

Eingabeparameter:

control(0..4) 122,1,1,1,0

intin(0) Anzahl der zu lesenden Zeichen

addrin(0) Adresse des Puffers, in den geschrieben werden soll

Ausgabeparameter:

intout(0) Status: 0=Fehler, >0=OK

SHEL_PUT OP: 123

Dies ist das Gegenstück zu SHEL_GET. Es wird eine Anzahl Bytes in den Environment-Speicher geschrieben.

Eingabeparameter:

control(0..4) 123,1,1,1,0

intin(0) Anzahl der zu schreibenden Bytes

addrin(0) Adresse des Puffers, aus dem die Bytes gelesen werden sollen

Ausgabeparameter:

intout(0) Status: 0=Fehler, >0=OK

SHEL_READ OP: 120

Ermittlung des Namens und der aufrufenden Kommandozeile, mit der das Programm gestartet wurde.

Eingabeparameter:

control(0..4) 120,0,1,2,0

addrin(0) Adresse des Puffers, in den die Kommando-Zeile geschrieben werden soll

addrin(1) Adresse des Puffers, in den der Pfadname kommt

Ausgabeparameter:

intout(0) Status: 0=Fehler, >0=OK

SHEL_WRITE OP: 121

Laden und Starten von Programmen, ähnlich P_EXEC im GEMDOS. Scheint nicht so recht zu funktionieren.

Eingabeparameter:

control(0..4) 121,3,1,2,0

intin(0) Flag sh_wdoex:
 0 = GEM verlassen
 1 = Programm starten

intin(1) Flag sh_wisgr:
 0 = keine Grafik
 1 = mit Grafik

intin(2) Flag sh_wiscr:
 0 = kein GEM-Programm (TOS)
 1 = GEM-Programm

addrin(0) Kommandozeile für Programmaufruf

addrin(1) Pfadname des Programmes

Ausgabeparameter:

intout(0) Status: 0=Fehler, >0=OK

28. Glossar

Ein Makel der Informatik ist sicherlich, daß Sie sich mit vielen für den Laien schwer verständlichen Fachbegriffen herumschlägt. Als altem Hasen sagt ihnen der Begriff "Harddisk" wahrscheinlich etwas - aber wenn man mit der "Computerei" gerade erst anfängt, ahnt man nicht einmal, daß dieses Wort mit dem Problem zu tun haben könnte, das einen vielleicht im Moment plagt. Aus diesem Grund möchte ich in den nächsten Zeilen einige Wegweiser in die verschiedenen Abteilungen unseres gemeinsamen Hobbies aufstellen - sprich elementare Begriffe erläutern.

Nun aber endlich zu den Fachbegriffen:

A/D-Wandler

Abkürzung für Analog-Digital-Wandler, eine elektronische Schaltung, die analoge Signale in digitale (und damit per Computer nutzbare) umwandelt.

Akustikkoppler

Gerät zur Verbindung eines Computers mit dem Telefonnetz zur Datenfernübertragung.

ANSI

Abkürzung für American National Standards Institute. Das amerikanische Gegenstück zu DIN.

ASCII

Abkürzung für "American Standard Code for Information Interchange". Ein genormter -> Zeichensatz, der ursprünglich von Fernschreibern stammt. Inzwischen ist der ASCII von verschiedenen Computerherstellern weiterentwickelt worden. Leider sind diese "Weiterentwicklungen" in keiner Weise genormt, so daß fast jedes Computermodell einen eigenen Zeichensatz benutzt und deshalb bei der Übertragung von Daten Probleme entstehen können.

Assembler

Übersetzer für die vereinfachte Programmierung in -> Maschinensprache.

AT

Abkürzung für Advanced Technology, der Bezeichnung für den IBM-Personal-Computer mit dem Intel-Prozessor 80286. In den letzten Jahren auch Oberbegriff für alle Rechner mit 80286-Prozessor.

BASIC

Abkürzung für "Beginners All purposes Symbolic Instruction Code". Gemeint ist damit eine Programmiersprache, die Mitte der 60er Jahre entworfen wurde, um Anfängern das Programmieren zu erleichtern. Man kann BASIC ohne Übertreibung als die am weitesten verbreitete Programmiersprache überhaupt bezeichnen.

Baud

Einheit für die Übertragungsgeschwindigkeit von Daten.

BCD

Abkürzung für "Binary Coded Decimals". Spezielle Kodierung von Zahlen für Berechnungen. Bei der Verwendung von BCD-Zahlen werden Rundungsfehler weitgehend vermieden, dafür arbeitet eine solche Arithmetik in der Regel langsamer. Bedeutsam vor allem im geschäftlichen Bereich.

Benchmark-Programm

Programm zur Messung der Leistungsfähigkeit eines Computers.

Betriebssystem

Eine Sammlung von Steuerprogrammen, die das Zusammenspiel von Computer und Peripheriegeräten ermöglicht.

Binärsystem

Ein Zahlensystem, das nur zwei Ziffern (0 und 1) kennt. Für den Menschen sehr unübersichtlich (der Zahl 123 entspricht im Binärsystem die Bit-Folge 01111011), ist es das System, mit dem jeder Computer arbeitet. Der Vorteil: Zur Darstellung von Zahlen können Signale einfachster Art (Strom an, Strom aus) benutzt werden.

Bit

Die kleinste denkbare Informationseinheit. Sie kann nur zwischen zwei Zuständen wechseln: JA oder NEIN, was meistens durch die Zahlen 1 und 0 gekennzeichnet wird. Für die Computer sind Bits von besonderer Bedeutung, weil man sie leicht durch Schalter darstellen kann: "Strom aus" ergibt eine 0, "Strom an" eine 1.

Byte

Technisch gesehen ist ein Byte eine Gruppe von acht -> Bits. Einfacher zu verstehen ist die Tatsache, daß ein Byte die Menge von Bits ist, die nötig ist, um ein Zeichen aufzunehmen. Ein Speicher der Größe 1000 Byte kann also tausend Zeichen speichern (was ungefähr einer halben Schreibmaschinenseite entspricht). 1024 ($=2^{10}$) Bytes sind ein Kilo-Byte, was mit K oder KByte abgekürzt wird. Achtung! Das K muß groß geschrieben sein, denn ein kleines k (wie in kg) bezeichnet die Zahl 1000! 1024 KByte sind übrigens ein Megabyte (MByte), also 1048576 Zeichen.

C

Programmiersprache, die Ende der sechziger Jahre von Brian W. Kernighan und Dennis M. Ritchie an den berühmten Bell-Laboratorien entworfen wurde. C verbindet die Vorteile einer -> Hochsprache mit den Möglichkeiten der maschinennahen Programmierung. So wurde es möglich, ganze Betriebssysteme in C zu schreiben.

CAD

Abkürzung für "Computer Aided Design", also den Entwurf und die Konstruktion mit Hilfe des Computers.

CISC

Abkürzung für "Complex Instruction Set Computer". Gegenteil von -> RISC.

COBOL

Programmiersprache für Anwendungen im Geschäftsbereich.

Compiler

Computer (wenn man sie als reine Maschine betrachtet) können nur Programme in der sogenannten -> Maschinensprache verarbeiten. Nun ist das

Programmieren in dieser Maschinensprache sehr kompliziert und mitunter auch lästig. Deshalb hat man sogenannte Hochsprachen entwickelt. Beispiele dafür sind z.B. -> BASIC, -> C oder -> PASCAL, um nur die bekanntesten zu nennen. In diesen stehen für den Menschen leicht zu lesende Befehle wie PRINT (Drucke) für längere Kombinationen von Maschinenbefehlen. Ein Compiler stellt nun ein Programm dar, das in der Lage ist, aus Texten in solchen Hochsprachen Maschinenprogramme zu machen, d.h. sie so zu übersetzen, daß der Computer sie auch "verstehen" kann.

Coprozessor

Coprozessoren sind - wie der Name schon sagt - eigenständige Verarbeitungseinheiten, die die -> CPU bei bestimmten Aufgaben unterstützen. Sehr verbreitet sind sogenannte Arithmetikprozessoren, die speziell für den Umgang mit Gleitkommazahlen ausgelegt sind. Wo ein normaler Mikroprozessor für die Berechnung einer Funktion wie Sinus viele hundert Befehle benötigt, reicht einem Arithmetikprozessor schon ein einziger Befehl. Es gibt aber auch Coprozessoren für andere Aufgaben, etwa die Darstellung von Grafiken. Allen gemeinsam ist, daß sie durch Ihre Spezialisierung dem eigentlichen Prozessor viel Arbeit abnehmen und die Datenverarbeitung stark beschleunigen.

CPU

Abkürzung für "Central Processing Unit" oder -> Zentraleinheit.

Cursor

Meistens ein kleines Rechteck oder ein Strich auf dem Bildschirm, das oft auch noch blinkt. Es markiert die Stelle, an der das nächste Zeichen (das Sie per Tastatur eingeben) auf dem Bildschirm erscheint.

D/A-Wandler

Digital-Analog-Wandler, elektronische Schaltung zum Umwandeln von digitalen Informationen in analoge Signale.

Disketten

Eine runde Magnetscheibe, die beim ST einen Durchmesser von 3,5 Zoll hat und in einem harten, quadratischen Plastikgehäuse sitzt.

Diskettenlaufwerk

Das Gerät, in das Sie die -> Disketten schieben können, um darauf Daten speichern oder davon lesen zu können.

DMA

Abkürzung für "Direct Memory Access". Wird bei schnellen Massenspeichern benutzt, um Daten direkt von der Platte in den Speicher (ohne Umweg über den Prozessor) transportieren zu können.

EBCDIC

Von IBM definierter Zeichensatz für Großrechner. Weist zum Teil starke Unterschiede zum -> ASCII auf.

EPROM

Die Abkürzung für "Erasable Programmable Read Only Memory". Dabei handelt es sich um -> ROMs, deren Inhalt nicht schon bei der Herstellung feststehen muß und die (mit einigem Aufwand zwar) später auch wieder gelöscht werden können.

Festplatte

Festplatten gehören zur Kategorie der Massenspeicher. Im Gegensatz zu -> Diskettenlaufwerken kann man die Magnetplatte, auf der die Daten gespeichert werden, bei Festplatten nicht auswechseln. Dafür sind Festplatten sehr schnell und haben viel mehr Speicherplatz. Siehe auch Kapitel 2.

Firmware

Bezeichnung für Programme, die in ein ROM programmiert wurden und damit fester Bestandteil des Rechners sind.

Floppy Disk

Alte Bezeichnung für -> Disketten. Die ersten Disketten hatten noch einen Durchmesser von 8 Zoll (heute: 3,5 und 5,25 Zoll) und sahen aus wie die heutigen 5,25-Zoll-Scheiben. Das Sensationelle an Ihnen war Ihre Biegsamkeit (engl. floppy = biegsam). Genau diese Eigenschaft versucht man heute durch die verstärkten Gehäuse wieder in der Hintergrund zu drängen, weil geknickte Disketten nicht mehr funktionieren.

FPU

Abkürzung für "Floating Point Unit", zu deutsch: Arithmetikprozessor.

Friktionsantrieb

Betriebsart eines Druckers, bei dem das Papier durch zwei oder mehr gegeneinander laufende Walzen transportiert wird. Gegenteil:--> Traktorantrieb.

Harddisk

Ein anderer Begriff für -> Festplatte.

Hardware

Eine populäre Definition lautet "Alles, was man anfassen kann". Gemeint sind schlicht und einfach alle Gerätschaften, die ihren Schreibtisch überlasten. Ursprünglich ist Hardware ein amerikanischer Begriff für "Werkzeug". Gegenteil: --> Software.

Hexadezimalsystem

Ein Zahlensystem mit 16 Ziffern (0-9, A, B, C, D, E, F). Es wird vor allem benutzt, um Bytes darzustellen, denn ein Byte besteht immer aus genau 2 Hexadezimalziffern.

Hochsprache

Programmiersprachen, die sich nicht an der internen Darstellung der Computer orientieren, sondern an den Sprachgewohnheiten des Menschen. Programme in Hochsprachen müssen durch -> Compiler oder -> Interpreter erst übersetzt werden, ehe sie ausgeführt werden können.

Initialisierung

Startphase, in der ein Gerät oder Programm in einen bestimmten Zustand gebracht wird.

Integer

Engl. Bezeichnung für ganze Zahlen.

Interface

Fachbegriff für -> Schnittstelle.

Interpreter

Auch Interpreter können wie -> Compiler -> Hochsprachen in -> Maschinensprachen übersetzen. Allerdings gibt es da einen gewichtigen Unterschied. Ein Compiler übersetzt erst, dann kann der Computer das entstandene Programm selbsttätig ausführen, er braucht den Compiler dazu nicht mehr. Ein Interpreter übersetzt das Programm Stück für Stück, während er die (Hochsprachen)-Befehle gleichzeitig ausführt. Der Nachteil: Interpreter sind generell langsamer. Wenn ein Programm 100 PRINT-Befehle ausführen soll, muß der Interpreter den Hochsprachentext auch 100mal analysieren. Ein Compiler übersetzt nur einmal und läßt die entstandene Folge von Maschinenbefehlen 100mal ausführen. Interpreter sind nur für einige Sprachen erhältlich, z.B. (und vor allem) für -> BASIC und FORTH.

ISO

"International Standards Organisation", der internationale Zusammenschluß von Normungsbehörden.

Kompatibilität

Wörtlich: "Vereinbarkeit". In der Informatik bedeutet dies, daß Geräte und/oder Software so ausgelegt sind, daß sie mit Erzeugnissen anderer Hersteller zusammenarbeiten können. Soweit die allgemeine Definition. Im täglichen Sprachgebrauch wird aber mit dem Wort "kompatibel" oft noch viel mehr gemeint. PC-Kompatible sind Computer, die dem IBM-PC soweit nachempfunden wurden (manche Insider sprechen auch von "Clonen" oder schlicht "Abkupfern"), daß sie in der Lage sind, alle Programme für den IBM-Computer abzuarbeiten und darüber hinaus auch noch einen Großteil der Zusatzhardware zu akzeptieren. Kompatibel kann aber auch heißen, daß Programme oder Programmiersprachen so gestaltet sind, daß sie ohne Schwierigkeiten auf andere Computersysteme übertragen werden können.

Laserdrucker

Spezielle Drucker, die mittels Lasertechnik eine lichtempfindliche Trommel belichten und das so entstehende Bild auf Papier übertragen (genau wie bei einem Kopierer). Laserdrucker bieten eine besonders hohe Druckqualität, sind aber auch ziemlich teuer.

Maschinensprache

Jeder Computer kann nur speziell für diesen Typ bestimmte Sprachen, die Maschinensprachen, verstehen. Sie bestehen aus einer Reihe von Zahlen, in denen die Befehle eines Programms kodiert sind. Diese können besonders schnell ausgeführt werden, sind aber schwer zu verstehen. Das Gegenstück dazu sind die sogenannten -> Hochsprachen.

Massenspeicher

Speicher, der in einen Computer eingebaut wird, muß schnell sein. Schneller Speicher ist teuer, und deshalb meistens klein. Nun möchte man aber auch große Datenmengen dauerhaft aufbewahren und speichert Sie deshalb auf Magnetbändern, -> Disketten oder -> Festplatten ab. Diese Geräte sind langsam, haben aber viel mehr Platz und heißen deshalb Massenspeicher.

Mikroprozessor

Der integrierte Schaltkreis in jedem PC, der die eigentlichen Datenmanipulationen vornimmt. Im ST ist ein Mikroprozessor vom Typ 68000 eingesetzt.

Modem

Modem ist ein Kunstwort aus MODulator/DEModulator. Es ist ein Gerät zum Anschluß eines Computers an das Telefonnetz zur Datenübertragung. Im Gegensatz zu den meisten -> Akustikkopplern sind Modems leistungsfähiger (Wahlmöglichkeit etc.).

Monitor

- a) Das Bildschirmgerät, auf dem der Computer Daten anzeigt.
- b) Bestimmte Programme, die einen Einblick in interne Abläufe der -> Hardware und Software erlauben.

MS-DOS

Microsoft Disk Operating System, ein Betriebssystem für Rechner mit Intel-Prozessoren, die kompatibel zum 8086 sind (IBM-PC-Kompatible). Läuft auch auf dem ST, wenn ein spezieller Emulator benutzt wird.

Pascal

Eine weitere Programmiersprache, die von Niklaus Wirth an der ETH Zürich entworfen wurde. Pascal wurde speziell für den Lehrbetrieb an der Universität entworfen, hat aber seit ca. zehn Jahren wachsende Bedeutung in der kommerziellen Programmierung erhalten.

PC

Ursprünglich "Personal Computer", also persönlicher Computer. Seit dem Erscheinen des IBM-PC Oberbegriff für die Klasse der -> Kompatiblen.

Peripheriegerät

Jedes Gerät, das mit der Ein- oder Ausgabe von Daten zu tun hat. Beispiele: -> Diskettenlaufwerk, Drucker, Tastatur, Bildschirm.

Pixel

Kleinster Bildpunkt bei Grafikdarstellung.

Plotter

Ein Gerät, das, wie der Mensch, mit Stiften auf Papier zeichnet - allerdings unter Computer-Regie und viel schneller und genauer.

Portabilität

Gemeint ist die Eigenschaft einiger Programme und Programmiersprachen, auf möglichst vielen Computern ohne Änderungen lauffähig zu sein. Im Normalfall ist es leider so, daß die meisten Programme auf bestimmte Hardware-eigenschaften eines Computers zurückgreifen und deshalb nur auf diesem speziellen Typ lauffähig sind.

Prozessor

Das eigentliche Gehirn des Computers (wenn man bei Computern von Gehirn sprechen kann). Der Prozessor ist ein elektronisches Bauteil, das logische und arithmetische Operationen ausführen kann.

RAM

Abkürzung für "Random Access Memory", was einfach "Speicher mit wahlfreiem Zugriff" heißt. Wahlfreier Zugriff bedeutet, daß jeder Teil des Speichers

gleich schnell und einfach zu erreichen ist. In der Praxis werden damit die Speicher bezeichnet, die direkt in den Computer eingebaut werden. Man kann sowohl Daten aus ihnen lesen, als auch neue Daten hineinschreiben. RAMs verlieren aber ihren Inhalt, wenn man den Strom abschaltet.

RISC

Abkürzung für "Reduced Instruction Set Computer". Damit ist nicht ein besonders dummer Computer gemeint, sondern Prozessoren, die nur sehr wenige Maschinenbefehle verstehen, diese dafür aber besonders schnell ausführen. RISC ist eine recht neue Technologie, die zunehmend an Bedeutung gewinnt.

ROM

Abkürzung für Read Only Memory (Nur-Lese-Speicher). Im Gegensatz zum -> RAM kann man in ROMs keine neuen Daten einschreiben. Dafür behalten sie ihren Inhalt auch dann, wenn man den Strom abschaltet. Sinn der Sache ist, einen Speicher zu haben, in den unveränderliche Daten geschrieben werden, die auch direkt nach dem Einschalten des Computers (wenn das RAM noch gar keinen Inhalt haben kann) bereits zur Verfügung stehen.

Schnittstelle

Eine Schnittstelle ist im Grunde genommen nichts anderes als ein Anschluß für ein -> Peripheriegerät. Dabei muß nicht jede Schnittstelle für den Benutzer in Form eines Steckers sichtbar sein. Auch zwischen Tastatur und Prozessor gibt es eine solche Schnittstelle.

Software

Das Gegenteil von -> Hardware - gemeint sind die Programme. Das bezieht sich allerdings nur auf das geistige Produkt. Die Disketten, auf denen Ihre Programme gespeichert sind, gehören zur Hardware!

ST

Abkürzung für Sixteen-Thirtytwo. Gemeint ist unser gemeinsamer Lieblingsrechner. Die Bezeichnung stammt von der Arbeitsweise des Mikroprozessors 68000, der intern mit 32-Bit-Daten hantiert, nach außen aber nur 16 Bit in einem Arbeitsgang weitergeben kann.

Traktor

Das hat nichts mit der Anwendung von Computern in der Landwirtschaft zu tun. Gemeint ist ein Aufsatz für einen Drucker, der das berühmte-berüchtigte Computerpapier mit der beidseitigen Perforation durch Stachelwalzen transportiert. Das Gegenteil ist der -> Friktionsantrieb.

UNIX

Betriebssystem für leistungsstarke Computer; ursprünglich von AT&T.

Winchester

Noch ein Begriff für -> Festplatte. Er stammt von der ersten überhaupt gebauten Festplatte, die (wohl eher zufällig) die gleiche Typenbezeichnung wie ein Winchester-Gewehr trug. Waffenkenner unter den Ingenieuren haben dem neuen Gerät daher sofort den nicht gerade friedlichen Spitznamen Winchester gegeben.

Zeichensatz

Das Repertoire von Buchstaben, die der ST auf seinem Bildschirm anzeigen kann.

Zentraleinheit

Teil des Computersystems, in dem die eigentliche Datenverarbeitung stattfindet. Die Zentraleinheit umfaßt den -> Prozessor, den Speicher und -> Schnittstellen. Das Gegenteil von Zentraleinheit ist ein -> Peripheriegerät.

Stichwortverzeichnis

Alternate und Help	62
Alternate	187
Caps Lock	186
Clr/Home	189
Ctrl-Alternate-Delete	189
Delete	188
Esc	44
Insert	189
Shift	34, 186
Undo	189
40-Ordner-Problem	83
6301	781
6850	747
8086/8088	707
AC_CLOSE	310
AC_OPEN	310
ACC	88
Accessories	20, 48, 309
ACIA	747
Additive Farbmischung	504
ADDRESS-MARK-DETECTOR	405
Addrin	1007
Addrout	1007
Adresse eines Objekts	336
Adreßregister	708
AES	287, 295
AHDI.PRG	103
Akustikkoppler	196
Aladin	237, 249
Algorithmus	265
AMC-GDOS	628
Anwendung anmelden	59
Anwendungsprogrammen	217
APP	87
Appl_exit	301, 1020
Appl_Id	310
Appl_init	300, 301, 1020
APPL_READ	1021
APPL_TPLAY	1022

APPL_TRECORD	1021
APPL_WRITE	1022
APPLBLK	317
Applikationsnummer	310
Arbeit sichern	61
Arbeitsbereich ermitteln	325
Arbeitsfläche beschreiben	328
Arbeitskopie	121
Arbeitsspeicher	71
ARC	595, 971
ASCII	143
Assembler	265, 271
ASSIGN.SYS	629
Attribut-Funktionen	560
Auflösung des Bildschirms	513
AUTO-Ordner	20, 23, 108
AUTOEXEC.BAT	242
Automatisch startende Programme	350
Backup	121
BAK	87
BAS	88
BASEPAGE	289
BASIC	266
Basic Input/Output System	287
Baudrate	195, 202
Bconin	873
Bconout	873
Bconstat	872
Bcostat	878
Betriebssysteme	236, 287
Bflags	877
Bildfrequenz	719
Bildschirm	685
Bildschirmfenster	652
Bildschirmschoner	691
Bildwiederholfrequenz	181, 690
BIOS	287, 871
Bioskeys	905
Bit-Map	229
BITBLK	317
BitBlit	763, 939
Blitter	285, 762

Block	221
Blocksatz	218
Blockschaltbild des WD1772	400
Bomben	20
Boot-Sektor	257
BPB	374, 384, 877
BPB-Analysator	385
Bss	396
Buchstaben verkleinern und vergrößern	620
Bus-frei-Phase	449
Buttons	67
C	88, 264, 270
CAD-Programme	230
CAUXIN	823
CAUXIS	832
CAUXOS	832
CAUXOUT	823
CCONIN	821
CCONIS	829
CCONOS	831
CCONOUT	822
CCONRS	828
CCONWS	827
CELL ARRAY	969
Centronics	133
CGA-Modus	241
CHECKED	314
Checksumme	373
Cinch-Buchsen	193
CIRCLE	595, 601, 973
CLEAR WORKSTATION	965
CLIPPING	653
Clockvec	913
CLOSE VIRTUAL SCREEN WORKSTATION	964
CLOSE WORKSTATION	963
CLOSER	325
Csiz	877
Cluster	119, 394
CNECIN	826
COLOR	565
Colorptr	927
Compiler	264

Composite-Signal	181
CON	1064
Conterm	822, 928
CONTOUR FILL	586, 952, 970
CONTROL.ACC	48, 140
Controller	812
COPY RASTER FORM	949
COPY RASTER, OPAQUE	658
COPY RASTER, TRANSPARENT	658, 990
COPY RASTER, OPAQUE	988
CP/M	237
CPRNOS	831
CPRNOUT	824
CR/LF	148
CRAWCIN	826
CRAWIO	824
CRC-Logic	405
Critical Error Handler	923
Cursconf	903
Cursor	616
Cursor im XOR-Modus	546
Cursor-Steuerung	919
Cursor-Tasten	189
Cursor-Tasten-Simulationsmodus	792
DATA SEPARATOR	407
Data-Segment	396
Data-Shift-Register (DSR)	402
Datei drucken	29
Datei in den Ordner hineinkopieren	40
Datei verschieben	76
Datei-Attribute	854
Datei-Header	290
Dateiattribut	847
Dateien anzeigen	28
Dateien kopieren	38, 75
Dateien löschen	38, 76
Datenaustausch	213
Datenbanken	225
Datenfeldern	227
Datensatz	227
Datensicherung	121
Datrec	877

Datum	49, 72
DCREATE	843
DDELETE	844
DEFAULT	313
DEFFILL	570
Definition eines eigenen Füllmusters	575
DEFLINE	565
DEFMARK	521
DEFTEXT	651
DEGAS	232
Desktop	19, 287
Desktop erscheint nicht	21
DESKTOP.INF	24, 61
DFREE	842
DGETDRV	833
DGETPATH	857
Diab630.PRG	161
Diablo	157
Diagnose-Cartridges	807
Dialogboxen	67, 335
DIN-Buchsen	194
DIP-Schalter	138
Disk Transfer Adress	392
Disk-Controller	372
Diskette	89
Diskette formatieren	94
Diskette löschen	76
Disketten kopieren	38, 95
Diskettenformat	214, 371
Diskettenlaufwerk	210, 276
DMA	157, 711
DMA-Port	102
DOC	88
DOODLE-Bilder	232
Doppelklick	50
Dosound	701, 910
Draft-Qualität	136
DRAW	558
DRAW SPRITE	948
Drop-Down-Menüs	47
Drucken in eine Datei	225
Drucker	131, 278
Drucker-Anpassung	48

Druckerkabel	199
Druckertreiber	157
Drvmap	879
DSETDRV	830
DSETPATH	845
DTA	392
Dump_vec	931
Duplex	202
Echtzeituhr	285, 777, 781
Editor	218, 616, 642
Eingabe-Array	295
Eingabefelder editieren	336
Eingabezeile	68
Eingebranntes Bild	691
Einseitige Disketten	92
Einzelblatteinzug	135
ELLIPSE	595, 601, 974
ELLIPTICAL ARC	595, 973
ELLIPTICAL PIE	595, 974
EMULATOR.ACC	48
Emulatoren	236
End_os	931
Endlospapier	134
Endung	86
Environment	863
Ereignisse	318
Ersetzen-Funktion	222
Erweiterungs-Bus	817
ESC	188
ESC/P	137
Etv_critc	923
Etv_term	923
Etv_timer	923
Etv_xtra	924
EVNT-Gruppe	339
EVNT_BUTTON	1014
EVNT_DCLICK	1015
EVNT_KEYBD	1015
EVNT_MESAG	1017
EVNT_MOUSE	1016
EVNT_MULTI	1017
EVNT_TIMER	1017

Exceptions	294, 711
EXCHANGE BUTTON CHANGE VECTOR	1000
EXCHANGE CURSOR CHANGE VECTOR	1001
EXCHANGE MOUSEMOVEMENT VECTOR	1000
EXCHANGE TIMERINTERRUPT VECTOR	998
Exec_os	931
EXtended BIOS	287
EXTENDED INQUIRE FUNCTION	1002
Extensions	86
Farbeinstellung	525
Farben	503, 523, 719
Farbindizes des VDI	526
Farbmonitore	690
Farbpalette	536
FAT	76, 105, 393
Fatrec	877
FATTRIB	854
FCLOSE	849
FCREATE	846
FDATIME	868
FDC	400
FDELETE	852
FDUP	856
Fehlermeldungen	113
Fehlernummern	118
Fenster	42, 318
Fenster aktivieren	45
Fenster berechnen	326
Fensterelemente	324
Festplatte	19, 25, 102, 210
Festplatte unter DOS	244
FFORCE	856
FGETDTA	840
File	71
File Allocation Table	393
File-Name	390
File-Select-Box	63
FILL	587
FILL RECTANGLE	577, 970
FILLED AREA	566, 968
FILLED POLYGON	938
FILLED RECTANGLE	938

FILLED ROUNDED RECTANGLE	577
FILLED ROUNDEDRECTANGLE	975
Finder	249
Flächen ausfüllen	592
Flattersatz	218
Fließtext	223
Flock	925
Flopfmt	891
Floppy-Disk-Controller WD 1772	722
Flopdr	888
Flopver	901
Flopwr	890
FNT	88
Font	627
FOPEN	848
FORCE INTERRUPT	442
FORM_ALERT	1030
FORM_BUTTON	1031
FORM_CENTER	1032
FORM_DIAL	336, 1032
FORM_DO	336, 1033
FORM_ERROR	1034
FORM_KEYBD	1034
Format Drive	458
FORMAT TRACK	431
Formatieren	92, 94, 104, 371
Formatierung der Harddisk	104
FORTH	265
FORTRAN	264
FREAD	850
Freeware	217
FRENAME	867
Frequenz	696
FSEEK	853
FSEL_INPUT	1030
FSETDTA	833
FSFIRST	865
Fsiz	877
FSNEXT	867
FULLER	325
Funktionstasten	186
FWRITE	851

G PLUS+	628
G_BOXCHAR	313
G_BOXTEXT	312
G_BUTTON	313
G_FBOXTEXT	313
G_FTEXT	313
G_IBOX	312
G_ICON	313
G_IMAGE	312
G_PROGDEF	312
G_STRING	313
G_TEXT	312
G_TITLE	313
GDOS	287, 628
GEM	88, 287
GEMDEFS.H	658
GEMDOS	287, 821
GET	671
GET PIXEL	936, 991
Getbpb	877
Getmpb	871
Getrez	886
Gettime	904
GFA-BASIC und Assembler	363
Giaccess	700, 907
GIOS	287
Global	295, 1007
Glocke	50
GLUE	706, 711
GRAF_DRAGBOX	1039
GRAF_GROWBOX	1040
GRAF_HANDLE	1038
GRAF_MKSTATE	1044
GRAF_MOUSE	1043
GRAF_MOVEBOX	1040
GRAF_RUBBERBOX	1038
GRAF_SHRINKBOX	1041
GRAF_SLIDEBOX	1043
GRAF_WATCHBOX	1042
Grafik-Auflösung	720
Grafikmodus	539
Grafikprogrammierung	499
GRAPHMODE	521, 539

Handshaking	195
Hardcopy	62
Harddisk	102
HDC-Tools	462
Hdv_boot	927
Hdv_bpb	927
Hdv_init	927
Hdv_mediach	928
Hdv_rw	927
HDX.PRG	104
Head-Crash	113
HGR	499
HIDE CURSOR	947, 999
HIDETREE	314
High water mark	894
Hintergrundrettung	672
Hochaufgelöste Grafik	499
Hochkapazitätsdisketten	244
HORIZONTAL LINE	937
Horizontale Ablenkfrequenz	181
Horizontale Schieber	322
HSLIDE	325
Hüllkurve	697
Icon bewegen	36
Icon-Gruppen	34
ICONBLK	316
Icons	19, 24
lkbdsys	913
lkbdws	905
Import	227
Index	56
INDIRECT	314
INF	88
Inhaltsverzeichnis	390
Initialisierung	935
Initiator	812
INPUT CHOICE, REQUEST MODE	995
INPUT CHOICE, SAMPLE MODE	996
INPUT LOCATOR, REQUEST MODE	993
INPUT LOCATOR, SAMPLE MODE	993
INPUT STRING, REQUEST MODE	996
INPUT STRING, SAMPLE MODE	997

INPUT VALULATOR, REQUEST MODE	994
INQUIRE CURRENT FILL AREA ATTRIBUTES	1005
INQUIRE CHARACTER CELL WIDTH	1006
INQUIRE COLOR REPRESENTATION	1003
INQUIRE CURRENT GRAPHIC TEXT ATTRIBUTES	1005
INQUIRE CURRENT POLIMARKER ATTRIBUTES	1004
INQUIRE CURRENT POLYLINE ATTRIBUTES	1004
INQUIRE TEXT EXTEND	1006
INSERT VALULATOR, SAMPLE MODE	994
INTEL	707
Interface	193
Interpreter	264
Interrupt Pending Bit	737
Interrupts	740
Intin	295, 1007
Intout	295, 1007
lorecl	894
Janus-Disketten	252
Jdisint	906
Jenabint	906
Joystick	191
Joysticks	781
Joyvec	913
JUSTIFIED GRAPHICS TEXT	633, 976
Kabel	194
Kbdvbase	912
Kbrate	914
Kbshift	880
KEYBGR	246
Keytbl	897
Kontrollfeld	180
Kopieren	217, 655
Korrespondenz-Qualität	136
Kreise	593
Landezone	112
Laserdrucker	132, 155
LASTOB	314
Laufwerk B	98
Lautstärke	695
Lesen von Sektoren	874

Lexika	220
Lichtgriffel	193
LINE	558, 936
Line A-Variablen im Überblick	953
LINE-A	935
Linie	552
Linien-End-Typen	562
Linfendicke	562
Linke Maustaste	191
LISP	265
Lissajous-Kurven	508
LOAD FONTS	628, 630, 965
Logbase	885
LOGO	265
Low water mark	894
Lüftungsschlitze	273
Macintosh-Emulator	249
Macintosh-Format	249
Malloc	293, 858
Malloc()	665
Markbad	106
Maschinensprache	265
Matrixdrucker	131
Maus	190, 274, 781, 785
MC68000	707
Mediach	879
Mega ST	285
Mega-Bus	817
Megafile 30	102
Memctrl	924
Memory Form Definition Blocks	657
Memory-Parameter-Block	871
Memval2	925
Memval3	932
Memvalid	924
MENU_BAR	339, 1035
MENU_ICHECK	339, 1035
MENU_IENABLE	339, 1036
MENU_REGISTER	1037
Menu_Register()	310
MENU_TEXT	339, 1037
MENU_TNORMAL	1036

Menüleiste	47
Message-Buffers	320
META.SYS	629
Metafile	231
MFDB	657, 658
MFP 68901	730
Mfpint	893
Mfree	293, 859
Mfree()	667
MIDI	208
Midisys	913
Midivec	913
Midiws	892
MINIX	255
MMU	706, 711
Mode Select	461
Modem	201
Modulator	182
Modulschacht	806
Monitore	179
MOTOROLA	707
Mousevec	913
MOVER	325
MS-DOS	238, 247
MS-DOS-Format	242
MSHRINK	860
Musik	695
Nadeldrucker	131
NDC	500
NEO	89
Netzwerke	196
Nicht maskierbarer Interrupt	710
NLQ	136
NMI	710
Nullmodem	206
Numcl	877
Nvbis	926
OBJC_ADD	1023
OBJC_CHANGE	1023
OBJC_DELETE	1024
OBJC_DRAW	1024

OBJC_EDIT	1025
OBJC_FIND	1026
OBJC_OFFSET	1026
OBJC_ORDER	1027
Objektbäume	311
Offgibit	908
Ongibit	909
OPEN VIRTUAL SCREEN WORKSTATION	963
OPEN WORKSTATION	961
Open_vwork	306
Ordner	26, 77
Originaldisketten	121
OUTLINED	314
Overlay	859
Paddles	193
Parallele Schnittstellen	197
Parität	202
Park-Spur	112
PARMBLK	318
Partition	105
PASCAL	264
Patches	251
PBOX	579
PC-DITTO	237
PCIRCLE	601
PELLIPSE	602
PEXEC	289, 862
Pfad	83
Pfeil	562
Pfeiltasten	189
Pflege und Wartung	273
Physbase	885
Phystop	924
Plane	686
PLOT	506, 508
Plotter	132
POLYFILL	570
Polygone	552
POLYLINE	552, 967
POLYMARK	522
POLYMARKER	506, 510, 967
PRBOX	579

Prekompensation	729
PRINT	608
PRINTF	608
Program-Counter	708
Programm-Header	396
Programmaufbau	395
Programmiersprachen	217, 264
Programmierung des FDC	727
Proportionalschrift	627
Protobt	898
Prozessor	707
Prozessor-Bus	817
Prt_stat	931
Prt_vec	931
Prtblk	914
Pterm	293, 821, 864
PTERMRES	841
Ptsin	295
Ptsout	295
Public-Domain-Software	217
Pull-Down-Menüs	338
Pun_ptr	932
Punkt setzen	506
Puntaes	916
PUT	671
PUT PIXEL	935
Radio-Buttons	69
RAM-Disk	25, 99, 473
Random	898
Rasterbefehle	677
Rauschen	699
RBOX	579
RBUTTON	314
RC	500
RCS	335
Rdlen	877
READ ADDRESS	427
READ SECTOR	420f, 727
READ TRACK	429, 727
Rechte Maustaste	36, 191
Rechtecke	577
Recsiz	877

REDRAW-Routine	324
Register	716, 725
Register des 2149	758
Register des 68000	708
Register des 6850	749
Register des Blitters	764
Register des MFP	734
Reloizierung	291
Reparaturen	283
Replace	539
Request Sense	458
Reset-Phase	449
ResetVektor	924
Resource	87
Resource-Contruction Sets	335
RESTORE	414, 458
Resvalid	924
Resvector	924
Reverse Transparent	539
RGB	181
RND()	677
ROM einbauen	280
ROM-Cartridges	806f
ROM-Modul	25, 211
ROUNDED RECTANGLE	577, 975
RS232-Schnittstelle	803
RSC	87
Rsconf	895
RSRC_FREE	1027
RSRC_GADDR	336, 339, 1027
RSRC_LOAD	335, 339, 1028
RSRC_OBFIX	1029
RSRC_SADDR	1029
Rts/Cts	203, 207
Rwabs	874
 SAMPLE KEYBOARD KEY INFORMATION	 1001
SAMPLE MOUSE BUTTON STATE	999
Sav_context	929
Save_row	929
Savptr	928
SC1224	179
Scan-Codes	797

SCART-Buchse	179
Schieber	44
Schließfeld	42
Schnittstellen	193, 781
Schnittstellenkabel	133
Schreib-Operationen	408
Schreiben von Sektoren	874
Scrdmp	902
Screenpt	927
SCR_P_READ	1045
SCR_P_WRITE	1045
Sdump.PRG	175
Search first	392
Search next	392
Sector-Register (SR)	402
SEEK	415, 461
Seek to Shipping-Position	461
Seekrate	925
Seitenlänge	147
Sektoren	94, 372
SELECTABLE	313
SELECTED	314
Serielle Schnittstelle	833
SET CHARACTER BASELINE VECTOR	983
SET CHARACTER BASELINE VEKTOR	633
SET CHARACTER CELL HEIGHT	622
SET CHARACTER CELL HEIGHT, POINT MODE	983
SET CHARACTER HEIGHT	622
SET CHARACTER HEIGHT, ABSOLUTE MODE	982
SET CLIPPING RECTANGLE	652
SET CLIPPING RECTANGLE OP	966
SET COLOR REPRESENTATION	526
SET COLORREPRESENTATION	978
SET FILL COLOR INDEX	566
SET FILL COLORINDEX	987
SET FILL INTERIOR STYLE	566, 986
SET FILL PERIMETER VISIBILITY	566, 987
SET FILL STYLE INDEX	566
SET FILL STYLEINDEX	987
SET GRAPHIC TEXT ALIGNMENT	633, 985
SET GRAPHIC TEXT COLOR INDEX	984
SET GRAPHIC TEXT SPECIAL EFFECTS	622, 985
SET INPUT MODE	992

SET MOUSE FORM	997
SET POLYLINE COLOR INDEX	561, 978
SET POLYLINE END STYLES	561, 980
SET POLYLINE LINE TYPE	561, 979
SET POLYLINE LINE WIDTH	561, 980
SET POLYMARKER COLOR INDEX	982
SET POLYMARKER HEIGHT	981
SET POLYMARKER TYPE	981
SET TEXT FACE	630, 984
SET USER DEFINED FILL PATTERN	566, 988
SET USER-DEFINED LINE STYLE PATTERN	561, 979
SET WRITING MODE	538, 977
SETCOLOR	537, 888
Setexec	876
SETPALETTE	536, 887
Setprt	911
Setscreen	886
Settime	354, 904
Setup630.PRG	161
SGET	670
SH204	102
SH205	102
SHADOWED	314
Shareware	217
SHEL_FIND	1046
SHEL_GET	1046
SHEL_PUT	1046
SHEL_READ	1047
SHEL_WRITE	1047
Shifter	690, 706, 711
SHIP.PRG	112
SHOW CURSOR	998
SHOW MOUSE	947
SIMM-Module	286
SIZE-Box	323
SIZER	325
SLM605	155, 157, 158
SLM804	157f, 160
SLM804.HEX	175
SLMC804	160
SM124	179
SMM804	138
SNAPSHOT.ACC	235

Software	217, 287
Software End-of-Interrupt	737
Sound	695
Sound-Programmierung	695
Sprites	671, 673
Spur Null	727
SPUT	671
Ssbrk	884
Sshiftmd	926
ST-Book	286
Stack	292
Stackpointer	708
Stacy	286
Status-Meldungen	795
Status-Register (STR)	403
Statvec	913
STE	285
STEP	415
Stepping Rate	412
Stereo-Ton	193
Struktogramme	231
SUB-D-Verbinder	195
SUPER	834
SUPERVISOR MODE	708
Supexec	916
SVERSION	840
Swv_vec	927
Symboltabelle	291
Synchronisation	603
SYS	88
Systemdatei	847
Systemdiskette	24
Systemvariablen	923
Tail-Index	894
Tastatur	274, 747
Tastaturbelegung	905
Tastaturmatrix	784
Tastaturprozessor	185, 781, 788
TEDINFO	315
TEXT	640, 651, 968
Textattribute	651
Textausrichtung	641

TEXTBLT	942
Text-Cursor	611
Textformatierung	633
Textverarbeitung	218
TGETDATE	836
TGETTIME	838
The_env	929
Themd	928
Thermodrucker	131
Tickcal	876
Timer	740
Tintenstrahldrucker	131
Tools	217
TOS	86, 287
TOS-Programme	289
TOUCHEXIT	314
TPA	289
Track-Register (TR)	402
Trackballs	191
Tracks	371
Traktor	135
TRANSFORM FORM	658, 991
TRANSFORM MOUSE	947
Transparent	539
Transparent-Modus	540
Transport einer Festplatte	112
Trap #1	294
TRAP #2	295
TREE	79
Treiber	22
Treiberdiskette	103
Trennhilfe	219
TSETDATE	837
TSETTIME	839
TTP	86
Tuning	283
TXT	87
Typenraddrucker	132
Übertragungsgeschwindigkeit	195
Uhrzeit	49, 72
Umbenennen von Ordnern	82
Umlaute	246

Umweltpapier	135
UNDRAW SPRITE	948
Universal Synchronous/Asynchronous Receiver/Transmitter	743
UNIX	256
UNLOAD FONTS	630, 966
USART	743
User Byte	710
USER MODE	708, 710
V.24	133
V_clrwk	307
V_oprvwk	301
VBL-Routine	693
Vblsem	926
VDI	287, 295, 961
Vektororientiert	229
Verify	925
Verknüpfungsarten	659
Versteckte Datei	847
Verzeichnis	77
Verzögerung	50
Viren	257
Virtual Device Interface	287
Virtuelles Laufwerk	98
Vkbderr	913
Vmiderr	913
Volume-Label	847
Voreinstellung	60
Vorkompensation	729
Vsf_bar	328
Vsf_color"()	328
Vsf_interior	328
Vsf_style()	328
VSLIDE	325
Vsync	915
VT52-Emulator	919
Warmstart	189
Wave-Befehl	697
WD1772	400
Wechselplattenlaufwerk	102
WIND CREATE	319
WIND OPEN	319

Wind_calc	326, 1013
WIND_CLOSE	1010
Wind_create	326, 1009
WIND_DELETE	1011
WIND_FIND	1012
Wind_get	325, 1008
Wind_open	326, 1010
WIND_SET	320, 327, 1011
WIND_UPDATE	1013
Windows	318
WM_CLOSED	321
WM_FULLED	321
WM_HSLID	322
WM_MOVED	323
WM_NEWTOP	323
WM_SIZED	323
WM_TOPPED	320
WM_VSLID	322
WRITE TRACK	431, 727
WRITE-SECTOR	422, 423
XBIOS	287, 883
XBIOS(39)	916
Xbtimer	909
Xon/Xoff	203, 207
XOR	539
XOR-Modus	541, 672
YM-2149	755
Zeichendehnung	641
Zeichengröße	621
Zeichenmodi	538
Zeichensatz	627
Zeichensätze	143
Zweiseitige Disketten	92

Lassen Sie sich nicht unter Druck setzen!

Mit dem großen ST-Druckerbuch kann eigentlich nichts mehr schiefgehen. Beginnend mit der einfachen Installation des Druckers beschreibt dieser Band alles, was Sie bei der Arbeit mit Ihrem

Drucker beachten müssen. Sei es die Druckersteuerung über TOS und GEM, die Druckeranpassung der verschiedenen Anwendungsprogramme oder die Arbeit mit unterschiedlichen Fonts. Daneben finden Sie einen Nachschlageteil, der Ihnen bei auftretenden Pannen weiterhilft, und eine Diskette mit vielen nützlichen Utilities für eine komfortable Druckersteuerung.

Wann immer einmal

etwas beim Drucken schiefläuft, dieses Buch sagt Ihnen, woran es liegt und wie Sie den Schaden beheben. Das große ATARI-ST-Druckerbuch – mit zahlreichen nützlichen Utilities auf der mitgelieferten Diskette (beispielsweise, um aus dem CLI heraus Batch-Dateien aufzurufen, zum Umlenken der Zeichen von ATARI- auf IBM-Zeichensatz oder ein Hardcopyprogramm für 24-Nadeldrucker).



Jungbluth/Ockenfelds
Das große ATARI-ST-Druckerbuch
Hardcover, inkl. Diskette
572 Seiten, DM 59,-
ISBN 3-89011-362-1

Für alle Schön- und Vielschreiber.

Das große Signum!-Buch – und Sie haben das Zeug, diese Textverarbeitung voll auszunutzen. Ob Diskettenoperationen, TextEinstellungen, Fuß-

notenverwaltung, Spaltensatz oder Bildoperationen – mit diesem Buch lernen Sie Signum/SignumII von seiner besten Seite kennen. Dabei beschreibt der Autor das umfassende Thema Druckerausgabe selbstverständlich besonders detailliert. Vom Druckfonteditor bis zum perfekten Ausdruck sagt er Ihnen, worauf Sie besonders achten müssen. Das große Signum!-Buch – alle die

hervorragenden Features von Signum! werden in diesem Band beispielhaft erklärt. Er macht Sie Schritt für Schritt zum Signum!-Profi.



Kraus
Das große Signum!-Buch
346 Seiten
DM 39,-
ISBN 3-89011-313-3

Beste Ergebnisse. Mit dem ATARI und Calamus.

Bestes Computerwissen allein ist kein guter Bürge für perfekte DTP-Ergebnisse. Auf der einen Seite sollte man alle Funktionen und Möglichkeiten des entsprechenden Programms kennen und beherrschen, aber natürlich

auch in Sachen Typografie und Gestaltung auf der Höhe sein. Was Calamus angeht, gibt es ein Buch, das Ihnen all dieses Wissen vermitteln kann: Das große Buch zu Calamus. Im einzelnen werden behandelt: Installation, Benutzerführung, Text- und Grafikrahmen, Druckertreiber und Scanner. Zahlreiche Anwendungsbeispiele zeigen Ihnen dabei die Leistungsfähigkeit des



Programms und geben Ihnen ein sicheres Gespür für eine ansprechende Gestaltung. Im Anhang nochmals die wichtigsten Informationen für einen schnellen Zugriff. Das große Buch zu Calamus – für beste Ergebnisse.

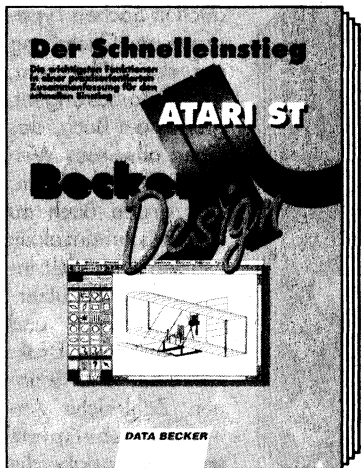
Kraus
Das große Buch zu Calamus
392 Seiten, DM 39,-
ISBN 3-89011-346-X

BeckerDesign im Handumdrehen: der Schnelleinstieg

Design muß sein, umständliche theoretische Erklärungen nicht: Mit dem Schnelleinstieg BeckerDesign ST nutzen Sie sofort die interessantesten Funktionen des vielseitigen Programms. Anhand von prakti-

schen Beispielen arbeiten Sie direkt mit den verschiedenen Funktionsbereichen und Werkzeugen. So erstellen Sie nacheinander eine architektonische Zeichnung, eine Konstruktionszeichnung, eine Illustration und eine Werbeseite. Außerdem lernen Sie, wie eine Funktionsgrafik gestaltet wird und können natürlich die wichtigsten Informationen im Anhang des Buchs nachschlagen

(wie beispielsweise die Tastenbelegung). Der Schnelleinstieg BeckerDesign: Ihre Garantie für schnelle Erfolge mit dem attraktiven Programm.



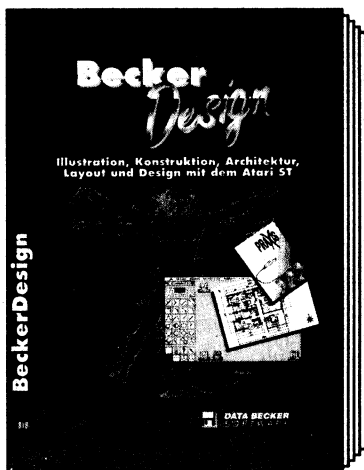
**Der Schnelleinstieg
BeckerDesign ST
155 Seiten, DM 19,80
ISBN 3-89011-757-0**

BeckerDesign ST: So verwirklichen Sie Ihre Ideen!

Illustration, Konstruktion, Architektur, Layout und Design – alles mit nur einem Programm: Das ist BeckerDesign ST, die starke Software, mit der Sie Ihre Ideen verwirklichen. Durch BeckerDesign ver-

binden Sie mathematische Strenge mit künstlerischer Freiheit – einschließlich der Freiheit, sich das Programm auf Ihre Bedürfnisse zuzuschneiden. BeckerDesign ST unterstützt Sie durch: objektorientiertes Zeichnen in allen Variationen mit hoher Präzision; drei Kurvenelemente für den Freihand-Entwurf; 250 (!) Zeichenebenen; eine Menü-Struktur, die sich dem Benutzer

anpaßt; eine Undo-Funktionen, mit der Sie alle (!) Zeichenvorgänge Schritt für Schritt widerrufen können; 38 Bemaßungsfunktionen und Werkzeuge zum bequemen Konstruieren u.v.a.m. BeckerDesign-ST-Grafiken lassen sich in alle Programme exportieren, die GEM-Metafiles einbinden. Natürlich können Sie auch Flächen mit Schraffuren, Mustern und Grauverläufen füllen, Symbole erstellen und einfügen, Vektor-Schriften nutzen ...



BeckerDesign ST
unverb. Preisempfehlung DM 99,-
ISBN 3-89011-818-6

Anschließen und gleich richtig loslegen.

Auspacken, anschließen und gleich richtig loslegen – das wünscht sich wohl jeder, der sich einen neuen Computer gekauft hat. „ST für Einsteiger“ erfüllt dem Anfänger diesen Wunsch. Hier ist der

schnelle Erfolg mit dem neuen Rechner vorprogrammiert. Denn alles wird systematisch und leichtverständlich erklärt: der Anschluß des Rechners, die Bedienung des GEM-Desktops, das Kopieren von Disketten, die Installation von Festplatte und Drucker und und und. Dabei beschreibt das Buch auch gleich den Umgang mit der wichtigsten Standardsoftware und gibt dem Einsteiger eine detaillierte Ein-

führung in die Programmierung mit OMIKRON-BASIC anhand einiger praxisorientierter Programmbeispiele. „ST für Einsteiger“ – Ihr erstes Buch zu Ihrem neuen Atari.



Schepers/Schulz
ST für Einsteiger
326 Seiten, DM 29,-
ISBN 3-89011-336-2

Das endgültige Atari ST Handbuch

Mit diesem Buch erhalten Sie das, was Sie schon lange vermißten; Sie werden schnell und professionell die ganze Leistungsfähigkeit des ATARI ST einsetzen, können sich in die Funktionen des TOS und des Desktops vertiefen oder die phantastischen Grafik- und Soundmöglichkeiten voll ausnutzen, das vorliegende Buch zeigt Ihnen eine umfangreiche Darstellung aller Fragen, die in diesem Zusammenhang auftreten können:

Von der BASIC-Programmierung über die Laufwerks- und DMA-Ansteuerung bis zur Grafik- und Systemprogrammiertechnik werden neben vielen praktischen Programmen Tips und Tricks vermittelt. In den Anhängen finden Sie schließlich alles, was man beim Umgang mit dem ATARI ST braucht: BIOS- und XBIOS-Funktionen, Fehlermeldungen, Befehlsstruktur, alles Wissenswerte zum Betriebssystem u.v.m.

- Die Möglichkeiten des Desktops vom ATARI ST
- Die Installation und der Umgang mit Festplatten
- Der Anschluß und die Inbetriebnahme der ATARI-Laserdrucker SLM804 und SLM605
- Umfangreiche Erläuterung zur Pflege und Wartung des ATARI ST
- Alles zur CPU des ATARI ST
- Alles Wissenswerte über die Programmierung von Grafik, Sound und GEM
- Ausführliche Dokumentation zur Ansteuerung von Laufwerken und Festplatten
- Dokumentierte Listings von nützlichen Utilities: Backup-Programm, Bildschirmschoner, RAMdisk usw.

Das endgültige ST-Handbuch für alle ATARI-ST-Anwender, die Ihren Computer effektiv und optimal einsetzen wollen.

ISBN N 3-89011-383-4 DM +039.80

DM 39,80
ÖS 310,-
sFr 37,80

**DATA
BECKER**



Hardwarevoraussetzungen:

Alle ATARI ST
und 1040 STE
mit mindestens 512
KByte Speicher.