Brückmann
Englisch
Gerits
Walkowiak

ATARI® ST

Tips & Tricks

EIN DATA BECKER BUCH

Brückmann
Englisch
Gerits
Walkowiak

ATARI® ST

Tips & Tricks

EIN DATA BECKER BUCH

ISBN 3-89011-118-1

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

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

1.0 DAS ATARI-BASIC

Das ATARI-BASIC gehört ebenso wie die Programmiersprache LOGO zum Lieferumfang des ST. Nun ist es eine alte Weisheit, daß die Hardware eines neuen Computers in deutlich kürzerer Zeit als die dafür benötigte Software fertig wird. Unter diesem Umstand leidet insbesondere auch das BASIC des ST. Bisher haben nur einige 'Vorab-Versionen' zu den Käufern gefunden.

Allerdings kann man der letzten, zum Zeitpunkt der Drucklegung dieses Buches allgemein verfügbaren BASIC-Version durchaus attestieren, in den weitaus meisten Punkten bereits fehlerfrei zu funktionieren. Auch sind viele Dinge im BASIC in einem Zustand, der vermuten läßt, daß sie sich in 'entgültigeren' Versionen nicht ändern werden. Von diesem Standpunkt aus ist es durchaus sinnvoll, bereits über das BASIC zu schreiben bzw. mit dieser Programmiersprache zu arbeiten. Es kann nur noch besser (und vielleicht noch etwas kleiner?) werden.

Wenn Sie bereits auf anderen Maschinen in BASIC programmiert haben, so werden Sie wahrscheinlich mit dem BASIC des ST gut zurecht kommen. Dieses von DIGITAL RESEARCH entwickelte BASIC lehnt sich in der Syntax stark an den verbreiteten MICROSOFT-Dialekt an. Der Sprachumfang ist vergleichbar mit dem des IBM-PC. Nur sehr spezielle, teilweise durch den Hardware-Aufbau des PC begründete Befehle im IBM-BASIC fehlen dem ST. Auf der anderen Seite stehen dem Programmierer im ST-BASIC Möglichkeiten zur Verfügung, von denen andere BASIC-Dialekte nur träumen können. So besteht eine sehr flexible Schnittstelle zum GEM oder besser ausgedrückt zum VDI. Im VDI sind viele Grafik-Möglichkeiten enthalten, die auch vom BASIC aus zugänglich sind. Diesem interessanten Kapitel werden wir uns später ausgiebig zuwenden.

1.1 Die Interessantesten BASIC-Befehle

Zunächst jedoch soll eine Auflistung der interessantesten und möglicherweise von anderen BASIC-Dialekten unbekannten Befehle mit kurzer Syntax-Beschreibung folgen. Alle Befehle hier aufzuführen würden den Rahmen dieses Buches bei weitem sprengen. Dazu ist der Vorrat an vorhandenen Befehlen einfach zu umfangreich.

FOLLOW, UNFOLLOW

Der FOLLOW-Befehl stellt für viele Fälle eine enorme Erleichterung bei der Fehlersuche da. Mit dieser Anweisung kann der Inhalt von angegebenen einfachen Variablen während des Programmlaufs beobachtet werden. Mit indizierten Variablen funktioniert der FOLLOW-Befehl jedoch nicht. Nach der Eingabe von UNFOLLOW wird die Ausgabe der Variablen abgestellt.

```
FOLLOW a,winkel%,text$
UNFOLLOW text$
```

BREAK, UNBREAK

Auch BREAK- und UNBREAK-Befehl dienen dem komfortablen Fehlersuchen in Programmen. Mußte bei früheren BASIC-Dialekten eine BREAK- oder STOP-Anweisung in das Programm geschrieben werden, so genügt beim ST die Eingabe von 'BREAK zeilennummer'. Nach dem Programmstart wird dann automatisch beim Erreichen der angegebenen Zeile der Programmlauf unterbrochen und die entsprechende Zeile ausgegeben. Danach reicht es, die RETURN- oder ENTER-Taste zu betätigen, und der Programmlauf wird fortgesetzt. Der BREAK-Modus ist damit allerdings nicht aufgehoben. Sobald die entsprechende Zeile wieder abgearbeitet wird (z.B. in Schleifen), wird auch das Programm wieder unterbrochen. Ausgeschaltet wird dieser Modus durch Eingabe von UNBREAK.

```
BREAK 120,512,2013
UNBREAK
```

TRON, TROFF

Der TRON-Befehl darf nur im Direktmodus und nicht im Programm eingesetzt werden. TRON steht für 'TRACE ON'; entsprechend bedeutet TROFF soviel wie 'TRACE OFF'. Wird TRON ohne Zeilennummer eingegeben, so wird beim Programmlauf die laufende Zeilennummer ausgegeben, bei TRON <zeilennummer<,zeilennummer>> werden nur die angegebenen Zeilennummern ausgegeben. TROFF schaltet diesen Mechanismus entweder vollständig oder nur für die angegebenen Zeilennumern aus.

```
TRON 50,100,121

TROFF

TROFF 121,30
```

TRACE, UNTRACE

Im Gegensatz zu TRON wird bei TRACE nicht nur die Zeilennummer, sondern der Inhalt der ganzen Zeile ausgegeben.
Die Syntax ist wie bei den zuvor genannten Befehlen.

BLOAD

Im Gegensatz zum einfachen LOAD- oder OLD-Befehl können mit dem BLOAD-Befehl komplette Speicherbereiche in den Speicher geladen werden. Mögliche Anwendungen sind das Laden von kleinen Maschinenprogrammen oder das Laden von Bildschirminhalten.

Als Parameter wird neben dem Namen der zu ladenden Datei die Ladeadresse erwartet. Diese Adresse wird keiner Bereichsüberprüfung unterzogen. Sie können also in jeden beliebigen Bereich hineinladen. Fatal wirkt sich dieser Umstand aus, wenn Sie die Angabe der Ladeadresse unterschlagen. In diesem Fall wird die Datei an den Anfang des Speichers bei der Adresse 0 geladen. Damit jedoch überschreibt man die wichtigen Exception-Vektoren, die diesen Bereich belegen. Damit geht zwar nichts kaputt, aber Sie dürfen anschließend neu booten.

```
BLOAD "screen.bin",&h78000
```

BSAVE

Mit diesem Befehl lassen sich Speicherbereiche auf Diskette schreiben. Praktisch kann dies sein, um den Bildschirm oder Teile davon zu speichern. Aber auch Maschinenprogramme oder Daten können mit BSAVE gespeichert werden. Die Parameter sind der Dateiname, die Startadresse und die Anzahl der zu speichernden Bytes.

```
BSAVE "screen.bin",&h78000,&h7d00
```

```
BSAVE "mprog.bin",&h7fd00,768
```

Im ersten Beispiel wird der Inhalt des Bidschirms des 520 und 260 gespeichert. Im 520+ dagegen beginnt der Bildschirmspeicher bei &hf8000.

Hinter dem Bildschirmspeicher liegt ein Bereich von 768 Bytes, der im ST ungenutzt ist. Hier lassen sich kleine Maschinenprogramme ablegen, da dieser Speicher nie vom BASIC oder vom Betriebssystem des ST berührt wird. Dieser Bereich wird im zweiten Beispiel auf Diskette gespeichert. Die Adressangabe bezieht sich auch in diesem Beispiel auf einen ST mit 512 KByte.

Das ATARI-BASIC verfügt über zwei verschiedene Möglichkeiten, Maschinenprogramme von BASIC aus aufzurufen. Die erste Möglichkeit ist der CALL-Befehl. Dieser Befehl bekommt die Adresse der gewünschten Routine und mögliche an die Routine zu übergebende Werte mitgeteilt. Allerdings muß die Adresse der Routine als Variable eingesetzt werden.

```
adresse = &h7fd00
CALL adresse

adresse = &h7fd00
wert1 = 33.33
wert2% = 100
z$ = "test"
call adresse(wert1,wert2%,100,z$,"leerstring")
```

Im ersten Beispiel wird die Routine ohne Parameter aufgerufen. Die Routine darf alle Register verändern. Allerdings muß am Ende der Routine der Stackpointer 'stimmen', da die Routine selbst mit RTS beendet werden muß. Wenn der Stackpointer nicht stimmt, wird eine falsche Rücksprungadresse benutzt.

Im zweiten Beispiel wird gezeigt, wie Parameter im CALL-Befehl angegeben werden müssen. Die Parameter müssen in Klammern stehen und durch Kommata voneinander getrennt sein. Alle Variablen-Typen sind als Parameter zugelassen. Allerdings werden die Parameter in eine vorzeichenbehaftete 32-Bit-Zahl ohne Nachkomma-Stellen gewandelt. Aus unserer Variablen 'wert1' würde also ein Wert von 33 an die Routine übergeben. Bei Strings, also den Zeichenketten (z$, "leerstring") wird die Adresse des Strings übergeben, die ja auch einen 32-Bit-Wert darstellt.

Wie aber kann im Maschinenprogramm auf die Daten zugegriffen werden? Wird die Maschinenroutine angesprungen, so sind die Register A0, A7 und D0 sehr wichtig.

A0 enthält die Adresse der Routine. Sie mögen jetzt denken, das sei überflüssig. Schließlich ist die Adresse einer Routine normalerweise bekannt. Lassen Sie sich überraschen. Später werden wir zeigen, wie überaus hilfreich der Inhalt von A0 werden kann.
Das Register D0 enthält in den unteren 16 Bit die Anzahl der übergebenen Werte. Auch dieser Wert ist für manche Anwendungen sehr hilfreich, besonders wenn die Routine mit unterschiedlicher Anzahl von Parametern aufgerufen werden kann. Die Verwendung eines 16-Bit-Zählers ist allerdings mehr als reichlich bemessen. Eine solch enorme Anzahl von Parametern ist in einer BASIC-Zeile gar nicht unterzubringen.
A7 schließlich, der USER-STACKPOINTER, enthält die Rücksprungadresse in den BASIC-Interpreter. Aber nicht nur die Rücksprungadresse liegt auf dem Stack. Mit dem Befehl MOVE.W 4(SP),Dx erhält man noch einmal die Anzahl der Parameter in das Register Dx. Mit dem Befehl MOVE.L 6(SP),Ax erhält man weiterhin den Anfang einer Tabelle. In dieser Tabelle stehen so viele Langworte (32-Bit Werte), wie Parameter im CALL-Befehl angegeben wurden. In diesen Langworten (auch kurz Longs genannt) stehen jetzt die Werte oder Adressen der Strings im beschriebenen Format.

PEEK

Obwohl die PEEK-Funktion auch von anderen Computern her bekannt ist, so hat gibt es im ST doch einige bemerkenswerte Besonderheiten. Die Funktion liefert beim PEEKen von Speicherplätzen 16-Bit-Werte. Bei einem PEEK(0) wird der Inhalt der Speicherzelle 0 als Low-Byte und der Inhalt von Speicherplatz 1 als High-Byte gelesen und entsprechend ausgegeben. Dieser Mechanismus ist jedoch nach Eingabe des Befehls DEF SEG aufgehoben. Danach liefert ein PEEK 'nur' noch einen 8-Bit-Wert (siehe DEF SEG).

Als weitere Besonderheit kann ein PEEK in Verbindung mit der Funktion VARPTR aber sogar einen 32-Bit-Wert liefern. Diese

Möglichkeit werden alle die schätzen, die gern spezielle Tricks in Verbindung mit Variablen anstellen. Einer dieser Tricks ist das Ablegen von Maschinenroutinen in Variablen. Mit der Zeile

```
adresse = PEEK (VARPTR (a$) +2)
CALL adresse
```

wird die 32-Bit-Adresse des Strings 'a$' in die Variable 'adresse' übertragen und das im String versteckte Maschinen-Programm ausgeführt.

Doch von solchen Tricks später mehr.

POKE

Für den POKE-Befehl gilt in etwa das bei der PEEK-Funktion gesagte. Es können also je nach Bedingung 1-Byte-, 2-Byte- und 4-Byte-Werte gePOKEd werden. Nach einem DEF SEG-Befehl wird als Wert ein Byte erwartet. Von größeren Werten werden nur die unteren 8 Bit gewählt.

4-Byte-Werte sind wieder im Zusammenhang mit der VARPTR-Funktion zugelassen.

DEF SEG

Die DEF SEG-Anweisung legt die Segmentadresse für die Befehle PEEK und POKE fest. Die Eingabe von DEF SEG oder DEF SEG=0 legt das Segment auf die physikalische Adresse 0 im Speicher fest. Dieses ist der nach dem Einschalten eingestellte Zustand.

Wird als Parameter ein größerer Wert als 0 eingegeben, so wird das Segment für PEEK und POKE auf diese Adresse festgelegt. Ein Beispiel mag dies verdeutlichen. Um in dem hinter dem Bildschirm liegenden RAM PEEKen und POKEn zu

können, besteht zum einen die Möglichkeit, bei den entsprechenden Befehlen die tatsächliche, also physikalische Adresse anzugeben.

```
wert = PEEK (&h7fd00)
```

Allerdings kann auch das Segment auf die gewünschte Adresse festgelegt werden. Dann sind die bei PEEK und POKE anzugebenden Adressen relativ zum Segmentanfang zu betrachten.

```
DEF SEG=&h7fd00
wert = PEEK (0)
```

Bedenken Sie aber, daß Sie im ersten Beispiel den Inhalt der Adressen &H7fd00 und &h7fd01 erhalten, im zweiten Beispiel jedoch nur den Inhalt der Adresse &h7fd00.

DEF USR, USR, USR0 - USR9

Die Befehle dieser Gruppe stellen die zweite Möglichkeit dar, Maschinenprogramme von BASIC aus aufzurufen. Leider ist diese Gruppe noch nicht endgültig fertiggestellt. Beim Definieren der Sprungadresse meldet der Interpreter in der vorliegenden Version, ein Programmteil mit dem Namen B.EVST sei noch nicht implementiert. Ein nachfolgender Aufruf der USR-Funktion veranlaßt den Interpreter zu der lakonischen Mitteilung 'function not yet done'. Warten wir's also ab.

GOTOXY x.pos,y.pos

Der GOTOXY-Befehl dient der Cursor-Positionierung auf dem Bildschirm. Ein Ausgabe-Befehl (PRINT oder WRITE) beginnt dann an der angegebenen Stelle. Die auf diese Weise angegebene Cursorposition ist auch für einen INPUT-Befehl wirksam.

Als Parameter werden die X- und die Y-Koordinate in Schreibpositionen erwartet. An Stelle der Variablen können natürlich auch Konstanten eingesetzt werden.

Der GOTOXY-Befehl zählt leider zu den reparaturbedürftigen Befehlen, da die X-Position nicht richtig ausgewertet wird. Der angegebene Wert wird fälschlicher Weise mit Zwei multipliziert. Auf diese Weise kommt man zu recht verblüffenden Ergebnissen. Wird jedoch nach der Ausgabe einer der Scroll-Streifen angeklickt, so wird der Inhalt des Ausgabefensters neu und diesmal mit den richtigen Positionen ausgegeben. Bei diesem Befehl sollten Sie also besser noch Vorsicht walten lassen. Bildschirmmasken lassen sich derzeit noch nicht sehr komfortabel gestalten.

```
GOTOXY 10,10:PRINT "Hier ist die Stelle 10,10"

GOTOXY x.pos,y.pos:INPUT wert
```

INKEY$

Wenn Funktion von funktionieren kommt, so trägt die INKEY$-Funktion ihren Namen derzeit zu Unrecht. Auch INKEY$ hat seine Macken. Es wird einfach kein Zeichen von der Tastatur geholt. Ursache ist wohl der Umstand, daß vor der Abarbeitung eines jeden Befehls nachgesehen wird, ob die Tasten 'Control' und 'G' oder 'Control' und 'C' gedrückt sind. In diesem Fall wird ja das Programm entweder vollständig beendet (Contrl C) oder aber unterbrochen (Contrl G).
Bei dieser Abfrage jedoch wird der interne Tastaturpuffer sehr regelmäßig entleert. Die INKEY$-Funktion ist nun so schnell, daß während der Abarbeitung kein neuer Tastendruck im Tastaturpuffer auftaucht. Somit kommt die Funktion immer ohne Tastenwert zurück. Für viele Anwendungen der INKEY$-Funktion gibt es aber guten Ersatz: die INPUT$-Funktion und die INP-Funktion. Wir wollen beide anschließend besprechen.

INPUT$

Die Funktion INPUT$ ist relativ ungewöhnlich und in herkömmlichen BASIC-Dialekten nur selten anzutreffen (z.B. IBM-PC). Mit dieser Funktion könnn ein oder mehrere Zeichen von der Tastatur oder auch aus einer Datei geholt werden. Das interessante an dieser Funktion ist, daß (fast) keine Interpretation von Steuerzeichen vorgenommen wird. Die Syntax lautet:

```
text$ = INPUT$ (10)

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

oder

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

Im ersten Fall werden 10 Zeichen von der Tastatur eingelesen, ohne diese Zeichen auf dem Bildschirm anzuzeigen. Dabei können auch die Tasten 'ENTER', 'RETURN', 'Contrl G' und 'Contrl C' gedrückt werden, ohne daß die Eingabe abgebrochen wird. Einzige Abbruchbedingung außer dem Erreichen der spezifizierten Anzahl von Zeichen ist die Eingabe von 'Contrl Z'. Dieses Zeichen mit dem ASCII-Wert 26 wird in Dateien üblicherweise als Kennung des Dateiendes eingesetzt. Die Eingabe von 10 Zeichen ist sicher nur selten erforderlich (z.B. unsichtbare Eingabe eines Passworts). Wenn die Anzahl jedoch auf eins verringert wird, so ist diese Funktion durchaus als Ersatz für die Konstruktion

```
10 a$ = INKEY$: IF a$="" then 10
```

geeignet. Allerdings erzeugen die speziellen Tasten des ATARI, also die Funktionstasten und die Tasten des Cursorblockes keinen ASCII-Wert. Diese Tasten sind mit der INPUT$-Funktion nicht abzufragen.

Im zweiten und dritten Beispiel werden je 10 Zeichen aus einer zuvor eröffneten Datei gelesen und in eine Variable übertragen. Wenn man mit Datensätzen fester Länge arbeitet, so kann man mit INPUT$ auch die sonst kritischen Zeichen wie Komma, Semikolon, Hochkomma und CR (ENTER-Taste) ohne Schwierigkeiten lesen. Für viele Anwendungen ist es weiterhin sehr vorteilhaft, wenn eine Datei zeichenweise gelesen werden kann. Genau dies ist mit einer Zeichenlänge von 1 bei der Angabe der Parameter der INPUT$-Funktion möglich.

INP, OUT

Bei Rechnern mit Z80- oder 8080-Prozessoren ist die INP-Funktion und der OUT-Befehl im BASIC oft zur Adressierung der bei diesen Prozessoren vorhandenen Portadressen eingebaut. Da aber der im ST eingesetzte 68000 keine Portadressierung kennt, stellt sich natürlich die Frage, was diese beiden Befehle im ATARI bewirken und welche Ergebnisse sie liefern.

Im BIOS des ST existieren drei Funktionsaufrufe mit den NamenBCONSTAT,BCONINundBCONOUT.ÜberdiesedreiAufrufe werden annähernd alle Ein- und Ausgaben des Systems an Tastatur und Bildschirm, Printer, V.24-Schnittstelle, MIDI-Schnittstelle und Tastatur-Prozessor durchgeführt. In Assembler sehen die Aufrufe so aus, daß die Nummer des gewünschten Gerätes im Aufruf mit angegeben wird. Dabei gilt die folgende Zuordnung:

Nummer	Gerät/Schnittstelle
0	Centronics-Schnittstelle / Drucker
1	V.24 Schnittstelle
2	Console, also Tastatur und Bildschirm
3	MIDI-Port
4	Tastatur-Prozessor

Genau diese Nummern werden auch bei den INP- und OUT-Befehlen eingesetzt. Man kann also von BASIC aus alle Schnittstellen sehr direkt ansprechen. So wird z.B. mit der Anweisung

```
OUT (0),65
```

der Wert 65 (ASCII-Wert des 'A') an den Drucker ausgegeben. Sie meinen, das die Anweisung

```
LPRINT "A"
```

genau so gut funktioniert? In diesem Fall haben Sie durchaus recht. Probieren Sie aber doch einmal, das Zeichen LF mit dem ASCII-Wert 10 mittels LPRINT an den Drucker zu schicken. Vermutlich werden Sie schnell merken, daß Sie keinen Erfolg haben. Der ST, besser das BASIC des ST schickt nämlich völlig unnötiger Weise die Zeichenfolge CR/LF, also die ASCII-Zeichen 13 und 10. Besonders beim Ausdruck von Grafik mit den weitverbreiteten EPSON-Druckern (und den kompatiblen) ist das nun wirklich nicht angebracht. Weder das Bitmuster noch die angegebene Zahl der Grafik-Bytes stimmt mit dem vorhergesehenen überein.
Ja, es kommt noch schlimmmer, der ATARI schickt nach 72 Zeichen ebenfalls die Zeichenfolge CR/LF, womit die Grafik dann entgültig durcheinander gerät. Da hätten die Programmierer des BASIC von den Fehlern anderer Programmierer lernen sollen.

Aber warum klagen? Der OUT-Befehl hilft uns in diesem Fall sehr schön aus der Patsche.

Nun kann aber nicht nur der Drucker mittels OUT-Befehl angesprochen werden. Auch die anderen Schnittstellen stehen uns zur Verfügung. So kann z.B. die V.24 auch unter BASIC vollwertig eingesetzt werden. Bei der V.24 ist es besonders interressant, daß mit der INP-Funktion auch Zeichen von der V.24 gelesen werden können. Damit kann mit nur wenig Aufwand ein komplettes Terminal-Programm für den ST in BASIC geschrieben werden.

Die INP- und OUT-Befehle mit der Geräte-Nummer 2 erlauben die Programmierung von BASIC-Programmen unter Umgehung des GEM. Dabei steht dann der komplette Bildschirm zur Verfügung. Fenster und ähnlicher 'Schnick-Schnack' werden dann nicht benötigt. Bei der Eingabe verhält sich der INP(2) ähnlich wie die Funktion INPUT$. Allerdings hat der INP(2) einen entscheidenden Vorteil. Auch die Funktionstasten und die Tasten des Cursor-Blocks liefern einen eindeutigen Wert und lassen sich somit abfragen.

Unter der Nummer 4 läßt sich die MIDI-Schnittstelle des ST sowohl als Eingang wie auch als Ausgang programmieren. Leser mit entsprechenden Geräten, also Orgeln oder Synthesizern können ihre Geräte in Zukunft vom ST aus 'fernsteuern'. Voraussetzung ist natürlich, daß Sie das bei MIDI verwendete Protokoll kennen. Mit diesem Wissen aber ist es dann verhältnismäßig einfach, die Geräte mit einem BASIC-Programm zu steuern.

Das letzte Gerät, welches in unserer Aufzählung möglich ist, ist die Tastatur. Wie Sie sicher wissen, handelt es sich beim ATARI um eine sogenannte intelligente Tastatur. Sie enthält einen eigenen kleinen Prozessor, der sowohl die Tasten als auch die Maus oder eventuell angeschlossene Joysticks abfragt. Zusätzlich ist noch eine Uhr im Programm des Tastatur-Prozessors eingebaut. Allerdings können ohne größeren Aufwand nur Werte an den Tastatur-Prozessor geschickt werden, da die 'Antwort' in der Regel vom Betriebssystem des ST weggeworfen wird. Ein INP(4) liefert immer nur den Wert 16.

VARPTR

VARPTR ist eine Funktion, welche als Ergebnis eine Adresse liefert. Als Argument wird der VARPTR-Funktion eine Variable oder eine Dateinummer(!) übergeben:

```
a$ = "TEST"
ad = VARPTR (a$)

a = 10
adr = VARPTR (a)

OPEN "I",1,"dudu.dat"
? VARPTR (#1)
```

Die Rückgabewerte des letzten Aufrufs entziehen sich leider einer vernünftigen Interpretation. Vielleicht finden Sie ja heraus, wie die gelieferte Adresse zu deuten ist. Endgültig geklärt wird dieser Punkt sicher erst, wenn ein vernünftiges BASIC-Handbuch geliefert wird.

In den anderen beiden Beispielen ist die Interpretation verhältnismäßig einfach. Betrachten wir zunächst den ersten Fall.

Nach dem VARPTR-Aufruf enthält die Variable ad die Adresse des Stringdescriptors (Descriptor = Beschreiber) der Variablen a$. Der Stringdescriptor selbst besteht aus sechs Bytes. Das erste Byte des Stringdescriptors enthält ein Flag, dessen Funktion gleich erklärt wird. Das zweite Byte im Descriptor gibt die Länge des Strings an. Damit ist festgelegt, daß Strings maximal 255 Zeichen lang sein können.
In dritten bis sechsten Byte schließlich ist die Adresse eingetragen, an der der String selbst im Speicher steht.

Wenn Sie jedoch diese Werte mit dem oben aufgeführten Beispiel durchprobieren, so wird Sie die String-Adresse sicher in Erstaunen versetzen. Die 'Adresse' entpuppt sich nämlich bei näherer Betrachtung als der String selbst. Alle Strings, die ein bis vier Zeichen lang sind, werden direkt in der 'Adresse' des Stringdescriptors gespeichert.
Damit wird auch die Bedeutung des Flags, also des ersten Bytes des Descriptors klar. Ist hier eine Null eingetragen,

so ist der String kleiner als fünf Zeichen und entsprechend im Descriptor selbst untergebracht. Ist jedoch der hexadezimale Wert 10 eingetragen, so handelt es sich bei den Bytes in den Positionen Drei bis Sechs um die tatsächliche Speicheradresse des Strings.

Die VARPTR-Funktion liefert bei numerischen Variablen direkt die Speicheradresse, an der die Zahl zu finden ist. Einfach genaue Zahlen werden in vier Byte, Interger (z.B. A%) in zwei Byte abgespeichert. Besonders die Integer-Array werden wir später noch unter die Lupe nehmen, da sie sich als gut geschützter Speicher für kleine Maschinenprogramme eignet.

SOUND

Der SOUND-Befehl des ST-BASIC ist recht leistungsstark und sehr einfach zu programmieren.

Der Sound-Chip des ST ist ein IC mit der Bezeichnung YM-2149. Dieses IC ist kompatibel zum bekannteren Sound Chip AY-3-8910, der in verschiedenen anderen Computersystemen (CPC, MSX-Rechner etc.) eingesetzt ist. Dieser Chip bietet eine breite Palette von Möglichkeiten, Töne über drei unterschiedliche Kanäle zu erzeugen. Weiterhin kann eine Rauschquelle eingeschaltet werden, die bei entsprechender Programmierung die Erzeugung spezieller Effekte (z.B. Schlagzeug) zuläßt.

Der SOUND-Befehl hat maximal 5 Parameter, die als numerische Werte hinter dem SOUND-Befehl angegeben werden. Ein vollständiger Befehl lautet also:

```
SOUND kanal,lautstärke,tonhöhe,oktave,dauer
```

Der Wert von 'kanal' kann 1, 2 oder 3, je nach gewünschtem Kanal sein. Der Wert von 'lautstärke' liegt zwischen 1 und 15, wobei 1 ganz leise, 15 dagegen volle Lautstärke bedeutet. Dieser Wert wird je nach ausgewähltem Kanal in die

Bits 0 bis 3 der Register 8 (Kanal A), 9 (Kanal B) oder 10 (Kanal C) übertragen. Die Variable 'tonhöhe' erlaubt Werte zwischen 1 und 12. Da in der Musik eine Oktave aus 12 Tonstufen besteht, kann somit direkt mit Noten gespielt werden. Die 'oktave' kann zwischen 1 und 8 liegen, der ST kann also über acht Oktaven Töne erzeugen. Die 'dauer' schließlich kann Werte zwischen 1 und 255 annehmen. Gemessen wird die Dauer in 20 Millisekunden. Wenn Sie also einen Wert von 50 als Dauer angeben, so wird der Ton (ca.) 1 Sekunde ertönen.
Die folgende Tabelle gibt Aufschluß, wie die Töne in der Variable 'tonhöhe' zu verstehen sind:

1	C	2	C#	3	D
4	D#	5	E	6	F
7	F#	8	G	9	G#
10	A	11	A#	12	H

Den Kammerton A (440 Hertz) erhält man mit dem SOUND-Befehl:

```
SOUND 1,15,10,4,255
```

Die Oktave 4 ist also die normalerweise als Oktave Null bezeichnet. Kleinere Okav-Werte ergeben tiefere Töne, höhere Werte erzeugen höhere Töne.

WAVE

Mit dem SOUND-Kommando allein kann man zwar schon recht hübsche (einstimmige) Melodien programmieren, richtig interessant und polyphon wird es jedoch beim WAVE-Kommando. Hier stehen sehr viel mehr Möglichkeiten zur Verfügung. Dafür ist es allerdings auch um einiges undurchsichtiger. Es hat uns schon ziemliche Mühe gekostet, den Aufbau und die Parameter zu verstehen. Auch ist zum vollen Verständnis eine genaue Kenntnis des Hardware-Aufbaus des Soundchip nützlich.

Das WAVE-Kommando kennt wie das SOUND-Kommando fünf Parameter. Der erste Parameter ist etwa vergleichbar mit dem 'kanal'-Parameter des Sound-Kommandos. Mit ihm kann der Kanal ausgewählt werden, der einen Ton erzeugt. Allerdings sind hier die möglichen Werte etwas anders einzusetzen.

Am besten versteht man den Aufbau des Wertes, wenn man sich einmal ein spezielles Register des Sound Chip etwas näher ansieht. Gemeint ist das Register 7 mit der Bezeichnung Multifunktionsregister. In diesem Register haben einzelne Bits unterschiedliche Aufgaben.
Die Bits 0 bis 2 sind für das Ein- und Abschalten der drei Ton-Kanäle zuständig. Ist das Bit 0 des Registers 7 gesetzt, so ist der Kanal A ausgeschaltet. Bei gesetztem Bit 1 ist der Kanal B und bei gesetztem Bit 2 der Kanal C ausgeschaltet. Ein gelöschtes Bit erzeugt den für den Kanal programmierten Ton.
Die Bits 3 bis 5 sind für das Zuschalten von Rauschen zu den drei Kanälen verantwortlich. Auch hier wird bei gelöschtem Bit die Funktion eingeschaltet, ein auf Eins gesetztes Bit schaltet das Rauschen für den entsprechenden Kanal aus.
Weiterhin sind die Bits 6 und 7 für die Programmierung der Datenrichtung der beiden auf dem Sound Chip integrierten universellen 8-Bit-Ports zuständig. Diese beiden Bits sind jedoch für die Tonerzeugung ohne Belang, so daß wir hier nicht näher auf sie eingehen wollen.

Genau diese Bits 0 bis 5 nämlich können mit dem ersten Parameter des WAVE-Befehls gezielt manipuliert werden. Wird der Parameter als Binärwert betrachtet, so haben die einzelnen Bits des Parameters genau die umgekehrte (und besser zu merkende) Funktion. Wird als Parameter der Wert 1 übergeben, so wird das Bit 0 des Registers 7 gelöscht, der Kanal A wird demnach eingeschaltet. Alle anderen Bits des Registers 7 werden bei diesem Parameter gesetzt, die entsprechende Funktion also ausgeschaltet. Lautet der erste Parameter dagegen z.B. 37 (binär %100101), so werden die Kanäle A und C ein- und Kanal B ausgeschaltet. Zusätzlich wird zu Kanal C die Rauschquelle zugeschaltet.

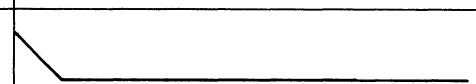
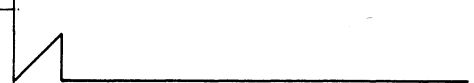
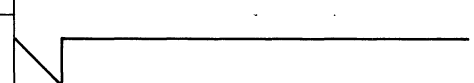
Null als erster Parameter schaltet demnach alle Kanäle und Rauschquellen aus.

Der zweite Parameter des WAVE-Kommandos beeinflußt gleich drei Register des Sound Chips. Es sind dies die Register mit den Nummern 8, 9 und 10. Allerdings werden nicht alle Bits der drei Register sondern jeweils nur das Bit 4 beeinflußt. Dieses Bit 4 in den drei genannten Registern entscheidet, ob die Lautstärke der drei Kanäle durch die Angabe im SOUND-Kommando (Inhalt der Bits 0 bis 3 der drei Register, siehe SOUND) oder durch eine Hardware-Hüllkurve beeinflußt wird.

Die Hardware-Hüllkurve ist eine spezielle Möglichkeit des Sound Chip. Bei Nutzung der Hüllkurve wird die Lautstärke des Tones periodisch oder auch nur einmalig geändert. Gerade mit dieser Hüllkurve bestehen sehr weitreichende Möglichkeiten, den Klang eines Tones zu verändern.
Auch der zweite Parameter muß als Binär-Wert betrachtet werden. Die Bedeutung der Bits im Kommando zeigt die folgende Aufstellung.

Bit-Nummer	Funktion
1 gelöscht	Lautstärke Kanal A aus Bits 0-3 Reg. 8
1 gesetzt	Lautstärke Kanal A über Hüllkurve
2 gelöscht	Lautstärke Kanal B aus Bits 0-3 Reg. 9
2 gesetzt	Lautstärke Kanal B über Hüllkurve
3 gelöscht	Lautstärke Kanal C aus Bits 0-3 Reg. 10
3 gesetzt	Lautstärke Kanal C über Hüllkurve

Damit ist der Wertebereich auf 0 bis 7 festgelegt. Wird als zweiter Parameter der Wert 0 übergeben, so wird die Lautstärke aller drei Kanäle durch die im SOUND-Kommando angegebene Lautstärke bestimmt. Bei einem Wert von 5 wird die Lautstärke der Kanäle A und C durch die Hardware-Hüllkurve manipuliert, Kanal B läuft jedoch weiter über die in den Bits 0 bis 3 des Registers 9 eingestellte Lautstärke.
Der dritte Parameter des WAVE-Kommandos steht im engen

REG. 15				
B3	B2	B1	B0	
CONTINUE	ATTACK	ALTERNATE	HOLD	
0	0	–	–	
0	1	–	–	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Zusammenhang mit Parameter 2. Dieser Parameter wählt eine der 9 verschiedenen Hüllkurven aus. die möglichen Werte können zwischen 0 und 15 liegen, allerdings erzeugen einige Werte identische Hüllkurven. Die möglichen Resultate lassen sich mit Worten auch nicht annähernd beschreiben. Darum finden Sie die Werte und die entsprechenden Hüllkurven in der nebenstehenden Darstellung.

Auch der vierte Parameter manipuliert direkt Register im Sound Chip und steht im Zusammenhang mit der Hüllkurve. Der Sound Chip verfügt nämlich über zwei 8-Bit-Register, deren Inhalt die Periodendauer der periodischen Hüllkurven beeinflußt. Damit ist auch der Wertebereich des vierten Parameters klar. Er beträgt von 0 bis 65535. Je größer der Wert gewählt wird, desto länger läuft eine Periode der Hüllkurve. Bei extrem kleinen Werten (<1000) laufen die Hüllkurven mit einer so hohen Frequenz, daß eine zusätzliche hörbare Frequenz entsteht. Das kann für verschiedene Verfremdungseffekte durchaus einsetzbar sein.
Der fünfte Parameter bestimmt wie beim SOUND-Befehl die Länge des erzeugten Tones. Er ist nur Programm-Modus wirksam, wenn ein weiterer SOUND- oder WAVE-Befehl folgt. Im Direkt-Modus oder aus dem Editor ertönt der Ton bis zum Drücken der nächsten Taste, also dem Tastatur-Klick.

LINEF

Die LINEF-Anweisung ist die einfachste Grafik-Funktion des ST-BASIC. Mit diesem Befehl können beliebige Punkte oder Linien auf dem Bildschirm gezeichnet werden. Dazu sind vier Parameter nötig, welche die Start- und Endpunkte der Linie angeben. Um einen Punkt darzustellen, werden einfach Start und Ende auf den selben Bildschirmpunkt gelegt. Grundsätzlich ist eine Strichdicke von einem Punkt eingestellt. Wir werden aber später Möglichkeiten aufzeigen, nicht nur die Strickdicke, sondern auch das verwendete Linienmuster oder das Aussehen von Anfangs- und Endpunkt der Linien zu verändern.

```
LINEF 10,10,50,40
```

CIRCLE

Die CIRCLE-Anweisung erlaubt das Zeichnen beliebiger Kreise oder Kreis-Auschnitte auf dem Bildschirm. Dazu werden entweder drei oder fünf Parameter benötigt. Zum Zeichnen einfacher Kreise sind drei Parameter ausreichend. Die ersten beiden Parameter bestimmen die X- und Y-Koordinate des Kreismittelpunktes, der dritte Parameter den Kreisradius. Vierter und fünfter Parameter werden nur benötigt, wenn Kreisausschnitte gezeichnet werden sollen. Dann kann mit diesen Parametern Start und Ende des Kreisbogens in Grad angegeben werden. Allerdings muß berücksichtigt werden, daß die Grad-Angabe in 1/10-Grad erfolgen muß. Ein Halbkreis mit einem Radius von 100 Punkten würde also mit der Anweisung

```
CIRCLE 320,199,100,0,1800
```

erzeugt werden. Wenn Sie einmal diese Zeile eingeben, können Sie auch gleich den 0-Punkt für die Winkel-Angabe sehen. Gezeichnet wird der Kreis immer gegen den Uhrzeigersinn ab der Position, die bei einer Zeigeruhr 3 Uhr bedeutet.

Auch bei der CIRCLE-Anweisung ist eine Strichdicke von einem Pixel eingestellt, die wir aber später ändern werden.

Als letzte Besonderheit ist zu bemerken, daß der Kreis in Wirklichkeit gar kein Kreis ist. Die CIRCLE-Anweisung zeichnet statt dessen ein Vieleck, das annähernd Kreisform hat. Besonders bei einem Radius von 30 kann man sehr schön sehen, daß es sich nicht um einen richtigen Kreis sondern eher um ein Achteck handelt. Wenn also ein 'echter' Kreis nötig ist, müssen Sie die Werte selbst berechnen und mittels LINEF-Anweisung zeichnen.

PCIRCLE

Auch die PCIRCLE-Anweisung liefert einen 'Kreis' oder Teilkreis als Ergebnis. Die benötigten Parameter sind identisch mit denen des CIRCLE-Befehls. Allerdings wird dieser Kreis zusätzlich mit einer Farbe oder einem Muster gefüllt. Die Einstellung von Farbe und Muster geschieht mit dem COLOR-Befehl.

ELLIPSE

Neben der reinen Kreisform können auch Ellipsen oder Teile davon gezeichnet werden. Aus diesem Grund hat die ELLIPSE-Anweisung entweder vier oder sechs Parameter. Die ersten beiden Parameter bestimmen wieder die X- und Y-Lage des Mittelpunktes, die folgenden beiden Parameter geben den Radius der Ellipse in X- und Y-Richtung an. Die (optionalen) Parameter 5 und 6 geben den Winkel eines Ellipsenausschnittes an, wobei die unter CIRCLE gemachten Aussagen auch hier zutreffen.

```
ELLIPSE 320,200,100,30,450,2700
```

PELLIPSE

Das 'P' vor dem Namen kennzeichnet auch hier, daß die entstehende Form mit der aktuellen Farbe und dem aktuellen Muster gefüllt wird. Die Parameter sind identisch mit dem ELLIPSE-Befehl.

COLOR

Der COLOR-Befehl ermöglicht das Einstellen der Schriftfarbe, der Farbe der Füllmuster, der Farbe der mit LINEF gezogenen Linien und der beim Füllen von Bildausschnitten verwendeten Muster. Dazu sind insgesamt 5 Parameter vorgesehen.

Der erste Parameter bestimmt die für die nächsten Textausgaben zu verwendende Farbe. Bei einem Monochrom-Monitor sind nur Werte zwischen 0 und 1 möglich. Bei einer 0 als Parameter erscheint die Schrift 'unsichtbar', also in der Hintergrundfarbe. Bei Verwendung eines Farbmonitors ist die maximale Größe vom benutzten Modus abhängig. Im Low-Resolution-Modus (320 x 200) kann zwischen 16 Farben (0 bis 15), im Mid-Res-Modus kann zwischen den Werten 0 bis 3 gewählt werden.
Der zweite Parameter gibt die Farbe für die nächsten PCIRCLE-, PELLIPSE- und FILL-Kommandos an. Die Werte entsprechen denen des ersten Parameters.
Auch der dritte Parameter bestimmt eine Farbe, diesmal aber für die Grafik-Befehle CIRCLE, ELLIPSE und LINEF. Wie man sieht, kann man es bei Verwendung eines Farbmonitors ganz schön bunt treiben.

Die beiden letzten Parameter bestimmen das Füllmuster bei Verwendung der Befehle FILL, PCIRCLE und PELLIPSE. Der fünfte Parameter kann zwischen 0 und 4 liegen und bestimmt die grundsätzliche Auswahl des Musters.
Ist dieser Wert 0, so wird unabhängig von anderen Einstellungen kein Muster gezeichnet.
Bei einem Wert von 1 wird auch kein Muster gezeichnet, statt dessen wird der angegebene Bereich vollständig mit der Zeichenfarbe gefüllt.
Bei einem Wert von 2 werden dann aber Muster mit der eingestellten Farbe erzeugt. Die Form des Musters (insgesamt sind hier 24 verschiedene Muster verfügbar) wird jetzt durch den vierten Parameter bestimmt.
Lautet der fünfte Parameter 3, so werden andere Muster gewählt. Diese Muster stellen verschiedene Schraffuren dar. Wird jedoch der fünfte Parameter auf den Wert 4 gesetzt, so wird ein Muster gewählt, das unter GEM selbst definiert werden kann. Die ATARI-Leute haben als Benutzer-definiertes Zeichen das ATARI-Symbol eingetragen.

FILL

Das FILL-Kommando erlaubt das Einfüllen beliebiger Flächen. Auch hier werden die mit dem COLOR-Befehl eingestellten Vorgaben verwendet. Als Parameter wird ein Punkt auf dem Bildschirm mit X- und Y-Koordinate angegeben, der innerhalb der zu füllenden Fläche liegt.
Optional kann ein dritter Parameter angegeben werden. In diesem Fall wird der angegebene Wert als die Farbe angesehen, an der die Füll-Routine endet. Ohne diesen Parameter endet die Füllroutine an jedem Punkt mit einer anderen als der Hintergrundfarbe.

FULLW

Dieser Befehl ist der erste einer ganzen Reihe von Kommandos zur Manipulation der Windows. Mit dem FULLW-Kommando kann ein beliebiges der vier Windows auf Maximalgröße eingestellt werden. Die vier im BASIC vorhandenen Fenster sind über die folgenden Nummern ansprechbar:

```
0  EDIT-Window
1  LIST-Window
2  OUTPUT-Window
3  COMMAND-Window
```

Mit der Zeile:

```
FULLW 2
```

wird also das Ausgabefenster über die gesamte verfügbare Bildschirmfläche gelegt.Die anderen drei Fenster werden durch den Befehl überdeckt.

CLEARW

Dieser Befehl dient zum Löschen eines Fensterinhaltes. Er

ist vergleichbar mit dem CLS-Befehl im MICROSOFT-BASIC, bezieht sich im ST-BASIC jedoch auch auf ein spezielles Window. Nach der Zeile

```
CLEARW 2
```

ist das Ausgabe-Fenster gelöscht. Allerdings wird die Position des Ausgabe-Cursors durch diese Anweisung nicht verändert, so daß im allgemeinen nach einem CLEARW 2 die Anweisung GOTOXY 0,0 folgen sollte. Damit steht dann auch der Cursor in der linken oberen Ecke.

CLOSEW

Mit dem CLOSEW-Kommando können Windows gelöscht werden. Sie sind danach vollständig vom Bildschirm verschwunden. Die Nummern der Windows entsprechen denen der anderen Windowanweisungen.

OPENW

Mit dem OPENW-Befehl können gelöschte (CLOSEW) Windows wieder eröffnet werden. Allerdings funktioniert diese Anweisung nur, wenn noch mindestens ein Fenster geöffnet ist. Hier scheint noch ein Fehler im BASIC zu sein.

Zusammenfassung

Damit wären wir am Ende unserer Beschreibung der speziellen Befehle des ST-BASIC. Alle anderen Befehle und Funktionen des BASIC funktionieren mehr oder weniger genau so wie in anderen BASIC-Dialekten. Da es eine große Menge Literatur über die Standard-Funktionen und -Befehle gibt, wollen wir diese nicht auch noch aufführen.
Allerdings haben wir ganz bewußt zwei besondere Befehle in unserer Aufstellung ausgelassen, die Sie sicher in keinem BASIC-Standard-Werk finden werden. Es sind die Befehle GEMSYS und VDISYS. Da diese Befehle aber so umfangreich und leistungsfähig sind, wollen wir diesen beiden Spezialitäten ein eigenes Kapitel einräumen. Sie werden wirklich überrascht sein, welche Möglichkeiten Sie vom BASIC des aus haben.

1.2 BASIC und GEM

Neben den vielen Möglichkeiten, die das BASIC des ST bereits von Haus aus kennt, gibt es noch sehr viel weitreichendere Möglichkeiten, speziell in Hinblick auf die Funktionen, die unter GEM verfügbar sind. Diese Funktionen wollen wir im nun folgenden Kapitel näher betrachten. Dazu beschäftigen wir uns zunächst mit dem GEM und einigen häufig benutzten Begriffen.

GEM, die Abkürzung für GRAPHICS ENVIRONMENT MANAGER, ist ein System, das entwickelt wurde, um auf verschiedenen grafikfähigen Computersystemen grafik-intensive Programme hardwareunabhängig zu ermöglichen. Auch besteht unter GEM die Möglichkeit, die Ausgaben nicht nur auf den Bildschirm sondern auch auf einen Plotter, einen Matrixdrucker oder de facto jedes grafikfähige Ausgabegerät zu schicken.

Sie mögen jetzt fragen wie ein Programm auf verschiedenen Rechnern laufen kann. Dazu wollen wir einmal kurz die 'große weite CP/M-Welt' betrachten. Nur durch die sehr strikte Festlegung der unter CP/M verfügbaren Möglichkeiten und bei Vermeidung von hardware-spezifischen Möglichkeiten des einen oder anderen CP/M-Rechners kann ein und das selbe Programm, z.B. WORDSTAR oder M-BASIC ohne Anpassungen auf verschiedenen Rechnern mit CP/M als Betriebssystem laufen. Ähnlich verhält es sich bei GEM. Wenn sich Programme an die unter GEM vorhandenen Möglichkeiten halten, so können Sie ohne weiteres auch auf anderen Rechnern mit GEM laufen.

Natürlich kann ein Programm für den 68000, also den ATARI ST nicht auf einem IBM-PC unter GEM laufen, da unterschiedliche Prozessoren in den Rechnern Verwendung finden. Wenn die Programme jedoch in einer höheren Programmiersprache (z.B. C, Pascal, Modular2) geschrieben sind, so genügt bei 'richtiger' Programmierung ein einfacher Compiler-Lauf auf der neuen Hardware und das Programm läuft. In der Praxis werden schon kleinere Änderungen am Programm erforderlich

sein, da in fast jedem Programm hardware-spezifische Möglichkeiten einer Maschine genutzt werden. Wenn man aber wirklich strikt diese Dinge vermeidet, so kommt man dem Ideal, der ausschließlichen Compilierung sehr nahe.

GEM wird unterteilt in zwei Funktionsblöcke: das AES und das VDI. Das AES ist der Teil des GEM, der für die Organisation der Fenster, der Menues, der ICONs und verschiedener anderer Dinge zuständig ist. Das AES beinhaltet also die übergeordneten komplexeren Funktionen. Die AES-Funktionen sind in der Mehrzahl ungeeignet, um von BASIC-Programmen aus aufgerufen zu werden. Allerdings gibt es einige Ausnahmen, wie Sie später sehen werden.

Das VDI dagegen enthält die grundsätzlichen Grafik-Routinen zur Ausgabe von Text, Grafik und zur Eingabe von Text oder zur Abfrage der Mausposition und -tasten. Das VDI ist seinerseits unterteilt in einen GDOS (Graphics Device Operating System) genannten Teil und einen sogenannten Device Driver. Der Devive Driver (Geräte Treiber) ist der eigentlich hardware-abhängige Teil des GEM und muß für verschiedene Ausgabegeräte angepasst werden. In der derzeitigen GEM-Version des ST ist der einzige verfügbare Gerätetreiber die Schnittstelle zum Monitor. Weitere Treiber werden sicher im Laufe der Zeit dazu kommen.

1.2.1 Der VDISYS-Befehl

Der Befehl VDISYS ist die Schnittstelle zum VDI. Wird dieser Befehl in einem Programm ausgeführt, so wird eine der vielen Funktionen des VDI aufgerufen und ausgeführt. Um jedoch eine Funktion aufrufen zu können, müssen dem VDI verschiedene Parameter und die Nummer der Funktion mitgeteilt werden.

Für die Parameterübergabe benötigt das VDI fünf Arrays oder Speicherbereiche, in denen die Werte untergebracht werden. Die Arrays haben die Namen CONTRL, INTIN, INTOUT, PTSIN und PTSOUT. Die Programmierer, die das ST-BASIC entwickelt

haben, fanden die Möglichkeiten des VDI wohl so reizvoll, daß sie die Adressen dieser Arrays als reservierte Variablen mit eben diesen Namen im BASIC zur Verfügung stellten. Die Adressen der Arrays erhalten Sie durch Eingabe von

```
? contrl;intin;intout,ptsin,ptsout
```

(Anm.: die genannten Arrays sind nicht die tatsächlich vom VDI adressierten Arrays. Die Werte der BASIC-Arrays werden jedoch automatisch in die VDI-Arrays übertragen).

Da die Arrays zur Verfügung stehen, steht nun einem Aufruf des VDI nichts mehr im Weg. Ein Beispiel soll die grundsätzliche Vorgehensweise zeigen.

```
POKE CONTRL   ,(nummer des aufrufs)
POKE CONTRL+ 2,(anzahl parameter in intin)
REM  CONTRL+ 4,(anzahl parameter in ptsout)
POKE CONTRL+ 6,(anzahl parameter in ptsin)
REM  CONTRL+ 8,(anzahl parameter in ptsin)
POKE CONTRL+10,(eventuell unternummer der fuktion)
POKE CONTRL+12,(device handle, zwischen 1 und 10)
REM
POKE INTIN    ,(erster parameter)
POKE INTIN + 2,(zweiter parameter)
POKE INTIN + 4,(dritter parameter)
:
bis
:
POKE INTIN + n,(letzter parameter)
REM
POKE PTSIN    ,(erster parameter)
POKE PTSIN + 2,(zweiter parameter)
POKE PTSIN + 4,(dritter parameter)
:
bis
:
POKE PTSIN + n,(letzter parameter)
REM
```

```
VDISYS
REM
```

In diesem Beispiel können Sie sehen, wie die einzelnen Parameter in die entsprechenden Array-Elemente gePOKEd werden. Da die einzelnen Elemente alle 16 Bit breit sind, genügt ein POKE, um den Wert zu übertragen. Daraus erklären sich auch die Zweier-Schritte bei den POKEs. Die Elemente in CONTRL-ArrayCONTRL+4undCONTRL+8werdennichtgePOKEd. Nach dem Aufruf kann in diesen Elementen abgefragt werden, wie viele Parameter die Funktion als Ergebnis in INTOUT und PTSOUT geliefert hat.

Ein praktisches Beispiel soll das Ganze noch einmal verdeutlichen. In diesem Beispiel wollen wir den normalerweise unsichtbaren Maus-Cursor per VDI-Aufruf einschalten. Dazu müssen wir natürlich die Funktionsnummer und die benötigten Parameter wissen. Diese Angaben finden sich z.B. in der Dokumentation zum GEM von DIGITAL RESERACH, der Firma, die GEM entwickelt hat. Aber auch im GEM-Buch von DATA BECKER sind die benötigten Werte für alle GEM-Funktionen nachzulesen. Wir können hier nur einige besonders wichtige Funktionen aufführen, um den Rahmen dieses Buches nicht zu sprengen.

Der benötigte OPCODE, also die Nummer unserer Funktion lautet 122. Die Funktion selbst trägt den Name 'SHOW MOUSE'. Die Funktionsnummer, also 122, müssen wir in CONTRL eintragen. Im PTSIN-Array werden für die SHOW MOUSE-Funktion keine Parameter erwartet, darum muß CONTRL+2 auf Null gesetzt werden. Im INTIN-Array jedoch wird ein Parameter erwartet. Aus diesem Grunde muß in CONTRL+6 eine 1 eingetragen werden. In CONTRL+10 wird von SHOW MOUSE kein Wert erwartet. Dieses Element enthält bei einigen VDI-Funktionen eine weitere Spezifizierung der gewünschten Funktion.
CONTRL+12 schließlich enthält die sogenannte Device Handle. Nach dem Starten des BASIC ist hier der Wert 2 eingetragen. Da als Handle für die Ausgabe auf den Bildschirm jeder Wert

zwischen 1 und 10 zugelassen ist, muß hier nicht unbedingt ein Wert angegeben werden. Bei einem Wert im Bereich von 11 bis 20 würde (bei Vorhandensein eines geeigneten Gerätetreibers) die Ausgabe auf einem Plotter, im Bereich zwischen 21 und 30 auf einen Drucker geleitet.

Damit haben wir die Parameter des CONTRL-Arrays zusammen. Wichtig ist jedoch noch der im INTIN benötigte Wert. Dazu die folgende Erklärung:

Neben der SHOW MOUSE-Funktion gibt es auch eine HIDE MOUSE-Funktion, die den Maus-Cursor abschaltet. Das VDI merkt sich die Anzahl der HIDE MOUSE-Aufrufe. Wird SHOW MOUSE mit einem Wert <> Null in INTIN aufgerufen, so wird von dieser gemerkten Anzahl eins abgezogen. Der Cursor wird also nicht unbedingt nach dem Aufruf sichtbar.
Wird jedoch in INTIN eine Null eingetragen, so wird die Anzahl der HIDE MOUSE-Aufrufe ignoriert und der Cursor wird unbedingt eingeschaltet.

Das vollständige Beispiel sieht demnach so aus:

```
10  POKE CONTRL   ,122
20  POKE CONTRL+ 2,0
30  POKE CONTRL+ 6,1
40  REM
50  POKE INTIN    ,0 : REM unbedingt einschalten
60  REM
70  VDISYS
80  REM
```

Nach dem Aufruf finden Sie in CONTRL+4 und CONTRL+8 die Werte Null. Damit wird signalisiert, daß die Funktion keine Rückgabewerte in INTOUT und PTSOUT geliefert hat.

1.2.2 Von BASIC aus nutzbare VDI-Aufrufe

Fast jeder VDI-Aufruf läßt sich irgendwann einmal auch von BASIC aus nutzen. Allerdings werden etliche Aufrufe überflüssig, weil es bereits entsprechende BASIC-Befehle gibt. So ist es sicher etwas umständlich, eine Linie mit Hilfe des VDISYS zu zeichnen. Das geht einfacher mit dem LINEF-Befehl. Auch ist eine Textausgabe über das VDI nur in ganz wenigen Fällen erforderlich. Allerdings können diese Effekte mitunter recht angebracht sein. Probieren Sie doch einfach die Beispiele einmal durch und entscheiden Sie im Einzelfall, ob Sie die eine oder andere Funktion verwenden können.

Der erste Punkt, dem wir uns zuwenden wollen, ist die Erforschung der möglichen Texteffekte.

Text Effekte
Set Graphic Text Special Effect

Diese Funktion mit der Nummer 106 kann sicher recht häufig eingesetzt werden. Mit diesem Aufruf kann das Aussehen der Schrift, also der Zeichen auf dem Bildschirm verändert werden. Doch zunächst ein Beispiel:

```
100   fullw 2:clearw 2
110   a$    = "dies ist normal       , intin = "
120   a$(0) = "dies ist fett         , intin = "
130   a$(1) = "dies ist light        , intin = "
140   a$(2) = "dies ist kursiv       , intin = "
150   a$(3) = "dies ist unterstrichen, intin = "
160   a$(4) = "dies ist outlined     , intin = "
170   gotoxy 6,3
180   ?a$;i
190   for i=0 to 4
200   gotoxy 6,5+2*i
210   poke contrl   ,106
```

```
220   poke contrl+ 2,0
230   poke contrl+6 ,1
240   poke intin    ,2^i
250   vdisys
260   ? a$(i);2^i
270   next
280   poke contrl   ,106
290   poke contrl+ 2,0
300   poke contrl+6 ,1
310   poke intin    ,0
320   vdisys
330   a=inp(2)
```

Mit diesem Beispiel werden alle grundsätzlich möglichen Texteffekte gezeigt. Allerdings können die Effekte auch gemischt werden. So ergibt z.B. der Wert 9 in INTIN den Effekt 'fett/unterstrichen'. In den Zeilen 280 bis 320 wird mit dem Wert Null in INTIN wieder auf den normalen Darstellungsmodus umgeschaltet. Ohne dieses Zurückschalten bleibt der eingestellte Effekt weiterhin gültig. Nur der Editor schaltet mögliche Effekte grundsätzlich wieder ab und kehrt zur normalen Darstellung zurück.

Zeichengröße verändern
Set Character Cell Height, Points Mode

Neben der Veränderung der Zeichenform kann auch die Zeichenhöhe geändert werden. Insgesamt sind sechs verschiedene Zeichenhöhen möglich. Da sich bei diesem Aufruf auch die Zeichenbreite ändert, kommt es allerdings bei den drei großen Zeichenhöhen zu Schwierigkeiten mit der Ausgabe. Bei einer PRINT-Anweisung wird von einer Zeichenbreite von 8 Pixeln ausgegangen. Da die Zeichen aber breiter als die acht Pixel werden können, wird der rechte Rand der Zeichen abgeschnitten. Zumindest aber die drei kleineren Höhen lassen sich problemlos einsetzen. Aber zunächst das Programm:

```
10    fullw 2:clearw 2
20    a$(0) = "ganz klein              , intin = "
30    a$(1) = "schon größer            , intin = "
40    a$(2) = "normal                  , intin = "
50    a$(3) = "noch größer             , intin = "
60    a$(4) = "sehr groß               , intin = "
70    a$(5) = "gigantisch              , intin = "
80    a(0)=1:a(1)=9:a(2)=10:a(3)=16:a(4)=18:a(5)=20
90    gotoxy 6,3
100   for i=0 to 5
110   gotoxy 6,5+2*i
120   poke contrl   ,107
130   poke contrl+ 2,0
140   poke contrl+6 ,1
150   poke intin    ,a(i)
160   vdisys
170   ? a$(i);a(i)
180   next
190   poke contrl,107
200   poke contrl+ 2,0
210   poke contrl+6 ,1
220   poke intin,10
230   vdisys
240   goto 240
```

Das Problem der zu breiten Zeichen läßt sich mit ein wenig Aufwand auch lösen. Dazu mehr im nächsten Programm.

Grafik Text Ausgabe
Text

Die Textausgabe mit der VDI-Funktion 8 erlaubt die korrekte Ausgabe einer Zeichenkette, auch wenn von verschiedenen Texteffekten Gebrauch gemacht wurde. Besonders interessant es diese Funktion für das Ausgeben von vergrößerten Zeichen, die mit der normalen PRINT-Anweisung sonst nur teilweise dargestellt würden.

Die Funktion 8 benötigt verschiedene Parameter. Im INTIN-Array muß der auszugebende Text-String abgelegt werden. Jedes Zeichen belegt das untere Byte eines Speicher-Wortes im Array. In unserem Beispiel wird der auszugebende Text in der Zeile 230 in das INTIN-Array übertragen. Das letzte Zeichen des Textes muß Null sein (Zeile 250).
Weiterhin muß die Ausgabeposition im PTSIN-Array übergeben werden. Die Ausgabeposition bezieht sich immer auf die tatsächlichen Bildschirmkoordinaten, nicht auf eine relative Angabe innerhalb eines Fensters. Das VDI kennt keine Fenster. Diese werden ja vom AES verwaltet. Aus diesem Grunde können alle VDI-Bildschirmausgaben über den ganzen Bildschirm gehen.
Die Positionsangabe bezieht sich üblicherweise auf den Punkt der linken unteren Ecke des auszugebenden Zeichens. Werte, die das Zeichen nach oben aus dem Bildschirm 'herausragen' lassen, sollte man vermeiden.
Doch jetzt zunächst das Programm, bevor wir auf weitere Einzelheiten eingehen.

```
100   a$(0) = "ganz klein"
110   a$(1) = "schon größer"
120   a$(2) = "normal"
130   a$(3) = "noch größer"
140   a$(4) = "sehr groß"
150   a$(5) = "gigantisch"
160   a(0)=1:a(1)=9:a(2)=10:a(3)=16:a(4)=18:a(5)=20
170   yp(0)=50:yp(1)=62:yp(2)=80:yp(3)=100:yp(4)=125:yp(5)=160
180   fullw 2:clearw 2
190   for c=0 to 5
200   a=a(c):a$=a$(c)
210   gosub sethight
220   for i=1 to len(a$(c))
230   poke intin+(i-1)*2,asc(mid$(a$(c),i,1))
240   next
250   poke intin+(i-1)*2,0
260   poke contrl   ,8
270   poke contrl+ 2,1
```

```
280   poke contrl+ 6,len(a$(c))+1
290   poke ptsin    ,100
300   poke ptsin+2  ,yp(c)
310   vdisys
320   next c
330   a=10
340   gosub sethight
350   a=inp(2)
360   end
370   sethight:
380   poke contrl   ,107
390   poke contrl+ 2,0
400   poke contrl+6 ,1
410   poke intin    ,a
420   vdisys
430   return
```

Eine weitere Besonderheit des ST-BASIC finden Sie in den Zeilen 210 und 370. Im ST-BASIC ist es nämlich erlaubt, beliebig Labels in das Programm einzufügen. Diese Labels müssen allerdings am Anfang einer Zeile stehen und mit einem Doppelpunkt abgeschlossen werden. Hinter dem Doppelpunkt kann beliebiger Programmtext stehen. Das schöne an den Labels ist nun, daß sie als Ersatz für Zeilennummern gültig sind. Sowohl die Befehle GOTO, GOSUB und ON GOTO sowie ON GOSUB können mit Labels arbeiten wie auch das RESTORE-Kommando, welches ein beliebiges Setzen des DATA-Zeigers auf bestimmte Elemente in DATA-Statements erlaubt.

In der Zeile 350 schließlich wird auf einen beliebigen Tastendruck gewartet, der dann das Programm beendet.

Richtung der Textausgabe ändern
Set Character Baseline Vector

Jetzt geht's rund - und das im wahrsten Sinne des Wortes. Mit der VDI-Funktion 13 kann die Richtung der Textausgabe geändert werden. Dazu muß in INTIN der gewünschte Winkel eingegeben werden. Allerdings sind nur Winkelschritte von 90 Grad zulässig. Für eine Textausgabe in einem Winkel von z.B. 45 Grad ist der verwendete Dreh-Argorithmus nicht eingestellt. Als weitere Besonderheit muß der Winkel in Zehntel Grad angegeben werden. 90 Grad werden also als 900 übergeben (Zeile 320).

Vergessen Sie nicht, den Winkel nach einer Ausgabe wieder auf Null zu setzen. Sonst bleibt Ihnen fast nur noch die 'blinde' Eingabe von QUIT, um das BASIC zu verlassen, da alle Ausgaben von der Änderung betroffen sind. Damit aber kommt das System nicht zurecht. Das Ergebnis ist ein ganz wilder Zeichensalat auf Ihrem Monitor.

```
100   a$ ="  jetzt gehts rund"
110   fullw 2:clearw 2
120   for winkel = 0 to 3
130   gosub txt.winkel
140   for i=1 to len(a$)
150   poke intin+(i-1)*2,asc(mid$(a$,i,1))
160   next
170   poke intin+(i-1)*2,0
180   poke contrl   ,8
190   poke contrl+ 2,1
200   poke contrl+ 6,len(a$)+1
210   poke ptsin    ,300
220   poke ptsin+2  ,200
230   vdisys
240   next winkel
250   a=inp(2)
260   winkel =0:gosub txt.winkel
270   end
280   txt.winkel:
```

```
290   poke contrl   ,13
300   poke contrl+ 2,0
310   poke contrl+6 ,1
320   poke intin    ,winkel*900
330   vdisys
340   return
```

Linienmuster festlegen
Set Polyline Line Type

Bei der Beschreibung des LINEF-, CIRCLE- und ELLIPSE-Befehls haben wir ja bereits angedeutet, daß die standardmäßige Einstellung der Linien geändert werden kann. Jetzt kommt für Sie der Moment, wo diese Änderung tatsächlich möglich wird. Dazu benötigen Sie den VDI-Aufruf mit der Nummer 15. Insgesamt kennt der ST sieben verschiedene Linienmuster. Die Nummer des gewünschten Musters muß in INTIN übergeben werden. Weitere Parameter sind nicht erforderlich, so daß das benötigte Programm zur Demonstration recht klein ausfallen kann.

```
10    fullw 2:clearw 2
20    i=20
30    for muster = 1 to 7
40    gosub set.muster
50    for c=1 to 20
60    linef 20,c+i,500,c+i
70    next c
80    i=i+30
90    next muster
100   a=inp(2)
110   end
120   set.muster:
130   poke contrl   ,15
140   poke contrl+ 2,0
150   poke contrl+ 6,1
160   poke intin    ,muster
170   vdisys
180   return
```

In diesem Programm werden alle 7 Muster jeweils 20 mal untereinander gezeichnet. Auch das Muster 7 erscheint als solider Strich. Allerdings läßt sich das Muster 7 vom Anwender selbst definieren. Das wollen wir einmal versuchen.

Linienmuster 7 definieren
Set User-defined Line Style Pattern

Die benötigte Funktionsnummer zum Programmieren des definierbaren Linienmusters dient der Opcode 113. Bei dieser Funktion muß in INTIN das gewünschte Linienmuster als Bitmuster in 16 Bit eingetragen werden. Dabei ist zu berücksichtigen, daß das höchstwertige Bit immer das erste, also das linke Pixel einer Linie ist.

```
100   fullw 2:clearw 2
110   poke contrl   ,113
120   poke contrl+ 2,0
130   poke contrl+ 6,1
140   poke intin    ,&haaaa : ' pattern
150   vdisys
160   poke contrl   ,15
170   poke contrl+ 2,0
180   poke contrl+ 6,1
190   poke intin    ,7 : ' muster
200   vdisys
210   for c=1 to 20
220   linef 20,c+i,500,c+i
230   next c
240   a=inp(2)
```

An Stelle des von uns gewählten Bitmusters &hAAAA entsprechend %1010101010101010 können Sie ja einmal eigene Bitmuster probieren. Versuchen Sie doch einmal herauszufinden, wo das höchstwertige Bit gezeichnet wird, wenn die Linie vertikal verläuft.

Strichdicke ändern
Set Polyline Line Width

Wenn Sie Programme mit ausgiebiger Nutzung der Grafik-Befehlē schreiben, so kann dieser Opcode (16) viel Programmieraufwand ersparen. Mit dem Aufruf läßt sich die Strichdicke festlegen. Damit ist es dann nicht mehr erforderlich, für dickere Striche die LINEF oder CIRCLE-Funktion wiederholt aufzurufen. Auch die Geschwindigkeit des Programms steigt, da es weniger Zeit benötigt, einen dicken Strich an Stelle mehrerer dünner Linien zu zeichnen.

Die Funktion 16 benötigt als Parameter die gewünschte Strichdicke in INTIN. Als Werte für die Dicke sind ungerade Werte ab 3 zugelassen. Wird als Wert die 2 übergeben, dann wird als Strichdicke eine Pixel-Breite angenommen, der ursprüngliche Wert also wieder eingestellt.

```
100   fullw 2:clearw 2
110   i=20
120   linef 20,c+i,500,c+i
130   i=i+24
140   for c = 3 to 25 step 2
150   gosub set.width
160   linef 20,  i,500,  i
170   i=i+25
180   next c
190   c=2:gosub set.width
200   a=inp(2)
210   end
220   set.width:
230   poke contrl   ,16
240   poke contrl+ 2,1
250   poke contrl+ 6,0
260   poke ptsin    ,c
270   poke ptsin + 2,0
280   vdisys
290   return
```

Dieses Programm zeichnet die verfügbaren Strichdicken von 1 Pixelbreite bis zu einer Breite von 25 Pixeln. Linie kann man das letzte Resultat schon fast nicht mehr nennen. Das Ergebnis ist schon fast ein Balken.

Aussehen der Endpunkte
Set Polyline End Style

Man muß wirklich lange suchen, bis man einen Computer findet, dessen BASIC solche ausgefuchsten Spielereien ermöglicht. Mit dem Aufruf 108 kann nämlich das Aussehen der Endpunkte einer Linie bestimmt werden. Wenn Sie das vorige Programm einmal haben laufen lassen, so werden Sie gesehen haben, daß die Endpunkte der Linien wie abgeschnitten, also rechteckig aussahen. Das ist die Standard-Einstellung bei der Darstellung von Linien. Allerdings können die Endpunkte auch abgerundet dargestellt werden. Für einen technisch-wissenschaftlichen Einsatz ist es weiterhin vorgesehen, die Endpunkte in Pfeil-Darstellung zu zeichen.

Versuchen Sie das doch einmal in BASIC. Das ist eine ganz enorme Rechnerei, die besonders bei größeren Strichdicken Probleme schafft, mindestens aber recht lange dauert. Aber das ATARI-BASIC bzw. das VDI entlastet uns von diesen Problemen. Wir können uns die gewünschte Strichdicke und das Aussehen von Anfang und Ende der Linie (getrennt) einstellen.

Zugegeben, diese Funktion wird sicher nicht jeden Tag in jedem Programm benötigt, kann aber doch so manche Aufgabe deutlich erleichtern.

Die gemachten Ausführungen treffen natürlich auch für CIRCLE- und ELLIPSE-Befehl zu. Besonders für Ausschnitte von Kreisen und Ellipsen ist diese Möglichkeit reizvoll. Modifizieren Sie doch einmal das Beispielprogramm in der Art, daß an Stelle des LINEF-Befehls ein CIRCLE-Auschnitt gezeichnet wird (Zeilen 140 und 180).

Als Parameter muß das gewünschte Aussehen von Anfang und Ende der Linie in INTIN und INTIN+2 übergeben werden. Möglich sind die Werte 0, 1 und 2. Der Wert 0 ist die nach dem Einschalten verfügbare rechteckige Form. Bei einer 1 werden Anfang und/oder Ende als Pfeil dargestellt und bei einer 2 erhalten Sie die abgerundete Form.

```
100   anfang = 0:ende =0
110   gosub set.end
120   i=20
130   fullw 2:clearw 2
140   linef 20,c+i,500,c+i
150   i=i+24
160   for c = 3 to 15 step 2
170   gosub set.width
180   linef 20,  i,500,  i
190   i=i+35
200   next c
210   c=2:gosub set.width
220   a=inp(2)
230   if a=27 then ende = 0:anfang = 0:gosub set.end:end
240   ende = ende +1
250   if ende = 3 then ende = 0:anfang = anfang +1
260   if anfang = 3 then anfang = 0
270   gosub set.end
280   goto 120
290   end
300   set.width:
310   poke contrl   ,16
320   poke contrl+ 2,1
330   poke contrl+ 6,0
340   poke ptsin    ,c
350   poke ptsin + 2,0
360   vdisys
370   return
380   set.end:
390   poke contrl   ,108
```

```
400   poke contrl+ 2,0
410   poke contrl+ 6,2
420   poke intin    ,anfang
430   poke intin + 2,ende
440   vdisys
450   return
```

Dieses Programm basiert auf dem vorigen zur Einstellung der Strichdicke. Sie können also das vorige Programm entsprechend erweitern und müssen es nicht vollständig neu eingeben.

Nach einem Durchlauf wird auf einen beliebigen Tastendruck außer ESC gewartet. Danach wird das Bild mit neuen Endpunkten gezeichnet. Ein Druck auf die ESC-Taste beendet das Programm und setzt die Parameter auf den Einschaltwert zurück.

Damit aber genug von Linien und Linienmustern. Es gibt noch andere Funktionen, die es zu ergründen gilt.

Abfrage der Mausposition
Sample Mouse Button State

Wenn man sich zum ersten Mal mit dem BASIC des ST geschäftigt, wird man mit Bedauern feststellen, daß keine Funktion zur Abfrage der Maus implementiert ist. Aber auch bei dieser Aufgabe hilft uns das VDI aus der Patsche. Mit der Funktion 124 erhält man nicht nur die genaue momentane Position der Maus geliefert, sondern erfährt auch, ob Tasten der Maus gedrückt werden. Diese Funktion benötigt außer dem Opcode in CONTRL keine weiteren Parameter.

Nach dem VDISYS-Aufruf findet sich in INTIN ein Wert, der Aufschluß über gedrückte Tasten an der Maus gibt. Bei einem Wert von 0 ist zum Zeitpunkt der Abfrage keine Taste gedrückt. Beträgt der Wert 1, so ist die linke Taste gedrückt, eine 2 signalisiert eine gedrückte rechte Taste.

Werden beide Tasten gleichzeitig gedrückt, so erhält man den Wert 3 in INTIN zurück.

In PTSOUT erhält man die X-Position der Maus geliefert. Dieser Wert bezieht sich wie alle Positionsangaben des VDI auf die tatsächliche Bildschirm-Position, nicht auf eine relative Angabe innerhalb eines Fensters. PTSOUT + 2 enthält die Y-Position der Maus zur Zeit der Abfrage.

Wir haben die VDI-Funktion 124 in ein etwas umfangreicheres Programm eingebaut, welches recht schön die Verwendung der Maus zum Aufbau von Auswahl-Menues zeigt. Dabei haben wir mit Tricks nicht gespart, so daß Sie dieses Programm einmal in Ruhe analysieren sollten.

```
10   a$(1)="Programm laden"
20   a$(2)="Programm starten"
30   a$(3)="Daten speichern "
40   a$(4)="Programm beenden"
50   p(1)=7:p(2)=8:p(3)=9:p(4)=10
60   aktiv = 1 : inaktiv = 2
70   fullw 2:clearw 2
80   gotoxy 5,5 : ? "bitte wählen Sie :"
90   effekt = inaktiv : gosub text.effekt :gosub 210
100  gosub mouse.button
110  auswahl = int((y.pos-108)/16) :rem gotoxy 1,1:?y.pos
120  gosub mouse.an : if button = 0 then 100
130  gosub mouse.aus
140  if auswahl <1 or auswahl > 4 then 90
150  gosub 210
160  effekt = aktiv : gosub text.effekt
170  gotoxy 5,p(auswahl) : ? a$(auswahl)
180  if auswahl <> 4 then effekt = inaktiv else effekt =0
190  gosub text.effekt
200  if auswahl =  4 then end else 100
210  for i=1 to 4
220  gotoxy 5,p(i) : ? a$(i)
230  next i
240  return
```

```
250   goto 100
260   '
270   mouse.an: rem ************************
280   poke contrl   ,122
290   poke contrl+2 ,0
300   poke contrl+6 ,1
310   poke intin    ,0
320   vdisys
330   return
340   '
350   mouse.aus: rem ************************
360   poke contrl   ,123
370   poke contrl+2 ,0
380   poke contrl+6 ,0
390   vdisys
400   return
410   '
420   mouse.button: rem *********************
430   poke contrl   ,124
440   poke contrl+2 ,0
450   poke contrl+6 ,0
460   vdisys
470   button=peek(intout)
480   x.pos =peek(ptsout)
490   y.pos =peek(ptsout+2) - 38
500   return
510   '
520   text.effekt: rem **********************
530   poke contrl   ,106
540   poke contrl+2 ,0
550   poke contrl+6 ,1
560   poke contrl+10,1
570   poke intin,effekt
580   vdisys
590   return
```

Wird das Programm gestartet, so erhalten Sie auf dem Bildschirm ein kleines Menue. Mit der Maus können Sie die einzelnen Menue-Punkte anfahren und durch Klicken anwählen.

Die ersten drei Punkte sind in diesem Beispiel nicht belegt. Wenn Sie jedoch den vierten Punkt anklicken, so wird das Programm beendet.

Zur Kenntlichmachung des angewählten Menue-Punktes wird dieser in Fettschrift ausgegeben, die anderen Punkte erscheinen in der Lighten-Schrift. Diese Festlegung wird in der Zeile 60 vorgenommen. Wählen Sie die Variablen 'aktiv' und 'inaktiv' nach ihrem eigenen Geschmack.

Zur Auswahl wird in unserem Beispiel nur die Y-Position benötigt. Der vom VDI gelieferte Wert wird in der Zeile 110 in eine Ausgabezeile umgerechnet. Hier müssen Sie für Ihre Programme die wesentlichen Änderungen vornehmen. Um die tatsächlichen Y-Positionen leichter bestimmen zu können, sollten Sie die REM-Anweisung in der Zeile 110 entfernen. Dann wird die Y-Position bei jeder Änderung in der linken oberen Ecke angezeigt.

Schreibmodus setzen
Set Writing Mode

Diese Funktion erweitert alle Möglichkeiten der Ausgabe beträchtlich. Mit der Funktion 32 kann der Schreibmodus bestimmt werden. Bisher haben wir alle Ausgaben im Replace-Modus vorgenommen. Sollte bei der Ausgabe im Replace-Mode bereits etwas auf dem Bildschirm stehen, so wurde das einfach überschrieben oder ersetzt. Für manche Anwendungen ist das aber nicht unbedingt günstig. Der Transparent-Mode z.B. löscht nicht den Hintergrund bei einer Ausgabe. Der Xor-Mode arbeitet nach dem Prinzip des Exklusiv-Oders, einer logischen Verknüpfung zweier Werte. Nur dann, wenn der eine oder der andere Wert 1 ist, wird auch das Ergebnis eine 1. Sind beide Werte Null oder beide Werte 1, so ist das Ergebnis 0. In diesem Fall wird kein Punkt auf dem Bildschirm ausgegeben. Die letzte Möglichkeit stellt der sogenannte Reverse Transparent Mode dar.

Wenn Sie das abgedruckte Beispielprogramm ausprobieren wollen, so sollten Sie beim ersten Lauf die Zeile 160 durch ein REM oder die Abkürzung, den "'" ausschalten oder gar nicht erst eingeben. In diesem Fall gehen die Ausgaben auf den leeren Bildschirm. Im zweiten Durchlauf probieren Sie dann einmal die Ausgabe mit der Zeile 160. Die Resultate erklären vermutlich die verschiedenen Modi besser als langatmige Erklärungen.

```
100   fullw 2:clearw 2
110   a$(1)="normaler text, replace mode"
120   a$(2)="text  im  transparent  mode"
130   a$(3)="dieser text im  xor - modus"
140   a$(4)="text  reverse  transparent!"
150   color 1,1,1,2,2
160   fill 1,1
170   for i=1 to 4
180   gosub set.wrt.mode
190   gotoxy 10,6+i: ?a$(i)
200   next
210   a=inp(2) : i=1
220   gosub set.wrt.mode
230   end
240   set.wrt.mode:
250   poke intin   ,1
260   poke contrl  ,32
270   poke contrl+2,0
280   poke contrl+6,i
290   vdisys
300   return
```

1.2.3 Der GEMSYS-Befehl

Dieser Befehl hat eine ähnliche Funktion wie der Befehl VDISYS. Er ruft das AES des GEM auf. Allerdings unterscheidet sich die Handhabung des GEMSYS-Aufrufes deutlich von den VDISYS-Aufrufen.

Auch für einen AES-Aufruf müssen Parameter in verschiedenen Arrays übergeben werden. Die Parameter-Arrays des AES sind mit denen des VDI nicht identisch. Auch stehen die Adressen der insgesamt 6 verschiedenen Arrays nicht unmittelbar zur Verfügung. Statt dessen existiert eine Tabelle mit den Adressen der Arrays. Diese Tabelle kann über die reservierte BASIC-Variable GB angesprochen werden.
Diese Tabelle besteht aus insgesamt 24 Bytes oder 6 Adressen zu je 32 Bit. Die einzelnen Arrays des AES tragen die von DIGITAL RESEARCH mehr oder weniger festgelegten Namen CONTROL,GLOBAL,INT.IN,INT.OUT,ADRR.INundADDR.OUT. Wir wollen es in den folgenden Beispielen bei diesen Namen belassen, wenn wir die Arrays ansprechen.

Das CONTROL-Array arbeitet ganz ähnlich wie das CONTRL-Array des VDI. Um dieses Array muß man sich als BASIC-Programmierer aber überhaupt nicht kümmern, da dieses Array bei einem GEMSYS-Aufruf vom BASIC aus versorgt wird.

Die zweite Adresse in der GB-Tabelle zeigt auf das GLOBAL-Array. In diesem Array sind verschiedene Parameter enthalten, die nicht verändert werden sollten. Die Werte werden von GEM festgelegt.

Die verbleibenden Arrays des AES arbeiten wie die entsprechenden Arrays des VDI. Allerdings muß bemerkt werden, daß die Elemente der INT.-Arrays in Wort-Größe eingegeben werden müssen, die Elemente der ADDR.-Arrays jedoch als LONG-Werte betrachtet werden, da hier bei verschiedenen Aufrufen Adressen eingetragen werden müssen.

Nun ist es bei den Funktionen des AES so, daß viele

Möglichkeiten unter BASIC nicht oder nur mit einem enormen Programmieraufwand einzusetzen sind. Gerade die Möglichkeiten des AES in Bezug auf die Tastatur oder die Maus werden mit sehr ausgeklügelten Interrupt-Routinen behandelt. Hier hat man mit BASIC überhaupt keine Chance einzugreifen. Versuche, dies doch zu tun werden vom Betriebssystem mit Absturz quittiert.
Wenn Sie wirklich Wert darauf legen, die Funktionen des AES vollwertig einzusetzen, so sollten Sie sich mit dem Gedanken anfreunden, eine Sprache wie z.B. C, Pascal oder Modular 2 zu erlernen (wenn Sie das nicht sowieso schon getan haben). Erst in diesen Sprachen lassen sich die AES-Funktionen effektiv und mit der erforderlichen Geschwindigkeit einsetzen.

Trotzdem wäre es sicher nett, wenn man auf einige Möglichkeiten zugreifen könnte. Wie wäre es z.B., wenn man den Namen des OUTPUT-Windows ändern könnte? Wir haben nicht gerastet und geruht (und die RESET-Taste des ST fürchterlich gequält), bis wir diesen Trick vorführen konnten. Das folgende Beispielprogramm zeigt es.

```
10    gosub gem.arrays
20    x1=0:a$="Dies ist unser Ausgabe-Window"
30    poke int.in  ,3
40    poke int.in+2,2
50    x1=varptr(a$)
60    poke int.in+4,peek(x1+2)
70    poke int.in+6,peek(x1+4)
80    poke int.in+8,0
90    poke int.in+10,0
100   gemsys 105
110   end
120   '
50000 gem.arrays:
50003 int.in   = peek(gb+8) *2^16 + peek(gb+10)
50007 return
```

Wie geht's? Im Unterprogramm 'gem.arrays' wird in der Zeile

50003 die Adresse des int.in-Arrays berechnet und in der gleichnamigen Variablen gespeichert. Die Variable xl muß zu diesem Zeitpunkt eingerichtet werden, um das Ergebnis der späteren VARPTR-Funktion nicht zu verfälschen. In a$ ist der Text für das Ausgabe-Fenster enthalten. Hier können Sie einen Text der eigenen Wahl eintragen. Allerdings sollten Sie keinen 'Roman' für die Titel-Zeile schreiben, da sonst der Text nicht in den vorgesehenen Platz paßt. Die Länge des Textes sollte etwa 20 Zeichen nicht übersteigen.

Anschließend werden die benötigten Parameter in unser int.in-Array gePOKEd. Damit unterscheidet sich der GEMSYS-Aufruf nicht vom VDISYS. In den Zeilen 60 und 70 übergeben wir dem Array die Adresse des Strings, die wir aus dem Ergebnis der VARPTR-Funktion (Zeile 50) ermitteln. Sollten Sie diese Parameterübergabe nicht vollständig verstehen, so können Sie ja noch einmal die Funktion des VARPTR im vorigen Kapitel nachlesen.

Interessanterweise POKEn wir überhaupt keine Werte in das CONTROL-Array. Dafür geben wir dem GEMSYS direkt die gewünschte Funktionsnummer mit. Aus dieser Nummer berechnet das BASIC die Werte für das CONTROL-Array und trägt diese auch selber ein. Dieser 'Luxus' wäre auch beim VDISYS wünschenswert.

1.3 Die Geschwindigkeit des BASIC

Natürlich hat uns brennend interessiert, wie wohl die Verarbeitungsgeschwindigkeit des BASIC ist. Wie aber soll man beim ST die Rechengeschwindigkeit ermitteln, wo dieser doch im BASIC keine Möglichkeit zur Zeitmessung bietet. Nun, das Problem läßt sich glücklicherweise relativ einfach lösen. Das Betriebssystem enthält einen interruptgesteuerten Zähler in den Speicherzellen von $4BA bis $4BD. Der Inhalt dieser Speicherzellen wird als Long, also als 32-Bit Wert aufgefaßt. 200 mal in der Sekunde wird der Long-Wert um eins erhöht. Das entspricht einer Zeit von 5 Millisekunden. Die meisten Befehle werden aber in deutlich kürzerer Zeit abgearbeitet. Darum sollte man den Befehl in einer Schleife sehr oft ausführen und dann die Ergebniszeit durch die Anzahl der Durchläufe teilen. Nach Abzug der für die FOR-NEXT-Schleife benötigten Zeit erhält man recht genau die Zeit für die Ausführung eines Befehls.

Wir haben mit dem folgenden kleinen Programm die Ausführungszeiten der wesentlichen Befehle ermittelt:

```
100   timer = &h4bc
110   zeit1 = peek(timer)
120   for i = 1 to 10000
130   let a = 1
140   next i
150   zeit2 = peek(timer)
160   zeit  = zeit2 - zeit1
170   zeit  = (zeit*5/10000)-.8495
180   ? "der befehl in 130 benötigt"zeit"millisekunden"
```

Im Durchschnitt benötigen alle Befehle des BASIC etwa zwischen 0.6 und 1.9 Millisekunden. Das Schlußlicht stellt der PRINT-Befehl dar. Die Zeit für die Ausgabe eines einzelnen Zeichens liegt bei etwa 4.5 Millisekunden. Allerdings ist die exakte Ermittlung eines PRINT nicht so

ganz einfach. Wir haben zunächst die Zeit für den GOTOXY-Befehl ermittelt und die Zeit notiert. Anschließend haben wir die Zeit für die Zeile:

```
130  gotoxy 0,0:?"a";
```

ermittelt und die Zeit für den GOTOXY abgezogen. Beachten Sie unbedingt das Semikolon am Ende der PRINT-Ausgabe. Ohne das Semikolon werden nach dem 'a' noch die Zeichen CR und LF ausgeführt. Die dafür benötigte Zeit hat aber mit dem PRINT-Befehl nicht ursächlich zu tun.

Ganz fürchterlich werden die Zeiten, wenn man die GOTOXY-Anweisung und das Semikolon weg läßt. Dann wird bei (fast) jeder Ausgabe der Bidschirm gescrollt. Wenn Sie wirklich einmal die dafür benötigte Zeit ermitteln wollen, so sollten Sie die Anzahl der Schleifendurchläufe auf 100 bis 200 reduzieren (Zeilen 120 und 170). Sonst dauert der Probelauf 'ewig'.

Die interessantesten Ergebnisse liefern übrigens die numerischen Fließkomma-Funktionen wie SQR, SIN oder LOG. Diese Funktionen sind 'affenschnell'. Vergleicht man die ermittelten Zeiten mit denen anderer Rechner, so stellt man eine enorme Geschwindigkeitssteigerung fest. Die Zeiten für die SQR-Funktion des C64 liegen bei etwa 54 Millisekunden, beim Schneider CPC liegen sie immerhin noch bei 27 Millisekunden. Bei ST benötigt die Funktion sage und schreibe ca. 1 Millisekunde!
Diese im Vergleich zu den anderen Funktionen sensationellen Zeiten haben ihre Ursache in dem Umstand, daß das BASIC in ganz großen Teilen in C geschrieben ist. Nur die Fließkomma-Funktionen sind direkt in Assembler geschrieben. Autor der Routinen für die Fließkomma-Funktionen ist übrigens die Firma MOTOROLA, die als Entwickler des 68000 eigentlich wissen sollte, wie man die Geschwindigkeit des Prozessors voll ausreizt.

1.4 BASIC und Maschinensprache

Ob es denn wirklich nötig ist, BASIC und Maschinensprache zu vermischen mögen Sie jetzt fragen. Es ist doch wirklich (fast) alles von BASIC aus möglich. Nun, genau dieses kleine Wörtchen 'fast' ist es, daß bei uns den Ausschlag gab, kleine Maschinen-Routinen von BASIC aus aufzurufen. Der erste Problem-Fall war die Uhrzeit. Da wird nun vom Betriebssystem eine recht genaue, in 2-Sekunden-Schritten laufende Uhr mitgeführt, aber von BASIC aus kommt man nicht an die Uhrzeit heran. Da mußte Abhilfe geschaffen werden.

1.4.1 'Ruhige' Plätze für Maschinenprogramme

Wenn es darum geht, in BASIC-Programmen die Uhrzeit auszugeben oder gar einzustellen, so bemüht man in der Regel das Kontroll-Feld. Dann kann man zwar die eingestellte Uhrzeit ablesen, muß sie aber über die Tastatur wieder an ein BASIC-Programm übergeben. Das ist auf Dauer natürlich kein Zustand. In Maschinensprache ist das Problem schnell gelöst. Damit stellt sich aber die Frage, wie man ein solches Programm möglichst problemlos mit dem BASIC zusammen bringt. Dafür haben wir mehrere Lösungen gefunden, die wir jetzt zeigen wollen.

Die sicher einfachste Lösung besteht darin, den freien Speicher 'über' dem Bildschirm zu verwenden. Die Speicherorganisation des ST sieht so aus, daß das Bildschirm-RAM immer die oberen 32 KByte des Speichers benutzt. Bei 260ST und beim 520ST bedeutet das eine Adresse von $78000. Beim 520ST+ lautet die Adresse $F8000. Der Bildschirm selbst belegt nur 640*400 Bit = 32000 Bytes. Die letzten 768 Bytes des Speichers werden nie vom Betriebssystem benutzt. Hier können Programme hingePOKEd werden, vorausgesetzt sie passen von der Größe her in den Bereich.

Allerdings gibt es einige Schwierigkeiten bei dieser Lösung.

Jedes Programm, das diesen Bereich belegt, müßte selbstständig ermitteln, ob es sich um einen Rechner mit 512 K oder 1024 KByte handelt und die Adresse entsprechend wählen. Dieses Problem ist sicher noch zu lösen. Es gibt aber einen weiteren Grund, diesen Bereich nicht zu belegen. Hat ein Rechner einen solchen unbenutzten Bereich, so wird dieser sehr warscheinlich von vielen verschiedenen Programmen in der Annahme genutzt, der Bereich sei nicht belegt. Spätestens wenn zwei Programme gleichzeitig diesen Bereich für sich in Anspruch nehmen, gibt es wirkliche Probleme.

Wohin also dann mit den Routinen? Es gibt einen alten und auch beim ST anwendbaren Trick. Man packt sie einfach in einen String. Dem BASIC-Interpreter ist es völlig egal, ob in der Variable A$ der Text 'Heini ist Doof' steht, oder ob wir ein Maschinenprogramm in den String 'basteln'. Probieren wir das doch einmal konkret. Als Beispiel soll eine Funktion entstehen, die uns die Uhrzeit liefert. Dafür benötigen wir zuerst ein passendes Assemblerprogramm.

In Maschinensprache kann man die Uhrzeit auf recht einfache Weise ermitteln. Wird der Wert $2C auf den Stack gelegt und anschließend der Befehl TRAP #1 ausgeführt, so erhält man nach dem TRAP die Zeit im Register D0 zurück. Leider steht die Zeit nicht im 'Klartext' in D0. Sie ist in einzelnen Bits verschlüsselt und muß entsprechend decodiert werden. Wie dies geschieht zeigen wir etwas später.

```
000000  move.l   a0,a5          adresse der routine nach a5
000002  move.w   #$2c,-(a7)     funktionsnummer uhrzeit holen
000004  trap     #1             funktion ausführen
000008  addq.l   #2,a7          stackpointer reparieren
00000a  move.w   d0,$10(a5)     uhrzeit in speicher schreiben
00000e  rts
000010  ds.w     1              platz für zeit
```

Steht dieses Programm assembliert im Speicher und wird es von BASIC aus mit einem CALL-Befehl aufgerufen, so steht die

Adresse der Routine im Register A0. Diese Adresse merken wir uns im Register A5, da wir sie später noch benötigen. Als nächstes legen wir die schon erwähnte Funktionsnummer auf den Stack und führen anschließend den TRAP-Befehl aus. Nach dem TRAP müssen wir den Stackpointer, das Register A7, wieder auf den ursprünglichen Wert bringen, da immer noch die Funktionsnummer auf dem Stack liegt.

Jetzt wird die in D0 gelieferte Uhrzeit in den Speicher übertragen. Dabei hilft uns das Register A5, in dem wir die Adresse des ersten Befehls unseres Programms gespeichert haben. Als Speicher für die Zeit wählen wir die Adresse, die relativ zum Anfang in A5 16 Bytes entfernt liegt. Diese Adresse liegt genau hinter dem RTS-Befehl der das Programm beendet.

Wenn das Programm assembliert wird, so erhält man die Opcodes, also die Hex-Zahlen, die der Prozessor im Endeffekt als Programm ausführt. Für unser Beispiel erhalten wir die Hex-Werte:

```
$2a,$48,$3f,$3c,$00,$2c,$4e,$41,$54,$8f,$3b,$40,$00,$10,$4e,$75
```

Diese Werte müssen wir jetzt in eine String-Variable bekommen. Das könnte mit dem folgenden Programm geschehen:

```
10    for i=0 to 17
20    read byte
30    uhr$ = uhr$ + chr$(byte)
40    next i
50    data &h2a,&h48,&h3f,&h3c,&h00,&h2c,&h4e,&h41
60    data &h54,&h8f,&h3b,&h40,&h00,&h10,&h4e,&h75
70    data &hff,&hff
```

Die letzten beiden Werte sind besonders wichtig. Hier wird ja später die Uhrzeit abgelegt. Der Platz für das Ergebnis muß natürlich im String mit angelegt werden, da sonst kein Zugriff auf das Ergebnis möglich wäre.

Damit wäre der erste Teil geschafft. Wir haben unseren String fertig. Jetzt müssen wir einen Weg finden, wie wir die Routine ausführen können. Dazu muß aber bekannt sein, wo der String im Speicher steht. Die Adresse wird ja als Parameter für den CALL-Befehl benötigt. Dazu verwenden wir die VARPTR-Funktion. Sie liefert als Ergebnis die Adresse des Stringdescriptors (siehe auch Beschreibung des VARPTR zu Beginn des BASIC-Kapitels). Im dritten bis sechsten Byte des Descriptors findet man die Adresse des Strings und damit des Uhrenprogramms im Speicher.

```
70    adress = 0
80    adress = peek(varptr(uhr$)+2)
```

Dankenswerter Weise liefert die PEEK-Funktion die vollständige 32-Bit-Adresse des Strings in die Variable 'adress'. Üblich wäre ein 16-Bit-Wert als Ergebnis. Wird jedoch als PEEK-Adresse das Ergebnis der VARPTR-Funktion genommen, so hilft uns das BASIC, indem zwei 16-Bit-Werte geliefert werden.
Übrigens ist die Zeile 70 recht wichtig. Wäre die Variable 'adress' nicht initialisiert, so würden wir möglicherweise durch die Initialisierung während des VARPTR das Ergebnis verfälschen.

Jetzt können wir endlich zu Tat schreiten und die Uhrzeit ermitteln. Dazu verwenden wir den CALL-Befehl.

```
90    call adress
```

Nach dem Aufruf ist die Uhrzeit in den letzten beiden 'Zeichen' des Strings 'uhr$' enthalten. diesen Teil kann man mit RIGHT$ isolieren und die Uhrzeit daraus berechnen.

```
100   time$ = right$(uhr$,2)
```

Dieses Verfahren mag Ihnen etwas umständlich vorkommen. Es funktioniert zwar, hat aber auch Schwachpunkte. Der wesentliche Schwachpunkt ist die Begrenzung der

Assembler-Routine auf maximal 256 Bytes. Diese Grenze ist durch die maximale Lände eines Strings vorgegeben. Auch ist die Parameterübergabe recht umständlich über einzelne Zeichen im String möglich. Wollen wir gar Parameter an eine Routine übergeben, so wird die Angelegenheit noch komplizierter.

Die dritte Methode, die wir Ihnen vorstellen möchten, beseitigt alle diese Probleme. Sie stellt die flexibelste Möglichkeit dar, Maschinenprogramme in BASIC-Programme einzubinden. Die Routine wird in einem INTEGER-ARRAY abgelegt.
Wenn man sich den Aufbau eines Integer-Array im Speicher des ST ansieht, so stellt man fest, daß die einzelnen Elemente des Arrays hintereinander im Speicher stehen. Das Element mit dem niedrigsten Index liegt auf der niedrigsten Adresse. Jedes Element ist zwei Bytes groß, genau passend für die Opcodes des 68000. Die Größe eines in einem Array untergebrachten Maschinenprogramms ist nicht so stark begrenzt wie bei der String-Methode. Die Programme können ohne weiteres 1000 Bytes oder größer werden.

```
10    dim uhr%(8)
20    for i = 0 to 8
30    read uhr%(i)
40    next i
50    data &h2a48,&h3f3c,&h002c,&h4e41,&h548f
60    data &h3b40,&h0010,&h4e75,&h0000
```

Wie Sie sehen, haben wir bisher das Array dimensioniert (Zeile 10) und anschließend das Programm in das Array übertragen. Allein diese Initialisierung ist um einiges kürzer als beim vorigen Programm, da die Data-Elemente jetzt 16 Bit umfassen.

Aber auch jetzt müssen wir wieder mit Hilfe der VARPTR-Funktion die Adresse unserer Uhr-Routine bestimmen. Allerdings geht's jetzt deutlich einfacher. Das Ergebnis des VARPTR zeigt direkt auf den ersten Befehl der Routine, so

daß wir das Ergebnis als Sprungadresse für den CALL-Befehl verwenden können!

```
70    ad = 0
80    ad = varptr (uhr%(0))
90    call ad
```

Das Ergebnis können wir auch ganz einfach abholen. Es ist im Array-Element uhr%(8) enthalten.

```
100   ?uhr%(8)
```

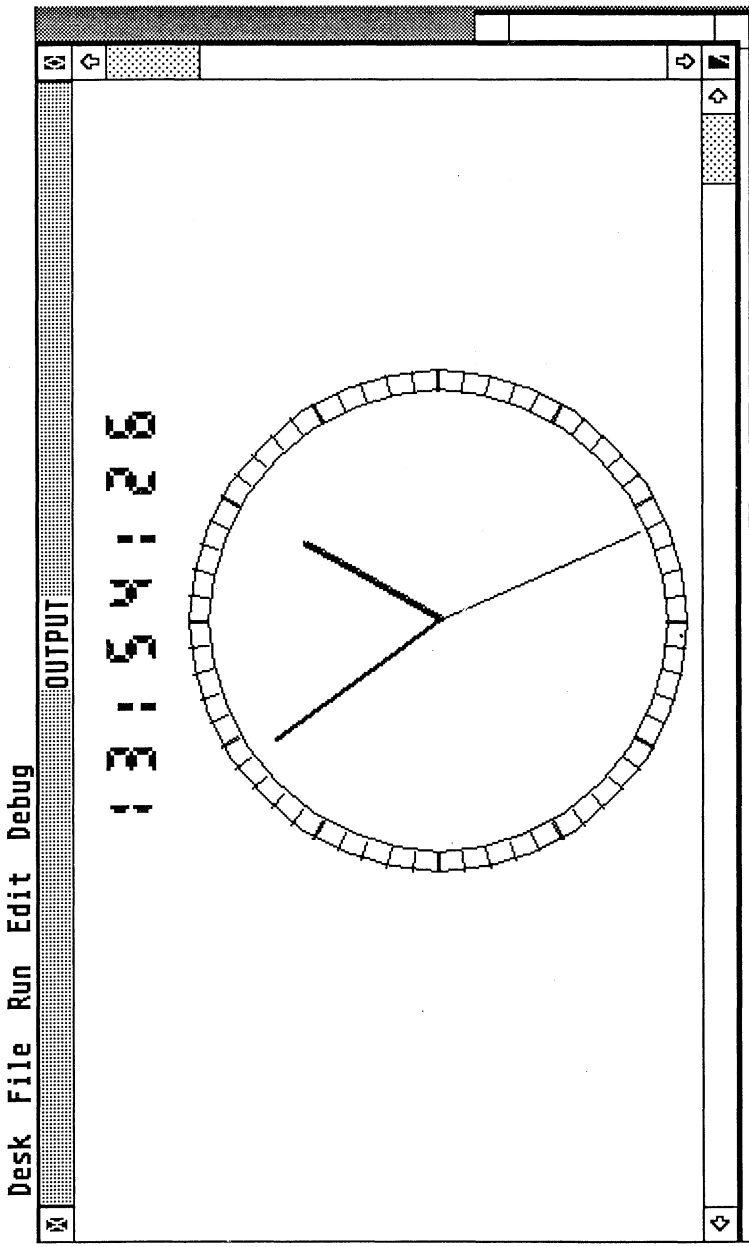
Genau so könnten auch Parameter an die Routine übergeben werden. Sie tragen dazu einfach die Parameter in die entsprechenden Elemente ein und das Programm holt sie sich aus den gewählten Speicherzellen ab. Wenn das nicht flexibel ist - was ist es dann!

1.5 Die teuerste Uhr in Ihrem Haushalt

Es gibt Leute, die Programme wie das Folgende für überflüssig erachten. Wir sind nicht dieser Meinung. Natürlich soll Ihr ST nicht ausschließlich als Uhr sein Dasein fristen. Dazu kann er ganz einfach zu viel. Mit Hilfe eines solchen Programms können aber sehr schön grundsätzliche Programmiertechniken gezeigt werden.

Wir haben im folgenden Programm verschiedene Möglichkeiten der Grafik des ST genutzt und ein etwas modifiziertes Programm zum Lesen der Uhrzeit eingebaut. Bevor wir aber auf einige Besonderheiten des Programms eingehen, zunächst das Programm:

```
1000  h0=10
1010  dim uhr%(23)
1020  xm% = 320:ym% = 200
1030  sec.z% = 115 : min.z% =105 : std.z% = 80
1040  pi=4*atn(1)
1050  g89 = 89.9*(pi/180) : g90 = pi/2 : g91 = 90.1*(pi/180)
1060  uhr  = 0
1070  '
1080  fullw 2:clearw 2
1090  gosub hoehe
1100  gosub init.uhr
1110  gosub zifferblatt
1120  '
1130  loop: '   *************************
1140  uhr = varptr (uhr%(0))
1150  call uhr
1160  if sec% = uhr%(20)*2 then loop
1170  '
1180  ' zeiger löschen ******************
1190  sec% = uhr%(20)*2
1200  color 0,0,0,0,0
1210  dmy% = std% : std% = uhr%(22)
1220  if dmy%<> std% then phi = phistd :r%=std.z%:gosub zeiger
1230  dmy% = min% : min% = uhr%(21)
```


Desk File Run Edit Debug
OUTPUT
13:54:26

```
1240  if dmy%<> min% then phi = phimin :r%=min.z%:gosub zeiger
1250  phi = phisec : r%=sec.z%: gosub zeiger
1260  '
1270  '
1280  ' zeiger neu zeichnen  ************
1290  color 1,0,1,1,1
1300  phisec = sec%*pi/30-g90: r%=sec.z%
1310  phi = phisec : gosub zeiger
1320  phimin = min%* 6 * (pi/180)-g90 : r%=min.z%
1330  phi = phimin : gosub zeiger
1340  phistd = std%* 30 * (pi/180)-g90 : r%=std.z%
1350  phi = phistd : gosub zeiger
1360  gosub digital
1370  goto loop
1380  '
1390  zeiger:   ' ***********************
1400  linef xm%  ,ym%  ,xm%+r%*cos(phi)  ,ym%+r%*sin(phi)
1410  if r%= sec.z% then return
1420  '
1430  linef xm%+1,ym%  ,xm%+r%*cos(phi)+1,ym%+r%*sin(phi)
1440  linef xm%  ,ym%+1,xm%+r%*cos(phi)  ,ym%+r%*sin(phi)+1
1450  linef xm%+1,ym%+1,xm%+r%*cos(phi)+1,ym%+r%*sin(phi)+1
1460  if r%= min.z%then return
1470  '
1480  linef xm%+2,ym%  ,xm%+r%*cos(phi)+2,ym%+r%*sin(phi)
1490  linef xm%  ,ym%+2,xm%+r%*cos(phi)  ,ym%+r%*sin(phi)+2
1500  linef xm%+2,ym%+2,xm%+r%*cos(phi)+2,ym%+r%*sin(phi)+2
1510  return
1520  '
1530  zifferblatt:  '*********************
1540  circle xm%,ym%,120,120
1550  circle xm%,ym%,130,130
1560  for std%= 1 to 12
1570  phi = std%* 30 * (pi/180)-g90 : r1%=130 : r0%=120
1580  linef xm%+r0%*cos(phi),ym%+r0%*sin(phi),xm%+r1%*cos(phi),ym%+r1%*sin(phi)
1590  phi = std%* 30 * (pi/180)-g89 : r1%=130 : r0%=120
1600  linef xm%+r0%*cos(phi),ym%+r0%*sin(phi),xm%+r1%*cos(phi),ym%+r1%*sin(phi)
1610  phi = std%* 30 * (pi/180)-g91 : r1%=130 : r0%=120
1620  linef xm%+r0%*cos(phi),ym%+r0%*sin(phi),xm%+r1%*cos(phi),ym%+r1%*sin(phi)
```

```
1630  next std%
1640  for min%= 1 to 59
1650  phi = min%* (pi/30) : r1%=130 : r0%=120
1660  linef xm%+r0%*cos(phi),ym%+r0%*sin(phi),xm%+r1%*cos(phi),ym%+r1%*sin(phi)
1670  next min%
1680  return
1690  '
1700  digital: ' ************************
1710  sec$=str$(sec%): if len(sec$)=2 then sec$=" 0"+right$(sec$,1)
1720  min$=str$(min%): if len(min$)=2 then min$=" 0"+right$(min$,1)
1730  std$=str$(std%): if len(std$)=2 then std$=" 0"+right$(std$,1)
1740  timdig$=right$(std$,2)+"Z"+right$(min$,2)+"Z"+right$(sec$,2)
1750  gosub printdig
1760  return
1770  '
1780  hoehe:   ' ************************
1790  poke contrl   ,107
1800  poke contrl+2,0
1810  poke contrl+6 ,1
1820  poke intin,h0
1830  vdisys
1840  return
1850  '
1860  printdig: rem *********************
1870  poke contrl   ,11
1880  poke contrl+2 ,2
1890  poke contrl+6 ,10
1900  poke contrl+10,10
1910  poke contrl+12,2
1920  poke intin    ,1
1930  poke intin+2  ,1
1940  for i%=1 to 8
1950  poke intin + (i%*2+2),asc(mid$(timdig$,i%,1))-32
1960  next i%
1970  poke ptsin,210
1980  poke ptsin+2,80
1990  poke ptsin+4,220
2000  poke ptsin+6,0
2010  vdisys
```

```
2020  return
2030  '
2040  init.uhr:   ' **********************
2050  data &h2a48,&h3f3c,&h002c,&h4e41,&h548f,&h3b40,&h0028,&h026d
2060  data &h001f,&h0028,&hea48,&h3200,&h0240,&h003f,&h3b40,&h002a
2070  data &hec49,&h3b41,&h002c,&h4e75,&h0000,&h0000,&h0000,&h0000
2080  for i% = 0 to 23
2090  read uhr%(i%)
2100  next i%
2110  return
```

In den ersten sieben Zeilen werden verschiedene Variablen initialisiert.

h0 die Größe der Schriftzeichen für die Digital-Anzeige. Wenn Sie das Programm abbrechen, dann wird nicht automatisch in die richtige Schriftgröße zurückgeschaltet. Das ist besonders beim Probelauf ärgerlich, wenn Sie noch Tippfehler im Programm haben. Setzen Sie diese Variable für den Probelauf auf 10. Dann arbeiten Sie immer in der normalen Ausgabe.

xm%, ym% bestimmen den Mittelpunkt des Ziffernblattes.

sec.z%, min.z%, std.z% bestimmen die Länge der drei Zeiger.

pi die Zahl pi (3.14159), die im ST-BASIC nicht direkt verfügbar ist.

Danach werden die Schrifthöhe eingestellt, das Array für das Maschinenprogramm eingerichtet und das Ziffernblatt gezeichnet.

In der Zeile 1130 beginnt das eigentliche Programm. Das Maschinenprogramm legt in uhr%(20) die Sekunden, in uhr%(21) die Minuten und in uhr%(22) die Stunde ab. Nach einen Aufruf wird verglichen, ob der neue Sekundenwert mit dem alten Wert identisch ist. Solange dies der Fall ist, wird erneut

geschleift. Erst wenn sich die Sekunden geändert haben, wird die Schleife verlassen.

Der Rest des Programms ist einigermaßen unproblematisch, so daß Sie mit einigen BASIC-Kenntnissen keine Verständnis-Schwierigkeiten haben sollten.

1.6 Automatische Hardcopy

Zur Lösung dieses Problems sind (mindestens) zwei Möglichkeiten vorhanden. Die eine Möglichkeit besteht im Ausführen eines bestimmten VDISYS-Aufrufes. Dazu müssen die folgenden Parameter in die Arrays gepoked werden:

```
contrl    ,5
contrl+  2,0
contrl+  6,0
contrl+ 10,17
```

Es gibt aber einen wesentlich einfacheren Weg. Dafür ist nur ein Poke nötig:

```
poke 1262,0
```

Das genügt. Die angesprochende Speicherzelle enthält normalerweise den Wert -1. Wenn Sie jedoch die ALT- und die HELP-Tasten gleichzeitig drücken, so wird der Inhalt um eins erhöht und die Hardcopy wird ausgelöst. Das kann durch den genannten Poke natürlich leicht simuliert werden.

Übrigens wird die Hardcopy abgebrochen, wenn Sie die genannten Tasten während der Hardcopy ein weiteres mal drücken.

2 Utilities für den ATARI ST

In diesem Kapitel stellen wir Ihnen eine ganze Reihe vom kleinen Programmen vor, die als Hilfsprogramme gute Dienste tun können. Es handelt sich dabei meist um Maschinenprogramme, die Funktionen des Betriebssystems erweitern oder verändern. Sie stehen also neben dem Betriebssystems ständig im Speicher und sollen Ihnen das Arbeiten mit dem Rechner und anderen Anwenderprogramme erleichtern bzw. erst ermöglichen.

Die einzelnen Abschnitte beinhalten eine kurze Einstimmung in die Problematik; dann werden das Programm, seine Anwendung und Möglichkeiten beschrieben. Es folgt dann das Quellprogramm in 68000-Assemblercode. Durch die vorhandenen Kommentare sollte es Ihnen - sofern Sie über einen Assembler verfügen - möglich sein, das Programm bei Bedarf nach Ihren Wünschen zu ändern oder zu erweitern. Damit auch diejenigen zum Zuge kommen, die nur in BASIC programmieren, ist dem Programm jeweils ein BASIC-Lader angefügt. Dieses Programm enthält das Maschinenprogramm in Form von DATA-Statements. Durch die eingebaute Prüfsumme im BASIC-Lader lassen sich Eingabefehler mit hoher Wahrscheinlichkeit feststellen und dann anhand des Listings korrigieren. Wenn Sie dieses Programm von BASIC aus starten, erzeugt es automatisch eine Programmdatei, wie sie sonst nach dem Assemblieren und Linken entsteht. Haben Sie diesen Vorgang einmalig ausgeführt, können Sie das Programm von jetzt an durch Anklicken des Icons mit der Maus starten.

2.1 Die aktuelle Uhrzeit

Im dem Programm, das wir Ihnen jetzt vorstellen wollen, sind gleich drei Spezialitäten enthalten, die Sie für Ihre eigenen Programme benutzen können. Zum einen geht es darum, ein Programm periodisch durch den Systeminterrupt ausführen zu lassen; dann wird gezeigt, wo man ein kurzes Programm im Speicher ablegen kann, ohne daß es durch andere nachgeladene Programme zerstört wird und zum dritten zeigen wir Ihnen, wie Sie die Systemfonts des ATARI ST direkt benutzen können. Doch alles hübsch der Reihe nach.

Das Programm versieht Ihren Rechner mit einer Digitaluhr, die in die obere rechte Ecke des Bildschirms eingeblendet wird. Am vorteilhaftesten arbeitet dieses Programm, wenn Sie auf Desktop-Ebene oder mit GEM-Programmen arbeiten. In diesem Fall dient die oberste Zeile als Statuszeile und die letzten zehn Zeichenpositionen sind normalerweise frei, was beim Arbeiten unter TOS nicht immer gewährleistet ist.

Wie zu Beginn schon angedeutet, muß die Uhrzeit ja ständig neu angezeigt werden. Dazu benutzen wir den Vertical Blank Interrupt (VBL) des Rechners, der jedesmal dann aufgerufen wird, wenn der Rechner ein komplettes Videobild dargestellt hat. Das geschieht bei angeschlossenem Monochrommonitor 70mal in der Sekunde. In der VBL-Routine prüft der Rechner jedesmal eine Sprungleiste von normalerweise 8 Einträgen, die die Adressen von Anwender-Routinen enthalten, die innerhalb des VBL ausgeführt werden sollen. Eine Adresse von Null in dieser Liste bedeutet, daß der entsprechende Eintrag nicht belegt ist. Um eine eigene Routine ausführen zu lassen, muß man also die Liste durchsuchen, bis man einen leeren (Null) Eintrag gefunden hat. Dort kann man die Adresse der eigenen VBL-Routine hinterlegen, die nun bei jedem VBL-Interrupt abgearbeitet wird.

Doch wo legt man sein Programm hin, damit es nicht von anderen Anwenderprogrammen, die man nach der Initialisierung ausführt, überschrieben wird? Wenn das Programm nicht größer

als 3 Pages ($300 gleich 768 Bytes) ist, so kann man es 'hinter' dem Bildschirm unterbringen. Der ATARI ST reserviert nach dem Einschalten die obersten 32 KByte des Speichers für den Bildschirm; bei einer 512-K-Maschine liegt der Bildschirm von $78000 bis $7FFFF. Wenn Sie jedoch einmal nachrechnen, so ergibt sich, daß für 640 * 400 Punkte 256000 Bit bzw. 32000 Byte benötigt werden. 32 KByte sind jedoch 32768 Bytes, so daß die oben genannten 768 Byte 'übrig' bleiben. Der Bildschirmspeicher belegt also nur die Adressen $78000 bis $7FCFF einschließlich; der Bereich von $7FD00 bis $7FFFF wird bei Bildschirmausgaben und auch beim Löschen des Bildschirms nicht angetastet. Dieser Speicherbereich ist der ideale Platz für kleine Routinen, die permanent im Speicher stehen müssen.

Ein Initialisierungsprogramm sollte also das Programm hinter den Bildschirm kopieren und den VBL-Vektor auf diese Adresse setzen. Dies geschieht zu Beginn unseres Programms.

Da wir auf Systemvariablen zugreifen müssen, wird zuerst der Supervisormodus eingeschaltet. Dann wird die VBL-Liste getestet, bis wir einen freien Platz gefunden haben. Die Adresse dieses Eintrags merken wir uns in A2. Die Länge unserer Routine, die wir kopieren wollen, dient uns als Zähler. Als Zieladresse holen wir uns den Bildschirmstart und addieren 32000 dazu, die Länge des Bildschirms. Dann kopieren wir das Programm in den so berechneten Speicherbereich. Dann rufen wir die Routine zur Initialisierung der Zeitanzeige auf. Zum Schluß setzen wir noch den VBL-Vektor auf unsere Routine und kehren zum Desktop zurück.

Die Init-Routine holt sich einen Zeiger auf den verwendeten Fontheader, den wir zur Darstellung der Ziffern benötigen. Dazu wird die Line-A-Routine aufgerufen, die in A1 einen Zeiger auf ein Vektorfeld auf die drei Systemfonts liefert. Wir holen uns dann die Adresse des Fonts Nummer 2, der 8*16 Font, die Standardschrift bei Monochrombetrieb. Dann setzen wir unseren VBL-Zähler auf 1, so daß unsere Anzeigeroutine nach dem nächsten VBL gestartet wird.

Der Zähler wurde eingeführt, da es nicht erforderlich ist, die Zeit 70mal in der Sekunde anzuzeigen. Wir dekrementieren nach jedem Aufruf den Zähler und zeigen die Uhrzeit erst dann an, wenn der Zähler abgelaufen ist. In der Anzeigeroutine wird der Zähler wieder neu geladen. Dann wird eine Routine aufgerufen, die die Uhrzeit vom Tastaturprozessor abholt. Sie ist identisch mit der entsprechenden XBIOS-Funktion. Als Ergebnis liefert diese Funktion die Zeit im DOS-Format mit einer Auflösung von 2 Sekunden. Der Tastaturprozessor übermittelt die Zeit jedoch auf die Sekunde genau im BCD-Format. Diese Information wird vom ATARI ST abgespeichert und zwar ab der Adresse $A46, Label time im Programm. Zur Anzeige bedienen wir uns dieser Information. Die drei Bytes, die Stunde, Minute und Sekunde im europäischen Format enthalten, werden mit der Routine wrtbcd auf den Bildschirm geschrieben, getrennt durch einen Doppelpunkt.

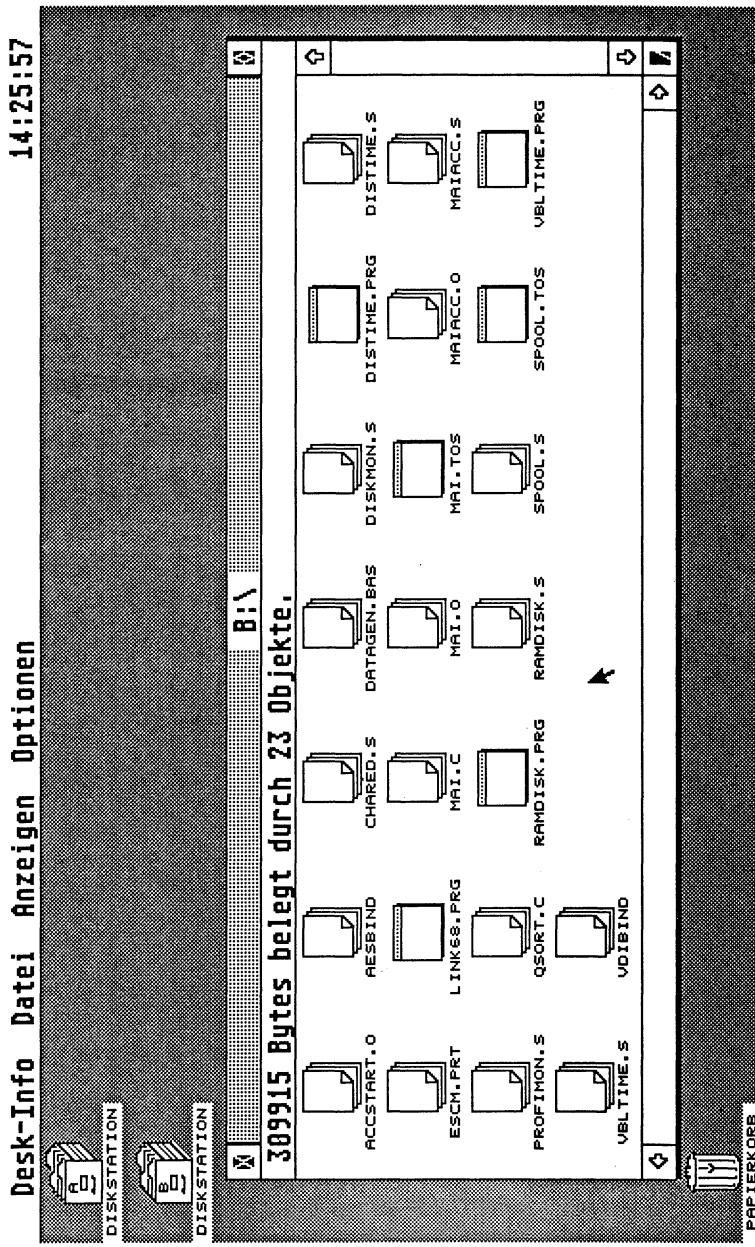
Die Routine wrtchar schreibt ein Zeichen, das in D0 erwartet wird, in die oberste Bildschirmzeile. Die Cursorposition wird dabei in Register D6 erwartet (Wert zwischen 0 und 79). Aus der Cursorposition und der Basisadresse des Bildschirms wird die aktuelle Position im Grafik-RAM berechnet. Dann werden aus dem Fontheader die Adresse der Fontdaten, der Offset für die nächste Rasterzeile sowie die Anzahl der Scanzeilen, die Höhe des Zeichen geholt. Die Schleife mit dem Label loop kopiert Rasterzeile für Rasterzeile direkt aus den Fontdaten auf den Bildschirm, bis das komplette Zeichen geschrieben wurde. Das Programm ist in unveränderter Form auf einem Monochrommonitor lauffähig.

Sie werden vielleicht fragen, warum die Zeichen nicht einfach mit den BIOS- bzw. GEMDOS-Funktionen zur Zeichenausgabe auf den Bildschirm geschrieben werden, sondern relativ aufwendig direkt aus dem Systemfont geholt werden. Die Antwort dafür liegt darin, daß wir es ja mit einer Interruptroutine zu tun haben. Zum einen kann ja während dieser Zeit eine normale Textausgabe unterbrochen worden sein und der Cursor steht an einer ganz anderen

Stelle des Bildschirms. Wir müßten also die Cursorposition retten, den Cursor dann in die oberste Zeile setzen, dort die Zeichen ausgeben, um dann den Cursor wieder an die ursprüngliche Stelle zu setzen, was ebenfalls aufwendig wäre. Weitere Probleme kann es geben, wenn zum Unterbrechungszeitpunkt gerade ein Zeichen ausgegeben wird oder wenn (unter GEM) gerade ein anderer Font benutzt wird. Diese Probleme konnten durch den direkten Zugriff auf den Font umgangen werden. Außerdem ist der direkte Zugriff ohne Benutzung von Systemroutinen noch schneller.

Die Abbildung zeigt die Uhrzeitanzeige in der Statuszeile des Desktop.

Auf den nächsten Seiten finden Sie das Assemblerlisting des Programms. Für diejenigen unter Ihnen, denen kein Assembler zur Verfügung steht, haben wir ein Ladeprogramm in BASIC abgedruckt. Wenn Sie dieses Programm eingeben und starten, erzeugt es automatisch ein Programmfile mit dem Namen 'TIME.PRG' auf Diskette, das Sie wie üblich durch Anklicken mit der Maus starten können.


Desk-Info Datei Anzeigen Optionen
14:25:57
DISKSTATION
DISKSTATION
B:\
309915 Bytes belegt durch 23 Objekte.
ACCSTART.O
AESBIND
CHARED.S
DATAGEN.BAS
DISKMON.S
DISTIME.PRG
DISTIME.S
ESCM.PRT
LINK68.PRG
MAI.C
MAI.O
MAI.TOS
MAIACC.O
MAIACC.S
PROFIMON.S
QSORT.C
RAMDISK.PRG
RAMDISK.S
SPOOL.S
SPOOL.TOS
VBLTIME.PRG
VBLTIME.S
VDIBIND
PAPIERKORB

```
*
*   programm in vertical blank queue einbinden
*
*   LE 2/8/85
*

_v_bas_ad  equ   $44e    screen adresse
nvbls      equ   $454    anzahl vbl routinen
_vblqueue  equ   $456    zeiger auf nvbls routinen

gemdos     equ   1

super      equ   38      routine im supervisor modus ausfuehren
xbios      equ   14

*

        move.l   #start,-(sp)   startadresse der routine
        move     #super,-(sp)
        trap     #xbios         im supervisor modus ausfuehren

start   move     nvbls,d0       anzahl der vbl routinen
        lsl      #2,d0
        move.l   _vblqueue,a0   adresse der routinen
        clr      d1
loop1   tst.l    (a0,d1)        routine genutzt ?
        beq      free           nein
        addq     #4,d1
        cmp      d0,d1          alle eintrage getestet ?
        bne      loop1          nein
        rts                     alle routinen belegt, abbrechen

free    lea      (a0,d1),a2     vektor merken
        move     #(ende-init)/2-1,d0   laenge der routine in worten
        move.l   _v_bas_ad,a1   screen
        add.l    #32000,a1      plus laenge
        move.l   a1,a3          als neue adresse merken
        move.l   #init,a0       startadresse
movloop move     (a0)+,(a1)+    programm kopieren
```

```
        dbra    d0,movloop

        movem.l a2-a3,-(sp)
        jsr     (a3)            init-routine ausfuehren
        movem.l (sp)+,a2-a3

        add.l   #vblrout-init,a3
        move.l  a3,(a2)         vektor auf neue routine

        clr     -(sp)
        trap    #gemdos         zurueck zum desktop


*   diese routine wird ans speicherende kopiert

*
*   jede sekunde aktuelle uhrzeit einblenden
*   LE 18/10/85
*

anzahl  equ     50              bei jedem 50. vbl uhrzeit anzeigen

init    dc.w    linea           adresse der system fonts holen
        moveq   #2*4,d0         fontnummer 2, 8*16 system font
        lea     fontptr(pc),a3
        move.l  (a1,d0),(a3)    zeiger auf font merken

        lea     count(pc),a0
        move    #1,(a0)         zaehler initialisieren
        rts

vblrout lea     count(pc),a0
        subq    #1,(a0)         zaehler dekrementieren
        beq     dis_time        zeit anzeigen
        rts

gettime equ     $6a90           zeit von ikbd holen
time    equ     $a46            puffer fuer time im bcd-format
```

```
dis_time:
        move    #anzahl,(a0)    zaehler neu laden

        clr.l   a5              a5 muss null sein
        jsr     gettime         time holen

        moveq   #70,d6          cursor position in oberster zeile

        lea     time,a5
        move.b  (a5)+,d0        stunden holen
        bsr     wrtbcd          und ausgeben
        bsr     wrtcol          ':' ausgeben

        move.b  (a5)+,d0        minuten holen
        bsr     wrtbcd          und ausgeben
        bsr     wrtcol          ':' ausgeben

        move.b  (a5)+,d0        sekunden holen

wrtbcd  move    d0,-(sp)        wert merken
        lsr     #4,d0           oberes nibble
        bsr     wrtnib          ausgeben
        move    (sp)+,d0        wert zurueck holen
wrtnib  and     #%1111,d0       unteres nibble isolieren
        add.b   #'0',d0         in ascii-ziffer wandeln
        bra     wrtchar         und ausgeben

wrtcol  moveq   #$3a,d0         ':' ausgeben

*
*       variablen des ATARI ST  system font
*

fontdat equ     76      ptr auf font data
formwd  equ     80      abstand naechste rasterzeile im font
formhg  equ     82      anzahl rasterzeilen fuer ein zeichen

linea   equ     $a000
```

```
zeile   equ     80      bytes pro bildschirmzeile

*
*       zeichen in grafik-ram schreiben
*       d0 = character
*       d6 = cursorspalte in oberster zeile
*

wrtchar:
        moveq   #0,d1
        move    d6,d1
        addq    #1,d6           cursor auf naechste spalte setzen
        add.l   _v_bas_ad,d1    plus screen adresse
        move.l  d1,a4           als startadresse merken
        move.l  fontptr(pc),a3  zeiger auf font holen
        move.l  fontdat(a3),a0  zeiger auf fontdaten
        move    formwd(a3),d2   offset der naechsten rasterzeile im font
        move    formhg(a3),d7   form hight (anzahl scanlinien)
        subq    #1,d7

loop    move.b  (a0,d0),(a4)    rasterzeile auf bildschirm
        add     #zeile,a4       zeiger auf naechste bildschirmzeile
        add     d2,a0           zeiger auf naechste rasterzeile im font
        dbra    d7,loop
        rts

ende    equ   *

count   ds.w    1               speicher fuer vbl-zaehler
fontptr ds.l    1               adresse des font headers
```

```
100 open "R",1,"time.prg",16
110 field#1,16 as bin$
120 a$="": for i=1 to 16: read x$: if x$="*"then 150
130 a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140 lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150 data 60,1A,00,00,00,EE,00,00,00,00,00,00,00,00,00
160 data 00,00,00,00,00,00,00,00,00,00,00,00,2F,3C,00,00
170 data 00,0C,3F,3C,00,26,4E,4E,30,39,00,00,04,54,E5,48
180 data 20,79,00,00,04,56,42,41,4A,B0,10,00,67,08,58,41
190 data B2,40,66,F4,4E,75,45,F0,10,00,30,3C,00,42,22,79
200 data 00,00,04,4E,D3,FC,00,00,7D,00,26,49,20,7C,00,00
210 data 00,62,32,D8,51,C8,FF,FC,48,E7,00,30,4E,93,4C,DF
220 data 0C,00,D7,FC,00,00,00,16,24,8B,42,67,4E,41,A0,00
230 data 70,08,47,FA,00,82,26,B1,00,00,41,FA,00,78,30,BC
240 data 00,32,4E,75,41,FA,00,6E,53,50,67,02,4E,75,30,BC
250 data 00,32,9B,CD,4E,B9,00,00,6A,90,7C,46,4B,F9,00,00
260 data 0A,46,10,1D,61,0A,61,1A,10,1D,61,04,61,14,10,1D
270 data 3F,00,E8,48,61,02,30,1F,C0,7C,00,0F,D0,3C,00,30
280 data 60,02,70,3A,72,00,32,06,52,46,D2,B9,00,00,04,4E
290 data 28,41,26,7A,00,22,20,6B,00,4C,34,2B,00,50,3E,2B
300 data 00,52,53,47,18,B0,00,00,D8,FC,00,50,D0,C2,51,CF
310 data FF,F4,4E,75,00,00,00,00,00,00,00,00,00,02,40,00
320 data *
330 close 1:if s<> 18160 then print "Fehler in DATAs!!": end
340 print "Ok."
```

2.2 Druckertreiber für Epson-Drucker

Der ATARI ST hat den Vorteil, daß in seinem Zeichensatz sowohl der komplette ASCII-Zeichensatz enthalten ist, als auch fast alle Sonderzeichen und Umlaute für die meisten europäischen Sprachen. Wollen Sie diese Zeichen jedoch alle ausdrucken, so wird es mit den meisten Druckern Probleme geben.

Ein Ausweg wäre ein Drucker mit IBM-kompatiblem Zeichensatz, wie z.B. der neue Epson FX-85, der per DIP-Switch auf diesen Zeichensatz umgestellt werden kann. Haben Sie jedoch einen Epson FX-80 oder einen der vielen dazu kompatiblen Drucker, so können Sie bei diesen Geräten zwischen dem amerikanischen und dem deutschen Zeichensatz wählen.

Mit dem amerikanischen Zeichensatz können Sie die eckigen und geschweiften Klammern drucken, wie Sie auf dem ATARI ST über die ALT-Taste zu erreichen sind. Um jedoch die deutschen Umlaute wiedergeben zu können, muß der deutsche Zeichensatz gewählt werden. Erschwerend kommt noch hinzu, daß die ASCII-Codes der Umlaute auf dem ATARI ST und dem EPSON-Drucker nicht übereinstimmen.

Um dieses Problem zu lösen, haben wir für Sie ein kleines Treiberprogramm geschrieben, das jedem ATARI-Code eine entsprechende Code-Sequenz zuordnet, die das Zeichen auf dem Drucker erzeugt.

Damit dieses Programm mit allen Programmen zusammen läuft, die die Druckerausgabe benutzen, müssen natürlich die TRAP-Routinen geändert werden, die für die Druckerausgabe erforderlich ist. Die Vorgehensweise bei der Installierung dieses Programms ist ähnlich wie bei dem Druckerspooler. Zuerst wird der Speicherbedarf des Programms berechnet, damit nach Abschluß der Initialisierung der Platz für das Programm, das resident im Speicher bleiben muß, reserviert werden kann. Zuvor wird jedoch noch der TRAP#13-Vektor auf die neue Routine gesetzt und der alte Wert gespeichert.

Bei der eigentlichen Routine werden die Parameter auf dem Stack dahingehend abgeprüft, ob es sich um einen Aufruf zur Druckerausgabe handelt. Ist dies nicht der Fall, so wird auf die alte TRAP#13-Routine gesprungen. War jedoch der Drucker angesprochen, so wird das auszugebende Zeichen vom Stack geholt und mit einer Tabelle verglichen, die die zu konvertierenden Zeichen enthält. Wird das Zeichen in der Tabelle nicht gefunden, benutzen wir weiterhin die ursprüngliche Druckroutine.

War das Zeichen jedoch in der Tabelle enthalten, so geben wir eine Codesequenz an den Drucker aus, die wir aus einer korrespondierenden Tabelle holen. Wollen Sie die Tabelle erweitern oder verändern, z.B. an einen anderen Drucker anpassen, so beachten Sie, daß der letzte Eintrag in der ersten Tabelle 'chartab' dem ersten Eintrag in der zweiten Tabelle 'ch1' entspricht. Die zweite Tabelle ist so aufgebaut, daß zuerst die Anzahl der Zeichen minus eins enthalten ist und dann die Zeichen selber. Auf diese Weise kann jedem Zeichen aus dem ATARI-Zeichensatz eine beliebige Sequenz auf dem Drucker zugeordnet werden. Es wäre z.B. auch denkbar, ein Zeichen, das im Zeichensatz des Druckers nicht enthalten ist, durch Einzelnadelgrafik zu erzeugen. Hierbei sind Ihrer Phantasie keine Grenzen gesetzt. Dieses Programm können Sie auch mit dem Druckerspooler zusammen einsetzen.

```
*
*       drucker treiber fuer epson-drucker
*       LE 9/11/85
*

bios    equ     13
keep    equ     $31             programm resident halten

gemdos  equ     1
setexec equ     5               exception vektor setzen
conout  equ     3               zeichen ausgabe
prn     equ     0               device # des druckers

*                                       programm groesse berechnen

        move.l  4(sp),a0                base page address
        move.l  #$100,d6                groesse der base page
        add.l   12(a0),d6               plus text leange
        add.l   20(a0),d6               plus data laenge
        add.l   28(a0),d6               plus bss laenge

*                                       vektoren initialisieren

        move.l  #trap13,-(sp)           neuer vektor
        move    #45,-(sp)               vektor nummer
        move    #setexec,-(sp)
        trap    #bios                   vektor setzen
        addq.l  #8,sp
        move.l  d0,trapsve              alten vektor merken

        clr     -(sp)
        move.l  d6,-(sp)                anzahl bytes
        move    #keep,-(sp)             programm resident halten
        trap    #gemdos                 zurueck zum desktop

*                                       neue trap#13 routine

trap13  move.l  sp,a2                   ssp merken
        btst    #5,(sp)                 call from supervisor ?
```

```
        bne     super                   ja
        move.l  usp,a2                  sonst usp benutzen
        subq    #6,a2
super   cmp     #conout,6(a2)           conout-call ?
        bne     normal
        cmp     #prn,8(a2)              printer ?
        bne     normal

        move    10(a2),d0               character
        lea     chartab(pc),a0          zeiger auf sonderzeichentabelle
        moveq   #ch1-chartab-1,d1       laenge der tabelle
cmploop cmp.b   (a0)+,d0                zeichen in tabelle ?
        beq     printspez               ja, epson-code ausgeben
        dbra    d1,cmploop
        bra     normal                  nicht gefunden, so ausgeben

printspez:
        lsl     #2,d1
        lea     adtab(pc),a0            zeiger auf adress tabelle
        move.l  (a0,d1),a0              zeiger auf tabelle
        move.b  (a0)+,d0                laenge der tabelle minus 1
spzloop move.b  (a0)+,d1                zeichen aus tabelle
        movem.l a0/d0,-(sp)             register auf stack

        move    d1,-(sp)                zeichen
        move    #prn,-(sp)              drucker
        move    #conout,-(sp)           ausgabe
        pea     retadr(pc)              ruecksprung adresse
        move    sr,-(sp)                status fuer trap aufruf

normal  move.l  trapsve,a0
        jmp     (a0)                    zum alten trap#13

retadr  addq.l  #6,sp                   parameter vom stack
        movem.l (sp)+,a0/d0             register zurueck
        dbra    d0,spzloop              naechstes zeichen
        rte
```

```
chartab dc.b    '[\]{|}@~äöüßÄÖÜ§'      spezielle zeichen

ch1     dc.b    0,64                    §
ch2     dc.b    0,93                    Ü
ch3     dc.b    0,92                    Ö
ch4     dc.b    0,91                    Ä
ch5     dc.b    0,126                   ß
ch6     dc.b    0,125                   ü
ch7     dc.b    0,124                   ö
ch8     dc.b    0,123                   ä
ch9     dc.b    6,27,"R",0,126,27,"R",2 ~
ch10    dc.b    6,27,"R",0,64,27,"R",2  @
ch11    dc.b    6,27,"R",0,125,27,"R",2 }
ch12    dc.b    6,27,"R",0,124,27,"R",2 |
ch13    dc.b    6,27,"R",0,123,27,"R",2 {
ch14    dc.b    6,27,"R",0,93,27,"R",2  ]
ch15    dc.b    6,27,"R",0,92,27,"R",2  \
ch16    dc.b    6,27,"R",0,91,27,"R",2  [

        .even

adtab   dc.l    ch1,ch2,ch3,ch4,ch5,ch6
        dc.l    ch7,ch8,ch9,ch10,ch11,ch12
        dc.l    ch13,ch14,ch15,ch16

        .bss
trapsve ds.l    1                       alter trap#13 vektor
```

```
100 open "R",1,"epson.prg",16
110 field#1,16 as bin$
120 a$="": for i=1 to 16: read x$: if x$="*"then 150
130 a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140 lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150 data 60,1A,00,00,01,3E,00,00,00,00,00,00,04,00,00
160 data 00,00,00,00,00,00,00,00,00,00,00,20,6F,00,04
170 data 2C,3C,00,00,01,00,DC,A8,00,0C,DC,A8,00,14,DC,A8
180 data 00,1C,2F,3C,00,00,00,38,3F,3C,00,2D,3F,3C,00,05
190 data 4E,4D,50,8F,23,C0,00,00,01,3E,42,67,2F,06,3F,3C
200 data 00,31,4E,41,24,4F,08,17,00,05,66,04,4E,6A,5D,4A
210 data 0C,6A,00,03,00,06,66,3E,0C,6A,00,00,00,08,66,36
220 data 30,2A,00,0A,41,FA,00,44,72,0F,B0,18,67,06,51,C9
230 data FF,FA,60,22,E5,49,41,FA,00,92,20,70,10,00,10,18
240 data 12,18,48,E7,80,80,3F,01,3F,3C,00,00,3F,3C,00,03
250 data 48,7A,00,0C,40,E7,20,79,00,00,01,3E,4E,D0,5C,8F
260 data 4C,DF,01,01,51,C8,FF,DA,4E,73,5B,5C,5D,7B,7C,7D
270 data 40,7E,84,94,81,9E,8E,99,9A,DD,00,40,00,5D,00,5C
280 data 00,5B,00,7E,00,7D,00,7C,00,7B,06,1B,52,00,7E,1B
290 data 52,02,06,1B,52,00,40,1B,52,02,06,1B,52,00,7D,1B
300 data 52,02,06,1B,52,00,7C,1B,52,02,06,1B,52,00,7B,1B
310 data 52,02,06,1B,52,00,5D,1B,52,02,06,1B,52,00,5C,1B
320 data 52,02,06,1B,52,00,5B,1B,52,02,00,00,00,AE,00,00
330 data 00,B0,00,00,00,B2,00,00,00,B4,00,00,00,B6,00,00
340 data 00,B8,00,00,00,BA,00,00,00,BC,00,00,00,BE,00,00
350 data 00,C6,00,00,00,CE,00,00,00,D6,00,00,00,DE,00,00
360 data 00,E6,00,00,00,EE,00,00,00,F6,00,00,00,18,12,62
370 data 72,04,04,04,04,04,04,04,04,04,04,04,04,04,04,04
380 data *
390 close 1:if s<> 19776 then print "Fehler in DATAs!!": end
400 print "Ok."
```

2.3 RAM-Disk für ATARI ST

Wenn Sie schon einmal mit dem 68000-Assembler oder mit dem C-Compiler auf dem ATARI ST gearbeitet haben, dann wissen Sie auch, daß vom Quellprogramm bis zum lauffähigen PRG-File eine Reihe von Schritten erforderlich sind, bei der eine Vielzahl von Zwischendateien erzeugt und wieder gelöscht werden. Compilieren, Assemblieren, Linken und Relozieren sind also mit regen Diskettenaktivitäten verbunden, und wer schon jemals ein Programm selbst geschrieben hat weiß, daß es mit einem Durchlauf nicht getan ist: Also Editor mit Quellprogramm wieder laden, Fehler korrigieren, Quellprogramm wieder abspeichern und das Ganze nochmal. Bei längeren Programmen kann sich diese Prozedur leicht über eine viertel Stunde hinziehen. Wenn man einmal untersucht, was die langen Zeiten bedingt, so findet man schnell heraus, daß das Abspeichern und Laden von Diskette die meiste Zeit benötigt, besonders wenn man nur mit einem Laufwerk arbeitet, so daß der Schreib-Lese-Kopf ständig auf Wanderschaft ist. Aber wie kann man diesem Verfahren auf die Sprünge helfen?

Eine Alternative wäre der Anschluß einer Harddisk, die einen ca. 30mal schnelleren Zugriff auf die Daten ermöglicht. Doch es gibt noch eine andere Möglichkeit, die noch schneller ist und zudem keinen Pfennig kostet: Eine RAM-Disk!

Was ist nun darunter schon wieder zu verstehen? Ganz einfach - eine RAM-Disk ist eine Diskettenstation, die im RAM-Speicher des Rechners steht. Man geht einfach hin und zweigt einen Teil des RAMs des Rechners ab und behandelt diesen so wie ein Peripheriegerät, eben wie eine Diskettenstation. Sollen Daten vom Rechner auf diese RAM-Disk geschrieben werden, so werden Sie nicht dem Diskcontroller übergeben, sondern einfach in den reservierten RAM-Bereich transferiert, was mit der vollen internen Geschwindigkeit des 68000-Prozessors geschieht. Weiterhin entfallen alle Such- und Positionierzeiten, die bei einem Diskettenlaufwerk zu Verzögerungen führen.

Lesen und Schreiben von und auf RAM-Disk reduziert sich also auf ein Kopieren von Daten von einem RAM-Bereich (der RAM-Disk) in einen anderen (den Diskettenpuffer). Ein Programm, das eine RAM-Disk installiert, muß also hauptsächlich diese Datenübertragung durchführen. Neben diesem verschwindend kleinen Teil kommt noch die Einbindung ins Betriebssystem. Doch da hat ATARI schon vorgesorgt. Ganze drei Vektoren müssen 'gepatched' werden. Diese Vektoren sind eigentlich zum Einbinden einer Harddisk gedacht, doch für unsere Zwecke eignen sie sich ebenso gut. Die angesprochenen drei Vektoren betreffen die BIOS-Aufrufe für Sektoren lesen/schreiben, BIOS Parameter Block holen (gibt Auskunft über Größe und Organisation einer Diskette) sowie die Funktion Media Change, die dem DOS mitteilt, wenn eine Diskette gewechselt wurde (geht bei einer RAM-Disk wohl schlecht - oder?). Unsere RAM-Disk soll die Laufwerksbezeichnung C haben, im BIOS wird der numerische Wert 2 benutzt (0= Laufwerk A, 1= Laufwerk B). Im Programm werden also die Vektoren auf unsere Routinen gesetzt, in denen dann getestet wird, ob das Laufwerk C gemeint ist. Dies kann durch Prüfen der Laufwerksnummer, die als Parameter auf dem Stack übergeben wird, geschehen. Sind wir nicht gemeint, wird einfach auf den ursprünglichen Vektor gesprungen, der die Laufwerke A und B abhandelt. Um das Programm für den Benutzer komfortabel und flexibel zu gestalten, soll dem Programm die Größe der RAM-Disk als Parameter übergeben werden. Dazu gibt man ähnlich wie bei dem Druckerspooler (die beiden Programme laufen auch zusammen) die Größe in KByte an. Hierbei ist man nicht an die vorgegeben Diskettengröße (180, 360 und 720K) gebunden, sondern kann je nach Bedarf und zur Verfügung stehendem Speicher Werte von 80 KByte (oder noch weniger, wenn das sinnvoll erscheint) bis zu ca. 640 KByte auf einer 1-MByte-Maschine angeben. Auf einer 512-K-Maschine sind Werte von ca. 100 bis 120 K möglich (das ist natürlich abhängig davon, wie groß der Speicherbedarf Ihrer Anwenderprogramme noch ist). Auch bei diesem Programm braucht man keinen Parameter einzugeben, es wird dann der Standardwert von 100 KByte genommen. Diesen Wert können Sie natürlich im Source-Listing an Ihre

Bedürfnisse anpassen. Benutzen Sie den BASIC-Lader, so müssen Sie die beiden unterstrichenen Bytes mit dem neuen Wert versorgen (Format Hi, Lo-Byte; für 300 K z.B. 01,2C). Natürlich dürfen Sie nicht vergessen, die Prüfsumme entsprechend zu ändern. Das Programm konfiguriert aus der Diskettenkapazität automatisch den BIOS Parameter Block, erzeugt einen Bootsektor in der RAM-Disk und löscht das Directory. Ehe das Programm beendet wird, wird ähnlich wie bei dem Drucker-Spooler der Speicherbedarf berechnet und entsprechend Speicher reserviert.

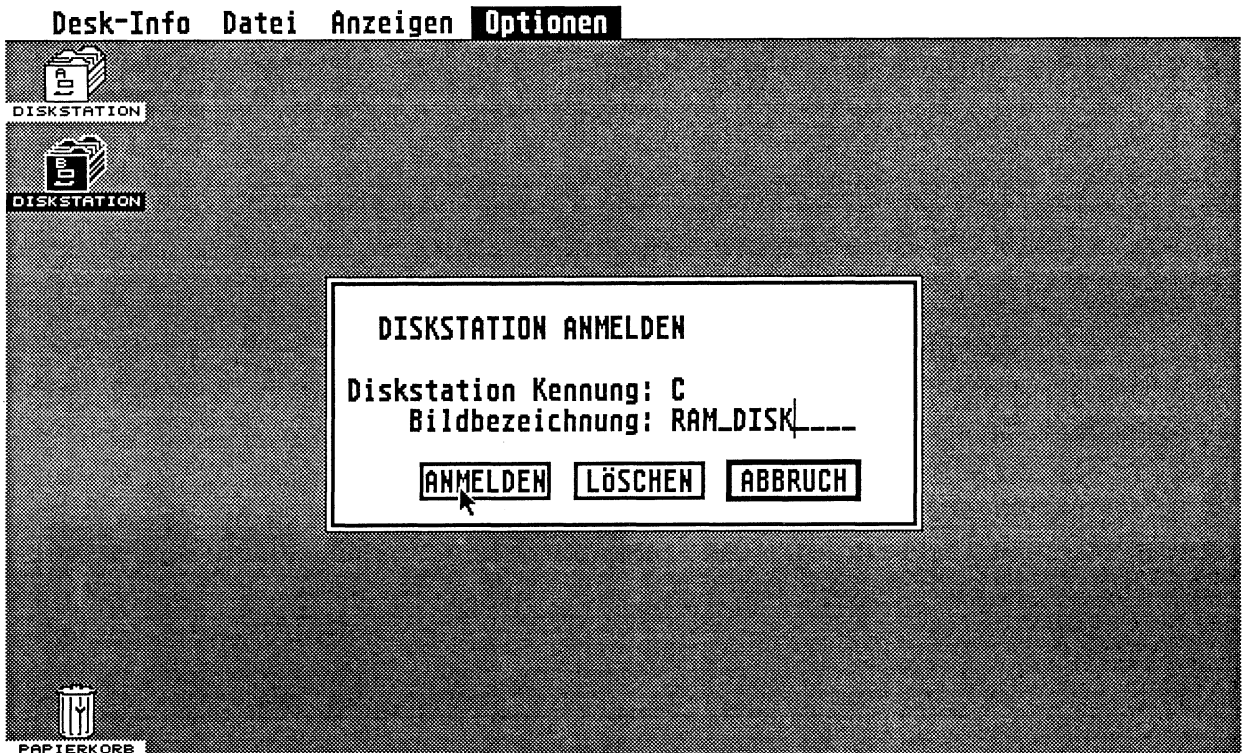
Wie installieren Sie nun Ihre RAM-Disk?

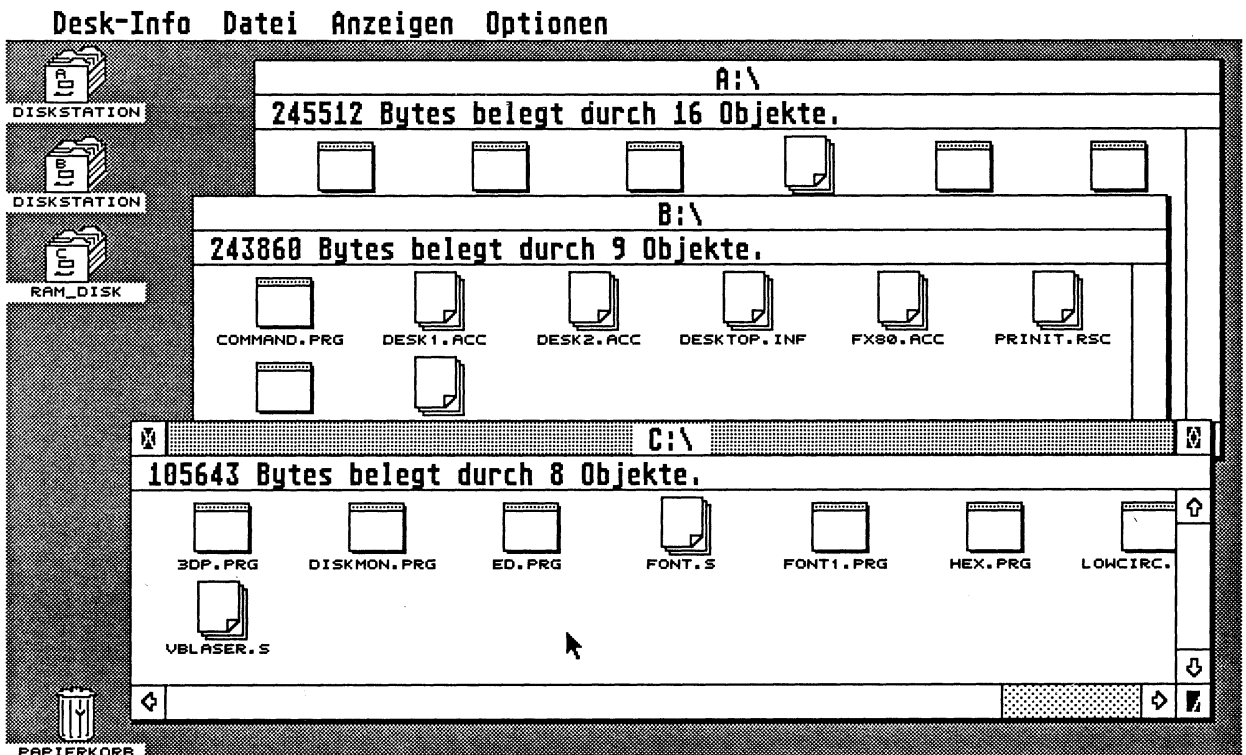
Führen Sie dazu das Programm aus (mit oder ohne Parametereingabe). Klicken Sie nun ein Disketten-Icon an (Laufwerk A oder B) und wählen aus dem Desktop unter 'Optionen' den Menuepunkt 'Diskettenstation anmelden'. Geben Sie jetzt für Diskettenstation Kennung 'C' ein und für Bildbezeichnung 'RAM_DISK' und klicken Sie dann auf 'ANMELDEN'. Auf dem Desktop erscheint jetzt ein weiteres Diskettensymbol, das die Bezeichnung 'C' und 'RAM_DISK' trägt. Jetzt können Sie durch Doppelklick auf dieses Symbol die RAM-Disk öffnen. Es erscheint ein Fenster mit dem Namen C in dem 0 Objekte mit 0 Bytes stehen. Jetzt können Sie von Laufwerk A oder B Programme oder Dateien auf die RAM-Disk kopieren. Dies geschieht ganz genauso, als handelte es sich um eine richtige Diskettenstation. Versuchen Sie doch jetzt einmal, ein Programm von der RAM-Disk zu laden. Ganz schön flott! Programme von 100 K brauchen noch nicht mal eine Sekunde. Wozu läßt sich die RAM-Disk am sinnvollsten einsetzen? Wenn Sie viel Programme oder Texte schreiben und Programme assemblieren, so sollten Sie Editor, Quellprogramm und - falls genügend Platz auf der RAM-Disk ist - auch Compiler, Assembler und Linker unterbringen, das beschleunigt Ihre Arbeit wesentlich. Ein kompletter Assemblerlauf, der auf normalen Disketten vielleicht 10 Minuten dauert, braucht mit RAM-Disk oft nicht mal eine Minute; eine Zeiterparnis, die die Entwicklungsphase eines Programms deutlich verkürzen kann. Doch hier noch eine Warnung: Denken Sie immer daran,

daß die Daten auf der RAM-Disk eben nur im RAM stehen und mit dem Ausschalten des Rechners unwiederbringlich verloren sind! Kopieren Sie also die Ergebnisse Ihrer Arbeit vor dem Ausschalten von der RAM-Disk auf eine richtige Diskette! Das gleiche sollten Sie machen, ehe Sie 'absturzgefährdete' Programme starten.

Noch ein paar Hinweise zum Umgang mit der RAM-Disk. Ein Backup von einem Laufwerk auf die RAM-Disk oder umgekehrt ist nicht möglich. Öffnen Sie statt dessen ein Fenster mit dem Laufwerk, auf das Sie kopieren wollen, und ziehen Sie das Icon der Diskette, von der kopiert werden soll, in das geöffnete Fenster. Auch sollten Sie nicht versuchen, die RAM-Disk zu formatieren, die Disketten in Laufwerk A und B könnten Schaden nehmen. Umrahmen Sie statt dessen die Icons sämtlicher Dateien auf der RAM-Disk und werfen Sie sie in den Papierkorb; auf der RAM-Disk geschieht dies blitzschnell.

Das Installieren der RAM-Disk nach dem Einschalten des Rechners können Sie auch automatisch durch den Rechner erledigen lassen. Legen Sie dazu auf Ihrer Systemdiskette einen Ordner mit dem Namen 'AUTO' an und kopieren Sie in diesen das Installationsprogramm 'RAMDISK.PRG'. Wenn Sie noch vom Desktop unter 'Optionen' den Menupunkt 'Arbeit sichern' wählen, so wird Ihre Konfiguration mit der installierten RAM-Disk in der Datei 'DESKTOP.INF' abgespeichert. Bei jedem Booten wird nun die RAM-Disk automatisch installiert und direkt ein Fenster für Laufwerk C geöffnet.


Desk-Info Datei Anzeigen Optionen
DISKSTATION
DISKSTATION
DISKSTATION ANMELDEN
Diskstation Kennung: C
Bildbezeichnung: RAM_DISK____
ANMELDEN
LÖSCHEN
ABBRUCH
PAPIERKORB


Desk-Info Datei Anzeigen Optionen
DISKSTATION
DISKSTATION
RAM_DISK
PAPIERKORB
A:\
245512 Bytes belegt durch 16 Objekte.
B:\
243860 Bytes belegt durch 9 Objekte.
COMMAND.PRG
DESK1.ACC
DESK2.ACC
DESKTOP.INF
FX80.ACC
PRINIT.RSC
C:\
105643 Bytes belegt durch 8 Objekte.
3DP.PRG
DISKMON.PRG
ED.PRG
FONT.S
FONT1.PRG
HEX.PRG
LOWCIRC.
VBLASER.S

```
*
*       RAM disk fuer ATARI ST
*       LE/RB, 6/11/85
*

hdv_bpb equ     $472            bios parameter block
hdv_rw  equ     $476            sectoren lesen/schreiben
hdv_mediach     equ     $47e

drvbits equ     $4c2            bit vektor aktive drives

gemdos  equ     1
keep    equ     $31

xbios   equ     14
super   equ     38

default equ     100             standard kapazitaet in kb

init    move.l  4(sp),a0        base page address
        move.l  #$100,d6        groesse der base page
        add.l   12(a0),d6       text laenge
        add.l   20(a0),d6       data laenge
        add.l   28(a0),d6       bss laenge

        moveq   #0,d7
        moveq   #0,d0
        lea     129(a0),a0      zeiger auf kommandozeile
nextchr move.b  (a0)+,d0        erstes zeichen aus kommandozeile
        sub.b   #'0',d0
        bmi     exit
        cmp.b   #9,d0           ziffer ?
        bgt     exit
        mulu    #10,d7
        add     d0,d7           naechstes digit
        bra     nextchr
```

```
exit    tst     d7              eingabe dagewesen ?
        bne     ok
        move    #default,d7     default wert

ok      moveq   #0,d1
        move    d7,d1           kapazitaet in k
        add     #9,d1           plus 9 k
        lsl.l   #8,d1
        lsl.l   #2,d1           * 1024
        add.l   d1,d6           zum speicherbedarf addieren


        move.l  #init1,-(sp)
        move    #super,-(sp)    initialisierung im supervisor
        trap    #xbios
        addq.l  #6,sp

        clr     -(sp)
        move.l  d6,-(sp)        anzahl bytes
        move    #keep,-(sp)     programm resident lassen
        trap    #gemdos         zurueck zum desktop


init1   move.l  hdv_bpb,bpbsave
        move.l  #bpb,hdv_bpb

        move.l  hdv_rw,rwsave   vektoren auf neue routinen setzen
        move.l  #rw,hdv_rw

        move.l  hdv_mediach,mediasave
        move.l  #media,hdv_mediach


install moveq   #0,d1
        lea     ramdisk,a0
        move    #2*9*512/4-1,d0 track 0 und 1
iloop1  move.l  d1,(a0)+        der ram disk loeschen
        dbra    d0,iloop1
```

```
*                                   boot sector generieren
        lea     ramdisk+11,a0
        lea     boottab,a1
        moveq   #tabend-boottab-1,d0
bloop   move.b  (a1)+,(a0)+         daten in boot sector kopieren
        dbra    d0,bloop

        move    d7,numcl            kapazitaet in kb in bpb

        lsl     #1,d7               kapazitaet in sectoren
        add     #18,d7              plus 18 sectoren
        lea     ramdisk+19,a0
        move.b  d7,(a0)+            lo byte
        lsl     #8,d7
        move.b  d7,(a0)             hi byte

        or.l    #%100,drvbits       drive c anmelden
        rts

bpb:    cmp     #2,4(sp)            drive c ?
        beq     bpb1                ja

        move.l  bpbsave,a0          alte routine
        jmp     (a0)

bpb1    move.l  #bpbtab,d0          zeiger auf bios parameter block
        rts

rw      cmp     #2,14(sp)           drive c ?
        beq     rw1                 ja

        move.l  rwsave,a0           alte routine
        jmp     (a0)

rw1     move    12(sp),d0           recno, logische sector 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 sectoren
        subq    #1,d1
        lea     ramdisk,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 vertauschen

rloop0  move    #511,d0         ein sector
rloop   move.b  (a1)+,(a0)+     puffer kopieren
        dbra    d0,rloop
        dbra    d1,rloop0       naechster sector
        moveq   #0,d0           ok
        rts


media   cmp     #2,4(sp)        drive c ?
        beq     media1          ja

        move.l  mediasave,a0    alte routine
        jmp     (a0)

media1  moveq   #0,d0           diskette nicht gewechselt
        rts

        .data
bpbtab:
recsiz: dc.w    $200            sectorgroesse
clsiz   dc.w    2               cluster groesse in sectoren
clsizb  dc.w    $400            cluster groesse in bytes
rdlen   dc.w    7               directory laenge in sectoren
fsiz    dc.w    5               fat groesse
fatrec  dc.w    6               fat sectoren
datrec  dc.w    18              sectoren fuer verwaltung
numcl   ds.w    1               kapazitaet in kb
```

```
flags   ds.w    8

boottab:                        * daten in 8086-format
        dc.b    0,2             bytes per sector
        dc.b    2               sectors per cluster
        dc.b    1,0             reserved sectors
        dc.b    2               fats
        dc.b    112,0           directory entries
        ds.b    2               sectors on media
        dc.b    0               media descriptor
        dc.b    5,0             sectors per fat
        dc.b    9,0             sectors per track
        dc.b    1,0             sides
        dc.b    0               hidden
tabend  equ     *

        .bss

bpbsave ds.l    1               platz fuer alte floppy vektoren
rwsave  ds.l    1
mediasave       ds.l    1

ramdisk equ     *               hier beginnt die ram disk
```

```
100 open "R",1,"ramdisk.prg",16
110 field#1,16 as bin$
120 a$="": for i=1 to 16: read x$: if x$="*"then 150
130 a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140 lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150 data 60,1A,00,00,01,5E,00,00,00,32,00,00,00,0C,00,00
160 data 00,00,00,00,00,00,00,00,00,00,00,20,6F,00,04
170 data 2C,3C,00,00,01,00,DC,A8,00,0C,DC,A8,00,14,DC,A8
180 data 00,1C,7E,00,70,00,41,E8,00,81,10,18,90,3C,00,30
190 data 6B,0E,B0,3C,00,09,6E,08,CE,FC,00,0A,DE,40,60,EA
200 data 4A,47,66,04,3E,3C,00,64,72,00,32,07,D2,7C,00,09
210 data E1,89,E5,89,DC,81,2F,3C,00,00,00,62,3F,3C,00,26
220 data 4E,4E,5C,8F,42,67,2F,06,3F,3C,00,31,4E,41,23,F9
230 data 00,00,04,72,00,00,01,90,23,FC,00,00,00,E8,00,00
240 data 04,72,23,F9,00,00,04,76,00,00,01,94,23,FC,00,00
250 data 01,00,00,00,04,76,23,F9,00,00,04,7E,00,00,01,98
260 data 23,FC,00,00,01,4A,00,00,04,7E,72,00,41,F9,00,00
270 data 01,9C,30,3C,08,FF,20,C1,51,C8,FF,FC,41,F9,00,00
280 data 01,A7,43,F9,00,00,01,7E,70,11,10,D9,51,C8,FF,FC
290 data 33,C7,00,00,01,6C,E3,4F,DE,7C,00,12,41,F9,00,00
300 data 01,AF,10,C7,E1,4F,10,87,00,B9,00,00,00,04,00,00
310 data 04,C2,4E,75,0C,6F,00,02,00,04,67,08,20,79,00,00
320 data 01,90,4E,D0,20,3C,00,00,01,5E,4E,75,0C,6F,00,02
330 data 00,0E,67,08,20,79,00,00,01,94,4E,D0,30,2F,00,0C
340 data 48,C0,E1,88,E3,88,20,6F,00,06,32,2F,00,0A,53,41
350 data 43,F9,00,00,01,9C,D3,C0,30,2F,00,04,08,00,00,00
360 data 67,02,C1,49,30,3C,01,FF,10,D9,51,C8,FF,FC,51,C9
370 data FF,F4,70,00,4E,75,0C,6F,00,02,00,04,67,08,20,79
380 data 00,00,01,98,4E,D0,70,00,4E,75,02,00,00,02,04,00
390 data 00,07,00,05,00,06,00,12,00,00,00,00,00,00,00,00
400 data 00,00,00,00,00,00,00,00,00,00,00,02,02,01,00,02
410 data 70,00,00,00,00,05,00,09,00,01,00,00,00,00,00,4C
420 data 1C,06,0E,06,0E,06,0C,10,06,0E,0C,20,08,10,1C,2E
430 data *
440 close 1:if s<> 26687 then print "Fehler in DATAs!!": end
450 print "Ok."
```

2.4 Druckerspooler für den ATARI ST

In diesem Abschnitt wollen wir Ihnen ein nützliches Werkzeug vorstellen, das Ihnen viel Zeit sparen kann. Wenn Sie schon einmal vor Ihrem Rechner gesessen und gewartet haben, bis ein zehnseitiges Listing endlich ausgedruckt war und Sie mit Ihrem Rechner weiter arbeiten können, dann wissen Sie, was wir meinen.

Die Mechanik eines Druckers kommt mit der Geschwindigkeit, mit der die Datem von Computer geliefert werden, nicht mit und der Rechner muß zwangsläufig warten. Deshalb ist in manchen Druckern ein Pufferspeicher eingebaut, in den die angekommenden Daten erst einmal mit der für Rechner üblichen Geschwindigkeit geschrieben werden. Aus diesem Puffer werden die Daten dann so, wie es die Druckgeschwindigkeit des Druckers erlaubt, herausgeholt und gedruckt.

Eine gängige Größe für solch einen Puffer sind 2 KByte; darin läßt sich etwa eine DIN-A4-Seite Text unterbringen. Ist der auszudruckende Text größer, so wird der Puffer bei der Übertragung schnell voll und der Rechner muß wieder warten. Eine Alternative wäre ein Drucker mit einem größerem Puffer; doch sind solche Geräte unverhältnismäßig teurer. Aber wir haben ja einen Rechner mit 512 KByte (oder sogar 1 MByte)! Verlegen wir doch einfach den Pufferspeicher in den Rechner!

Um die Arbeitsweise des nachfolgenden Programms zu verstehen, müssen wir kurz darauf eingehen, wie die Datenübertragung vom Rechner zum Drucker funktioniert. Die Daten werden parallel über eine sogenannte Centronics-Schnittstelle übertragen, bei der über acht Leitungen jeweils ein Byte gleich 8 Bit gleichzeitig übertragen werden. Damit sich Rechner und Drucker über den Zeitpunkt der Übertragung einigen können, werden noch zwei sogenannte Handshake-Leitungen benutzt. Ist der Drucker bereit, ein Datenbyte anzunehmen, so signalisiert er dies über einen Low-Pegel auf der Busy-Leitung. Jetzt schickt der Rechner

ein Datenbyte, was er dem Drucker über einen kurzen Low-Impuls über die Strobe-Leitung mitteilt.

Wenn wir jetzt im Rechner einen Zwischenpuffer für die Daten, die zum Drucker gehen, einrichten wollen, so brauchen wir zwei Routinen im Rechner. Die erste, die anstelle der bisherigen Druckerausgabe tritt, schreibt das Zeichen in den Puffer. Die zweite Routine muß dafür sorgen, daß immer dann, wenn der Drucker das nächste Zeichen annehmen kann, ein Zeichen aus dem Puffer geschickt wird.

Das Programm ist so ausgelegt, daß es Puffer bis zu einer Größe von 63 KByte verwalten kann. Wenn Sie das Programm durch Anklicken mit der Maus starten, so werden 32 KByte als Druckerpuffer reserviert. Darin lassen sich ca. 15 Textseiten zwischenspeichern. Installieren Sie das Programm als 'TOS übernimmt Parameter', so können Sie beim Starten die Puffergröße angeben. Ein Wert zwischen 1 und 63 bestimmt dabei die Größe in KByte. Das Programm reserviert sich automatisch den entsprechenden Speicherbereich vom Betriebssystem. Dieser Speicher steht anderen Programmen natürlich nicht mehr zur Verfügung. Falls der übrig bleibende Speicherplatz für bestimmte Anwenderprogramme nicht mehr ausreicht (wohl höchstens auf der 512-K-Maschine), müssen Sie entsprechend weniger Platz für den Puffer vorsehen.

Wenn Sie den Spooler ständig benutzen und nicht nach jedem Booten neu installieren wollen, so bietet das Betriebssystem dazu eine komfortable Möglichkeit, das Programm automatisch zu initialisieren. Legen Sie dazu auf der Betriebssystemdiskette einen Ordner mit der Bezeichnung 'AUTO' an und kopieren Sie das Programm in diesen Ordner. Nach dem Booten werden sämtliche Programme in diesem Ordner ausgeführt. Doch jetzt zu einer kurzen Beschreibung des Programms.

Damit das Programm den Platz für sich selbst sowie für den Puffer reservieren kann, wird zuerst die Größe des Programms berechnet. Diese Daten kann man aus der sogenannten Base Page holen, die $100 Bytes groß ist und unmittelbar unterhalb des Programms liegt. Die Adresse der Base Page wird vom Stack geholt. Dann werden die Längen von Text-, Data- und Block Storage-Segment zur Länge der Base Page addiert. Ebenfalls in der Base Page finden wir die Kommandozeile, das ist der Text, den wir als Parameter dem Programm mitgegeben haben, also unsere Puffergröße in KByte. Wir lesen die Ziffern und wandeln sie in eine Binärzahl um. Wurde nichts eingegeben, so nehmen wir den Default-Wert von 32 KByte. Durch Linksschieben um 10 Bit erhalten wir daraus den Wert in Byte, den wir als Puffergröße in einen Pufferdatensatz einbauen, der ähnlich wie die 'iorec' des Betriebssystems aufgebaut ist (siehe 'ATARI ST intern'). Als nächstes wird der TRAP#13-Vektor 'umgebogen'. Wir setzen ihn auf eine eigene Routine, in der geprüft wird, ob es sich um Druckerausgabe oder um den Druckerstatus handelt. Dies geschieht durch Testen der Parameter auf dem Stack. Falls eine andere Routine angesprochen wurde, springen wir auf die ursprüngliche TRAP#13-Routine.

Bei der Ausgabe eines Zeichen auf den Drucker müssen wir zwischen mehreren Situationen unterscheiden. In dem Fall, daß der Puffer noch leer ist, versuchen wir das Zeichen direkt an den Drucker auszugeben. Ist der Drucker jedoch nicht bereit, ein Zeichen anzunehmen, schreiben wir das Zeichen in den Puffer, was auch dann geschieht, wenn der Puffer schon Zeichen enthält. Sollte der Puffer voll sein, so warten wir maximal 30 Sekunden auf Platz im Puffer. Ist die Zeit abgelaufen, so melden wir der aufrufenden Routine, daß das Zeichen nicht ausgeben werden konnte. Dies kann jedoch nur dann eintreten, wenn der Puffer voll ist (32 KByte Daten) und der Drucker keine weiteren Zeichen mehr annimmt. Doch wie kommen die Zeichen aus dem Puffer an den Drucker?

Dazu nutzen wir die Interruptmöglichkeit der Busy-Leitung des Druckers aus. Dieser Interrupt wird in der Initialisierungsroutine freigegeben und auf unsere Routine 'busyint' gelegt. Jedesmal, wenn der Drucker bereit ist, das nächste Zeichen zu Empfangen, löst er im Rechner einen Interrupt aus. In der Interruptroutine wird nun geprüft, ob noch Zeichen im Puffer stehen. In diesem Fall wird ein Zeichen aus dem Puffer geholt und an den Drucker geschickt. Damit ist die Interruptroutine auch schon zu Ende und es kann in das unterbrochene Programm zurückgekehrt werden. Der Vorteil dieses Interrupt-Verfahren ist der, daß der Rechner zu keiner Zeit auf den Drucker warten muß (es sei denn, der Puffer ist voll, doch dann sind ja schon 15 Seiten im Puffer).

Wenn Sie mit installiertem Druckerspooler mal einen Text von 10 Seiten an den Drucker schicken, so meldet sich der Rechner schon nach wenigen Sekunden zurück, während der Drucker noch mehrere Minuten zu tun hat.

Auf den nächsten Seiten finden Sie das Assemblerlisting des Programms sowie eine Ladeprogramm in BASIC, das ein Programm auf Diskette erzeugt.

```
*
*       drucker spooler fuer atari st
*
*       LE/RB, 5/11/85
*

bios    equ     13
keep    equ     $31             programm resident halten

gemdos  equ     1
setexec equ     5               exception vektor setzen
conout  equ     3               zeichen ausgabe
constat equ     8               ausgabe status
prn     equ     0               device # des druckers
savptr  equ     $4a2            save area fuer register
hz 200  equ     $4ba            200 hz system takt

xbios   equ     14
mfpint  equ     13                      mfp interrupt installieren

mfp     equ     $fffa01                 mfp 68901
psg     equ     $ff8800                 psg ym 2149
isrb    equ     $10                     interrupt service register b

default equ     32                      standard puffer groesse in kb
timeout equ     30                      30 sekunden timeout

*                                       programm groesse berechnen

        move.l  4(sp),a0                base page address
        move.l  #$100,d6                groesse der base page
        add.l   12(a0),d6               plus text leange
        add.l   20(a0),d6               plus data laenge
        add.l   28(a0),d6               plus bss laenge

*                                       puffer groesse aus kommandozeile
        moveq   #0,d7
        moveq   #0,d0
        lea     129(a0),a0              zeiger auf kommandozeile
```

```
nextchr move.b  (a0)+,d0          zeichen holen
        sub.b   #'0',d0
        bmi     exit              keine ziffer ?
        cmp.b   #9,d0
        bgt     exit              keine ziffer ?
        mulu    #10,d7            naechste stelle
        add     d0,d7
        bra     nextchr

exit    tst     d7                zahl eingegeben ?
        bne     ok                ja
        move    #default,d7       sonst default wert nehmen
ok      ext.l   d7
        moveq   #10,d0
        lsl.l   d0,d7             wert in byte umrechnen
        add.l   d7,d6             zu platzbedarf addieren
        move    d7,laenge         und in iorec eintragen

*                                 vektoren initialisieren

        move.l  #trap13,-(sp)     neuer vektor
        move    #45,-(sp)         vektor nummer
        move    #setexec,-(sp)
        trap    #bios             vektor setzen
        addq.l  #8,sp
        move.l  d0,trapsve        alten vektor merken

        move.l  #busyint,-(sp)
        move    #0,-(sp)          int nummer
        move    #mfpint,-(sp)
        trap    #xbios            centronics interrupt enablen
        addq.l  #8,sp

        clr     -(sp)
        move.l  d6,-(sp)          anzahl bytes
        move    #keep,-(sp)       programm resident halten
        trap    #gemdos           zurueck zum desktop
```

```
*                                       neue trap#13 routine
trap13  move.l  sp,a2                   ssp merken
        btst    #5,(sp)                 call from supervisor ?
        bne     super                   ja
        move.l  usp,a2                  sonst usp benutzen
        subq    #6,a2
super   cmp     #conout,6(a2)           conout-call ?
        bne     normal
        cmp     #prn,8(a2)              printer ?
        bne     normal

        move.l  savptr,a1               zeiger auf save area
        move    (sp)+,-(a1)             status retten
        move.l  (sp)+,-(a1)             return adresse
        move.l  a1,savptr               save ptr updaten

        move    10(a2),d1               character
        bsr     print

        move.l  savptr,a1
        move.l  (a1)+,-(sp)             return adresse
        move    (a1)+,-(sp)             status
        move.l  a1,savptr
        rte

normal:
        cmp     #constat,6(a2)          drucker status ?
        bne     norm1
        cmp     #prn,8(a2)
        bne     norm1                   ueber alten trap#13 vektor

        moveq   #-1,d0                  status ok annehmen
        bsr     getptr                  zeiger holen
        move    tail(a0),d2
        bsr     wrap
        cmp     head(a0),d2             platz im puffer ?
        bne     platz                   ja
        moveq   #0,d0                   busy, kein platz
platz   rte
```

```
norm1   move.l  trapsve,a0          zum alten trap 13
        jmp     (a0)

print   move    #$2700,sr           interrupt sperren
        bsr     getptr              zeiger auf iorec und mfp
        move    head(a0),d2
        cmp     tail(a0),d2         puffer leer ?
        bne     inbuff              nein, zeichen in puffer

loop    btst    #0,(a1)             drucker busy ?
        bne     inbuff              ja, in puffer

notbusy lea     psg,a2              adresse des psg
        move.b  #15,(a2)            register nummer port b
        move.b  d1,2(a2)            datenbyte ausgeben

        move.b  #14,(a2)            register nummer port a
        move.b  (a2),d0
        and.b   #$df,d0             strobe low
        move.b  d0,2(a2)

        or.b    #$20,d0             strobe high
        move.b  d0,2(a2)

        moveq   #-1,d0              ok
        rts


inbuff  move    tail(a0),d2         schreib zeiger
        bsr     wrap                erhoehen
        cmp     head(a0),d2         puffer voll ?
        beq     buffull             ja
inbuff1 move.l  (a0),a1             puffer adresse
        move.b  d1,(a1,d2)          zeichen in puffer schreiben
        move    d2,tail(a0)         neuen tail index merken
        moveq   #-1,d0              zeichen losgeworden
        rts
```

```
buffull move.l  hz 200,d0
        add.l   #timeout*200,d0         anzahl sekunden warten
        move    #$2300,sr               interrupts freigeben
wait    cmp     head(a0),d2             wieder platz im puffer ?
        bne     inbuff1                 ja, zeichen in puffer
        cmp.l   hz 200,d0               zeit schon abgelaufen ?
        bhi     wait                    nein, noch warten

        moveq   #0,d0                   zeichen nicht losgeworden
        rts

*       interrupt routine zur ausgabe eines zeichens an den drucker

busyint movem.l d0-d2/a0-a2,-(sp)       register retten
        bsr     getptr                  zeiger holen
        move    head(a0),d2
        cmp     tail(a0),d2             sendepuffer leer ?
        beq     leer                    ja, schon fertig
        bsr     wrap                    lese zeiger erhoehen
        move.l  (a0),a2                 puffer adresse
        move.b  (a2,d2),d1              zeichen aus puffer
        bsr     notbusy                 an drucker schicken
        move    d2,head(a0)             neuen head index merken
leer    bclr    #0,isrb(a1)             interrupt service bit loeschen
        movem.l (sp)+,d0-d2/a0-a2       register zurueck holen
        rte

getptr  lea     iorec,a0                zeiger auf puffer datensatz
        lea     mfp,a1
        rts

wrap    addq    #1,d2                   zeiger auf naechste position
        cmp     len(a0),d2              ende des puffers erreicht ?
        bcs     nowrap                  nein
        moveq   #0,d2                   sonst wieder am anfang beginnen
nowrap  rts
```

```
        .data
iorec   dc.l    buf             adresse des puffers
laenge  ds.w    1               groesse des puffers
        dc.w    0               schreib index
        dc.w    0               lese index

buffer  equ     0               offset im iorec
len     equ     4
head    equ     6
tail    equ     8

        .bss
trapsve ds.l    1               alter trap#13 vektor
buf     equ     *               start des puffer speichers
```

```
100 open "R",1,"spool.prg",16
110 field#1,16 as bin$
120 a$="": for i=1 to 16: read x$: if x$="*"then 150
130 a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140 lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150 data 60,1A,00,00,01,B0,00,00,00,0A,00,00,00,04,00,00
160 data 00,00,00,00,00,00,00,00,00,00,00,00,20,6F,00,04
170 data 2C,3C,00,00,01,00,DC,A8,00,0C,DC,A8,00,14,DC,A8
180 data 00,1C,7E,00,70,00,41,E8,00,81,10,18,90,3C,00,30
190 data 6B,0E,B0,3C,00,09,6E,08,CE,FC,00,0A,DE,40,60,EA
200 data 4A,47,66,02,7E,20,48,C7,70,0A,E1,AF,DC,87,33,C7
210 data 00,00,01,B4,2F,3C,00,00,00,7C,3F,3C,00,2D,3F,3C
220 data 00,05,4E,4D,50,8F,23,C0,00,00,01,BA,2F,3C,00,00
230 data 01,6C,3F,3C,00,00,3F,3C,00,0D,4E,4E,50,8F,42,67
240 data 2F,06,3F,3C,00,31,4E,41,24,4F,08,17,00,05,66,04
250 data 4E,6A,5D,4A,0C,6A,00,03,00,06,66,30,0C,6A,00,00
260 data 00,08,66,28,22,79,00,00,04,A2,33,1F,23,1F,23,C9
270 data 00,00,04,A2,32,2A,00,0A,61,42,22,79,00,00,04,A2
280 data 2F,19,3F,19,23,C9,00,00,04,A2,4E,73,0C,6A,00,08
290 data 00,06,66,20,0C,6A,00,00,00,08,66,18,70,FF,61,00
300 data 00,C2,34,28,00,08,61,00,00,C8,B4,68,00,06,66,02
310 data 70,00,4E,73,20,79,00,00,01,BA,4E,D0,46,FC,27,00
320 data 61,00,00,A0,34,28,00,06,B4,68,00,08,66,2E,08,11
330 data 00,00,66,28,45,F9,00,FF,88,00,14,BC,00,0F,15,41
340 data 00,02,14,BC,00,0E,10,12,C0,3C,00,DF,15,40,00,02
350 data 80,3C,00,20,15,40,00,02,70,FF,4E,75,34,28,00,08
360 data 61,6E,B4,68,00,06,67,0E,22,50,13,81,20,00,31,42
370 data 00,08,70,FF,4E,75,20,39,00,00,04,BA,D0,BC,00,00
380 data 17,70,46,FC,23,00,B4,68,00,06,66,DC,B0,B9,00,00
390 data 04,BA,62,F2,70,00,4E,75,48,E7,E0,E0,61,24,34,28
400 data 00,06,B4,68,00,08,67,0E,61,26,24,50,12,32,20,00
410 data 61,82,31,42,00,06,08,A9,00,00,00,10,4C,DF,07,07
420 data 4E,73,41,F9,00,00,01,B0,43,F9,00,FF,FA,01,4E,75
430 data 52,42,B4,68,00,04,65,02,74,00,4E,75,00,00,01,BE
440 data 00,00,00,00,00,00,00,00,00,44,06,12,06,88,AE,18
450 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
460 data *
470 close 1:if s<> 29742 then print "Fehler in DATAs!!": end
480 print "Ok."
```

2.5 Automatisches Starten von TOS-Anwendungen

Beim ATARI ST wird das Betriebssystem von Diskette in den RAM-Speicher des Rechners geladen und dort gestartet. Um diesen Vorgang einzuleiten, hat der Rechner ein sogenanntes Boot-ROM. Dort beginnt die Programmausführung des Rechners nach dem Einschalten. Die Aufgabe des Boot-ROMs besteht nun darin, einen Bootsektor von Diskette zu laden und das in ihm enthaltene Programm zu starten. Dieses Programm lädt dann das eigentliche Betriebssystem einschließlich des GEMs.

Der Bootsektor belegt den ersten Sektor auf der Diskette (Track Null, Sektor eins) und enthält außer dem Bootprogramm noch Daten über das Diskettenformat, wie Kapazität, Anzahl Tracks und Sektoren sowie Größe und Anordnung des Directories. Das oben beschriebene Boot-Programm ist nur auf Systemdisketten enthalten. Damit der Rechner dies erkennen kann, ist die Checksumme über den kompletten Sektor (256 Worte gleich 512 Bytes) bei einer Systemdiskette gleich $1234.

Normalerweise wird nach dem Laden des Betriebssystems das GEM Desktop gestartet, die gewohnte Benutzeroberfläche des ATARI ST. Im Betriebssystem ist jedoch die Möglichkeit vorgesehen, nach dem Laden nicht ins GEM Desktop zu verzweigen, sondern statt dessen ein Programm namens COMMAND.PRG zu laden und zu starten. Das kann z.B. ein Anwenderprogramm sein, das unter TOS läuft. Wie veranlaßt man das Betriebssystem dazu?

Die Entscheidung über das Laden dieses Programms wird schon im Bootsektor getroffen. Enthält der Bootsektor an einer bestimmten Stelle den Wert Null, so wird wie gewöhnlich das Desktop geladen. Ist der Wert dieser Adresse jedoch ungleich Null, so wird statt dessen das oben genannte Programm COMMAND.PRG geladen. Der Wert aus dieser Speicherstelle wird nach dem Laden des Bootsektors in die Betriebssystemvariable

'cmdload' ($482) kopiert. Ist das Betriebssystem geladen, so wird anhand dieser Variablen entschieden, ob das Programm COMMAND.PRG geladen wird.

Wollen wir also statt des Desktop direkt eine Anwendung starten, müssen wir also den Bootsektor entsprechend modifizieren. Dazu haben wir ein kleines Hilfsprogramm geschrieben. Das Programm führt vier Schritte aus: Zuerst wird der Bootsektor vom Laufwerk A geladen, dann wird im Bootsektor das Flag für 'cmdload' gesetzt. Nun könnten wir den Bootsektor wieder zurückschreiben. Doch erinnern wir uns an die Checksumme, durch die ein ausführbarer Bootsektor gekennzeichnet ist. Durch das Ändern eines Bytes stimmt die Checksumme natürlich nicht mehr, das Betriebssystem erkennt solch eine Diskette nicht mehr als Systemdiskette. Statt die Checksumme selbst neu zu berechnen, machen wir es uns einfacher und benutzen dazu eine Funktion des Betriebssystems. Die Funktion 'protobt' dient dazu, einen Bootsektor zu erzeugen oder einen bestehenden zu verändern. Bei der Parameterübergabe geben wir an, daß der Bootsektor ausführbar werden soll; alle anderen Parameter sollen unverändert bleiben. Dadurch veranlassen wir, daß die Checksumme neu berechnet und in den Bootsektor eingetragen wird. Haben wir dies durchgeführt, kann der Sektor wieder zurückgeschrieben werden.

Jetzt können wir ein Anwenderprogramm, das nach dem Booten geladen und gestartet werden soll, unter dem Namen 'COMMAND.PRG' auf die Systemdiskette kopieren. Wenn wir jetzt neu booten, wird automatisch dieses Programm gestartet, ohne daß vorher ins Desktop verzweigt wird.

```
*
*       modifizieren des bootsectors fuer cmdload
*       LE 11/11/85
*

gemdos  equ     1

xbios   equ     14

floprd  equ     8       sector lesen
flopwr  equ     9       sector schreiben
protobt equ     18      boot sector generieren

cmdload equ     $1e     offset im boot sector

*               boot sector laden

        move    #1,-(sp)        einen sector
        move    #0,-(sp)        seite null
        move    #0,-(sp)        track null
        move    #1,-(sp)        sector null
        move    #0,-(sp)        laufwerk a
        clr.l   -(sp)
        move.l  #puffer,-(sp)   puffer adresse
        move    #floprd,-(sp)   boot sector lesen
        trap    #xbios
        add.l   #20,sp

        tst     d0              fehler aufgetreten ?
        bne     exit            ja, abbrechen

*               boot sector modifizieren

        lea     puffer,a0       adresse des puffers
        move.b  #1,cmdload(a0)  flag fuer cmdload setzen
```

```
*               boot sector wieder ausfuehrbar machen

        move    #1,-(sp)        boot sector ausfuehrbar machen
        move    #-1,-(sp)       disk typ nicht aendern
        move.l  #-1,-(sp)       serial nr nicht aendern
        move.l  #puffer,-(sp)   adresse des boot sectors
        move    #protobt,-(sp)  funktion aufrufen
        trap    #xbios
        add.l   #14,sp

*               geaenderten boot sector zurueck schreiben

        move    #1,-(sp)        einen sector
        move    #0,-(sp)        seite null
        move    #0,-(sp)        track null
        move    #1,-(sp)        sector null
        move    #0,-(sp)        laufwerk a
        clr.l   -(sp)
        move.l  #puffer,-(sp)   puffer adresse
        move    #flopwr,-(sp)   boot sector schreiben
        trap    #xbios
        add.l   #20,sp

exit    clr     -(sp)
        trap    #gemdos         zurueck zum desktop

        .bss
puffer  ds.b    512             platz fuer einen sector
```

```
100 open "R",1,"a:cmdload.prg",16
110 field#1,16 as bin$
120 a$="": for i=1 to 16: read x$: if x$="*"then 150
130 a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140 lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150 data 60,1A,00,00,00,84,00,00,00,00,00,02,00,00,00
160 data 00,00,00,00,00,00,00,00,00,00,00,3F,3C,00,01
170 data 3F,3C,00,00,3F,3C,00,00,3F,3C,00,01,3F,3C,00,00
180 data 42,A7,2F,3C,00,00,00,84,3F,3C,00,08,4E,4E,DF,FC
190 data 00,00,00,14,4A,40,66,54,41,F9,00,00,00,84,11,7C
200 data 00,01,00,1E,3F,3C,00,01,3F,3C,FF,FF,2F,3C,FF,FF
210 data FF,FF,2F,3C,00,00,00,84,3F,3C,00,12,4E,4E,DF,FC
220 data 00,00,00,0E,3F,3C,00,01,3F,3C,00,00,3F,3C,00,00
230 data 3F,3C,00,01,3F,3C,00,00,42,A7,2F,3C,00,00,00,84
240 data 3F,3C,00,09,4E,4E,DF,FC,00,00,00,14,42,67,4E,41
250 data 00,00,00,18,16,1A,28,00,00,00,00,00,00,00,00,00
260 data *
270 close 1:if s<> 8274 then print "Fehler in DATAs!!": end
280 print "Ok."
```

2.6 C und Maschinensprache

Im folgenden Abschnitt wollen wir Ihnen anhand eines praktischen Beispiels demonstrieren, wie man in Assembler geschriebene Unterprogramme in C-Programm einbindet.

Für ein größeres Programmiervorhaben wird man wohl mit der Programmsprache C am schnellsten und sichersten zum Ziele kommen. Wenn es jedoch darum geht, zeitkritische Teile des Programms zu optimieren, so ist es oft angebracht, einzelne Unterprogramme in Assembler zu schreiben. Da der C-Compiler als Zwischenstufe selbst Assembler-Quellprogramme erzeugt, mag es auf den ersten Blick naheliegen, diese Quellprogramme von Hand zu optimieren. Dies mag in seltenen Fällen möglich sein; meist wird man jedoch nicht umhinkommen, das Problem selbst in Assembler zu lösen. Man braucht dann später beim Linken nur noch das Assemblerprogramm ähnlich wie sonst irgendwelche Bibliotheks-Unterprogramme dazu zu linken.

Doch wie kann man von C aus Parameter an das Unterprogramm übergeben und wie ein Ergebnis zurückbekommen? Dazu genügt es, das Prinzip einmal verstanden zu haben. Parameter werden grundsätzlich auf dem Stack übergeben.

```
int parameter1, parameter2;
long parameter3;
funktion(parameter1,parameter2,parameter3);
```

Der C-Compiler generiert aus solch einem Parameteraufruf folgende Sequenz:

```
move.l   parameter3,-(sp)
move.w   parameter2,-(sp)
move.w   parameter1,-(sp)
jsr      _funktion
addq.l   #8,sp
```

Sie sehen also, daß die Parameterliste von hinten nach vorne abgearbeitet wird. Dann wird die Funktion über einen

JSR-Aufruf angesprungen. Der C-Compiler stellt dem Namen einen Unterstrich voran. Damit der Linker den Namen im Assemblerprogramm wiederfinden kann, muß er dort als global erklärt werden. Für das Assemblerprogramm stellen sich die Parameter auf dem Stack wie folgt dar:

```
8(sp)    long, parameter3
6(sp)    word, parameter2
4(sp)    word, parameter1
0(sp)    long, Rücksprungadresse vom jsr-Aufruf
```

Der Programmierer hat dafür zu sorgen, daß die Typen der Parameter beim Aufruf und im Unterprogramm übereinstimmen; Compiler und Linker können solche Fehler nicht entdecken. Dies gilt jedoch für in C geschriebene Unterprogramme ebenso.

Weiterhin muß die Registerverwendung noch berücksichtigt werden. Ein Assemblerunterprogramm darf die Inhalte der Register D0-D2 und A0-A2 verändern, sämtliche anderen Registerinhalte dürfen nicht verändert werden. Wenn eine Funktion ein Ergebnis zurückliefert, so erwartet sie dies im Register D0. In diesem Falle geht der Compiler davon aus, daß der Funktionswert vom Typ int bzw. word ist, wie bei folgendem Aufruf:

```
a=funktion(parameter);
```

Gibt die Funktion ein long-Ergebnis, so muß dies vor dem Aufruf der Funktion explizit deklariert werden, z.B. so:

```
long funktion();
long a;

a=funktion(parameter);
```

Mit diesen Kentnissen steht Ihnen nun nichts mehr bei der Programmierung von Assemblerunterprogrammen im Wege. Als Beispiel für solch ein Unterprogramm haben wir einmal die

Anzeige des Directories genommen. Das Programm zeigt Ihnen außerdem, wie die einzelnen GEMDOS-Aufrufe anzuwenden sind. Im Anschluß daran finden Sie ein kleines Hauptprogramm in C, das den Aufruf veranschaulicht. Die Funktion erwartet zwei Parameter: Der erste bestimmt das Laufwerk (0= A, 1=B) und der zweite ist ein Auswahlstring, mit dem Sie z.B. Unterdirectories selektieren können. Geben Sie statt dessen einen Leerstring an, so werden sämtliche Files dieses Laufwerks angezeigt. Die Anzeige hält nach jeweils 20 Dateien an; sie kann dann durch Drücken einer Taste fortgeführt werden. Zum Abschluß wird noch der freie Speicherplatz in KByte angezeigt.

```
*
*       Directory anzeigen
*
*       LE 11/11/85
*

*       BIOS-Funktionen
bios    equ     13      TRAP#
conin   equ     2       consol eingabe
conout  equ     3       consol ausgabe
con     equ     2       consol device#

*       GEMDOS-Funktionen
gemdos  equ     1       TRAP#
wrtstr  equ     9       string ausgeben
setdrv  equ     $e      drive selektieren
setdma  equ     $1a     dma-adresse bestimmen
getspc  equ     $36     freie bytes
sfirst  equ     $4e     search first
snext   equ     $4f     search next

cr      equ     13      carriage return
lf      equ     10      line feed

filetyp equ     %11001  file attribut

wrtchar move    d0,-(sp)        zeichen in d0 ausgeben
        move    #con,-(sp)
        move    #conout,-(sp)
        trap    #bios
        addq.l  #6,sp
        rts

blank   move.b  #' ',d0         leerzeichen ausgeben
        bra     wrtchar

newline lea     crlf(pc),a0     neue zeile
```

```
wrttxt  move.l  a0,-(sp)        adresse des texts
        move    #wrtstr,-(sp)   string ausgeben
        trap    #gemdos
        addq.l  #6,sp
        rts

        .globl  _directory      zugriff fuer C ermoeglichen

*       6(sp)   zeiger auf filenamen
*       4(sp)   drive nummer
*       0(sp)   ruecksprung adresse

_directory:

        move    4(sp),curdrv    drive nummer
        move.l  6(sp),a0        file namen
        movem.l d3-d7/a3-a6,-(sp)       C-register retten
        move.l  a0,a3
        move.l  #dmabuf,-(sp)
        move    #setdma,-(sp)   dma puffer adresse
        trap    #gemdos
        addq.l  #6,sp
        move    curdrv,-(sp)
        move    #setdrv,-(sp)   drive selectieren
        trap    #gemdos
        addq.l  #4,sp
        tst.b   (a3)            filename vorhanden ?
        bne     dir1            ja
        lea     allfile(pc),a3  '*.*' als name
dir1    move    #filetyp,-(sp)
        move.l  a3,-(sp)        zeiger auf filenamen
        move    #sfirst,-(sp)
        trap    #gemdos
        addq.l  #8,sp
        tst     d0              file vorhanden ?
        bne     enddir
dircont moveq   #20-1,d7        anzahl zeilen
nxtfile bsr     wrtname         filenamen ausgeben
        move.l  size,d0         groesse in bytes
```

```
        bsr     wrtlng          als decimalzahl ausgeben
        bsr     blank
        move    date,d3         datum
        bsr     wrtdate         ausgeben
        bsr     blank           leerzeichen
        move    time,d3         zeit
        bsr     wrttime         ausgeben
        bsr     newline         neue zeile
        move    #snext,-(sp)
        trap    #gemdos         naechstes file suchen
        addq.l  #2,sp
        tst     d0              vorhanden ?
        dbne    d7,nxtfile
        bne     enddir          nein
        move    #con,-(sp)      auf tastendruck warten
        move    #conin,-(sp)
        trap    #bios
        addq.l  #4,sp
        bra     dircont         und weitermachen

enddir  move    curdrv,-(sp)    drive
        addq    #1,(sp)         1=a, 2=b
        move.l  #buffer,-(sp)
        move    #getspc,-(sp)   freien platz auf diskette
        trap    #gemdos
        addq.l  #8,sp
        move    buffer+2,d0     groesse
        bsr     wrt3dec         decimal 3stellig ausgeben
        lea     kfree(pc),a0
        bsr     wrttxt
        movem.l (sp)+,d3-d7/a3-a6       C-register zurueck
return  rts

wrtname lea     filenam,a6      filename formatiert ausgeben
        clr     d6
namloop move.b  (a6)+,d0        zeichen holen
        beq     endnam1         name zu ende ?
        cmp.b   #'.',d0
        beq     extens          bei extension weitermachen
```

```
        addq    #1,d6
        bsr     wrtchar         zeichen ausgeben
        bra     namloop
extens  cmp     #9,d6           name auf 8 stellen auffuellen
        beq     weiter
        addq    #1,d6
        bsr     blank           mit blanks auffuellen
        bra     extens
weiter  move.b  (a6)+,d0        extension ausgeben
        beq     endnam1
        addq    #1,d6
        bsr     wrtchar
        bra     weiter
endnam1 cmp     #14,d6          ende des namens ?
        beq     return
        bsr     blank           mit blanks auffuellen
        addq    #1,d6
        bra     endnam1

wrtdate bsr     blank           datum ausgeben
        move    d3,d0
        and     #%11111,d0      tag isolieren
        bsr     wrt2dec         und ausgeben
        bsr     wrtpkt          '.' als trennzeichen
        move    d3,d0
        lsr     #5,d0
        and     #%1111,d0       monat isolieren
        bsr     wrt2dec         und ausgeben
        bsr     wrtpkt          '.' als trennzeichen
        move    d3,d0
        lsr     #8,d0
        lsr     #1,d0           jahr isolieren
        add     #80,d0          offset addieren
        bra     wrt2dec         und ausgeben

wrtpkt  move.b  #'.',d0         punkt ausgeben
        bra     wrtchar
```

```
wrttime bsr     blank           zeit ausgeben
        move    d3,d0
        lsr     #8,d0
        lsr     #3,d0           stunden isolieren
        bsr     wrt2dec         und ausgeben
        bsr     wrtcol          ':' als trennzeichen
        move    d3,d0
        lsr     #5,d0
        and     #%111111,d0     minuten isolieren
        bsr     wrt2dec         und ausgeben
        bsr     wrtcol          ':' als trennzeichen
        move    d3,d0
        and     #%11111,d0
        lsl     #1,d0           sekunden isolieren
        bra     wrt2dec         und ausgeben

wrtcol  move.b  #':',d0         doppelpunkt ausgeben
        bra     wrtchar

wrt3dec moveq.l #3,d6           d0 3stellig ausgeben
        clr     d4              vornullen unterdruecken
        ext.l   d0
        bra     wrtlng1

wrt2dec moveq   #2,d6
        ext.l   d0              d0 als zweistellige dec-zahl
        st      d4              vornullen nicht unterdruecken
        bra   wrtlng1

*           hexzahl in d0.l nach decimal

wrtlng  clr     d4              flag fuer vornullen unterdruecken
        moveq   #10,d6

wrtlng1 movem.l d1-d3/d6-d7,-(sp)
        move.l  d0,d7
wrtdec5 moveq   #1,d2
        move.l  d6,d1
        subq.l  #1,d1
```

```
        beq     wrtdec1
wrtdec0 move    d2,d3           10*d3.l nach d3
        mulu    #10,d3
        swap    d2
        mulu    #10,d2
        swap    d3
        add     d3,d2
        swap    d2
        swap    d3
        move    d3,d2
        subq.l  #1,d1
        bne     wrtdec0
wrtdec1 clr.l   d0
wrtdec3 cmp.l   d2,d7
        blt     wrtdec2
        addq.l  #1,d0
        sub.l   d2,d7
        bra     wrtdec3
wrtdec2 tst.b   d0              null ?
        bne     wrtdec4         nein, ausgeben
        tst     d4
        bne     wrtdec4         vornullen unterdruecken
        cmp     #1,d6           letzte stelle ?
        beq     wrtdec4         ja, dann null ausgeben
        bsr     blank           vornullen als blanks ausgeben
        bra     wrtdec6
wrtdec4 add.b   #'0',d0
        bsr     wrtchar         ziffer ausgeben
        st      d4              flag setzen
wrtdec6 subq.l  #1,d6
        bne     wrtdec5
        movem.l (sp)+,d1-d3/d6-d7
        rts

allfile dc.b    "*.*",0         alle files
kfree   dc.b    " K free."
crlf    dc.b    cr,lf,0
```

```
        .bss
dmabuf  ds.b    22      dmabuffer fuer gemdos
time    ds.w    1       uhrzeit
date    ds.w    1       datum
size    ds.l    1       file groesse
filenam ds.b    14      file namen
curdrv  ds.w    1       aktuelle drive nummer
buffer  ds.b    16      puffer fuer file groesse
```

Das folgende kleine Programm in C kann als Test für das Directory-Unterprogramm dienen.

```
/*
 *      testprogramm fuer directory-anzeige
 *      LE 11/11/85
 */

main()
{

        directory (0,"");       /* drive a, alle files  */

        directory (1,"*.PRG");  /* drive b, nur prg-files */

}
```

Wenn Sie das C-Quellprogramm 'direc.c' nennen und das Assemblerunterprogramm mit 'dir.s' bezeichnen, dann müssen Sie nach dem Compilieren bzw. Assemblieren beim Linken folgende Kommandozeile benutzen:

dir.68k=apstart,direc,dir

```
100 open "R",1,"dir.prg",16
110 field#1,16 as bin$
120 a$="": for i=1 to 16: read x$: if x$="*"then 150
130 a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140 lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150 data 60,1A,00,00,02,74,00,00,01,64,00,00,04,42,00,00
160 data 00,00,00,00,00,00,00,00,00,00,00,00,2A,4F,2E,7C
170 data 00,00,07,D8,2A,6D,00,04,20,2D,00,0C,D0,AD,00,14
180 data D0,AD,00,1C,D0,BC,00,00,01,00,2F,00,2F,0D,3F,00
190 data 3F,3C,00,4A,4E,41,DF,FC,00,00,00,0C,4E,B9,00,00
200 data 00,4A,2F,3C,00,00,00,00,4E,41,22,2F,00,04,30,3C
210 data 00,C8,4E,42,4E,75,4E,56,FF,FC,2E,BC,00,00,03,D0
220 data 42,67,4E,B9,00,00,00,9A,54,8F,2E,BC,00,00,03,D1
230 data 3F,3C,00,01,4E,B9,00,00,00,9A,54,8F,4E,5E,4E,75
240 data 3F,00,3F,3C,00,02,3F,3C,00,03,4E,4D,5C,8F,4E,75
250 data 10,3C,00,20,60,EA,41,FA,01,E4,2F,08,3F,3C,00,09
260 data 4E,41,5C,8F,4E,75,33,EF,00,04,00,00,08,08,20,6F
270 data 00,06,48,E7,1F,1E,26,48,2F,3C,00,00,07,DC,3F,3C
280 data 00,1A,4E,41,5C,8F,3F,39,00,00,08,08,3F,3C,00,0E
290 data 4E,41,58,8F,4A,13,66,04,47,FA,01,96,3F,3C,00,19
300 data 2F,0B,3F,3C,00,4E,4E,41,50,8F,4A,40,66,46,7E,13
310 data 61,70,20,39,00,00,07,F6,61,00,01,16,61,92,36,39
320 data 00,00,07,F4,61,00,00,9E,61,86,36,39,00,00,07,F2
330 data 61,00,00,C0,61,80,3F,3C,00,4F,4E,41,54,8F,4A,40
340 data 56,CF,FF,CE,66,0E,3F,3C,00,02,3F,3C,00,02,4E,4D
350 data 58,8F,60,BA,3F,39,00,00,08,08,52,57,2F,3C,00,00
360 data 08,0A,3F,3C,00,36,4E,41,50,8F,30,39,00,00,08,0C
370 data 61,00,00,AC,41,FA,01,1E,61,00,FF,40,4C,DF,78,F8
380 data 4E,75,4D,F9,00,00,07,FA,42,46,10,1E,67,28,B0,3C
390 data 00,2E,67,08,52,46,61,00,FF,08,60,EE,BC,7C,00,09
400 data 67,08,52,46,61,00,FF,0A,60,F2,10,1E,67,08,52,46
410 data 61,00,FE,EE,60,F4,BC,7C,00,0E,67,C4,61,00,FE,F2
420 data 52,46,60,F2,61,00,FE,EA,30,03,C0,7C,00,1F,61,56
430 data 61,18,30,03,EA,48,C0,7C,00,0F,61,4A,61,0C,30,03
440 data E0,48,E2,48,D0,7C,00,50,60,3C,10,3C,00,2E,60,00
450 data FE,B0,61,00,FE,BC,30,03,E0,48,E6,48,61,28,61,16
460 data 30,03,EA,48,C0,7C,00,3F,61,1C,61,0A,30,03,C0,7C
470 data 00,1F,E3,48,60,10,10,3C,00,3A,60,00,FE,84,7C,03
480 data 42,44,48,C0,60,0E,7C,02,48,C0,50,C4,60,06,48,C0
```

```
490 data 42,44,7C,0A,48,E7,73,00,2E,00,74,01,22,06,53,81
500 data 67,1A,36,02,C6,FC,00,0A,48,42,C4,FC,00,0A,48,43
510 data D4,43,48,42,48,43,34,03,53,81,66,E6,42,80,BE,82
520 data 6D,06,52,80,9E,82,60,F6,4A,00,66,10,4A,44,66,0C
530 data BC,7C,00,01,67,06,61,00,FE,38,60,0A,D0,3C,00,30
540 data 61,00,FE,1E,50,C4,53,86,66,B0,4C,DF,00,CE,4E,75
550 data 2A,2E,2A,00,20,4B,20,66,72,65,65,2E,0D,0A,00,00
560 data 00,01,00,02,01,01,02,01,01,00,01,01,02,01,01,01
570 data 01,01,00,00,00,00,00,00,00,00,00,01,00,00,01
580 data 00,03,05,00,05,05,00,00,01,01,02,01,00,10,07,01
590 data 02,01,00,00,00,00,00,00,00,00,00,00,01,01,01,02
600 data 01,01,02,01,01,02,01,01,01,01,02,01,01,01,00,00
610 data 00,00,00,00,00,00,00,00,00,00,02,01,01,01,01,01
620 data 06,01,01,04,01,01,01,03,01,02,01,01,04,02,01,08
630 data 01,01,00,00,00,00,00,00,01,01,01,09,01,01,01,01
640 data 01,01,01,00,00,05,01,00,00,00,00,00,00,00,00,00
650 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
660 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
670 data 00,00,00,00,04,03,00,08,03,00,06,01,00,08,01,00
680 data 08,01,00,04,01,01,03,01,01,00,05,00,01,01,01,00
690 data 05,00,00,01,01,00,01,01,00,00,00,00,00,00,00,00
700 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
710 data 00,02,02,00,00,00,00,00,00,00,00,00,00,00,00,00
720 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,05,01
730 data 00,05,01,00,01,01,00,01,01,00,02,05,00,06,01,00
740 data 02,01,00,01,01,00,06,05,00,00,00,00,00,01,01,00
750 data 01,00,02,01,00,02,01,01,01,01,01,00,00,00,00,00
760 data 00,00,00,00,00,00,00,00,00,00,00,01,02,03,01,02
770 data 01,01,01,01,01,01,00,01,01,00,01,02,00,2A,2E,50
780 data 52,47,00,00,00,00,00,04,2E,1E,08,08,0A,34,10,0E
790 data 2C,0C,0C,2A,08,0E,18,00,00,00,00,00,00,00,00,00
800 data *
810 close 1:if s<> 49700 then print "Fehler in DATAs!!": end
820 print "Ok."
```

3 HARDCOPY IN FARBE

Zu den faszinierendsten Themen beim ATARI ST gehören sicherlich die vielfältigen Grafikmöglichkeiten.
Wie dicht und angenehm für die Augen der Hires-Modus auf dem mitgelieferten monochromen Monitor ist, brauchen wir Ihnen sicher nicht erzählen. Dessen Vorzüge können Sie gleich nach dem Einschalten genießen.
Die Erfahrungen mit den farblichen Qualitäten mußten bisher allerdings verborgen bleiben, da ein entsprechender Farbmonitor von ATARI derzeit nicht angeboten wird. Wie Sie sich denken können, ließ uns das Thema Farbe keine Ruhe und wir suchten eifrig nach einer Lösung. Wie Sie an der Titelseite dieses Kapitels sehen können, haben wir eine gefunden. Nähere Einzelheiten darüber erfahren Sie im nächsten Abschnitt.

Was uns weiter beschäftigte war die Frage, wie solche traumschönen Bilder wohl zu Papier zu bringen seien. Zwar ist im GEMDOS eine Hardcopyroutine vorhanden. Diese arbeitet aber nur auf einem 'normalen' Drucker. Die verschiedenen Farben werden hierbei in unterschiedlichen Grautönen dargestellt. Es gibt wohl auch eine Routine für einen speziellen Farbdrucker, aber für welchen haben wir noch nicht herausfinden können. Also haben wir eine eigene geschrieben, die so parametrisiert ist, daß sie leicht an verschiedene Fabrikate angepaßt werden kann.
Damit noch nicht zufrieden, suchten wir angeregt durch eine Demoversion des Zeichenprogrammes GEMDRAW nach einer Möglichkeit, auch handelsübliche (Farb-)Plotter zu einer Hardcopy zu bewegen. Da taten sich zunächst Abgründe auf, da uns ein geeigneter Algorithmus nicht bekannt war. Aber auch dieses Problem ließ sich lösen, wie Sie in 3.3.2 sehen werden.

Damit Sie von alledem auch mehr haben als nur zwei durch stures Abtippen erhaltene Programme von möglicherweise für Sie fragwürdiger Brauchbarkeit, haben wir noch einen Abschnitt über den Aufbau des Grafikram vorangestellt, der

sicher sehr hilfreich zur Realisierung Ihrer eigenen Vorstellungen ist.
Nicht zuletzt sind natürlich die Programme dokumentiert, so daß Ihnen zweckdienliche Veränderungen nicht schwerfallen sollten.

Damit auch diejenigen unter Ihnen, die nicht über einen Assembler usw. verfügen, in den Genuß der Programme kommen, haben wir beide auch noch als BASIC-Programm abgedruckt.
Das Ganze ist mit Bildschirmfotos und Hardcopies reichlich garniert, um Arbeitsweise und Problematik der Verfahren anschaulich zu machen.

Achtung: Aus produktionstechnischen Gründen mußten die Farbseiten zusammengefaßt werden. Wenn Sie also eine Abbildung nicht in der Nähe des Bezuges vorfinden, schauen Sie bitte im Farbteil nach.

So, genug der Vorrede. Wir beginnen damit, wie Sie einen preiswerten Farbmonitor an Ihren ATARI anschließen können.

3.1 Anschluß eines Farbmonitors

Es juckt einem schon mächtig in den Fingern, wenn man weiß, daß der ATARI über hervorragende Farbmöglichkeiten verfügt, man jedoch vorerst keine Chance sieht, diese auch sichtbar zu machen.

So galt es zunächst, einen geeigneten Farbmonitor ausfindig zu machen.
Die handelsüblichen Farbbildschirme, wie sie in großer Auswahl für die IBM-Kompatiblen zu haben sind, schieden von vornherein aus, da sie nur über einen digitalen RGB-Eingang verfügen, das heißt, die entsprechende Farbe ist entweder an oder aus, kombiniert mit einem Helligkeitsbit. Der ATARI stellt jedoch die einzelnen Farben in acht analogen Stufen (drei Bit für jede Farbe) zur Verfügung. Passende Monitore gibt es wohl, aber zu einem Preis, der gewöhnlich den Hobby-Etat sprengt.
Bei der Arbeit mit anderen Home-Computern kam uns der Schneider CPC unter die Hände. Der zugehörige Farbmonitor (CTM 640) ist analoger Natur und zum Glück auch einzeln erhältlich. Blitzschnell wurde ein entsprechendes Kabel angefertigt. Das Ergebnis war überwältigend, wenngleich durch die etwas breiten Stege der Schlitzmaske des Bildschirmes der Eindruck einer groben Rasterung entsteht. Ein eigenes Urteil können Sie sich leicht anhand der vielen Bildschirmfotos bilden.

Wir wollen Ihnen natürlich den Aufbau des Adapterkabels nicht vorenthalten. Hier ist er.

Alles, was Sie brauchen, ist ein kleines Stück Lochrasterplatine mit einem Raster von 1/10" (2.54mm), 12 Vierkantstifte, zwei Dioden (1N4148 o.ä.), eine 6-polige DIN-Kupplung und eine vieradrige abgeschirmte Leitung. Das war's.

Den Aufbau und die Verdrahtung sehen Sie auf den Abbildungen 3.1-1 und 3.1-2.
Falls es Sie interessiert: Die Dioden dienen dazu, die beiden Synchronisationssignale (vertikal und horizontal) des ATARI zu mischen, da der Monitor nur über einen SYNC-Eingang verfügt.
Den Pin 13 auf der ATARI-Seite können Sie weglassen. Er paßt nicht in das Raster der Platine.

Achtung: Möglicherweise wird das Bild vertikal durchlaufen. Sie können es durch Drehen des Reglers an der Rückseite des Monitors zum Stillstand bringen.

Und nun viel Spaß.

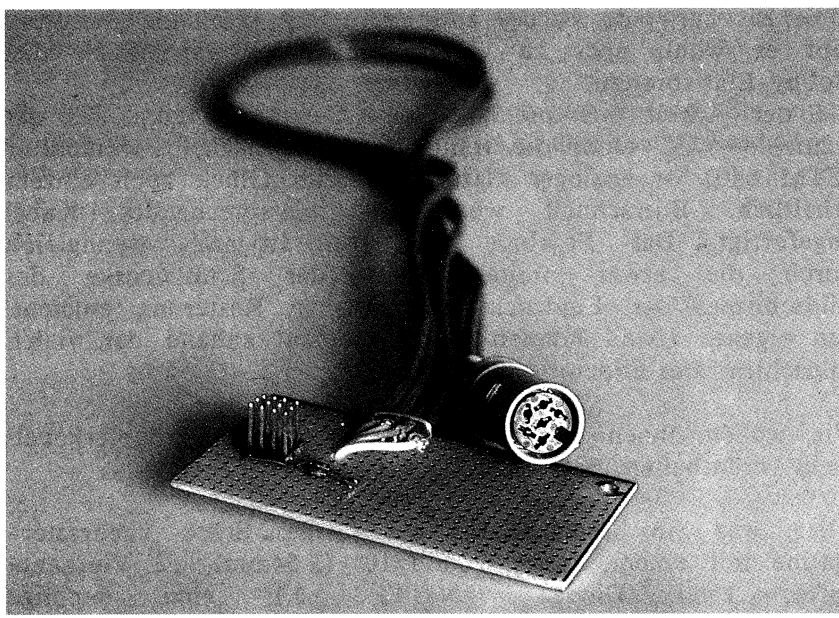


Abb. 3.1-1

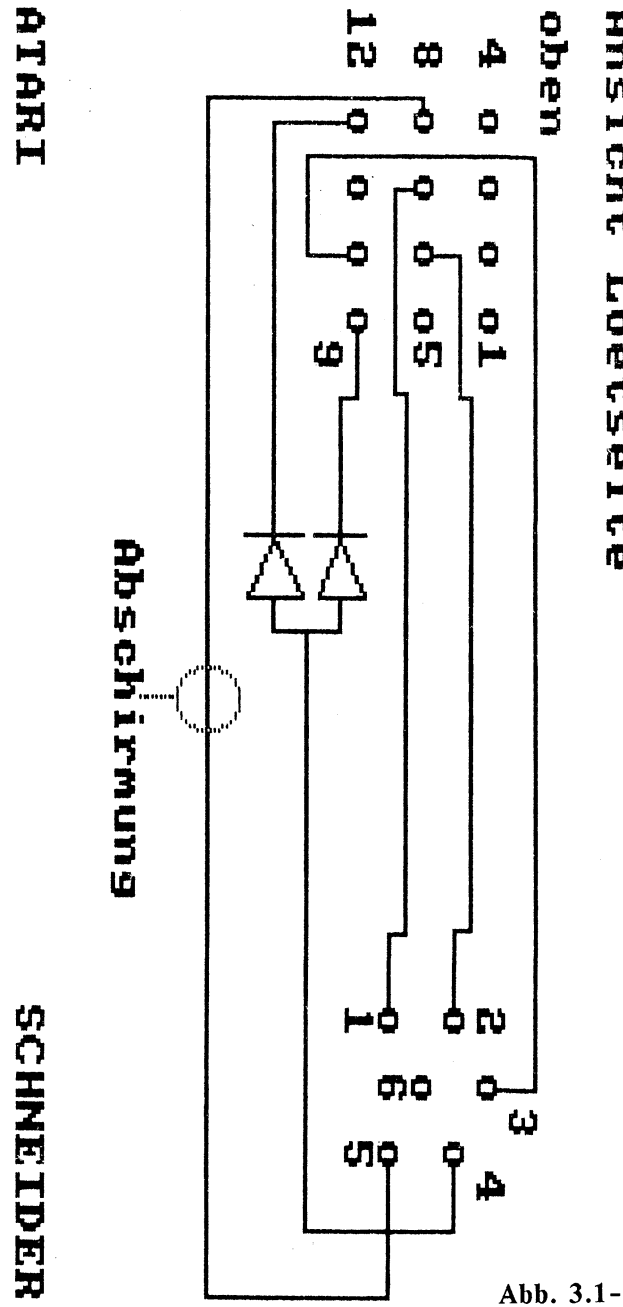

Ansicht Loetseite
oben
4
8
12
01
05
9
2
1
6
3
5
4
Abschirmung
ATARI
SCHNEIDER

Abb. 3.1-2

3.2 Aufbau des Grafik-RAM

Des ATARI ST Stärke und Schwäche zugleich ist das Video-RAM, also der Teil des Speichers, der für den Bildaufbau zuständig ist. Dieser 16K große Bereich ist grafikmäßig orientiert, d.h. bei normaler Schrift wird das Muster beim Wiederauffrischen des Bildes nicht etwa schnell aus einem im Rom befindlichen Zeichengenerator geholt, wie vielfach üblich, sondern die Bildpunkte stammen unmittelbar aus dem Videoram, wohinein sie, um bei der Schrift zu bleiben, aus einem Zeichengenerator kopiert wurden. Man kann dieses Verfahren als Schwäche ansehen, weil es naturgemäß bedeutend langsamer ist als mit einem Hardware-Zeichengenerator, was besonders beim Scrollen des Bildschirmes auffallen könnte, denn es ist schon ein Unterschied, ob 16K oder nur 2K verschoben werden müssen. Auffallen könnte, wenn der Prozessor 68000 nicht so unheimlich schnell wäre.
In dem gleichen Umstand ist auch die Stärke des ATARI zu sehen, denn wenn das Videoram ohnehin schon grafikorientiert ist, lassen sich auch bequem Punkte an beliebige Stellen setzen und auf diese Weise Bilder erzeugen.

Hier soll nun besprochen werden, wie der Aufbau bezüglich der Farben organisiert ist.

Am durchsichtigsten ist natürlich der Modus 2, also die hochauflösende Betriebsart mit zwei Farben, nämlich schwarz und weiß. In Abb. 3.2-1 sehen Sie das Schema. Abb. 3.2-2 zeigt eine Hardcopy in diesem Modus. Das Videoram ist wortweise organisiert. Da es nur zwei Farben gibt, genügt zur Bestimmung der Farbnummer ein Bit, d.h. ein Bit im Videoram entspricht einem Punkt auf dem Bildschirm. Die Reihenfolge ist so zu verstehen, daß das höchstwertige Bit des ersten Wortes im Videoram dem Bildpunkt links oben entspricht.

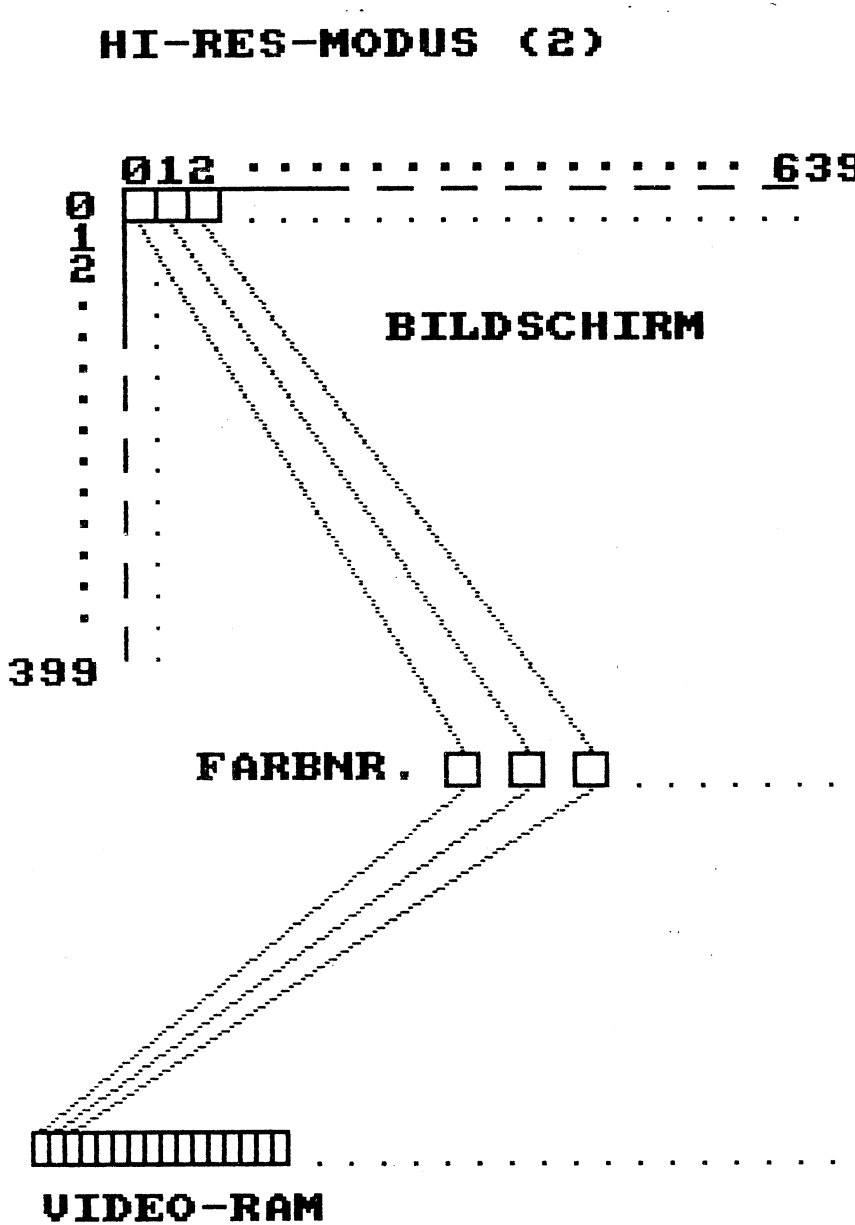


Abb. 3.2-1

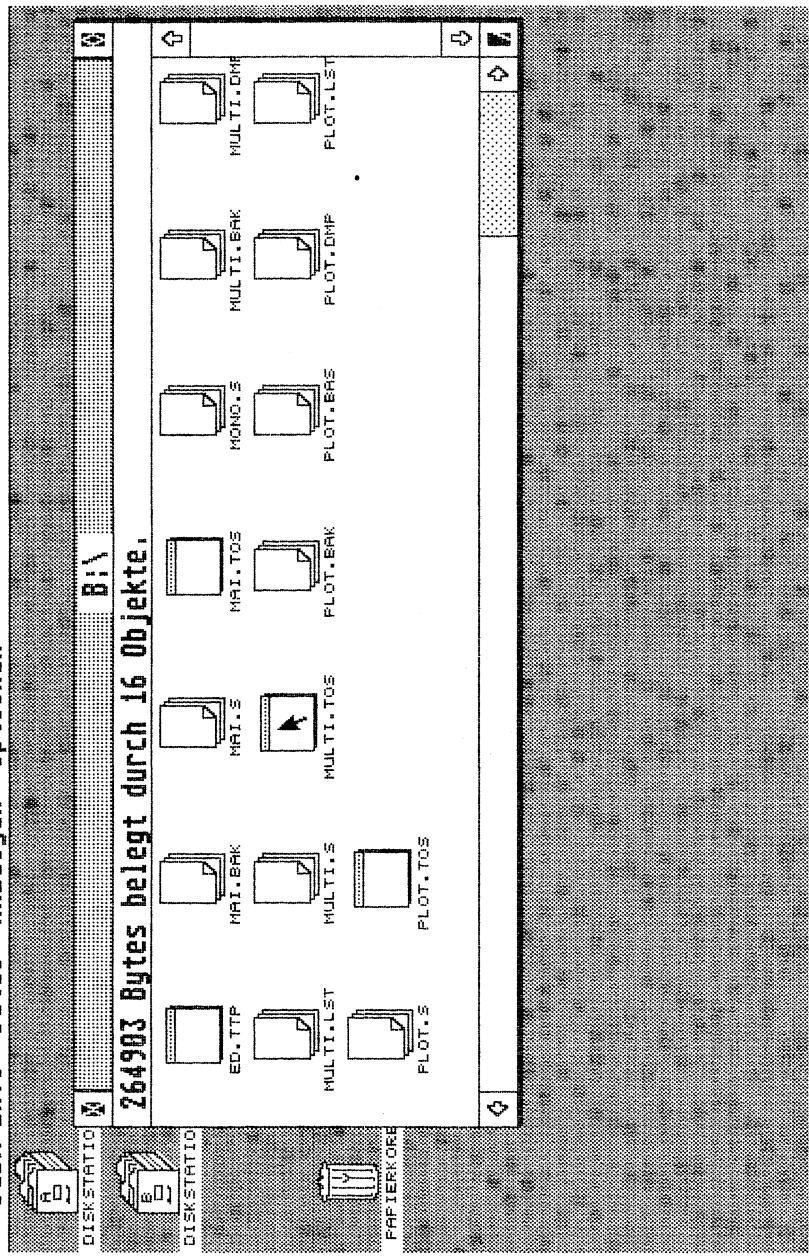

Desk-Info Datei Anzeigen Optionen
B:\
264903 Bytes belegt durch 16 Objekte.
ED.TTP
MAI.BAK
MAI.S
MAI.TOS
MONO.S
MULTI.BAK
MULTI.DMP
MULTI.LST
MULTI.S
MULTI.TOS
PLOT.BAK
PLOT.BAS
PLOT.DMP
PLOT.LST
PLOT.S
PLOT.TOS
DISKSTATIO
DISKSTATIO
PAPIERKORB

Abb. 3.2-2

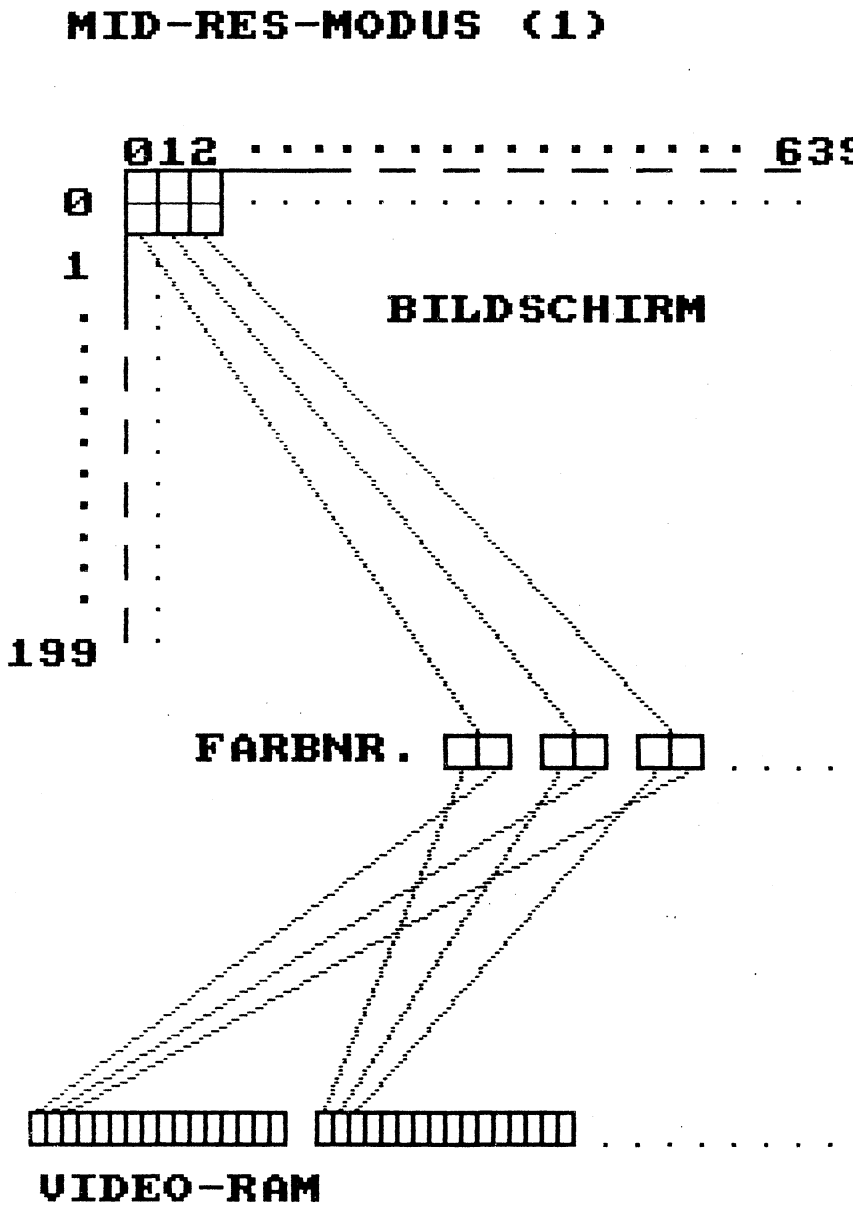


Abb. 3.2-3

Etwas schwerer ist der Modus 1 zu verstehen. Hier stehen ja 640 mal 200 Punkte in vier Farben zur Verfügung. Die Frage ist, wie die Farbinformation versteckt ist, denn die Zuordnung 1 Bit = 1 Punkt kann ja nicht mehr zutreffen.
In der Tat sind hier zwei Bits für einen Punkt zuständig, indem sie die Farbnummer dieses Punktes bilden, nämlich 0 bis 3. Wie Sie in Abb. 3.2-3 sehen können, werden diese beiden Bits nicht etwa demselben Wort entnommen, sondern es werden immer zwei korrespondierende Bits aus zwei benachbarten Wörtern zur Farbnummer zusammengefaßt. Damit der Bildschirm bei der vertikal verringerten Punktzahl dennoch 'voll' wird, werden die Punkte sozusagen senkrecht ein wenig in die Länge gezogen, so daß ein Punkt in Wirklichkeit ein kurzer vertikaler Strich ist.
Auf Abb. 3.2-4 im Farbteil sehen Sie, daß die Buchstaben wirklich etwas länglich wirken.

Im Modus 0 schließlich, der mit seinen 320 mal 200 Punkten zwar die geringste Auflösung, dafür aber mit 16 Farben die größte Pracht bietet, sind gleich vier Bits für einen Punkt zuständig, mit denen eine Farbnummer zwischen 0 und 15 gebildet wird. Hier werden gleichwertige Bits aus vier aufeinanderfolgenden Wörtern zusammengefaßt. Beachten Sie bitte, falls es aus Abb. 3.2-5 nicht so deutlich wird, daß das Bit des jeweils ersten Wortes aus einem Quartett das niedrigstwertige Bit der Farbnummer bildet.
Zum Aufbau des Bildes werden die Punkte hier nicht nur in die Länge gezogen, sondern auch horizontal verdoppelt. Wie Sie auf Abb. 3.2-6 sehen können, stimmen nun die Proportionen wieder.

Soviel zum Aufbau des Videoram.

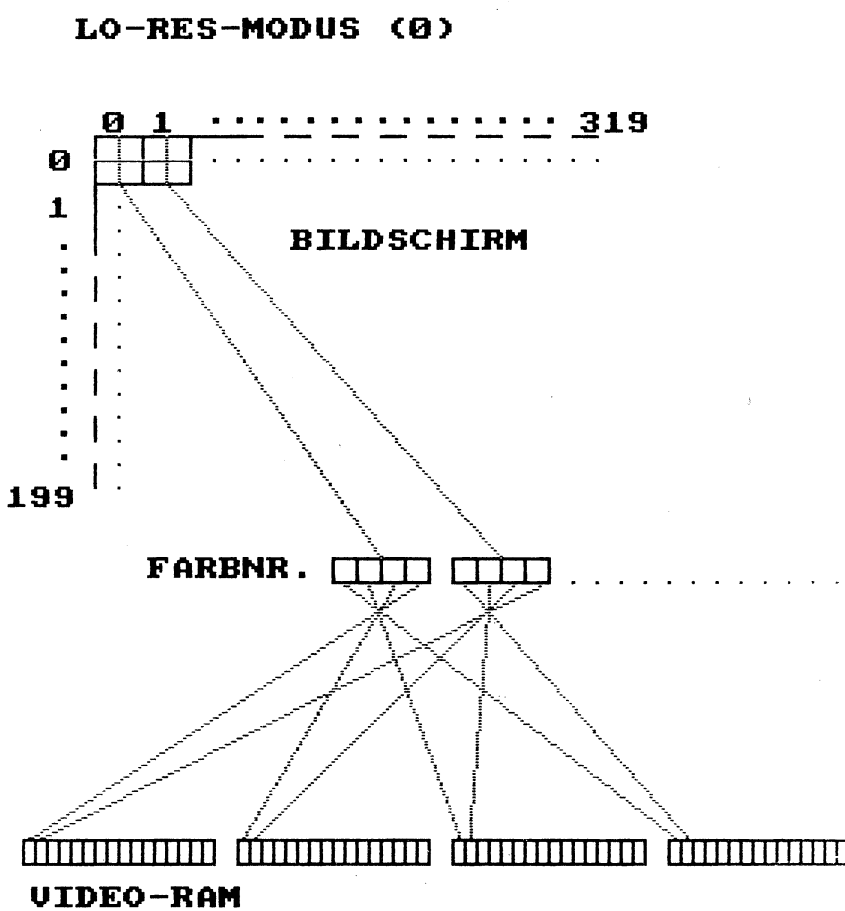


Abb. 3.2-5

3.3 Hardcopy

Wie in der Einleitung zu diesem Kapitel schon erwähnt, werden wir Ihnen zwei Verfahren zur Hardcopy anbieten, nämlich einmal für einen Matrixdrucker und einmal für einen Plotter.
Die Programme sind für handelsübliche Geräte gedacht, so daß Sie im Bedarfsfalle keinerlei Beschaffungsschwierigkeiten haben dürften. Wir haben Geräte von EPSON gewählt, weil diese zum einen recht weit verbreitet sind und zum anderen Ihren Geldbeutel nicht allzu sehr belasten.
Natürlich können Sie ohne weiteres auch andere Geräte benutzen. Die Programme sind so aufgebaut, daß sie durch Änderung nur weniger Konstanten leicht den jeweiligen Erfordernissen angepaßt werden können.

Gehandhabt werden die Programme so, daß sie einfach gestartet werden. Das gilt gleichermaßen für die Assembler- und Basic-Versionen. Danach tut sich nichts, denn das Programm hat sich nur selbst hinter das Video-RAM (dort sind gerade noch 768 Bytes frei) kopiert und bleibt dort für die Zeit, in der der Rechner eingeschaltet ist. Ausgelöst wird die Hardcopy genau so wie die 'eingebaute' durch Drücken der Tasten ALTERNATE und HELP. Das Programm hat die alte Routine praktisch ersetzt.

Wir bieten Ihnen zwei Verfahren deshalb an, weil sowohl Drucker als auch Plotter ihre Stärken auf ganz bestimmten Gebieten haben. So ist z.B. ein Matrixdrucker durchaus in der Lage, relativ feine Farbnuancen durch Rasterung wiederzugeben, nicht jedoch, einen wirklich dichten und dicken Strich zu ziehen. Dies wiederum kann ein Plotter ganz hervorragend.
Entsprechend sind auch die bevorzugten Anwendungen: Drucker, wenn viele Farben in vielen Abstufungen zu reproduzieren sind, Plotter, wenn wenige Farben (vorzugsweise nur eine) im Stile einer technischen Zeichnung für reprofähige Vorlagen verlangt werden.

Von den jeweiligen Vorzügen können Sie sich anhand der Bilder leicht selbst überzeugen. Für jede Auflösungsstufe ist mindestens ein Beispiel mit Foto und Hardcopy vorhanden.

Die beiden Programme unterscheiden sich auch hinsichtlich des Zugriffs auf das Video-RAM. Bei der Druckerhardcopy wird physikalisch zugegriffen, bei der Plotterversion über den Line-A-Emulator. So lernen Sie beide Möglichkeiten kennen.

Ein Wort noch zu den Bildern: Diese stammen fast ausnahmslos aus Demoprogrammen von ATARI. Wir durften sie mit freundlicher Genehmigung von ATARI Deutschland in diesem Buch abdrucken, wofür wir den Verantwortlichen herzlich danken.

3.3.1 MATRIXDRUCKER

Wir hätten nicht gedacht, daß eine simple Hardcopy solche Probleme grundsätzlicher Natur aufwerfen könnte. Die Farbenzuordnung war noch das geringste davon.
Länger haben wir über der Frage gebrütet, ob denn nun eine voll 'aufgezogene' Farbe auf dem Papier kräftiger oder schwächer abgebildet werden soll. Die Problematik, die dahinter steht, ist folgende: Der Bildschirm ist ohne Farbe dunkel, das Papier jedoch hell. Macht man eine auf dem Bildschirm kräftige (und damit auch helle) Farbe auf dem Papier ebenfalls kräftig (und damit dunkler), gehen evtl. beabsichtigte Schattierungseffekte (vgl. Abb. 3.3.1-7) verloren.
Also haben wir uns entschieden, helle Farben auch auf dem Papier hell zu machen. Natürlich können dabei auch unglückliche Bilder entstehen. Abb. 3.3.1-3 ist ein Beispiel hierfür. Sollten Sie anderer Ansicht sein, drehen Sie im Programm einfach die Reihenfolge der Masken herum. Bei Abb. 3.2-4 haben wir ein wenig gepfuscht. Die Farbsättigung für das Foto war eine andere als für die Hardcopy.

Natürlich hat man mit dieser Frage im Modus 2 nichts zu tun. Hier gibt es nur schwarz auf weiß. Ein schönes Beispiel finden Sie in Abb. 3.3.1-1.

Nun zu dem Programm. In der vorliegenden Form läuft es auf einem Drucker EPSON JX-80. Das ist 'bunte' Version des bekannten FX-80. Er besitzt ein breites Farbband, auf dem die drei Grundfarben und Schwarz in schmalen Streifen übereinander angeordnet sind. Im direkten Zugriff stehen sieben Farben, wobei der Drucker bedarfsweise die Farben selbsttätig mischt. Auf diese Farben haben wir uns auch beschränkt.

Zum Farbwechsel bewegt ein Motor die gewünschte Farbe vor den Druckkopf. Diesem Vorgang trägt auch das Programm Rechnung. Jede Zeile auf dem Bildschirm wird auf eine

Abb. 3.3.1-1

bestimmte Farbe hin untersucht und dann die gesamte Zeile in dieser Farbe ausgedruckt. Ein Farbwechsel von Fall zu Fall innerhalb der Zeile wäre viel zu zeitraubend.
Kommt die gesuchte Farbe nicht vor, wird auch kein Farbwechsel veranlaßt. Dadurch wird bei nur wenigen Farben viel Zeit gespart.
Ein wenig Geduld müssen Sie jedoch aufbringen, wenn im Modus 0 auch alle 16 Farben reichlich verwendet wurden. Dann kann eine solche Hardcopy leicht eine halbe Stunde dauern wie z.B. bei Abb. 3.3.1-5.

Ab der nächsten Seite finden Sie das Assemblerlisting der Druckerhardcopy. Einige Bemerkungen finden Sie noch im Anschluß daran.

```
1         org     $cba
2    gemdos equ     1
3    xbios  equ     14
4    prchar equ     5
5    sbase  equ     2
6    getres equ     4
7    aff    equ     -2        anz farben
8    afc    equ     -4        farb cnt
9    pwf    equ     -6        worte/pixel
10   hmf    equ     -8        hor multipl
11   vmf    equ     -10       vert multipl
12   zbl    equ     -14       zeilenbasis
13   zwf    equ     -16       anz worte/zeile
14   zwc    equ     -18       anz worte cnt
15   znf    equ     -20       anz nadeln/zeile
16   znc    equ     -22       anz nadeln cnt
17   baf    equ     -24       vert abstand
18   zzc    equ     -26       zeilen cnt
19   zol    equ     -30       zeilenoffset
20   ab     equ     -31       gerade gef bit
21   fl     equ     -32       div flags
22   *      bit     0         bit entspr farbe gefunden
23   *      bit     1         0=test / 1=druck
24   ctf    equ     -48       farbtafel
25   maf    equ     -64       masken nr
26   pflag  equ     $4ee      flag alt/help
27   super  equ     32        supervisor mode
28   stcol  equ     7         setcolor
60
61
62   ********************************************************************
63   *                                                                  *
64   *       programm hinter den videoram verschieben                   *
65   *                                                                  *
66   ********************************************************************
67
30            clr.l   -(a7)
31            move.w  #super,-(a7)
```

```
32          trap    #gemdos
33          addq.l  #6,a7
34          move.l  d0,d6
35          move.w  #sbase,-(a7)
36          trap    #xbios
37          addq.l  #2,a7
38          movea.l d0,a0
39          adda.w  #$7d00,a0
40          lea     (a0),a2
41          lea     start(pc),a1
42          move.l  #fin-start-1,d0
43  reloc   move.b  (a1)+,(a0)+
44          dbra    d0,reloc
45
46          movea.l $456,a0
47          adda    #28,a0
48          move.l  a2,(a0)
49          move.l  d6,-(a7)
50          move.w  #super,-(a7)
51          trap    #gemdos
52          addq.l  #6,a7
53          clr     -(a7)
54  *       rts                     falls von basic aufgerufen
55          trap    #gemdos
56  start:
57          tst     pflag           hardcopy gewünscht ?
58          beq     st0             ja >
59          rts
60
61
62  ************************************************************************
63  *                                                                      *
64  *       parameter initialisieren                                       *
65  *                                                                      *
66  ************************************************************************
67
68  st0     link    a6,#-66         platz für arbeitsbereich schaffen
69          move.w  #sbase,-(a7)    physikalische
70          trap    #xbios          bildschirmbasis holen
```

```
71          addq.l  #2,a7
72          move.l  d0,zbl(a6)
73          move.w  #getres,-(a7)
74          trap    #xbios
75          addq.l  #2,a7
76          lsl.w   #1,d0
77          move.w  d0,(a6)
78          lea     aft(pc),a1
79          move.w  0(a1,d0.w),aff(a6)
80          moveq   #1,d7           farbnr. und maske vorbereiten,
81          moveq   #8,d0           falls hi-res
82          move.b  #7,ctf(a6)
83          clr.b   ctf+1(a6)
84          cmpi.w  #1,aff(a6)      hi-res ?
85          beq     st52            ja >
86          move.w  aff(a6),d7
87  st1     move.w  #-1,-(a7)
88          move.w  d7,-(a7)
89          move.w  #stcol,-(a7)
90          trap    #xbios          farbe nach d0
91          addq.l  #6,a7
92          clr.w   d4
93          clr.b   maf(a6,d7.w)
94          move.w  d0,d1
95          moveq   #2,d5
96          lsl.w   #4,d1
97  st10    lsr.b   #4,d1
98          or.b    d1,d4           helligkeit > d4
99          lsr.w   #4,d1
100         dbra    d5,st10
101
102         move.b  d4,maf(a6,d7.w)
103         cmpi.b  #1,d4           schwarz-schwelle unterschritten ?
104         bls     st22            ja >
105         moveq   #2,d6
106         move    #$444,d5        maske laden
107 st11    move    d5,d3
108         and     d0,d3           das höchste bit suchen
109         bne     st12            gefunden >
```

```
110             lsr     #1,d5           maske heruntersetzen
111             dbra    d6,st11
112
113     st12    moveq   #2,d4
114             clr.w   d5
115     st2     andi    #$7ff,d3        farbe feststellen (nach d5)
116             cmpi.w  #$ff,d3
117             bls     st21
118             bset.l  d4,d5
119     st21    lsl.w   #4,d3
120             dbra    d4,st2
121
122             cmpi.b  #7,d5           weiss ?
123             bne     st5             nein >
124             cmpi.b  #5,maf(a6,d7)   rein weiss ?
125             bhi     st5             ja >
126             addq.b  #2,maf(a6,d7)   maske abmagern und
127     st22    clr     d5              schwarz setzen
128     st5     move.b  d5,ctf(a6,d7.w)
129             cmpi.b  #6,d5           gelb ?
130             bne     st50            nein >
131             subq.b  #2,maf(a6,d7)   fettere maske
132     st50    dbra    d7,st1
133
134             moveq   #15,d7
135     st51    moveq   #8,d0
136             cmpi.b  #3,maf(a6,d7)   helligkeit > untere schwelle ?
137             bls     st52            nein >
138             lsr     #1,d0
139             cmpi.b  #6,maf(a6,d7.w) helligkeit > obere schwelle ?
140             bls     st52            nein >
141             clr     d0
142     st52    move.b  d0,maf(a6,d7.w)
143             dbra    d7,st51
144
145             move.w  (a6),d0
146             lea     pwt(pc),a1
147             move.w  0(a1,d0.w),pwf(a6)
148             lea     hmt(pc),a1
```

```
149          move.w  0(a1,d0.w),hmf(a6)
150          lea     vmt(pc),a1
151          move.w  0(a1,d0.w),vmf(a6)
152          lea     zwt(pc),a1
153          move.w  0(a1,d0.w),zwf(a6)
154          lea     znt(pc),a1
155          move.w  0(a1,d0.w),znf(a6)
156          lea     bat(pc),a1
157          move.w  0(a1,d0.w),baf(a6)
158          move.w  #50,zzc(a6)
159          clr.b   fl(a6)
160          bra     nl0
161
162
163     ************************************************************************
164     *                                                                      *
165     *       nächste zeile                                                  *
166     *                                                                      *
167     ************************************************************************
168
169     nl:
170          subq.w  #1,zzc(a6)      zeilenzähler abgelaufen ?
171          beq     exit            ja >
172          move.l  zbl(a6),d7      zeilenbasis
173          addi.l  #640,d7         um eine zeile
174          move.l  d7,zbl(a6)      erhöhen
175     nl0  lea     lftab(pc),a5    zeilenvorschub
176          moveq   #4,d7           an
177          bsr     lf              drucker
178          move.w  aff(a6),afc(a6) farbzähler
179          movea.l zbl(a6),a3      zeilenbasis
180          bra     sl0
```

```
181
182
183  ********************************************************************
184  *                                                                  *
185  *       nächste farbe                                              *
186  *                                                                  *
187  ********************************************************************
188
189  sl:
190          tst.w   pflag           hardcopy abbrechen ?
191          bne     exit            ja >
192          subq.w  #1,afc(a6)      farbzähler abgelaufen ?
193          bmi     nl              ja > neue zeile
194          bra     sl0
195  sl00    bchg.b  #1,fl(a6)       letzter durchgang nur test ?
196          bne     sl              nein > es wurde gedruckt
197          btst.b  #0,fl(a6)       punkt in der zeile gefunden ?
198          beq     sl00            nein >
199          lea     ctf(a6),a1
200          adda.w  afc(a6),a1
201          clr.w   d6
202          move.b  (a1),d6
203          cmpi.b  #7,d6           weiss ?
204          beq     sl00            ja > kein druck
205          lea     pre1(pc),a5     farbwechsel
206          moveq   #3,d7           an
207          bsr     lf              drucker
208          lea     ct(pc),a1
209          move.b  0(a1,d6.w),d0
210          bsr     chout
211          lea     pre2(pc),a5
212          moveq   #5,d7
213          bsr     lf
214  sl0     move.w  zwf(a6),zwc(a6) anzahl worte/zeile
215          bclr.b  #0,fl(a6)
216          lea     0,a4
217          move.w  afc(a6),d7      gesuchte farbnummer
218          clr.w   d0
219          move.b  maf(a6,d7.w),d0 maske laden
```

```
220         lea     mask(pc),a0
221         move.l  0(a0,d0.w),d2
222         bra     sw0
223
224
225 ********************************************************************
226 *                                                                  *
227 *       nächstes wort                                              *
228 *                                                                  *
229 ********************************************************************
230
231 sw:
232         subq.w  #1,zwc(a6)      wortzähler abgelaufen ?
233         beq     sl00            ja >
234         movea.l zol(a6),a4      zeilenoffset
235         adda.w  pwf(a6),a4      worte/pixel
236         adda.w  pwf(a6),a4      *2
237 sw0     move.w  #$8000,d5       bitmaske far test
238         move.l  a4,zol(a6)      zeilenoffset retten
239         bra     sb0
240
241
242 ********************************************************************
243 *                                                                  *
244 *       nächstes bit                                               *
245 *                                                                  *
246 ********************************************************************
247
248 sb:
249         lsr.w   #1,d5           alle bits im wort fertig ?
250         beq     sw              ja >
251 sb0     move.w  znf(a6),znc(a6) anzahl nadeln/zeile
252         clr.b   d4
253         movea.l zol(a6),a4
254         bra     tb
```

```
255
256
257  ************************************************************************
258  *                                                                      *
259  *       nächste nadel                                                  *
260  *                                                                      *
261  ************************************************************************
262
263  bs:
264          clr.l   d7
265          move.w  vmf(a6),d7      vertikaler multiplikator
266          subq    #1,d7
267  bs0     lsl.b   #1,d4
268          or.b    ab(a6),d4
269          dbra    d7,bs0
270          adda.w  baf(a6),a4      vertikaler abstand der punkte
271          subq.w  #1,znc(a6)      nadelzähler abgelaufen ?
272          bne     tb              nein > punkt testen
273          tst.b   d4              gab es einen punkt ?
274          beq     bs00            nein >
275          bset.b  #0,fl(a6)
276  bs00    btst.b  #1,fl(a6)       soll gedruckt werden ?
277          beq     sb              nein >
278          clr.l   d7
279          move.w  hmf(a6),d7      horizontaler multiplikator
280          subq    #1,d7
281  bs1     move.b  d4,d0
282          and.b   d2,d0           byte maskieren
283          bsr     chout           und ausgeben
284          ror.l   #8,d2           rastermaske rotieren
285          dbra    d7,bs1
286          bra     sb
```

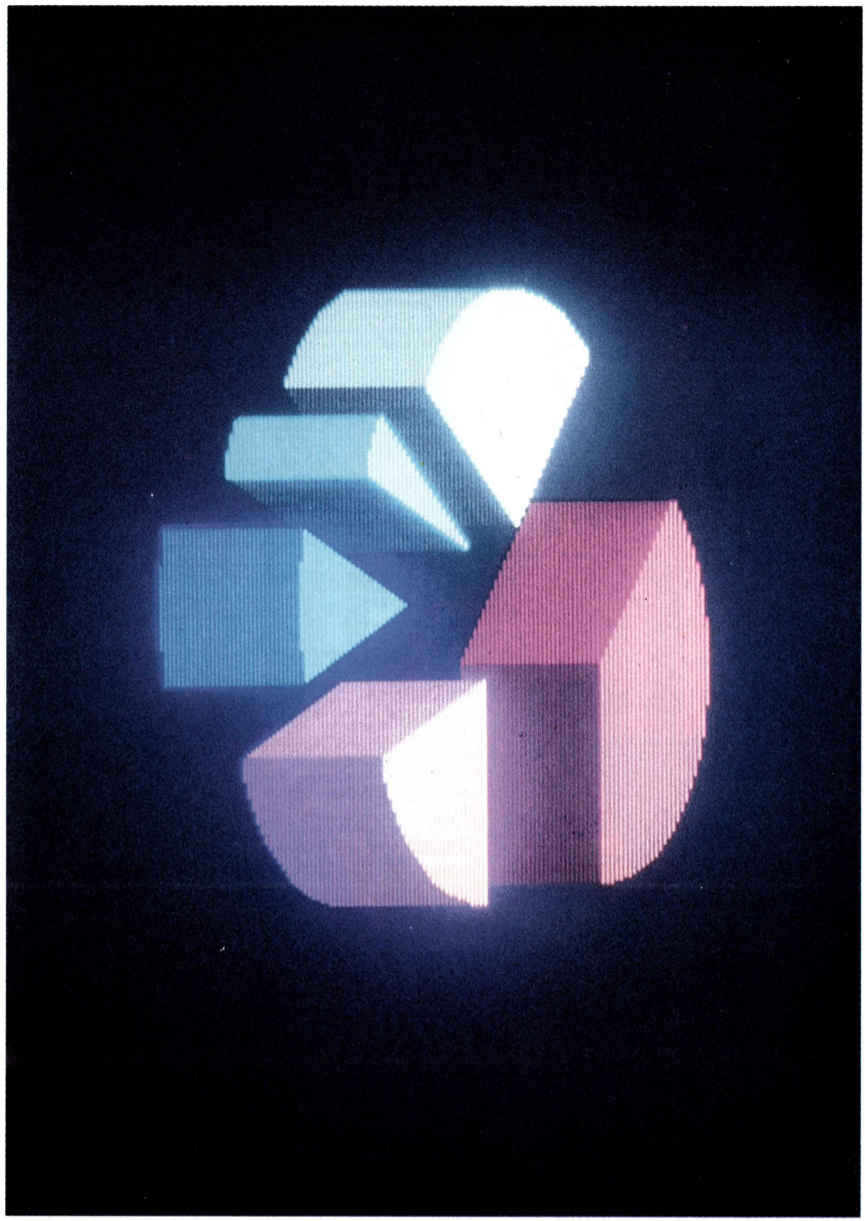


Abb. 3.1-1

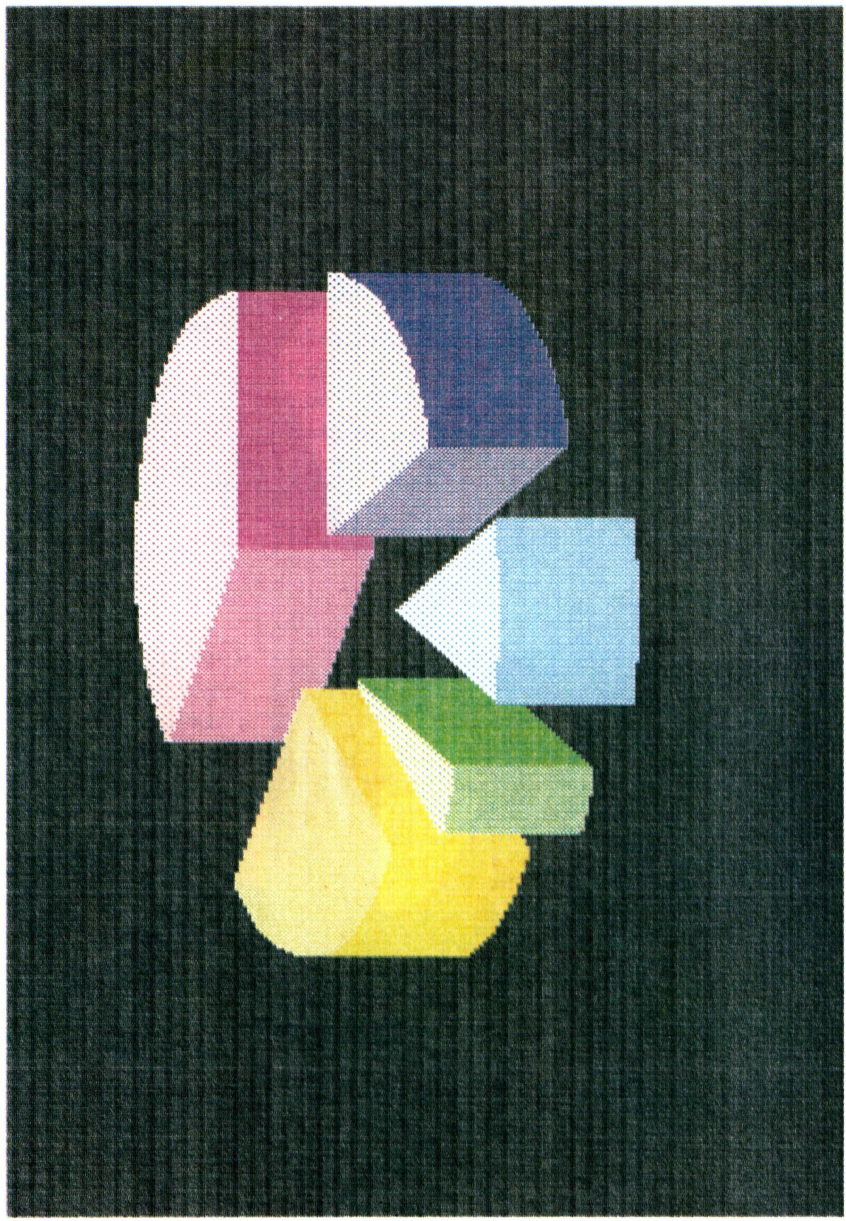


Abb. 3.1-2

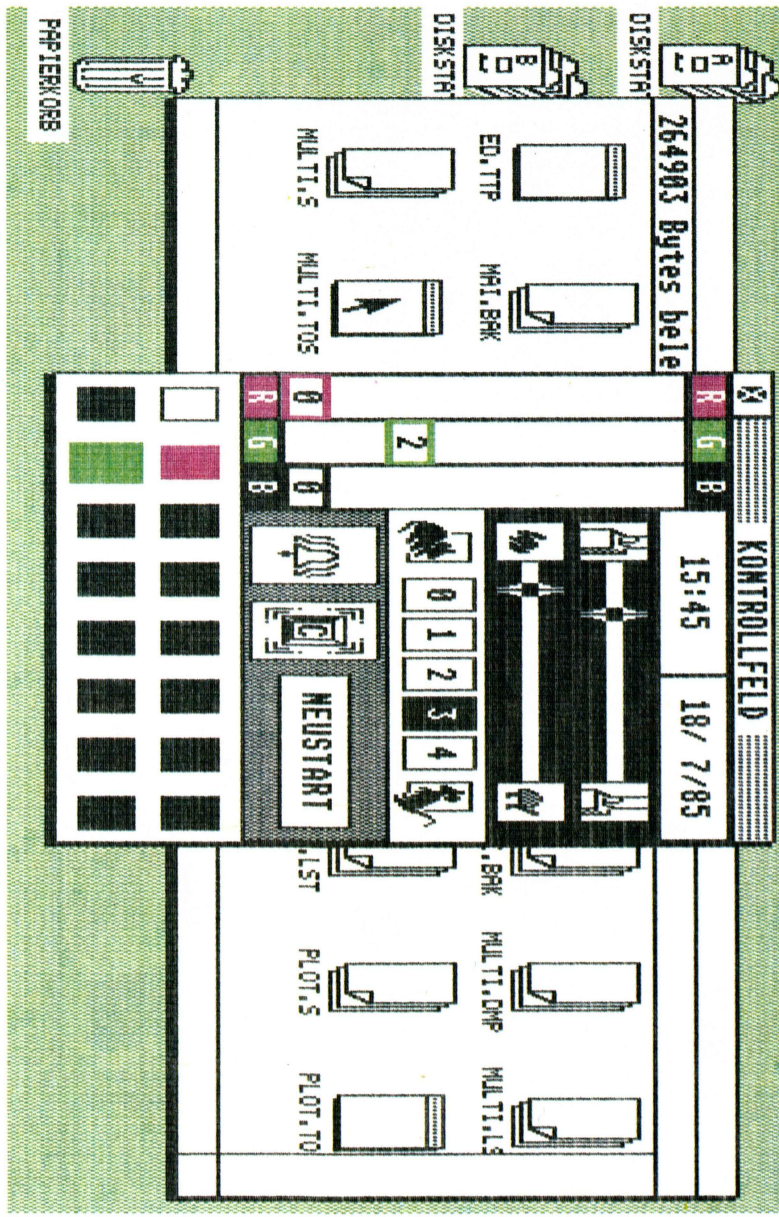

Desk-Info Datei Anzeigen Optionen
KONTROLLFELD
15:45
18/ 7/85
264903 Bytes bele
DISKSTA
DISKSTA
PAPIERKORB
ED.TTP
MAI.BAK
MULTI.S
MULTI.TOS
MULTI.OMP
PLOT.S
PLOT.TO
.LST
.BAK
NEUSTART
R
G
B

Abb. 3.2-4

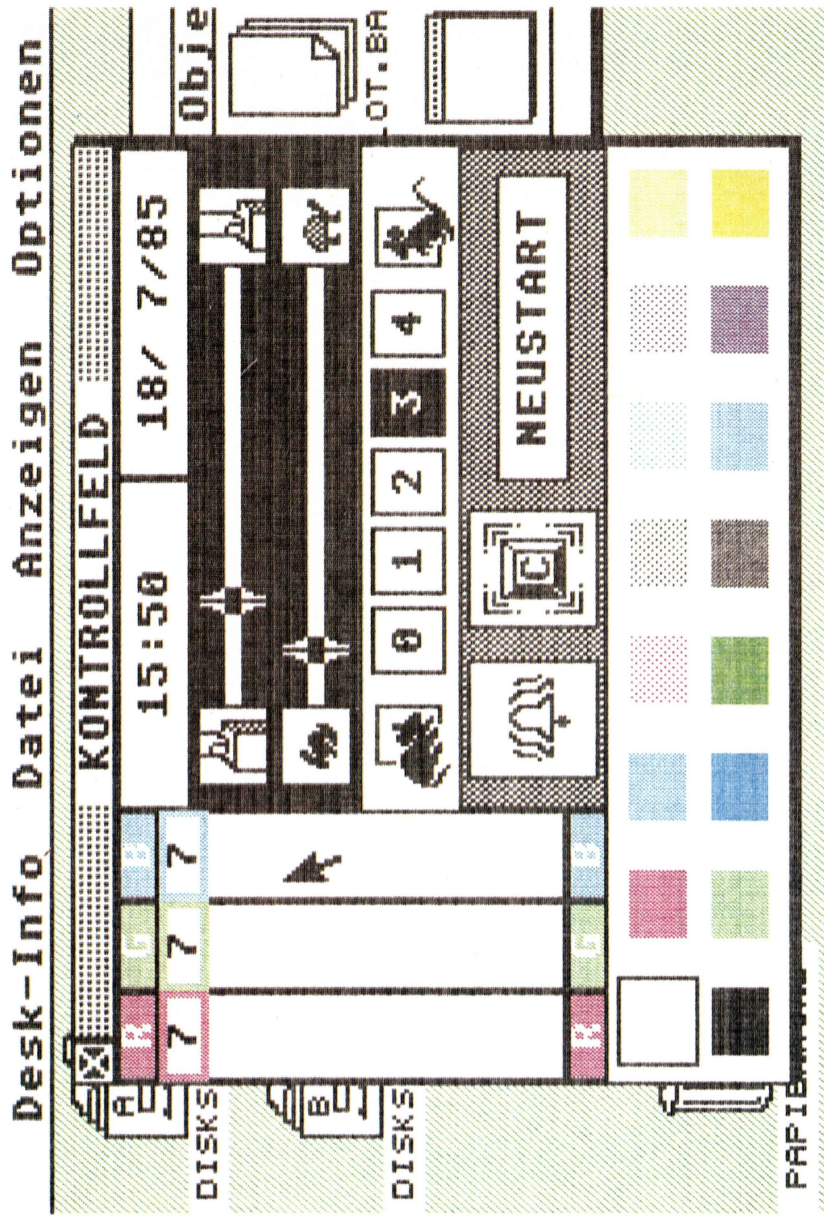

Desk-Info Datei Anzeigen Optionen
KONTROLLFELD
15:50
18/ 7/85
Objekt
0 1 2 3 4
NEUSTART
DISKS
DISKS

Abb. 3.2-6

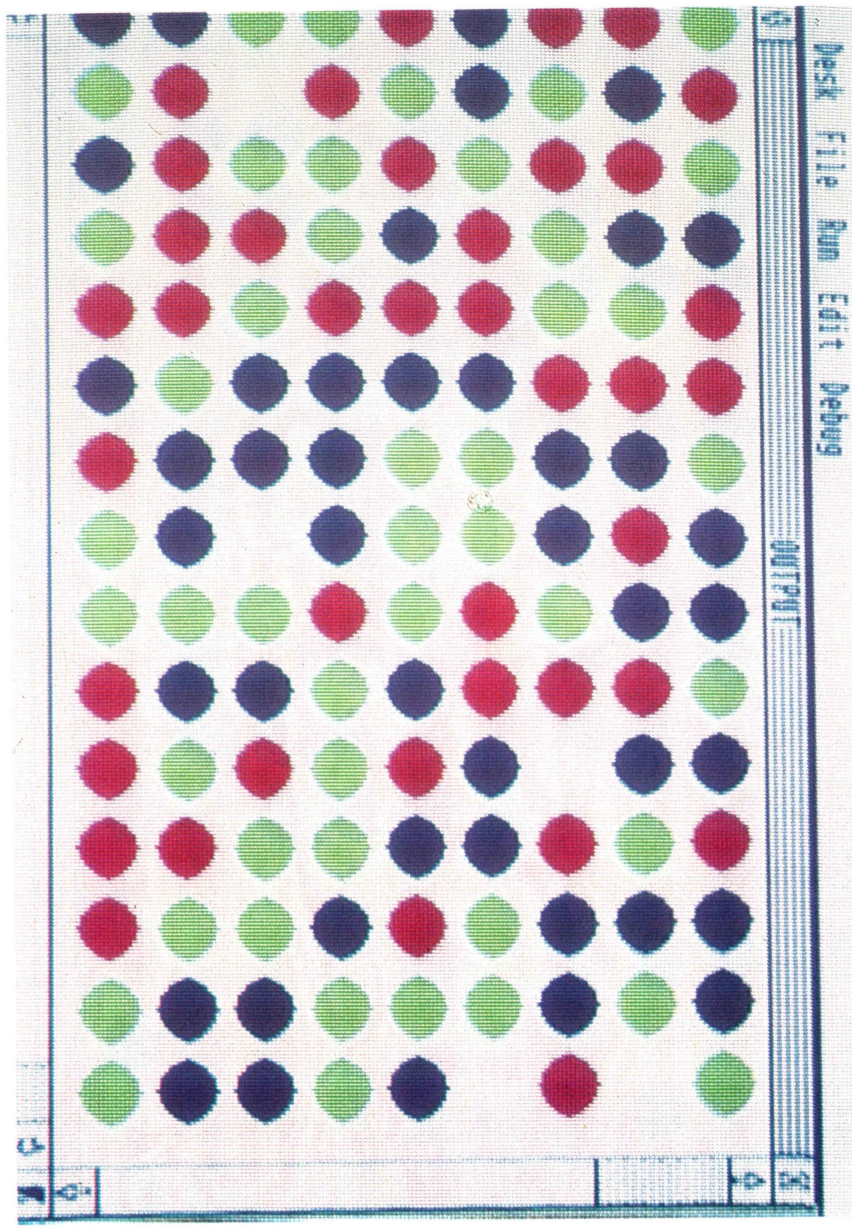


Abb. 3.3.1-2

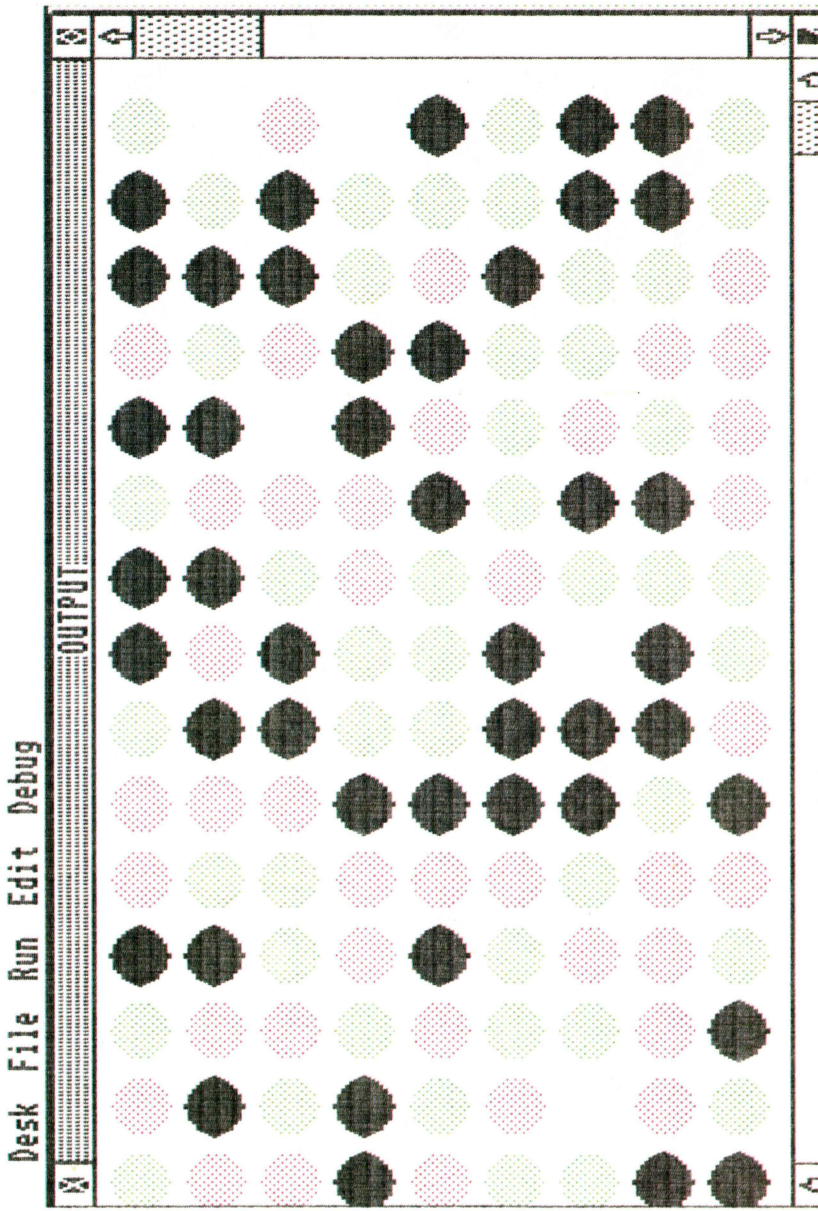


Abb. 3.3.1-3

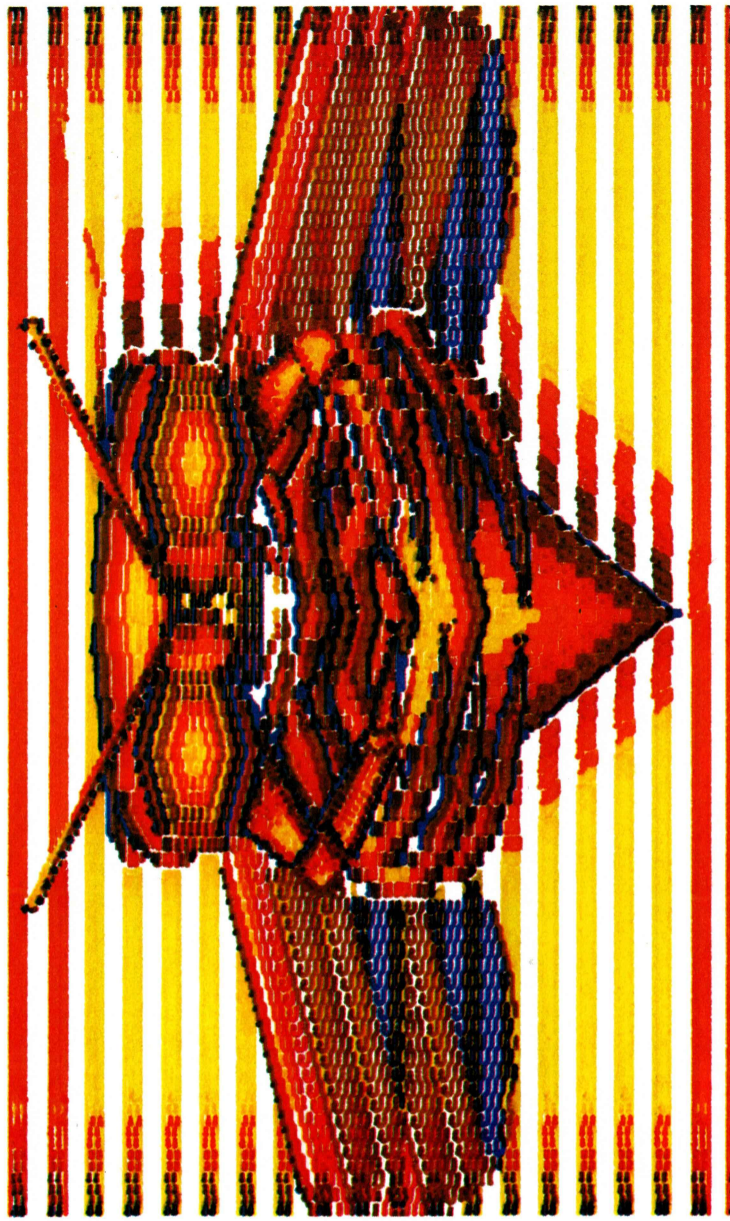


Abb. 3.3.1-4

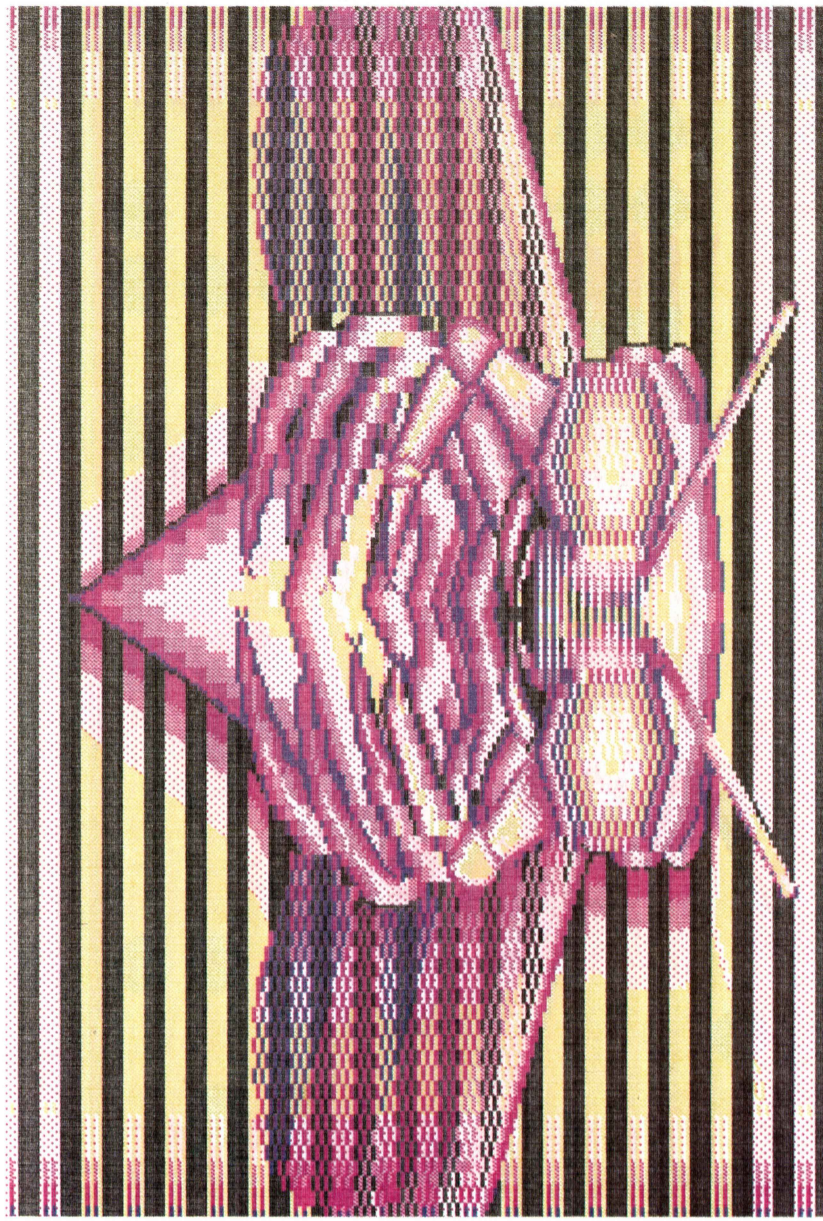


Abb. 3.3.1-5

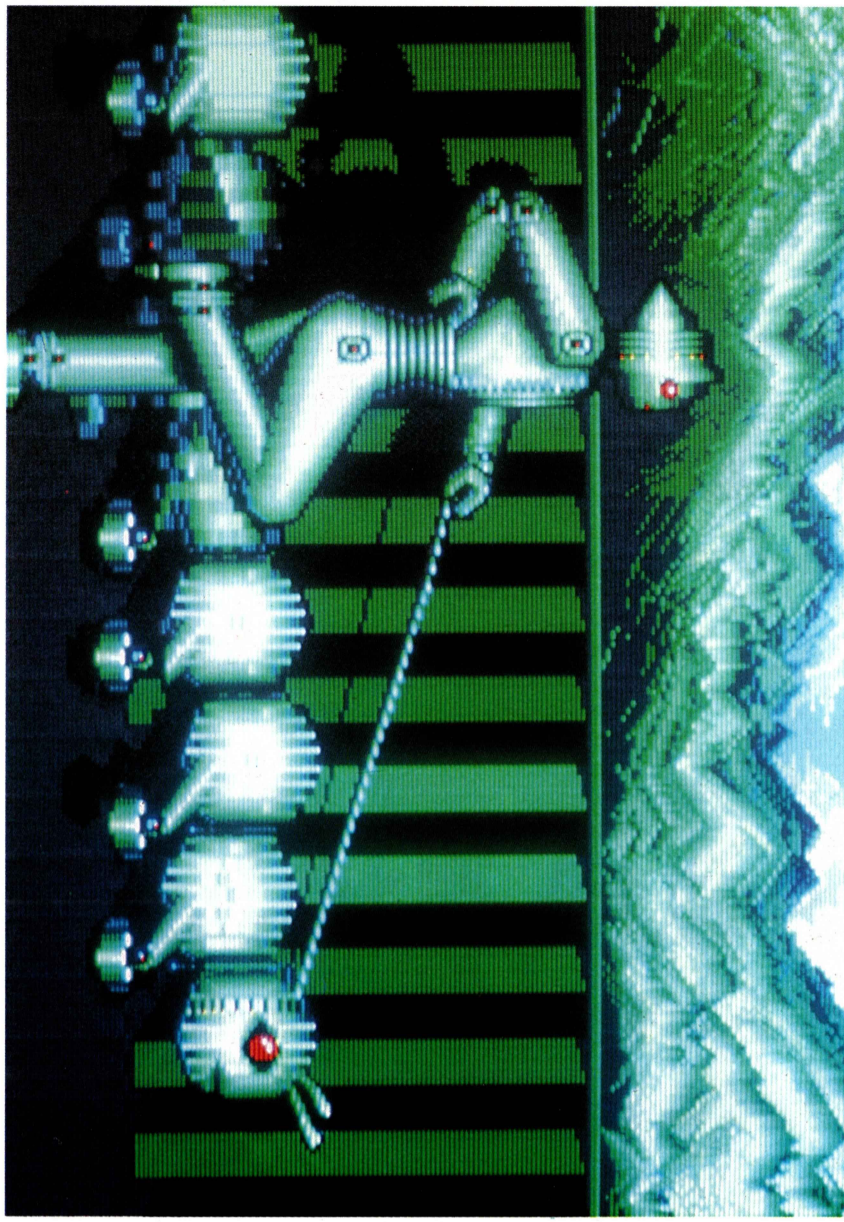


Abb. 3.3.1-6

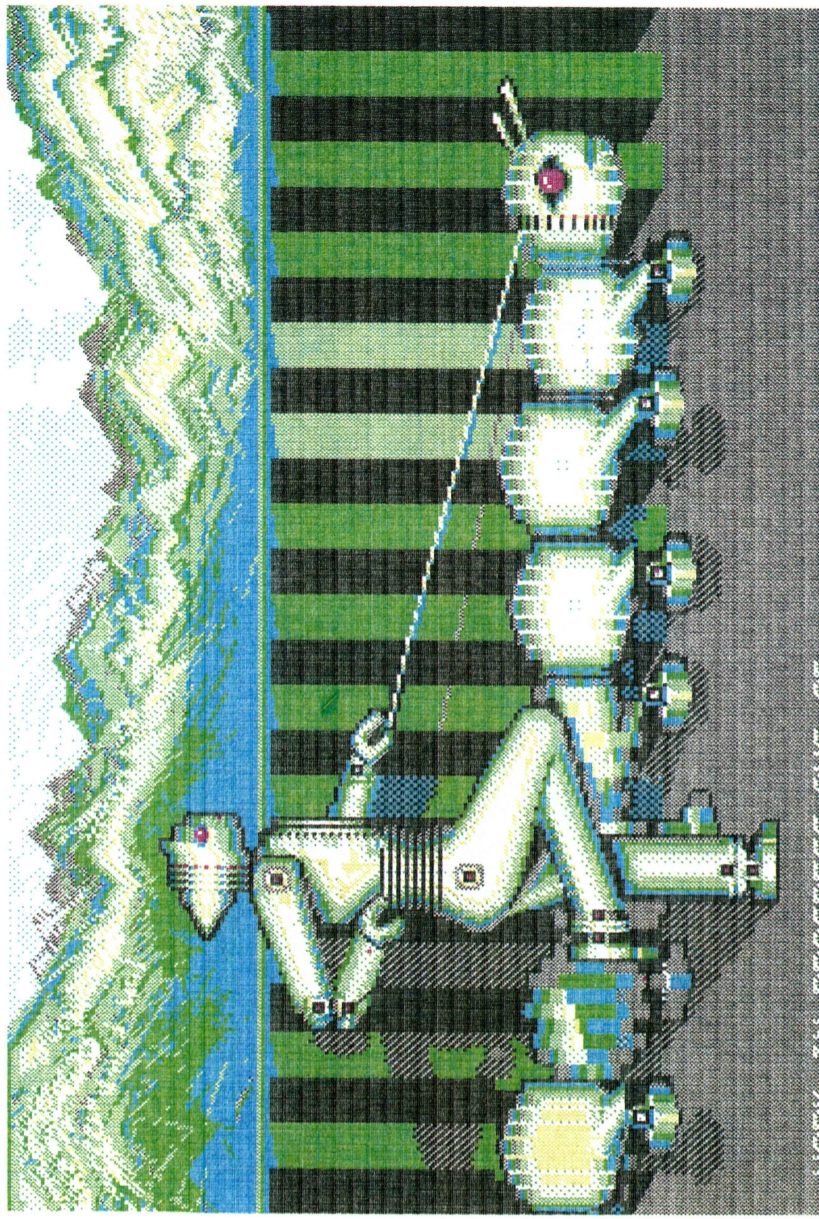


Abb. 3.3.1-7

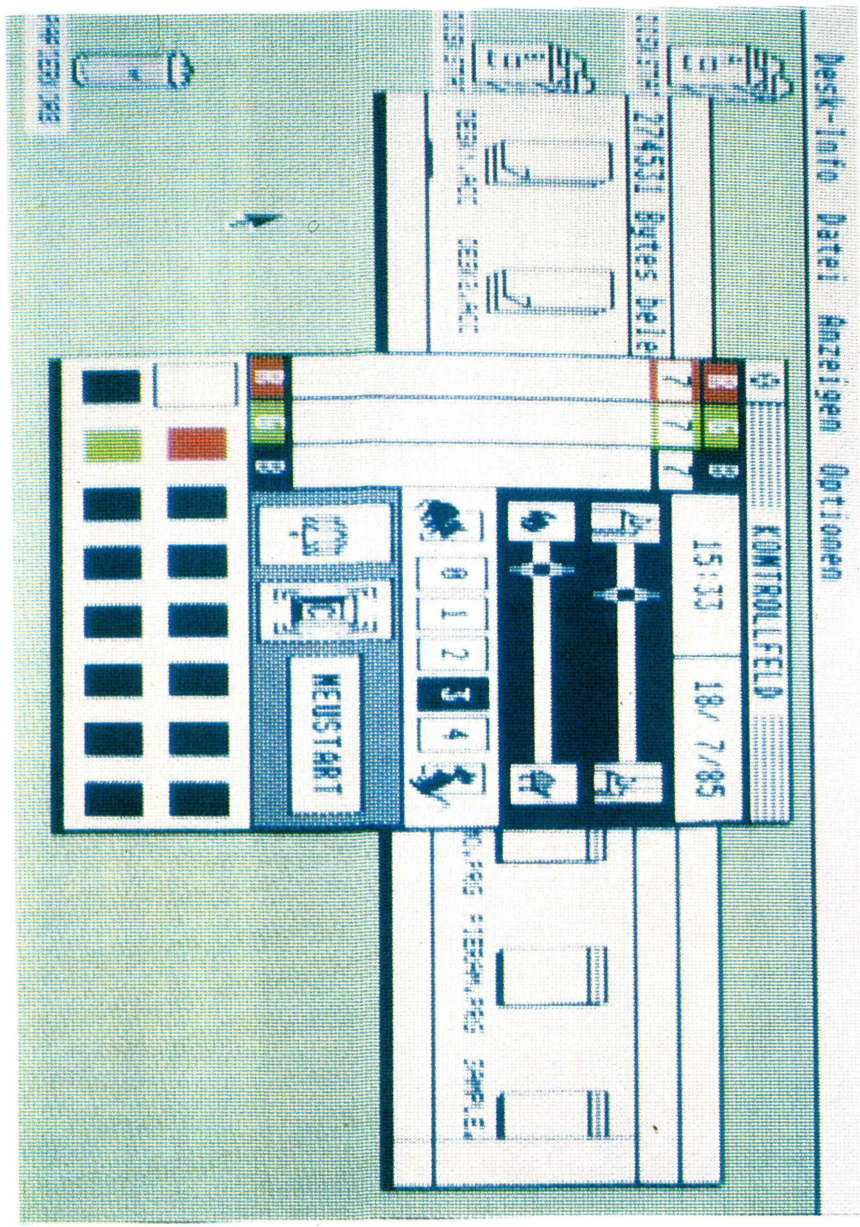

Mask-info Datei Anzeigen Optionen
KONTROLLFELD
15:33
18/ 7/85
NEUSTART

Abb. 3.3.2-5

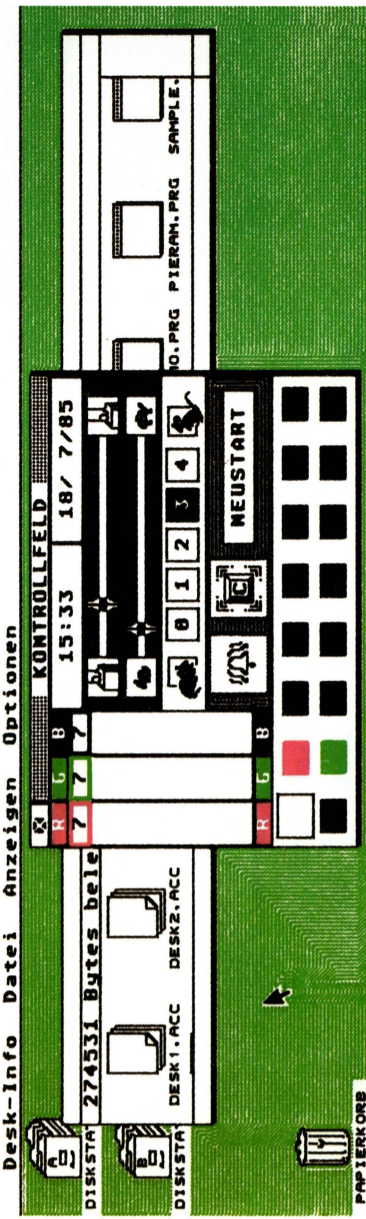

Desk-Info Datei Anzeigen Optionen
KONTROLLFELD
15:33
18/ 7/85
274531 Bytes bele
DESK1.ACC
DESK2.ACC
NEUSTART
DISKSTA
DISKSTA
PAPIERKORB
SAMPLE.
PIERAM.PRG
O.PRG

Abb. 3.3.2-6

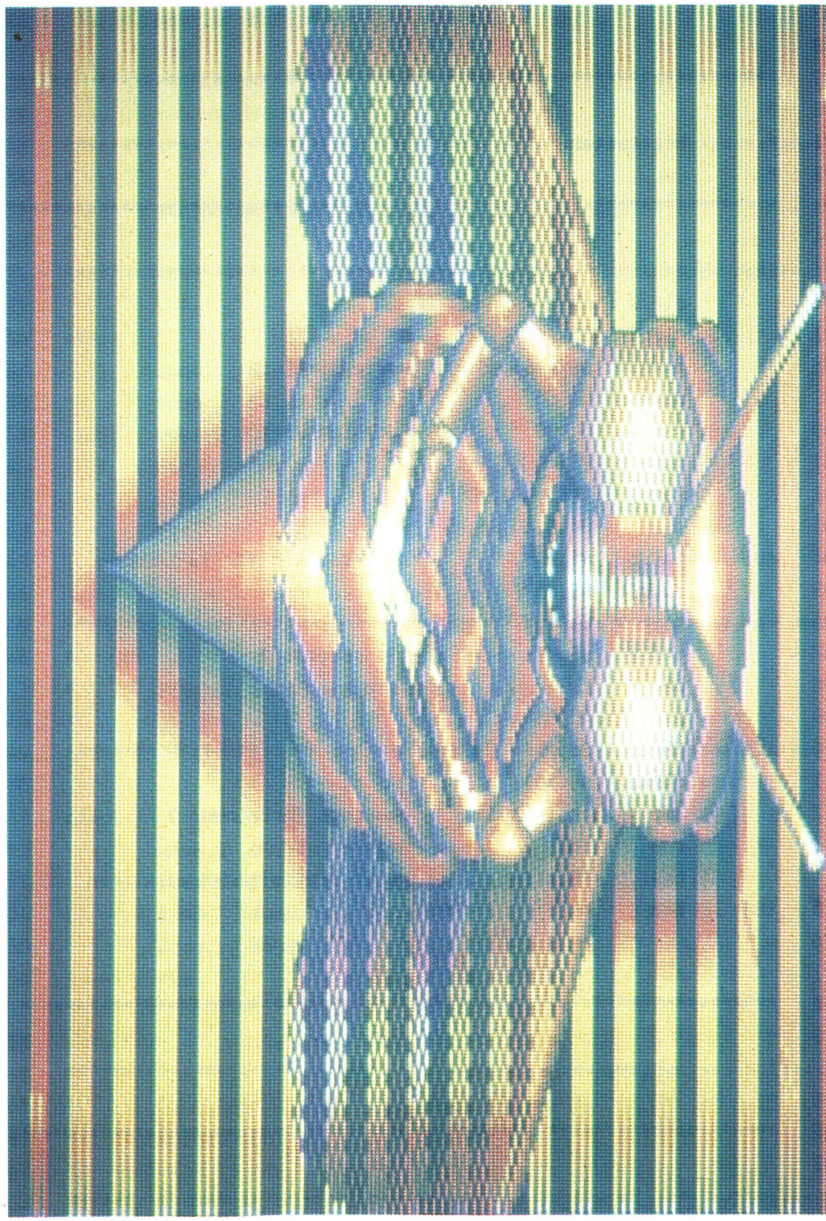


Abb. 3.3.2-7

```
287
288
289  ************************************************************************
290  *                                                                      *
291  *       bit testen                                                     *
292  *                                                                      *
293  ************************************************************************
294
295  tb:
296          clr.w   d3
297          clr.l   d6
298          move.w  pwf(a6),d6      worte/pixel
299          move.w  d6,d0
300          lsl.w   #1,d0           nächstes wort
301          subq.b  #1,d6
302          lea     0(a3,a4),a0
303          lea     0(a0,d0.w),a5
304  tb1     lsl.b   #1,d3           bits sammeln für farbnummer
305          subq.l  #2,a5
306          move.w  (a5),d7
307          and.w   d5,d7           bit gesetzt ?
308          beq     tb2             nein >
309          bset.l  #0,d3
310  tb2     dbra    d6,tb1
311          clr.b   ab(a6)
312          cmp.w   afc(a6),d3      gesuchte farbnummer ?
313          bne     tb3             nein >
314          bset.b  #0,ab(a6)       punkt als gefunden markieren
315  tb3     bra     bs
316
317
318  ************************************************************************
319  *                                                                      *
320  *       ausgang                                                        *
321  *                                                                      *
322  ************************************************************************
323
324  exit:
325          unlk    a6              arbeitsbereich freigeben
```

```
326         move.w  #-1,pflag       hardcopy fertig
327         rts
328
329
330 ********************************************************************
331 *                                                                  *
332 *       string an (a5) mit zähler in d7 ausgeben                   *
333 *                                                                  *
334 ********************************************************************
335
336 lf:
337         andi.l  #$ffff,d7
338         subq    #1,d7
339 lf0     move.b  0(a5,d7),d0
340         bsr     chout
341         dbra    d7,lf0
342         rts
343
344
345 ********************************************************************
346 *                                                                  *
347 *       zeichen in d0 an drucker                                   *
348 *                                                                  *
349 ********************************************************************
350
351 chout:
352         move.w  d0,-(a7)
353         move.w  #prchar,-(a7)
354         trap    #gemdos
355         addq.l  #4,a7
356         rts
357
358
359 ********************************************************************
360 *                                                                  *
361 *       konstanten                                                 *
362 *                                                                  *
363 ********************************************************************
364
```

```
365   aft     dc.w    15,3,1          anzahl farben
366   pwt     dc.w    4,2,1           zu einem pixel gehörende worte
367   hmt     dc.w    2,1,1           horizontale verdopplung
368   vmt     dc.w    2,2,1           vertikale verdopplung
369   zwt     dc.w    20,40,40        worte/zeile
370   znt     dc.w    4,4,8           nadeln/zeile
371   bat     dc.w    160,160,80      vertikaler abstand der zeilen
372   mask    dc.l    $44001100       farbe schwach
373           dc.l    $aa55aa55       mittel
374           dc.l    -1              voll
375   ct      dc.b    0,2,6,2,1,3,4,0 druckerfarben
376   lftab   dc.b    24,"J",27,13    zeilenvorschub 8 nadeln
377   pre1    dc.b    "r",27,13       farbauswahl
378   pre2    dc.b    2,128,4,"*",27  grafikmodus und punktzähler
379   fin     equ     *
380           .end
```

Eine Eingriffsmöglichkeit zur Anpassung an andere Drucker bietet sich in den Zeilen 375 bis 378. Hier können Sie die druckerspezifische Steuersequenz eingeben, aber bitte, wie Sie auch hier sehen, von hinten nach vorne, weil die Stringausgabe mit dem letzten Byte beginnt (Zähler gleich Pointer).

Sollten Sie nicht über einen Assembler verfügen und Änderungen 'zu Fuß' in dem BASIC-Programm vornehmen wollen, schauen Sie unbedingt darauf, daß die Länge und Lage der Strings nicht verändert wird, es sei denn, daß Sie die Referenzadressen ebenfalls anpassen.
Das Maschinenprogramm für das Laden von BASIC aus weist einen kleinen Unterschied gegenüber der Assemblerversion auf. Da das Programm ja mit CALL aufgerufen wird, muß es mit RTS abgeschlossen sein und nicht mit TERM über GEMDOS

```
10    dim a%(415)
20    for i=0 to 415
30    read a%(i)
40    next i
50    b=varptr(a%(0))
60    call b
70    end
950   data &H42A7,&H3F3C,&H0020
960   data &H4E41,&H5C8F,&H2C00,&H3F3C,&H0002,&H4E4E,&H548F,&H2040
970   data &HD0FC,&H7D00,&H45D0,&H43FA,&H0028,&H203C,&H0000,&H02F9
980   data &H10D9,&H51C8,&HFFFC,&H2079,&H0000,&H0456,&HD0FC,&H001C
990   data &H208A,&H2F06,&H3F3C,&H0020,&H4E41,&H5C8F,&H4E75,&H4E41
1000  data &H4A79,&H0000,&H04EE,&H6702,&H4E75,&H4E56,&HFFBE,&H3F3C
1010  data &H0002,&H4E4E,&H548F,&H2D40,&HFFF2,&H3F3C,&H0004,&H4E4E
1020  data &H548F,&HE348,&H3C80,&H43FA,&H0288,&H3D71,&H0000,&HFFFE
1030  data &H7E01,&H7008,&H1D7C,&H0007,&HFFD0,&H422E,&HFFD1,&H0C6E
1040  data &H0001,&HFFFE,&H6700,&H009A,&H3E2E,&HFFFE,&H3F3C,&HFFFF
1050  data &H3F07,&H3F3C,&H0007,&H4E4E,&H5C8F,&H4244,&H4236,&H70C0
1060  data &H3200,&H7A02,&HE949,&HE809,&H8801,&HE849,&H51CD,&HFFF8
1070  data &H1D84,&H70C0,&H0C04,&H0001,&H633A,&H7C02,&H3A3C,&H0444
1080  data &H3605,&HC640,&H6606,&HE24D,&H51CE,&HFFF6,&H7802,&H4245
1090  data &H0243,&H07FF,&H0C43,&H00FF,&H6302,&H09C5,&HE94B,&H51CC
1100  data &HFFF0,&H0C05,&H0007,&H660E,&H0C36,&H0005,&H70C0,&H6206
1110  data &H5436,&H70C0,&H4245,&H1D85,&H70D0,&H0C05,&H0006,&H6604
1120  data &H5536,&H70C0,&H51CF,&HFF86,&H7E0F,&H7008,&H0C36,&H0003
1130  data &H70C0,&H630C,&HE248,&H0C36,&H0006,&H70C0,&H6302,&H4240
1140  data &H1D80,&H70C0,&H51CF,&HFFE4,&H3016,&H43FA,&H01CA,&H3D71
1150  data &H0000,&HFFFA,&H43FA,&H01C6,&H3D71,&H0000,&HFFF8,&H43FA
1160  data &H01C2,&H3D71,&H0000,&HFFF6,&H43FA,&H01BE,&H3D71,&H0000
1170  data &HFFF0,&H43FA,&H01BA,&H3D71,&H0000,&HFFEC,&H43FA,&H01B6
1180  data &H3D71,&H0000,&HFFE8,&H3D7C,&H0032,&HFFE6,&H422E,&HFFE0
1190  data &H6016,&H536E,&HFFE6,&H6700,&H014C,&H2E2E,&HFFF2,&H0687
1200  data &H0000,&H0280,&H2D47,&HFFF2,&H4BFA,&H01A4,&H7E04,&H6100
1210  data &H0140,&H3D6E,&HFFFE,&HFFFC,&H266E,&HFFF2,&H6054,&H4A79
1220  data &H0000,&H04EE,&H6600,&H011E,&H536E,&HFFFC,&H6BC4,&H6042
1230  data &H086E,&H0001,&HFFE0,&H66E6,&H082E,&H0000,&HFFE0,&H67F0
1240  data &H43EE,&HFFD0,&HD2EE,&HFFFC,&H4246,&H1C11,&H0C06,&H0007
1250  data &H67DE,&H4BFA,&H015E,&H7E03,&H6100,&H00F6,&H43FA,&H0148
```

```
1260  data &H1031,&H6000,&H6100,&H00FE,&H4BFA,&H014B,&H7E05,&H6100
1270  data &H00E0,&H3D6E,&HFFF0,&HFFEE,&H08AE,&H0000,&HFFE0,&H49F9
1280  data &H0000,&H0000,&H3E2E,&HFFFC,&H4240,&H1036,&H70C0,&H41FA
1290  data &H010A,&H2430,&H0000,&H6012,&H536E,&HFFEE,&H6792,&H286E
1300  data &HFFE2,&HD8EE,&HFFFA,&HD8EE,&HFFFA,&H3A3C,&H8000,&H2D4C
1310  data &HFFE2,&H6004,&HE24D,&H67E0,&H3D6E,&HFFEC,&HFFEA,&H4204
1320  data &H286E,&HFFE2,&H6044,&H4287,&H3E2E,&HFFF6,&H5347,&HE30C
1330  data &H882E,&HFFE1,&H51CF,&HFFF8,&HD8EE,&HFFE8,&H536E,&HFFEA
1340  data &H6628,&H4A04,&H6706,&H08EE,&H0000,&HFFE0,&H082E,&H0001
1350  data &HFFE0,&H67C0,&H4287,&H3E2E,&HFFF8,&H5347,&H1004,&HC002
1360  data &H6162,&HE09A,&H51CF,&HFFF6,&H60AA,&H4243,&H4286,&H3C2E
1370  data &HFFFA,&H3006,&HE348,&H5306,&H41F3,&HC000,&H4BF0,&H0000
1380  data &HE30B,&H558D,&H3E15,&HCE45,&H6704,&H08C3,&H0000,&H51CE
1390  data &HFFF0,&H422E,&HFFE1,&HB66E,&HFFFC,&H6606,&H08EE,&H0000
1400  data &HFFE1,&H6082,&H4E5E,&H33FC,&HFFFF,&H0000,&H04EE,&H4E75
1410  data &H0287,&H0000,&HFFFF,&H5347,&H1035,&H7000,&H6106,&H51CF
1420  data &HFFF8,&H4E75,&H3F00,&H3F3C,&H0005,&H4E41,&H588F,&H4E75
1430  data &H000F,&H0003,&H0001,&H0004,&H0002,&H0001,&H0002,&H0001
1440  data &H0001,&H0002,&H0002,&H0001,&H0014,&H0028,&H0028,&H0004
1450  data &H0004,&H0008,&H00A0,&H00A0,&H0050,&H4400,&H1100,&HAA55
1460  data &HAA55,&HFFFF,&HFFFF,&H0002,&H0602,&H0103,&H0400,&H184A
1470  data &H1B0D,&H721B,&H0D02,&H8004,&H2A1B
```

3.3.2 PLOTTER

Eine völlig andere Situation finden wir vor, wenn es darum geht, den Inhalt des Bildschirmes mittels eines Plotters zu Papier zu bringen.
Sicher wäre es möglich, wie beim Drucker Punkt neben Punkt zu setzen, aber dann muß es nicht unbedingt ein Plotter sein. Konstruktionsbedingt ist ein Plotter dazu da, Linien zu ziehen. Wie also muß ein Programm aussehen, das den Plotter auch möglichst Linien ziehen läßt?
Eine Linie auf dem Bildschirm erkennen wir ja nur aufgrund unserer Erfahrung mit ähnlichen Formen als solche. Einem analysierenden Programm (zumal einem recht kurzen) erscheint sie nur als Menge von Punkten. Genau hier ist der Hebel anzusetzen. Es geht also darum, zusammenhängende Punkte zu erkennen und, soweit es geht, als Kurve oder Linie zu zeichnen.

Wir gehen also in unserem Programm so vor, daß wir, wenn ein Punkt gefunden wurde, den Stift absenken und dann eine viertel Drehung nach links schwenken, um zu sehen, ob dort ein Punkt gesetzt ist. Falls nicht, wird in achtel Schritten nach rechts gedreht, um diese Richtungen auf Punkte zu untersuchen. Ein gefundener Punkt bewegt den Plotterstift dorthin, wonach der Vorgang wieder von vorne beginnt. Das Ganze wird so lange fortgeführt, bis rund um den jeweiligen Standort kein angrenzender Punkt zu finden ist. Dieses möglicherweise etwas umständlich erscheinende Verfahren hat die Wirkung, daß bei größeren Objekten zunächst die Konturen abgefahren werden. Das ist wichtig für das spätere Aussehen des fertigen Bildes. An der nicht vollständigen Abb. 3.3.2-2 können Sie diesen Effekt sehr schön beobachten.

Wir wollen nicht verschweigen, daß das Verfahren auch einen Nachteil hat: Ein gefundener Punkt wird, nachdem er entsprechend verarbeitet wurde, vom Bildschirm gelöscht, damit er bei der weiteren Suche nicht erneut erkannt wird. Das Verfahren arbeitet also destruktiv. So können Sie am Bildschirm den Fortschritt der Arbeit bequem verfolgen.

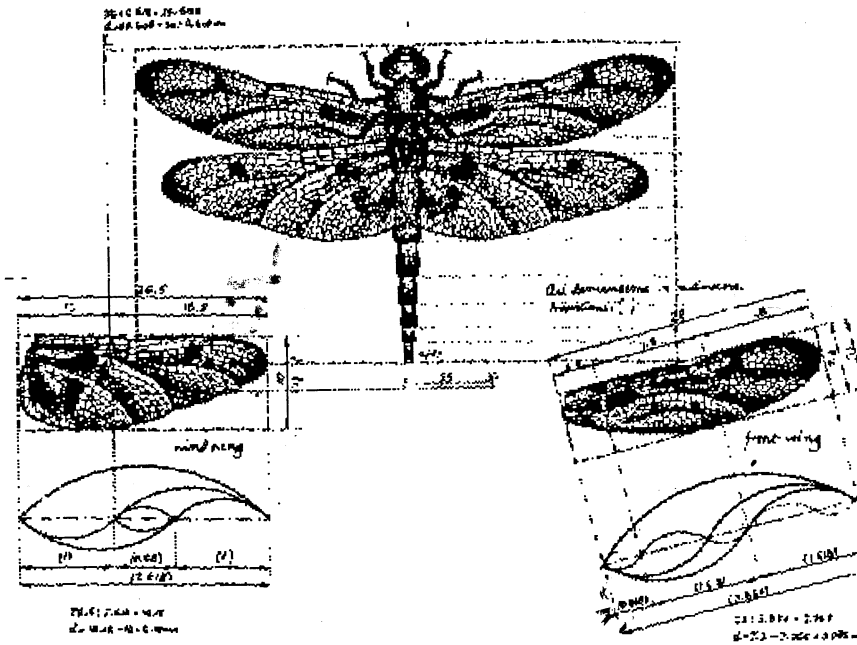


Abb. 3.3.2-1

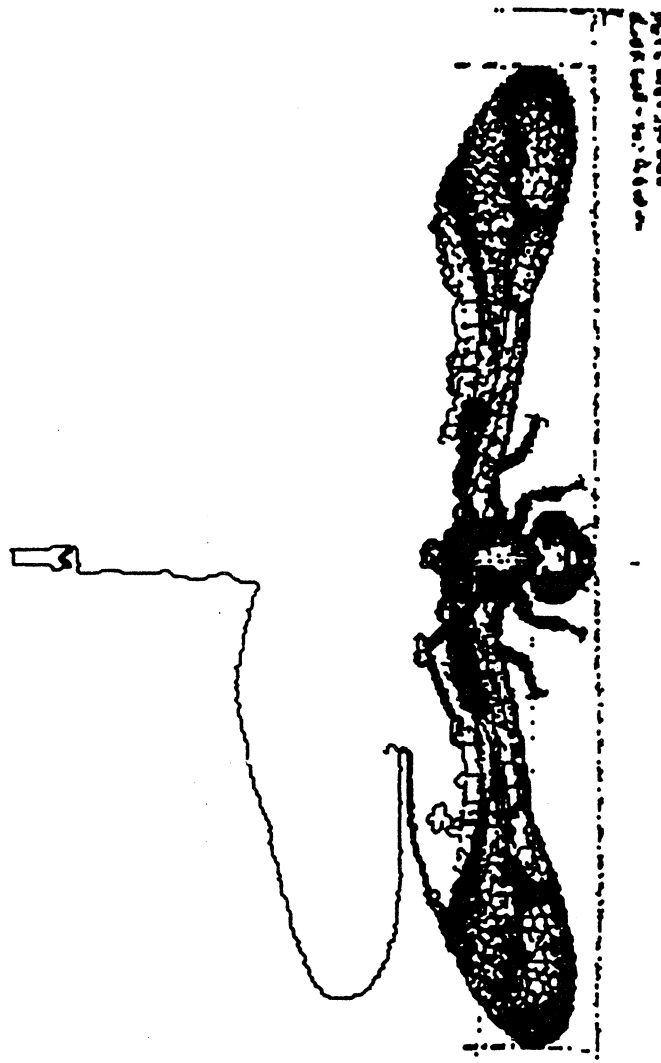


Abb. 3.3.2-2

Wenn die Hardcopy fertig ist, ist der Bildschirm weiß. Sie sollten also peinlich darauf achten, daß ein derart zu Papier zu bringendes Bild kein Unikat ist, sondern möglichst auf Diskette gesichert wurde.

Zur vorliegenden Form des Programmes ist zu sagen, daß es für den Plotter EPSON HI-80 geschrieben wurde. Eine Anpassung an andere Plotter ist kein Problem, da die Kommandosprache vollständig parametrisiert ist
Die Bedienung des Programmes ist gegenüber der Drucker-Hardcopy etwas erweitert. Um Ihnen die Möglichkeit zu geben, bei mehr als vier Farben den Stift zu wechseln, hält der Plotter bei nochmaligem Drücken von ALT/HELP beim nächsten Farbwechsel an. Wenn Sie die Stifte gewechselt haben, drücken Sie erneut ALT/HELP. Das klappt ganz gut, wie Sie an Abb. 3.3.2-7 sehen können. Vergleichen Sie diese Abbildung einmal mit Abb. 3.3.1-5. Sie sehen deutlich die Grenzen eines Plotters bei Vielfarbigkeit.
Im Gegensatz zum Drucker wird hierbei die Farbe des Hintergrundes nie berücksichtigt. Er bleibt immer weiß.

Einen Abbruch der laufenden Farbe erreichen Sie durch dreimaliges Drücken von ALT/HELP.

Eine Besonderheit der Plotter-Hardcopy sehen Sie in Abb. 3.3.2-6. Im Modus 1 ist ja die Vertikale dadurch ein wenig verzerrt, daß ein logischer Punkt zwei Bildpunkten entspricht. Da die Routine aber, anders als bei der Drucker-Hardcopy, keine Punkte verdoppelt, wird das Bild natürlich gestaucht. Sehen Sie hierzu auch die Erläuterungen zum Assemblerlisting.

Hinter den nächsten Abbildungen beginnt das Assemblerlisting. Daran anschließend wieder einige Erläuterungen.

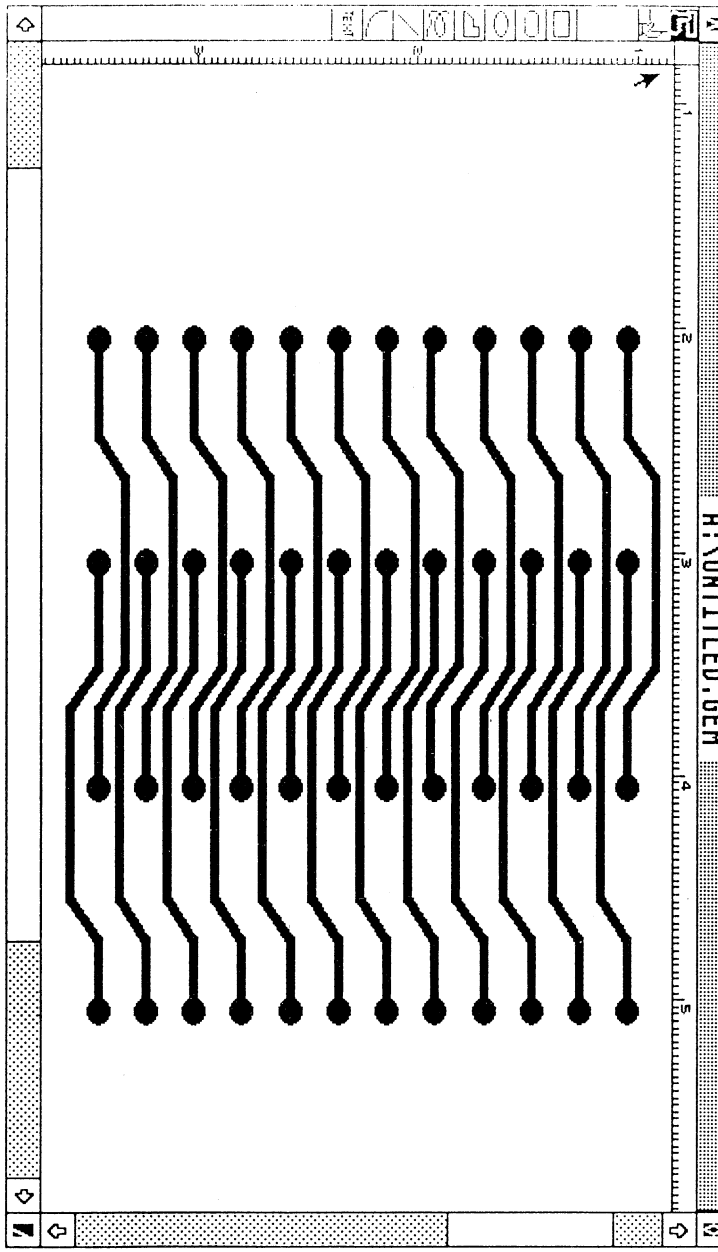

Desk File Edit Font Page Arrange Pattern Line Color
A:\UNTITLED.GEM

Abb. 3.3.2-3

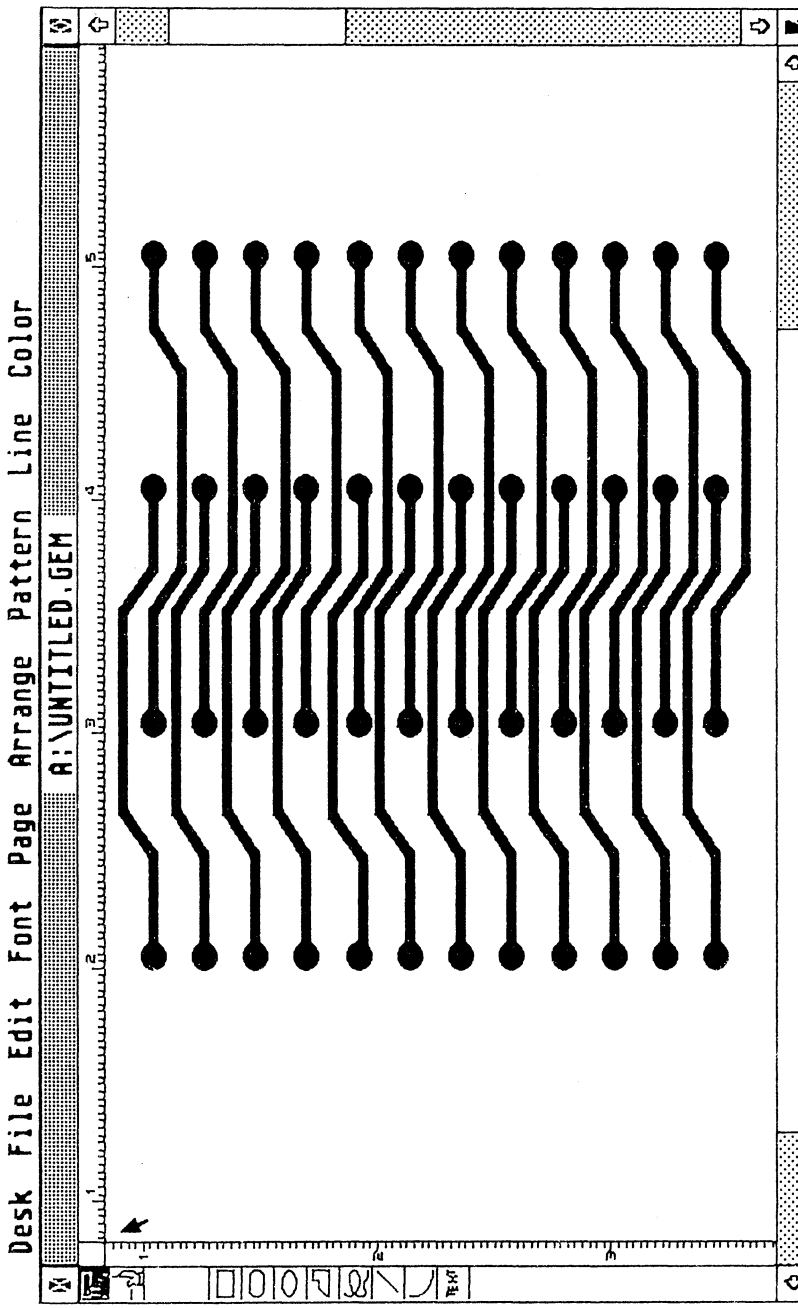

Desk File Edit Font Page Arrange Pattern Line Color
A:\UNTITLED.GEM

Abb. 3.3.2-4

```
1          org     $cba
2   gemdos equ     1
3   bios   equ     13
4   xbios  equ     14
5   bconout equ    3
6   prt    equ     0
7   phybas equ     2
8   setscr equ     5
9   super  equ     32
10  intin  equ     8
11  ptsin  equ     12
12  wrmod  equ     36
13  init   equ     $a000
14  setpix equ     $a001
15  getpix equ     $a002
16  yko    equ     2
17  pflag  equ     $4ee            flag alt/help
18  apix   equ     -4              gesamtzahl pixel
19  pscalx equ     -6              faktor x
20  pscaly equ     -8              faktor y
21  adir   equ     -10             gerade verfolgte richtung
22  pdir   equ     -12             urspr. richtung
23  maxx   equ     -14             anzahl pixel x
24  maxy   equ     -16             anzahl pix y
25  ccol   equ     -18             gerade gesuchte farbnr.
26  acol   equ     -20             anzahl farben
27  komma  equ     -22             komma bei draw
28
29
30  ********************************************************************
31  *                                                                  *
32  *       programm hinter den video-ram verschieben                  *
33  *                                                                  *
34  ********************************************************************
35
36          clr.l   -(a7)           es wird auf
37          move.w  #super,-(a7)    privilegierte
38          trap    #gemdos         bereiche
```

```
39          addq.l  #6,a7           zugegriffen
40          move.l  d0,d6
41          move.w  #phybas,-(a7)   programmstart
42          trap    #xbios          ist
43          addq.l  #2,a7           video-basis
44          movea.l d0,a0           +
45          adda.w  #$7d00,a0       länge des
46          lea     (a0),a2         video-ram
47          lea     start(pc),a1
48          move.l  #fin-start-1,d0 zähler laden
49  reloc   move.b  (a1)+,(a0)+     programm
50          dbra    d0,reloc        verschieben
51
52          movea.l $456,a0         programm in
53          adda    #28,a0          die vblank-queue
54          move.l  a2,(a0)         einklinken
55          move.l  d6,-(a7)        priv.-
56          move    #super,-(a7)    status
57          trap    #gemdos         zurück-
58          addq.l  #6,a7           setzen
59          clr     -(a7)
60  *       rts                     falls von basic aufgerufen
61          trap    #gemdos         terminate
62  start:
63          tst     pflag           hardcopy gewünscht ?
64          beq     param           ja >
65          rts
66
67
68  ************************************************************************
69  *                                                                      *
70  *       parameter initialisieren                                       *
71  *                                                                      *
72  ************************************************************************
73
74  param   move    #phybas,-(a7)   physikalische
75          trap    #xbios          bildschirmbasis
76          addq.l  #2,a7           holen
77          move    #-1,-(a7)       und
```

```
78              move.l  d0,-(a7)        die
79              move.l  d0,-(a7)        logische
80              move    #setscr,-(a7)   basis
81              trap    #xbios          angleichen
82              adda.l  #12,a7
83              dc.w    init            bildschirm-params holen
84              link    a6,#-24         platz für arbeitsbereich schaffen
85              movea.l intin(a0),a3
86              movea.l ptsin(a0),a4
87              clr     wrmod(a0)       schreibmodus setzen
88              move    (a0),d7         anzahl planes
89              andi    #6,d7
90              lea     scalx(pc),a0
91              move    0(a0,d7),pscalx(a6)
92              lea     scaly(pc),a0
93              move    0(a0,d7),pscaly(a6)
94              lea     max(pc),a0
95              move    0(a0,d7),maxx(a6)
96              lea     may(pc),a0
97              move    0(a0,d7),maxy(a6)
98              lea     colc(pc),a0
99              move    0(a0,d7),acol(a6)
100             move    maxx(a6),d6
101             mulu    maxy(a6),d6
102             move.l  d6,apix(a6)
103             move    #1,ccol(a6)     farbnr. 1
104     init1   cmpi    #1,pflag        stop vor farbwahl ?
105             bne     init3           nein >
106             bsr     caps
107     init2   cmpi    #2,pflag        weiter ?
108             bne     init2           nein >
109             clr     pflag
110     init3   bsr     setcol          farbauswahl
111             bsr     home            plotter in ausgangsstellung
112             pea     -1              die suche beginnt links oben
```

```
113
114
115  ********************************************************************
116  *                                                                  *
117  *       erstes pixel einer linie suchen                            *
118  *                                                                  *
119  ********************************************************************
120
121  such    move.l  (a7)+,d7
122          addq.l  #1,d7
123          cmp.l   apix(a6),d7     alle pixel untersucht ?
124          beq     exit            ja >
125          move.l  d7,-(a7)        aktuelle position retten
126          bsr     chkpix          nächsten punkt suchen
127          cmp     ccol(a6),d0     gesuchte farbe ?
128          bne     such            nein >
129          move    #3,adir(a6)     suchrichtung ist rechts
130
131
132  ********************************************************************
133  *                                                                  *
134  *       zusammenhängende punkte zeichnen                           *
135  *                                                                  *
136  ********************************************************************
137
138  plot    bsr     mov             plotter auf neue position
139          bsr     pendwn          stift absenken
140          bsr     erase           gefundenen punkt löschen
141  plot1   clr     d0
142          bsr     nexpix          einen anschliessenden punkt suchen
143          tst     d0              passende farbe gefunden ?
144          bne     plot2           ja >
145          bsr     outcr           delimiter ausgeben
146          bsr     penup           stift anheben
147          bra     such
148  plot2   bsr     draw            linie zum nächsten punkt
149          bsr     erase           gezeichneten punkt löschen
150          bra     plot1           nächsten punkt suchen
```

```
151
152
153  ********************************************************************
154  *                                                                  *
155  *       nächsten zusammenhängenden punkt suchen                    *
156  *                                                                  *
157  ********************************************************************
158
159  nexpix  subq    #2,adir(a6)      1/4 drehung links
160          andi    #7,adir(a6)      nur 0-7 erlaubt
161          move    adir(a6),pdir(a6) ausgangsrichtung merken
162          bra     nex3
163  nex1    movem   (a7)+,d3-d4      alte koordinaten holen
164          addq    #1,adir(a6)      1/8 drehung rechts
165          andi    #7,adir(a6)      nur 0-7 erlaubt
166          move    pdir(a6),d7
167          cmp     adir(a6),d7      wieder am ausgangspunkt ?
168          bne     nex3             nein >
169          clr     d0
170          rts
171  nex3    move    adir(a6),d7      sprung
172          lsl     #1,d7            abhängig von
173          lea     j(pc),a0         richtung
174          adda    0(a0,d7),a0      vorbereiten
175          movem   d3-d4,-(a7)      koordinaten retten
176          jsr     (a0)             absprung
177          cmp     ccol(a6),d0      passende farbe gefunden ?
178          bne     nex1             nein > in anderer richtung versuche
n
179          addq.l  #4,a7            stack korrigieren
180          rts                      punkte verbinden
```

```
181
182
183  ********************************************************************
184  *                                                                  *
185  *       richtungsabhängige sprünge                                 *
186  *                                                                  *
187  ********************************************************************
188
189  j       dc.w    re-j,ru-j,un-j,lu-j,li-j,lo-j,ob-j,ro-j
189
190
191  re      addq    #1,d3           rechts
192          cmp     maxx(a6),d3     zeilenende erreicht ?
193          bcs     askpix          nein >
194          rts
195
196  ru      addq    #1,d3           rechts unten
197          cmp     maxx(a6),d3     zeilenende ?
198          bcs     un              nein >
199          rts
200
201  un      addq    #1,d4           unten
202          cmp     maxy(a6),d4     bildschirmende ?
203          bcs     askpix          nein >
204          rts
205
206  lu      addq    #1,d4           links unten
207          cmp     maxy(a6),d4     bildschirmende ?
208          bcs     li              nein >
209          rts
210
211  li      subq    #1,d3           links
212          bpl     askpix          noch kein ende
213          rts
214
215  lo      subq    #1,d3           links oben
216          bpl     ob              noch kein ende
217          rts
218
```

```
219  ob      subq    #1,d4           oben
220          bpl     askpix          noch kein ende
221          rts
222
223  ro      subq    #1,d4           rechts oben
224          bpl     re              noch kein ende
225          rts
226
227
228  ********************************************************************
229  *                                                                  *
230  *       auf gesetztes pixel testen                                 *
231  *                                                                  *
232  ********************************************************************
233
234  chkpix  divu    maxx(a6),d7     d7 nach
235          move    d7,d4           y und
236          swap    d7              x
237          move    d7,d3           umwandeln
238  askpix  cmpi    #3,pflag        abbruch der farbe ?
239          bcs     ask1            nein >
240          move    #1,pflag        evtl. stiftwechsel ermöglichen
241          bra     exit            farbe fertig >
242  ask1    move    d3,(a4)         koordinaten
243          move    d4,yko(a4)      laden
244          dc.w    getpix          line a  punkt abfragen
245          rts
246
247
248  ********************************************************************
249  *                                                                  *
250  *       pixel löschen                                              *
251  *                                                                  *
252  ********************************************************************
253
254  erase   move    d3,(a4)         koordinaten
255          move    d4,yko(a4)      laden
256          clr     (a3)            farbe 0
257          dc.w    setpix          line a  punkt setzen
```

```
258             rts
259
260
261     ********************************************************************
262     *                                                                  *
263     *       diverse ausgabe-routinen                                   *
264     *                                                                  *
265     ********************************************************************
266
267     home    lea     hm(pc),a2       plotter in grundstellung
268             bra     outstrx
269
270     setcol  lea     scols(pc),a2    farbe aus ccol einstellen
271             bsr     outstrx
272             lea     scoln(pc),a2
273             move    ccol(a6),d7
274             move.b  -1(a2,d7),d0
275             bsr     outchr
276             bra     outcr
277
278     penup   lea     pup(pc),a2      stift anheben
279             bra     outstrx
280
281     pendwn  clr     komma(a6)       stift absenken
282             lea     pdw(pc),a2
283             bra     outstrx
284
285     mov     lea     mv(pc),a2       positionieren ohne stift
286             bsr     outstrx
287             bsr     outkor
288             bra     outcr
289
290     draw    tst     komma(a6)       positionieren mit stift
291             bne     draw1
292             st      komma(a6)
293             lea     dr(pc),a2
294             bsr     outstrx
295             bra     outkor
296     draw1   bsr     outkom
```

```
297
298  outkor  move    d3,d6           koordinatenpaar als ascii ausgeben
299          mulu    pscalx(a6),d6
300          bsr     outw
301          bsr     outkom          komma ausgeben
302          move    maxy(a6),d7     umkehrung der
303          sub     d4,d7           y-koordinate
304          move    d7,d6
305          mulu    pscaly(a6),d6
306
307  outw    move.l  #1000,d7        hexwort in d6 als ascii ausgeben
308  outw1   andi.l  #$3fff,d6
309          divu    d7,d6
310  outw3   move    d6,d0
311          ori     #48,d0
312          bsr     outchr
313          swap    d6
314  outw4   divu    #10,d7
315          bne     outw1
316          rts
317
318  outstrx clr     d2              string ausgeben (zähler-1 an (a2))
319          move.b  (a2)+,d2
320  outstr  move.b  (a2)+,d0        string an (a2) ausgeben (zähler in
d2)
321          bsr     outchr
322          dbra    d2,outstr
323          rts
324
325  caps    lea     cap(pc),a2
326          bsr     outstrx
327
328  outcr   move    #13,d0          cr ausgeben
329          bra     outchr
330
331  outkom  move    #44,d0          komma ausgeben
332
333  outchr  movem.l d0-d2/a0-a2,-(a7)
334          andi    #255,d0
```

```
335           move    d0,-(a7)        zeichen in d0
336           move    #prt,-(a7)      an drucker
337           move    #bconout,-(a7)  ausgeben
338           trap    #bios
339           addq.l  #6,a7
340           movem.l (a7)+,d0-d2/a0-a2
341           rts
342
343
344   ********************************************************************
345   *                                                                  *
346   *       ausgang                                                    *
347   *                                                                  *
348   ********************************************************************
349
350   exit    addq    #1,ccol(a6)     gesuchte farbe +1
351           move    acol(a6),d7
352           cmp     ccol(a6),d7     alle farben abgearbeitet ?
353           bpl     init1           nein >
354   exitx   unlk    a6              reservierten platz freigeben
355           move    #-1,pflag       hardcopy-flag löschen
356           bsr     home            plotter in grundstellung
357           bra     caps            hut auf die stifte
358
359
360   ********************************************************************
361   *                                                                  *
362   *       konstanten                                                 *
363   *                                                                  *
364   ********************************************************************
365
366   scalx   dc.w    4,4,4           faktoren x
367   scaly   dc.w    4,4,4           faktoren y
368   max     dc.w    640,640,320     anzahl pixel x
369   may     dc.w    400,200,200     anzahl pixel y
370   colc    dc.w    1,3,15          anzahl farben
371   mv      dc.b    1,"MA"          move absolut
372   cap     dc.b    3,"SP-1"        stiftwechsel
373   pup     dc.b    5,"MR0,0",13    move relativ (pen up)
```

```
374   pdw     dc.b    5,"DR0,0",13     draw relativ (pen down)
375   dr      dc.b    1,"DA"           draw absolut
376   hm      dc.b    3,13,"HO",13     grundstellung
377   scols   dc.b    1,"SP"           farbwechsel
378   scoln   dc.b    "123412341234123"
378
379   fin     equ     *
```

Auch hier können Sie wieder das Programm mittels der Zeilen 371 bis 378 an Ihr Gerät anpassen. Es wurde jedoch für die Stringausgabe ein anderes Verfahren als bei der Druckerhardcopy gewählt. Hier steht der Bytezähler-1 vor dem Text, der diesmal von vorne nach hinten ausgegeben wird.
Die Zeilen 366 und 367 geben an, wieviele Steps der Plotter für einen Punkt auf dem Bildschirm machen soll. Sinnvollerweise wählt man die Schrittweite entsprechend der Dicke des Stiftes. In unserem Falle sind das 0.4mm.

Nachfolgend finden Sie das entsprechende BASIC-Programm. Auch hier ist der TERM am Ende des Verschiebeprogramms wieder durch RTS ersetzt.

```
10    dim a%(411)
20    for i=0 to 411
30    read a%(i)
40    next i
50    b=varptr(a%(0))
60    call b
70    end
950   data &H42A7,&H3F3C,&H0020
960   data &H4E41,&H5C8F,&H2C00,&H3F3C,&H0002,&H4E4E,&H548F,&H2040
970   data &HD0FC,&H7D00,&H45D0,&H43FA,&H0028,&H203C,&H0000,&H02F1
980   data &H10D9,&H51C8,&HFFFC,&H2079,&H0000,&H0456,&HD0FC,&H001C
990   data &H208A,&H2F06,&H3F3C,&H0020,&H4E41,&H5C8F,&H4E75,&H4E41
1000  data &H4A79,&H0000,&H04EE,&H6702,&H4E75,&H3F3C,&H0002,&H4E4E
1010  data &H548F,&H3F3C,&HFFFF,&H2F00,&H2F00,&H3F3C,&H0005,&H4E4E
1020  data &HDFFC,&H0000,&H000C,&HA000,&H4E56,&HFFE8,&H2668,&H0008
1030  data &H2868,&H000C,&H4268,&H0024,&H3E10,&H0247,&H0006,&H41FA
1040  data &H0264,&H3D70,&H7000,&HFFFA,&H41FA,&H0260,&H3D70,&H7000
1050  data &HFFF8,&H41FA,&H025C,&H3D70,&H7000,&HFFF2,&H41FA,&H0258
1060  data &H3D70,&H7000,&HFFF0,&H41FA,&H0254,&H3D70,&H7000,&HFFEC
1070  data &H3C2E,&HFFF2,&HCCEE,&HFFF0,&H2D46,&HFFFC,&H3D7C,&H0001
1080  data &HFFEE,&H0C79,&H0001,&H0000,&H04EE,&H6614,&H6100,&H01CE
1090  data &H0C79,&H0002,&H0000,&H04EE,&H66F6,&H4279,&H0000,&H04EE
1100  data &H6100,&H012C,&H6100,&H0120,&H4879,&HFFFF,&HFFFF,&H2E1F
1110  data &H5287,&HBEAE,&HFFFC,&H6700,&H01CC,&H2F07,&H6100,&H00D2
1120  data &HB06E,&HFFEE,&H66E8,&H3D7C,&H0003,&HFFF6,&H6100,&H012C
1130  data &H6100,&H011E,&H6100,&H00E4,&H4240,&H6118,&H4A40,&H660A
1140  data &H6100,&H0180,&H6100,&H0104,&H60C4,&H6100,&H0118,&H6100
1150  data &H00CA,&H60E4,&H556E,&HFFF6,&H026E,&H0007,&HFFF6,&H3D6E
1160  data &HFFF6,&HFFF4,&H601C,&H4C9F,&H0018,&H526E,&HFFF6,&H026E
1170  data &H0007,&HFFF6,&H3E2E,&HFFF4,&HBE6E,&HFFF6,&H6604,&H4240
1180  data &H4E75,&H3E2E,&HFFF6,&HE34F,&H41FA,&H0016,&HD0F0,&H7000
1190  data &H48A7,&H1800,&H4E90,&HB06E,&HFFEE,&H66CA,&H588F,&H4E75
1200  data &H0010,&H001A,&H0024,&H002E,&H0038,&H003E,&H0044,&H004A
1210  data &H5243,&HB66E,&HFFF2,&H6542,&H4E75,&H5243,&HB66E,&HFFF2
```

```
1220  data &H6502,&H4E75,&H5244,&HB86E,&HFFF0,&H652E,&H4E75,&H5244
1230  data &HB86E,&HFFF0,&H6502,&H4E75,&H5343,&H6A1E,&H4E75,&H5343
1240  data &H6A02,&H4E75,&H5344,&H6A12,&H4E75,&H5344,&H6AC2,&H4E75
1250  data &H8EEE,&HFFF2,&H3807,&H4847,&H3607,&H0C79,&H0003,&H0000
1260  data &H04EE,&H650C,&H33FC,&H0001,&H0000,&H04EE,&H6000,&H00D6
1270  data &H3883,&H3944,&H0002,&HA002,&H4E75,&H3883,&H3944,&H0002
1280  data &H4253,&HA001,&H4E75,&H45FA,&H0113,&H6000,&H0082,&H45FA
1290  data &H0110,&H6100,&H007A,&H45FA,&H010B,&H3E2E,&HFFEE,&H1032
1300  data &H70FF,&H6100,&H0084,&H6000,&H007A,&H45FA,&H00DE,&H605E
1310  data &H426E,&HFFEA,&H45FA,&H00DB,&H6054,&H45FA,&H00C6,&H614E
1320  data &H6116,&H605E,&H4A6E,&HFFEA,&H660C,&H50EE,&HFFEA,&H45FA
1330  data &H00C8,&H613A,&H6002,&H614E,&H3C03,&HCCEE,&HFFFA,&H610E
1340  data &H6144,&H3E2E,&HFFF0,&H9E44,&H3C07,&HCCEE,&HFFF8,&H2E3C
1350  data &H0000,&H03E8,&H0286,&H0000,&H3FFF,&H8CC7,&H3006,&H0040
1360  data &H0030,&H6124,&H4846,&H8EFC,&H000A,&H66E8,&H4E75,&H4242
1370  data &H141A,&H101A,&H6112,&H51CA,&HFFFA,&H4E75,&H45FA,&H0067
1380  data &H61EC,&H700D,&H6002,&H702C,&H48E7,&HE0E0,&H0240,&H00FF
1390  data &H3F00,&H3F3C,&H0000,&H3F3C,&H0003,&H4E4D,&H5C8F,&H4CDF
1400  data &H0707,&H4E75,&H526E,&HFFEE,&H3E2E,&HFFEC,&HBE6E,&HFFEE
1410  data &H6A00,&HFDF0,&H4E5E,&H33FC,&HFFFF,&H0000,&H04EE,&H6100
1420  data &HFF26,&H60B8,&H0004,&H0004,&H0004,&H0004,&H0004,&H0004
1430  data &H0280,&H0280,&H0140,&H0190,&H00C8,&H00C8,&H0001,&H0003
1440  data &H000F,&H014D,&H4103,&H5350,&H2D31,&H054D,&H5230,&H2C30
1450  data &H0D05,&H4452,&H302C,&H300D,&H0144,&H4103,&H0D48,&H4F0D
1460  data &H0153,&H5031,&H3233,&H3431,&H3233,&H3431,&H3233,&H3431
1470  data &H3233
```

4.1 Die Programmierumgebung GEM

Genauso einfach und komfortabel wie sich GEM dem Anwender in der Bedienung präsentiert, so offenbart GEM sich auch dem Programmierer als Schnittstelle zu den eigenen und den Funktionen von GEMDOS. Prinzipiell wird ein GEM-Programm auch immer einfacher und schneller zu schreiben sein als ein vergleichbares Programm für einen herkömmlichen Rechner mit den üblichen Betriebssystemroutinen. Denn GEM enthält eine Vielzahl von Unterprogrammen, die der Programmierer auf unkomplizierte Art und Weise aufrufen kann - nur muß er erst einmal wissen, was ihm mit GEM eigentlich zur Verfügung steht, und wie er diese Routinen in seine Applikation einbinden kann. Denn ein Programm unter GEM unterscheidet sich doch erheblich von einem Programm unter einem Betriebssystem wie beispielsweise CP/M 68k - auch dann, wenn man die feine Unterscheidung zwischen Accessory und Applikation erst einmal unberücksichtigtläßt.

Dies ist dann wohl auch der Grund, warum die für den ST typische Software so lange auf sich warten läßt, erfordert doch die ganz andere Benutzeroberfläche auch ganz andere Programmkonstrukte. Bestand früher der Initialisierungsteil eines Programmes nur aus Wertzuweisungen an bestimmte Variablen, so muß der Programmierer eines ST sich zunächst erst einmal um seine Arbeitsstation kümmern, muß er beispielsweise festlegen, mit welchen Koordinaten er arbeiten will und dergleichen mehr. Doch sind diese prinzipiellen Vorgänge erst einmal verstanden worden, die entsprechenden Routinen geschrieben, und deren Gebrauch in Fleisch und Blut übergegangen, dann kann man sich als Softwareersteller auf das Wesentliche konzentrieren und hat Routineaufgaben wie Fensterverwaltung, Abfrage von Maus, Tastatur und dergleichen jeweils mit einem absoluten Minimum an Arbeit erledigt. Denn schließlich lassen sich diese Initialisierungsteile doch für jede weitere Anwendung ohne große Änderungen - meist kann man sogar auf jegliche Änderung verzichten - nutzen.

Und so wird jeder Programmierer, der unter GEM arbeitet, nach kurzer Zeit auch seine eigene Sammlung an Unterprogrammen vorweisen können, er wird eine eigene Bibliothek zusammengestellt haben und einen wesentlichen Teil der Programmierarbeit durch das Zusammenstellen der Routinen erledigen.

An einem konkreten Beispiel, nämlich an einem Programm, mit dessen Hilfe Sie Ihren Drucker auf die gewünschte Schriftart einstellen können, wollen wir Ihnen im folgenden einige dieser benötigten Standardroutinen vor- wie auch die Entwicklung sowohl von Accessories als auch von Applikationen verständlich darstellen. Um jedoch Aufbau und Arbeitsweise der Programme verstehen zu können, ist ein grober Überblick über das Betriebssystem der ST-Computer unabdingbar, weshalb wir Ihnen zunächst in groben Zügen den Graphics Environment Manager nahebringen wollen.

4.1.1 GEM Intern

Die komplette GEM-Software, größtenteils übrigens in C geschrieben, stellt eine grafisch orientierte Funktionseinheit dar, die mit ihren besonderen Techniken eine starke Vereinfachung der Handhabung der Atari-Computer bietet. Ein Anwender nimmt die Dienstleistungen von GEM einfach als gegeben hin, er macht sich keine Gedanken über den programmtechnischen Aufbau der Maussteuerung, der Handhabung von Pictogrammen und Drop-Down-Menüs oder der Windows. Ihm kann es auch egal sein, über welche technische Ausstattung er verfügt, ob er nun einen 520ST oder 520ST+ benutzt.

Wieviel Arbeitsstunden müßte ein Programmierer wohl aufwenden, um standesgemäße Programme zu schreiben, wenn er für all diese zu bewältigenden Aufgaben eigene Routinen schreiben müßte?
Doch warum sollte er das Rad zum zweiten Male erfinden, kann

er statt dessen doch auch die GEM-Routinen nutzen. Soweit sie uns zum heutigen Zeitpunkt bekannt sind, verfügen alle augenblicklich zum Atari angebotenen Programmiersprachen auch über die notwendigen Bibliotheken, um 'vernünftige' ST-Programme zu schreiben. Glücklicherweise sieht es dabei sogar so aus, daß die Namensgebung der einzelnen Routinen beibehalten wurde, so daß die folgenden Programmbeispiele gleich informativ für Pascal, Modula-2 oder C Programmierer sind. Die Sourcecodes, und in großem Maße auch die daraus resultierenden Programme, werden sich durch den Gebrauch der GEM-Bibliotheken nur unwesentlich voneinander unterscheiden.

Denn GEM bietet sämtliche Routinen zur Datenein- und ausgabe, angefangen von der Ausgabe eines Textstrings auf dem Bildschirm bis hin zur Verwaltung von Ein-/Ausgabemasken, die unter GEM besser Formulare genannt werden.

Zuständig für solche Aufgaben sind zum einen das 'Virtual Device Interface' und zum anderen das 'Application Environment Services'. Hinter diesen Bezeichnungen verstecken sich die Programmteile, die für die Bedienung der Hardware des Computers zuständig sind und es dem Programmierer ermöglichen, diesen als Black Box zu betrachten. Das VDI geht dabei sogar so weit, daß es zwei verschiedene Koordinatensysteme erlaubt, von denen eines ein mathematisch exaktes Normgerät adressiert, so daß es Programmen, die mit diesen Koordinaten arbeiten, egal ist, ob die Ausgabe schließlich an einen Bildschirm oder einen Drucker gerichtet wird. Hiermit erklärt sich allerdings auch schon der erste unabdingbare Schritt einer jeden GEM-Anwendung, nämlich das Eröffnen der Arbeitsstation. Schließlich muß das VDI wissen, wohin, und vor allem, in welcher Form die Daten ausgegeben werden müssen. Für den Programmierer liegt der größte Vorteil nun darin, daß er keinerlei Probleme beim Umschreiben eines Programmes entweder für ein anderes Ausgabegerät oder gar für einen anderen GEm-Rechner hat, ändert sich an den Ein- und Ausgaberoutinen doch rein gar nichts.

Weiterhin sieht das VDI Systemcalls für die unterschiedlichsten Grafikoperationen vor, worunter beispielsweise Zeichenroutinen (Polyline, Circle, Fill) und Textein- und ausgabefunktionen gehören.

Das AES schließlich übernimmt die ankommenden Daten von allen denkbaren Eingabegeräten. Zu seinem Aufgabenbereich gehört das Auslesen des Keyboards, die Auswertung der Bewegungen der Maus und die Kontrolle des Maustasters, wie auch die Kontrolle der einzelnen Ausgabeelemente, also der Fenster, Drop-Down-Menüs und Icons. Dabei ist das AES auch für den geordneten Ablauf eines Multitaskingbetriebs zuständig. Denken Sie nur an unseren Druckerspooler, der schließlich auch immer neben dem eigentlichen Hauptprogramm läuft, oder die Uhr des Kontrollfeldes.

4.1.2 Das Virtual Device Interface

Genau genommen besteht das VDI wiederum aus zwei Teilen, nämlich dem Graphics Device Operating System (GDOS), das im wesentlichen eine Vielzahl von geräteunabhängigen Grafikroutinen enthält, und den Gerätetreibern, die auch die Informationen über die zu verwendenden Zeichensätze, die Fonts, enthalten. So muß dem VDI beispielsweise mitgeteilt werden, daß die Arbeitsstation SW-Monitor nur über zwei Farbstifte verfügt und der Bereich der adressierbaren Koordinaten sich von 0,0 bis 639,399 erstreckt. Wird hingegen als Arbeitsstation ein Colormonitor angegeben, kann das VDI auf sechzehn Farbstift zurückgreifen, und es weiß auch, daß das Auflösungsvermögen dieser Arbeitsstation nicht ganz so hoch ist.

Nun ist es recht einleuchtend, daß, wenn der Programmierer eine waagrechte Linie der Länge 600 Punkte festlegt, diese auf dem Farbmonitor bei 320 darstellbaren Punkten pro Zeile

nur ungefähr zur Hälfte abgebildet wird. Um nun jedoch die erwähnte Geräteunabhängigkeit anbieten zu können, kennt das VDI zwei voneinander verschiedene Koordinatensysteme, nämlich

- **NDC, Normalized Device Coordinates,** und
- **RC, Raster Coordinates.**

Während die Raster-Koordinaten den physikalisch vorhandenen Bildpunkten entsprechen, die im Falle des ATARI somit im Bereich von 320 x 200 und 640 x 400 Punkten liegen, oder bei Plottern in x- und y-Schritten gemessen werden, beziehen sich die 'normalisierten' Koordinaten auf eine idealisierte Bildfläche. Deren Orientierung entspricht dann auch unserem gewohnten kartesischen Koordinatensystem, d.h. der Punkt 0,0 liegt unten links und der Punkt mit den größtmöglichen Werten für x und y liegt in der rechten oberen Ecke der Zeichenfläche.

So erstreckt sich der Bereich bei den normalisierten Koordinaten von 0,0 bis 32767,32767, was einer sich geometrisch korrekt verhaltenden Bildfläche mit sehr hohem Auflösungsvermögen (oder eben entsprechender Größe) entspricht.

Der Programmierer unter GEM kann nun wählen, welches Koordinatensystem er benutzen möchte. Entscheidet er sich für NDC, so werden die eingegebenen Werte während der Programmausführung von GDOS, dem dafür zuständigen Teil des GEM, auf die wirklich vorhandenen Koordinaten umgerechnet. Wird ein Quadrat der Länge 100 eingegeben, so erscheint es auch als Quadrat auf dem Monitor. Hat er sich hingegen für die Rasterkoordinaten entschieden, so nimmt das VDI keinerlei Umrechnungen vor. Der Programmierer beziehungsweise das laufende Programm muß selbst sämtliche Maßstabsveränderungen beachten.

Vorteile bringt das NDC-System in Verbindung mit dem VDI vor allem beim Austausch von Grafiken zwischen unterschiedlichen Peripheriegeräten. So werden wir mit den üblichen Rasterkoordinaten und der bislang gewohnten Ansteuerung von Druckern immer unsere Schwierigkeiten haben, besonders was das Verhältnis zwischen Länge und Breite der Abbildung betrifft. Denn bei einem Koordinatensystem von 600 x 200 wird das Verhältnis zwischen den Koordinaten auf dem Bildschirm ungefähr 1:1.8 betragen, hier als Quadrat erscheinende Rechtecke werden auf einem Drucker, sofern er nicht zufällig mit genau dem gleichen System arbeitet, niemals mehr als Quadrat erscheinen.

Doch im Falle einer Adressierung an das VDI kann der Geräte-Treiber die notwendigen Umrechnungen vornehmen; Darstellungen gleich welcher Art werden demnach auf sämtlichen Peripheriegeräten gleich in Erscheinung treten. Als nachteilig kann sich dabei jedoch der erhöhte Zeitbedarf solcher Programme erweisen, schließlich muß die Lage eines jeden einzelnen Grafikpunktes erst auf das aktuelle Koordinatensystem umgerechnet werden. Aus diesem Grunde empfiehlt es sich eigentlich, immer mit den Rasterkoordinaten zu arbeiten, es sei denn, Sie müssen unbedingt portable Programme schreiben.

4.1.3 Das Application Environment Services

Das AES setzt sich aus mehreren Teilen zusammen, als da wären:

- die Unterprogrammbibliotheken
- der Dispatcher
- die Shell
- der Desk Accessory Buffer
- der Menü- / Alarmbuffer

Der Menü-/Alarmpuffer trägt wesentlich zur schnellen Funktion von GEM bei. So wird innerhalb des Menü-Datenpuffers beispielsweise der Teil des Bildschirmes zwischengespeichert, der durch das Entfalten eines Drop-Down-Menus überlagert wird. Nach Gebrauch des betreffenden Menüs sorgt ein Unterprogramm des AES für eine blitzschnelle Restaurierung des Arbeitsplatzes. Es ist also insbesondere nicht notwendig, daß die Applikation, also Sie als der zuständige Programmierer sich um diese Aktion kümmern müssen. Anzumerken ist hier lediglich, daß nur soviel Platz reserviert wird, daß maximal ein Viertel des Bildschirminhaltes zwischengespeichert werden kann. Aus diesem Grunde sollten Sie sich auch keine Mühe geben, jemals eine Warntafel auf die volle Bildschirmgröße auszudehnen!

In ähnlicher Weise wird auch der Desk Accessory Buffer, der uns später noch in besonderem Maße interessieren wird, eingesetzt. Nur werden hier nicht bloß Daten gespeichert, sondern in diesem reservierten Speicherbereich werden Hilfsprogramme wie beispielsweise PR-INIT geladen, die der Benutzer während eines Programmlaufes kurzzeitig aufrufen will.

Die Aufgabe des Dispatchers ist es, gleichzeitig auszuführende Arbeiten dem Prozessor zugänglich zu machen. Wobei 'gleichzeitig' unbestreitbar ein relativ zu verstehender Begriff ist, denn was für uns zur scheinbar gleichen Zeit geschieht, kann bei einer mit 8 Mhz getakteten CPU Millionen Schritte auseinanderliegen.

Um nun nicht wertvolle Rechenzeit zu verschwenden, führt der Dispatcher zwei Listen: die 'Ready List', in der nacheinander alle Programme aufgeführt sind, die gerade laufen und die auf eine CPU-Zuweisung warten, und die 'Not Ready List', in der alle Prozesse aufgeführt sind, die auf ein bestimmtes Ereignis warten müssen, ehe sie bearbeitet werden. Solch ein Ereignis kann beispielsweise

- ein Tastendruck
- die Betätigung eines Mausknopfes
- eine Bewegung der Maus
- eine Mitteilung
- oder das Verstreichen eines bestimmten Zeitintervalles

sein. So wird unser Druckereinstellprogramm zunächst immer in der 'Not Ready List' stehen und dort auf die Mitteilung warten, daß das Desk Accessory FXINIT aufgerufen wurde.

Ist dies der Fall, hat der Benutzer seinen Mauspointer somit innerhalb des Menüs DESK in die entsprechende Zeile bewegt und den linken Maustaster betätigt, wird das Programm von der Not Ready List gestrichen und in die ready List eingetragen. Der dort jeweils obenanstehende Prozeß wird bearbeitet und nach einer bestimmten Zeit in das Ende der Liste eingetragen, so daß der nächste Prozeß bearbeitet werden kann. Auf diese Art und Weise verteilt der Dispatcher die CPU-Zeit auf die gerade laufende Anwendung, auf eventuell laufende Hintergrundprogramme, wie beispielsweise den Drucker-Spooler, und auf die Betriebssystemroutinen. Insgesamt kann der Dispatcher neben diesen AES-Calls bis zu sechs Aufgaben wie die im Desk Accessory Code Buffer befindlichen Programme verwalten.

Wesentlich für das Verständnis von GEM-Programmen ist der hieraus resultierende Unterschied zur üblichen Praxis:

Sofern Sie schon früher einmal programmiert haben, dann sah es so aus, daß Sie innerhalb Ihres Programmes eine Schleifenkonstruktion vorgesehen hatten, die in mehr oder weniger regelmäßigen Abständen überprüft, ob ein Ereignis eingetroffen ist (INKEY$). Oder aber, Sie haben gleich den ganzen Programmablauf angehalten (INPUT). Profan ausgedrückt: Sie sagen dem Programm, und somit auch dem Rechner, daß es auf dieses oder jenes warten soll.

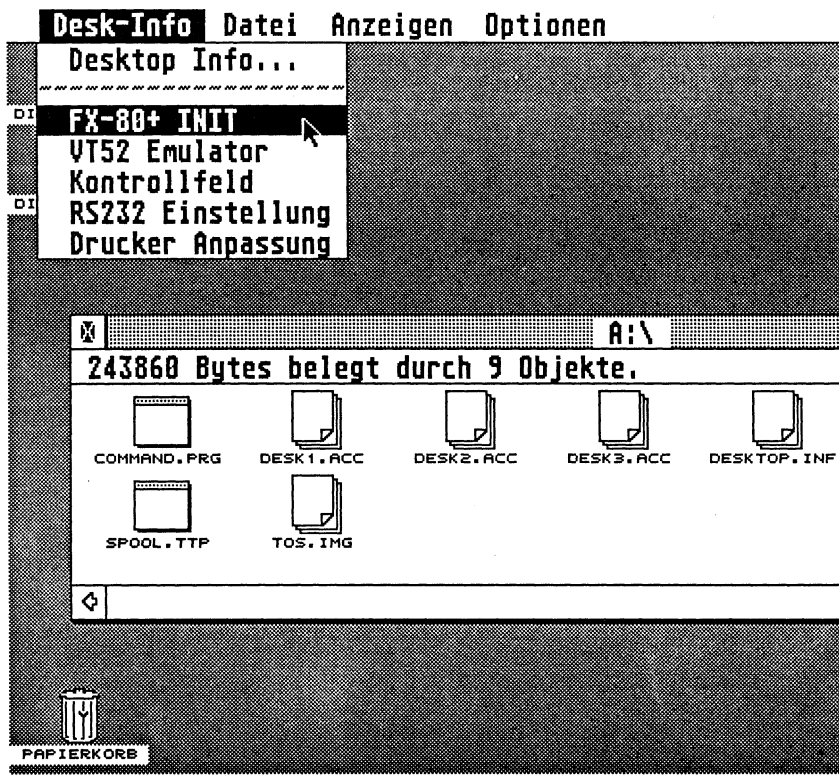
Im Gegensatz dazu müssen Sie auf dem Atari dem Betriebssystem sagen, daß es so lange mit etwas anderem

weiterarbeiten kann, bis das gewünschte Ereignis eingetroffen ist. Bis zu diesem Zeitpunkt verbleibt das Programm auf der 'Not Ready List', es wird keine CPU-Zeit mit Warten verschwendet. Erst durch diese Vorgehensweise wird Multitaskingbetrieb möglich, denn die anderen Prozesse der 'Ready List' werden weiterhin bearbeitet.

Die Library schließlich enthält sämtliche Unterprogramme zur Manipulation der Fenster, zur Abfrage und Behandlung der Maus wie auch zur Anzeige von Systemmeldungen wie der Dialogboxen und der Drop-Down-Menüs. Selbst die Abfrageroutinen der Dialogboxen (OK, Cancel, Retry) und die Steuerfunktionen für die Fenster lassen sich hier finden.
Selbstverständlich kann und muß der unter GEM arbeitende Programmierer auch auf diese Routinen zugreifen. Unter anderem bietet die AES-Bibliothek Routinen, die das zu startende Programm initialisieren, d.h. Routinen, die dafür sorgen, daß das Anwendungsprogramm Vorrang vor allen anderen Tasks hat. Natürlich muß der Programmierer hier dann auch festlegen, ob das Programm sich als Applikation oder als Accessory verhalten soll.

Der Screen Manager schließlich übernimmt die Kontrolle über die Maus, sobald diese sich außerhalb der Arbeitsfläche des gerade aktiven Fensters befindet. Als Arbeitsfläche ist dabei der Inhalt des Fensters definiert, Titel- und Informationszeile gehören also nicht dazu.
Er wird somit aktiv, wenn der Anwender den umschlossenen Bereich des zuoberst liegenden Fensters verläßt, wenn er beispielsweise die Menüleiste mit ihren Drop-Down-Menüs benutzt. Er überwacht dann sämtliche Aktionen des Benutzers und protokolliert diese gegebenenfalls, d.h. er sendet die Nachricht an die gerade laufende Applikation das Fenster neu zu zeichnen.

Die Shell schließlich ist eigentlich ein Teil der AES-Bibliotheken. Nur steht sie nach Aufruf des Desktops zu Beginn der Ready-Liste. Sie ist verantwortlich für den ordnungsgemäßen Aufruf einer Applikation und steht eigentlich immer zwischen dem Desktop und der Applikation. So übergibt das Desktop beim Programmstart an die Shell die Information, ob es sich um eine TOS- oder um eine GEM-Anwendung handelt, außerdem wird ihr der Pathname zur korrekten Subdirectory (die bei GEM natürlich als Ordner erscheint) übermittelt. Anschließend terminiert das Desktop, und die Shell ist für das Laden und Starten der Applikation verantwortlich. Nach Beendigung des Programmlaufes wird wiederum die Shell aufgerufen, und diese wird dann entweder das Desktop neu aktivieren oder aber gleich ein weiteres Anwenderprogramm starten.



Bevor wir nun daran gehen, ein Programm für den Atari ST zu schreiben, müssen wir uns über dessen Erscheinungsweise klar werden. So haben wir zunächst einmal zwischen Applikationen und Accessories zu unterscheiden.
Unter einer Applikation verstehen wir dabei ein ganz normales Programm wie beispielsweise eine Textverarbeitung oder ein Datenbanksystem, das vor dem Programmstart von der Shell in den Hauptspeicher geladen wird.
Ein Accessory hingegen ist eine Miniapplikation, dienoch während des Bootvorganges des TOS in den Accessory-Buffer geladen wird, und, wie wir später noch sehen werden, auch sogleich gestartet wird. Doch besteht die einzige Aufgabe eines Accessories zu diesem Zeitpunkt daraus, auf den Aufrufbefehl zu warten, der jederzeit gegeben werden kann, insbesondere also auch, wenn bereits eine (Haupt-) Applikation abgearbeitet wird.

Weiterhin haben wir uns zu entscheiden, in welcher Umgebung die Applikation laufen soll.
Zur Wahl stehen:

- das TOS,
- die Schreibtischoberfläche,
- ein zuvor definiertes Fenster.

Eine Tos-Anwendung entspricht ganz und gar einer herkömmlichen Anwendung. Die einzige Programmumgebung ist GEMDOS, AES- und VDI- Routinen können nicht benutzt werden.

Die zweite Möglichkeit sieht zwar eine Nutzung von AES- und VDI- Routinen vor, doch wird auf die Umrahmung der Arbeitsfläche mit einem Fenster verzichtet. Der Arbeitsbereich entspricht also dem gesamten darstellbaren Bildschirmbereich. Als nachteilig kann es sich jedoch erweisen, daß in diesem Falle natürlich auch immer nur ein einziges Programm bearbeitet werden kann, fehlen doch entsprechende Applikationsfenster und ist auch die Menüleiste des Desktops nicht greifbar, um beispielsweise Accessories aufzurufen.

Eröffnet das Programm bei seiner Initialisierung ein Fenster, dann entspricht es zwar dem Atari-Standard, doch hat der Programmierer dann auch zahlreiche zusätzliche Routinen zu implementieren. Bei voller Nutzung der Fenstersteuerung fällt darunter beispielsweise auch das Verschieben der Fenster, Aktivieren des angeklickten Fensters, Vergrößern und Verkleinern des Fensters mit eventuellem Clippen des Fensterinhaltes usw.

4.1.4 Das Resourcefile

Hinzu kommt bei einer GEM-Anwendung noch die Einbindung des sogenannten Resourcefiles (.RSC), das vor allem für eine leichte Anpassung der Software an die jeweilige Landessprache des Benutzers sorgt.

Mußte früher noch das ganze Programm neu geschrieben, an den entsprechenden Stellen des Sourcecodes die auszugebenden Mitteilungen ersetzt und daran anschließend das Programm wiederum in ausführbaren Maschinencode übersetzt werden, so erstrecken sich die Änderungen bei der GEM-Software einzig und allein auf das Resourcefile. In dieser Binärdatei befinden sich sämtliche an den Benutzer auszugebende Mitteilungen (Alerts) wie auch in verschlüsselter Form die gesamte Menüstruktur des Programmes (Dialogs und Drop-down-Menüs).

Auf einfachste Art und Weise kann das RSC-File eines Programmes mit dem Resource Construction Set, auf das wir später noch ausführlich eingehen werden, editiert bzw. auch erzeugt werden. Natürlich ist aber auch ein 'Schreiben per Hand' möglich, ebenso, daß die entsprechenden Daten, die nach dem Programmstart in einen bestimmten Speicherbereich geladen werden, dort auch vom Programm verändert und ergänzt werden.

4.1.5 Arbeiten unter TOS

Der Verzicht auf die Besonderheiten eines mit GEM ausgestatteten Computers empfiehlt sich vor allem, wenn es darum geht, bestehende Software umzuschreiben.

So lassen sich dann auch C-Programme, die für einen anderen Rechner entwickelt worden sind, problemlos auf den Atari ST umschreiben. Auch sollte ein Anwender, für den die Programmiersprache C völlig neu ist, sich zunächst an einer reinen TOS-Anwendung versuchen, muß er sich dann doch nicht um spezielle AES- und VDI-Aufrufe kümmern.

Aus diesem Grunde wollen wir Ihen an dieser Stelle zunächst ein ganz einfaches C-Listing vorstellen.

4.2 Einundzwanzig

Ziel der beiden Mitspieler in diesem Spiel, von denen einer natürlich durch den Computer repräsentiert werden soll, ist es, durch abwechselnde Addition von eins oder zwei einen Zähler ausgehend von Null auf den Stand einundzwanzig zu bringen. 'Einundzwanzig' gehört somit zur Reihe der Nim-Spiele, einer Klasse einfacher Strategiespiele, für deren Lösung sich einfache mathematische Regeln aufstellen lassen. Natürlich wollen wir an dieser Stelle nicht ins Reich der Mathematik abschweifen, doch um die Entdeckung einer einfachen Strategie kommen auch wir nicht herum. Überlegen wir uns dazu folgendes: Es gibt einen genau definierten Anfangs- (stand = 0) wie auch Endzustand (ziel = 21). Als legale Spielzüge sind die Erhöhung des Spielstandes um eins oder zwei definiert. Als Sieger geht aus dem Spiel derjenige hervor, der durch den letzten legalen Spielzug den Zählerstand 21 erreicht.

Unter Berücksichtigung dieser Regeln läßt sich zur Findung einer geeigneten Strategie unser Endziel, den Wert 21 zu

erlangen, in mehrere Teilziele aufsplittern. Denn wir werden bestrebt sein, unserem Gegner niemals einen Zählerstand von 19 zu präsentieren, da dieser sonst die 21 erreichen würde. Vielmehr trachten wir danach, selber die 19 zu erreichen. Eines unserer Teilziele lautet somit 19, und unter Berücksichtigung derselben Überlegungen erhalten wir als weiteres Teilziel die Zahl 16.

Auf diese Art und Weise können wir die Zählerstände 1, 4, 7, 10, 13, 16 und 19 als unsere Gewinn- und als gegnerische Verluststellungen ermitteln. Unsere Spielstrategie sieht dann vor, daß wir den entsprechenden Zug ausführen (+1 oder +2), um die nächste Gewinnzahl zu erreichen.

Bevor wir nun mit der Programmierung beginnen, sollten wir uns auch noch einige Gedanken zum Aufbau des Programmes machen. Es wird sich dabei um ein linear ablaufendes Programm handeln, das sich in die Teile Initialisierung (init), Spielstandausgabe (ausgabe), Spielerzug (player), Computerzug (computer), Auswertung und Schluß aufgliedert.

Im ersten Teil des Programmes sollte eine kurze Anweisung an den Sieler ausgegeben, der Zählerstand auf Null gesetzt und dem Spieler die Wahl überlassen werden, wer mit dem Spiel beginnen soll.

Der Programmteil Spielerzug wird aus einer einfachen Eingabefunktion mit anschließender Mehrfachauswahl bestehen (getchar, switch case), getchar ist dabei eine Funktion der Standardbibliothek zu C, deren Aufgabe es ist, ein einzelnes Zeichen von der Tastatur zu übernehmen. Für das Spiel ist dieser einfache Aufruf völlig ausreichend, schließlich soll nur die Übername einer einzelnen Ziffer erfolgen. Diese wird dann zum augenblicklichen Zählerstand addiert, was in unserem Beispiel durch den Incrementoperator ++ geschieht (stand++ ist analog zur Basicformulierung stand=stand+1).

Bevor innerhalb der Hauptschleife des Programmmes (while (stand<ziel)) jeweils die andere Partei zum Zuge kommt,

empfiehlt es sich, zunächst zu überprüfen, ob das Endziel nicht schon erreicht wurde, was im Falle von 21 mit dem Gleichheitsoperator == festgestellt wird: if (stand == ziel) break;, ansonsten kann es geschehen, daß das Programm läuft und läuft, ohne jemals zum Ende zu kommen.

Break dient in C immer zum vorzeitigen Verlassen einer Schleifenkonstruktion wie auch zum ordnungsgemäßen Abschluß einer Case-Anweisung, so auch innerhalb der Funktion computer(). Bei dieser Form der Mehrfachauswahl läßt C einen Default-Zweig zu, der immer dann ausgeführt wird, wenn keine der Bedingungen zutrifft. Wurde jedoch ein Case-Fall nicht mit break abgeschlossen, würde C aufgrund der sequentiellen Abarbeitungsfolge auch zusätzlich noch den Default-Zweig ausführen.

Innerhalb der Routine Computerzug() wird der Computer natürlich ebenfalls versuchen, die nächste Gewinnzahl zu erreichen - sollte dies unmöglich sein, weil der Spieler bereits eine der 'richtigen' Zahlen erlangt hat, spielt es keine Rolle, ob der Computer den Zug +1 oder +2 ausführt (default).

Sofern Sie zu den C-Neulingen gehören, werden sie auch folgende allgemeine Erläuterungen interessieren:
C-Programme bestehen aus einer Aneinanderreihung von Funktionen. Bei der Abarbeitungsfolge sieht es dann so aus, daß bei Programmstart immer die Funktion main() angesprungen wird; main() muß somit innerhalb eines jeden C-Programmes auftauchen.

Variablen in C können sowohl lokal als auch global definiert sein. Im Beispiel des Programmmes 21 handelt es sich ausschließlich um globale Variablen; sie wurden bereits innerhalb des Programmkopfes vor den einzelnen Funktionen definiert.

Auffällig sind weiterhin die dort zu findende #include wie auch die #define Anweisungen.
Mit include wird dem Präprozessor, einem Teil des Compilers, mitgeteilt, daß er die Datei stdio.h an dieser Stelle in die Textdatei 21 einsetzen soll. Im Falle der Standart Input/Output Datei handelt es sich überwiegend um maschinenspezifische Definitionen, die dafür sorgen, daß der jeweils gleiche Quellcode nach der Compilation auf den verschiedensten Geräten lauffähig ist. Doch können Sie auf diese Art und Weise ebenso ganze Programmteile angeben, von denen Sie bereits wissen, daß sie sich während der Compilation als auch während der Laufzeit völlig korrekt verhalten, und so eine Sourcecode-Bibliothek aufbauen.

Mit #define hingegen definieren Sie symbolische Konstanten, beispielsweise YES als 1 (und somit als wahr) und NO als 0 (was für den ablaufenden Code eine nicht wahre Bedingung bedeutet). Der Präprozessor wird auch für jede so definierte Konstante den entsprechenden Wert innerhalb des Quelltextes einsetzen, wodurch uns die Möglichkeit gegeben ist, für Menschen leichter lesbare und verständlichere Programme zu schreiben. Zur Unterscheidung dieser symbolischen Konstanten von den Variablen sollten auch Sie diese ausschließlich groß schreiben.

```
/* 21 - JW 16.08.1985 */

#include "stdio.h"

#define YES 1
#define NO 0

int ziel,stand,sp,spiel;


main()
{
        hallo();
start:  init();
        if (sp == YES )
        {
                ausgabe();
                player();
        }

        while(stand<ziel)
                {
                ausgabe();
                computer();
                if (stand == ziel)
                        break;
                ausgabe();
                player();
                }
        ende();
     printf("Noch ein Spiel ?\n");
     spiel=getchar();
     if (spiel == 'j')
        {
        goto start;
        }
```

```
hallo()
{
        printf("********** E I N U N D Z W A N Z I G *********\n");
        printf("Ziel des Spieles ist es,  durch Addition  von\n");
        printf("wahlweise 1 oder 2 den  Wert 21 zu erreichen.\n");
}


init()
{
        ziel=21;
        stand=0;
        printf("\n\nWollen Sie beginnen ?");
        spiel=getchar();
          if (spiel == 'j')
               sp = YES;
          else sp = NO;
        printf("\n");
}


ausgabe()
{
        printf("spielstand: %d\n",stand);
}


player()
{
     sp = YES;
        spiel=0;
        printf("\nWollen Sie um 1 oder 2 erhoehen ?\n");
        spiel = (getchar() - '0');
        switch(spiel)
        {
                case 1:
                {
                        printf("\nOkay !\n");
```

```
                    stand++;
                    break;
                }

                case2:
                {
                    printf("\nAuchgut !\n");
                    stand++;
                    stand++;
                    break;
                }

                default:
                {
                    printf("\nSoviel doch wohl nicht!!\n");
                    player();
                }
        }
}

computer()
{
    sp = NO;
      switch(stand)
      {
            case 2:
            case 5:
            case 8:
            case 11:
            case 14:
            case 17:
            case 20:
            {
                pluseins();
                break;
            }
            case 1:
            case 4
            case 7:
```

```
                case 10:
                case 13:
                case 16:
                case 19:
                {
                        printf("\nIch erhoehe um 2.\n");
                        stand++;
                        stand++;
                        break;
                }
                default:
                {
                        pluseins();
                }
        }
}


pluseins()
{
        printf("\nIch erhoehe um 1.\n");
        stand++;
}


ende()
{
        if (sp == YES)
           printf("\n\nHERZLICHEN GLUECKWUNSCH.\n\n");
        else
           printf("\n\nDa war ich wohl klueger ... \n\n");
}
```

4.3. Der nächste Schritt: Eine GEM-Applikation

Nachdem Sie die Sprache C nun kennengelernt haben, und Begriffe wie Include-Dateien und symbolische Konstanten keine Bücher mit sieben Siegeln mehr sind, wollen wir Ihnen nun Schritt für Schritt den Weg zur perfekten GEM-Anwendung weisen.

Wie wir bereits auf den vorstehenden Seiten aufgezeigt haben, stellt das GEM und insbesondere das Virtual Device Interface eine genormte, relativ einfach zu handhabende Schnittstelle zu den verschiedensten grafikfähigen Peripheriegeräten dar. Dabei spielt es keine Rolle, ob ein Programm nun einen Rasterbildschirm, einen grafikfähigen Matrixdrucker oder einen Plotter ansprechen will. Denn sämtliche gerätespezifischen Steuercodes werden bei Bedarf über das VDI ausgeben; der Programmierer ist von dieser Aufgabe freigestellt. Natürlich muß das VDI dazu über die gerätespezifischen Daten, wie kleinster Zeichenschritt in x/y-Richtung, Anzahl der Farbstifte usw., Bescheid wissen.
Zu diesem Zweck können unterschiedliche Gerätetreiber geladen werden, zur Zeit allerdings kennt das GEM des Atari ST nur den Bildschirmtreiber.

Um dem VDI nun mitzuteilen, was man eigentlich von ihm will, bedient man sich einer Subroutine, der man fünf Argumentwerte übergibt, vielmehr sind es fünf Arrays von Daten:

- das Kontroll-Aarray (contrl)
- das Eingabearray (intin)
- das Eingabearray für Punktkoordinaten (ptsin)
- das Ausgabearray (intout)
- das Ausgabearray für Punktkoordinaten (ptsout)

An diesen genau definierten Speicherstellen müssen wir sämtliche für die auszuführende Funktion benötigten Parameter ablegen, anschließend können wir das VDI über seine Einsprungadresse aufrufen.

Maschinenspracheprogrammierer werden diese Werte jeweils 'von Hand' laden müssen, Anwender von Hochsprachen können auf umfangreiche Bibliotheken zurückgreifen, die entsprechende Funktionen anbieten. Wodurch das Verfahren natürlich erheblich vereinfacht und, wie wir später noch sehen werden, sogar teilweise genormt wird.

Alle Arrayelemente sind jeweils zwei Bytes lang, weshalb wir die entsprechenden Variablen in unseren C-Programmen auch immer als Integers definieren werden. In unseren folgenden Beispielprogrammen finden Sie diese Definitionen jeweils gleich zu Beginn der als global deklarierten Variablen.

Der erste Schritt eines jeden GEM-Programmes wird nun daraus bestehen, nach Initialisierung dieser Kontroll-Arrays die Parameter für die benutzte Arbeitsstation einzustellen und diese für den weiteren Gebrauch zu öffnen. Dazu ist die VDI-Routine OPEN WORKSTATION vorgesehen. Ihr Aufruf bewirkt, daß der entsprechende Treiber geladen wird (beim ST vorläufig natürlich noch nicht), daß das Ausgabegerät auf Grafikbetrieb umgestellt wird, und daß es mit bestimmten Werten initialisiert wird. So können wir gleich während des Open-Workstation-Calls festlegen, daß Linien gestrichelt gezeichnet werden, und die benutzte Farbe schwarz sein soll. Für uns als Programmierer bedeutet das nichts anderes, als daß wir in das Kontroll-Array die Parameter der gewünschten Werte schreiben müssen. Dazu gehören:

1. Linientyp (gestrichelt, durchgezogen ...)
2. Farbe von Linienzügen
3. Marker-Typus
4. Farbe der Polymarker
5. Schriftart
6. Schriftfarbe
7. Füllmuster bei Polygonzeichnungen
8. Füllmuster
9. Füllfarbe

Glücklicherweise ist die Normaleinstellung sämtlicher Parameter mit einem Wert von eins definiert, weshalb sich diese Initialisierung bequem in einer Schleife erledigen läßt.
Als Ausnahme gilt lediglich der Wert in int_in von 10, denn mit diesem Parameter legen wir das zu benutzende Koordinatensystem fest. Wie Sie ja wissen, können wir die tatsächlich vorhandenen oder aber auch die normalisierten Koordinaten verwenden. Da wir jedoch Wert auf Schnelligkeit legen und unsere Programme für den Monitor schreiben, legen wir hier mit zwei die Rasterkoordinaten fest (eine Null würde NDC auswählen, eins ist für zukünftige Anwendungen reserviert).

```
open_vwork()
{
int i;
      for (i = 1; i <10; i++){
        int_in[i] = 1;
        }
      int_in[10] = 2;
      v_opnvwk(int_in, &handle, int_out);
}
```

Der Aufruf v_opnvwk(int_in, &handle, int_out) führt schließlich die Initialisierung durch. Als Resultat erhält man zahlreiche Parameter innerhalb des int_out-Arrays, die für uns an dieser Stelle aber nicht weiter von Interesse sind. Wichtig wird für uns nur noch der an handle übergebene Wert. Denn mittels dieses Parameters können wir jeweils ganz genau den für diese Applikation geschaffenen Arbeitsbereich ansprechen.

Wenn Sie sich main() im folgenden Programm ansehen, dann werden Sie jedoch bemerken, daß zuvor noch zwei andere GEM-Calls stattfinden.

Appl_init bereitet ein ähnliches Kontroll-Array für den Gebrauch von AES vor, als Funktionswert erhält man unter

anderem einen Identifikationscode der rufenden Applikation. Dieser Code wird vor allem dann wichtig, wenn mehrere Anwenderprogramme sich den Speicher teilen; diese können dann gezielt über den ap_id angesprochen werden. Graf_handle schließlich liest die Parameter des Desktops aus und kann diese Werte für den späteren Gebrauch bereitstellen; da wir vorläufig jedoch keinen Bedarf an diesen Parametern haben, weisen wir die eingehenden Daten einer Dummyvariablen zu.

Draw() schließlich ist unser eigentliches Hauptprogramm, hier wird stellvertretend für jede noch so komplizierte Anwendung der Umriß eines Hauses gezeichnet.

Was keinem Programm fehlen darf, ist letztendlich der korrekte Abschluß. Die Applikation muß den von ihr belegten Speicherplatz wieder freigeben, und auch die Workstation muß aufgegeben werden, um das Desktop zu reaktivieren.

Anmerkung zu folgendem Listing:

Bis auf draw() sollten Sie sämtliche Teile innerhalb einer eigenen Datei ablegen und diese als Startfile kennzeichnen. Sofern Sie dann später eigene Programme schreiben wollen, bei denen Sie auf den Gebrauch der Fenster verzichten können (z. B. Grafikdemonstrationen), können Sie den Programmcode unverändert übernehmen und Ihre eigenen Routinen ab draw() beginnend einsetzen. Um eine ordnungsgemäße Termination Ihrer Applikation sicherzustellen, sollte Ihre letzte Programmzeile dann

desktop();

lauten.

Außerdem wurde das Listing um die Funktion click() ergänzt, die nichts anderes tut, als auf eine Betätigung der linken Maustaste zu warten. Denn sonst würde Ihnen der ST kaum Zeit lassen, das Kunstwerk zu betrachten.

```
/*************************************************************************/
/**************          programm: HAUS1.C                 *************/
/************** zeichnet Haus und wartet auf linke Maustaste *************/
/**************          JW Oktober 1985                   *************/
/*************************************************************************/
                                /* include files                         */
#include "obdefs.h"             /* erst mal alle - denn wer weiss, was   */
#include "define.h"             /* gebraucht wird ...                    */
#include "gemdefs.h"
#include "osbind.h"
#include "gembind.h"

                                /*globale variablen                      */
int contrl[12];
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];                /* genug Platz fuer alle Faelle          */

int handle,i;                   /* virtual workstation handle            */
int phys_handle;                /* physical workstation handle           */
int pxyarray[12];               /* Array fuer x,y Koordinaten            */
int int_in[11];                 /* Eingabe ins GSX Array                 */
int int_out[57];                /* Ausgabe vom GSX Array                 */

int ap_id;                      /* Kennung der Applikation               */

int dummy;

main()
{
      ap_id=appl_init();        /* initialisiert GEMAES Array-Strukturen */
      handle=graf_handle(&dummy,&dummy,&dummy,&dummy);
                                /* Handhabung des Desktops               */
      open_vwork();             /* Arbeitsbereich erstellen              */
      graf_mouse(256,&dummy);   /* Maus verstecken                       */
      draw();                   /* Kunstwerk erzeugen                    */
      v_gtext(handle,1,350,"Bitte LINKS klicken ...");
```

```
      click();                    /* auf linke Maustaste warten ...      */
      desktop();                  /* Programmende                        */
}


open_vwork()
{
int i;
      for (i = 1; i <10; i++){
        int_in[i] = 1;            /* init int_in array: linetype, farbe, */
        }                         /* fillstyles usw.                     */
      int_in[10] = 2;             /* benutze RC - Koordinaten            */
      v_opnvwk(int_in, &handle, int_out); /* jetzt kanns losgehen ...     */
}


desktop()
{
      v_clsvwk(handle);           /* workstation aufgeben                */
      appl_exit();                /* keine GEM-Calls mehr                */
}


click()                           /* wartet auf Mausklick (links)        */
{
      evnt_button(1,1,1,&dummy,&dummy,&dummy,&dummy);
}


/*---------------- ab hier folgen die Programmteile ----------------------*/


draw()
      {
      int style;                   /* Variable fuer Fuellmuster          */
      style = 3;                   /* Fuellmuster waehlen                */
      pxyarray[0] = 100;           /* x-Koordinate Punkt 1               */
      pxyarray[1] = 100;           /* y-Koordinate Punkt 1               */
```

```
    pxyarray[2] = 100;              /* Punkt 2                              */
    pxyarray[3] = 300;
    pxyarray[4] = 500;              /* Punkt 3                              */
    pxyarray[5] = 300;
    pxyarray[6] = 500;
    pxyarray[7] = 100;
    pxyarray[8] = 300;
    pxyarray[9] = 50;
    pxyarray[10] = 100;
    pxyarray[11] = 100;

    v_pline(handle, 6, pxyarray); /* Polygonzug im Arbeitsbereich:          */
                                  /* 6 Punkte mit Koordianten aus pxyarray*/

    vsf_interior(handle, style);  /* set fill interior style: solid/hollow*/
    v_fillarea(handle, 6, pxyarray); /* von Polygonzug umschlossene Flache*/
                                     /* fuellen                           */
}
```

Damit ist es uns bereits gelungen, eine Anwendung auf dem Desktop laufen zu lassen. Der nächste Schritt, den wir bewältigen müssen, ist nun die Schaffung einer Routine open_window(), die unsdann ein Fenster als Arbeitsbereich bereitstellt.

Machen wir uns zunächst einige Gedanken über die Größe, die unser Fenster haben soll. Für GEM üblich ist die Konvention, bei jedem Objekt gleich welcher Art jeweils die linke obere Ecke als Bezugspunkt in Pixelkoordinaten anzugeben, und relativ zu diesem Punkt dann die Weite und auch Höhe, ebenfalls in Pixeleinheiten, zu nennen.

Doch wer von uns möchte schon Pixel auszählen oder Umrechnungen tätigen. Schließlich ist das Desktop doch auch ein Fenster, und dann noch eins mit maximaler Größe. Warum sollte man zunächst nicht einfach dessen Maße übernehmen, zumal uns VDI mit wind_get eine entsprechende Funktion bereitstellt?
Alles was wir noch wissen müssen, ist die Fensternummer des Desktops, und diese beträgt in jedem Falle null.

Ergänzen wir main() des vorangegangenen Programmes HAUS1 daher um den Aufruf

```
wind_get(0,WF_WORKXYWH,&xdesk,&ydesk,&wdesk,&hdesk);
```

wobei wir natürlich auch nicht vergessen, die neuen Variablen in die Vereinbarungsliste einzutragen:

```
int xdesk, ydesk, wdesk, hdesk;
```

Zur Schaffung unseres Fensters dient die Funktion wind_create(), die als Funktionswert die Nummer des Fensters liefert (wi_handle). Als Funktionsparameter verlangt wind_create zum einen selbstverständlich die Koordinaten nach der üblichen Konvention, weiterhin müssen wir jedoch spezifizieren, mit welchen Kontrollelementen unser Fenster

ausgestattet sein soll. So ist jedem einzelnen Fensterteil und jeder Eigenschaft ein gesetztes Bit einer Binärzahl zugeteilt worden:

0x0001 NAME : Titelzeile mit Namen
0x0002 CLOSER: Schließfeld
0x0004 FULLER: Feld für volle Größe (oben rechts)
0x0008 MOVER: Fenster kann verschoben werden
0x0010 INFO: Infozeile (z.B. 123456 Bytes used in)
0x0020 SIZER: Vergrößerungsfeld (unten links)
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

Wenn Ihr Fenster somit nur aus einer Umrandung der Arbeitsfläche einschließlich einer Titelzeile bestehen soll, so muß Ihr erster Parameter beim Aufruf von wind_create eine 1 sein. Wünschen Sie ebenfalls ein Schließfeld, um Ihre Applikation standesgemäß terminieren lassen zu können, müssen Sie eine drei (1+2; Bits gesetzt: 00000011) programmieren.

Nun ist es jedoch recht umständlich, jedesmal mit Bits hantieren zu müssen. Als Ausweg bietet sich hier der Gebrauch von symbolischen Konstanten an, die zweckmäßigerweise innerhalb einer #include-Datei definiert sind.
Eingebürgert haben sich die oben aufgeführten Symbole, die im Falle des C-Paketes beispielsweise innerhalb der Datei GEMBIND.H definiert sind. Der Programmierer kann dann nur innerhalb seines Programms eine weitere symbolische Konstante benutzen, und dieser stellvertretend die gewünschten Werte zuweisen:

```
#define WI_KIND (SIZER MOVER FULLER CLOSER NAME)
```

Somit kann das Fensterformat problemlos festgelegt werden. Daran anschließend wird der Titel festgelegt, wozu unter anderem die Funktion wind_set() dient, und zum Abschluß kann das Fenster geöffnet werden.
Es empfiehlt sich, diese Anweisungsfolge wiederum innerhalb einer eigenen Funktion abzulegen, die dann problemlos in sämtliche Programme eingebunden (und dank der symbolischen Konstanten leicht für jedes beliebige Fenster verwandt) werden kann:

```
open_window()
{
      wi_handle=wind_create(WI_KIND,xdesk,ydesk,wdesk,hdesk);
      wind_set(wi_handle, WF_NAME, " Tips & Tricks ",0,0);
      wind_open(wi_handle,xdesk,ydesk,wdesk,hdesk);
}
```

Damit haben wir zwar unser Fenster schon recht ordentlich auf dem Schirm, doch lassen sich die meisten Elemente nicht bedienen. Die Abfrage, welche Fensteraktion der Anwender ausführen möchte, ist eigentlich noch ganz einfach. Denn ein wesentlicher Bestandteil der AES-Bibliothek ist die Event-Library. Sie stellt dem Programmierer zahlreiche Routinen zur Abfrage von bestimmten Ereignissen zur Verfügung. Es kann sich dabei um eine Abfrage nach einer Maustastenbetätigung handeln, wie wir sie bereits mit evnt_button in click() kennengelernt haben. Oder aber auch um ein Fensterereignis, daß für die wartende Applikation mittels einer entsprechenden Nachricht (message) näher spezifiziert wird.

Der Aufbau einer solchen Nachricht muß natürlich genau festgelegt sein, was im Falle von GEM auch durch das AES Message Protocol geschehen ist. Dazu wird ein weiteres Array, der Nachrichtenpuffer (msgbuff) benötigt, der eine maximale Länge von acht Einträgen hat. Das Protokoll sieht nun vor, daß die Nachricht selbst in msgbuff(0) mitgeteilt wird, msgbuff(1) enthält den Identifikationscode der Applikation, für die diese Nachricht gilt, und die übrigen Einträge geben Aufschluß über notwendige Parameter.

Selektiert der Anwender beispielsweise mit seiner Maus einen bestimmten Menüeintrag, so schreibt das AES in msgbuff(0) eine 10, den Code für MN_SELECTED. Unter msgbuff(3) kann die Applikation dann die Nummer des betreffenden Menüs (beispielsweise DESK, FILE), und unter msgbuff(4) die Nummer desbetreffendenObjektes(KONTROLLFELD,FORMATIEREN)finden. Stehen die Nummern erst fest, kann mittels einer switch-case Konstruktion schnell die gewünschte Aktion eingeleitet werden.

Vielleicht haben Sie aus der Großschreibung von MN_SELECTED bereits auf die Bezeichnung einer weiteren symbolischen Konstanten, die in irgendeinem Include-File definiert ist, geschlossen. Und in der Tat, auch hier muß man sich nicht mit schnell zu verwechselnden Nummern plagen, sondern kann

leicht zu merkenden, und vor allem im Falle einer späteren Editierung auch leicht zu interpretierende Namen einsetzen:

MN_SELECTED - Menüeintrag gewählt
WM_REDRAW - Das Fenster muß neu gezeichnet werden
WM_TOPPED - Dieses Fenster soll aktiviert werden
WM_CLOSED - Das Schließfeld wurde betätigt
WM_FULLED - Die maximale Größe eingestellen
WM_ARROWED - Ein Pfeil wurde angeklickt
WM_HSLID - Betätigung des horiz. Schiebers
WM_VSLID - Betätigung des senkrechten Schiebers
WM_SIZED - Die Fenstergröße wurde verändert
WM_MOVED - Das Fenster wurde verschoben
WM_NEWTOP - Das Fenster wurde aktiviert
AC_OPEN - Wird an das im Deskmenü gewählte Accessory gesandt
AC_CLOSE - Wird an das stillzulegende Acc. gesandt

Wie Sie sehen, ist diese Liste recht umfangreich. Und sobald Sie wirklich sämtliche Features der GEM-Fenster nutzen wollen, werden Sie auch Routinen zur Behandlung all dieser Ereignisse schreiben müssen.
Doch kann dies glücklicherweise fast immer alles innerhalb eines einzigen Funktionsblockes (mit einer großen switch-Anweisung) geschehen. Sie müssen nur darauf achten, daß Ihre Applikation auch wirklich ununterbrochen auf Ereignisse solcher Art wartet.

Zweckmäßigerweise bedienen Sie sich dazu eines einzigen event_multi()-calls, mit dem Sie gleichzeitig das Eintreffen einer Meldung, ein Mausereignis oder auch eine Betätigung der Tastatur abwarten können. An dieser Stelle möchten wir noch einmal betonen, daß die Ablauffolge wirklich auf diese Art vonstatten geht, also daß Sie GEM immer mitteilen müssen, worauf Sie eigentlich warten. Vergessen Sie die Verfahrensweise, die Sie vielleicht von anderen Computern -

oder auch von reinen TOS-Anwendungen wie 21 her kennen, wo nämlich das Programm auf die Aktion warten mußte.

Denken Sie an diesen Tip, sobald Sie ein Programm geschrieben haben, das scheinbar 'hängt'. Denn sofern die Maus sich noch bewegen läßt - was Sie natürlich nur kontrollieren können, wenn Sie sie nicht unsichtbar gemacht haben - ist Ihr Atari mit irgendwelchen Aktionen beschäftigt und keineswegs abgestürzt. Nur haben Sie dann vergessen, ihm zu sagen daß er auf äußere Reaktionen warten soll. Als einziges Mittel, ihn dann wieder auf andere Wege zu bringen, bleibt Ihnen dann die Reset-Taste.

Daher sollte Ihr Programm innerhalb einer großen Schleife ablaufen, die erst verlassen wird, wenn die Abbruchbedingung erfülltist. Dies kann ein beliebiger Mausklick sein, oder aber auch eine Betätigung des Schließfeldes.
Im letzteren Fall müßte Ihr Programm demnach auf eine Message der Art WM_CLOSED warten:

```
do(
        event_multi(....);

        Fenstersteuerung;

        hier folgt Ihr Programm;

        )while Schließfeld wurde nicht betätigt
```

Dieser Rahmen ist typisch für eine Applikation. Ein Accessory verlangt einen etwas anderen Aufbau, doch davon später mehr. Jetzt wollen wir vielmehr unser Demonstrationsprogramm mit einem entsprechenden Rahmen versehen, so daß Sie das Fenster beliebig vergrößern, verkleinern und auch umherschieben können.

```
/*************************************************************************/
/**************          Programm: HAUS .C                  **************/
/**************       komplette Fenstersteuerung            **************/
/**************             JW Oktober 1985                 **************/
/*************************************************************************/

                                /* include files                          */
#include "obdefs.h"             /* erst mal alle - denn wer weiss, was    */
#include "define.h"             /* gebraucht wird ...                     */
#include "gemdefs.h"
#include "osbind.h"
#include "gembind.h"

                                /* Definitionen fuer spaeteren Gebrauch   */

#define WI_KIND   (SIZER|MOVER|FULLER|CLOSER|NAME)
                                /* Arbeitsfenster: Titel, Schliessfeld...*/
#define MIN_WIDTH  (2*gl_wbox)
#define MIN_HEIGHT (2*gl_hbox)

extern int gl_apid;
                                /*globale variablen                       */
int contrl[12];
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];                /* genug Platz fuer alle Faelle           */

int handle,i;                   /* virtual workstation handle             */
int phys_handle;                /* physical workstation handle            */
int pxyarray[12];               /* Array fuer x,y Koordinaten             */
int int_in[11];                 /* Eingabe ins GSX Array                  */
int int_out[57];                /* Ausgabe vom GSX Array                  */

int wi_handle;                  /* Handhabung des Appl. Fensters          */
int top_window;                 /* Fenster ganz oben                      */
int xdesk, ydesk, wdesk, hdesk; /* Parameter Fenstergroessen              */
int xold, yold, hold, wold;
```

```
int xwork, ywork, hwork, wwork;

int mx, my;                          /* x und y Koordinate der Maus          */
int butdown;

int ap_id;                           /* Kennung der Applikation              */
int menu_id;                         /* Kennung des Arbeitsfensters          */
int fulled;
int hidden;

int msgbuff[8];                      /* event message buffer                 */
int keycode;                         /* enthaelt Zeichencode nach evnt_keybrd */

int gl_wchar, gl_hchar;              /* Hoehe der Zeichen                    */
int gl_wbox, gl_hbox;

int dummy;



/*************************************************************/
/* notwendige Initialisierungen                              */
/*************************************************************/
open_vwork()
{
int i;
      for (i = 1; i <10; i++){
        int_in[i] = 1;               /* init int_in array: linetype, farbe,  */
        }                            /* fillstyles usw.                      */
      int_in[10] = 2;                /* benutze RC - Koordinaten             */
      handle=phys_handle;
      v_opnvwk(int_in, &handle, int_out); /* jetzt kanns losgehen ...        */
}


/*****************************************************************/
/* open window                                                   */
/*****************************************************************/
open_window()
```

```
{
   wi_handle=wind_create(WI_KIND,xdesk,ydesk,wdesk,hdesk);
   wind_set(wi_handle, WF_NAME," Das T&T Haus ",0,0);
   graf_growbox(xdesk+wdesk/2,ydesk+hdesk/2,gl_wbox,gl_hbox,xdesk,ydesk,wdesk,hdesk);
   wind_open(wi_handle,xdesk,ydesk,wdesk,hdesk);
   wind_get(wi_handle,WF_WORKXYWH,&xwork,&ywork,&wwork,&hwork);
}


/*************************************************************/
/* Maus zeigen / verstecken                                  */
/*************************************************************/
show_mouse()
{
      graf_mouse(257,&dummy);
}

hide_mouse()
{
      graf_mouse(256,&dummy);
}


/*****************************************************************/
/* clipping-Parameter setzen                                     */
/*****************************************************************/
set_clip(x,y,w,h)
int x,y,w,h;
{
int clip[4];
   clip[0]=x;
   clip[1]=y;
   clip[2]=x+w;
   clip[3]=y+h;
   vs_clip(handle,1,clip);
}


/*****************************************************************/
```

```
/* Nach Fenstermanipulation neu zeichen                              */
/*********************************************************************/
do_redraw(xc,yc,wc,hc)
int xc,yc,wc,hc;
{
GRECT t1,t2;

   hide_mouse();
   wind_update(TRUE);
   t2.g_x=xc;
   t2.g_y=yc;
   t2.g_w=wc;
   t2.g_h=hc;
   wind_get(wi_handle,WF_FIRSTXYWH,&t1.g_x,&t1.g_y,&t1.g_w,&t1.g_h);
   while (t1.g_w && t1.g_h)
      {
      if (rc_intersect(&t2,&t1))
         {
         set_clip(t1.g_x, t1.g_y, t1.g_w, t1.g_h);
         draw_haus();
         }
      wind_get(wi_handle,WF_NEXTXYWH,&t1.g_x,&t1.g_y,&t1.g_w,&t1.g_h);
      }
     wind_update(FALSE);
     show_mouse();
}


/*********************************************************************/
/*  Abfrage von Ereignissen: Window, Maus, Keyboard                  */
/*********************************************************************/
multi()
{
int event;

      do{
      event = evnt_multi(MU_MESAG | MU_BUTTON | MU_KEYBD,
         1,1,butdown,
         0,0,0,0,0,
```

```
        0,0,0,0,0,
        msgbuff,0,0,&mx,&my,&dummy,&dummy,&keycode,&dummy);


/*****************************************************************/
/* WINDOW(): Fensterbehandlung: schieben, vergr. usw.            */
/*****************************************************************/

   wind_update(TRUE);

   if (event & MU_MESAG)
     switch (msgbuff[0]) {

     case WM_REDRAW:
       do_redraw(msgbuff[4],msgbuff[5],msgbuff[6],msgbuff[7]);
       break;

     case WM_NEWTOP:
     case WM_TOPPED:
       wind_set(wi_handle,WF_TOP,0,0,0,0);
       break;

     case WM_SIZED:
     case WM_MOVED:
       if(msgbuff[6]<MIN_WIDTH)msgbuff[6]=MIN_WIDTH;
       if(msgbuff[7]<MIN_HEIGHT)msgbuff[7]=MIN_HEIGHT;
       wind_set(wi_handle,WF_CURRXYWH,msgbuff[4],msgbuff[5],msgbuff[6],msgbuff[7]);
       wind_get(wi_handle,WF_WORKXYWH,&xwork,&ywork,&wwork,&hwork);
       break;

     case WM_FULLED:
       if(fulled){
      wind_calc(WC_WORK,WI_KIND,xold,yold,wold,hold,
            &xwork,&ywork,&wwork,&hwork);
      wind_set(wi_handle,WF_CURRXYWH,xold,yold,wold,hold);}
       else{
      wind_calc(WC_BORDER,WI_KIND,xwork,ywork,wwork,hwork,
            &xold,&yold,&wold,&hold);
      wind_calc(WC_WORK,WI_KIND,xdesk,ydesk,wdesk,hdesk,
```

```
            &xwork,&ywork,&wwork,&hwork);
        wind_set(wi_handle,WF_CURRXYWH,xdesk,ydesk,wdesk,hdesk);
         }
         fulled ^= TRUE;
         break;

      } /* switch (msgbuff[0]) */

    if ((event & MU_BUTTON)&&(wi_handle == top_window))
      if(butdown) butdown = FALSE;
      else butdown = TRUE;

      if(event & MU_KEYBD){
         do_redraw(xwork,ywork,wwork,hwork);
      }

    wind_update(FALSE);

      }while(!((event & MU_MESAG) && (msgbuff[0] == WM_CLOSED)));
                                /* Schliessfeld wurde betaetigt          */
       wind_close(wi_handle);
       graf_shrinkbox(xwork+wwork/2,ywork+hwork/2,gl_wbox,gl_hbox,xwork,ywork,wwork,hwork);
       wind_delete(wi_handle);      /* Speicher freigeben                   */
       v_clsvwk(handle);            /* workstation aufgeben                 */
       appl_exit();                 /* und zum Desktop                      */

}


main()
{
       ap_id=appl_init();           /* initialisiert GEMAES Array-Strukturen */
       phys_handle=graf_handle(&gl_wchar, &gl_hchar, &gl_wbox, &gl_hbox);
                                    /* Handhabung des Desktops              */
       wind_get(0,WF_WORKXYWH, &xdesk, &ydesk, &wdesk, &hdesk);
       open_vwork();                /* Arbeitsbereich erstellen             */
       open_window();               /* Applikationsfenster oeffnen          */
       graf_mouse(ARROW,&dummy);    /* Mausform                             */
       v_gtext(handle,10,gl_hchar*3,"Das ist das T&T Haus.");
```

```
      hidden=FALSE;
      fulled=FALSE;
      butdown=TRUE;
      multi();                        /* Was macht der Anwender ?              */

}



/*---------------- ab hier folgen die Programmteile -----------------------*/


draw_haus()
      {
      int style;                      /* Variable fuer Fuellmuster             */
      style = 3;                      /* Fuellmuster waehlen                   */
      pxyarray[0] = 100;              /* x-Koordinate Punkt 1                  */
      pxyarray[1] = 100;              /* y-Koordinate Punkt 1                  */
      pxyarray[2] = 100;              /* Punkt 2                               */
      pxyarray[3] = 300;
      pxyarray[4] = 500;              /* Punkt 3                               */
      pxyarray[5] = 300;
      pxyarray[6] = 500;
      pxyarray[7] = 100;
      pxyarray[8] = 300;
      pxyarray[9] = 50;
      pxyarray[10] = 100;
      pxyarray[11] = 100;

      v_pline(handle, 6, pxyarray); /* Polygonzug im Arbeitsbereich:         */
                                    /* 6 Punkte mit Koordianten aus pxyarray*/

      vsf_interior(handle, style);  /* set fill interior style: solid/hollow*/
      v_fillarea(handle, 6, pxyarray); /* von Polygonzug umschlossene Flache*/
}
```

4.3.1 PRINIT - eine Beispielapplikation

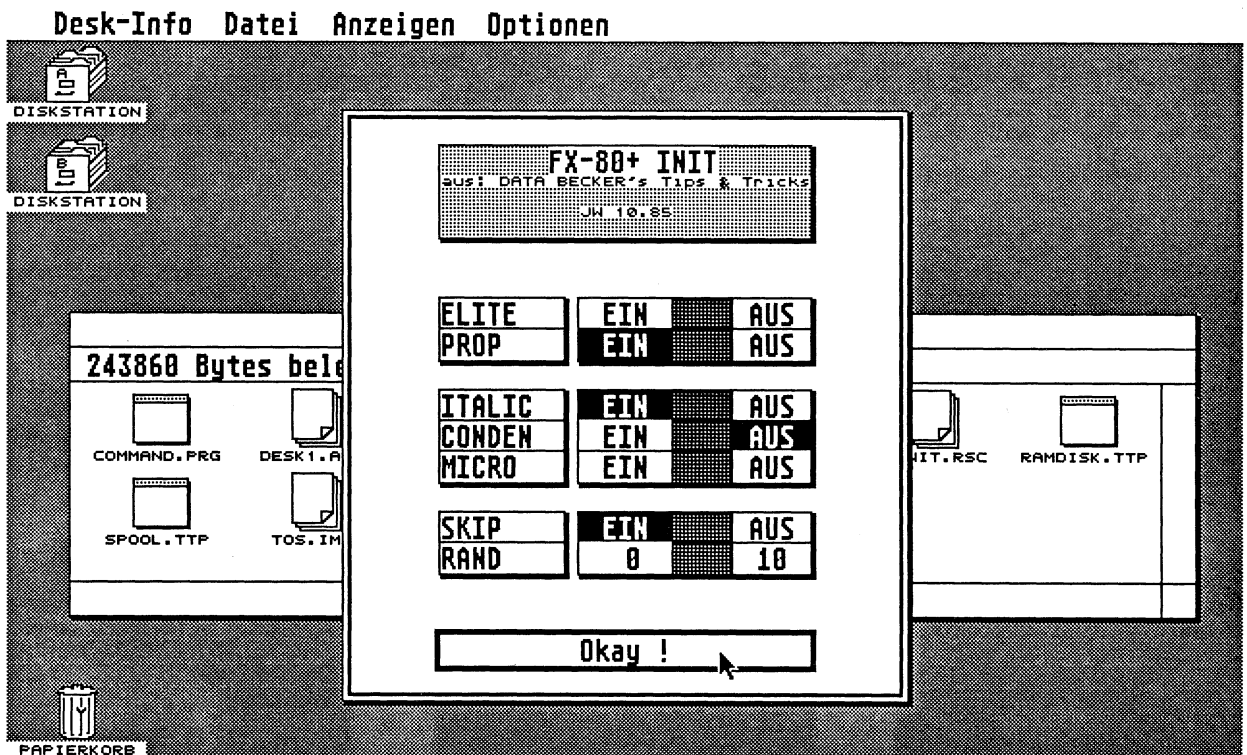
Nachdem die Vorarbeit nun erledigt ist, wollen wir uns mit der ersten sinnvollen Applikation befassen - natürlich nicht ohne den Hintergedanken, diese später als Accessory innerhalb von DESK auf der Menüleiste zu installieren.

Es soll sich dabei um PRINIT, ein kurzes Programm zur Druckereinstellung, handeln. Dies ist als Accessory sicher sinnvoll, denn wie oft muß man den Drucker auf eine anderes Schriftart umstellen und dazu erst Basic laden, um eine Folge von CHR$ abzusenden. Hat man dagegen PRINIT, so klickt man einmal ins Feld ELITE, ein zweites Mal in RAND10, dann noch in die Box OKAY und schon kann man jedes C-Listing sauber ausdrucken lassen.

Solch ein Auswahlmenü erlaubt natürlich herrlich einfaches Arbeiten, doch will solch eine 'Dialogbox' erst einmal geschaffen sein. Was uns dann interessiert, ist die Frage, woher wir wissen, über welchem Objekt dieser Box sich der Mauszeiger gerade befindet.

Nun, um die zweite Frage beantworten zu können, müssen wir uns zunächst erst einmal der ersten zuwenden. GEM erwartet all diese Resourcen in wohlstrukturierter Form, und zwar in Form eines Baumes. Das heißt, für jedes Objekt gibt es irgendwo eine Wurzel, und von diesem Punkt können dann zahlreiche Verzweigungen ausgehen, bis schließlich ein ganzer Menübaum voller Objekte entstanden ist.

Stellen Sie sich nun einfach einmal ein großes Rechteck vor, das zwei kleine, untereinanderstehende Rechtecke enthält. Das untere dieser beiden Rechtecke soll dann auch noch irgendeinen Text enthalten.
Die Wurzel dieses Baumes wäre dann die äußere große Box. Sie erscheint durchsichtig und kann durch die Angabe ihrer typischen Daten beschrieben werden. Dazu gehören zunächst einmal die üblichen Koordinaten, also x- und y-Position der linken oberen Ecke sowie Breite und Höhe. Weiterhin gehört


Desk-Info Datei Anzeigen Optionen
DISKSTATION
DISKSTATION
243860 Bytes bele
COMMAND.PRG
DESK1.A
SPOOL.TTP
TOS.IM
NIT.RSC
RAMDISK.TTP
PAPIERKORB
FX-80+ INIT
aus: DATA BECKER's Tips & Tricks
JW 10.85
ELITE EIN AUS
PROP EIN AUS
ITALIC EIN AUS
CONDEN EIN AUS
MICRO EIN AUS
SKIP EIN AUS
RAND 0 10
Okay !

zur Beschreibung einer jeden Box aber auch eine Angabe über die Zeichenstärke ihres Rahmens wie auch über die Vorder- und Hintergrundfarbe.
Nehmen Sie es im Moment bitte einfach hin, daß solch eine durchsichtige Box einfach G_BOX heißen soll.

Wenn wir, wie bei solchen Konstruktionen üblich, diese Box einmal als PARENT (also als Elternteil) betrachten, dann dürfte CHILD (also Abkömmling) eine jede der inneren Boxen genannt werden. Die obere dieser beiden sollte einfach leer sein, nach allem was wir nun wissen, müßte es demnach ebenfalls eine G_BOX sein. Die untere hingegen sollte eine Box mit Text sein; sie wird deshalb mit dem Typ G_BOXTEXT bezeichnet.

Sobald man sich als Programmierer über den Aufbau seiner Menüstrukturen klar ist und eine Liste aller Daten, die zur Beschreibung der einzelnen Objekte notwendig sind, erstellt hat, könnte man all diese Werte innerhalb das Programmes niederlegen und den Baum zur Laufzeit mittel OBJC_ADD konstruieren lassen.
Anschließend würde ein OBJC_DRAW den betreffenden Baum auf dem Ausgabegerät erscheinen lassen. Doch zählen Sie einmal nach, aus wie vielen einzelnen Objekten der zuvor abgebildete Dialogbaum zu PRINIT besteht: Es sind genau 34. Wieviel Zeit würde es allein kosten, um alle notwendigen Angaben exakt zu bestimmen?

Schnelle Hilfe verspricht hier das Resource Construction Set. Mit diesem Hilfsprogramm lassen sich sämtliche benötigten Menüstrukturen in Windeseile erstellen - und später gegebenenfalls auch wieder editieren. Denn das RCS erzeugt die .RSC-Files, die alle benötigten Angaben enthalten. Diese werden dann zur Laufzeit des Programmes mittels der Funktion rsrc_load(filename) in einen dafür vorgesehenen Speicherbereich geladen.

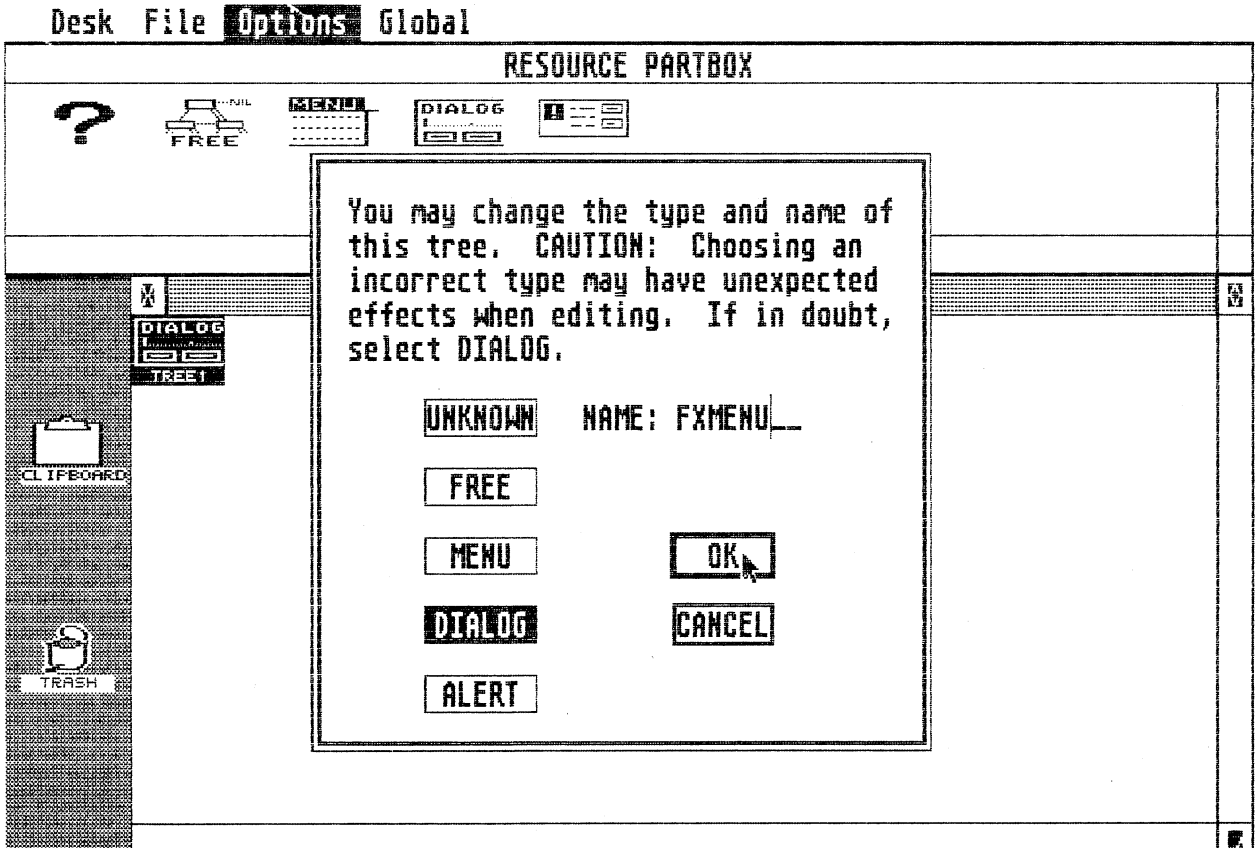
Im folgenden wollen wir Ihnen alle notwendigen Schritte aufzeigen, die zur Konstruktion der benötigten PRINIT.RSC-Datei führen.

4.4 Konstruktion einer RSC-Datei

Nachdem das RCS geladen wurde, finden Sie auf dem Bildschirm zunächst zwei Fenster. Im oberen, der RESOURCE PARTBOX, werden jeweils sämtliche zur Verfügung stehenden Bauteile abgebildet. Nach der Initialisierung kann es sich natürlich zunächst nur um die verschiedenen Baumstrukturen handeln.
Sie müssen sich nun entscheiden, ob Sie ein Menü aufbauen wollen, das später innerhalb der DESKTOP-Menüleiste erscheint, oder ob Sie einen Dialogbaum aufbauen wollen, innerhalb dessen der Anwender zwischen mehrere Alternativen wählen kann.
Diese beiden Typen dürften die am häufigsten verwandten sein, daneben gibt es noch den Alert-Tree, der dem Dialogbaum rechtähnlich ist, aber nicht so viele Optionen offen läßt. Außerdem kennt das RCS noch den Baum FREE, der dem Programmierer bei seiner Gestaltung kaum Restriktionen auferlegt. Die einzige Bedingung, die bei diesem Baum noch gilt, ist die Regel, daß kein Objekt aus einem anderen herausragen kann, während die anderen auch noch gewisse Formatierungsregeln beachten.

Der mit einem Fragezeichen symbolisierte Baum schließlich ist nur ein Platzhalter bis zu dem Zeitpunkt, an dem der Programmierer endlich weiß, was er tut, und ihn entsprechend benennt. Sollte jedenfalls irgendwo innerhalb des Resourcefiles noch ein Baum vom Typ unbekannt stehen, können Sie getrost mit einem Absturz des Programmes rechnen.

Wir werden gemäß dieser Erläuterungen daher das Symbol für einen Dialogbaum anklicken und eine Kopie dessen in das untere Fenster 'draggen', also bei niedergehaltenem Mausknopf dorthin schieben. Das RCS meldet sich sogleich mit

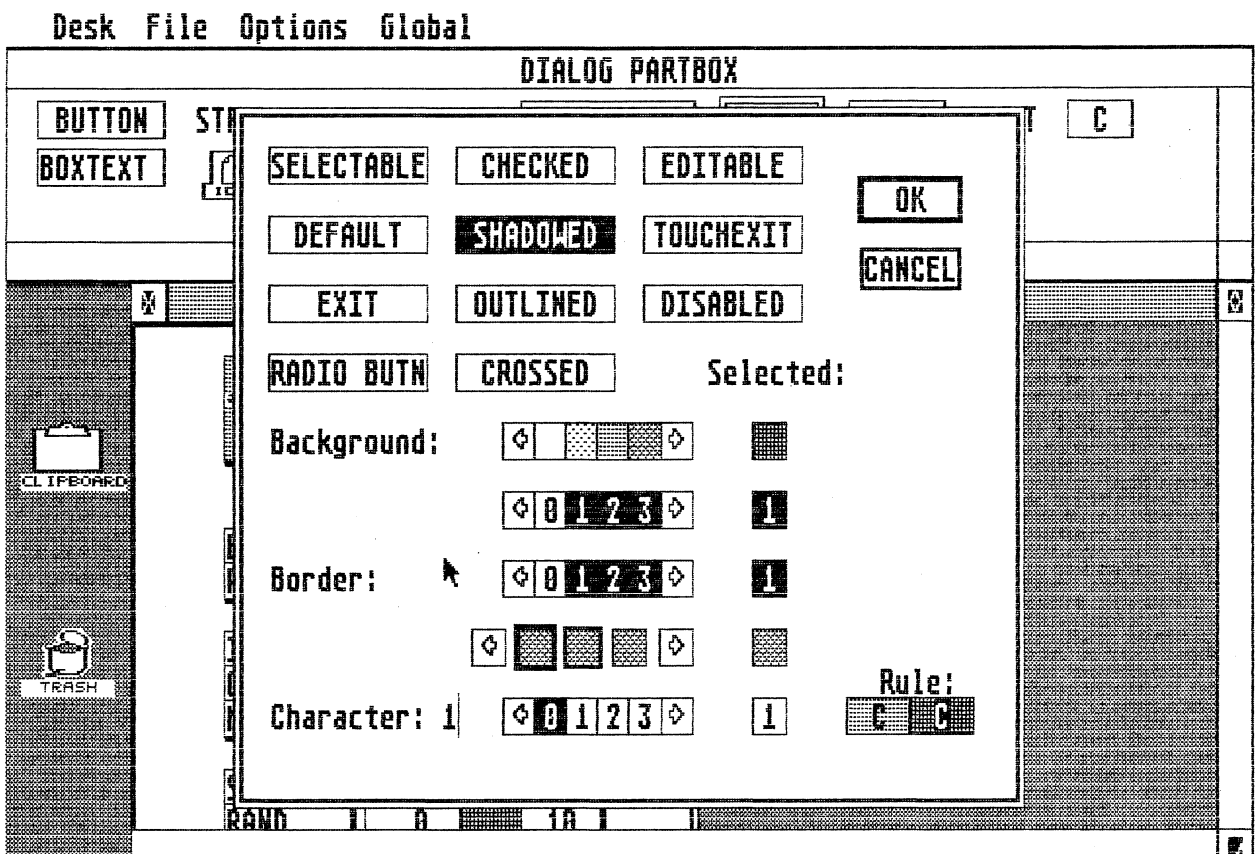

Desk File Options Global
RESOURCE PARTBOX
FREE
MENU
DIALOG
You may change the type and name of this tree. CAUTION: Choosing an incorrect type may have unexpected effects when editing. If in doubt, select DIALOG.
UNKNOWN
NAME: FXMENU
FREE
MENU
DIALOG
ALERT
OK
CANCEL
DIALOG
TREE1
CLIPBOARD
TRASH

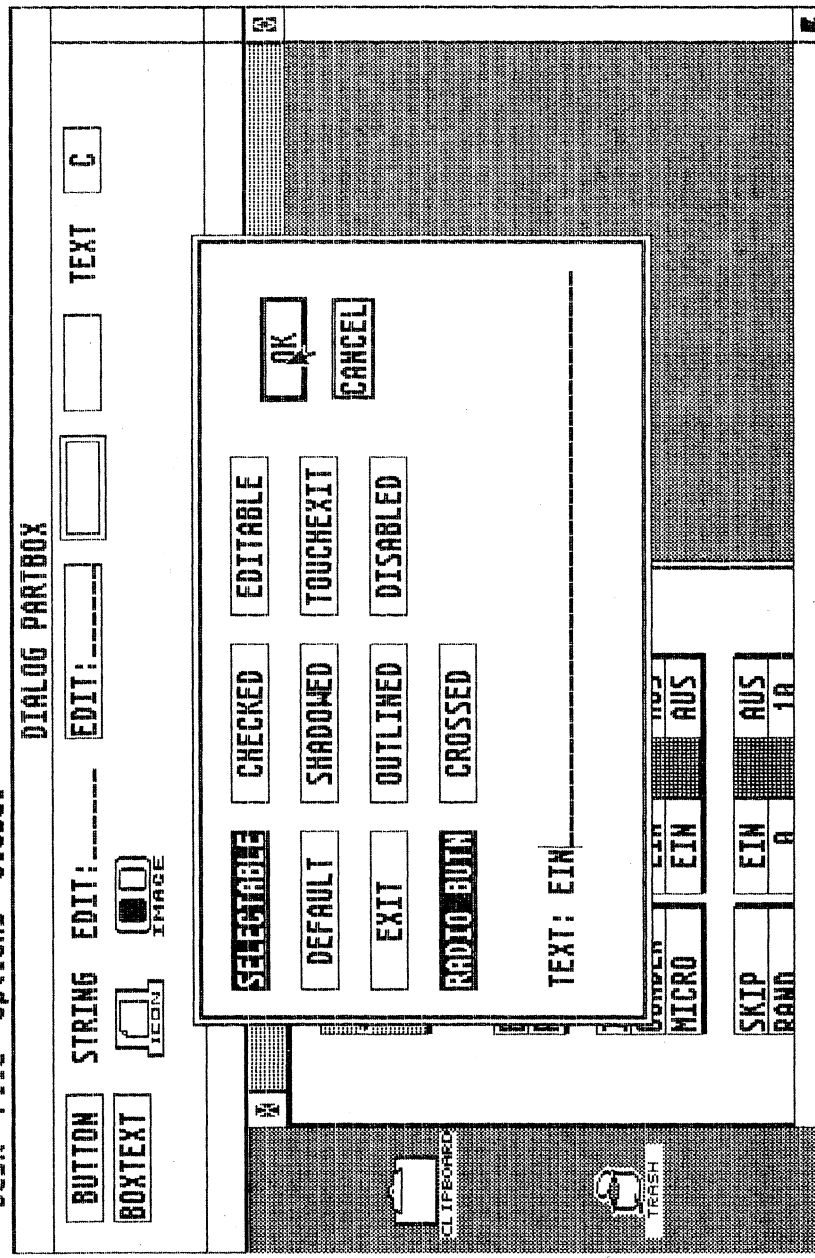
einer eigenen Dialogbox und fordert uns auf, unseren Baum zu benennen. Geben Sie FXMENU ein, und bestätigen Sie mit einem Druck auf Return oder einem Klick in die Okay-Box.

Anschließend bewegen Sie den Mauszeiger in das untere Fenster und betätigen den linken Maustaster einmal, um es zu aktivieren. Führen Sie dann einen Doppelklick auf dem mit FXMENU bezeichnetem Dialogbaum aus.

Wählen Sie dann aus dem oberen Fenster den Baustein BOXTEXT aus und schieben Sie eine Kopie in das untere Fenster FXMENU. BOXTEXT, eine einfache Box, die einen Text enthält, benutzen wir, um die verschiedenen Optionen zu benennen (ELITE, ITALIC ...), wir werden demgemäß mindestens sieben Boxen dieses Typs brauchen. Nun ist es aber recht umständlich, jedesmal aufs neue eines der beiden Fenster zu aktivieren, weshalb wir von einer Kopieroperation des RCS Gebrauch machen. Dazu müssen Sie die bereits im unteren Fenster vorhandene Box anklicken und an die gewünschte Position 'draggen', diesmal dabei aber zusätzlich eine der Shift-Tasten niederhalten. Wiederholen Sie diesen Vorgang, bis dieAnzahl von BOXTEXT-Boxen ausreichend ist.

Jeder der EIN/AUS-Schalter des FXMENÜS besteht zunächst erst einmal aus einer großen BOX (das dritte Element von rechts innerhalb der Partbox). Bringen Sie daher eine solche in das untere Fenster, und vergrößern Sie diese wie sonst auch die Fenster auf das gewünschte Format.
Führen Sie nun einen Doppelklick auf diese BOX aus oder wählen Sie Open innerhalb des File-Menüs. Das RSC präsentiert Ihnen nun eine weitere Dialogbox, innerhalb der Sie die Parameter für das Aussehen dieser Box gemäß folgender Abbildung einstellen. Jedes EIN bzw. AUS wird durch einen BUTTON dargestellt. Wählen Sie in der Partbox einen BUTTON, und positionieren Sie diesen auf einer Seite innerhalb der zuvor erzeugten Box. Anschließend schieben Sie eine Kopie dieses zuvor noch auf seine korrekte Größe gebrachten Buttons auf die andere Seite.


Desk File Options Global
DIALOG PARTBOX
BUTTON
BOXTEXT
C
SELECTABLE
CHECKED
EDITABLE
DEFAULT
SHADOWED
TOUCHEXIT
EXIT
OUTLINED
DISABLED
RADIO BUTN
CROSSED
OK
CANCEL
Selected:
Background:
Border:
Character: 1
Rule:
CLIPBOARD
TRASH


Desk File Options Global
DIALOG PARTBOX
BUTTON
BOXTEXT
STRING
EDIT:
EDIT:
TEXT
C
ICON
IMAGE
SELECTABLE
DEFAULT
EXIT
RADIO BUTN
CHECKED
SHADOWED
OUTLINED
CROSSED
EDITABLE
TOUCHEXIT
DISABLED
OK
CANCEL
TEXT: EIN
MICRO
EIN
AUS
SKIP
EIN
AUS
CLIPBOARD
TRASH

Öffnen Sie nun per Doppelklick die beiden Buttons, und geben Sie als Text EIN bzw. AUS ein. Außerdem sollten Sie SELECTABLE und RADIO BUTN wählen.

Nachdem auch dies geschehen ist, kopieren Sie die ganze Konstruktion in ausreichendem Maße.

Als letztes wichtiges Kontrollelement schaffen Sie sich einen weiteren BUTTON, weisen diesem den Text 'Okay !' zu und definieren ihn als SELECTABLE, DEFAULT und EXIT.

Damit ist der wesentliche Aufbau erledigt, abgesehen vom Titelfeld das ebenfalls aus einer großen BOX besteht, die dann mit drei Elementen vom Typ TEXT ausgestattet wurde.

Jetzt müssen wir die Referenzen erzeugen, damit unser Programm später genau weiß, welche Box denn nun angewählt wurde. Dazu dient die Funktion NAME innerhalb des OPTIONS-Menüs der Titelzeile.
Selbstverständlich müssen Sie dabei nur denjenigen Objekten einen Namen zuweisen, denen später eine Programmfunktion zugeordnet werden soll. In unserem Falle haben wir die sinngemäßen Bezeichnungen ELITEEIN, ELITEAUS, PROPEIN, PROPAUS, ITALEIN, ITALAUS, CONDENEI, CONDENAU, MICROEIN, MICROAUS, SKIPEIN, SKIPAUS, RAND0, RAND 10 und EXIT für das Okay!-Feld verwandt.

Gehen Sie abschließend noch ins File-Menü, und veranlassen Sie das RCS, mit der Option Save As das ganze unter dem Namen 'PRINIT.RSC' abzuspeichern.
Klicken Sie dann auf das Schließfeld des unteren Fensters, bis das RCS schließlich die gewünschten Dateien erzeugt hat (dann ist das View-Window wieder leer!).

Auf Ihrer Diskette werden Sie dann folgende Dateien finden:

PRINIT.RSC - das Resourcefile für das folgende Programm
PRINIT.H - eine include-Datei mit sämtlichen
symbolischen Konstanten
PRINIT.DEF - eine RCS-eigene Hilfsdatei

Nachdem wir Sie bislang im Gebrauch des RCS unterwiesen haben, wollen wir Ihnen abschließend noch kurz eine Aufstellung über sämtliche vorhandenen Bauelemente wie auch über deren optionale Parameterbelegung geben.

Da wären zunächst einmal die Objekt-Typen zu nennen. Bei den meisten Objekten, die Sie zur Konstruktion Ihrer Resourcen verwenden werden, wird es sich vermutlich um die BOXEN handeln:

G_IBOX, G_BOX leere Boxen
G_BOXCHAR enthält ein einziges Schriftzeichen

Sobald Sie Texte in irgendeiner Form in die RSC-Datei einbinden müssen, können Sie auf folgende Text Objekte zurückgreifen, die allesamt die Zeichen des normalen Satzes unter Berücksichtigung von Vor- und Hintergrundfarbe darstellen können:

G_STRING ein text-string
G_BUTTON ein von einer Box umschlossener String
G_TITLE ein String innerhalb einer Menüleiste

Außerdem kennt das RCS formatierte Texttypen, die ihren Einsatz bei zu editierenden Meldungen finden (z. B. im File-Selektions-Menü):

G_TEXT ist ein formatierter Text
G_BOXTEXT formatierter Text innerhalb einer Box
G_FTEXT editierbarer Text

G_FBOXTEXT editierbarer Text innerhalb einer Box

Nachdem Sie die gewünschten Objekte in Ihren Baum eingebracht haben, werden Sie meist auch noch den Objekt-Status festlegen und einige Flags setzen müssen:

Selected zeichnet ein Objekt revers
Crossed durchkreuzt eine Box
Checked ein Häkchen wird an der linken Seite des Objektes angezeigt

Disabled das Objekt wird mit halber Intensität dargestellt
Outlined das Objekt erhält zusätzlich eine Umrandung (nicht zusammen mit Shadowed)
Shadowed ein Schattenriß der so definierten Box wird gezeichnet (nicht mit Outlined)

Selectable das so bezeichnete Objekt kann während des Programmlaufes aktiviert werden
Default eine Betätigung der Return-Taste selektiert dieses Objekt; es wird mit einer stärkeren Umrandung dargestellt
Exit beeendet einen Dialog
Editable das Objekt enthält editierbaren Text
Rbutton das Objekt gehört zu einem Verband, aus dem jeweils immer nur ein Objekt selektiert werden kann
Hidetree das so spezifizierte Objekt wird bei einem Objc_Draw-Aufruf nicht gezeichnet
Touchexit sobald der Mauszeiger sich über einem solchen Objekt befindet, gilt der Dialog als beendet (auch ohne Klickoperation)

Auf den folgenden Seiten finden Sie nun das Listing des dazugehörigen Programmes. Es ist anzumerken, daß die .H Datei hier bereits mittels einer Editorfunktion in die Hauptdatei eingebunden wurde.
Sofern Sie mittels des Resource Construction Sets ein eigenes RSC-File erzeugt haben, sollten Sie unbedingt darauf verzichten, die Ziffern zu jedem Symbol abzuschreiben. Sehen Sie sich Ihr eigenes .H-File an, und setzen Sie stattdessen Ihre eigenen Zahlenwerte ein!

```
#define FXMENU 0        /* TREE */
#define EXIT 5          /* OBJECT in TREE #0 */
#define ELITEEIN 7      /* OBJECT in TREE #0 */
#define PROPEIN 10      /* OBJECT in TREE #0 */
#define PROPAUS 11      /* OBJECT in TREE #0 */
#define ITALEIN 13      /* OBJECT in TREE #0 */
#define ITALAUS 14      /* OBJECT in TREE #0 */
#define CONDENEI 16     /* OBJECT in TREE #0 */
#define CONDENAU 17     /* OBJECT in TREE #0 */
#define MICROEIN 19     /* OBJECT in TREE #0 */
#define MICROAUS 20     /* OBJECT in TREE #0 */
#define SKIPEIN 22      /* OBJECT in TREE #0 */
#define SKIPAUS 23      /* OBJECT in TREE #0 */
#define RAND0 26        /* OBJECT in TREE #0 */
#define RAND10 25       /* OBJECT in TREE #0 */
#define ELITEAUS 8      /* OBJECT in TREE #0 */
```

```
/**************                PROGRAMM: PR-INIT                 *************/
/********      Einstellprogramm fuer Drucker am Parallelport       **********/
/**************       (c) J. Walkowiak, Oktober 1985            *************/
/*****************************************************************************/
#include "obdefs.h"                 /* Objektdefinitionen                    */
#include "gemdefs.h"                /* Definitionen zu GEM                   */
#include "define.h"
#include "osbind.h"
#include "gembind.h"
#include "vdibind.h"
#include "portab.h"

/*****************************************************************************/
/*                                    Definitionen zum RSC-File              */
/*****************************************************************************/
#define FILENAME "PRINIT.RSC"       /* Name der RSC-Datei                    */
#define MAX_DEPTH 34                /* Anzahl aller Objekte, Zeichentiefe    */
#define FXMENU 0                    /* TREE                                  */
#define EXIT 5                      /* OBJECT in TREE #0                     */
#define ELITEEIN 7                  /* OBJECT in TREE #0                     */
#define ELITEAUS 8                  /* OBJECT in TREE #0                     */
#define PROPEIN 10                  /* OBJECT in TREE #0                     */
#define PROPAUS 11                  /* OBJECT in TREE #0                     */
#define ITALEIN 13                  /* OBJECT in TREE #0                     */
#define ITALAUS 14                  /* OBJECT in TREE #0                     */
#define CONDENEI 16                 /* OBJECT in TREE #0                     */
#define CONDENAU 17                 /* OBJECT in TREE #0                     */
#define MICROEIN 19                 /* OBJECT in TREE #0                     */
#define MICROAUS 20                 /* OBJECT in TREE #0                     */
#define SKIPEIN 22                  /* OBJECT in TREE #0                     */
#define SKIPAUS 23                  /* OBJECT in TREE #0                     */
#define RAND0 26                    /* OBJECT in TREE #0                     */
#define RAND10 25                   /* OBJECT in TREE #0                     */

long menu_tree;                     /* Adresse des gew. RSC-Objektes         */
```

```
/*****************************************************************************/
/*                                Definitionen BUTTON-Arten in Menues        */
/*****************************************************************************/
#define SELECTED 0x0001
#define NORMAL 0x0000
#define WI_KIND 0x0001          /* Fenster hat nur Namenszeile               */
/*****************************************************************************/
/*                                Steuercodes des Druckers                   */
/*                                 hier: EPSON FX-80+                        */
/*****************************************************************************/
#define RET 13                  /* Return                                    */
#define ESC 27                  /* Escape                                    */
#define SCHMAL 15               /* Schmalschrift                             */
#define SCHMALOFF 18
#define ELITE 77                /* Elite                                     */
#define ELITEOFF 80
#define PROPORTIONAL 112        /* Proportionalschrift:                      */
#define PSET 1                  /*  ein                                      */
#define PRESET 0                /*  aus                                      */
#define ITALIC 4                /* Schraegschrift                            */
#define ITALICOFF 5
#define MICRO1 83               /* Superscript1                              */
#define MICRO2 0
#define MICROOFF 84
#define SKIP 78                 /* Skip over Perforation                     */
#define SKIPOFF 79
#define LRAND 108               /* linken Rand setzen                        */
#define LAUSSEN 0               /*  nach links aussen                        */
#define POS10 10                /*  drucke ab Position  10                   */

/*****************************************************************************/
/*                                globale Variablen                          */
/*****************************************************************************/
int contrl[12];                 /* Kontroll-Arrays                           */
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];                /* jeweils genug Platz fuer alle Faelle      */
```

```
int pxyarray[12];               /* Array fuer x,y Koordinaten          */

int int_in[11];                 /* Eingabe ins GSX Array               */
int int_out[57];                /* Ausgabe vom GSX Array               */


int handle,i;                   /* virtual workstation handle          */
int phys_handle;                /* physical workstation handle         */
int wi_handle;                  /* Window handle                       */

int ap_id;                      /* Kennung der Applikation             */

int gl_hchar, gl_wchar;         /* Hhe und Weite der Zeichen           */
int gl_wbox, gl_hbox;

int xwork,ywork,wwork,hwork;    /* Arbeitsgroesse Fenster              */
int xdesk,ydesk,wdesk,hdesk;    /* Groesse Desktop                     */
int xold, yold, hold, wold;     /* Hilfsvariablen bei Fenstermanipulation*/
int xobj,yobj,wobj,hobj;        /* Groesse eines Objektes              */
int mausx, mausy;               /* wo ist die Maus, wenn was passiert ?  */

int dummy;                      /* ... fuer ueberfluessige Parameter   */

int event;                      /* welches Eingabegeraet hat was zu melden*/
int title, item;                /* Menuetitel und aktuelles Objekt     */



/************************************************************************/
/*                                Fenster oeffnen, schlieen             */
/************************************************************************/
open_window()
{
   wi_handle=wind_create(WI_KIND,xdesk,ydesk,wdesk,hdesk);
   graf_growbox(xdesk+wdesk/2,ydesk+hdesk/2,gl_wbox,gl_hbox,xdesk,ydesk,wdesk,hdesk);
   wind_open(wi_handle,xdesk,ydesk,wdesk,hdesk);
   wind_get(wi_handle,WF_WORKXYWH,&xwork,&ywork,&wwork,&hwork);
}
```

```
close_window()
{
   wind_close(wi_handle);
   graf_shrinkbox(xwork+wwork/2,ywork+hwork/2,gl_wbox,gl_hbox,xwork,ywork,wwork,hwork);
   wind_delete(wi_handle);
}

open_vwork()
{
int i;
      for (i = 1; i <10; i++){
        int_in[i] = 1;              /* init int_in array: linetype, farbe,   */
        }                           /* fillstyles usw.                       */
      int_in[10] = 2;               /* benutze RC - Koordinaten              */
      handle=phys_handle;
      v_opnvwk(int_in, &handle, int_out); /* jetzt kanns losgehen ...        */
}


/*************************************************************************/
/*                                  Hauptprogramm                        */
/*************************************************************************/
main()
{
int ende;                           /* ist TRUE wenn EXIT angeklickt         */
long  gemdos();                     /* fuer gemdos-call                      */

      ap_id=appl_init();            /* initialisiert GEMAES Array-Strukturen */

      phys_handle=graf_handle(&gl_wchar,&gl_hchar,&gl_wbox,&gl_hbox);
                                    /* Parameter des Desktops ermitteln      */
      wind_get(0,WF_WORKXYWH,&xdesk,&ydesk,&wdesk,&hdesk);
      open_vwork();                 /* Arbeitsstation oeffnen                */
      if(!rsrc_load(FILENAME))      /* RSC-file laden                        */
        {
        form_alert(1,"[3][Raubkopierer, was?|PRINIT.RSC|kann ich nicht finden.][Abort]");
        close_window;
        desktop();
        }
```

```
if(rsrc_gaddr(0,0,&menu_tree)== 0)
   {
   form_alert(1,"[3] [Schwerer Fehler!|Resource File nicht OK.][Abort]");
   close_window;
   desktop();
   }
rsrc_gaddr(R_TREE,FXMENU,&menu_tree);
form_center(menu_tree,&xobj,&yobj,&wobj,&hobj);
form_dial(0,xobj,yobj,wobj,hobj);
form_dial(1,1,1,1,1,xobj,yobj,wobj,hobj);

objc_draw(menu_tree,0,MAX_DEPTH,0,0,wdesk,hdesk);

graf_mouse(3,&dummy);          /* Maus = Hand                          */
while (ende != TRUE){
    event=evnt_button(1,1,1,&mausx,&mausy,&dummy,&dummy);
                               /* Warten auf Einzelclick links         */

    item=objc_find(menu_tree,FXMENU,13,mausx,mausy);
                               /* welches Objekt in menu_tree an Mauspos*/

    switch(item){
         case ELITEEIN:
         objc_change(menu_tree,ELITEEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
         objc_change(menu_tree,ELITEAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
         gemdos(0x5,ESC);
         gemdos(0x5,ELITE);
         break;

         case ELITEAUS:
         objc_change(menu_tree,ELITEAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
         objc_change(menu_tree,ELITEEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
         gemdos(0x5,ESC);
         gemdos(0x5,ELITEOFF);
         break;

         case CONDENEI:
         objc_change(menu_tree,CONDENEI,0,xwork,ywork,wwork,hwork,SELECTED,1);
         objc_change(menu_tree,CONDENAU,0,xwork,ywork,wwork,hwork,NORMAL,1);
```

```
gemdos(0x5,SCHMAL);
break;

case CONDENAU:
objc_change(menu_tree,CONDENAU,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,CONDENEI,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,SCHMALOFF);
break;

case PROPEIN:
objc_change(menu_tree,PROPEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,PROPAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,PROPORTIONAL);
gemdos(0x5,PSET);
break;

case PROPAUS:
objc_change(menu_tree,PROPAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,PROPEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,PROPORTIONAL);
gemdos(0x5,PRESET);
break;

case ITALEIN:
objc_change(menu_tree,ITALEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,ITALAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,ITALIC);
break;

case ITALAUS:
objc_change(menu_tree,ITALAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,ITALEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,ITALICOFF);
break;
```

```
case MICROEIN:
objc_change(menu_tree,MICROEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,MICROAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,MICRO1);
gemdos(0x5,MICRO2);
break;

case MICROAUS:
objc_change(menu_tree,MICROAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,MICROEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,MICROOFF);
break;

case SKIPEIN:
objc_change(menu_tree,SKIPEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,SKIPAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,SKIP);
break;

case SKIPAUS:
objc_change(menu_tree,SKIPAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,SKIPEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,SKIPOFF);
break;

case RAND0:
objc_change(menu_tree,RAND0,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,RAND10,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,LRAND);
gemdos(0x5,LAUSSEN);
break;
```

```
            case RAND10:
            objc_change(menu_tree,RAND10,0,xwork,ywork,wwork,hwork,SELECTED,1);
            objc_change(menu_tree,RAND0,0,xwork,ywork,wwork,hwork,NORMAL,1);
            gemdos(0x5,ESC);
            gemdos(0x5,LRAND);
            gemdos(0x5,POS10);
            break;

            case EXIT:
            objc_change(menu_tree,EXIT,0,xwork,ywork,wwork,hwork,SELECTED,1);
            gemdos(0x5,RET);
            form_dial(3,xobj,yobj,wobj,hobj);
            form_dial(2,1,1,1,1,xobj,yobj,wobj,hobj);
            ende=TRUE;
            break;

         } /* Ende switch  */

   } /* Ende while */
   desktop();

} /* Ende main() */

desktop()
{
   v_clsvwk();
   appl_exit();
}
```

Im vorangegangenen Listing dürfte für Sie eventuell erst einmal ein Teil der Symboldefinitionen interessant sein. Zumindest dann, wenn Sie nicht über einen EPSON- oder epson-kompatiblen Drucker verfügen sollten, denn in diesem Falle müssen Sie die entsprechenden Codes für Ihr Gerät innerhalb der Initialisierungsliste einsetzen.

Ansonsten dürfte das Listing Ihnen keine allzugroßen Kopfschmerzen bereiten, ist doch fast alles wie gehabt. Kleine Änderungen, die allerdings nur Schönheitsoperationen darstellen, sind die Aufrufe graf_growbox und graf_shrinkbox innerhalb der Window-Routinen. Sie sind für den sich schnell vergrößernden/verkleinernden Umriß verantwortlich.

Neu hinzugekommen ist lediglich der Aufruf des RSC-Files. Und da es sich bei rsrc_load() ebenfalls um eine Funktion handelt, wird diese auch einen Funktionswert zurückliefern, nämlich TRUE oder FALSE. Deshalb kann auch im Falle eines Fehlschlagens des Ladevorganges sogleich ein Alert-tree angezeigt und der Programmlauf abgebrochen werden.

Das Wichtigste überhaupt ist jedoch der Aufruf rsrc_gaddr(). Denn diese Funktion liefert als Ergebnis den Zeiger auf die Adresse eines gesuchten Objektes.

Beispiel:
Sicherlich muß nach Programmstart zunächst erst einmal der ganze Baum vom ersten bis zum letzten Objekt angezeigt werden. Wir müssen daher in Erfahrung bringen, wo im Speicher sich die Wurzel, in unserem Falle ist das der Baum FXMENU, befindet.
Also rufen wir rsrc_gaddr() auf, und teilen AES auch gleich mit, wonach wir suchen: nämlich nach dem Objekt FXMENU in einem Baum (R_TREE). Die Adresse von FXMENU soll dem Zeiger &menu_tree zugewiesen werden, daher:

```
rsrc_gaddr(R_TREE,FXMENU,&menu_tree);
```

Um den Baum, oder auch jedes andere Objekt, dann darstellen zu lassen, rufen wir Object_Draw auf; eine Funktion, die beliebige Teilbereiche eines jeden Baumes zeichnet:

```
objc_draw(menu_tree,0,MAX_DEPTH,0,0,wdesk,hdesk);
```

Die Parameter in der Reihenfolge ihres Auftretens legen fest, welcher Baum gezeichnet wird, beginnend mit welchem Objekt (hier Null, also dem ersten), bis zu welchem (die maximale Anzahl läßt sich unter INFO im RCS ablesen), und schließlich auch noch eine Fläche, die maximal für diese Aufgabe bereitgestellt wird.

Der Aufruf Object_Find innerhalb der Hauptschleife schließlich gibt nach Eingabe der Mausposition - die ja als vierter und fünfter Parameter bei event_button ermittelt wird - die Nummer des unter dem Mauspointer liegenden Objektes aus. Diese wird dann der Reihe nach mit all unseren Objektnummern, für die die symbolischen Konstanten stehen, verglichen, bis eine Übereinstimmung gefunden und die entsprechende Aktion ausgelöst wurde.
Die ebenfalls dort befindlichen Object_Change-Anweisungen haben keinen anderen Sinn, als jeweils das betreffende Objekt schwarz zu hinterlegen.

Die auch noch im Listing zu findenden form_dial-Aufrufe dienen der Darstellung der Dialogbox. Und zwar stellen sie entsprechenden Speicherplatz frei - schließlich muß der darunterliegende Bildschirmbereich gerettet werden - zeichnen die sich vergrößernde bzw. verkleinernde Box und geben abschließend den zuvor belegten Speicherbereich wieder frei.

4.5 PRINIT als ACCESSORY

Um die Applikation nun als Accessory zu installieren, sind eine Reihe von Änderungen notwendig. So sollte ein Accessory auf ein Fenster verzichten, zumindest sollte es nicht größer sein als die eigentliche Dialogbox. Meist ist es auch wenig zweckmäßig, wenn das Accessory auf dem Bildschirm verschoben werden kann, schließlich wird es nur kurzzeitig aufgerufen werden. Das erspart zum einen natürlich etwas Programmierarbeit, zum anderen jedoch auch einiges an Speicherplatz und Ladezeit. Ausreichend sind im Bereich open_window daher eigentlich wind_create und wind_get. Für diejenigen, die allerdings gerne ein Fenster haben würden, sind im folgenden Listing auch die dafür zuständigen Zeilen als Kommentare mit eingeschlossen.

Der wichtigste Schritt, um ein Accessory greifbar zu machen ist die Installation innerhalb des Desk-Menüs, ein Vorgang, für den die Funktion menu_register zuständig ist. Als Eingabeparameter benötigt sie zum einen den ID-Code der Applikation (von appl_init) und zum anderen einen String, unter dem das Programm im Menü erscheinen soll. Als Resultat liefert menu_register dann eine Zahl zwischen null und fünf, was einem Accessory-ID-Code entspricht.

Um jedoch erst einmal so weit zu kommen, muß das Programm gestartet werden, was nach dem Laden des Betriebssystemes mit jedem Accessory auch geschieht. Der Anwender bemerkt davon jedoch nichts, weil die Workstation zuvor nicht geöffnet worden ist.

Nach dieser Phase der Initialisierung springt das Accessory in einen event_multi-Aufruf, welcher unbedingt alle Ereignisse zulassen muß, um nicht ein laufendes Hauptprogramm zu behindern. Dort wartet es dann auf ein Message-event, das im Messagepuffer den eigenen Accessory-Id zurückläßt.

Wie bereits verdeutlicht, steht der Code der aufzurufenden Applikation in msgbuff(4), d.h. wenn die Bedingung

```
if (msgbuff(4) == menu_id)
```

ein TRUE liefert, hat der Anwender das betreffende Accessory aufgerufen.

Erst jetzt werden die Workstation und das Window geöffnet, anschließend wird zum eigentlichen Programm verzweigt (hier: output();). Dieses verfügt dann über eine eigene Event-Abfrage und entspricht im wesentlichen auch dem Normalprogramm. Nur ist darauf zu achten, die Abbruchbedingung rechtzeitig wieder als FALSE zu erklären; dies kann vor Verlassen (nicht beenden!) des Programmes geschehen, oder aber wie hier zu Beginn der Hauptschleife, da ein erneuter Aufruf des Accessories sonst schwerlich möglich ist.

Vor allem gilt es zu beachten, daß der Programmlauf eines Accessories niemals beendet ist, folglich darf sich auch nirgends ein appl_exit finden. Accessories laufen immer im Multitasking-Betrieb, d.h. jedes Accessory steht innerhalb der READY-Liste, die event_multi-Abfrage eines jeden Accessories findet andauernd statt.

Daher lautet die Struktur eines solchen event_multi-Calls auch:

```
while (TRUE)
      ... event_multi /* Abfrage */
      ... Message_event ?
            ... wenn ja: eigener menu_id ?
                  ... wenn ja: los geht's
            ... wenn nein: weiterwarten
      ... wenn nein: weiterwarten
... /* Ende while */
```

Nirgends innerhalb des gesamten Programmcodes darf sich ein Statement finden, das diese Schleife abbrechen könnte. Deshalb lautet die Bedingung auch einfach TRUE, was natürlich immer der Fall ist!

```
/*************************************************************************/
/**************                PROGRAMM: PR-INIT                **************/
/********       EinstellACCESSORY fuer Drucker am Parallelport      **********/
/**************     (c) J. Walkowiak, 4. November 1985          **************/
/*************************************************************************/
#include "obdefs.h"                 /* Objektdefinitionen                    */
#include "gemdefs.h"                /* Definitionen zu GEM                   */
#include "define.h"
#include "gembind.h"
#include "vdibind.h"


-------------------------------------------------------------------------
ACHTUNG!
Alle folgenden Definitionen sind identisch zu vorangegangenem Programm
und muessen von dort uebernommen werden.
Oder schreiben Sie saemtliche #defines in eine eigene include-Datei
um sich das hantieren mit allzu langen Sourcecodes zu ersparen.
-------------------------------------------------------------------------


/*************************************************************************/
/*                                  globale Variablen                    */
/*************************************************************************/
int contrl[12];                 /* Kontroll-Arrays                       */
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];                /* jeweils genug Platz fuer alle Faelle  */

int work_in[11];                 /* Eingabe ins GSX Array                */
int work_out[57];                /* Ausgabe vom GSX Array                */


int handle,i;                   /* virtual workstation handle            */
int phys_handle;                /* physical workstation handle           */
int wi_handle;                  /* Window handle                         */
```

```
extern gl_apid;                 /* Kennung der Applikation              */
extern long gemdos();           /* fuer GEMDOS-Call                     */

int menu_id;                    /* Accessory Kennung im Deskmenu        */

int gl_hchar, gl_wchar;         /* Hoehe und Weite der Zeichen          */
int gl_wbox, gl_hbox;

int xwork,ywork,wwork,hwork;    /* Arbeitsgroesse Fenster               */
int xdesk,ydesk,wdesk,hdesk;    /* Groesse Desktop                      */
int xobj,yobj,wobj,hobj;        /* Groesse eines Objektes               */
int mausx, mausy;               /* wo ist die Maus, wenn was passiert ? */

int dummy;                      /* ... fuer ueberfluessige Parameter    */

int event;                      /* welches Eingabegeraet hat was zu melden*/
int msgbuff[8];
int title, item;                /* Menuetitel und aktuelles Objekt      */

int ende;



/*****************************************************************/
/* open virtual workstation                                      */
/*****************************************************************/
open_vwork()
{
int i;
   for(i=0;i<10;work_in[i++]=1);
   work_in[10]=2;
   handle=phys_handle;
   v_opnvwk(work_in,&handle,work_out);
}


/*****************************************************************/
/* open window                                                   */
```

```
/*****************************************************************/
open_window()
{
   wi_handle=wind_create(0x0000,xobj,yobj,wobj,hobj);
                                 /* Fenster nur so gross wie Dialogbox(obj)*/
/*   wind_set(wi_handle, WF_NAME," hier den Namen ",0,0);  nur wenn Fenster mitTitelzeile
   graf_growbox(xdesk+wdesk/2,ydesk+hdesk/2,gl_wbox,gl_hbox,xdesk,ydesk,wdesk,hdesk);*/
   wind_open(wi_handle,xobj,yobj,wobj,hobj);
                                 /* Arbeitsfenster oeffnen              */
   wind_get(wi_handle,WF_WORKXYWH,&xwork,&ywork,&wwork,&hwork);
}


/*****************************************************************/
/* Initialisierung des Acc.                                      */
/*****************************************************************/
main()
{
   appl_init();
   phys_handle=graf_handle(&gl_wchar,&gl_hchar,&gl_wbox,&gl_hbox);
   menu_id=menu_register(gl_apid,"  FX-80+ INIT");
   wind_get(0, WF_WORKXYWH, &xdesk, &ydesk, &wdesk, &hdesk);

      if(!rsrc_load(FILENAME))     /* RSC-file laden                  */
         {
         form_alert(1,"[3][Raubkopierer, was?|PRINIT.RSC|kann ich nicht finden.][Abbruch]");
         }
      if(rsrc_gaddr(0,0,&menu_tree)== 0)
         {
         form_alert(1,"[3] [Schwerer Fehler!|Resource File nicht OK.][Abbruch]");
         }

      rsrc_gaddr(R_TREE,FXMENU,&menu_tree);
      form_center(menu_tree,&xobj,&yobj,&wobj,&hobj);

   multi();
}
```

```
/*****************************************************************/
/* Warten auf den eigenen Aufruf                                 */
/*****************************************************************/
multi()
{
   while (TRUE) {
      event = evnt_multi (MU_MESAG | MU_BUTTON | MU_KEYBD,
         1,1,1,
         0,0,0,0,0,
         0,0,0,0,0,
         msgbuff,0,0,&mausx,&mausy,&dummy,&dummy,
         &dummy,&dummy);


   if (event & MU_MESAG)
      switch (msgbuff[0]) {

         case AC_OPEN:
            if (msgbuff[4] == menu_id) {
                open_vwork();
                open_window();
                output();
                wind_close(wi_handle);
                wind_delete(wi_handle);
                v_clsvwk(handle);
            }
            break;
       }  /* switch */
   } /*while TRUE */
}


/*****************************************************************/
/* hier das eigentliche Programm ...                             */
/*****************************************************************/
output()
{
      rsrc_gaddr(R_TREE,FXMENU,&menu_tree);
      form_center(menu_tree,&xobj,&yobj,&wobj,&hobj);
```

```
    form_dial(0,xobj,yobj,wobj,hobj);
    form_dial(1,1,1,1,1,xobj,yobj,wobj,hobj);

    objc_draw(menu_tree,0,MAX_DEPTH,0,0,wdesk,hdesk);

    ende = FALSE;                /* sonst laeuft es nur einmal          */
    while (ende != TRUE){
        event=evnt_button(1,1,1,&mausx,&mausy,&dummy,&dummy);
        item=objc_find(menu_tree,FXMENU,13,mausx,mausy);
                                 /* welches Objekt in menu_tree an Mauspos*/

        switch(item){


--------------------------------------------------------------------------
ab hier identisch zu vorangegangenem Listing !
--------------------------------------------------------------------------


            case EXIT:
            objc_change(menu_tree,EXIT,0,xwork,ywork,wwork,hwork,SELECTED,1);
            gemdos(0x5,RET);
            form_dial(2,xobj,yobj,wobj,hobj);
            form_dial(3,1,1,1,1,xobj,yobj,wobj,hobj);
            ende=TRUE;
            objc_change(menu_tree,EXIT,0,xwork,ywork,wwork,hwork,NORMAL,1);
            /* Ruecksetzen; ist sonst beim naechsten Accessory call noch aktiv */
            break;

         } /* Ende switch  */

    } /* Ende while */
}
```

Das Informationspaket zum ATARI ST mit ausführlicher Hardwarebeschreibung, detaillierter Erläuterung der Schnittstellen: V.24, Expansion-Interface, MIDI-Interface, Aufbau von Grafiken, BIOS, GEM, wichtige Systemadressen und was man damit machen kann, die Funktionsweise der Maus. Unentbehrlich für's professionelle Arbeiten mit dem ATARI ST.

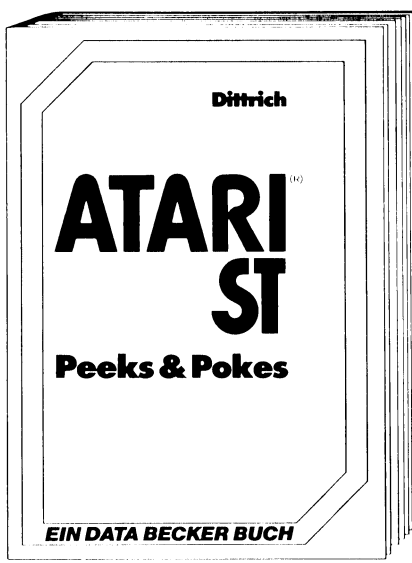
Gerits/Englisch/Brückmann
ATARI ST INTERN
ca. 400 Seiten, DM 69,–
ISBN 3-89011-119-X

Ein Buch für jeden, der unter GEM Programme erstellen will! Arbeiten mit der Maus, Icons, Virtual Device Interface, Application Environment Services und Graphics Device Operating System. Ein besonderer Schwerpunkt liegt im Einbinden von GEM-Routinen in BASIC und 68000-Assembler und der Programmierung in diesen Sprachen. GEM – das Betriebssystem der Zukunft!

Szczepanowski/Günther
Das große GEM-Buch zum ATARI ST
ca. 350 Seiten, DM 49,–
ISBN 3-89011-125-4

Kein Programmierer, der die Vorteile des 68000-Prozessors nutzen will, sollte auf dieses Handbuch verzichten. Sie finden detailliertes Sachwissen anschaulich dargestellt, zur Technik und Programmierung: Entwicklung des 68000, Aufbau, Signal- und Busbeschreibung, Peripheriebausteine, Befehlssatz, Programmierbeispiele, Vergleich mit anderen 16-Bit-Prozessoren, weitere Prozessoren der Familie u.v.m. Ein Buch für echte Computerfreaks!

Grohmann/Eichler
Das Prozessorbuch zum 68000
516 Seiten, DM 59,–
ISBN 3-89011-094-0



Schlagen Sie dem Betriebssystem Ihres ATARI ST ein Schnippchen. Wie? Mit PEEKS & POKES natürlich! Dieses Buch erklärt Ihnen leichtverständlichh den Umgang damit. Mit einer riesigen Anzahl wichtiger POKES und ihren Anwendungsmöglichkeiten. Dabei wird der Aufbau Ihres ST's prima erklärt: Betriebssystem, Interpreter, Zeropage, Pointer und Stacks sind nur einige Stichworte dazu. Der erste Schritt hin zur Maschinensprache!

PEEKS & POKES zum ATARI ST
ca. 250 Seiten, DM 29,–
ISBN 3-89011-148-3

Empfängerabschnitt

DM 39,-- | Pf ---

für Postgirokonto Nr. 789 - 436

Absender (mit Postleitzahl)

Verwendungszweck

Diskette z. Buch

376 118

DM 39,-- | Pf --- | für Postgirokonto Nr. 789 - 436

Absender:

Zahlkarte (Mit Schreibmaschine, Tinte oder Kugelschreiber deutlich ausfüllen)

DM 39,-- | Pf --- | (DM-Betrag in Buchstaben wiederholen) Neununddreißig

für DATA BECKER GmbH
Merowingerstr. 30

in 4000 Düsseldorf

Postgirokonto Nr. 789 - 436

Postgiroamt Essen

Postvermerk

Für Vermerke des Absenders

Einlieferungsschein

– Bitte sorgfältig aufbewahren –

DM 39,-- | Pf ---

für DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Postgirokonto Nr. 789 - 436

Postvermerk

LADEN, STARTEN - KLAR !

Für alle, die Spitzenprogramme aus „ATARI ST Tips & Tricks“ schnell nutzen wollen, hier das Super-Angebot:

Mit der nebenstehenden Post-Zahlkarte einfach die Diskette zum Buch bestellen !

Diskette zum Buch „ATARI ST Tips & Tricks“ DM 39,--

Diskette zum Buch
„ATARI ST Tips & Tricks“
376 118

Für Mitteilungen an den Empfänger

Bedienen Sie sich der Vorteile
eines eigenen Postgirokontos!
Auskunft hierüber erteilt jedes Postamt

Feld
für
postdienstliche
Zwecke

Einlieferungsschein
(nicht zu Mitteilungen an den Empfänger benutzen)

Gebühr für die Zahlkarte
(wird bei der Einlieferung bar erhoben)

bis 10 DM . . . . . 90 Pf
über 10 DM (unbeschränkt) 1,50 DM

DAS STEHT DRIN:

Eine riesige Fundgrube wirkungsvoller Tips & Tricks rund um den neuen ATARI ST. Alle Programme sind gut erklärt und können in eigene Anwendungen eingebaut werden. Diese Routinen sind wirklich absolut neu.

Aus dem Inhalt:

- BASIC und GEM
- Der VDISYS-Befehl
- BASIC und Maschinensprache
- Automatische Hardcopy
- Druckertreiber für EPSON-Drucker
- RAM-Disk für ATARI ST
- Druckerspooler
- Automatisches Starten von TOS-Anwendungen
- C und Maschinensprache
- Hardcopy in Farbe
- GEM intern
- GEM-Anwendungen
  und vieles mehr

UND GESCHRIEBEN HABEN DIESES BUCH:

Das bewährte DATA BECKER-Autorenteam mit Rolf Brückmann, Lothar Englisch, Jörg Walkowiak und Klaus Gerits. Alle sind nicht nur begeisterte Programmierer, sondern verstehen sich ausgezeichnet darauf, komplexe Materie verständlich darzulegen.

ISBN 3-89011-118-1