DE RE ATARI

Alles über den ATARI

ATARI®

Gescannt & Aufbereitet für den

ATARI Bit Byter User Club e.V.
(www.abbuc.de)
c/o Wolfgang Burger
Wieschenbeck 45
45699 Herten
Wolfgang@abbuc.de

Von...

Marc Brings
Auf dem Postberg 21
50169 Kerpen
Marc@abbuc.de

Fehler bitte per Mail an mich!

INHALT

VORWORT

Dieses Manual behandelt das ATARI Personal Computer System. Der Sinn des Manuals liegt in der detailierten Erklärung der Möglichkeiten des ATARI Personal Computer Systems. Dieses ist eine Maschine, die sehr leistungsstark und dabei komplex ist; deshalb sind die Erläuterungen entsprechend lang und ausführlich. Weiterhin wird vom Leser ein gewisses Fachwissen verlangt, da dieses Buch nicht für den angehenden Programmierer vorgesehen ist. Der Leser sollte mit dem BASIC Referenz-Manual, welches zusammen mit dem Computer geliefert wird, vertraut sein. Kenntnis der Assembler-Sprache ist ebenfalls erforderlich.

Dieses Buch wurde als Trainingsmaterial für professionelle Programmierer, die das ATARI Personal Computer System benutzen wollen, geschrieben. Möglicherweise wird es später für den allgemeinen Gebrauch modifiziert. Dieses Manual ersetzt das technische Referenz-Manual (ATARI Art.-Nr. C016555) nicht. Es ist ein Referenz-Manual; das bedeutet, dass es sehr nützlich für den Programmierer ist, der das System bereits kennt. Dieses Buch ist eine universelle Zusammenfassung, d.h., es werden eher Ideen und Möglichkeiten erklärt, als Register oder Kontroll-Codes.

Dieses Buch wurde von der "Software Development Support Group" geschrieben. Kapitel 1 bis 6 stammen von Chris Crawford. Lane Winner schrieb mit Assistenz von Jim Cox Kapitel 7. Von Amy Chen stammt Anhang III. Mike Ekberg schrieb Kapitel 8 und 9 mit Assistenz von John Eckstrom. Kathleen Pitta schrieb Anhang VII. Kapitel 10 stammt aus der Feder von Bob Fraser. Gus Makreas verfaßte das Inhaltsverzeichnis. Das endgültige Ergebnis hat sicherlich einige Mängel, aber wir sind stolz, daß wir es nun vorlegen können. Anhang IX erläutert die zusätzlichen Fähigkeiten der XL-Computer.

John Eckstrom
Michael A. Eckberg
Gus Makreas
Lane Winner
Elizabeth Hernaton

Kathleen Pitta
Jim Cox
Bob Fraser
Chris Crawford
Axel Kawa
(Übersetzung)

Kapitel 1
Überblick über das System

Das Atari Personal Computer System ist ein Computer der zweiten Generation. An erster Stelle ist es ein Verbraucher-orientierter Computer. Das Ziel des Designs ist es, dem Verbraucher einen bequem handhabbaren Computer zu liefern. Diese Verbraucher-Orientierung offenbart sich auf viele Arten. Erstens ist die Maschine narrensicher, d.h. der Verbraucher wird vor Fehlern (die das Gerät beschädigen könnten), geschützt. Dieses geschieht z.B. durch polarisierte Stecker, die nicht verkehrt zusammenpassen.

Zweitens besitzt die Maschine eine große Anzahl an Graphik-Möglichkeiten. Der Mensch reagiert auf bildhafte Darstellung besser als auf Text. Schließlich sind Joysticks und Paddles an das Gerät anschließbar, die eine direktere Steuerung gestatten, als es über die Tastatur möglich ist. Der springende Punkt hierbei ist nicht, daß der Computer viele Möglichkeiten besitzt, sondern, daß diese Teil einer konsequenten Philosophie sind, die direkt auf den Verbraucher abzielt. Der Programmentwickler, der diese fundamentale Tatsache nicht anerkennt, oder zumindest berücksichtigt, wird bald merken, daß er gegen den Strich des Systems arbeitet.

Der interne Aufbau des ATARI Computers unterscheidet sich stark von anderen Systemen. Natürlich besitzt er einen Mikroprozessor (einen 6502), RAM, ROM und einen PIA. Zudem befinden sich im Gerät aber noch drei spezielle LSI-Chips: der ANTIC, der POKEY und der CTIA. Diese Chips wurden von ATARI-Ingenieuren entworfen, um den größten Teil der Verwaltungsarbeit zu übernehmen und dadurch den 6502 zu entlasten. So kann dieser sich auf Berechnungen konzentrieren. Bei dieser Entwicklung wurde gleichzeitig eine große Leistungsfähigkeit in diese Chips implementiert. Jeder dieser Chips ist, (im Bezug auf die Schaffung), genauso komplex wie der 6502, so daß die drei zusammen eine optimale Leistungsfähigkeit liefern. Die Beherrschung des ATARI Computers liegt in der Beherrschung dieser Chips.

ANTIC ist ein Mikroprozessor, der speziell für das Fernsehbild zuständig ist. Er ist ein "wirklicher" Mikroprozessor, d.h. er besitzt einen Befehlssatz, ein Programm (genannt Display List) und Daten. Die Display List und die Bildschirmdaten werden durch den 6502 in den RAM-Bereich geschrieben. ANTIC holt diese Information aus

dem RAM, indem er den DMA (Direct Memory Access = Direkter Zugriff auf den Speicher) benutzt. Er führt die übergeordneten Anweisungen in der Display List aus und übersetzt diese Instruktionen in einen Echtzeit-Fluß von einfachen Daten zum CTIA-chip.

CTIA ist ein Fernseh-Interface Chip. ANTIC kontrolliert direkt die meisten Operationen des CTIA, aber es ist möglich, den 6502 so zu programmieren, daß er einige oder sogar alle CTIA-Funktionen steuert. CTIA wandelt die digitalen Kommandos von ANTIC (oder des 6502) in ein Signal um, welches dann zum Fernseher geht. CTIA manipuliert außerdem einige andere Faktoren, wie z.B. Farbwerte, Player-Missile-Graphik oder Kollisions-Abfrage. Bei den meisten deutschen ATARI Computern ist der GTIA eingebaut (siehe Anhang X).

POKEY ist ein digitaler I/0 Chip. Er bearbeitet solch verschiedene Aufgaben, wie die Verwaltung des seriellen I/O-Busses, Tonerzeugung, Tastatur-Abfrage und Erzeugung von Zufallszahlen. Er fragt zudem die Widerstandseingabe ab, die die Paddles liefern, und kontrolliert die Peripherieanforderung "maskierbarer" Interrupts (IRQ).

All diese Chips arbeiten parallel, d. h. gleichzeitig. Besondere Trennung ihrer Funktionen in der Entwurfsphase begrenzen mögliche Konflikte zwischen den Chips auf ein Minimum. Der einzige kritische Zustand taucht auf, wenn ANTIC die
Adress- und Datenbusse benutzt, um seine Bildschirminformation zu bekommen. Um diese durchführen zu können, wird der 6502 angehalten und die Kontrolle der Busse von ANTIC übernommen.

Wie bei allen 6502-Systemen erfolgt die Eingabe/Ausgabe (I/0) in derselben Reihenfolge wie die Abfolge der Daten im Speicher. Abbildung 1.1 zeigt grob die Speicheraufteilung des Computers. Abbildung 1.2 stellt die Anordnung der Hardware im Gerät dar.

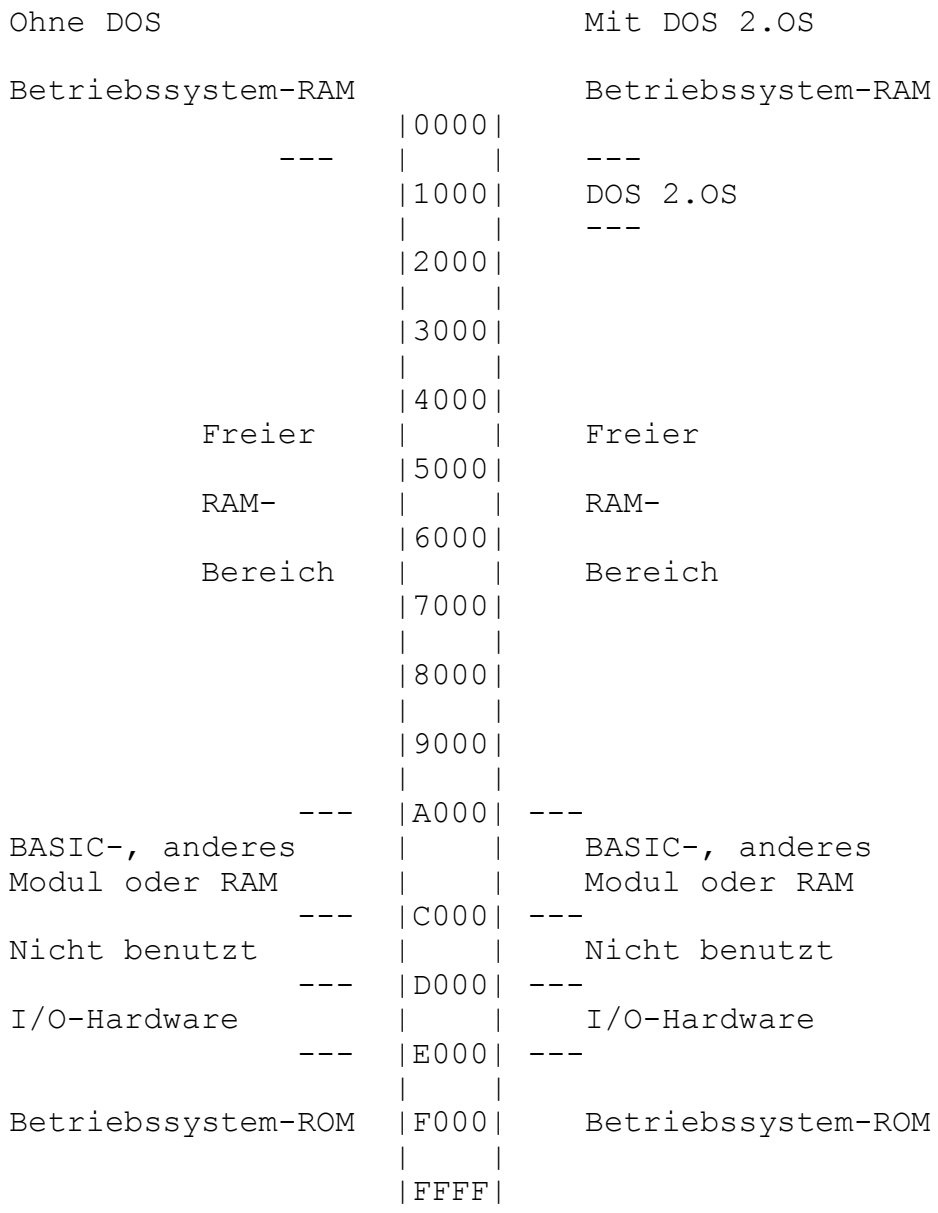


Abbildung 1.1:
Speicheraufteilung
400/800

Speicheraufteilung bei der XL-Serie
siehe Anhang XI

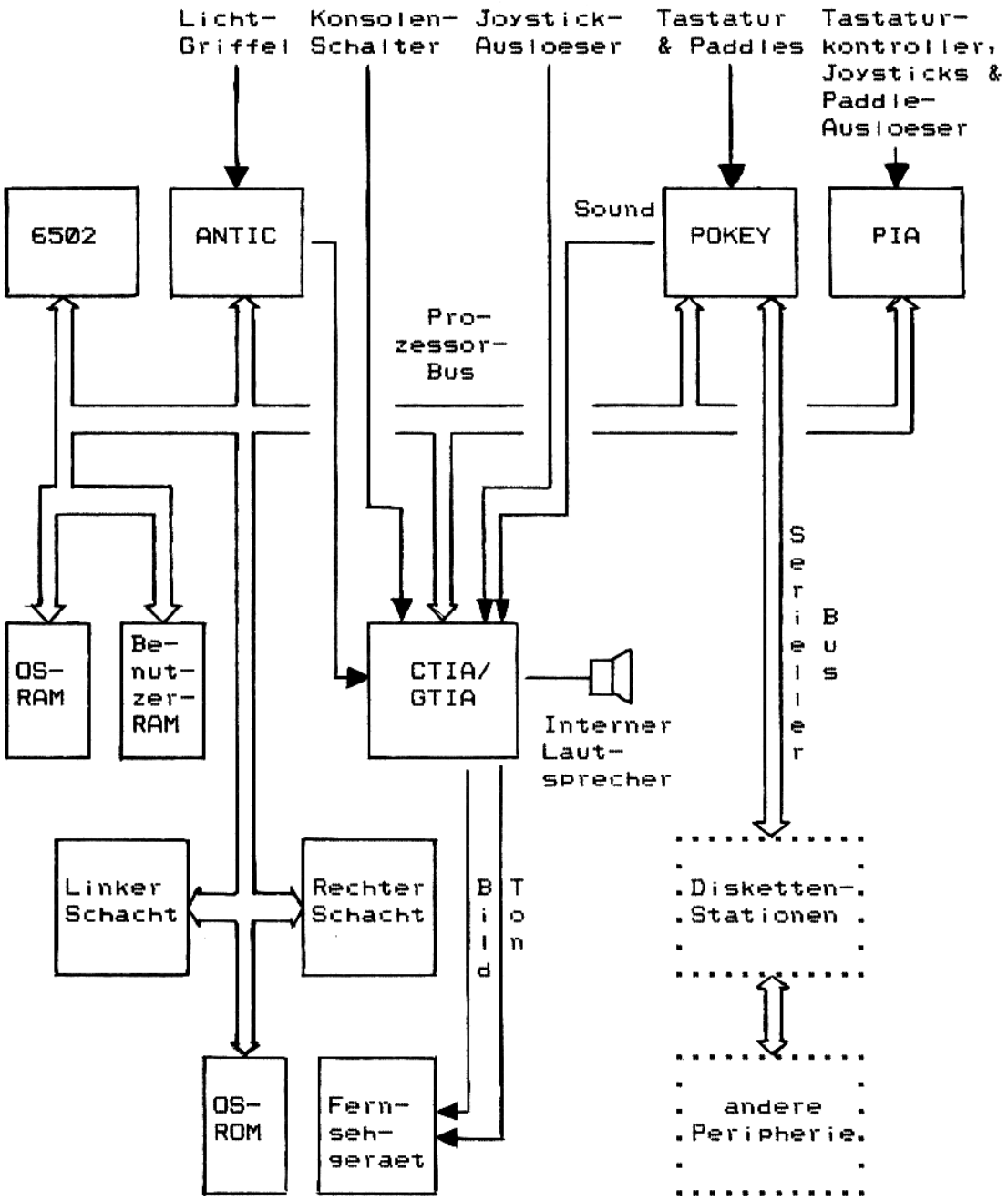


Abbildung 1.2:
Hardware-Layout des ATARI Computers

Kapitel 2
ANTIC und die Display List

FERNSEH-BILDSCHIRME

Um die Graphik-Möglichkeiten des ATARI„TM" Personal Computer Systems ganz zu begreifen, muß man erst die Funktionsgrundlagen eines Fernsehgerätes verstehen. Fernseher bedienen sich eines Systems, das als "Raster Scan Display" (Raster Bildschirm-Anzeige) bezeichnet wird. Am hinteren Ende der Fernsehröhre wird ein Elektronenstrahl erzeugt, der dann auf die Mattscheibe geworfen wird. Dabei wird er von waagerecht und senkrecht angebrachten Magnetspulen abgelenkt, sofern diese unter Spannung stehen. Auf diese Weise kann der Strahl auf jeden Punkt des Schirms geworfen werden. Durch die im Fernseher befindliche Elektronik wird diese Ablenkung so gesteuert, dazu sich der Strahl mit gleichmäßiger Geschwindigkeit über die Mattscheibe bewegt. Weiterhin kann die Stärke, d.h. die Intensität das Strahls gesteuert werden. Wird der Strahl intensiver, so leuchtet der getroffene Punkt auf dem Schirm heller; wird der Strahl schwächer, dann leuchtet der Punkt weniger stark oder überhaupt nicht mehr.

Der Strahl startet in der linken oberen Ecke des Schirmes und bewegt sich waagerecht über denselben. Während dieses Vorgangs wird durch Intensitätsänderung des Strahls ein Bild auf den Fernsehschirm gezeichnet. Sobald der Strahl den rechten Bildschirmrand erreicht, wird er abgeschaltet und zurück zur linken Seite gebracht. Gleichzeitig wird er um eine Elektronenstrahldicke nach unten bewegt. Danach wird er wieder eingeschaltet und läuft ein weiteres Mal über den Schirm. Dieser Vorgang wird insgesamt 312 mal wiederholt. (Bei einem "abwechselnden" System gibt es in Wahrheit 625 Bewegungen über den Fernsehschirm, was als "Interlace" bezeichnet wird. Wir wollen dieses Interlace ignorieren und vorgehen, als ob der Fernseher nur 312 Zeilen hätte.) Diese 312 Zeilen füllen den Bildschirm von oben bis unten aus. Am unteren Ende des Schirmes wird der Strahl (nachdem die 312te Zeile gezeichnet wurde) ausgeschaltet und wieder in die linke obere Bildschirmecke gebracht. Dann beginnt der gesamte Prozeß von vorne. Dieser Zyklus wiederholt sich 50mal in einer Sekunde.

Das Ganze jetzt noch einmal in Fachsprache: eine einzelne Bewegung des Elektronenstrahls quer über den Bildschirm bezeichnet man als „Horizontal Scan Line". Sie ist die grundlegende Einheit beim Meßen von senkrechten Entfernungen auf dem Schirm. Die Höhe eines Bildes wird festgelegt, indem man die Anzahl der Horizontal Scan Lines, über die es sich erstreckt, angibt. Die Periode, in welcher der Elektronenstrahl vom rechten zum linken Bildschirmrand zurückläuft, wird "Horizontal Blank" genannt. Der Zeitraum, in

welchem der Strahl zum oberen Mattscheibenrand zurückkehrt, wird als „Vertical Blank" bezeichnet. Der Zeichenvorgang für den gesamten Bildschirm dauert 16684 Mikrosekunden. Der „Vertical Blank" erstreckt sich über ca. 1400 Mikrosekunden. Der Horizontal Blank benötigt 14 Mikrosekunden. Eine einzelne Horizontal Scan Line braucht 64 Mikrosekunden, um gezeichnet zu werden.

Die meisten Fernsehgeräte besitzten einen „Overscan", das bedeutet, sie ziehen die Bilder soweit auseinander, daß sich deren Ränder außerhalb des Mattscheibenrandes befinden. Dieses garantiert ein Fernsehbild ohne störende Randbalken. Andererseits ist dies für Computer sehr ungünstig, da die Information, die sich außerhalb des Schirmes befindet, für den Benutzer keine Bedeutung mehr hat und somit verlorengeht. Aus diesem Grunde ist das Bild, welches der Computer ausgibt, etwas kleiner als jenes, das der Fernseher theoretisch anzeigen könnte. Es werden nur 192 Horizontal Scan Lines von ATARI „TM"-Display benutzt. Dies bedeutet, daß die normale Begrenzung der senkrechten Auflösung eines Fernsehers in Verbindung mit diesem Computer 192 Pixel beträgt.

Die standardmäßige Einheit für horizontale Entfernungen ist das „Color Clock". Die Breite eines Bildes wird festgelegt, indem man die Anzahl der Color Clocks, über die es sich erstreckt, angibt. Es gibt 228 Color Clocks in einer Horizontal Scan Line, von denen allerdings maximal nur 176 wirklich sichtbar sind. Dieses bedeutet wiederum, daß die absolute Begrenzung für vollfarbige waagerechte Auflösung mit einem handelsüblichen Fernseher 176 Pixel beträgt. Mit dem ATARI „TM" Personal Computer System ist es möglich, noch weiter zu gehen und individuelle Halb-Clocks zu kontrollieren. Dieses ergibt dann eine waagerechte Auflösung von 352 Pixeln.

COMPUTER UND FERNSEHGERÄTE

Das grundsätzliche Problem eines Computers mit einem Raster Scan Fernseh-System für Anzeigezwecke ist, daß die Erzeugung des Fernsehbildes ein dynamischer Vorgang ist; der Fernseher kann sich nicht an das Bild erinnern. Folglich fällt diese Aufgabe dem Computer zu. Er muß ständig ein Signal zum Fernseher senden, das diesem mitteilt, was er anzeigen soll. Dieser Sendevorgang ist ein ununterbrochener Prozeß und erfordert daher andauernde Beachtung. Aus diesem Grunde haben die meisten Mikrocomputer spezielle Hardware, um den Fernseher zu bedienen. Die Grundanordnung ist auf fast allen Systemen die gleiche:

Mikroprozessor - Bildschirm-RAM - Video-Hardware - TV-Schirm

Der Mikroprozessor schreibt Informationen in den Bereich des Bildschirm-RAMs, der die Bilddaten beinhaltet. Die Video-Hardware greift nun andauernd in diesen Bereich, um die

Daten zu erhalten, die sie dann in Fernsehsignale umwandelt. Diese Signale gehen zum Fernsehgerät, das diese schließlich als Information anzeigt. Der Bildschirmspeicher wird in der gleichen Reihenfolge auf den Schirm geworfen, wie die Daten im RAM stehen. Das bedeutet, das erste Byte im Speicher gehört zur linken oberen Ecke des Bildschirms, das zweite Byte im Speicher zur rechts daneben liegenden Position und so fort, bis zum letzten Byte, welches zur rechten unteren Ecke der Mattscheibe gehört.

Die Qualität des Bildes, das zum Fernseher gelangt, hängt von zwei Faktoren ab: der Qualität der Video-Hardware und der Größe des Bildschirm-RAMs, welches für das Display benutzt wird. Die einfachste Anordnung findet sich bei TRS-80 und PET (TRS-80 ist ein Warenzeichen der Radio Shack Co.; PET ist ein Warenzeichen der Commodore Business Machines). Diese beiden Geräte benutzen einen festgelegten 1K-Bereich des RAMs als Bildschirmspeicher, Die Video-Hardware holt einfach Daten aus diesem Bereich und interpretiert sie mit Hilfe eines Zeichensatzes im ROM als Zeichen, was eine Anzahl von 256 verschiedenen Zeichen erlaubt. Mit einem Bildschirm-RAM von 1K können ca. 1000 Zeichen auf den Schirm gebracht werden. Mit so einem System gibt es allerdings nicht viele Möglichkeiten. Der Apple benutzt ausgefeiltere Video-Hardware (Apple ist ein Warenzeichen der Apple Computers). Es sind drei verschiedene Graphik-Modi vorhanden: Text, Lo(w)-Res(olution)-Graphik und Hi(gh)-Res(uolution)-Graphik. Der Text-Modus funktioniert etwa auf die gleiche Art, wie das Anzeige-System von PET und TRS-80.

Im Lo-Res-Graphik-Modus greift die Video-Hardware auf den Bildschirmspeicher zu und interpretiert ihn folgendermaßen: anstatt ein Byte als ein Zeichen darzustellen, wird jedes Byte in ein Paar von Farb-Nybbles (Halbbytes) aufgeteilt. Der Wert eines jeden Nybbles im Speicher legt die Farbe eines Pixels (Bildschirmpunktes) fest. Im Hi-Res-Graphik-Modus entspricht jedes Speicher-Bit einem Pixel auf dem Schirm. Enthält das Bit eine 1, dann leuchtet das entsprechende Pixel; enthält es eine 0, dann leuchtet es nicht. Hierdurch ergibt sich eine Vielzahl von Farbnuancen, welches aber die Grundidee ist. Die entscheidende Tatsache ist, daß der Apple drei Anzeige-Modi besitzt; drei unterschiedliche Arten die Daten im Bildschirmspeicher zu interpretieren. Die Apple-Hardware für den Bildschirm kann ein Byte des Speichers als 8-Bit Zeichen (Text-Modus), als zwei 4-Bit Farb-Nybbles (Lo-Res-Modus) oder als 7 individuelle Bits für Bit-Map (Hi-Res-Modus) auslegen.

ANTIC, EIN VIDEO-MIKROPROZESSOR

Das ATARI Computer Display List ist eine Erweiterung der oben genannten Systeme. Wo PET- und TRS-80 Geräte einen, und der Apple drei verschiedene Modi besitzen, da hat der ATARI ganze 14 Modi. Der zweite wichtige Unterschied ist, daß die

Anzeige-Modi auf dem Bildschirm gemischt werden können. Das bedeutet, daß der Benutzer nicht gezwungen ist, sich zwischen einem Bildschirm voll Graphik oder einem Bildschirm voll Text zu entscheiden. Der dritte grundlegende Unterschied ist, daß das Bildschirm-RAM überall im Speicher des Computers abgelegt und sogar bewegt werden kann, während das Programm läuft, wogegen andere Geräte festgelegte RAM-Bereiche benutzen.

All dieses wird durch einen Mikroprozessor, den ANTIC, möglich gemacht. Wo die früheren Geräte sehr einfache Video-Schaltkreise benutzen, hat ATARI „TM" einen ganzen Mikroprozessor eingesetzt, der nur für das Fernsehbild zuständig ist. ANTIC ist ein wirklicher Mikroprozessor, d.h. er besitzt einen Befehlssatz, ein Programm und Daten. Das Programm für den ANTIC wird als Display List bezeichnet. Diese Display List legt drei Dinge fest: wo die Bildschirmdaten zu finden sind, welche Anzeige-Modi benutzt werden sollen um die Bildschirmdaten zu interpretieren, und welche speziellen Anzeige-Einstellungen eingefügt werden sollen.

Beim Benutzen der Display List muß man sich von der Vorstellung eines alten Bildschirmes freimachen. Der Schirm ist kein gleichmässiges Bild in einem einzelnen Modus, sondern ein "Stapel" von sogenannten "Mode-Lines". Eine solche Mode-Line ist eine Zusammenstellung von Horizontal Scan Lines. Sie erstreckt sich waagerecht über die ganze Bildschirmbreite. Eine Graphik-Modus 2 Mode-Line ist 16 Horizontal Scan Lines hoch, wogegen eine Graphik-Modus 7 Mode-Line nur zwei Scan-Lines hoch ist. Viele Graphik-Modi, die in BASIC verfügbar sind, sind durchgehende Modi, d.h. der gesamte Bildschirm besteht aus einem einzelnen Modus. Dies bedeutet aber nicht, daß man darauf beschränkt ist: mit der Display List ist es möglich, jede Folge von Mode-Lines auf dem Schirm zu erzeugen. Die Display List ist eine Zusammenstellung von Code-Bytes, die diese Abfolge festlegen.

ANTICs Befehlssatz ist sehr einfach. Es gibt vier Befehlsklassen: Map-Mode-Befehle, Charakter-Mode- (Zeichen-Modus) Befehle , Blank-Line- (Leerzeilen) Befehle und Sprunganweisungen. Map-Mode-Befehle veranlassen, daß ANTIC eine Mode-Line mit einfach gefärbten Pixeln anzeigt. Charakter-Mode-Befehle veranlassen ANTIC eine Mode-Line mit Zeichen anzuzeigen. Blank-Line-Befehle bewirken, daß ANTIC eine bestimmte Anzahl von Horizontal Scan Lines mit fester Hintergrundfarbe ausgibt.

Sprunganweisungen entsprechen den 6502-Sprungbefehlen: sie laden den Programmzähler von ANTIC neu. Es können außerdem vier spezielle Einstellungen festgelegt werden, indem ein bestimmtes Bit des Befehls für ANTIC gesetzt wird. Die Operationen sind:

Display-List-Interrupt (DLI), Load-Memory-Scan (LMS), sowie senkrechtes und waagerechtes Scrolling.

Map-Modus-Befehle veranlassen ANTIC eine Mode-Line zu zeigen die Pixel mit festen Farben enthält. Die angezeigte Farbe wird durch ein Farbregister festgelegt. Die Wahl zwischen den einzelnen Farbregistern wird durch den Wert der Bildschirmdaten bestimmt. In 4-Farb-Map-Modi (BASIC-Modi 3, 5, 7 und 15; ANTIC-Modi 8, A, D und E) wird ein Bitpaar benötigt, um eine Farbe festzulegen.

Wert des Bitpaares	ausgewähltes Farbregister
00 = 0	COLBAK
01 = 1	COLPFO
10 = 2	COLPF1
11 = 3	COLPF2

Da nur zwei Bits zur Festlegung eines Pixels gebraucht werden, sind in jedem Bildschirm-Datenbyte 4 Pixel codiert. Beispiel: ein Byte der Bildschirm-Daten enthält den Wert $1B. Es würde 4 Pixel in den folgenden Farben anzeigen: das erste in Hintergrundfarbe, das zweite nach Farbregister 0, das dritte nach Farbregister 1 und das vierte nach Farbregister 2:

```
$1B = 00011011 = 00 01 10 11
```

In 2-Farb-Map-Modi (BASIC-Modi 4, 6, 14, und 8; ANTIC-Modi 9, B, C und F) wählt jedes Bit eines von zwei Farbregistern aus. Ein Bit-Wert von 0 bedeutet Hintergrundfarbe für das entsprechende Pixel, ein Bit-Wert von 1 wählt Farbregister 0 für das Pixel aus.

Es gibt acht unterschiedliche Map-Anzeige-Modi. Sie differieren in der Anzahl der Farben, die sie anzeigen (2 oder 4), der senkrechten Größe einer Mode-Line (1, 2, 4 oder 8 Scan-Lines) und in der Anzahl der Pixel, die waagerecht in eine Mode-Line paßen (40, 80, 160 oder 320). Das bedeutet, einige Map-Modi geben eine bessere Auflösung, benötigen dann aber entsprechend mehr RAM. In Abbildung 2.1 werden die verschiedenen Modi mit den zugehörigen Werten gezeigt:

ANTIC-Modus	BASIC-Modus	Far-ben	Scan Lines/ Mod Line	Pixel/ Mode Line	Bytes/ Zeile	Bytes/ Schirm
2	0	2	8	40	40	960
3	---	2	10	40	40	760
4	12	4	8	40	40	960
5	13	4	16	40	40	480
6	1	5	8	20	20	480
7	2	5	16	20	20	240
8	3	4	8	40	10	240
9	4	2	4	80	10	480
A	5	4	4	80	20	960
B	6	2	2	160	20	1920
C	14	2	1	160	20	3840
D	7	4	2	160	40	3840
E	15	4	1	160	40	7680
F	8	2	1	320	40	7680

Abbildung 2.1:

Charakter-Mode-Befehle bewirken, daß ANTIC eine mit Zeichen gefüllte Mode-Line anzeigt. Jedes Byte im RAM legt ein Zeichen fest. Es gibt 6 Zeichen Charakter-Anzeige-Modi. Zeichen-Displays werden in Kapitel 3 besprochen.

Blank-Line-Befehle erzeugen Leerzeilen mit fester Hintergrundfarbe. Es gibt 8 verschiedene Blank-Line Instruktionen; sie legen das Anzeigen von einer bis acht Leerzeilen (Scan-Lines) fest.

Es gibt zwei Sprungbefehle. Der erste (JMP) ist ein direkter Sprung; er lädt den Programmzähler von ANTIC mit der neuen Adresse, die der JMP-Anweisung als Operand folgt. Seine einzige Funktion ist es, eine Lösung zu einem ansonsten schwierigen Problem zu liefern: ANTICs Programmzähler besitzt nur 10 Zähler-Bits; die 6 restlichen Bits werden anderweitig benutzt. Daher kann die Display List eine 1K-Grenze normalerweise nicht überschreiten. Wenn es aber notwendig ist, eine solche Grenze zu übergehen, muß eine JMP-Instruktion verwendet werden. Dieses bedeutet, daß die Display List nicht voll „relocatable" (frei verschiebbar) ist.

Der zweite Sprung-Befehl (JVB) wird normalerweise benutzt. Er lädt den Programmzähler mit dem Wert des 0peranden und wartet auf die Ausführung eines Vertical Blanks durch den Fernseher. Diese Anweisung wird normalerweise verwendet, um eine Display Liest abzuschließen, indem zu deren Anfang zurückgesprungen wird. Durch diesen Sprung zum Anfang entsteht eine Endlos-Schleife; durch das Warten auf den Vertical Blank wird die Schleife mit der Bildfrequenz des Fernsehers synchronisiert. Sowohl der JMP-, als auch der JVB-Befehl sind 3-Byte-Anweisungen: das erste Byte ist der Opcode, das zweite und dritte Byte bilden den Operanden = anzuspringende Adresse (nieder-, danach höherwertig).

Die oben genannten 4 speziellen Einstellungen werden in den Kapiteln 5 und 6 erklärt. Die Load Memory Scan (LMS) Instruktion benötigt eine einleitende Erklärung. Diese Option wird gewählt, indem Bit 6 einer Map- oder Zeichenmodus-Anwendung gesetzt wird. Wenn ANTIC au+ solch einen Befehl stößt, wird sein Memory-Scan-Zähler mit den nachfolgenden zwei Bytes neugeladen. Dieser Memory-Scan-Zähler teilt ANTIC mit, wo sich das Bildschirm-RAM befindet. ANTIC beginnt dann damit, sich Anzeige-Daten aus diesem Bereich zu holen. Der LMS-Befehl ist eine 3-Byte-Anweisung: ein Byte Opcode gefolgt von zwei Bytes des Operanden.

In einfachen Display Lists wird der LMS-Befehl nur einmal verwendet; dieses geschieht am Anfang der Liste. Manchmal aber wird es notwendig eine zweite LMS-Anweisung anzubringen. Dieses tritt z. B. dann auf, wenn der Bildschirm-RAM eine 4K-Grenze überschreitet. Der Memory-Scan-Zähler nutzt nur 12 Bits und läßt 4 Bits unberücksichtigt; d. h. die Anzeigedaten können eine 4K-Grenze nicht überqueren. In diesem Falle muß eine neue LMS-Anweisung benutzt werden, um über die 4K-Grenze zu springen. Weiterhin bedeutet dieses, daß die Anzeigedaten nicht frei verschiebbar sind. LMS-Anweisungen besitzen noch größere Anwendungsbereiche, die später besprochen werden.

AUFSTELLEN VON DISPLAY LISTEN

Jede Display List sollte mit 3 "Blank-8-Lines"-Anweisungen beginnen. Dieses ist notwendig, um den Overscan am Bidschirmrand zu umgehen, indem die gesamte Anzeige 24 Scan Lines tiefer anfängt. Danach wird die erste benutzte Zeile festgelegt, wobei gleichzeitig die LMS-Anweisungen angebracht werden muß, um ANTIC den Anfang des Bildschirmrandes mitzuteilen. Dann wird die Display List fortgesetzt, welche die Anzeige-Bytes für die Mode-Lines auf dem Fernsehschirm auflistet. Die Gesamtzahl der Horizontal-Scan-Lines sollte 192 nicht überschreiten, darf aber darunter liegen. ANTIC berücksichtigt die Synchronisation für den Bildschirm nicht. Wenn ANTIC zu viele Scan-Lines anzeigen soll, führt er dieses zwar durch, doch das Bild kann anfangen zu laufen. Das Anzeigen von weniger als 192 Scan-Lines verursacht keinerlei Probleme; im Gegenteil, es erhöht die verfügbare Zeit des 6502s, da weniger Zyklen von ANTIC verbraucht werden. Der Programmierer muß also die Anzahl, der durch seine Display List angezeigten Horizontal-Scan-Lines berechnen und überprüfen. Die Display List endet mit einer JVB-Anweisung. Die folgende Abbildung zeigt eine Display List für BASIC Graphik-Modus 0 (alle Werte sind in hexadezimaler Notation angegeben):

```
70 Blank-8-Line
70 Blank-8-Line
70 Blank-8-Line
42 ANTIC-Modus 2 (BASIC-Modus 0)+LMS
20 die Adresse des Bildschirm-RAMs = $7C20
7C
02 ANTIC-Modus 2
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
41 Warte auf Vertical Blank
EO und springe zum Display List-Anfang
7B bei $7BEO.
```

Wie man sehen kann, ist diese Display List nicht sehr lang nur 32 Bytes. Die meisten Display Lists sind weniger als 100 Bytes lang. Außerdem sind sie von der Struktur her sehr einfach und leicht zu erstellen.

Um eine eigene Display List zu schreiben und zu implementieren, muß das Anzeige-Format entworfen werden. Dieses geschieht am besten auf einem Blatt Papier. Danach wird das entworfene Bild in eine Folge von Mode-Lines übersetzt, wobei auf die Anzahl der Scan-Lines des Displays geachtet werden muß (siehe Abbildung 2.1). Die Folge der Mode-Lines wird dann in eine Folge von Mode-Bytes übertragen. An den Anfang der Display List werden 3 Blank-8-Line Bytes ($70) gesetzt. Danach wird Bit 6 des ersten Display-List-Bytes gesetzt (d.h. das obere Nybble wird zu einer 4). Dieses ergibt das Load-Memory-Scan-Kommando. Dann folgen zwei Bytes, welche die Adresse des Bildschirm-RAMs festlegen (nieder-, dann höherwertig). Nun folgt der Rest der Display-List. Am Ende der Display List muß die JVB-Anweisung ($41) und die Anfangsadresse der Display List eingetragen werden. Wenn

dieses erfolgt ist, werden alle Bytes in´s RAM gebracht. Dieses kann überall hin geschehen, nur muß dabei darauf geachtet werden, daß sie nicht überlagern und daß die JVB-Anweisung wirklich auf den Anfang der neuen Display Liest zeigt.

Die Display List darf eine 1K-Grenze nicht überschreiten. Andernfalls muß eine JMP-Anweisung genau vor dieser 1K-Grenze plaziert werden. Der Operand ist dann die Adresse des ersten Bytes auf der anderen Seite der 1K-Grenze.

Als nächstens muß ANTIC für den Bruchteil einer Sekunde ausgeschaltet werden, damit der Display Listen-Zeiger neu eingespeichert werden kann. Dieses geschieht, indem eine Null nach SDMCTL ($22F) geschrieben wird. Danach wird die Adresse der neuen Display List in die Speicherstellen $230 und $231 (nieder-, dann höherwertig) gespeichert. Schließlich wird ANTIC wieder angeschaltet, indem eine $22 nach SDMCTL geschrieben wird. Während des Vertical Blanks, wo ANTIC inaktiv ist, lädt das Betriebssystem ANTICs Programmzähler mit den neuen Werten.

MODIFIZIEREN EINER NORMALEN DISPLAY LIST

Der Bildschirmspeicher kann irgendwo im Adressbereich des Computers abgelegt werden. Normalerweise legt die Display List den Anfang des Bildschirmspeichers mit der ersten LMS-Anweisung fest. Natürlich kann ANTIC mit jeder neuen Zeile der Display List einen neuen LMS-Befehl ausführen. Auf diese Art können Informationen aus jedem Bereich des Adressraumes des Computers auf einem Bildschirm zusammengebracht werden. Dieses kann zum Erzeugen unabhängiger Textfenster von Wert sein.

Es gibt allerdings ein paar Einschränkungen beim Plazieren des Bildschirmspeichers. Erstens: der Bildschirmspeicher kann eine 4K-Grenze normalerweise nicht überschreiten. Wird dieses aber doch nötig (wie z.B. bei BASIC-Modus 8, der 8K des Speichers benötigt), muß der Memory-Scan-Zähler mit einer LMS-Anweisung neu geladen werden. Zweitens: wenn die Bildschirm-Routinen des OS verwendet werden sollen, muß den Regeln des OS Folge geleistet werden. Dieses kann manchmal sehr schwierig sein; z.B. wenn eine geänderte Display List in einem BASIC-Programm benutzt wird. Wird eine standardmäßige Display List mit einem Programm in BASIC geändert und dann versucht, zu PLOTten oder zu PRINTen, dann führt das OS dieses unter der Annahme aus, daß die Display Liest sich noch in ihrem originalen Zustand befindet. Dieses kann zu einem unerwarteten Bild führen.

Es gibt drei Wege, auf die die Anzeige zerstört werden kann. Erstens: BASIC verweigert die Ausführung einer Bildschirmoperation, weil es sonst unmöglich ist, sie in dem Graphik-Modus, den das OS als gegeben annimmt, durchzuführen.

Das OS speichert den Graphik-Modus in Adresse $57. Es ist möglich, das OS durch das Ändern des Wertes in dieser Speicherstelle zu täuschen. Geschieht dieses, muß allerdings die Nummer des entsprechenden BASIC-Modus anstelle einer ANTIC-Modus-Nummer benutzt werden.

Der zweite Fehler kann durch das Mischen von Mode-Lines mit unterschiedlichen Speicher-Erfordernissen erfolgen. Einige Mode-Lines benötigen nur 40 Bytes pro Zeile, einige 20 Bytes und noch andere nur 10 Bytes. Angenommen, man führt eine 20 Byte Mode-Line in eine Display List mit 40 Byte Mode-Line ein. Was geschieht, wenn man nun den PRINT-Befehl verwendet? Überhalb der eingeschobenen Zeile ist alles in Ordnung, aber unterhalb sind alle Zeichen um 20 Stellen nach rechts verschoben. Dieses erfolgt aus der Annahme des Systems, daß jede Zeile 40 Bytes benötigen würde. Dementsprechend werden die Zeichen positioniert. ANTIC nimmt aber, sobald es auf die eingeschobene Zeile stößt, nur 20 Bytes von dem, was das OS in einer 40 Byte Zeile als vorhanden annimmt. ANTIC interpretiert die anderen 20 Bytes als zur nächsten Zeile gehörend an und bringt sie daher dorthin. Dadurch werden die folgenden Zeilen um 20 Stellen nach rechts verschoben.

Eine Möglichkeit, diesem Problem aus dem Weg zu gehen ist, keine BASIC PRINTs und PLOTs zu verwenden, um auf einen Bildschirm mit geänderter Display List zu schreiben. Die andere, einfachere Lösung ist, einen Schirm in Zeilengruppen zu organisieren, die ganzzahlige Vielfache der standardmäßigen Byte-Erfordernisse enthalten. Das bedeutet, in eine 40-Byte-Mode-Line darf keine 20-Byte-Mode-Line eingefügt werden; statt dessen müßen zwei 20-Byte-Mode-Lines oder eine 20-Byte-Mode-Line und zwei 10-Byte-Mode-Line eingeschoben werden. Solange das beachtet wird, tritt keine waagerechte Verschiebung auf.

Diese Lösung ist bezeichnend für das dritte Problem beim Benutzen von gemischten Display Lists in BASIC: senkrechte Verschiebungen. Das OS positioniert Bildschirmmaterial senkrecht, indem die Anzahl der zu überspringenden Bytes vom oberen Bildschirmrand berechnet werden. Bei einem standardmäßigen 40 Byte Zeilen Display würde BASIC die Zeichen auf die zehnte Zeile bringen, indem 360 Bytes vom Anfang an übersprungen werden. Wenn vier 10-Byte-Mode-Lines eingeschoben werden würden, dann würde BASIC die Zeichen drei Zeilen tiefer auf dem Schirm ausgeben. Weiterhin verbrauchen unterschiedliche Mode-Lines unterschiedlich viele Scan-Lines, so daß die Bildschirmposition nicht dem Erwarteten entspricht, sofern die Scan-Line-Erfordernisse nicht in Betracht gezogen werden.

Wie man sieht, können Displays mit gemischten Mode-Lines in Verbindung mit dem OS problematisch sein. Oft muß das OS getäuscht werden, damit solche Displays überhaupt arbeiten. Um in ein Mode-Window (=Fenster) zu PRINTen oder zu PLOTten, muß

die Nummer des BASIC-Modus´ für dieses Fenster in die Adresse $57 gebracht werden. Danach wird die Adresse des linken oberen Pixels für dieses Fenster in die Speicherstellen $58 und $59 geschrieben (nieder-, dann hochwertig). In Zeichen-Modi muß eine Position 0,0 ausgeführt werden, damit der Cursor in die linke obere Ecke des Mode-Fensters gebracht wird. In Map-Modi werden alle PLOT- und DRAWTO-Befehle ausgeführt, indem die linke obere Ecke als Koordinatenursprung des Mode-Fensters benutzt wird.

Display Lists können optimal zum Erzeugen ansprechender Bilder benutzt werden. Die augenfälligste Anwendung ist das Mischen von Text und Graphik. Es wäre z.B. möglich, einen Schirm mit einen großen Titel in BASIC-Modus 2, einem mittelgroßen Titel in BASIC-Modus 1 und einem Text in BASIC-Modus 0 aufzustellen. Ein Bild in BASIC-Modus 8 könnte in die Bildschirmmitte und ein Text an den unteren Rand gedruckt werden. Ein gutes Beispiel für diese Technik ist das Programm „Europäische Länder + Hauptstädte".

Die oben genannten Probleme schrecken wahrscheinlich von der Benutztung dieser Techniken in BASIC ab. Modifizierte Display-Lists könnten am besten durch Assembler-Routinen bearbeitet werden, wobei der Schirm in eine Reihe von Fenstern organisiert wird. Dabei besitzt jedes Fenster seine eigene LMS-Instruktion und einen unabhängigen RAM-Bereich.

VERWENDUNG VON MODIFIZIERTEN DISPLAY LISTS

Eine andere bedeutende graphische Möglichkeit, die nur über die Manipulation von Display Lists erreicht werden kann, ist z.B. der Zugriff auf in BASIC nicht verfügbare Graphik-Modi des Gerätes. Es gibt drei Text-Modi, die in BASIC nicht verfügbar sind, aber von ANTIC erzeugt werden können. Lediglich die Manipulation der Display List gestattet dem Benutzer den Zugriff auf diese Modi. Es gibt außerdem die Möglichkeiten des „Display List Interrupts" und des „Feinen Scrollings", die nur erreichbar sind, nachdem die Display List vom Programmierer geändert wurde. Diese Dinge sind das Thema von Kapitel 5 und 6.

Manipulation mit der LMS-Anweisung und ihrem Operanden eröffnen dem kreativen Programmierer viele Möglichkeiten. So können z.B. während des Vertical Blanks die Bilder auf dem Schirm geändert werden. Dieses kann in einer „langsameren" Geschwindigkeit geschehen, um zwischen vorgezeichneten Bildern zu wechseln, ohne jedes neu zeichnen zu müßen. Jedes Bild wäre weiterhin im RAM vorhanden und wäre daher sofort verfügbar. Durch schnelles Wechseln einer Folge von Displays kann eine sogenannte zyklische Animation erreicht werden. Das hierfür erforderliche Programm müßte lediglich zwei Adress-Bytes manipulieren, um tausende von RAM-Bytes anzuzeigen.

Es ist außerdem möglich, Bilder übereinander zu legen, indem mit hoher Geschwindigkeit gewechselt wird. Das menschliche Auge hat eine zeitmäßige Auflösung von einem 16tel einer Sekunde. So kann ein Programm zwischen vier Bildern wechseln, so daß eines jeden 15ten Teil einer Sekunde wiedererscheint. Auf diese Weise können sich scheinbar bis zu vier Bilder gleichzeitig auf dem Schirm befinden.

Natürlich gibt es einige Nachteile bei dieser Methode. Als erstes brauchen vier separate Bilder einen großen Bereich des Speichers. Zum zweiten erscheinen die einzelnen Bilder verwaschen, weil sie nur jedes 4. Mal zu sehen sind. Außerdem muß der Hintergrund schwarz sein, damit sich die Bilder klar abzeichnen. Weiterhin gibt es ein unangenehmes Flackern beim Benutzen dieser Technik. Ein konservativer Programmierer wird es vorziehen, nur zwischen drei oder sogar nur zwischen zwei Bildern zu wechseln.

Diese Technik kan auch dazu verwendet werden, die Farb- und Helligkeitsauflösung des Displays zu steigern. Dieses geschieht, indem man zwischen vier Versionen des gleichen Bildes wechselt, wobei jede Version andere Farb- und Helligkeitswerte besitzt. Hierdurch ist ein breites Spektrum an Farbe und Helligkeit verfügbar. Wenn z.B. ein Balken gezeichnet werden soll, der verschiedene Helligkeitsstufen besitzt, dann werden als erstes die vier Farbregister auf die folgenden Werte gesetzt:

Hintergrund: 00
Playfield 1: 02
Playfield 2: OA
Playfield 3: OC

Danach setzen wir die folgenden Bilder in verschiedene RAM-Bereiche:

Pixelinhalte (Auswahl der Playfield-Register)

1. Reihe:	1	1	1	1	2	3	2	3	2	3	2	3
2. Reihe:	B	1	1	1	B	B	2	3	2	3	2	3
3. Reihe:	B	B	1	1	B	B	B	B	2	3	2	3
4. Reihe:	B	B	B	1	B	B	B	B	B	B	2	3

Effektive Helligkeit x 4:

2 4 6 8 10 12 20 24 30 36 40 48

Erzielte Helligkeit:

Auf diese Weise ist eine viel höhere Helligkeits- und Farbauflösung möglich.

Ein abschließender Gedanke betrifft ein Objekt, das viele Möglichkeiten liefert, aber bis jetzt noch schwer zugänglich und verständlich ist: die dynamische Display-List. Dieses ist eine Display-List, die der 6502 während des Vertical Blanks ändert. Es sollte möglich sein, interessante Effekte mit Hilfe der dynamischen Display-List zu produzieren. Ein Text-Editierprogramm könnte z.B. dynamisch Leerzeilen über und unter die zu editierende Zeile setzen, um sie von den anderen abzuheben. Wenn der Cursor sich senkrecht bewegt, ändert sich die Display-List. Diese Technik ist sicherlich umständlich, aber auch sehr effektiv.

Kapitel 3
Indirekte Graphik Adressierung
(Farbregister & Zeichensätze)

Indirekte Adressierung ist eine leistungsstarke Möglichkeit, die aber für den Programmieranfänger schwer zu verstehen ist. Im Assembler des 6502s besteht eine Auswahl zwischen 3 verschiedenen Stufen der indirekten Adressierung. Die erste und einfachste (direkteste) Stufe ist die direkte Adressierung, bei welchem die zu ladende Zahl selbst angegeben wird:

```
LDA #$FA
```

Die zweite Stufe (indirekte Adressierung) liegt vor, wenn das Programm auf eine Speicheradresse verweist, welche die zu verwendende Zahl enthält:

```
LDA $0602
```

Die dritte und zugleich höchste Stufe der indirekten Adressierung beim 6502 wird erreicht, sobald das Programm auf ein Paar von Speicherstellen verweist, die zusammen die Adresse enthalten, in welcher wiederum die zu benutzende Zahl steht. Die Möglichkeiten der Anweisung werden beim 6502-Assembler durch das Anfügen eines Indexes vergrößert:

```
LDA ($DO),Y
```

Je höher die Stufe der indirekten Adressierung ist, desto allgemeiner und damit leistungsfähiger ist sie. Anstelle jedesmal die gleichen Zahlen einzeln laden zu müßen, kann der Programmierer durch einfaches Ändern eines Zeigers ganze Datenblöcke lesen lassen. Indirekte Adressierung ist offenbar eine wichtige Möglichkeit beim Programmieren.

Indirekte Graphik-Adressierung ist auf zwei Arten im ATARI”TM” Personal Computer System realisiert: durch Farbregister und durch Zeichensätze. Benutzer, die zum ersten Mal mit diesem Gerät arbeiten, nachdem sie auf anderen Maschinen programmiert haben, denken oft in direkten Farbformen. Eine Farbe ist zwar ein andaürnder Wert, ein Farbregister jedoch funktioniert auch indirekt; es kann jeden Farbwert enthalten. Ein Farbregister ist flexibler als eine Farbe, allerdings auch schwerer zu handhaben.

Es gibt im ATARI Computer System 9 Farbregister; vier davon werden für Player-Missile-Graphiken benutzt, welche in Kapitel 4 besprochen werden. Von den 5 übrigen Registern werden nicht immer alle benutzt; die verwendete Anzahl hängt vom Graphik-Modus ab. Im BASIC-Modus 0 werden nur 1 ½ Farbregister

benutzt, da der Farbwert der Buchstaben (und Zeichen) ignoriert wird; Buchstaben besitzen die Farbe, die durch Playfield-Register 2 angegeben wird, bekommen aber ihren Helligkeitswert aus Register 1. Die Farbregister befinden sich bei den Adressen $D016 bis $D01A des CTIA. Sie werden vom OS während der Vertical Blanks in den RAM-Bereich übertragen. Abbildung 3.1 zeigt Hardware- und Schatten-Adressen der einzelnen Farbregister.

Beeinflußtes Objekt		Hardware-Adresse		Schatten-Adresse	
		LABEL	ADRESSE	LABEL	ADRESSE
Player	0	COLPMO	D012	PCOLR0	2C0
Player	1	COLPM1	D013	PCOLR1	2C1
Player	2	COLPM2	D014	PCOLR2	2C2
Player	3	COLPM3	D015	PCOLR3	2C3
Playfield	0	COLPFO	D016	COLOR0	2C4
Playfield	1	COLPF1	D017	COLOR1	2C5
Playfield	2	COLPF2	D018	COLOR2	2C6
Playfield	3	COLPF3	D019	COLOR3	2C7
Hintergrund		COLBK	D01A	COLOR4	2C8

Abbildung 3.1:
Farbregister-Label und -Adressen

In den meisten Fällen kontrolliert der Benutzer die Farbregister, indem er in ihre SCHATTEN-Adresse schreibt. Es gibt lediglich zwei Situationen, in denen der Programmierer direkt in die CTIA-Adressen schreiben würde: beim Display List Interrupt (siehe Kapitel 5), oder beim Abschalten der Vertical-Blank-Interrupt-Routinen des OS. Geschieht dieses, werden die Werte nicht mehr automatisch in die CTIA-Register übertragen. Vertical Blank Interrupts sind das Thema von Anhang I.

Farben werden über eine einfache Formel in die Farbregister codiert. Das obere Nybble nennt den Farbwert, der identisch zum zweiten Parameter des SETCOLOR-Befehls in BASIC ist. Tabelle 9.3 des BASIC-Referenz Manuals listet die Farbwerte auf. Das untere Nybble des Farbregisters gibt den Helligkeitswert an. Dieser entspricht dem dritten Parameter des SETCOLOR-Kommandos. Das niederwertigste Bit dieses Nybbles hat keine Bedeutung, so daß es für jede Farbe 8 Helligkeitswerte gibt. Daraus ergibt sich eine Gesamtzahl von 128 Farben, unter denen sich der Benutzer entscheiden kann (8 Helligkeits-Stufen mal 16 Farben). In diesem Buch bezeichnet das Wort „Farbe” eine Farb-Helligkeits-Kombination.

Sobald eine Farbe in einem Register codiert wurde, wird sie auf den Schirm gebracht, indem auf das Farbregister verwiesen

wird, das diesen Farbwert enthält. In Map-Modi, bei denen 4 Farben angezeigt werden, bestimmen die Bilddaten, welches Farbregister auf dem Schirm geworfen werden soll. Da es vier Register gibt, werden nur zwei Bits zur Definierung eines Pixels benötigt. Dieses wiederum bedeutet, daß jedes Byte der Bildschirmdaten 4 Pixel festlegt. Die Werte in jedem Bit-Paar legen das Farbregister fest, welches die Farbe für das entsprechende Pixel liefert.

In Text-Modi (BASIC-Modi 1 und 2) wird die Auswahl des Farbregisters durch die zwei obersten Bits des Zeichencodes getroffen. Dieses läßt andererseits nur 6 Bits zur Auswahl des Zeichens über. Daher stehen bei diesen beiden Modi nur 64 Zeichen anstelle von 128 + Inversen zur Verfügung.

Indirekte Farbadressierung liefert dem Programmierer vier spezielle Möglichkeiten. Erstens kann er zwischen 128 verschiedenen Farben für seine Displays wählen. Es gibt also sicherlich einen Farbton, der seinen Wünschen entspricht bzw. nahekommt.

Zweitens kann der Programmierer die Farbregister in Echtzeit manipulieren, woraus sich hübsche Effekte ergeben können. Die folgende BASIC-Zeile zeigt hierfür ein einfaches Beispiel:

```
FOR I=0 TO 254 STEP 2:POKE 712,I:NEXT I
```

Diese Zeile bewirkt, daß die Randfarbe alle Farbtöne durchläuft. Der Effekt ist sehr ansprechend und zieht sicherlich die Aufmerksamkeit des Benutzers an. Diese grundlegende Technik kann auf eine Vielzahl von Arten erweitert werden. Eine spezielle Variation hiervon ist die einfache zyklische Animation, indem zuerst eine Figur gezeichnet wird. Die Animation wird dann durch Änderung der zugehörigen Farbregister realisiert. Das folgende Programm verdeutlicht die Idee:

```
10 GRAPHICS 23
20 FOR X=0 TO 39
30 FOR I=0 TO 3
40 COLOR I
50 PLOT 4*X+1,0
60 DRAWTO 4*X+I,95
70 NEXT I
80 NEXT X
90 A=PEEK(712)
100 POKE 712,PEEK(710)
110 POKE 710,PEEK(709)
120 POKE 709,PEEK(708)
130 POKE 708,A
140 GOTO 90
```

Die dritte Verwendung von Farbregistern ist die Zuordnung bestimmter Farben zu bestimmten Situationen. Bei einem

seitenmäßig angelegten Menü kann z.B. die Hintergrundfarbe geändert werden, wenn auf eine andere Menüseite gegangen wird. Der Bildschirm kann außerdem rot blinken, wenn eine nicht zuläßige Taste gedrückt wurde. Die Verwendung von Farbbuchstaben der BASIC-Modi 1 und 2 können die Wirkung und Übersichtlichkeit eines Textes erhöhen. Es ist möglich, eine Rechnung rot ausgeben zu lassen, wenn ihr Ergebnis negativ ist; bei einem positiven Resultat könnte sie in schwarz ausgedruckt werden. Es ist auch denkbar, wichtige Wörter oder Sätze eines Textes in speziellen Farben auszugeben, damit sie sich von den anderen abheben.

Die Wirkung von Graphiken in Map-Modi (kein Text) wird durch die Benutzung von Farben ebenfalls verbessert. Verschiedene Versionen eines Objektes könnten durch unterschiedliche Färbung des gleichen Bildes erreicht werden. Es wird relativ viel RAM benötigt, um ein Bild zu speichern, aber wenig, um die Farbe eines schon vorhandenen Bildes zu ändern. Es ist z.B. einfacher, drei verschiedene Boote zu zeigen, indem dreimal das gleiche Boot in anderen Farben gezeigt wird, als wenn drei unterschiedliche Bootsformen gezeigt werden.

Die vierte und wichtigste Verwendung von Farbregistern besteht in der Verbindung mit Display List Interrupts. Ein einzelnes Farbregister kann dabei zum Anzeigen von bis zu 128 verschiedenen Farben auf einem Bildschirm benutzt werden. Diese wichtige Möglichkeit wird in Kapitel 5 besprochen.

ZEICHENSÄTZE

Indirekte Adressierung liegt auch bei der Neudefinition eines Zeichensatzes vor. Im ROM befindet sich zwar ein standardmäßiger Zeichensatz, aber es gibt keinen Grund, warum ausgerechnet dieser immer benutzt werden muß. Der Benutzer ist in der Lage, jeden beliebigen Zeichensatz zu entwerfen und vom Computer anzeigen zu lassen. Dazu sind drei Schritte erforderlich. Als erstes muß der Programmierer natürlich einen Zeichensatz entwerfen, was auch der zeitraubendste Vorgang ist. Jedes Zeichen wird auf dem Bildschirm in einem 8x8-Gitter angezeigt; im Speicher ist es als eine 8-Byte-Tafel codiert. Abbildung 3.2 zeigt die Codierungs-Anordnung:

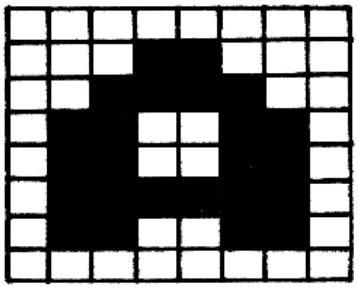
Zeichen	Binär-Darstellung	Hexadezimale-Darstellung
	00000000	00
	000**11**000	18
	00**1111**00	3C
	0**11**00**11**0	66
	0**11**00**11**0	66
	0**111111**0	7E
	0**11**00**11**0	66
	00000000	00

Abbildung 3.2:
Codierung des Buchstaben "A"

Ein vollständiger Zeichensatz besteht aus 128 Zeichen, jeweils normal und invertiert (d.h. Nullen werden zu Einsen und umgekehrt). Ein solcher Zeichensatz benötigt 1024 Bytes und muß bei einer 1K-Grenze beginnen. Die Zeichensätze der BASIC-Modi 1 und 2 bestehen aus nur 64 verschiedenen Zeichen. Daher brauchen sie nur die Hälfte der oben angegebenen Speichermenge - 512 Bytes. Dieser Zeichensatz muß bei einer ½K-Grenze beginnen. Die ersten 8 Bytes legen den ersten Buchstaben fest, die nächsten 8 Bytes den zweiten usw. Das Definieren eines neuen Zeichensatzes ist offensichtlich eine umfangreiche Arbeit. Hierfür gibt es aber glücklicherweise schon Software auf dem Markt, die diese Arbeit erleichtert.

Sobald ein Zeichensatz definiert und in´s RAM gebracht wurde, muß dem ANTIC-Prozessor mitgeteilt werden, wo dieser sich befindet. Dieses geschieht, indem die Seitennummer (page) der Start-Adresse des Zeichensatzes in die Speicherstelle $D409 (dezimal 54281) geschrieben wird. Die Schattenadresse, welche normalerweise verwendet wird, trägt die Bezeichnung CHBAS und liegt bei $2F4 (dezimal 756). Der dritte Schritt beim Benutzen eigener Zeichensätze besteht in der Ausgabe des Zeichens auf dem Bildschirm. Dieses kann wie gewöhnlich durch einfache PRINT-Befehle in BASIC oder durch direktes Schreiben von Daten in den Bereich der Bildschirmspeichers geschehen.

Eine spezielle Möglichkeit des Systems, die in BASIC nicht verfügbar ist, besteht in den 4-Farb-Zeichen-Modi (bei der XL-Serie sind diese Modi verfügbar). Die BASIC-Modi 1 und 2 liefern zwar 5 Farben aber jeder Buchstabe ist zweifarbig; er besitzt eine Vordergrund- und eine Hintergrundfarbe. Die Vordergrundfarbe kann eine der vier möglichen Farben sein, wobei aber pro Zeichen nur eine ausgewählt werden kann.

Dieses ist ein Hindernis bei der Erstellung von Zeichensatz-Graphiken, z. B. bei geographischen Karten. Es gibt
allerdings noch zwei weitere Text-Modi, die speziell für solche Fälle vorgesehen sind. Es sind die ANTIC-Modi 4 und 5. Jedes Zeichen hat bei diesen Modi nur eine Breite von vier Pixeln, aber jedes Pixel kann eine von vier Farben annehmen (einschließlich Hintergrund). Die Zeichen werden genau wie BASIC-Modus 0 definiert, wobei aber jedes Pixel doppelte Breite und zwei Bits zum Festlegen der Farbe besitzt. Anders als bei den ANTIC-Modi 6 und 7 (BASIC-Modi 1 und 2) wird die Auswahl des Farbregisters nicht durch die einzelnen Bitpaare innerhalb eines Zeichens getroffen. Jedes Byte der Zeichentabelle wird in 4 Bitpaare aufgespalten, von denen jedes die Farbe für den zugehöriges Pixel bestimmt (aus diesem Grunde gibt es waagerecht nur 4 Pixel in jedem Zeichen). Das höchste Bit des Zeichennamens (D7) modifiziert das benutzte Farbregister. Diese Auswahl wird in Abbildung 3.3 dargestellt:

Bitpaar	D7=0	D7=1
00	COLBAK	COLBAK
01	PF0	PFO
10	PF1	PF1
11	PF2	PF3

Abbildung 3.3
Farbregister-Auswahl bei 4-Farb-Zeichen

Durch Benutzen dieser Text-Modi können vielfarbige Zeichen-Graphiken auf den Bildschirm gebracht werden.

Ein weiterer interessanter ANTIC-Zeichen-Modus ist der Kleinbuchstaben-Unterlängen-Modus (ANTIC-Modus 3). Dieser Modus zeigt 10 Scan-Lines in einer Mode-Line an, da Buchstaben aber nur 9 Bytes senkrecht benutzen, ist anzunehmen, daß die unteren beiden Reihen leer sind. Dies trifft zu, solange das angezeigte Zeichen aus den drei ersten Vierteln des Zeichensatzes stammt. Steht das Zeichen aber im letzten Viertel, so bleiben die oberen beiden Scan-Lines leer und die normalerweise dort hingehörenden Daten werden in den unteren beiden Scan-Lines angezeigt. Dieses gestattet dem Benutzer das Entwerfen von Kleinbuchstaben mit Unterlängen (z.B. „g" & „y").

Durch indirekte Zeichensatz-Adressierung ergeben sich interessante und nützliche Möglichkeiten. Die offensichtlichste Anwendung ist der modifizierte Schriftsatz. Ein neuer Zeichensatz gibt einem Programm ein eigenes Gesicht. Es ist z.B. möglich, einen griechischen, kyrillischen oder einen anderen Schriftsatz aufzustellen. Außerdem kann der Benutzer graphische Zeichensätze einsetzen. Das „ENERGY

CZAR”TM””-Programm benutzt einem neudefinierten Zeichensatz um Balkendiagramme anzeigen zu können. Ein Zeichen umfaßt 9 Pixel; das bedeutet, daß Balkendiagramme mit normalen Zeichen nur eine grobe Auflösung von jeweils 8 Pixeln besitzen. Bei ENERGY CZAR wurde ein Zeichensatz entworfen, bei dem die normalerweise weniger benutzten Symbole („#", „$” usw.) durch spezielle Zeichen für ein Balkendiagramm ersetzt werden. Ein solches Zeichen wurde zu einem Ein-Pixel-Balken, ein weiteres zu einem Zwei-Pixel-Balken usw. bis zum 8-Pixel-Balken. So ist es dem Programm möglich, detaillierte Balkendiagramme zu zeichnen, die eine Auslösung von maximal einem Pixel besitzen. Abbildung 3.4 zeigt ein typisches Bild dieses Programms. Es scheint, als ob Text- mit Graphik-Modi gemischt wurden; in Wahrheit besteht das gesamte Display aus Zeichen.

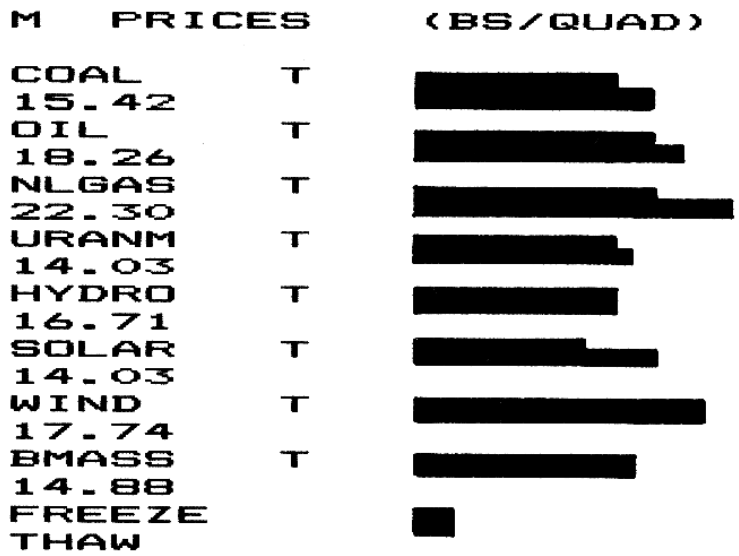


Abbildung 3.4
Balkendiagramm des "ENERGY CZAR´TM´"-Programms

Es gibt viele Möglichkeiten, bei denen Zeichensätze geschaffen und benutzt werden können, um spezielle Bilder zu zeigen. Man könnte z. D. einen Zeichensatz für Landschaften entwerfen, der jeweils Zeichen für Flüsse, Wälder, Berge usw. enthält. Mit diesem Satz wäre es dann machbar, Karten jedes Gebietes auf der Erde aufzubauen. Gleiches gilt auch für andere Planeten. Mit ein wenig Phantasie ist die Erstellung eines hierfür benötigten Zeichensatzes möglich.

Beim Aufbauen solcher Landschafts-Zeichensätze sollte für jeden Oberflächentyp 5 bis 8 verschiedene Zeichen entworfen werden. Dadurch wird ein eintöniges Aussehen der Karte vermieden, das charakteristisch für einfache Graphiken dieser Art ist. Die meisten Menschen werden nicht bemerken, daß eine Karte, die mit verschiedenen Variationen eines Oberflächentyps

ausgestattet ist, mit einem Zeichensatz anstelle von Graphik erstellt worden ist.

Die Schwarz-weiße Abbildung wird dem Original allerdings nicht gerecht, welches z.B. bis zu 19 verschiedene Farben gleichzeitig enthält.

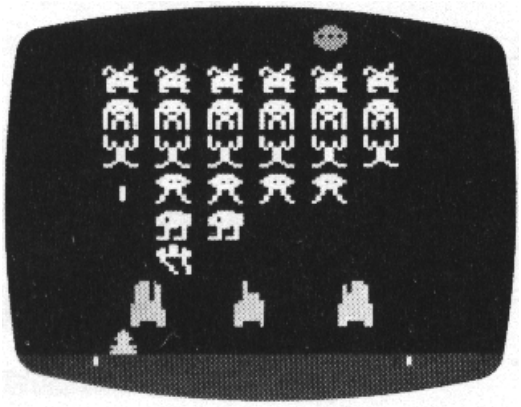


Abbildung 3.5
Animation mit Zeichensatz-Graphik

Es wäre möglich, einen Elektronik-Zeichensatz zu entwerfen, der Symbole für Transistoren, Dioden, Spulen usw. enthält. Mit einem solchen Zeichensatz könnten elektrische Schaltkreise aufgebaut werden. Gleiches wäre mit einem Architektur-Zeichensatz für Wohnraumgestaltung machbar. Die Möglichkeiten für Zeichensatz-Graphiken sind schier unerschöpflich.

Zeichen können auf den Kopf gestellt werden, indem eine vier in die Adresse 755 geschrieben wird. Diese Möglichkeit wäre z.B. beim Anzeigen von Spielkarten auf dem Bildschirm nützlich. Die obere Hälfte einer Karte würde richtig herum stehen, die untere Hälfte auf dem Kopf. Dieser Trick könnte auch beim Anzeigen von Spieleffekten Anwendung finden (z.B. Seen, Swimmingpools usw.).

Eine weitere leistungsstarke Anwendung von Zeichensätzen besteht im Wechseln zwischen mehreren Sätzen, während das Programm läuft. Ein Zeichensatz braucht entweder 512 oder 1024 Bytes. In beiden Fällen ist es relativ „billig” (im Bezug auf den benötigten Speicher), mehrere Zeichensätze im RAM zu haben und zwischen ihnen zu wechseln. Es gibt drei Geschwindigkeitsstufen für solches Zeichensatz-"Multiplexing": menschlich langsam (länger Aus als 1 Sekunde), menschlich schnell (eine 60stel Sekunde bis 1 Sekunde), maschinenmäßig schnell (schneller als ein 60stel Sekunde).

Menschlich langsames Zeichensatz-Multiplexing ist sinnvoll beim Ändern von Szenen. Ein Raumfahrt-Programm könnte z.B. verschiedene Zeichensätze für Planeten und Weltraum haben. Ändert der "reisende" Benutzer den Standort, wechselt das Programm den Zeichensatz. Besonders praktisch ist diese Methode bei sogenannten "Adventure"-Spielen, bei denen die Position des Spielers relativ häufig wechselt.

Menschlich schnelles Multiplexen ist an erster Stelle sinnvoll für Animation. Dieses kann auf zwei Arten geschehen: entweder werden die Zeichen innerhalb eines Zeichensatzes geändert, oder der gesamte Zeichensatz wird gewechselt. Das „SPACE-INVADERS (Warenzeichen der Taito America Corp.)"-Programm auf dem ATARI Computer benutzt die erste Technik. Die Gegner sind in Wahrheit alles Zeichen, die durch rasches Wechseln animiert werden. Da es nur 6 verschiedene Monster mit 4 Darstellungen gibt, ist dieses relativ einfach.

Ist jedes Zeichen eine leicht geänderte Variation des gleichen Objektes, so durchläuft dieses eine Animations-Sequenz, sobald zwischen den Zeichen gewechselt wird. Auf diese Art kann ein ganzer Bildschirm mit Objekten in Bewegung gebracht werden. Befinden sich die Zeichensätze einmal im Speicher, so ist es lediglich erforderlich, die folgende Programmschleife durchlaufen zu lassen:

```
1000 FOR I=1 TO 10: REM ANZAHL DER ZEICHENSÄTZE
1010 POKE 756, CHARBASE(I):REM ADRESSE DES JEWEILIGEN SATZES
1020 NEXT I
1030 GOTO 1000
```

Computermäßig schnelle Zeichensatz-Animation wird benutzt, um mehrere verschiedene Zeichensätze auf einen Bildschirm zu bringen. Dabei wird die Display List Interrupt-Möglichkeit des Gerätes angewandt, die in Kapitel 5 besprochen wird.

Die Benutzung von Zeichensätzen für Grafik und Animation besitzt viele Vorteile und nur wenige Einschränkungen. Der größte Vorteil ist, daß nur wenig RAM gebraucht wird, um detaillierte Displays zu produzieren. Ein Graphik-Display, das BASIC-Modus-2-Zeichen verwendet (wie in Abbildung 3.5 gezeigt), kann besser Details und Farbe anzeigen, als ein Display im BASIC-Modus 7. Außerdem benötigt das Bild aus Zeichen 200 Bytes, wogegen das Map-Modus-Display ganze 4000 Bytes verbraucht.

Für jeden Zeichensatz werden 512 Bytes des Speichers benötigt. Aus diesem Grund ist es praktischer, mehrere Zeichensätze gleichzeitig im RAM zu haben. Schirmmanipulationen mit Zeichensätzen sind schneller, da nur wenige Datenz zu verändern sind. Natürlich sind Zeichensatz-Graphiken nicht so flexibel wie Map-Modus Graphiken, da nicht alles überallhin

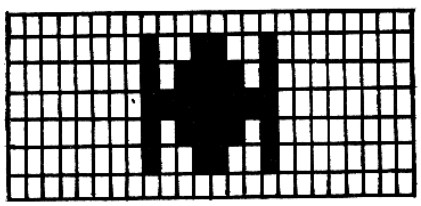
auf den Bildschirm gebracht werden kann. Dieses unterbindet die Verwendung von Zeichensätzen in bestimmten Fällen. Andererseits gibt es viele Fälle, in denen eine nur geringe Anzahl von Figuren an festen Positionen gezeigt werden muß. Hierbei sind Zeichensätze von einzigartigem Nutzen.

Kapitel 4
Player-Missile-Graphiken

Animation ist eine wichtige Fähigkeit eines jeden Personal Computer Systems. Sie erhöht die Begeisterungsfähigkeit und den Realismus eines Programms. Noch wichtiger aber ist, daß ein bewegtes Bild mehr Information mit größerer Klarheit übertragen kann, als ein stehendes Bild. Durch Bewegung eines Objektes wird die Aufmerksamkeit des Benutzers hierauf gezogen. Ein dynamischer Vorgang wird durch Animation deutlicher, als wenn er indirekt beschrieben wird. Animation ist logischerweise entscheidend für die Wirkung vieler Computerspiele. Animation muß folglich als ein wichtiges Element der Graphikmöglichkeiten eines Computers angesehen worden.

Der normale Weg, Animation zu erreichen, ist, die Bilddaten durch den Bereich des Bildschirm-RAMs zu bewegen. Dieses erfordert einen 2-Stufen-Prozess. Als erstes muß das Programm das alte Bild löschen, indem es Hintergrund (Farb-) -Werte in den Bereich schreibt, der das augenblickliche Bild enthält. Danach müssen die Bilddaten in den RAM-Bereich geschrieben werden, welcher der neuen Bildposition entspricht. Durch wiederholtes Ausführen dieses Vorgangs erscheint auf dem Bildschirm ein sich bewegendes Bild.

Bei dieser Technik tauchen allerdings zwei Probleme auf. Erstens: wird die Animation mit größeren Pixeln ausgeführt, so ist die Bewegung nicht mehr „gleitend"; das Bild bewegt sich ruckartig über den Schirm. Bei vielen Computern besteht die einzige Lösung dieser Schwierigkeit in der Wahl kleinerer Pixel (höhere Auflösung). Das zweite Problem ist allerdings weit wichtiger. Der Bildschirm ist ein zweidimensionales Gebilde, wogegen der gesamte (und damit auch der ihm zugehörige) RAM-Bereich eindimensional organisiert ist. Das bedeutet, daß Objekte, welche auf dem Schirm nebeneinander liegen, im RAM nicht hintereinander liegen. Abbildung 4.1 verdeutlicht dieses.

Bild	Entsprechende Bytes im RAM-Bereich
	00 00 00 00 99 00 00 BD 00 00 FF 00 00 BD 00 00 00 00

Verteilung der Bild-Bytes im RAM

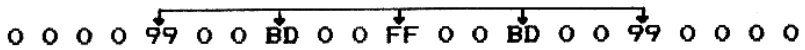


Abbildung 4.1
Bilder auf dem Schirm liegen im RAM nicht hintereinander

Die Bedeutung dieses Umstandes wird erst offensichtlich, wenn man ein Programm zum Bewegen eines solchen Bildes schreibt. Um die im RAM verstreuten Bytes zu löschen, muß das Programm ihre jeweilige Adressen berechnen. Diese Berechnung ist im allgemeinen nicht sehr einfach. Ein Assembler-Programm, das auf ein einzelnes Byte an der Bildschirmposition XPOS, YPOS zugreift, sieht wie folgt aus (es wird eine Zeilenlänge von 40 Zeichen vorausgesetzt):

```
LDA  SCRNRM    Anfangsadresse des Bildschirm-RAMs
STA  POINTR    Zero-Page-Zeiger
LDA  SCRNRM+1  Höherwertiges Byte der Adresse
STA  POINTR+1  Höherwertiger Zeiger
LDA  #$00
STA  TEMPA+1   Zwischenregister
LDA  YPOS      Senkrechte Position
ASL  A         Mal 2
ROL  TEMPA+1   Schiebe Übertrag nach TEMPA+1
ASL  A         Mal 4
ROL  TEMPA+1   Schiebe ein weiteres Mal
ASL  A         Mal 8
ROL  TEMPA+1   Schiebe erneut
LDX  TEMPA+1   Speichere YPOS mal 8
STX  TEMPB+1   in TEMPB
STA  TEMPB     Niederwertiges Byte
ASL  A         Mal 16
ROL  TEMPA+1
ASL  A         Mal 32
ROL  TEMPA+1
CLC
ADC  TEMPB     Addiere zu YPOS x 8 um YPOS x 40 zu erhalten
STA  TEMPB
LDA  TEMPA+1   Nun bearbeite das höherwertige Byte
ADC  TEMPB+1
STA  TEMPB+1
LDA  TEMPB     TEMPB enthaelt den Offset vom oberen
CLC            Bildschirmrand zum Pixel
ADC  POINTR
STA  POINTR
LDA  TEMPB+1
ADC  POINTR+1
STA  POINTR+1
LDY  XPOS
LDA  (POINTR),Y
```

Es ist klar, daß dieses Programm zu „schwerfällig" zum Zugriff auf eine Bildschirmadresse ist. Dieses ist also nicht die schnellste und eleganteste Methode, um das Problem zu lösen; ein guter Programmierer könnte allerdings das Programm mit Hilfe von speziellen Schaltungen verdichten und kompakter machen. Der springende Punkt hierbei ist aber, daß das Zugreifen auf Pixel auf dem Schirm sehr viel Berechnung benötigt. Die obere Routine braucht ungefähr 100 Maschinenzyklen, um auf ein einziges Bildschirm-Byte zuzugreifen. Um ein Bild zu bewegen, das eine Größe von 50 Bytes besitzt, würden 100 solcher Zugriffe benötigt, was 10.000 Maschinenzyklen oder ca. 10 Millisekunden entspricht. Dieses klingt nicht nach viel, aber wenn eine gleitende Bewegung erreicht werden soll, muß ein Objekt alle 17 Millisekunden bewegt werden. Werden nun noch andere Objekte bewegt oder Berechnungen ausgeführt, dann hat der Prozessor nicht mehr viel Zeit, um diese zu bearbeiten. Hinzu kommt allerdings noch, daß diese Art der Animation (sog. „Playfield"-Animation: Playfield = Spielfeld) für viele Zwecke zu langsam ist. Animation kann natürlich trotzdem auf diesem Wege erreicht werden, sie muß dann allerdings auf wenige (oder kleinere) Objekte oder langsamere Bewegung beschränkt werden. Die bei dieser Art der Animation vom Programmierer zu machenden Abstriche sind also schwerwiegend.

Die ATARI-Computer-Lösung dieses Problems sind die sogenannten „Player-Missile" Graphiken. Um diese Graphik-Möglichkeit zu verstehen, ist es wichtig, daß der Leser das grundsätzliche Problem der Playfield-Animation begreift: das Bild auf dem Schirm ist zweidimensional, wogegen das Bild im RAM eindimensional ist. Die Lösung dieser Schwierigkeit besteht in der Schaffung eines Bildes, das sowohl im RAM, als auch auf dem Bildschirm eindimensional ist. Dieses Objekt, genannt Player, erscheint im RAM als ein 128 bzw. 256 Bytes großer Bereich. Dieser Bereich wird aus dem Speicher direkt auf den Bildschirm projeziert. Es erscheint auf dem Bildschirm als ein senkrechtes Band, das vom oberen bis zum unteren Rand reicht. Jedes Byte dem Bereichs entspricht entweder einer oder zwei Horizontal Scan Lines. Diese Auswahl trifft der Programmierer. Das Bild auf dem Schirm ist ein einfacher bitweiser Auswurf der Daten dieses Speicherabschnittes. Besitzt ein Bit in diesem Bereich den Wert 1, so leuchtet das entsprechende Pixel auf der Mattscheibe; enthält es den Wert 0, so ist das Pixel dunkel. Player-Bilder sind also nicht strikt eindimensional; sie sind 8 Bits breit.

Das Zeichnen eines Players auf dem Schirm ist sehr einfach. Als erstes muß das gewünschte Bild auf kariertem Papier entworfen werden. Dieses Bild darf allerdings nicht breiter als 8 Pixel sein. Danach wird das Bild in Binär-Code umgesetzt, d.h. für jedes erleuchtete Pixel wird eine 1 notiert; für jedes dunkle Pixel eine 0. Die so entstehende

Zahl wird in eine Dezimal- oder Hexadezimalzahl übersetzt, abhängig davon, was dem Programmierer vertrauter ist. Schließlich werden die Bilddaten in den Player-RAM-Bereich geschrieben, wobei das erste Byte des Player-Bildes oben und das letzte unten im Speicher steht (d.h. bei niedriger bzw. höherer Adresse). Je weiter unten die Bilddaten innerhalb des entsprechenden Bereichs im RAM plaziert werden, desto weiter unten erscheint das Bild auf der Mattscheibe.

Die Animation dieses Bildes geht sehr einfach von statten. Senkrechte Bewegung wird erreicht, indem die Bilddaten durch den RAM-Bereich geschoben werden. Dieses ist im Prinzip die gleiche Methode, die auch bei der Playfield-Animation verwendet wird. Es gibt allerdings einen großen Unterschied: die Bewegungsroutine für senkrechte Bewegung ist eine eindimensionale, anstelle einer zweidimensionalen Bewegung. Das Programm muß zur Errechnung der senkrechten Position des Objektes die Zeilenlänge berücksichtigen, und es wird nur sehr selten von indirekter Adressierung Gebrauch gemacht. Ein Programm für diese Aufgabe könnte folgendermaßen aussehen:

```
     LDX #$01
LOOP LDA PLAYER,X
     STA PLAYER-1,X
     INX
     BNE LOOP
```

Diese Routine benötigt ungefähr 4 Millisekunden, um den gesamten Player zu bewegen; das entspricht fast der halben Zeit, die die Playfield-Animation benötigt. Weiterhin bewegt die Playfield-Animation nur 50 Bytes, wogegen die obige Routine 256 Bytes verschiebt. Wenn noch höhere Geschwindigkeiten nötig sind, könnte die Schleife darauf ausgerichtet werden, nur die Bild-Bytes selbst (anstelle des gesamten Players) zu bewegen; eine solche Schleife würde leicht eine Ausführungszeit von 100 bis 200 Mikrosekunden erreichen. Der maßgebliche Punkt bei dieser Art der Animation ist, daß senkrechte Bewegung der Player sehr viel einfacher ist als mit Playerfield-Objekten.

Einfacher noch als die senkrechte Bewegung eines Players ist dessen waagerechte. Es gibt für jeden Player ein Register, daß als "Horizontal Position"-Register bezeichnet wird. Der Wert in diesem Register legt die waagerechte Position des Players auf dem Bildschirm fest. Um einen Player an eine bestimmte Bildschirmposition zu bringen, muß lediglich der entsprechende Wert in dieses Register geschrieben werden.

Waagerechte und senkrechte Bewegung eines Players sind voneinander unabhängig; sie können beliebig miteinander kombiniert werden.

Die Einheit für das Horizontal-Position-Register ist das Color Clock. Wird der Wert des Registers um 1 erhöht, dann bewegt sich der Player um ein Color Clock nach rechts. Es gibt 228 Color Clocks in einer einzelnen Scan Line, von denen aber aus Gründen des Overscans nicht alle sichtbar sind. Abhängig vom Overscan des jeweiligen Fernsehgerätes befinden sich die Positionen 0 bis 44 außerhalb des linken, und die Positionen 220 bis 255 außerhalb des rechten Bildschirmrandes. Die sichtbaren Player-Positionen liegen zwischen den Stellen 44 bis 220. Dieses kann manchmal Probleme aufwerfen, eröffnet aber eine praktische Möglichkeit: um den Player vom Bildschirm verschwinden zu lassen, muß seine waagerechte Position auf 0 gesetzt werden. Durch einfaches Laden und Speichern (mit POKE in BASIC) wird der Player unsichtbar.

Das soweit beschriebene System macht Animation mit hoher Geschwindigkeit möglich. Es sind allerdings noch einige zusätzliche Anmerkungen zu machen. Als erstes gibt es vier voneinander unabhängige Player. Jeder dieser Player besitzt einen eigenen RAM-Bereich; d.h. ihre Operationen sind voneinander völlig unabhängig. Die Player besitzten die Label (Bezeichnungen) P0 bis P3. Sie können nebeneinander benutzt werden, um eine Auflösung bis zu 32 Bits (in der Breite) zu liefern, oder um einzelne Objekte darzustellen.

Jeder Player besitzt weiterhin ein eigenes Farbregister, das unabhängig von allen anderen (Playfield- oder Player-Farbregister) die Farbe des Players bestimmt. Dieses Farbregister wird mit COLP(X) bezeichnet und besitzt eine Schattenadresse mit dem Namen PCOLR(X). Dieses gibt dem Benutzer die Möglichkeit, mehr Farbe auf den Bildschirm zu bringen. Jeder Player besitzt allerdings, da für jeden nur ein Register vorhanden ist, eine Farbe; mehrfarbige Player sind nur mit Display Liest Interrupts möglich (siehe Kapitel 5).

Jedem Player wird außerdem ein Register zur Festlegung seiner Breite zugeordnet. Er kann auf einfache, doppelte und vierfache Breite gesetzt werden. Dieses geschieht mit dem SIZEP(X)-Register.

Schließlich kann die senkrechte Auflösung vom Benutzer gesteuert werden. Es gibt eine Auswahl zwischen Einzel-Zeilen- und Doppel-Zeilen-Auflösung. Bei ersterer legt ein Byte des Speicherbereiches das Aussehen einer Scan Line fest; bei Doppel-Zeilen-Auflösung das Aussehen von zwei Scan Lines. Bei Einzel-Zeilen-Auflösung ist der Speicherbereich für einen Player 256 Bytes lang, bei Doppel-Zeilen-Auflösung 128 Bytes lang.

Dieses ist auch der einzige Fall, bei dem die einzelnen Player voneinander abhängig sind; die senkrechte Auslösung gilt für alle Player. Sie wird durch Bit D4 des DMACTL-Registers bestimmt. Bei Doppel-Zeilen-Auflösung befinden sich die ersten

10 Bytes des Bereiches außerhalb des oberen und die letzten 20 Bytes außerhalb des unteren Bildschirmrandes. Bei Einzel-Zeilen-Auflösung sind es entsprechend 20 und 40 Bytes, die verlorengehen.

Die nächste Erweiterung liegt in den sogenannten Missiles. Dieses sind 2-Bit breite Graphik-Objekte, ähnlich den Playern. Jeweils eine Missile wird einem Player zugeordnet; sie erhält ihre Farbe aus dem entsprechenden Farbregister des Players. Das Aussehen einer Missile wird ebenfalls in einem bestimmten RAM-Bereich festgelegt. Dieser liegt genau vor dem RAM-Bereich der Player. Alle Missiles befinden sich innerhalb des gleichen Bereichs (4 Missiles mit jeweils 2 Bits sind zusammen 8 Bits). Missiles können unabhängig von den Playern bewegt werden; sie besitzen ihre eigenen Horizontal-Position-Register. Außerdem besitzen Missiles ein eigenes Register für die Größe (SIZEM), das genauso wie das SIZEP(X)-Register für die Player funktioniert. Dieses Register gilt allerdings für alle Missiles, d.h. sie werden immer auf die gleiche Breite gesetzt.

Missiles können als Kugeln oder dünne senkrechte Linien auf dem Bildschirm benutzt werden. Wenn erforderlich, können die Missiles zu einem fünften Player zusammengesetzt werden, wobei dessen Farbe dann durch das Farbregister für Playfield 3 bestimmt wird. Dieses wird durch Setzen des Bits D4 des Prioritäten-Kontroll-Registers (PRIOR) erreicht. Die Missiles können natürlich auch in dieser Option unabhängig voneinander bewegt werden, da ihre Positionen von den zugehörigen Horizontal-Position-Registern gesteuert werden. Das „Einschalt-Bit” für den 5. Player beeinflußt nur die Farbe der Missiles.

Eine senkrechte Bewegung der Missiles wird wie bei den Playern erreicht: durch Schieben der Bilddaten durch den zugehörigen RAM-Bereich. Hierbei können sich allerdings Probleme ergeben, da die Missiles, wie schon angesprochen, in ein- und demselben Bereich stehen. Um also eine Missile senkrecht zu bewegen, müssen die Bits der anderen ausmaskiert werden.

Eine wichtige Fähigkeit der Player-Missile-Graphik ist die absolute Unabhängigkeit der Player und Missiles vom Playfield. Diese Objekte können mit jedem Graphik- oder Textmodus gemischt werden. Dieses wirft wiederum ein Problem auf: was geschieht, wenn ein Player oder eine Missile sich mit einem Playfield-Objekt überschneidet? Welches Bild hat Vorrang? Hierfür gibt es ein sogenanntes Prioritäten-Kontroll-Register, mit dem es dem Programmierer möglich ist, diese Vorrangigkeit zu definieren. Dieses Register wird mit PRIOR bezeichnet und besitzt eine Übertragungsadresse, genannt GPRIOR. Durch diese Auswahlmöglichkeit können interessante Effekte erzeugt werden indem ein Player vor einem und hinter einem anderen Playfield-Objekt vorbeiläuft.

Die letzte Erweiterung liegt in der Lieferung der sogenannten Kollisions-Abfrage (Collision Detection = Kollisions-Erfassung), die über die Hardware läuft. Es ist hiermit möglich zu prüfen, ob ein Objekt (Player oder Missile) mit irgendeinem anderen (Player, Missile oder Playfield) zusammengestoßen ist. Genaür gesagt, können folgende Kollisionen erfaßt werden: Missile - Player, Missile - Playfield, Player - Playfield, Player - Player sowie Player - Missile. Es gibt 54 Möglichkeiten solcher Überschneidungen, denen jeweils ein Bit zugeordnet wird. Dieses Bit enthält eine 1, wenn eine zugehörige Kollision auftrat. Die Bits befinden sich in 15 CTIA-Registern (wobei jeweils nur die unteren Nybbles benutzt werden). Diese Register sind Nur-Lese-Register, d.h. sie können nicht durch Nullschreiben gelöscht werden. Letzteres geschieht, indem irgendein beliebiger Wert in das HITCLR-Register geschrieben wird, wodurch dann alle Kollisions-Register gelöscht werden.

Hardwaremäßig wird eine Kollision durch die Überschneidung von zwei Bildern, z.B. Player und Playfield, verursacht; das bedeutet, das Kollisionsbit wird erst gesetzt, nachdem der Teil des Bildschirmes, auf dem sich die Überlappung ereignet, gezeichnet wurde. Eine Kollision wird daher frühestens 16 Millisekunden nachdem der Player bewegt wurde, registriert. Die beste Lösung ist, eine Abfrage der Kollisionsregister während der Vertical-Blank-Interrupt-Routine (siehe Anhang I) durchzuführen. In diesem Fall sollte als erstes die Kollisions-Abfrage durchgeführt, dann die Register gelöscht und schließlich der Player bewegt werden. Es kann allerdings auch 16 Millisekunden gewartet werden, bevor die Kollisions-Register abgefragt werden.

Es gibt eine Anzahl von Schritten, die für die Benutzung der Player-Missile-Graphiken erforderlich sind. Als erstes muß ein RAM-Bereich für die Player und Missiles vorbereitet werden. Danach wird dem Computer mitgeteilt, wo dieser Bereich sich befindet. Wenn Einzel-Zeilen-Auflösung benutzt wird, ist dieser RAM-Bereich 1280 Bytes lang; bei Doppel-Zeilen-Auflösung ist er 640 Bytes lang. Ein guter Platz für diesen ist das RAM direkt vor dem Display-Bereich am Anfang des Speicherbereiches. Das Layout des Player-Missile-Bereiches wird in Abbildung 4.2 dargestellt.

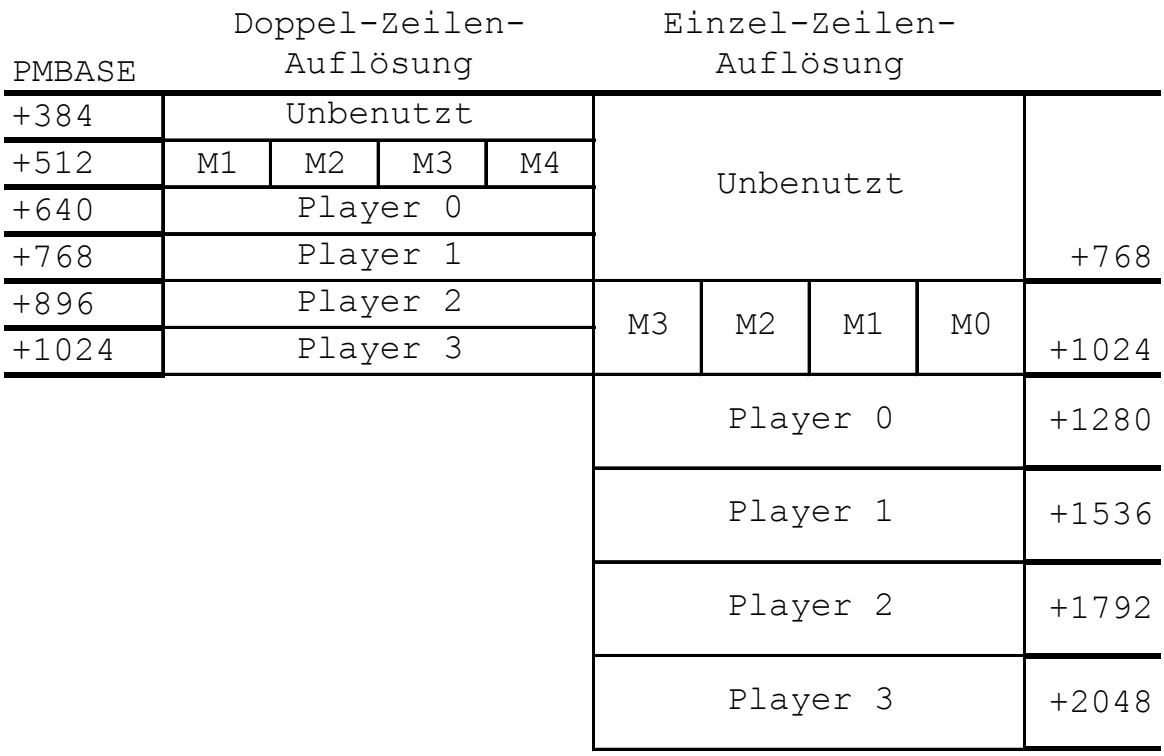


Abbildung 4.2
Layout des Player-Missile Bereiches im RAM

Der Zeiger, der auf den Anfang des Player-Missile Bereiches zeigt, wird mit PMBASE bezeichnet. Aufgrund der internen Beschränkungen ANTICs muß PMBASE bei Einzel-Zeilen-Auflösung an einer 1K-Grenze und bei Doppel-Zeilen-Auflösung an einer 2K-Grenze liegen. Werden nicht alle Player benutzt, ist es vorteilhaft, den restlichen RAM-Bereich wieder für andere Zwecke zu verwenden. Sobald festgelegt wurde, wo der RAM-Bereich für die Player und Missiles liegen soll, wird ANTIC informiert, indem die Page-Nummer von PMBASE in das PMBASE-Register ANTICs geschrieben wird.

Als nächstes werden die Parameter der Player, z.B. Farbe, horizontale Position und Breite, auf die gewünschten Werte gesetzt. Ebenso können die Prioritäten gesetzt werden. Schließlich muß ANTIC noch über die senkrechte Auflösung informiert werden. Für Einzel-Zeilen-Auflösung wird das Bit D4 im DMACTL-Register gesetzt. Bei Doppel-Zeilen-Auflösung wird dieses Bit gelöscht. Die Schattenadresse von DMACTL trägt die Bezeichnung SDMCTL. Als letztes wird die Player-Missile-Graphik durch Setzen des PM-DMA-Bits in DMACTL aktiviert. Bei diesem Setzen muß darauf geachtet werden, daß die anderen Bits dieses Registers nicht geändert werden. Das folgende Programm ist ein Beispiel für das Aufbauen und Bewegen eines Players.

```
1    PMBASE=54279:REM         Zeiger zum Anfang des
                              PM-Bereichs
2    RAMTOP=106:REM           Zeiger zum Ende des freien RAMs
3    SDMCTL=559:REM           Schattenadresse von DMACTL
4    GRACTL=53277:REM         Graphik-Kontroll-Register
                              im CTIA
5    HPOSP0=53248:REM         Horizontal-Position-Register PO
6    PCOLR0=704:REM           Farbregister für Player 0
10   GRAPHICS 0:SETCOLOR 2,0,0:REM Hintergrundfarbe-Schwarz
20   X=100:REM                waagerechte Playerposition
30   Y=48:REM                 senkrechte Playerposition
40   A=PEEK(RAMTOP)-8:REM     2K-Grenze unter dem RAM-Anfang
50   POKE PMBASE,A:REM        Position vom PM-Bereich an ANTIC
60   MYPMBASE=256*A:REM       Adresse des PM-RAMs „errechnen”
70   POKE SDMCTL,46:REM       Doppel-Zeilen-Auflösung
80   POKE GRACTL,3:REM        PM-Graphik aktivieren
90   POKE HPOSP0,100:REM      Waagerechte Position festlegen
100  FOR I=MYPMBASE+512 TO MYPMBASE+640:REM Bereich löschen
110  POKE I,0
120  NEXT I
130  FOR I=MYPMBASE+512+Y TO MYPMBASE+518+Y
140  READ A:REM               Bilddaten für Player lesen
150  POKE I,A
160  NEXT I
170  DATA 8,17,35,255,32,16,8
180  POKE PCOLR0,88:REM       Playerfarbe ist Pink
190  A=STICK(0):REM           Joystick abfragen
200  IF A=15 THEN GOTO 190:REM wenn inaktiv, nochmal versuchen
210  IF A=11 THEN X=X-1:POKE HPOSP0,X
220  IF A= 7 THEN X=X+1:POKE HPOSP0,X
230  IF A<>13 THEN GOTO 280
240  FOR I=8 TO 0 STEP -1
250  POKE MYPMBASE+512+Y+I,PEEK(MYPMBASE+511+Y+I)
260  NEXT I
270  Y=Y+1
280  IF A<>14 THEN GOTO 190
290  FOR I== TO 8
300  POKE MYPMBASE+511+Y+I,PEEK(MYPMBASE+512+Y+I)
310  NEXT I
320  Y=Y+1
330  GOTO 190
```

Sobald Player einmal angezeigt wurden, ist es aufgrund der einzelnen Schritte, die beim Anzeigen nötig sind, etwas schwierig, sie wieder vom Bildschirm zu löschen. Erstens holt sich ANTIC die Player-Missile-Daten aus dem RAM (sofern dieses durch das DMACTL-Register gestattet wird). Danach werden die Daten von ANTIC an den CTIA übergeben (abhängig von dem Wert in GRACTL). Der CTIA zeigt an, was sich in seinen Player-Missile-Registern befindet, egal was es ist (GRAFP0 bis GRAFP3, sowie GRAFM). Durch Ausschalten der entsprechenden Kontrollbits in den DMACTL- und GRACTL-Registern werden die Player-Missile-Objekte nicht gelöscht. Dieses verhindert nur, daß ANTIC neue Bilddaten für die Player-Missile-Graphiken an

den CTIA weitergibt; die alten Daten in den GRAF(X)-Registern werden weiterhin angezeigt. Um die Player-Missile-Graphiken zu löschen, müßen die Daten in den GRAFP(X)-Registern gelöscht werden, nachdem die Kontrollbits in DMACTL und GRACTL auf Null gesetzt worden sind. Eine einfachere Lösung besteht darin, die waagerechten Positionen der zu löschenden Objekte auf 0 zu setzen. ANTIC wird hierdurch allerdings nicht daran gehindert, weiter Daten an den CTIA zu übertragen, was pro Sekunde ungefähr 10.000 Maschinenzyklen verbraucht.

Player-Missile-Graphiken eröffnen eine Anzahl spezieller Möglichkeiten. Sie sind z.B. für Animation von großem Wert. Es gibt allerdings auch Einschränkungen. So sind nur vier Player vorhanden, wobei jeder Player eine Breite von nur 8 Bits, d.h. Pixel hat. Werden für die waagerechte Auflösung mehr Bits benötigt, so muß au+ normale Playfield-Animation zurückgegriffen werden. Für schnelle und einfache Animation sind Player allerdings optimal.

Es ist auch möglich, ANTIC zu „umgehen" und die Player-Missile-Bilddaten direkt in die PM-Graphik-Register des CTIA-Prozessors (GRAFP(X)) zu schreiben, wodurch dem Programmierer eine direktere Kontrolle über die Player-Missile-Graphiken gestattet wird. Hierbei wird aber dem Programmierer eine größere Verantwortung übertragen, da er die Bit-Ausgabe für die Bilddaten aufrechterhalten und im richtigen Augenblick in die Graphik-Register bringen muß. Hierbei muß der 6502-Prozessor herangezogen werden und im Bildschirmzyklus arbeiten (siehe „Kernel" in Kapitel 5). Dieses ist eine sehr umständliche Technik, die viel Konzentration beim Programmieren erfordert und wenig Vorteile in der Ausführung einbringt.

Player-Missile-Graphiken sind auch für andere Verwendungszwecke, als nur für Animation von Nutzen. Player sind eine gute Möglichkeit, die Farbenzahl eines Displays zu erhöhen. Die vier zusätzlichen Farbenregister liefern pro Zeile des Displays vier weitere Farben. Die Spannweite der Anwendungen von Playern in dieser Richtung wird natürlich durch ihre lediglich 8 Bit breite Auflösung eingeschränkt. Manchmal kann dies allerdings umgangen werden: als erstes wird ein Player auf vierfache Breite gesetzt. Danach werden die Prioritäten so gesetzt, daß er hinter einer Playfield-Farbe verschwindet. Als nächstes wird diese Playfield-Farbe mit der Hintergrund-Farbe vertauscht, so daß der scheinbare Hintergrund aus einer Playfield-Farbe besteht. Der Player verschwindet hinter diesem falschen Hintergrund. Nun wird in letzteren ein Loch geschnitten. Dieses geschieht, indem der richtige Hintergrund an die betreffende Stelle gemalt wird. Der Player ist vor diesem richtigen Hintergrund sichtbar, allerdings nur in dem Bereich, wo dieser gezeichnet wurde. Auf diese Weise besitzt ein Player eine größere Auflösung als 8 Bits. Das folgende Programm zeigt oben beschriebenen Trick:

```
1 RAMTOP=105
2 PMBASE=54279:REM PM Basis-Zeiger ANTICs
3 SDMCTL=559:REM DMACTL-Schattenadresse
4 GRACTL=53277:REM CTIA Graphik-Kontroll-Register
5 HPOSP0=53248:REM Horizontal Pos.-Reg. von P0
6 PCOLR0=704:REM Farbregister-Schattenadresse von P0
7 SIZEP0=53256:REM Breitenregister für Player 0
8 GPRIOR=623:REM Prioritäts-Kontroll-Register
10 GRAPHICS 7
20 SETCOLOR 4,8,4
30 SETCOLOR 2,0,0
40 COLOR 3
50 FOR Y=0 TO 79:REM Diese Schleife füllt den Schirm
60 PLOT 0,Y:REM mit dem „falschen" Hintergrund.
70 DRAWTO 159,Y
80 NEXT Y
90 A=PEEK(RAMTOP)-20:REM Muß wegen GRAPHICS 7 weiter „nach
100 POKE PMBASE,A:REM oben" gesetzt werden.
110 MYPMBASE=256*A
120 POKE SDMCTL,46
130 POKE GRACTL,3
140 POKE HPOSP0,100
150 FOR I=MYPMBASE+512 TO MYPMBASE+640
160 POKE I,255:REM Feste Farbe für den Player
170 NEXT I
190 POKE PCOLR0,88
190 POKE SIZEP0,3:REM Player auf Einfache Breite setzen
200 POKE GPRIOR,4:REM Prioritäten setzen
210 COLOR 4
220 FOR Y=30 TO 40:REM „Loch schneiden"
230 PLOT Y+22,Y
240 DRAWTO Y+43,Y
250 NEXT Y
```

Dieses Programm erzeugt folgendem Bild auf der Mattscheibe:

Abbildung 4.3
Maskieren eines Players zum
Erreichen einer höheren Auflösung

Eine andere Anwendungsmöglichkeit von Playern besteht in der Darstellung von speziellen Zeichen. Es gibt eine Anzahl von Zeichen, die die normalen vertikalen Begrenzungen eines Zeichensatzes überschreiten würden. Eine Möglichkeit für die Lösung dieses Problems wäre, spezielle Zeichensätze hierfür zu entwerfen. Eine andere Möglichkeit besteht in der Verwendung der Player. Unterstriche, Integralzeichen und andere spezielle Symbole können auf diese Weise dargestellt werden. Das

folgende Programm ist ein Beispiel für eine solche Playerverwendung:

```
1 RAMTOP=106
2 PMBASE=54279
3 SDMCTL=559
4 GRACTL=53277
5 HPOSP0=53248
6 PCOLR0=704
10 GRAPHICS 0:A=PEEK(RAMTOP)-16:REM Muß wegen Einzel-Zeilen 20
POKE PMBASE,A:REM              Auflösung weiter „nach
30 MYPMBASE=256*A:REM               oben” gesetzt werden.
40 POKE SDMCTL,62
50 POKE GRACTL,3
70 FOR I=MYPMBASE+1024 TO MYPMBASE+1280
60 POKE I,0
90 NEXT I
100 POKE PCOLR0,140
110 FOR I=0 TO 15
120 READ X
130 POKE MYPMBASE+1100+I,X
140 NEXT I
150 DATA 14,29,24,24,24,24,24,24
160 DATA 24,24,24,24,24,24,184,112
170 ? " ":REM                       Bildschirm löschen
180 POSITION 15,6
190 ? „xdx”
```

Dieses Programm erzeugt folgendem Bild:

Abbildung 4.4
Benutzung eines Players als spezielles Zeichen

Eine teilweise nützliche Verwendung von Playern wäre, sie als Cursor zu benutzen. Mit ihrer Fähigkeit, sich fließend an jede Bildschirmposition zu bewegen, sind sie ideal für solche Zwecke. Der Cursor kann außerdem seine Farbe ändern, während er sich über den Bildschirm bewegt, um anzuzeigen, was sich unter ihm befindet.

Player-Missile-Graphiken besitzen viele Anwendungsmöglichkeiten. Ihre Verwendung als bewegtes Objekt bei Telespielen ist offensichtlich. Sie besitzen aber mindestens ebensoviele "ernsthafte" Anwendungen. Sie können die Farbenvielfalt und Auflösung eines jeden Displays verbessern, ebenso als spezielle Zeichen verwendet, aber auch als Cursor benutzt werden. Man kann daher nur zu ihrem Gebrauch aufrufen.

Kapitel 5
Display-List-Interrupts

Der Display-List-Interrupt ist eine der leistungsstärksten Fähigkeiten des ATARI „TM" Personal Computer Systems, es ist allerdings auch eine der am schwersten zugänglichen. Er erfordert tiefste Kenntnis der Maschinensprache und der Maschinen-Charakteristiken. Display-List-Interrupts an sich liefern keine Verbesserungen; sie müßen in Verbindung mit den anderen Möglichkeiten des Systems, wie z.B. Player-Missile-Graphiken, indirekte Zeichensatz- oder Farb-register-Adressierung, benutzt werden. Mit Display-List-Interrupts kann die volle Leistungsstärke dieser Möglichkeiten erreicht werden.

Display-List-Interrrupts ziehen den Vorteil aus der sequentiellen Anordnung des Raster-Scan-Fernsehsytems. Der Fernseher zeichnet das Bild in einer bestimmten Zeitfolge vom oberen zum unteren Bildschirmrand. Dieser Zeichenvorgang verbraucht ungefähr 13.000 Mikrosekunden, was dem menschlichen Auge als ein gleichmäßiges Bild erscheint, für den Computer aber ein großer Zeitraum ist. Das Gerät hat sehr viel Zeit, um die Parameter des Displays zu ändern, während der Bildschirm gezeichnet wird. Das bedeutet, der Zyklus des Parameter-Änderns muß zum Zyklus des Bildschirmzeichnens synchronisiert werden. Dieses kann zum einen geschehen, indem der 6502 in eine Schleife geschickt wird, deren Ausführungsfrequenz exakt bei 50 Hertz liegt. Dieses macht andere Berechnungen allerdings nahezu unmöglich. Eine andere (und bessere) Methode ist, zu unterbrechen, d.h. anzuhalten <->Interrupt). Der 6502 reagiert auf den den Interrupt, indem er die Bildschirmparameter ändert und danach zu seinen normalen Aufgaben zurückkehrt. Der Interrupt hierfür muß zeitlich präzise abgestimmt werden, damit er exakt beim Bildschirm-Zeichen-Prozeß auftritt. Dieser speziell getimte Interrupt wird von ANTIC ausgelöst und als Display-List-Interrupt (DLI) bezeichnet.

Das Timing und die Ausführung eines jeden Interrupt-Prozesses ist kompliziert; daher soll hier erst die Folge von Ereignissen, die bei einem sauber arbeitenden Display-List-Interrupt auftritt, geschildert werden. Der Vorgang beginnt damit, daß der ANTIC-Chip auf eine Instruktion in der Display-List stößt, deren Interrupt-Bit (D7) gesetzt ist. ANTIC wartet dann bis zur letzten Scan-Line der im Augenblick angezeigten Mode-Line und „sieht" dann im NMIEN-Register „nach", ob Display-List-Interrupts eingeschaltet, d.h. zugelassen sind. Ist das entsprechende Bit ausgeschaltet, dann ignoriert ANTIC den Interrupt und fährt mit seiner normalen Anzeige-Arbeit fort. Andernfalls schickt

ANTIC ein NMI-Signal (NMI = Non Maskable Interrupt = Nicht maskierbares Interrupt) an den 6502 und kehrt dann erst zu seiner normalen Aufgabe zurück. Der 6502 springt über den NMI-Vektor zur Interrupt-Routine des OS. Diese Routine stellt als erstes den Grund für den Interrupt fest. Wenn letzterer tatsächlich ein Display-List-Interrupt ist, wird die angesprungene Routine über die Adressen $0200 und $0201 (nieder-, danach höherwertig) zu einer DLI-Service-Routine umgeleitet. Diese DLI-Routine ändert ein oder mehrere Graphik-Register, die die Anzeige kontrollieren. Danach führt der 6502 sein Hauptprogramm weiter.

Es gibt eine Anzahl von Schritten, die für das Erstellen eines DLIs erforderlich sind. Als erstes muß natürlich die DLI-Routine selbst geschrieben werden. Diese Routine muß alle 6502-Register, die evt. geändert werden, auf den Stapel schieben, da die OS-Routine keine Register rettet (Das Status-Register des Prozessors wird automatisch auf den Stapel gebracht). Die Routine sollte kurz und schnell sein; sie sollte nur Register ändern, die in Zusammenhang mit dem Display stehen. Sie muß außerdem damit enden, daß alle auf den Stapel gebrachten Register wieder zurückgespeichert werden.

Als nächstes muß die DLI-Service-Routine irgendwo im Speicher plaziert werden. Page 6 ist hierfür ideal. Danach wird der Vektor bei $0200 und $0201 so gesetzt, daß er auf diese Routine zeigt. Nun wird der vertikale Punkt auf dem Schirm festgelegt, an dem der DLI auftreten soll, dann die zugehörige Anweisung in der Display-List gesetzt. Dort wird Bit D7 der vorangegangenen Instruktion gesetzt. Schließlich wird der DLI eingeschaltet, indem Bit D7 des NMIEN-Registers ($D40E) gesetzt wird. Der DLI ist dann sofort aktiviert.

Wie bei jeder Interrupt-Service-Routine können durch zeitliche Erfordernisse Probleme aufgeworfen werden. ANTIC leitet den Interrupt nicht sofort an den 6502 weiter, sobald er auf eine entsprechende Anweisung stößt, sondern wartet bis zur letzten Scan-Line der unterbrechenden Mode-Line. Der 6502 und die Interrupt-Service-Routine des OS verbrauchen zusammen zwischen 18 und 25 Maschinenzyklen. Die erste Instruktion der DLI-Service-Routine wird also erst erreicht, nachdem mindestens 18 Maschinenzyklen vergangen sind. Diese 18 Maschinenzyklen entsprechen 36 Color Clocks auf dem Bildschirm. Das bedeutet, daß sich der Elektronenstrahl in der Mitte des Schirms (in der letzten Scan-Line der unterbrechenden Mode-Line) befindet, wenn die DLI-Routine gestartet wird. Ändert die DLI-Service-Routine ein Farbregister, dann erscheint die alte Farbe auf der linken und die neue Farbe auf der rechten Bildschirmseite. Aufgrund des unsicheren Timings beim Reagieren dem 6502 auf einen Interrupt, ist diese Grenzlinie zwar scharf, zittert aber hin und her.

Es gibt allerdings eine Lösung für dieses Problem. Sie besteht im WSYNC- (Warten auf waagerechte Synchronisation) Register. Immer wenn dieses Register auf irgendeine Weise adressiert wird, setzt ANTIC das RDY-Signal des 6502 Low und „friert" damit den 6502 solange ein, bis das Register durch eine waagerechte Synchronisation wieder zurückgesetzt wird. Wird in der Interrupt-Routine eine STA WSYNC-Anweisung genau vor der Anweisung, die den Wert eines Farbregisters ändert, plaziert, so wird die entsprechende Farbe geändert, während sich der Elektronenstrahl außerhalb des linken Bildschirmrandes befindet. Dadurch wechselt die entsprechende Farbe erst eine Scan-Line tiefer, als ursprünglich beabsichtigt, ist dafür aber "sauber".

Die richtige Benutzung eines DLIs ist deshalb, das DLI-Bit der Mode-Line vor der Mode-Line zu setzen, in welcher der DLI auftreten soll. Die DLI-Service-Routine muß also als erstes die 6502-Register auf den Stapel schieben. Danach werden diese mit den zu benutzenden neuen Graphik-Werten geladen. Die Routine muß eine STA WSYNC-Anweisung ausführen und dann die neuen Werte in die entsprechenden ANTIC- bzw. CTIA-Register speichern. Schließlich müßen die 6502-Register wieder vom Stapel geholt werden, woraufhin zum Hauptprogramm zurückgesprungen wird. Dieser Vorgang garantiert, daß die Graphik-Register am Anfang der gewünschten Zeile geändert werden, während der Elektronenstrahl sich außerhalb des Bildschirmrandes befindet.

Das folgende Programm ist ein Beispiel für einen DLI:

```
10 DLIST=PEEK(560)+256*PEEK(561):REM Display-List finden
20 POKE DLIST+15,130:REM Einfügen der Interrupt-Anweisung
30 FOR I=O TO 19:REM Schleife zum Einpoken der DLI-
40 READ A:POKE 1536+I,A:NEXT I:REM Service-Routine
50 DATA 72,138,72,169,80,162,88
60 DATA 141,10,212,141,23,208
70 DATA 142,24,208,104,170,104,64
90 POKE 512,0:POKE 513,6:REM Setzen des Interrupt-Vektors
90 POKE 54286,192:REM DLI einschalten
```

Dieses Programm benutzt die folgende DLI-Service-Routine in Assembler:

```
PHA           Akkumulator sichern
TXA
PHA           X-Register sichern
LDA #$50      Dunkle Farbe für Zeichen
LDX #$58      Purpur
STA WSYNC     Warten
STA COLPF1    Farbe speichern
STX COLPF2    Farbe speichern
PLA
TAX           Zurückspeichern der Register
```

```
        PLA
        RTI             Abschluß der Routine
```

Dieses ist eine sehr einfache DLI-Routine. Sie ändert die Hintergrundfarbe von Blau nach Purpur, sowie die Farbe der Buchstaben, so daß diese dunkler vor dem hellen Hintergrund angezeigt werden. Es ist vielleicht verwunderlich, daß die obere Hälfte des Bildschirmes weiterhin die blaue Farbe behält, obwohl die DLI-Rautine andauernd Purpur in das Farbregister schiebt. Die Antwort hierauf liegt in der Vertical-Blank-Routine des OS, die während des Vertical-Blanks immer wieder den blauen Farbwert in das Farbregister schreibt. Das Blau wird von der Schattenadresse dieses Farbregisters geliefert. Jedes Hardware-Farbregister besitzt eine Schattenadresse an einer festgelegten RAM-Speicherstelle. Diese Adressen liegen bei dezimal 708 bis 712. In den meisten Fällen werden die Farben geändert, indem die neuen Werte in diese Register geschrieben werden. Wenn direkt in die Hardware-Register geschrieben wird, „wischt" das OS die neuen Werte durch den Übertragungsprozeß innerhalb einer 50stel Sekunde wieder aus dem Register. Bei DLIs muß der neue Farbwert allerdings direkt in die Hardware-Register geladen werden. Ein DLI kann nicht, verwendet werden, um die Farbe der ersten Scan-Line zu ändern; das OS übernimmt deren Farbgebung. DLIs sind also nur für die nachfolgenden Zeilen anwendbar.

Durch das direkte Speichern von Farbwerten in die Hardware-Register wird ein neues Problem hervorgerufen: der „Attract-Modus" des Gerätes wird außer Kraft gesetzt. Dieser Modus ist eine Einrichtung, die durch das OS gesteuert wird. Wird neun Minuten lang keine Taste des Computers gedrückt, so beginnt das System die Bildschirmfarben durch zufällige Werte auf geringeren Helligkeitsstufen laufen zu lassen. Dieses garantiert, daß der mehrere Stunden nicht bediente Computer kein Bild in die Mattscheibe brennt. Es ist sehr einfach, einen solchen Modus in einen DLI einzufügen. Es müßen lediglich zwei neue Zeilen in das Assemblerprogramm eingeschoben werden:

```
Alt             Neu

LDA NEWCOL      LDA NEWCOL
STA WSYNC       EOR COLRSH
STA COLPF2      AND DRKMSK
                STA WSYNC
                STA COLPF2
```

DRKMSK und COLRSH sind Zero-Page-Adressen ($4E und $4F), die während des Vertical-Blank-Interrupts vom OS auf den neuesten Stand gebracht werden. Wird der Attract-Modus außer Kraft gesetzt, nehmen die Register COLRSH und DRKMSK die Werte $00 bzw. $FF an. Ist der Attract-Modus aktiviert, dann erhält COLRSH alle 4 Sekunden einen neuen Zufallswert, wogegen DRKMSK

den Wert $F6 enthält. Das bedeutet, COLRSH bestimmt die Farbe und DRKMSK „maskiert" das höchste Helligkeits-Bit „aus".

Die Einfügung des Attract-Modus in einen DLI ist allerdings ein Problem: die Ausführungszeit eines DLIs wird erhöht. Dieses Problem wird vielleicht durch die Beschreibung des DLI-Timings deutlicher. Die Ausführung eines DLIs läßt sich in drei Phasen auf teilen: Phase Eins reicht vom Beginn des DLIs bis zur STA WSYNC-Anweisung. Während dieses Zeitraumes zeichnet der Elektronenstrahl die letzte Scan-Line der unterbrechenden Mode-Line. Phase Zwei ist die Periode vom STA WSYNC-Befehl bis zum Auftauchen des Strahls auf dem Bildschirm. Diese Phase gehört zum Horizontal-Blank; alle Graphik-Änderungen sollten während dieser Periode ausgeführt werden. Phase Drei umfaßt den Zeitraum vom Auftauchen des Elektronenstrahls auf dem Schirm bis zum Ende der DLI-Service-Routine. Das Timing dieser Phase ist nicht als kritisch zu bezeichnen.

Eine Horizontal-Scan-Line benötigt 114 Taktzyklen. Ein DLI erreicht den 6502 ungefähr beim 15. Zyklus. Der 6502 benötigt nun weitere 7 Zyklen, um auf den Interrupt zu reagieren. Die OS-Routine zum Bearbeiten des Interrupts, sowie deren Sprung zur DLI-Service-Routine verbraucht 11 Maschinenzyklen. Dieses bedeutet, die Service-Routine wird nicht vor Verstreichen von 33 Taktzyklen erreicht. Außerdem muß mit der STA WSYNC-Anweisung beim 103. Zyklus begonnen werden; dieses reduziert die in Phase Eins verfügbare Zeit um weitere 11 Zyklen. Schließlich gehen dem 6502 noch weitere Zyklen durch den DMA vom ANTIC-Prozessor verloren. Dieses läßt maximal 61 Zyklen für die erste Phase zu; dieses Maximum wird allerdings nur mit Blank-Line-Mode-Lines erreicht. Zeichen- und Map-Modus-Befehle resultieren in dem Verlust eines Zyklus´ für jedes Datenbyte der Anzeige. Im ungünstigsten Fall sind dieses die BASIC-Modi 0,7 oder 8, die jeweils 40 Bytes pro Zeile verbrauchen. In solchen Modi sind lediglich 21 Maschinenzyklen für Phase Eins vorhanden. Das bedeutet, die Routine, die in Phase Eins abgearbeitet werden soll, besitzt eine Ausführungszeit von 21 bis 61 Maschinenzyklen.

Phase Zwei erstreckt sich über 24 Taktzyklen und ist die kritische Periode eines DLIs. Ebenso wie bei Phase Eins gehen einige Zyklen durch den DMA ANTICs verloren. Player-Missile-Graphiken verbrauchen 5 Zyklen, sofern sie benutzt werden. Die Anzeige-Anweisung kostet einen Zyklus; ist es eine LMS-Anweisung, so gehen zwei weitere Zyklen verloren. Schließlich werden noch ein oder zwei Taktzyklen durch die Speicherauffrischung oder durch das Einholen von Daten verbraucht. Das heißt für Phase Zwei sind 14 bis 23 Maschinenzyklen verfügbar.

Das Timing-Problem bei DLIs wird nun offensichtlich. Um eine einzelne Farbe laden, ändern und speichern zu können, werden 14 Zyklen benötigt. Das Sichern des A-, X- und Y-Registers auf den Stapel, sowie das Laden der 3 Farben in diese Register und deren abschließendes Abspeichern verbraucht 47 Zyklen, was den größten, wenn nicht sogar ganzen Teil von Phase Eins ausmacht. Der Programmierer, der DLIs für ausgedehnte Graphik-Änderungen benutzt, muß das Timing derselben genau beachten. Glücklicherweise muß der angehende Programmierer sich nicht den Kopf über Timing-Berechnungen zerbrechen, da das Zählen der Zyklen und eine Geschwindigkeits-Optimierung bei einfachen Graphik-Operationen überflüssig ist. Diese Berücksichtigungen sind nur in Situationen erforderlich, bei denen eine hohe Leistung gewünscht wird.

Es gibt keine Vereinfachungen für den Programmierer, der mehr als drei Farbregister in einem einzigen DLI ändern will. Es mag möglich sein, eine vierte Farbe während der dritten Phase zu ändern, sofern diese nicht am linken Bildschirmrand angezeigt wird. Entsprechend kann eine Farbe, die nicht am rechten Rand erscheint, während der ersten Phase nicht geändert werden. Eine andere Möglichkeit wäre, den überaktiven DLI in zwei weniger beschäftigte DLIs aufzuspalten, die jeweils die Hälfte der Arbeit des ursprünglichen ausführen. Der zweite DLI könnte durch Einfügen einer einzigen Scan-Line-Blank-Line-Anweisung (deren DLI-Bit gesetzt ist), plaziert werden, die direkt unter die unterbrechende Mode-Line in die Display-List eingefügt wird. Hierdurch wird natürlich ein kleiner Bereich des Bildschirmes verbraucht.

Eine andere Teillösung wäre, die Berechnungen für den Attract-Modus während des Vertical-Blanks durchzuführen. Hierfür müßten zwei Farbtafeln im RAM gespeichert werden: die erste Tafel enthält die Farbwerte, die durch die DLI-Routinen angezeigt werden sollen; die zweite beinhaltet die Attract-Werte der ersten. Während des Vertical-Blanks holt dann eine vom Benutzer geschriebene Routine die Farben aus der ersten Tafel, wandelt sie um und speichert diese Attract-Farben in der zweiten Tafel. Die DLI-Routine nimmt dann die Werte direkt aus der zweiten Tafel, ohne daß dabei Zeit für das Umwandeln der Attract-Farben verloren geht.

Vielfach wird der Einsatz von mehreren DLIs gewünscht, die an verschiedenen senkrechten Bildschirmpositionen auftreten. Dieses ist eine wichtige Möglichkeit, um mehr Farbe auf den Bildschirm zu bringen. Es gibt aber nur einen DLI-Vektor; sollen mehrere DLIs eingesetzt werden, dann muß die Vektorenänderung mit in die DLI-Routine eingebracht werden. Es gibt verschiedene Wege, um dieses zu tun. Falls die DLI-Routine jedesmal den gleichen Arbeitsgang nur mit anderen Farben ausführt, kann sie über eine Speichertafel gesteuert werden. Jedesmal, wenn die DLI-Routine angesteuert wird, wird ein Zähler inkrementiert und als Index für die zu benutzende

Tafel verwendet. Das folgende Programm ist ein, Beispiel für diese Idee:

```
          PHA
          TXA
          PHA
          INC COUNTR
          LDX COUNTR
          LDA COLTAB,X   Page $F0 wird als Farbtafel benutzt
          STA WSYNC      Warten
          STA COLBAK
          CPX #$4F       Letzte Zeile
          BNE ENDDLI     Nein - DLI beenden
          LDA #$00       Ja - Zähler zurücksetzen
          STA COUNTR
ENDDLI    PLA
          TAX
          PLA            Akkumulator zurückspeichern
          RTI
```

Das BASIC-Programm zum Aufrufen dieser Routine sieht wie folgt aus:

```
10   GRAPHICS 7
20   DLIST=PEEK(560)+256*PEEK(561):REM Display-List finden
30   FOR J=6 TO 84:REM In jede Mode-Line ein DLI einfügen
40   POKE DLIST+J,141:REM BASIC-Modus 7 mit gesetztem DLI-Bit
50   NEXT J
60   FOR J=0 TO 30
70   READ A:POKE 1536+J,A:NEXT J:REM Laden der Service-Routine
                                     in Page 6
80   DATA 72,138,72,238,32,6,175,32,6
90   DATA 189,0,240,141,10,212,141,26,208
100  DATA 224,79,208,5,169,0
110  DATA 141,32,6,104,170,104,64
120  POKE 512,0:POKE 513,6:REM Vektor zur Service-Routine
130  POKE 54286,192:REM DLI einschalten
```

Dieses Programm bringt 80 verschiedene Farben auf den Schirm.

Es gibt noch eine andere Möglichkeit, um mehr DLIs einzusetzen. Einerseits könnte ein DLI-Zähler zum Verzweigen zu den Service-Routinen benutzt werden. Dieses verlangsamt allerdings die „Reaktion" der DLI-Routine; besonders die der am Ende der Abfrage liegenden Routine. Eine andere Möglichkeit wäre, daß jede DLI-Routine die Adresse der DLI-Vektoren bei $200 und $201 schreibt. Es ist die beliebteste Methode beim Einsetzen mehrerer DLIs. Sie hat außerdem den Vorteil, daß die Vektor-Logik nach anstatt vor der kritischen Zeitphase ausgeführt wird.

Die Routine für das Klicken der Tastatur wird mit der Funktion eines DLIs zwischengeschaltet. Bei jedem Tastendruck ertönt aus dem internen Lautsprecher ein Klicken, sofern dieser Druck anerkannt wurde. Das Timing für dieses Geräusch wird durch mehrere STA WSYNC-Anweisungen geliefert. Hierdurch kann allerdings das Timing einer DLI-Routine gestört worden, die Bildschirmfarben rutschen für den Bruchteil einer Sekunde eine Scan Line nach unten. Es gibt allerdings keine einfache Lösung dieses Problems. Eine Möglichkeit bezieht sich auf das VCOUNT-Register. Es ist ein Nur-Lese-Register im ANTIC-Chip, das ANTIC mitteilt, welche Zeile dieser augenblicklich anzeigt. Eine DLI-Routine könnte dieses vorgenannte Register überprüfen und entscheiden, ob eine Farbe zu ändern ist.

Eine andere Lösung wäre das Abschalten der OS-Routine und das Einsetzen einer eigenen Routine, was allerdings eine komplizierte Aufgabe ist. (Die brutalste Lösung wäre die Verweigerung sämtlicher Tastatureingaben. Werden keine Tastendrucke anerkannt so kann auf dem Bildschirm auch kein Zittern auftreten.)

Der DLI wurde entworfen, um eine primitive Software/Hardware-Technik zu ersetzen, welche als „Kernel" bezeichnet wird. Ein Kernel ist eine Programmschleife des 6502-Prozessors, die präzise zum Anzeige-Zyklus des Fernsehgerätes synchronisiert ist. Durch Überwachung des VCOUNT-Registers und Konsultieren einer Tafel, in welcher Bildschirm-Änderungen als eine Funktion von VCOUNT-Werten katalogisiert sind, kann der 6502 eigenmächtig alle Graphikwerte des gesamten Schirmes kontrollieren.

Für diese Leistungsfähigkeit wird allerdings ein hoher Preis bezahlt: der 6502 kann während der Bildschirm-Anzeige Periode nicht mehr für Berechnungen benutzt werden, was einen Zeitraum von ungefähr 75% der gesamten Prozessorzeit ausmacht. Außerdem darf keine Berechnung größer als ca. 4000 Maschinenzyklen sein, die während des Vertical Blanks und der Overscanzeit zur Verfügung stehen. Das heißt, Kernel können nur in Programmen benutzt werden, die wenig Berechnungen erfordern, wie es bei einigen Telespielen der Fall ist. Das „BASKETBALL „TM""-Programm auf dem ATARI Computer benutzt z.B. die Kerneltechnik; das Spiel benötigt wenig Berechnungen, aber viel Farbe. Die mehrfarbigen Player dieses Programms könnten nicht mit DLIs erzeugt werden, da diese auf vertikale Bildschirm- anstatt auf Playerpositionen bezogen werden.

Es ist möglich, die Kernel-Idee soweit zu erweitern, daß sie nur auf einzelne Scan-Lines angewandt wird. Auf diese Weise kann ein einzelnes Farbregister in einer Scan-Line mehrere Farben erzeugen. Die waagerechte Position des Farbwechsels wird durch die Zeit bestimmt, die bis zur Farbänderung vergeht. Das heißt, durch sorgfältiges Abzählen von

Maschinenzyklen kann der Programmierer mehr Farbe auf den Bildschirm bringen.

Unglücklicherweise ist dieses in der Praxis sehr schwer durchführbar. Durch ANTICs DMA ist es schwierig zu bestimmen, wieviel Zyklen in Wirklichkeit verstreichen; ein einfaches Zählen der 6502-Zyklen reicht hierbei nicht aus. Wenn der DMA ANTICs abgeschaltet wird, kann der 6502 zwar die volle Kontrolle über die Anzeige übernehmen, muß aber zusätzlich alle Arbeit verrichten, die der ANTIC-Prozessor sonst durchführt. Aus diesem Grunde sind waagerechte Kernels sehr selten den Aufwand wert, den sie erfordern. Natürlich wäre diese Technik durchführbar, wenn zwei Bilder, die verschiedene Farben besitzen sollen, einen einigermaßen großen Abstand zueinander haben. Hierdurch muß das Timing nicht so genau abgestimmt sein.

Der ungeheure Wert der indirekten Graphik-Adressierung und aller Register wird nun offensichtlich. Mit Hilfe von Display-List-Interrupts können diese Register „im Vorbeigehen" geändert werden. Deshalb ist es möglich mehr Farbe, bessere Graphik oder spezielle Effekte auf den Bildschirm zu bringen. Die offensichtlichste Verbesserung ist die Erzeugung von vielfarbigen Displays. Jedes Farbregister kann so oft geändert werden, wie DLIs vorhanden sind. Das gilt natürlich sowohl für Playfield- als auch für Player-Missile-Farbregister, von denen Jedes 128 Farben speichern kann.

Natürlich können die hierdurch gegebenen Möglichkeiten nicht mit jedem Programm voll ausgenutzt werden. Zu viele DLIs beeinflussen die Programmgeschwindigkeit. In der Praxis bedeutet dieses: ein Dutzend Farben sind einfach einzubringen, zwei Dutzend Farben erfordern eine genaue Planung, und noch mehr Farben schon von vornherein sehr günstige Umstände.

DLIs können allerdings nicht nur benutzt werden, um mehr Farben auf den Bildschirm zu bringen, mit ihnen kann auch die Leistungsfähigkeit von Player-Missile-Graphiken gesteigert werden. Die waagerechte Position eines Players kann mit Hilfe eine DLIs geändert werden. Auf diese Weise kann ein Player auf dem Bildschirm etwas tiefer ein zweites, drittes oder viertes Mal erscheinen. Ein einzelner Player kann verschiedene Figuren auf dem Bildschirm darstellen. Wenn man sich einen Player als einen senkrechten Papierstrei+en vorstellt, auf dem sich Bilder befinden, dann wäre der DLI eine Schere, mit der dieser Streifen in mehrere Teile zerschnitten wird, die dann neu positioniert werden. Hierbei können natürlich nicht zwei Teile eines Players auf der gleichen Zeile stehen. Wird ein Display aufgebaut, daß Graphik-Objekte benötigt, die sich niemals auf den gleichen Zeilen befinden, kann hierfür ein einziger Player benutzt werden.

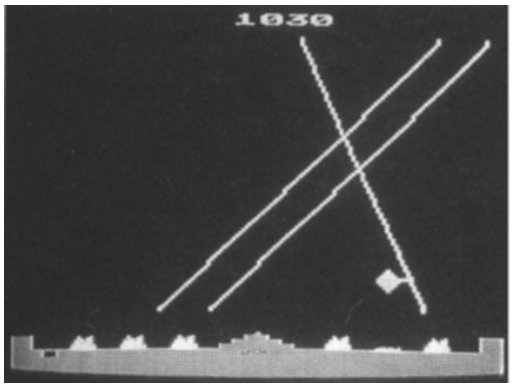
Eine andere Art DLIs in Verbindung mit Playern zu verwenden wäre, ihre Breite oder Priorität zu ändern. Dieses geschieht

meistens mit dem Maskierungs-Trick, der in Kapitel 4 beschrieben wurde.

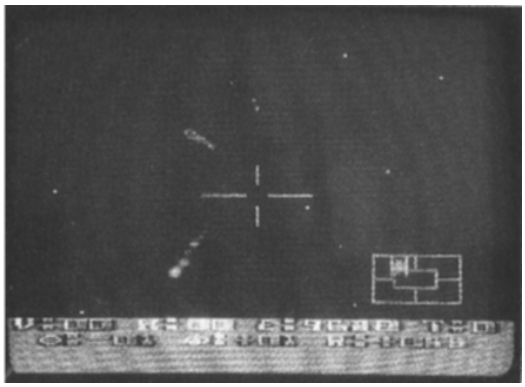
Die letzte Anwendung von DLIs betrift das Ändern von Zeichensätzen mit zunehmender Zahl der Bildschirmzeilen. Dieses ermöglicht einem Programm die Verwendung von Zeichengraphiken innerhalb eines großen und normalen Textes innerhalb eines kleinen Fensters. Ein Programm könnte also einen Graphik-Satz im oberen Bildschirmrand, einen Großschrift-Satz in der Bildschirmmitte und normalen Text am unteren Rand ausgeben. Außerdem könnte das Reflektions-Bit mit einer DLI-Routine geändert werden, was besonders für schon angesprochene Kartenspiele von Nutzen ist.

Die einwandfreie Benutzung eines DLIs erfordert sorgfältiges Anlegen des Displays, bei welcher der Programmierer die senkrechte Architektur der Bildschirmaufteilung in Erwägung ziehen muß. Das Raster-Scan-Fernseh-System ist nicht zweidimensional symetrisch; es besitzt aufgrund der Geschwindigkeitsverhältnisse beim Zeichnen des Schirmes eine mehr in senkrechte Richtung orientierte Struktur. Das waagerechte Zeichnen läuft ungefähr 200 mal schneller ab als das senkrechte. Das Display System des ATARI Computers wurde speziell für dieses Raster-Scan-System entwarfen, daher spiegelt sich dessen Struktur in der Technik dem Computers wieder. Das ATARI„TM”-Display ist kein flaches, leeres Blatt, auf welchem gezeichnet wird; es ist ein Stapel von dünnen Streifen, von denen jeder andere Werte annehmen kann. Der Programmierer, der auf einem speziellen Display besteht, schlägt viele Möglichkeiten aus. Optimale Resultate werden erzielt, wenn die anzuzeigende Information in einer streng senkrechten Struktur angelegt wird. Dieses erlaubt die Entfaltung der vollen Leistungsfähigkeit von DLIs.

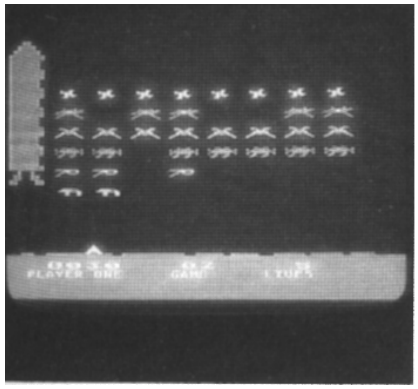
Abbildung 5.1 zeigt einige Bildschirme verschiedener Programme und gibt die relative Menge der verwendeten DLIs an.



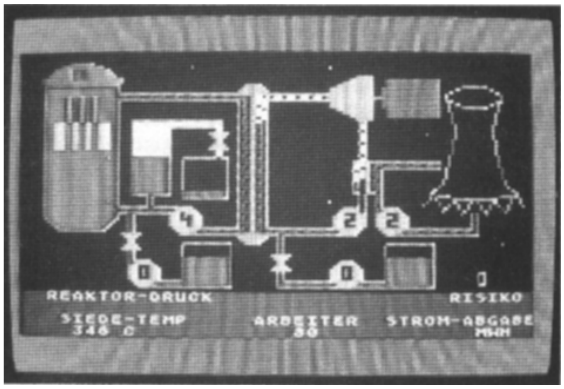
Missile Command
---einige---



Star Raiders
---wenige---



Space Invaders
---sehr viele---



Scram
---wenige---

Abbildung 5.1.
Beispiele senkrechter Bildschirmarchitektur

Kapitel 6
Scrolling

Sehr oft Übersteigt der Informationsbetrag, den ein Programmierer anzeigen möchte, dem, der auf den Bildschirm paßt. Eine Möglichkeit, um dieses Problem zu lösen, besteht im „Scrollen" (= Rollen) der Information über den Schirm. So scrollen BASIC-Programm Listings z.B. über die Mattscheibe. Alle Personal Computer besitzen diesen Typ des Scrollens. Zusätzlich besitzt der ATARI-Personal Computer aber noch zwei weitere Scroll-Möglichkeiten. Es sind grobes Scrolling (über die LMS-Anweisung) und feines Scrolling.

Andere Computer benutzen außchließlich grobes Scrolling. Bei dieser Art sind allerdings die Pixel, welche die Zeichen beinhalten, auf der Bildschirmposition fixiert. Der Text wird dabei gescrollt, indem die Bytes durch den Bildschirmspeicher bewegt werden. Die Auflösung bei dieser Methode beträgt ein Zeichen, das sehr grob ist, wie man sicherlich zugeben muß. Das erzeugte Scrolling ist springend und nicht sehr augenfreundlich. Außerdem kann es nur dadurch erreicht werden, daß jeweils bis zu tausend Bytes im RAM herumgeschoben werden. Dieses ist auch für einen Computer eine sehr mühselige Arbeit. Das System muß also die Daten bewegen, um ein Scrollen der Information zu erzeugen.

Einige Personal Computer produzieren ein etwas feineres Scrolling, indem die Bilder in einem Graphik-Modus mit höherer Auflösung gezeichnet und dann erst gescrollt werden. Obwohl hierbei eine bessere Auflösung beim Scrolling erreicht wird, müßen noch mehr Daten als beim groben Scrolling bewegt werden, wodurch das Programm natürlich sehr stark verlangsamt wird. Das grundsätzliche Problem ist also, daß Daten durch den Bildschirmbereich des Speichers bewegt werden müßen, damit ein Scrollen erreicht wird.

Auf dem ATARI Computer gibt es eine bessere Möglichkeit, um ein grobes Scrolling zu erreichen: der Bildschirmbereich wird über den anzuzeigenden Datenbereich bewegt, was mit Hilfe der LMS-Anweisung geschieht. Der Load-Memory-Scan-Befehl wurde zum ersten Mal in Kapitel 2 behandelt. Er teilt dem ANTIC-Prozessor mit, wo sich das Bildschirm-RAM befindet. Eine normale Display List besitzt nur eine LMS-Anweisung am Anfang; der angewählte RAM-Bereich enthält die anzuzeigenden Daten in linearer Reihenfolge. Durch Manipulation der Bytes des LMS-Operanden kann ein einfaches Scrolling erreicht werden; das Playfield-Fenster wird über die Daten bewegt. Das heißt durch Ändern von lediglich zwei Adress-Bytes kann ein Effekt erzielt werden, der dem Bewegen der gesamten Bildschirmdaten gleichkommt. Das folgende Programm realisiert die oben beschriebene Methode:

```
10  DLIST=PEEK(560)+256*PEEK(561):REM Display-List finden
20  LMSLOW=DLIST+4:REM Adresse des niederwertigen Operanden
30  LMSHIGH=DLIST+5:REM Adresse des höherwertigen Operanden
40  FOR I=0 TO 255:REM Äußere Schleife
50  POKE LMSHIGH,I
60  FOR J=0 TO 255:REM Innere Schleife
70  POKE LMSLOW,J
80  FOR Y=1 TO 50:NEXT Y:REM Verzögerungs-Schleife
90  NEXT J
100 NEXT I
```

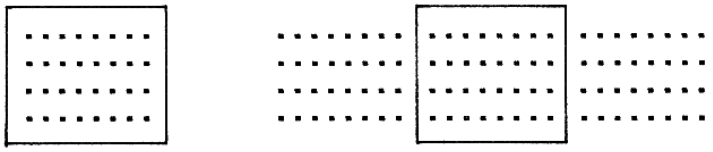
Dieses Programm bewegt das Display-List über den gesamten Adressbereich des Computers. Die Inhalte aller Speicherstellen werden auf dem Bildschirm ausgegeben. Das Scrolling ist ein einfaches serielles Scrolling, das durch Kombination von waagerechtem und senkrechtem Scrolling erreicht wird. Ein reines senkrechtes Scrolling kann erzielt werden, indem ein festgelegter Betrag (die Zeilenlänge in Bytes) zum LMS-Operanden addiert oder subtrahiert wird. Das folgende Programm tut dies:

```
10  GRAPHICS 0
20  DLIST=PEEK(560)+256*PEEK(561)
30  LMSLOW=DLIST+4
40  LMSHIGH=DLIST+5
50  SCREENLOW=0
60  SCREENHIGH=0
70  SCREENLOW=SCREENLOW+40:REM Nächste Zeile
80  IF SCREENLOW<256 THEN GOTO 120:REM übertrag?
90  SCREENLOW=SCREENLOW-256:REM Ja, Zeiger korrigieren
100 SCREENHIGH=SCREENHIGH+1
110 IF SCREENHIGH=256 THEN END
120 POKE LMSLOW,SCREENLOW
130 POKE LMSHIGH,SCREENHIGH
140 GOTO 70
```

Ein reines waagerechtes Scrolling ist nicht ganz so leicht zu erreichen wie ein reines senkrechtes. Das Problem hierbei ist, daß das Bildschirm-RAM für eine einfache Display-List sequentiell organisiert ist. Die Bildschirm-Bytes für die einzelnen Zeilen folgen direkt aufeinander, wobei das erste Byte einer Zeile genau dem letzten der vorangehenden folgt. Es könnte gescrollt werden, indem alle Bytes nach links oder rechts geschoben werden; dieses kann durch de- bzw. inkrementieren des LMS-Operanden geschehen. Bei dieser Methode wird aber das Byte am Rand einer Zeile eine Zeile höher bzw.

tiefer an den linken bzw. rechten Rand gebracht. Das erste Beispielprogramm zeigte dieses.

Die Lösung ist, den Bereich der Bildschirmdaten zu erweitern und in eine Folge von unabhängigen waagerechten Datenzeilen aufzuspalten. Abbildung 6.1 illustriert diese Idee schematisch:



Normale Datenanordnung Datenanordnung für waagerechtes Scrolling

Abbildung 6.1.
Anordnung der Daten im Bildschirmspeicher

Die in obiger Abbildung auf der linken Seite befindliche Anordnung stellt die normale Aufteilung der Bildschirmdaten dar. Das eindimensionale, serielle RAM wird zu einer linearen Folge gestapelt. Das rechte Bild zeigt die Datenanordnung für ein wagerechtes Scrolling. Das RAM ist hierbei natürlich immer noch eindimensional und seriell, wird aber jetzt anders benutzt. Das RAM für jede einzelne Zeile ist größer als eine Bildschirmbreite. Dieses ist kein Fehler; die grundliegende Absicht beim Scrolling ist ja, mehr Information anzuzeigen, als auf den Bildschirm paßt. Ist kein RAM-Bereich vorhanden, der diese zusätzliche Information fassen kann, wird sie auch nicht angezeigt. Mit dieser Datenanordnung kann ein echtes waagerechtes Scrolling erreicht werden. Das Bildschirmfenster kann über den gesamten Datenbereich bewegt werden, ohne daß ein senkrechtes Rollen wie bei den früheren Versuchen auftritt.

Der erste Schritt bei der Erzeugung waagerechten Scrollings ist die Festlegung der Länge einer Zeile, wobei der zugehörige RAM-Bereich entsprechend angelegt werden muß. Als nächstes muß eine neue Display-List geschrieben werden, die für jede Mode-Line eine LMS-Anweisung besitzt. Hierdurch wird die Display-List natürlich länger als gewöhnlich (wofür es allerdings auch keinen Hinderungsgrund gibt). Welche Werte sollen nun für die LMS-Operanden verwendet werden? Es könnten z.B. die Adressen der ersten Bytes jeder Datenzeile sein. Für jede Mode-Line, d.h. LMS-Operanden, gibt es dann eine solche Adresse.

Sobald die neue Display-List geschrieben und im Speicher plaziert wurde, muß der ANTIC-Prozessor auf sie umgeschaltet werden. Ebenso müßen die Bildschirmdaten in den entsprechenden RAM-Bereich geschrieben werden, damit der Schirm gefüllt wird.

Soll nun ein Scrolling durchgeführt werden, müßen die LMS-Operanden de- bzw. inkrementiert werden, wodurch ein Scrolling nach links bzw. nach rechts erfolgt. Das Programm für diese Aktion muß darauf achten, daß nicht über die Ränder des angelegten Bildschirm-RAMs hinaus gescrollt wird. Andernfalls erscheint ein falsches Bild auf der Mattscheibe. Der LMS-Operand muß beim Aufbau des Programms auf das linksseitigste Byte der angezeigten Zeile deuten. Das Maximum für den Wert dieses LMS-Operanden ist die Adresse des letzten Bytes der Speicherzeile minus der Anzahl der Bytes, die in einer Bildschirmzeile angezeigt werden.

Da waagerechtes Scrolling ein etwas komplizierter Prozeß ist, soll es hier an einem Beispiel verdeutlicht werden. Als erstes muß die Länge der Speicherzeilen bestimmt werden. Im Beispiel werden Zeilen mit einer Länge von 256 Bytes verwendet, weil dadurch die Berechnungen vereinfacht werden. Jede Zeile verbraucht hierdurch eine Page des Speichers. Im Beispiel wird der BASIC-Modus 2 verwendet, d.h. der Bildschirm besteht aus 12 Zeilen. Dieses wiederum bedeutet einen Speicherbereich von 12 Pages, was 3K-RAM entspricht. Der Einfachheit halber (und um zu garantieren, daß der Bildschirm nicht leer ist) werden die untersten 3K des Speichers benutzt. Dieser Bereich wird vom OS und vom DOS belegt und ist daher mit interessanten Daten gefüllt. Damit das Ganze noch interessanter wird, wird die Display-List auf Page 6 abgelegt. So erscheint sie mit auf dem Schirm, wenn gescrollt wird.

Die Anfangswerte der LMS-Operanden sind leicht zu berechnen; die niederwertigen Bytes haben alle den Wart 0, wogegen die höherwertigen Bytes die Werte 0, 1, 2, 3 usw. annehmen. Das folgende Programm führt alle oben genannten Aktionen aus und scrollt dann den Bildschirm horizontal.

```
10   REM  Als erstes wird die Display-List aufgebaut
20   POKE 1536,112:REM   Blank-8-Line
30   POKE 1537,112:REM   Blank-8-Line
40   POKE 1539,112:REM   Blamk-8-Line
50   FOR I=1 TO 12:REM   Schleife zum Einlesen der DL
60   POKE 1536+3*I,71:REM BASIC-Modus 2 mit gesetztem LMS
70   POKE 1536+3*I+1,0:REM Niederwertiges LMS-Operanden Byte
80   POKE 1536+3*I+2,I:REM Höherwertiges LMS-Oparanden Byte
90   NEXT I
100  POKE 1575,65:REM JVB-Anweisung für ANTIC
110  POKE 1576,0:REM DL fängt bei $600 an
120  POKE 1577,6
130  REM  ANTIC wird mitgeteilt, wo sich die DL befindet
140  POKE 560,0
150  POKE 561,6
160  REM  Nun wird waagerecht gescrollt
170  FOR I=1 TO 235:REM  Schleife für niederwertige LMS-Bytes
180  REM 235 anstatt 256, da eine Zeile aus 20 Zeichen besteht
190  FOR J=1 TO 12:REM Für jede Mode-Line
```

```
200  POKE 1536+3*J+1,I:REM Niederwertiges LMS-Byte
210  NEXT J                neuschreiben
220  NEXT I
230  GOTO 170:REM Endlos-Schleife
```

Dieses Programm scrollt die Daten von rechts nach links. Wird das Ende einer Page erreicht, beginnt einfach wieder alles von vorn. Die Display-List kann in der 6. Reihe von oben gefunden werden (Page 6). Sie erscheint als eine Folge von Anführungszeichen.

Der nächste Schritt wäre das Mischen von senkrechtem und waagerechtem Scrolling. Senkrechtes Scrolling wird durch Addition oder Subtraktion einer 1 am LMS-Operanden erzielt; senkrechtes entsprechend durch Addition bzw. Subtraktion der Zeilenlänge. Diagonales Scrolling kann demnach erreicht werden, wenn diese beiden gemischt werden, d.h. gleichzeitig ausgeführt werden.

Es gibt 4 mögliche Richtungen des diagonalen Scrollings. Beträgt die Zeilenlänge z.B. 256 Bytes und es soll nach rechts unten gescrollt werden, so muß ein Wert von 256+(-1)=255 zum LMS-Operanden addiert werden. Dieses ist eine 2-Byte Addition; das obere Programm umgeht die Schwierigkeiten bei der Manipulation von 2-Byte Adressen, was bei den meisten Programmen aber nicht möglich ist. Für schnelles Scrollen in 2 Richtungen ist ein Maschinensprachen-Programm erforderlich.

Alle Anordnungen sind möglich, wenn die LMS-Bytes unterschiedlich manipuliert werden. Die Zeilen können gleichmäßig scrollen oder sich gegenseitig überholen. Vieles könnte natürlich mit einem normalen Display gemacht werden, wobei allerdings mehr Daten zur Durchführung notwendig wären. Der wirkliche Vorteil des LMS-Scrollings ist die Geschwindigkeit. Anstatt einen gesamten Bildschirm mit Daten zu bewegen, werden vom Programm nur 2 bzw. etwas mehr Bytes manipuliert.

FEINES SCROLLING

Die zweite wichtige Möglichkeit auf dem ATARI 400/800 „TM" ist die des Fein-Scrollings. Feines Scrolling bedeutet, daß die Zeichen in Schritten, die der Größe eines Bildschirmpixels entsprechen, gescrollt werden. Beim groben Scrolling beträgt diese Größe jeweils nur ein Zeichen; das waagerechte Scrolling hat eine waagerechte Auflösung von einem Color Clock, sowie eine senkrechte von einer Scan-Line. Das feine Scrolling kann allerdings nicht über diese Werte hinaus kleiner werden. Soll feines Scrolling über eine längere Strecke durchgeführt

werden, so müßen grobes und feines Scrolling kombiniert werden.

Zum Durchführen des feinen Scrollings sind lediglich zwei Schritte erforderlich. Als erstes muß das entsprechende Bit der Mode-Line-Anweisung, die zu der zu scrollenden Zeile gehört, gesetzt werden (in den meisten Fällen gilt dieses für den gesamten Bildschirm, d.h. in sämtlichen Mode-Line-Anweisungen werden die „Fein-Scrolling"-Bits gesetzt). Bit D5 der DL-Anweisung ist das Einschalt-Bit für feines Scrolling in senkrechter Richtung, Bit D4 für das in waagerechter Richtung. Die Werte für die senkrechte und waagerechte Verschiebung werden in zwei Registern gespeichert. Das Register für feines Scrolling in waagerechter Richtung (HSCROL) liegt bei $D404, das für feines senkrechtes Scrolling (VSCROL) liegt bei $D405.

Für feines horizontales Scrolling wird die Anzahl der Color Clocks, um die gescrollt werden soll, in HSCROL gespeichert. Beim feinen senkrechten Scrolling wird die Anzahl der Scan-Lines, um die gescrollt werden soll, nach VSCROL gebracht. Die in diesen Registern gespeicherten Werte gelten dann für alle Mode-Lines, deren entsprechende Bits gesetzt sind.

Es gibt zwei Faktoren, durch die der Einsatz von feinem Scrolling verkompliziert wird. Beide liegen in der Tatsache begründet, daß ein gescrolltes Display mehr Information als ein nicht gescrolltes anzeigt. Was geschieht z.B., wenn eine Zeile waagerecht um ein halbes Zeichen nach links gescrollt wird? Die Hälfte des ersten Zeichens befindet sich außerhalb des linken Bildschirmrandes und das 40. Zeichen der Zeile wird ebenfalls nach links verschoben. Was nimmt nun diesen rechts freigewordenen ein? Das entsprechende Zeichen wäre das 41. Es gibt aber nur 40 Zeichen in einer Zeile; was geschieht also? Wird grobes Scrolling verwendet, dann erscheint das 41. Zeichen plötzlich auf dem Schirm, sobald das erste Zeichen am linken Rand aus dem Bild verschwunden ist. Dieses plötzliche Erscheinen irritiert das Auge sehr, aber es wurde bereits eine hardwaremäßige Lösung in das Gerät eingebaut: es gibt drei Anzeige-Optionen, welche die Zeilenbreite bestimmen. Es sind folgende: das „Narrow-Playfield" (narrow = eng) mit einer Breite von 128 Color Clocks, das „Normal-Playfield" mit einer Breite von 160 Color Clocks, und das „Wide-Playfield" (wide = ausgedehnt), welches eine Breite von 192 Color Clock besitzt. Diese Optionen werden durch Setzen des zugehörigen Bits im DMACTL-Register ausgewählt.

Wird feines waagerechtes Srolling benutzt, dann holt ANTIC mehr Daten aus dem RAM, als vom Display angezeigt werden. Wird DMACTL z.B. auf ein normales Playfield eingestellt, das im BASIC-Modus 0 vierzig Bytes pro Zeile besitzt, dann holt ANTIC in Wirklichkeit Daten, die denen eines Wide-Playfield entsprechen. (In diesem Falle wären es 49 Bytes pro Zeile.)

Hierdurch werden die Zeilen waagerecht verschoben, sofern diese Länge nicht berücksichtigt wird. Dieses Problem taucht allerdings nicht auf, wenn das RAM vom Programmierer bereits in lange waagerechte Zeilen aufgeteilt wurde, wie in Abbildung 6.1 dargestellt.

Das entsprechende Problem beim senkrechten Scrolling kann auf zwei Arten gelöst werden. Die erste Möglichkeit wäre, es einfach zu ignorieren. In diesem Fall ergeben sich an den Bildschirmrändern keine Halbbilder. Statt dessen werden die Bilder am unteren bzw. am oberen Rand nicht sauber gescrollt; sie springen plötzlich ins Bild. Der richtige Weg erfordert wenig Arbeit. Um ordentliches feines Scrolling zu erhalten (in und aus dem Bildschirm), muß eine Mode-Line als Buffer eingesetzt werden. Dieses geschieht, indem bei dieser (der letzten Mode-Line des zu scrollenden Bereiches) das Bit für horizontales Scrolling nicht gesetzt wird. Das Fenster wird zwar um eine Mode-Line gekürzt, scrollt dann aber ohne das irritierende Springen.

Es ist möglich, Bilder für den Schirm zu entwerfen, die eine Höhe von mehr als 192 Scan-Lines besitzen. Bei „festen" Displays würde dies störend wirken. Mit scrollenden Displays aber können Bilder, die sich ober- oder unterhalb der angezeigten Region befinden, immer in den sichtbaren Bereich gescrollt werden.

Feines Scrolling besitzt eine senkrechte Begrenzung von 16 Scan-Lines; das waagerechte Limit beträgt 16 Color Clocks. Wird versucht, über diese Begrenzungen hinauszugehen, ignoriert ANTIC einfach die höherwertigen Bits der Scrolling-Register. Um den gesamten Bildschirm fein zu scrollen (und zwar über eine beliebige Entfernung), muß feines mit grobem Scrolling gekoppelt werden. Als erstes wird das Bild fein gescrollt, wobei auf die gescrollte Entfernung geachtet wird. Entspricht der Betrag des feinen Scrollings der Größe eines Zeichens, wird das Feinscrolling-Regigter auf Null gesetzt und eine grobes Scrolling durchgeführt. Abbildung 6.2 veranschaulicht diesen Vorgang:

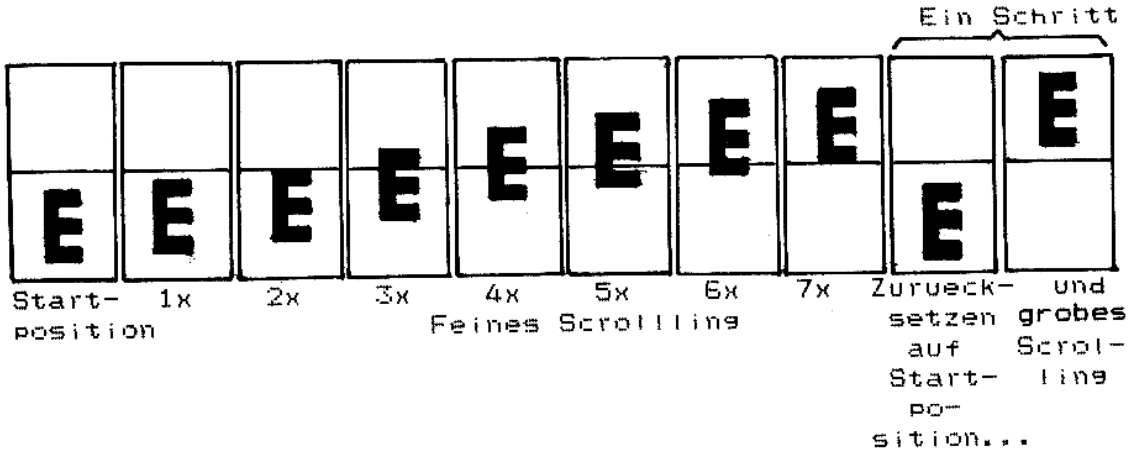


Abbildung 6.2.
Verbinden von feinem und grobem Scrolling

Das folgende Programm ist ein Beispiel für einfaches feines Scrolling:

```
1    HSCROL=54276
2    VSCROL=54277
10   GRAPHICS 0:LIST
20   DLIST=PEEK(560)+256*PEEK(561)
30   POKE DLIST+10,50:REM Für zwei Mode-Lines werden beide
40   POKE DLIST+11,50:REM Scrolling-Möglichkeiten
50   FOR Y=0 TO 7:REM eingeschaltet
60   POKE VSCROL,Y:REM Senkrecht scrollen
70   GOSUB 200:REM Verzögerung
80   NEXT Y
90   FOR X=0 TO 3
100  POKE HSCROL,X:REM Waagerecht scrollen
110  GOSUB 200:REM Verzögerung
120  NEXT X
130  GOTO 40
200  FOR J=1 TO 200
210  NEXT J:RETURN
```

Dieses Programm zeigt feines Scrolling in sehr langsamem Tempo und demonstriert einige Probleme, die beim Benutzen dieser Möglichkeit auftreten. Als erstes sind die Zeilen unter dem gescrollten nach rechts verschoben. Grund hierfür ist ANTIC, der sich mehr Daten holt (48 statt 40 Bytes pro Zeile). Dieses Problem taucht allerdings nur in unrealistischen Programmen auf; bei richtiger Verwendung des feinen Scrollings schließt eine ordnungsgemäße Aufteilung der Daten (wie in Abbildung 6.1. gezeigt) dieses aus. Das zweite und ernstzunehmendere Problem taucht auf, wenn ein Scrolling-Register geändert wird, während ANTIC sich in der Mitte seines Anzeigeprozesses

befindet. Diese Änderung „verwirrt" den ANTIC-Prozessor, und hat zur Folge, daß der Schirm zittert. Die Scrolling-Register sollten also nur während des Vertical-Blanks geändert werden. Dieses kann allerdings nur mit Assembler-Routinen geschehen. Folglich kann feines Scrolling nur mit Programmen in Maschinensprache erreicht werden.

ANWENDUNGEN

Die Anwendungen für feines Scrolling mit Graphiken sind zahllos. Die offensichtlichste Anwendung ist bei großen Karten gegeben, die mit Zeichensatzgraphiken aufgebaut wurden. Im Programm „Eastern Front" wurde mit dem BASIC-Modus 2 eine Karte von Rußland aufgebaut, die sich über 10 Bildschirme erstreckt. Der Fernsehschirm ist ein Ausschnitt dieser Karte. Der Benutzer ist durch Betätigen des Steuerknüppels in der Lage, über die gesamte Fläche zu scrollen. Dieses System ist sehr effizient: das gesamte Kartenprogramm mit Daten, Display-List und Zeichensatz-Definition benötigt einen Speicherbereich von ca. 4K-RAM.

Es gibt viele Verwendungen für diese Technik. Jedes mit Zeichensatz erstellte Bild kann mit diesem System verwendet werden. (Scrolling benötigt nicht unbedingt Zeichensatz-Graphiken. Map-Modi-Graphiken sind allerdings ungünstiger, da sie sehr viel Speicherplatz benötigen.) Auf diese Weise könnten große Schaltpläne gezeigt werden, wobei der Joystick zum Scrollen und zum Anzeigen spezieller Dinge auf dem Schirm benutzt werden könnte. Mit dieser Technik kann jedes Bild dargestellt werden, daß nicht ganz auf den Schirm paßt (Z.B. Baupläne).

Auch für große Textblöcke könnte dieses System verwendet werden. Da es aber nicht sehr praktisch ist, fortgesetzte Textteile durch Scrolling über das Display zu bewegen und dann zu lesen, ist es für unabhängige Textpassagen besser geeignet.

Eine andere Verwendung dieser Technik besteht im Anzeigen spezieller Menüs. Der Benutzer arbeitet mit dem Menü, indem er den Joystick zum Angeben der Operation verwendet. Hat er sich für etwas entschieden, plaziert er einfach über den Steuerknüppel ein Fadenkreuz an der zugehörigen Stelle und drückt den Feuerknopf. Obwohl dieses System nicht für alle Programme verwertbar ist, mag es für einige von um so größerem Nutzen sein.

Es gibt noch zwei weitere Anwendungen für das feine Scrolling. Die erste ist das selektive Feinscrolling. Hierbei werden verschiedene Scrolling-Bits unterschiedlicher Mode-Lines eingeschaltet. Normalerweise wird der gesamte Schirm gescrollt, dieses ist aber nicht unbedingt erforderlich. Ein Beispiel hierfür ist das ATARI„TM" DOS 2.0S.

Die zweite Möglichkeit ist, die Scrolling-Register mit Hilfe von Display-List-Interrupts zu ändern. Die Änderung des VSCROL-Registers ist ein schwieriges Unterfangen; ANTIC könnte durch diese verwirrt werden und unerwünschte Resultate anzeigen. Das Ändern des HSCROL-Registers mit dieser Methode ist zwar auch nicht ganz leicht, aber einfacher als das des VSCROL-Registers.

Kapitel 7
ATARI BASIC

Dieses Kapitel behandelt den BASIC-Interpreter des ATARI„TM" Personal Computer Systems. Die vier Hauptthemen sind:

1. **WAS IST ATARI BASIC** - Eine Beschreibung dessen, was erforderlich ist, um das BASIC laufen zu lassen und eine Erläuterung seiner Stärken und Schwächen.

2. **WIE DAS ATARI BASIC ARBEITET** - Eine detaillierte Analyse über die „Tokenisierung" und Ausführung von Programmen.

3. **VERBESSERUNG DER PROGRAMMAUSFÜHRUNG** - Eine Liste von Methoden, um die Ausführungsgeschwindigkeit eines Programmes zu erhöhen und seine Länge zu verkürzen.

4. **FORTGESCHRITTENE PROGRAMMIER-TECHNIKEN** - Eine Reihe von Möglichkeiten, um verschiedenen Programm-Erfordernissen besser gerecht zu werden.

WAS IST ATARI BASIC?

ATARI BASIC ist wie andere BASICs eine interpretierte Sprache. Das bedeutet, Programme können gestartet werden, sobald sie eingegeben wurden, und nicht erst, nachdem sie „compiliert" und „gelinkt" (d.h. in Maschinensprache übersetzt und im Speicher plaziert) worden sind. Der ATARI BASIC INTERPRETER liegt in einem 8K-ROM-Modul, das in den linken Schacht des Systems gesteckt wird. Dieses Modul belegt die Adressen $A000 bis $BFFF. Der BASIC-Interpreter speichert die Programme des Benutzers im RAM, wofür mindestens 8K-RAM benötigt werden.

Um das ATARI BASIC effektiv benutzen zu können, muß man seine Stärken und Schwächen kennen. Mit folgenden Informationen ist es möglich, Programme zu schreiben, die sämtliche Vorteile aus den Fähigkeiten und Möglichkeiten des ATARI BASICs ziehen.

Stärken des ATARI BASICs:

1. **ANSPRECHEN DER GRAPHIK-ROUTINEN DES OPERATING SYSTEMS** - Mit einfachen Graphik-Befehlen ist es möglich, ansprechende Bilder auf den Schirm zu bringen.

2. **ANSPRECHEN EINES TEILS DER HARDWARE** - Dieses betrifft z.B. die Befehle SOUND, STICK oder PADDLE.

3. **EINFACHER AUFRUF VON ASSEMBLER-ROUTINEN** - Die USR-Funktion gestattet dem Benutzer den Zugriff auf Assembler-Routinen.

4. **DER INTERPRETER LIEGT im ROM** - Dieses schützt vor versehentlichen Änderungen des BASIC-Interpreters durch den Programmierer oder sein Programm.

5. **ZUGRIFF AUF DAS DOS** - Spezielle Befehle, wie z.B. NOTE oder POINT (DOS 2.0S) gestatten dem Benutzer über das Disk-Operating-System einen freien Zugriff auf die Diskettenstationen.

6. **ZUGRIFF AUF DIE PERIPHERIE** - Jedes vom OS erkannte und akzeptierte Peripheriegerät kann über BASIC angesteuert werden.

Schwächen des ATARI BASICs:

1. **KEINE INTEGER-ARITHMETIK** - Alle Zahlen werden als 6 Byte-BCD Fließkomma-Zahlen gespeichert.

2. **LANGSAME MATHEMATISCHE BERECHNUNGEN** - Da alle Zahlen im oben genannten Format gespeichert werden, sind die Berechnungen entsprechend langsam.

3. **KEINE STRING-ARRAYS** - Es können nur eindimensionale Strings aufgebaut werden.

WIE DAS ATARI-BASIC ARBEITET

Ein kurzer Überblick über die Arbeit des BASIC-Interpreters.

1.
Der Interpreter bekommt vom Benutzer eine Zeile eingegeben und wandelt diese in eine Token-Form um.

2.
Diese Token-Zeile wird in das bisherige Token-Programm eingefügt.

3.
Das Programm wird ausgeführt.

Die Details dieser Operationen werden in den folgenden 4 Abschnitten besprochen:

A. Tokenisierung

B. Struktur der tokenisierten Dateien

C. Programmausführung

D. BASIC und das Betriebssystem

A. TOKENISIERUNG

Die Tokenisierung einer Zeile in BASIC sieht wie folgt aus:

1.
BASIC nimmt eine Zeile entgegen.

2.
Die Zeilensyntax wird überprüft (SYNTAX CHECK).

3.
Während dieses Checks wird die Zeile tokenisiert.

4.
Die tokenisierte Zeile wird in das Token-Programm eingefügt.

5.
Wurde die Zeile im Direkt-Modus, d.h. ohne Zeilennummer eingegeben, so wird sie sofort ausgeführt.

Um den Tokenisierungs-Vorgang besser zu verstehen, müßen als erstes einige Begriffe geklärt werden:

TOKEN
Ein 8-Bit-Zeichen, das einen speziellen interpretierbaren Code enthält.

STATEMENT
Eine vollständige Folge von Tokens, die BASIC veranlassen, etwas auszuführen. Beim LISTen werden die einzelnen Statements durch Doppelpunkte voneinander getrennt.

ZEILE
Ein oder mehrere Statements, denen entweder eine Nummer von 0 bis 32767, oder keine (im Falle des direkten Ausführens) vorangeht.

KOMMANDO
Der erste ausführbare Token eines Statements, das BASIC mitteilt, wie die folgenden Tokens zu interpretieren sind.

VARIABLE
Ein Token, der ein indirekter Zeiger zu seinem augenblicklichen Wert ist; der Wert kann geändert werden, ohne den Token zu ändern.

KONSTANTE
Ein 6-Byte-BCD-Wert, dem ein spezieller Token vorangeht. Dieser Wert wird während der Programmausführung nicht geändert.

OPERATOR
Einer von 46 Tokens, die auf irgendeine Weise den folgenden Wert ändern oder bewegen.

FUNKTION
Ein Token, der bei Ausführung einen bestimmten Wert für das Programm erzeugt.

EOL
"End of Line" (Ende der Zeile). Ein Zeichen mit dem hexadezimalen Wert 98.

Der BASIC-Interpreter beginnt den Tokenisierungs-Vorgang mit Empfang einer vom Benutzer geschriebenen Zeile. Diese Eingabe geschieht über einen Handler des OS. Normalerweise ist dieses der Bildschirm-Editor, mit dem ENTER-Kommando ist allerdings die Festlegung eines beliebigen Gerätes möglich. Der von BASIC an irgendein Gerät gesendete Befehl wird als GET RECORD-Kommando bezeichnet.

Die hierdurch erhaltenen Daten stehen im ATASCII-Format und werden durch ein EOL-Zeichen abgeschloßen. Sie werden mittels CIO in den sog. Eingabe-Zeilen-Buffer ($580 bis $5FF) gespeichert.

Nachdem das GET RECORD-Kommando aus und ein Syntax-Check durchgeführt wurde, beginnt der Tokenisierungs-Vorgang. Als erstes überprüft der BASIC-Interpreter, ob eine Zeilennummer angegeben wurde. Ist dieses der Fall, so wird sie in eine 2-Byte-Integer-Zahl umgewandelt. Ist keine Zeilennummer vorhanden, so wird Direktmodus angenommen und die Zeilennummer wird intern auf $8000 gesetzt. Diese beiden Bytes sind die ersten Tokens der tokenisierten Zeile, die in den Token-Ausgabe-Puffer gebracht wird. Dieser Puffer besitzt eine Länge von 256 Bytes und liegt am Ende des fürs Operating-System reservierten RAMs.

Das nächste Token ist ein sog. „Dummy"-Byte. Es ist für das Zählen der Bytes vom Anfang dieser Zeile bis zum Anfang der nächsten (=Offset) nötig. Danach folgt ein weiteres Dummy-Byte, das die Anzahl der Bytes vom Anfang dieser Zeile bis zum Anfang des nächsten Statements abgeschloßen wurde. Der Sinn dieser Parameter wird im Abschnitt zum Programm-Ausführungs-Prozeß erläutert.

Der BASIC-Interpreter überprüft nun den Befehl des ersten Statements der eingegebenen Zeile mit Hilfe der im ROM befindlichen Liste aller zugelassenen Kommandos. Wird ein passendes Kommando gefunden, dann wird das nächste Byte der tokenisierten Zeile die Nummer des zum Befehl passenden Eintrages der Liste im ROM. Wird kein passender Befehl gefunden, so wird das "Syntax-Error"- (Syntax-Fehler)-Token eingesetzt und der Interpreter stoppt den

Tokenisierungs-Vorgang. Die restliche Zeile im Eingabe-Buffer wird im ATASCII-Format in den Token-Ausgabe-Puffer kopiert, woraufhin die fehlerhafte Zeile ausgedruckt wird.

In einer zulässigen Zeile muß hinter einem Kommando eines der folgenden Dinge stehen: eine Variable, eine Konstante, ein Operator, eine Funktion, ein Anführungszeichen, ein anderes Statement oder ein EOL-Zeichen. Der BASIC-Interpreter testet, ob das nächste eingegebene Zeichen numerisch ist. Ist dieses nicht der Fall, dann wird das Zeichen (sowie die nachfolgenden) mit den bisherigen Einträgen in eine Variablenliste verglichen. (Bei der ersten eingegebenen Zeile kann hierbei natürlich nichts gefunden werden, da es noch keine Einträge gibt.) Die Zeichen werden dann mit den Funktions- und Operatortafeln verglichen. Wird hierbei ebenfalls nichts gefunden, so nimmt der Interpreter an, daß es sich um eine neue Variable handelt. (Da dieses in der ersten Zeile die erste Variable ist, ist es auch der erste Eintrag in die Variablenliste.) Die Zeichen werde aus dem Eingabe-Buffer in die Variablenliste kopiert, wobei das höchstwertigste Bit (MSB) des letzten Bytes vom Variablen-Namen gesetzt wird. Danach werden für diesen Eintrag 8 Bytes in der Variablentafel reserviert. (Siehe Erklärung der Variablentafel im Abschnitt über die Struktur der tokenisierten File) Das Token, das für eine Variable in eine tokenisierte Zeile eingesetzt wird, ist die Nummer derselben minus 1, wobei das MSB gesetzt ist. Das bedeutet, das Token der ersten eingegebenen Variablen wäre $80, der zweiten $81 usw., bis $FF, was insgesamt 128 verschiedene Variablen-Namen zuläßt.

Wird eine Funktion gefunden, dann nimmt das erste Token den zugehörigen Wert der Funktionstafel an. Funktionen erfordern bestimmte Folgen von Parameter; diese sind in Syntax-Tafeln enthalten. Stimmen sie nicht überein, so wird eine Syntax-Fehlermeldung ausgegeben.

Wird ein Operator gefunden, so enthält das Token die zugehörige Nummer der Operatortabelle. Operatoren können in sehr komplexer Form aufeinander folgen (z.B. verschachtelte Klammern), daher ist der entsprechende Syntax-Check etwas komplizierter.

Im Falle von Anführungszeichen nimmt der Interpreter an, daß ein String (= Reihe, Folge) von Zeichen folgt. Das Token enthält den hexadezimalen Wert 0F, wobei ein Dummy-Byte für die Länge des Strings reserviert wird. Die Zeichen werden dann aufeinanderfolgend von Eingabe-Puffer in den Ausgabe-Puffer übertragen, bis ein weiteres Anführungszeichen gefunden wird. Das Dummy-Byte erhält dann einen Wert, die Anzahl der Zeichen dieses Strings.

Ist das folgende Zeichen im Eingabe-Puffer numerisch, so wird es vom BASIC-Interpreter in eine 6-Byte-BCD-Konstante umgewandelt. Ein Token mit dem wert $0E, dem die

6-Byte-Konstante folgt, wird danach in den Ausgabe-Puffer gebracht.

Stößt der Interpreter auf einen Doppelpunkt, so wird eine $14 als Token eingesetzt und in den Ausgabe-Puffer gebracht. Das vorher bereitgestellte Dummy-Byte erhält ebenfalls seinen Wert, d.h. die Bytes vom Anfang der Zeile bis zum Anfang des nächsten Statements. Danach wird ein weiteres Dummy-Byte reserviert und der BASIC-Interpreter beginnt wieder mit dem Abfragen eines Befehls.

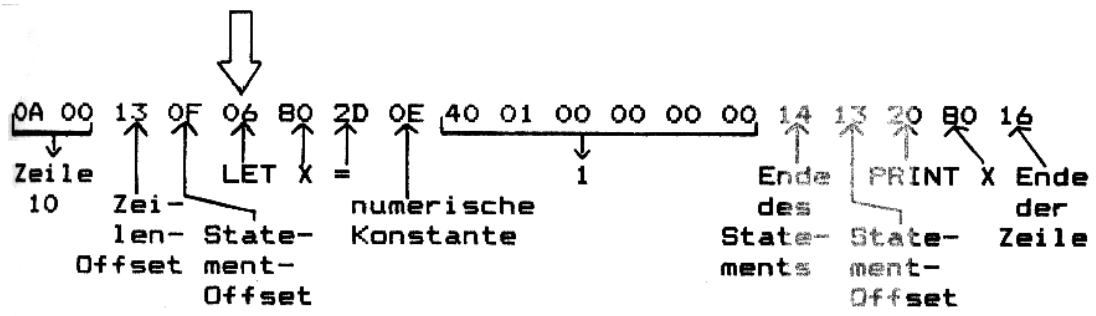
Wenn ein EOL-Zeichen gefunden wird, dann wird ein $16-Token gespeichert und die Anzahl der Bytes vom Zeilenanfang in das für den Zeilen-Offset reservierte eingesetzt. Zu diesem Zeitpunkt ist die Tokenisierung abgeschloßen, und der BASIC-Interpreter bringt die tokenisierte Zeile in das Token-Programm. Als erstes wird hierbei nach der Nummer der eben tokenisierten Zeile gesucht. Wird die Nummer gefunden, so wird die betreffende Zeile durch die neue ersetzt. Andernfalls wird die neue Zeile an der (numerisch) richtigen Stelle in das tokenisierte Programm eingefügt. In beiden Fällen werden die Daten, die der Zeile folgen, im Speicher auf- oder abbewegt, wodurch das Programm weiter ausgedehnt bzw. verkleinert wird.

Der Interpreter prüft nun, ob die tokenisierte Zeile direkt ausgeführt werden soll. Ist dieses der Fall, dann führt er diese wie unter Interpretationsprozeß beschrieben aus. Andernfalls wartet der BASIC-Interpreter auf eine neue Zeile.

Überschreitet die Länge der tokenisierten Zeile zu irgendeinem Zeitpunkt 256 Bytes, so wird eine „Fehler 14"-Meldung auf dem Bildschirm ausgegeben. Danach wartet der Interpreter wieder auf eine neue Zeile.

Die Tokenisierung einer Zeile sieht wie folgt aus (alle Werte stehen in hexadezimaler Notation.):

10 LET X=1:PRINT X



KOMMANDO			OPERANDEN			FUNKTIONEN		
Hex	Dez		Hex	Dez		Hex	Dez	
00	0	REM	0E	14	(numer. Konstante)	3D	61	STR$
01	1	DATA	0F	15	(String-Konstante)	3E	62	CHR$
02	2	INPUT	10	16	„	3F	63	USR
03	3	COLOR	11	17	(nicht belegt)	40	64	ASC
04	4	LIST	12	18	,	41	65	VAL
05	5	ENTER	13	19	$	42	66	LEN
06	6	LET	14	20	: (Ende d. Statem.)	43	67	ADR
07	7	IF	15	21	;	44	68	ATN
08	8	FOR	16	22	(Ende der Zeile)	45	69	COS
09	9	NEXT	17	23	GOTO	46	70	PEEK
0A	10	GOTO	18	24	GOSUB	47	71	SIN
0B	11	GO TO	19	25	TO	48	72	RND
0C	12	GOSUB	1A	26	STEP	49	73	FRE
0D	13	TRAP	1B	27	THEN	4A	74	EXP
0E	14	BYE	1C	28	=	4B	75	LOG
0F	15	CONT	1D	29	<= (Numerische)	4C	76	CLOG
10	16	COM	1E	30	<>	4D	77	SGR
11	17	CLOSE	1F	31	>=	4E	78	SGN
12	18	CLR	20	32	<	4F	79	ABS
13	19	DEG	21	33	>	50	80	INT
14	20	DIM	22	34	=	51	81	PADDLE
15	21	END	23	35	.	52	82	STICK
16	22	NEW	24	36	*	53	83	PTRIG
17	23	OPEN	25	37	+	54	84	STRIG
18	24	LOAD	26	38	-			
19	25	SAVE	27	39	/			
1A	26	STATUS	28	40	NOT			
1B	27	NOTE	29	41	OR			
1C	28	POINT	2A	42	AND			
1D	29	XIO	2B	43	(			
lE	30	ON	2C	44	)			
1F	31	POKE	2D	45	= (Arithmetische)			
20	32	PRINT	2E	46	= (Strings)			
21	33	RAD	2F	47	<=(Strings)			
22	34	READ	30	48	<>			
23	35	RESTORE	31	49	>=			
24	36	RETURN	32	50	<			
25	37	RUN	33	51	>			
26	38	STOP	34	52	=			
27	39	POP	35	53	+ (Vorzeichen)			
28	40	?	36	54	- (Vorzeichen)			
29	41	GET	37	55	((String)			
2A	42	PUT	38	56	((Array)			
2B	43	GRAPHICS	39	57	((DIM Array)			
2C	44	PLOT	3A	58	((Sonstige)			
2D	45	POSITION	3B	59	((DIM String)			
2E	46	DOS	3C	60	, (Array-Komma)			
2F	47	DRAWTO						
30	48	SETCOLOR						
31	49	LOCATE						
32	50	SOUND						
33	51	LPRINT						
34	52	CSAVE						
35	53	CLOAD						
36	54	(implizites LET)						
37	55	ERROR (Syntax)						

B. STRUKTUR DER TOKENISIERTEN DATEI

Die tokenisierte Datei enthält zwei Hauptbestandteile: 1) eine Gruppe von Zero-Page-Zeigern, die in die tokenisierte Datei zeigen und 2) die tokenisierte Datei selbst. Die Zero-Page-Zeiger sind 2-Byte-Werte, die auf verschiedene Abschnitte der Token-Datei zeigen. Es gibt 9 dieser 2-Byte-Zeiger. Sie liegen bei den Adressen $80 bis $91. Die folgende Liste reiht die Zeiger und die Abschnitte der Token-Datei auf, auf welche sie zeigen.

Zeiger (Hex)	Abschnitt der Token-Datei (Fortlaufende Blöcke)
LOMEM 80,81	**Token Ausgabe-Puffer** - Diesen Puffer benutzt der BASIC-Interpreter, um eine Zeile zu codieren. Er ist 256 Bytes lang und liegt am Ende des vom OS benutzten RAMs.
VNTP 82,83	**Variablen-Namentafel** - Eine Liste aller in das Programm eingegebenen Variablen. Diese werden als ATASCII-Zeichen gespeichert und stehen in der gleichen Reihenfolge, in der sie in das Programm eingegeben wurden. Es gibt drei verschiedene Arten von Einträgen: **1.numerische Variablen** - MSB des letzten Zeichens des Namens gesetzt. **2.String-Variablen** - Das letzte Zeichen ist ein „$", dessen MSB gesetzt ist. **3.Variablenfelder** - Das letzte Zeichen ist ein "(", dessen MSB gesetzt ist.
VNTD 84,85	**Ende der Variablen-Namenstafel** - Der BASIC-Interpreter benutzt diesen Zeiger zum Anzeigen des Namenstafel-Endes. Dieser zeigt normalerweise auf ein Dummy-Byte mit dem Wert Null (weniger als 128 Variablennamen). Gibt es 128 Variablen, so zeigt er auf das letzte Byte des letzten Variablennamens.
VVTP 86,87	**Variablen-Wertetafel** - Diese Tafel enthält den augenblicklichen Wert jeder Variablen. Die Information für den jeweiligen Variablentyp sieht wie folgt aus:

Byte-Nummer: 1	2	3 4	5 6	7 8
num. Var. 0	Var-Nr.	6-Byte-BCD-Konstante		
Feld (DIM) 41 (ohne DIM) 40	Var-Nr.	Offset von STARP (8C,8D)	erstes DIM+1	zweites DIM+1
STRING (DIM) 81 (ohne DIM) 80	Var-Nr.	Offset von STARP (8C,8D)	Länge	DIM

Eine numerische Variable enthält einen numerischen Wert. Ein Beispiel hierfür wäre X=1. X ist die numerische Variable und 1 ihr Wert (6-Byte BCD-Format). Ein Feld wird aus numerischen Werten zusammengesetzt, die im String/Array-Bereich gespeichert werden. Er besitzt einen Eintrag in die Wertetafel. Ein String, bestehend aus im String/Array-Bereich gespeicherten Zeichen, besitzt ebenfalls einen Eintrag in der Wertetafel.

Das erste Byte jedes Eintrages benennt den Variablentyp: 00 steht für eine Variable, 40 für ein Feld und 90 für einen String. Wurde das Feld oder der String dimensioniert, dann wird das LSB des ersten Bytes gesetzt.

Im Falle der numerischen Variablen enthalten die Bytes 3 bis 8 die 6-Byte-BCD-Zahl, die ihrem augenblicklichen Werten entsprechen.

Für Strings und Felder enthalten die Bytes 3 und 4 den Offset von Anfang des Strings/Array-Bereiches bis zum Anfang der Daten (siehe unten).

Das fünfte und sechste Byte eines Arrays enthalten den ersten Parameter des DIM-Befehls. Die Zahl ist ein 2-Byte-Integer, dessen Wert um 1 größer ist, als der vom Benutzer eigegebene. Das siebte und, achte Byte enthalten den zweiten Parameter des DIM-Befehls, dessen Wert ebenfalls um 1 größer ist, als der vom Benutzer eingegebene.

STMTAB 88,89 **Statement-Tafel** - Dieser Datenblock enthält alle vom Benutzer eingegebenen Zeilen, die von BASIC tokenisiert wurden. Zeilen im Direktmodus werden ebenfalls hier abgelegt. Das Format dieser Zeile wird im Abschnitt über den Tokenisierungs-prozeß beschrieben.

STMCUR 8A,8B **Augenblickliches Statement** - Dieser Zeiger wird vom BASIC-Interpreter benutzt, um auf bestimmte Tokens innerhalb einer Zeile der Statement-Tafel hinzuweisen. Erwartet der Interpreter eine Eingabe, so wird dieser Zeiger auf den Anfang der Direktmodus-Zeile gesetzt.

STARP 8C,8D **String/Array-Bereich** - Dieser Block enthält alle

String- und Felddaten. String-Zeichen werden als jeweils ein Byte ATASCII-Code gespeichert, d.h. ein String von 20 Zeichen Länge benötigt 20 Bytes des Speichers. Felder werden im 6-Byte-BCD-Format gespeichert. Ein Feld mit 10 Elementen benötigt demnach 60 Bytes des RAMs.

Dieser Bereich wird bei jedem neuen DIM-Befehl um die Länge bzw. das 6-fache der Feldgröße vergrößert.

RUNSTK 8E,8F **RUN-TIME-STACK** - Dieser Software-Stapel enthält GOSUB- und FOR/NEXT-Einträge. Der GOSUB-Eintrag besteht aus 4 Bytes. Das erste ist eines mit dem Wert 0, welches das GOSUB anzeigt. Danach folgt als 2-Byte-Integer die Zeilennummer, in welcher die Anweisung steht. Schließlich folgt ein Byte, welches den Offset des GOSUBs zum Zeilenanfang angibt, so daß der Interpreter wieder dorthin zurückspringen kann (RETURN), um das nächste Statement auszuführen.

Jeder FOR/NEXT-Eintrag besteht aus 16 Bytes. Der erste Eintrag enthält den Endwert für den Zähler. Der zweite ist die Schrittweite, d.h. der Wert, um den der Zähler jedesmal erhöht wird. Jeder dieser beiden Werte wird im 6-Byte-BCD-Format gespeichert. Das 13. Byte ist die Nummer der Zählervariablen, deren MSB gesetzt ist. Das 14. und 15. Byte enthalten die Zeilennummer, und das 16. Byte ist der Zeilenoffset des FOR-Statements.

MEMTOP 90,91 **Oberstes Ende des verwendeten RAMs** - Dieses ist das Ende des BASIC-Programms. Die Ausdehnung des Programms kann von hier aus bis zum Ende des freien RAMs, das durch den Anfang der Display-List festgelegt wird, geschehen. Die FRE-Funktion berechnet die Größe des freien RAMs, in dem MEMTOP von HIMEM ($2E5,$2E6) abgezogen wird. Anmerkung: MEMTOP des BASICs ist nicht identisch mit der OS-Variablen MEMTOP!

C. PROGRAMMAUSFÜHRUNG

Um eine Programmzeile auszuführen, ist es erforderlich, die, Tokens, die während dem Tokenisierungs-Vorgangs erzeugt werden, zu lesen. Jedes Token hat eine bestimmte Bedeutung und veranlaßt den Interpreter zur Ausführung einer Folge von Operationen. Hierfür ist es erforderlich, daß sich der BASIC-Interpreter jeweils ein Token aus dem tokenisierten Programm holt und dieses ausführt. Das Token ist ein Index zu einer Sprungtafel von Routinen. Das bedeutet, ein PRINT-Token zeigt auf eine Routine zur Durchführung eines solchen Befehls. Nach Abschluß der Routine(n) holt sich der Interpreter das nächste Token. Der Zeiger, der auf das neue Token zeigt, heißt STMCUR und liegt bei $8A und $8B.

Die erste Zeile, die innerhalb eines Programmes ausgeführt wird, ist die Direktmodus-Zeile. Sie enthält gewöhnlich ein RUN oder GOTO xxxxx. Im Falle eines RUN-Befehls holt der Interpreter sich die erste tokenisierte Zeile aus der Statement-Tafel und führt diese aus. Weisen die Tokens in der Zeile nicht auf eine andere, dann fährt der BASIC-Interpreter nach Durchführung dieser Zeile mit der Abarbeitung der nächsten fort.

Wird auf ein GOTO gestoßen, dann muß die anzuspringende Zeile gefunden werden; Die Statement-Tafel enthält eine verkettete Liste von Zeilennummern und Statements, wobei die niedrigste Nummer an erster und die höchste an letzter Stelle steht. Wird eine andere Zeile in der Mitte der Tafel benötigt, dann läuft der folgende Prozeß ab:

Die Adresse der ersten Zeile wird im STMTAB-Zeiger ($88,$89) gefunden und in einem Hilfszeiger gespeichert. Die ersten zwei Bytes der Zeile sind ihre Nummer. Diese wird mit der anzuspringenden Zeilennummer verglichen. Ist die Nummer kleiner, dann holt sich der Interpreter die nächste Zeile, indem der Wert des dritten Bytes der ersten Zeile zum Wert des Hilfszeigers addiert wird. Hierdurch zeigt der Hilfszeiger auf die zweite Zeile. Nun werden wieder die ersten beiden Bytes der neuen Zeile mit dem Wert der anzuspringenden verglichen. Sind die ersten beiden Bytes kleiner, so wird das dritte Byte zum Hilfszeiger addiert.

Ist die anzuspringende Zeilennummer gefunden, dann wird der Hilfszeiger nach STMCUR übertragen. Der Interpreter holt sich dann das nächste Token aus der neuen Zeile.

Wenn die gesuchte Zeile nicht gefunden wird, dann gibt der BASIC-Interpreter eine „ERROR 12"-Meldung aus („LINE NOT FOUND" - Zeile nicht gefunden).

Die Ausführung eine GOSUBs erfordert mehr Arbeit als die eines GOTOs. Die Routine für das Auffinden der gewünschten Zeile ist die gleiche. Bevor der Interpreter aber zur neuen Zeile geht,

wird ein Eintrag auf den RUN-TIME-STACK gebracht. Dieser Eintrag umfaßt 4 Bytes am Ende den Stapels und speichert eine 0 im ersten Byte, um anzuzeigen, daß es sich um einen GOSUB-Befehl handelt. Als nächstes werden die beiden Bytes, welche die Zeilennummer angeben, in der der GOSUS-Befehl steht, auf den Stapel gebracht. Das letzte Byte enthält den Offset vom Anfang der Zeile bis zum GOSUB-Token. Danach führt der BASIC-Interpreter die angesprungene Zeile aus. Wenn er auf eine RETURN-Anweinung stößt, dann holt er den letzten Eintrag vom Stapel und geht zu der Zeile zurück, in welcher die GOSUB-Anweisung stand. Danach wird der nächste Befehl ausgeführt.

Das FOR-Kommando veranlaßt den Interpreter zu einer Reservierung von 16 Bytes auf dem RUN-TIME-STACK. Die ersten 6 Bytes geben den Endwert an, den die Zählervariable erreichen kann (6-Byte-BCD-Format). Die nächsten 6 Bytes enthalten die Schrittweite im gleichen Format. Danach wird die Variablennummer (mit gesetztem MSB) gespeichert. Als nächstes folgt die augenblickliche Zeilennummer (2 Bytes) und der Offset in die Zeile. schließlich wird der Rest der Zeile ausgeführt.

Findet der Interpreter das NEXT-Kommando, so ließt er den letzten Stapeleintrag. Es wird überprüft, ob die Variable, die durch das NEXT ausgegeben wird, die gleiche ist, die als letztes auf den Stapel gebracht wurde. Außerdem wird überprüft, ab der Zähler den Endwert erreicht oder überschritten hat. Ist dieses nicht der Fall, dann kehrt der BASIC-Interpreter zur Zeile mit der FOR-Anweisung zurück und fährt mit der Programm-Ausführung fort. Wird der Endwert des FOR-Befehls erreicht oder überschritten, so wird der FOR/NEXT-Eintrag vom Stapel geholt und die Programmausführung von diesem Punkt an fortgesetzt.

Stößt der Interpreter auf einen mathematischen Ausdruck, dann werden die Operatoren auf einen Operatoren-Stapel gebracht. Danach werden sie nacheinander wieder vom Stapel geholt und ausgeführt. Die Reihenfolge, in der sie auf den Stapel gebracht werden, kann beeinflußt werden. Werden die einzelnen Operatoren durch Klammern eingeschlossen so werden sie in der Reihenfolge, in der sie abgearbeitet worden sollen, auch auf den Stapel gebracht. Andernfalls „schaut" der Interpreter in einer ROM-Tafel „nach", in der die Prioritäten der einzelnen Operatoren verzeichnet sind, und bringt sie dementsprechend auf den Stapel (z.B. Punkt- vor Strichrechnung).

Wird zu irgendeinem Zeitpunkt die BREAK-Taste gedrückt dann wird vom OS ein Flag gesetzt, um dieses Ereignis festzuhalten. Der BASIC-Interpreter überprüft dieses Flag jeweils nach der Ausführung eines Tokens. Wenn dieses Flag gesetzt ist, wird die Nummer der Zeile, in der die Taste gedrückt wurde, gespeichert und eine „STOPPED AT LINE XXXXX"-Meldung ausgegeben, bei der dieser Wert eingesetzt wird. Danach wird

das BREAK-Flag gelöscht und auf eine Eingabe vom Benutzer gewartet. Durch Eingabe eines „CONT"-Befehls kann der Benutzer erreichen, daß die Programmausführung ab der nächsten Zeile wieder aufgenommen wird.

D. BASIC UND DAS BETRIEBSSYSTEM

Der BASIC-Interpreter benutzt das OS hauptsächlich durch I/O-Aufrufe des CIO. Die folgende Liste zeigt eine Reihe durch den Benutzer ausführbare Aufrufe und die zugehörigen IOCBs des OS.

BASIC	OS
OPEN #1,12,0,E:	IOCB=l Kommando=3 (OPEN) Aux1=12 (Ein-/Ausgabe) Aux2=0 Pufferadresse=ADR("E:")
GET #1,X	IOCB=! Kommando=7 (GET CHARACTER) Pufferlänge=0 => Zeichen wird im Akku- mulator übergeben
PUT #1,X	IOCB=1 Kommando=ll (PUT CHARACTER) Pufferlänge=0 => Zeichenausgabe aus dem Akkumulator
INPUT #1,A$	IOCB=l Kommando=5 (GET RECORD) Pufferlänge=Größe von A$ (nicht über 120) Pufferadresse=Eingabe-Zei- len-Puffer
PRINT #I,A$	IOCB=l Der Interpreter benutzt einen speziellen PUT BYTE- Vektor im IOCB, um direkt mit dem Handler zu kommunizieren.
XIO 18,#6,12,0,”S:”	IOCB=6 Kommando=18 (FILL-Kommando) Aux1=12 Aux2=0

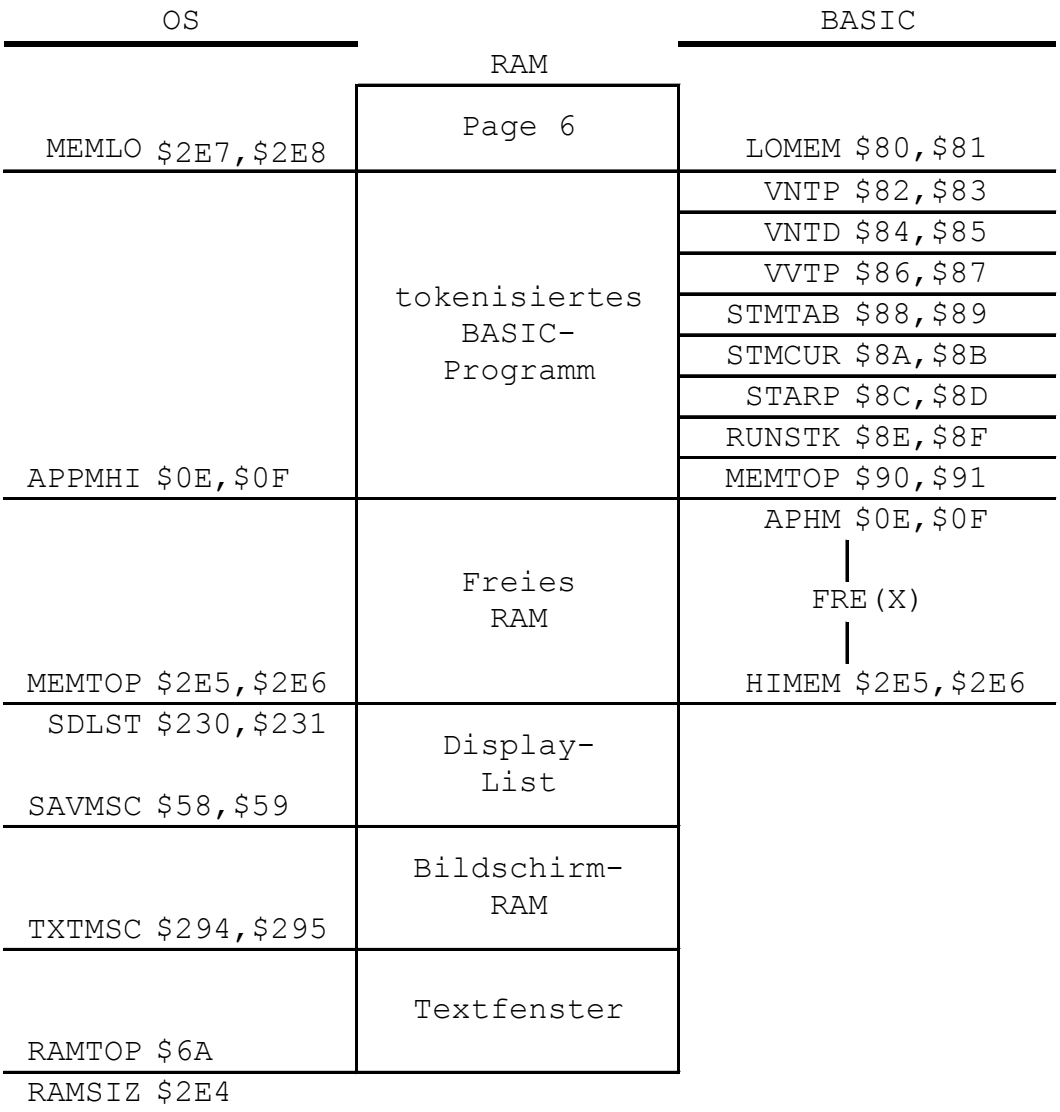
SAVE/LOAD: Wenn ein Token-Programm auf einem Gerät mit SAVE gespeichert wird, dann werden zwei Informationsblöcke geschrieben. Der erste Block besteht aus 7 von 9 Zero-Page-Zeigern, die der Interpreter benutzt, um die Token-Files zu verwalten und aufrecht zu erhalten. Diese Zeiger sind LOMEM ($80,$81) bis STARP ($8C,$8D). Bei diesem Schreiben wird allerdings eine Änderung gemacht: die

2-Byte-Zeiger werden erst auf ein Gerät geschrieben, nachdem der Wert des LOMEM-Zeigers von jedem subtrahiert wurde. Dieses bedeutet, daß die ersten beiden gespeicherten Werte immer 0 und 0 sind.

Der zweite gespeicherte Informationsblock besteht aus folgenden Abschnitten der tokenisierten Datei:
1) der Namenstafal der Variablen,
2) der Wertetafel der Variablen,
3) dem tokenisierten Programm und
4) der Direktmodus-Zeile.

Wird dieses Programm wieder in den Speicher geladen (LOAD), dann „sieht" der Interpreter „auf" die OS-Variable MEMLO ($2E7,$2E8), und addiert deren Wert zu jedem der 2-Byte-Zero-Page-Zeiger, sobald diese gelesen werden. Die Zeiger werden wieder in der Zero-Page des Speichers plaziert, wonach die Werte von RUNSTK ($8E,$8F) und MEMTOP ($90,$91) auf den Wert von STARP gebracht werden.

Als nächstes werden 256 Bytes ab der Speicherstelle reserviert, die von MEMLO angegeben wird, um Platz für den Token-Ausgabe-Puffer zu schaffen. Danach wird die Information der tokenisierten Datei eingelesen. Diese Daten werden genau hinter dem Token-Ausgabe-Puffer plaziert.



OS- und BASIC-Zeiger (ohne DOS)

VERBESSERUNG DER PROGRAMMAUSFÜHRUNG

Um die Effektivität eines Programms zu steigern, können zwei Dinge getan werden. Zum einen kann die Ausführungszeit, zum anderen der benötigte Speicherplatz verringert werden. Um diese beiden Ziele zu erreichen, kann der Benutzer die nachfolgend Tips verwenden. Die einzelnen Methoden sind in einer Folge aufgereiht, bei der die erste die größte und die letzte die geringste Effektivität besitzt.

Methoden zur Erhöhung der Ausführungsgeschwindigkeit von BASIC-Programmen:

1. **NEUCODIEREN** - Da BASIC keine strukturierte Sprache ist, werden in dieser Sprache geschriebene Programme nicht ineffizient. Ein Programm kann auch nach mehreren Überarbeitungen noch immer nicht optimal sein. Es lohnt sich also immer, ein Programm noch einmal durchzusehen.

2. **UBERPRÜFEN DER ALSORITHMISCHEN LOGIK** - Der Programmteil für die Durchführung einer Operation muß so durchschlagend wie nur irgendmöglich sein.

3. **Oft BENUTZTE UNTERPROGRAMME UND FOR/NEXT-SCHLEIFEN MÖGLICHST AN DEN ANFANG DES PROGRAMMES STELLEN** - Der Interpreter beginnt mit der Suche nach einer Zeilennummer immer am Anfang des Programms, so daß weiter hinten stehende Zeilen erst sehr viel später erreicht werden.

4. **MEHRFACH BENUTZTE OPERATIONEN INNERHLAB EINER SCHLEIFE DIREKT ALS UNTERPROGRAMM EINGEBEN** - Der BASIC-Interpreter benötigt sehr viel Zeit, um den RUN-TIME-STACK zu aktualisieren.

5. **DIE SICH AM HÄUFIGSTEN ÄNDERNDE SCHLEIFE SOLLTE BEI VERSCHACHTELTEN SCHLEIFEN DIE INNERSTE SEIN** - Der RUN-STACK-TIME wird dadurch seltener benutzt.

6. **VEREINFACHEN DER FLIEßKOMMA-ARITHMETIK INNERHALB VON SCHLEIFEN** - Wird ein Ergebnis durch Multiplikation mit dem Index erreicht, so sollte diese statt dessen in eine Addition von Konstanten umgewandelt werden.

7. **SCHLEIFEN IN EINE ZEILE BRINGEN** - Der BASIC-Interpreter muß dadurch nicht erst in die nächste Zeile springen, um die Schleife fortzusetzen.

9. **ABSCHALTEN DES BIDSCHIRMS** - Ist eine Bildschirmanzeige (über einen Zeitraum) nicht unbedingt erforderlich, so können 30% der Ausführungszeit durch Setzen einer 0 in die Speicherstelle 559 gespart werden (POKE 559,0).

9. **BENUTZEN EINES SCHNELLEREN GRAPHIKMODUS´ BZW. EINER KÜRZEREN DISPLAY LIST** - Ist nicht unbedingt ein voller Bildschirm erforderlich, so können hierdurch bis zu 25% der Ausführungszeit gespart werden.

10. **BENUTZEN VON ASSEMBLER-UNTERPROGRAMMEN** - Durch Aufrufen von Assembler-Unterprogrammen über die USR-Funktion kann die Ausführungszeit eines Programms verkürzt werden.

Sparen von Speicherplatz:

1. **NEUCODIEREN** - Wie schon genannt. Durch Neuorganisation das Programms kann Speicherplatz eingespart werden.

2. **LÖSCHEN VON BEMERKUNGEN** - REMs werden als ATASCII-Daten gespeichert und verbrauchen dementsprechend Speicherplatz.

3. **ERSETZEN VON KONSTANTEN, DIE MEHR ALS DREIMAL BENUTZT WERDEN** - Der BASIC-Interpreter benötigt für eine Konstante 7 Bytes, für einen Variablenaufruf aber nur 1 Byte. Durch das Einsetzen von Variablen können also jeweils 6 Bytes gespart werden.

4. **INITIALISIEREN VON VARIABLEN MIT DER READ-ANWEISUNG** - Ein Data-Statement wird als ATASCII-Code gespeichert, d.h. ein Byte pro Zeichen. Durch Gleichsetzen mit einer Konstanten gehen Jedesmal 7 Bytes verloren.

5. **ERSETZEN VON KONSTANTEN DURCH AUSDRUCKE AUS BEREITS BENUTZTEN VARIABLEN** - so kann eine 1 durch Z1, und eine 2 durch Z2 ersetzt werden. Wird die Zahl 3 benötigt, so schreibt man Z1+Z2.

6. **HÄUFIG BENUTZTE ZEILENNUMMERN IN VARIABLEN UMWANDELN** - Wird z.B. die Zeile 100 über einen GOSUB- oder GOTO-Befehl 50 mal angesprungen, so können ca. 300 Bytes gespart worden, indem man die 100 jedes Befehls durch ein Z100 ersetzt (GOTO Z100).

7. **DIE ANZAHL DER VARIABLEN MÖGLICHST KLEIN HALTEN** - Jeder neue Eintrag in die Namenstafel für Variablen verbraucht 8 Bytes, zuzüglich der Bytes des Namens.

8. **LöSCHEN DER NICHT MEHR BENÖTISTEN VARIABLEN** - Variablen werden nicht durch Löschen aus dem Programm aus der Namenstafel entfernt. Um sie aus letzterer zu entfernen, muß das Programm auf Diskette oder Cassette ge**LIST**et werden. Danach wird **NEW** eingegeben und das Programm über **ENTER** wieder geladen.

9. **VARIABLENNAMEN KURZ HALTEN** - Jeder Variablenname wird in der Namenstafel als ATASCII-Daten gespeichert. Je kürzer die einzelnen Namen sind, desto kürzer ist die Tafel, d.h. umso um so weniger Speicher wird verbraucht.

10. **ERSETZEN VON MEHRFACH BENUTZTEM TEXT DURCH STRINGS** - Durch diese Methode kann viel Speicherplatz gespart werden, da jeder Buchstabe eines Textes einem Byte im Speicher entspricht.

11. **INITIALISIEREN EINES STRINGS DURCH GLEICHSETZEN** - Ein Gleichsetzen mit Zeichen innerhalb zweier Anführungszeichen benötigt weniger Speicherplatz, als eine READ-Anweisung oder eine CHR$-Funktion.

12. **ZUSAMMENFASSEN VON ZEILEN** - 3 Bytes können gespart werden, wenn anstelle von zwei Zeilen mit jeweils einem Statement eine Zeile mit zwei durch Doppelpunkt getrennten Statements geschrieben wird.

13. **VERMEIDEN NUR EINMAL BENUTZTER UNTERPROGRAMME** - Nur einmal aufgerufene Unterprogramme sollten anstelle ihres Aufrufes direkt in den Programmcode eingefügt werden (GOSUB und RETURN verschwenden Bytes, wenn sie nur einmal benutzt werden).

14. **ERSETZEN VON NUMERISCHEN FELDERN DURCH STRINGS, SOFERN DIE DATENWERTE NICHT GRÖßER ALS 255 WERDEN** - Einträge für numerische Felder verbrauchen jeweils 5 Bytes, wogegen Stringelemente nur ein Byte benötigen.

15. **ERSETZEN VON SETCOLOR-BEFEHLEN DURCH POKE-KOMMANDOS** - Dieses erspart jedesmal 9 Bytes.

16. **ERSETZEN VON POSITION-ANWEISUNGEN DURCH CURSOR-STEUERCODES** Die POSITION-Anweisung benötigt 15 Bytes für X- und Y-Parameter, wogegen ein Cursor-Steuerzeichen nur ein Byte verbraucht.

17. **LÖSCHEN VON ZEILEN DURCH DAS PROGRAMM SELBST** - Siehe Abschnitt „Fortgeschrittene Programmier-Techniken".

18. **MODIFIZIEREN DES STRING/ARRAY-ZEIGBERS ZUM LADEN VON VORHER DEFINIERTEN DATEN** - Siehe Abschnitt über fortgeschrittenel Programmier-Techniken.

19. **VERKETTEN VON PROGRAMMEN** - Ein Programmteil wird vom Benutzer gestartet und lädt dann nach seiner Ausführung selbsttätig den nächsten Teil von Diskette oder Cassette. Diese Technik läßt sich beliebig oft wiederholen.

FORTGESCHRITTENE PROGRAMMIER-TECHNIKEN

Wenn die Grundprinzipien des ATARI BASICs verstanden worden sind, können einige interessante Programme geschrieben werden, die diese Prinzipien anwenden. Es können sowohl reine BASIC-Programme sein, als auch solche, die die Möglichkeiten des OS mit einbeziehen.

BEISPIEL 1 -*String-Initialisierung* - Dieses Progamm setzt alle Bytes eines Strings (bliebiger Länge) auf den gleichen Wert. Der BASIC-Interpeter kopiert das erste Byte des Quellenstrings in das erste Byte des Zielstrings. Danach folgt das zweite, das dritte usw. Da der Zielstring das zweite Byte des Quellenstrings ist, wird das gleiche Zeichen in den gesamten String geschrieben.

BEISPIEL 2 - *Löschen von Programmzeilen* - Durch Benutzen einer Möglichkeit des OS kann ein Programm Zeilen selbst löschen oder ändern. Der Bildschirm-Editor kann so geschaltet werden, daß er Informationen vom Bildschirm annimmt, ohne auf die Eingabe des Benutzers warten zu müßen. Als erstes wird der Cursor am oberen Bildschirmrand positioniert und das Programm gestoppt. Hierdurch liest der Interpreter die auf dem Bildschirm stehenden Kommandos ein.

BEISPIEL 3 - *Sichern eines String/Array-Bereiches* - Besitzt ein String oder Array immer die gleiche Größe und den gleichen Inhalt, so kann ein großer Betrag des Speicherbereiches gespart werden, indem die entsprechende Information während des SAVEns gespeichert und die Initialisierung beim nächsten Programmstart gelöscht wird.

BEISPIEL 4 - *Sichern von BCD-Zahlen auf Diskette* - Immer wenn numerische Daten auf ein Gerät geschrieben werden, geschieht dies im ATASCII-Format. Das heißt: die Zahl 10 wird als ATASCII-Information 1, gefolgt von einer 0 ausgeschrieben. Dieses kann bei Records mit festgelegter Länge problematisch sein. Eine Möglichkeit, um dieses zu umgehen, wäre, die Zahlen in 6-Byte-BCD-Zahlen umzuformen und in Strings zu speichern. Letztere werden dann zum Lesen und Schreiben benutzt.

BEISPIEL 5 - *Player/Missile-Graphik mit Strings* - In diesem Beispiel wird eine Möglichkeit der schnellen Bewegung von Player/Missile-Bilddaten gezeigt. Man ändert den String/Array-Offset eines Strings, so daß er auf den Bereich für die Player/Missile-Bilddaten zeigt. Durch Schreiben von Daten in diesen String (Gleichsetzen) können die Player/Missile-Bilddaten mit einer Geschwindigkeit bewegt werden, die der von Maschinensprache gleichkommt.

```
10 REM Beispiel 1: String-Initialisierung
20 DIM A$(1000)
30 A$(1)="*": A$(1000)=”*”
40 A$(2)=A$

10 REM Beispiel 2: Löschen von Zeilen
20 GRAPHICS 0:POSITION 2,4
30 ? 70:? 80:? 90:? ”CONT”
40 POSITION 2,0
50 POKE 842,13:STOP
60 POKE 842,12
70 REM Diese Zeilen
80 REM werden
90 REM gelöscht

10 REM Beispiel 3: Sichern von Strings/Arrays
20 REM Beim ersten Lauf ein GOTO 50 ausführen
30 REM Beim zweiten Lauf Zeile 50 löschen
40 GOTO 110
50 DIM A$(10):A$="WWWWWWWWW"
60 STARP=PEEK(140)+PEEK(141)*256
70 STARP=STARP+10
80 HI=INT(STARP/256):LO=STARP-HI*256
90 POKE 140,LO:POKE 141,HI
100 SAVE"0:STRING":STOP
110 STARP=PEEK(140)+PEEK(141)*256
120 STARP=STARP-10
130 HI=INT(STARP/256):LO=STARP-HI*256
140 POKE 140,LO:POKE 142,LO:POKE 144,LO
150 POKE 141,HI:POKE 142,HI:POKE 145,HI
160 DIM A$(10)
170 A$(10)="W"
180 STOP

10 REM Beispiel 4: Sichern und Laden von
20 REM BCD-Zahlen über die Diskettenstation
30 DIM A(0),B$(6)
40 B$(6,6)=CHR$(32)
50 VTAB=PEEK(134)+PEEK(135)*256
60 POKE VTAB+1,0
70 OPEN #1,8,0,"D:TEST"
80 FOR C=1 TO 15:A(0)=C:? #1;B$:NEXT C
90 CLOSE #1
100 OPEN #1,4,0,"D:TEST"
110 FOR C=1 TO 15:INPUT #1,B$:? A(0):NEXT C
120 CLOSE #1:END

10 REM Beispiel 5: Player/Missile-Graphik über Strings
20 DIM A$(512),B$(20)
30 X=X+1:READ A:IF A<>-1 THEN B$(X,X)=CHR$(A):GOTO 30
40 DATA 0,255,129,129,129,129,129,129,129,129,255,0,-1
50 POKE 559,62:POKE 704,88
```

```
60 I=PEEK(106)-16:POKE 54279,I
70 POKE 53277,3:POKE 710,244
80 VTAB=PEEK(134)+PEEK(135)*256
90 ATAB=PEEK(140)+PEEK(141)*256
100 OFFS=I*256+1024-ATAB
110 HI=INT(OFFS/256):LO=OFFS-HI*256
130 Y=60:Z=100:V=1:H=l
140 A$(Y,Y+11)=B$:POKE 53248,Z
150 Y=Y+V:Z=Z+H
160 IF Y>213 OR Y<33 THEN V=-V
170 IF Z>206 OR Z<49 THEN H=-H
180 GOTO 140
```

Kapitel 8
Das Betriebssystem

EINFÜHRUNG IN DAS BETRIEBSSYSTEM

Dieses Kapitel ist eine einfache Einführung in das Betriebssystem (OS=Operating System) des ATARI Computer Systems (bei der XL-Serie lesen Sie bitte Anhang XI). Es enthält außerdem eine kurze Beschreibung einzelner OS-Teile. Diese werden in den folgenden Abschnitten beschrieben:

Input/Output-Untersystem
Vektoren der System-Routinen
Interrupt-Handler
Der Monitor
Die Systemvariablen
Die System-Timer
Die Fließkomma-Arithmetik

Das OS gestattet dem Programmierer den Zugriff auf die gesamte Hardware des Computers. So können die I/O-Funktionen der Hardware durch das I/O-Untersystem angesprochen werden. Dieses I/O-Untersystem besteht aus einer Reihe von System-Routinen, welche die für die Ein- und Ausgabe zuständige Hardware ansprechen.

Die System-Vektoren sind im gewissen Sinn die Knotenpunkte, mit denen das gesamte System „zusammengehalten" wird. Das OS benutzt diese Vektoren, um von einem „System-Programm" (z.B. BASIC, DUP oder STAR RAIDERS"TM") zu einem anderen zu wechseln. Jede System-Routine kann durch Springen zum zugehörigen Vektor aufgerufen werden. Dieses Springen geschieht zum größten Teil in regelmäßigen Abständen, um z.B. Timer zu setzen, die I/0-Routinen aufzurufen oder die Kontrolle an verschiedene System-Programme zu übergeben.

Die System-Routinen können auf zwei verschiedene Arten angesprochen werden. Die ROM-Vektoren sind Speicherstellen, welche JMP-Instruktionen zu Systemroutinen enthalten. Diese können nicht geändert werden. Die RAM-Vektoren sind Speicherstellen, die veränderliche Adressen für Systemroutinen enthalten. Die Speicherstellen für diese Routinen bleiben auch bei zukünftigen Versionen des Betriebssystems die gleichen.

Der Computer erzeugt aus verschiedenen Gründen Interrupts. Die am häufigsten auftretenden sind: Tastatur-, Break-, Vertical-Blank- und Serieller Bus-Interrrupt.

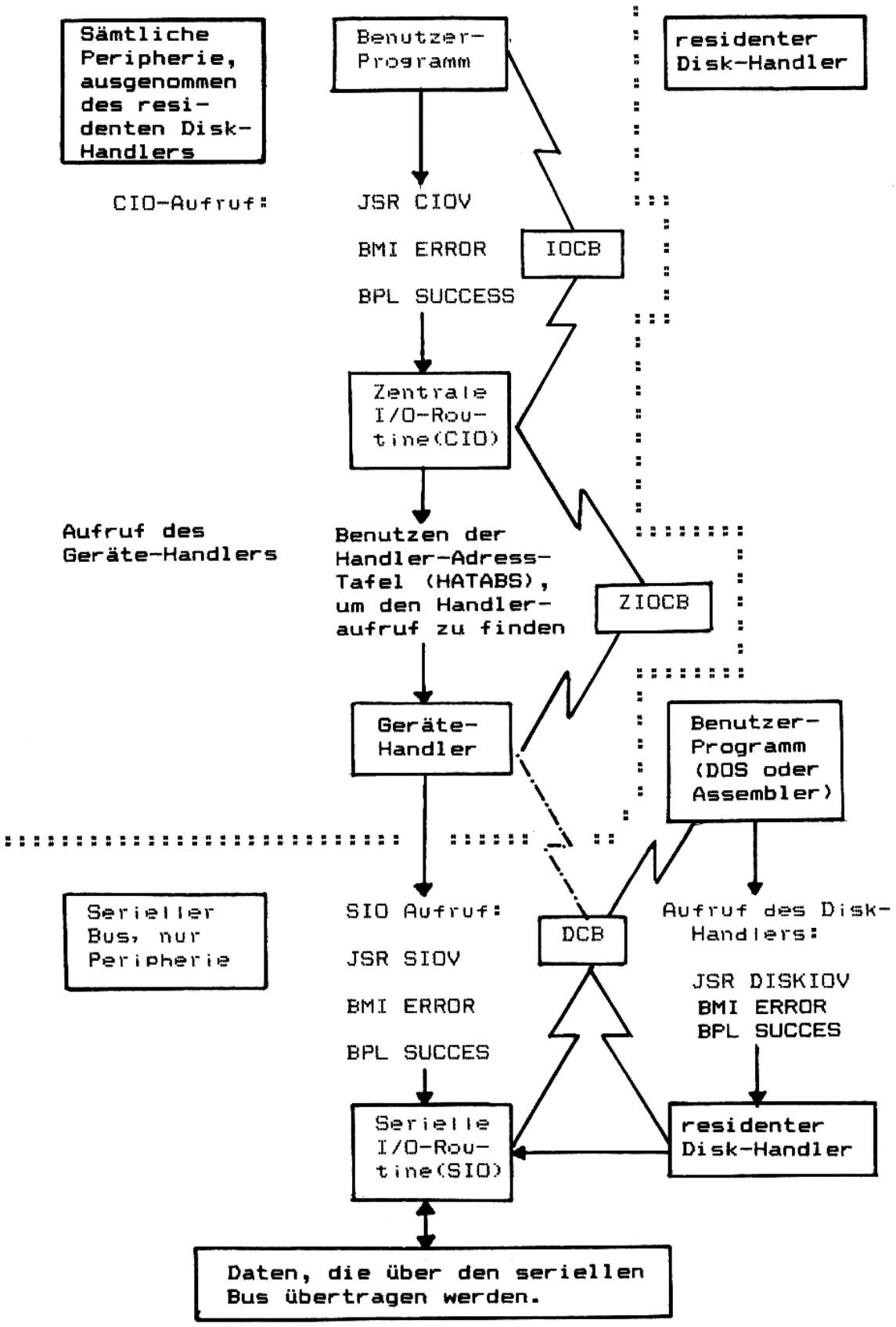
Der Monitor des OS ist eine System-Routine, die zum Initialisieren des Computers bei dessen Einschalten oder bei

Drücken der SYSTEM-RESET-Taste angesprochen wird. Der Monitor initialisiert das I/O-Untersystem, legt die Vektoren fest und wählt den Arbeitszustand, nachdem die Vorbereitung abgeschlossen wurde (BLACKBOARD-Modus, BOOT-Programm usw.).

Das OS nutzt weiterhin eine Vielzahl an Systemvariablen. Diese besteht aus verschiedenen System-Flags, I/0-Puffern und Bildschirm/Graphik-Registern. Der größte Teil dieser Variablen wird dabei vom I/O-Untersystem benutzt, aber sie enthalten auch Speicherstellen, die von anderen Teilen des OS benutzt werden. Der Programmierer ist durch Zugriff auf diese Systemvariablen in der Lage, sein Programm leistungsfähiger zu gestalten.

Es gibt zwar zwei Arten von Timern: System-Timer und Hardware-Timer. Die System-Timer werden von bestimmten Programmen als Software-Timer für verschiedene Zwecke benutzt. Die Hardware-Timer werden zur zeitlichen Festlegung sogenannter „Realzeit”-Aktionen, wie z.B. das Zeichnen von Scan-Lines, verwendet.

Die Fließkomma-Arithmetik besteht aus einer Reihe von mathematischen Routinen, die eine sog. „binär codierte Dezimal-Arithmetik” (BCD) benutzen. Es gibt sowohl Routinen für standardmäßige mathematische Funktionen (+, -, *, /), wie auch für die Umwandlung von ATASCII nach BCD bzw. umgekehrt. Im 8. Kapitel des OS-Benutzer-Manuals findet sich eine Beschreibung der Fließkomma-Arithmetik sowie deren Benutzung. In Anhang 5 dieses Buches ist ein Beispiel für die Benutzung dieser Routinen.


Sämtliche Peripherie, ausgenommen des residenten Disk-Handlers
Benutzer-Programm
residenter Disk-Handler
CIO-Aufruf:
JSR CIOV
BMI ERROR
BPL SUCCESS
IOCB
Zentrale I/O-Routine(CIO)
Aufruf des Geräte-Handlers
Benutzen der Handler-Adress-Tafel (HATABS), um den Handler-aufruf zu finden
ZIOCB
Geräte-Handler
Benutzer-Programm (DOS oder Assembler)
Serieller Bus, nur Peripherie
SIO Aufruf:
JSR SIOV
BMI ERROR
BPL SUCCES
DCB
Aufruf des Disk-Handlers:
JSR DISKIOV
BMI ERROR
BPL SUCCES
Serielle I/O-Routine(SIO)
residenter Disk-Handler
Daten, die über den seriellen Bus übertragen werden.

I/O-UNTERSYSTEM

Einführung

Das I/O-Untersystem gestattet dem Programm den bequemen Zugriff auf die von PIA, ANTIC, POKEY oder CTIA benutzten Hardwareregister. Die speziellen Chips kontrollieren die I/O-Geräte, wie z.B. die Tastatur, den Drucker oder die Diskettenstation. Der Benutzer übergibt einfach die Kontrolldaten an das I/O-Untersystem und dieses führt dann die erforderliche I/O-Funktion für das entsprechende Gerät durch.

Das I/O-Untersystem besteht aus zwei Elementen: den I/O-System-Routinen und den I/O-Kontrollblöcken. Die I/O-System-Routinen sind: die zentrale I/O-Routine (CIO), die Geräte-Handler (E:, P:, K: usw.) und die serielle I/O-Routine (SIO). Die I/O-Kontrollblöcke enthalten die Kontrolldaten, die an das I/O-Untersystem übergeben werden. Für den Benutzer bedeutet dieses Untersystem jedesmal ein ähnliches Kommando. Soll z.B. eine Zeile auf dem Drucker ausgegeben werden, so muß der Programmierer die Zeichen "P:" eingeben; soll sie über den Display Editor ausgegeben werden, dann sind es „E:”.

Um das I/O-Untersystem am effektivsten einsetzen zu können, muß man seine Struktur kennen. Die vorangegangene Abbildung zeigt die Zusammenhänge zwischen den I/O-System-Routinen und den I/O-Kontrollblöcken.

Kontrollblöcke des I/O-Untersytems

Es gibt drei verschiedene Arten von Kontrollblöcken:

Eingabe/Ausgabe-Kontrollblock (IOCB)
Zero-Page I/O-Kontrollblock (ZIOCB)
Geräte-Kontrollblock (DCB=Device Control Block)

Die Kontrollblöcke des I/O-Untersystems werden zum Austauschen von Information über die auszuführende Information verwendet. Die Kontrollblöcke liefern den I/O-System-Routinen die zur Ausführung einer I/O-Funktion erforderlichen Daten. In Abschnitt 5 des OS-Manuals finden sich weitere detaillierte Information zur Struktur dieser drei Kontrollblockarten.

IOCB-Überblick

Aufruf	ICHID	ICDNO	ICCOM	ICSTA	ICBAL	ICBAH	ICPTL	ICPTH	ICBLL	ICBLH	ICAX1	ICAX2
Open File - Lesen	x	x	3	N1	$80	06	X	X	X	X	4	0
Open File - Schreiben	x	x	3	N1	$80	06	X	X	X	X	8	N2
GET BYTES	x	x	7	N1	00	06	X	X	$80	00	X	X
PUT BYTES	x	x	$B	N1	00	06	X	X	$80	00	X	X
GET RECORD	x	x	5	N1	00	06	X	X	$80	00	X	X
PUT RECORD	x	x	9	N1	00	06	X	X	$80	00	X	X
CLOSE FILES	x	x	$C	N1	X	X	X	X	X	X	X	X
STATUS	x	x	$D	N1	X	X	X	X	X	X	X	X

Abbildung 8.2

N1 = Hier wird der Status des I/O-Befehls gespeichert. Er befindet sich beim Rücksprung vom CIO im Y-Register.

N2 = Die AUX-Bytes der IOBs werden von einigen Handlern zum Anzeigen von speziellen Modi benutzt.

x = Wird nicht beachtet, der augenblickliche Wert wird aber nicht geändert.

Anmerkung: Die obere IOCB-Definition setzt folgendes voraus:

```
*=$600
IOBUFF .RES 80                  I/O-Buffer des Benutzers.
FILE .BYTE "D:HPROG.BAS" Filename vom Benutzer.
```

DCB-Überblick

Funktion	Name	Diskettenstation 810					Drucker 820 schrbn.
		Speicher-stelle	Lese-Sektor	Schreib-Sektor m.Verify	Schreib-Sektor o.Verify	FORMAT Disk	
Serieller Bus-Id.	DDEVIC	Ä$0300ü	$30	$30	$30	$30	$40
Geräte-Nummer	DUNIT	Ä$0301ü	1-4	1-4	1-4	1-4	1
Kommando-Byte	DCOMND	Ä$0302ü	$52	$57	$50	$21	$57
Status	DSTATS	Ä$0303ü	$40	$80	$80	$40	$80
Puffer-adresse	DBUFLO DBUFHI	Ä$0304ü Ä$0305ü	B B	B B	B B	B B	B B
Geräte-fehler-zeit	DTIMLO	Ä$0306ü	$30	$30	$31	$130	5
Puffer-länge	DBYTLO DBYTHI DAUX1 DAUX2	Ä$0308ü Ä$0309ü Ä$030Aü Ä$030Bü	$80 $00 2* 2*	$80 $80 2* 2*	$80/0 $00/1 -2* -2*	- - - -	$40 $88 1* 1*

Abbildung 8.3

1* = Dieses Byte bestimmt den Druckermodus (siehe 820-Manual)
2* = DAUX1 & DAUX2 legen den Sektor für READ, WRITE (PUT oder überprüfung) fest.
B = Die Adresse wird vom Benutzer bestimmt.
- = Wird nicht beachtet.

Die 8 IOCBs des OS werden zum Informationsaustausch zwischen dem Benutzerprogramm und dem CIO verwendet. Abbildung 8.2 zeigt den Inhalt eines IOCBs bei einigen normalen I/O-Funktionen. Die ICCBs sind:

Name	Speicherstelle, Länge
IOCB0	Ä$340,16ü
IOCB1	Ä$350,16ü
IOCB2	Ä$360,16ü
IOCB3	Ä$370,16ü
IOCB4	Ä$380,16ü
IOCB5	Ä$390,16ü
ICCN6	Ä$3A0,16ü
IOCB7	Ä$3B0,16ü

Die zweite Art von Kontrollblöcken, der ZIOCB Ä$0020,16ü, wird zum Austauschen von Kontrolldaten zwischen den Geräte-Handlern und dem CIO benutzt. Bei Aufruf benutzt die CIO den im X-Register enthaltenen Wert als Index, der auf die Startadresse des IOCBs (einer von 8), welcher benutzt werden soll, zeigt. Die CIO übergibt die Kontrolldaten vom gewählten IOCB an den ZIOCB ist von geringerem Interesse, solange der Benutzer keinen neuen Geräte-Handler schreibt und einen alten durch diesen ersetzt. Weitere Informationen über den ZIOCB siehe OS-Benutzer-Manual.

Die Geräte-Handler laden die Kontrollinformation in den DCB Ä$0300,16ü. Die SIO holt die DCB-Information und die Statuswerte im DCB für anschließende Verwendung durch den Geräte-Handler. Der DCB und die SIO werden nur von Geräte-Handlern angesprochen, die den seriellen Bus benötigen. Abschnitt 9 des OS-Manuals enthält eine detailierte Beschreibung des DCBs. Abbildung 8.3 listet einige normale Funktionen (I/0) und die entsprechenden Inhalte des DCBs.

Der residente Disk-Handler richtet sich nicht nach der regulären Aufrufsfolge (Benutzer-CIO-Handler-SIO). Statt dessen wird der DCB von Benutzer verwendet, um direkt mit dem residenten Disk-Handler zu kommunizieren. Kapitel 9 dieses Buches enthält nähere Informationen zum residenten Disk-Handler.

ZENTRALE I/O-SYSTEM-ROUTINE - CIO

Die Hauptaufgabe der CIO ist das Übertragen von Kontrolldaten an den richtigen Geräte-Handler. Die CIO ist außerdem eine Ausgangsroutine für alle Geräte-Handler, sowie der normale Einsprungpunkt für die meisten I/O-Funktionen des OS. Die CIO wird z.B. aufgerufen, wenn der BASIC-Interpreter eine OPEN-Anweisung ausführt.

OPEN	Gerät/Datei öffnen
CLOSE	Gerät/Datei schließen
GET CHARS	Lesen von n Zeichen
READ RECORD	Lesen des nächsten Satzes
PUT CHARS	Schreiben von n Zeichen
WRITE RECORD	Schreiben des nächsten Satzes
STATUS	Abfragen des Geräte-Zustandes
SPECIAL	Handler-spezifisch (z.B. NOTE für FMS)

Der Programmierer kann natürlich eigene CIO-Aufrufe erstellen. Die Aufruffolge für die CIO lautet wie folgt:

```
                ; Der Benutzer hat IOCB-Parameter gesetzt
LDX #IOCBNUM    ; Setzen des IOCB-Indexes (IOCB 16)
JSR CIOV        ; System-Routinen-Vektor zu CIO
BMI ERROR       ; Wenn kein Fehler auftaucht, wird
```

```
                    ; das Programm fortgesetzt, andern-
                    ; falls befindet sich der Fehler-Code
                    ; im Y-Register.
```

Wie im oberen Aufruf gezeigt, wird einer der IOCBs benutzt, um die Kontrolldaten zur CIO zu bringen. Dabei kann irgendeiner der 8 vorhandenen IOCBs verwendet werden. Die CIO benötigt den IOCB-Index im X-Register. Beim Rücksprung werden die Status-Bits des 6502-Prozessors gesetzt, um den Erfolg oder Fehler der I/O-Operation anzuzeigen. Ist das N-Bit gesetzt ("1"), dann tauchte ein Fehler bei der Ausführung der I/O-Operation auf und der entsprechende Fehler-Code wird im Y-Register übergeben. Der Fehler/Erfolgs-Wert wird außerdem im IOCB-Byte ICSTA (siehe IOCB-Definition) gespeichert. Im OS-Benutzer-Manual befindet sich ein Beispielprogramm, das die CIO anspringt, um ein Disk-File zu öffnen, einige RECORDs einzulesen und das File zu schließen (OPEN, READ und CLOSE).

Die CIO übergibt die I/O-Kontrolldaten, indem der IOCB-Index im X-Register zum Übertragen der IOCB-Inhalte zum ZIOCB benutzt wird. Die CIO berechnet dann den Einsprungspunkt für den Geräte-Handler und springt die entsprechende Handler-Routine an. Anhang VI zeigt ein Flußdiagramm der CIO-System-Routine.

Der Eingangspunkt eines Device-Handlers wird von der CIO indirekt berechnet. Während eines OPEN-Aufrufes bekommt die CIO die Geräte-Spezifikation für die bereit zu machende Datei. Ist das zu öffnende Gerät z.B. ein Drucker, so würde die Geräte-Spezifikation ”P:” lauten.

```
                01 ; Adresstafel für Handler
E430            02 PRINTV = $E430
E440            03 CASETV = $E440
E400            04 EDITRV = $E400
E410            05 SCRENV = $E410
E420            06 KEYBDV = $E420
                07 ;
0000            08       *= $031A
                09
                10 HATABS
031A 50         20        .BYTE ´P´      Drucker
031B 30 E4      30        .WORD PRINTV   Einsprungpunkt-Tafel
031D 43         40        .BYTE ´C´"     Cassettenstation
031lE 40 E4     50        .WORD CASETV   Einsprungpunkt-Tafel
0320 45         60        .BYTE ´E´      Bildschirm-Editor
0321 00 E4      70        .WORD EDITRV   Einsprungpunkt-Tafel
0323 53         80        .BYTE ´S´      Bildschirm-Handler
0324 10 E4      90        .WORD SCRENV   Einsprungpunkt-Tafel
0326 4B         0100      .BYTE ´K´      Tastatur
0327 20 E4      0110      .WORD KEYBDV   Einsprungpunkt-Tafel
0329 00         0120      .BYTE 0        Freier Eintrag 1
```

```
032A 00 00      0130      .BYTE 0,0       (DOS)
032C 00         0140      .BYTE 0         Freier Eintrag 2
032D 00 00      0150      .BYTE 0,0       (850-Modul)
032F 00         0160      .BYTE 0         Freier Eintrag 3
0330 00 00      0170      .BYTE 0,0
0332 00         0180      .BYTE 0         Freier Eintrag 4
0333 00 00      0190      .BYTE 0,0
0335 00         0200      .BYTE 0         Freier Eintrag 5
0336 00 00      0210      .BYTE 0,0
0338 00         0220      .BYTE 0         Freier Eintrag 6
0339 00 00      0230      .BYTE 0,0
033B 00         0240      .BYTE 0         Freier Eintrag 7
```

Abbildung 8.4a:
Adresstafel für Handler

```
               *= $PRINTV

E430 9E EE      .WORD     PHOPEN-1  Gerät öffnen (OPEN)
E432 DB EE      .WORD     PHCLOS-1  Gerät schließen (CLOSE)
E434 9D EE      .WORD     BADST-1   Gerät lesen - nicht
                                    vorhanden
E436 A6 EE      .WORD     PHWRIT-1  Gerät schreiben (WRITE)
E438 80 EE      .WORD     PHSTAT-1  Gerätezustand (STATUS)
E33A 9D EE      .WORD     BADST-1   Special - nicht vorhanden

E34C 4C 78 EE   JMP       PHINIT    Geräte Initialisierung
```

Abbildung 8.4b:
Einsprungpunkt-Tafel für Drucker-Handler

Die CIO benutzt eine mit HATABS (Abbildung 8.4a) bezeichnete Tafel, um den Handler-Einsprungpunkt indirekt zu berechnen. Diese Tafel verwendet eine Geräte-Spezifikation als Schlüssel zum Auffinden der Adresse des entsprechenden Handler-Einsprungpunktes. Die Geräte-Spezifikation wird benutzt, um den HATABS-Eingang des zu eröffnenden Gerätes zu finden. Die CIO beginnt mit der Suche nach dem ersten Eintrag mit der entsprechenden Geräte-Spezifikation am Ende von HATABS (in diesem Fall ´P:´, der erste Eintrag in die HATABS-Tafel). Die mit der Geräte-Spezifikation bestimmte Adresse ist ein Zeiger zu einer Liste von Handler-Einsprungpunkten (die für den Drucker stehen in Abbildung 8.4b).

Die CIO verwendet ICCOM, das IOCB-Kommando-Byte, um den Einsprungpunkt des anzusprechenden Handlers zu finden. Die Einsprungpunkt-Tafeln für alle Geräte-Handler sind im OS-Listing aufgeführt. Die Position in allen Einsprungpunkt-Tafeln der Handler haben die gleiche Bedeutung. So ist z.B. die erste Position in allen Tafeln der Vektor zur OPEN-Routine für den Geräte-Handler.

HATABS liegt im RAM ab der Speicherstelle $031A und enthält Platz für 14 Einträge. Beim Einschalten und beim Drücken der SYSTEM-RESET-Taste werden die Inhalte von HATABS entsprechend Abbildung 8.4 initialisiert. HATABS besitzt zu diesem Zeitpunkt Einträge für alle Handler mit Ausnahme des Disk-Handlers. Der Disk-Handler wird niemals über HATABS aufgerufen (siehe Kapitel 9 dieses Buches).

Durch den Programmierer oder das OS können weitere Einträge nach HATABS gebracht werden. Das OS kann neue Einträge als einen Teil der Einschalt- oder SYSTEM-RESET-Funktion anfügen. Jeder neue Eintrag würde bei Speicherstelle $0329 beginnen. Anfügungen an HATABS könnten z.B. Eingänge für die Diskettenstation (der File-Manager <FMSI> oder für das 850-Modul sein.

Man kann die flexible Natur HATABS' ausnutzen und das OS um einige neue Fähigkeiten erweitern. Ein Beispiel (Abbildung 8.5) zeigt einen Null-Handler. Ein Null-Handler ist genau das, was sein Name sagt: er führt KEINE FUNKTION aus! Ein Null-Handler kann zum Ausarbeiten von Programmen benutzt werden, welche Geräte verwenden, die eine langsame Zugriffsgeschwindigkeit besitzen. Anstatt 50.000 Disketten-Zugriffe abzuwarten, bis ein Fehler gefunden wird, muß die Ausgabe lediglich über den Null-Handler durchgeführt werden.

Eine andere Möglichkeit in Verbindung mit HATABS besteht darin, die Funktion eines bestehenden Eintrags zu ändern. Möchte der Benutzer z.B. einen Drucker an den Computer anschließen, welcher einige spezielle Fähigkeiten besitzt, die mit dem normalen Drucker-Handler nicht angesprochen werden können, so kann dieses durch Ändern der HATABS Einsprungpunkt-Tafel behoben werden. Alle ”P:" I/O-Anweisungen könnten mit einem neuen Drucker-Handler bearbeitet werden. Anhang VII zeigt ein Beispiel für einen Drucker-Handler, der die vorhandenen Kontroller-Eingänge zum schnellen Übertragen von zu druckenden Daten benutzt.

```
0000           10       *= $600
031A           20 HATABS = $031A
               30 START
0600 A0 00     40        LDY #0
               50 LOOP
0602 B9 1A 03  60        LDA HATABS,Y
0605 C9 00     70        CMP #0          Freier Eintrag?
0607 F0 09     80        BEQ FOUND
0609 C8        90        INY
060A C8        0100      INY
06OB C8        0110      INY             Zeigt zum nächsten
060C C0 22     0120      CPY #34         HATABS-Eintrag am Ende
06OE D0 F2     0130      BNE LOOP        von HATABS?? - Nein...
0610 38        0140      SEC             Ja, übertrage
                                         ERROR-Status
```

```
0611 60        0150      RTS             zum aufrufenden
                                         Programm
               0160 ;
               0170 FOUND
0612 A9 4E     0180      LDA #´N         Device-Name
0614 99 1A 03  0190      STA HATABS,Y
0617 C8        0200      INY
0618 A9 24     0210      LDA #NULLTAB&255
061A 99 1A 03  0220      STA HATABS,Y    Null-Handler Ein-
061D C8        0230      INY             sprung-Tafel
061E A9 06     0240      LDA #NULLTAB/256
0620 99 1A 03  0250      STA HATABS,Y
0623 60        0260      RTS
               0270 ;
               0280 NULLTAB
0624 32 06     0290      .WORD RTHAND-1 OPEN
0626 32 06     0300      .WORD RTHAND-1 CLOSE
0628 32 06     0310      .WORD NOFUNC-1 READ
062A 32 06     0320      .WORD RTHAND-1 WRITE
062C 32 06     0330      .WORD RTHAND-1 STATUS
062E 34 06     0340      .WORD NOFUNC-1 SPECIAL
0630 4C 33 06  0350      JMP RTHAND     Initialisierung
               0360 ;
               0370 RTHAND
0633 A0 01     0380      LDY #1         Erfolgreiche I/O-Fkt.
               0390 NOFUNC             Fkt. nicht vorhanden
               0400 ;
0635 60        0410      RTS
```

Abbildung 8.5:
Null-Handler

CIO-AUFRUF ÜBER BASIC

Die meisten CIO-Funktionen (OPEN, CLOSE usw.) sind durch BASIC-Befehle aufrufbar. Ein Teil CIO-Funktionen allerdings läßt sich vom BASIC-Interpreter nicht ansprechen. So besitzt der Interpreter nicht die Fähigkeit, I/0 mit mehr als einem Byte durchzuführen (GETCHARACTER und PUTCHARACTER), außer als Record.

Die Möglichkeit einen ganzen Puffer von Zeichen auf einmal ein- und auszugeben ist sehr elegant. So könnte eine Assembler-Routine z.B. direkt von einer Disk-Datei in den Speicher geladen werden. Bei einem BASIC-Programm wird die Assembler-Routine normalerweise über einen String eingelesen und dann mit der USR(ADR($))-Funktion aufgerufen. Da die Adresse eines BASIC-Strings sich während der Programmänderung verschieben kann, muß die Routine unabhängig von Speicherstellen sein. Dieses Bedeutet, daß Speicherzugriffe innerhalb des Strings nicht arbeiten.

Die Unterroutine in Abbildung 8.6 umgeht die Benutzung von Strings. So muß die Assembler-Routine nicht unbedingt unabhängig von Speicherstellen sein. Die Kontrolldaten werden in einem IOCB gesetzt, d.h. das Assembler-Programm wird direkt in die Speicherstellen gelesen, wo es ursprünglich assembliert wurde. Die BASIC-Unterroutine in Abbildung 8.6 kann auch dazu benutzt werden, Daten direkt vom Speicher auszugeben, wobei der Benutzer die Speicherstelle und die Pufferlänge festlegt.

DIE GERÄTE-HANDLER

Die Geräte-Handler können in zwei Gruppen aufgeteilt werden: die residenten und die nicht-residenten Handler. Die residenten Handler befinden sich im OS-ROM und können durch die CIO aufgerufen werden, sofern sie einen Eintrag in HATABS besitzen. Die residenten Handler sind:

(E:) Bildschirm-Editor
(S:) Screen (Bildschirm)
(K:) Keyboard (Tastatur)
(P:) Printer (Drucker)
(C:) Cassettenstation

```
30 REM Dieses Programm laedt PAGE 6 von der Datei "D:TEST"
100 DIM FILE$(20),CIO$(7)
101 CIO$="hhh*LVd": REM ”*” & "D" invers eingeben
105 REM CIO$ = PLA,PLA,PLA,TAX,JMP $E456 (=CIOV)
110 FILE$="D:TEST":REM Dateiname
120 CMD=7:STADR=1536:GOSUB 30000
130 IF ERROR<>1 THEN ? "FEHLER -";ERROR;"in Zeile”;
140 ? PEEK(186)+PEEK(187)*256,PEEK(195)
200 END
300 REM -       CIOVORBEREITUNG
310 REM

30000 REM Routine von M. Ekberg fuer ATARI 3.9.1980
30001 REM
30002 REM Diese Routine laedt oder sichert Dateien von
30003 REM BASIC, indem ein IOCB gesetzt und CIO direkt
30004 REM aufgerufen wird.
30005 REM
30006 REM Eingang CMD=7 bedeutet: laden
30007 REM -       CMD=11 bedeutet: sichern
30008 REM -       STADR= Die Adresse zum Laden o. Sichern
30009 REM -       BYTES= Die Anzahl der Bytes
30010 REM -       IOCB= Der zu benutzende IOCB
30011 REM -       FILE$= Ausgabe-Dateiname
30012 REM
30013 REM Abschluss mit ERROR=1 bedeutet erfolgreiche
30014 REM Ausfuehrung
30015 REM ERROR<>l bedeutet: Fehlerzustand
```

```
30016 REM
30017 REM
30018 REM *** IOCB-FESTLEGUNGEN ***
30019 REM
30024 IOCBX=IOCB*16:ICCOM=834+IOCBX:ICSTA=835+IOCBX
30026 ICBAL=836+IOCBX:ICBAH=837+IOCBX
30028 ICBLL=840+IOCBX:ICBLH=841+IOCBX
30030 AUX1=4:IF CMD=11 THEN AUX1=8
30035 TRAP 30900:OPEN #IOCB,AUX1,0,FILE$:IOCBX=IOCB*16
30040 TEMP=STADR:GOSUB 30500
30090 POKE ICBAL,LOW:POKE ICBAH,HIGH
30100 TEMP=BYTES:GOSUB 30500
30130 POKE ICBLL,LOW:POKE ICBLH,HIGH
30140 POKE ICCOM,CMD:ERROR=USR(ADR(CIO$),IOBX)
30150 ERROR=PEEK(ICSTA):RETURN
30200 REM
30300 REM *** Diese Routine errechnet hoeher und
30305 REM *** niederwertiges Byte einer 16-Bit-Zahl
30400 REM
30500 HIGH=INT(TEMP/256):LOW=INT(TEMP-HIGH*256):RETURN
30550 REM
30600 REM *** Hierher wird bei Fehler gesprungen
30900 ERROR=PEEK(195)
30920 CLOSE #IOCB:RETURN
```

Abbildung 8.6
Direkter CIO-Aufruf über BASIC

Die nicht-residenten Handler befinden sich nicht im OS-ROM. Sie können dem OS beim Einschalten oder SYSTEM-RESET hinzugefügt werden. Dieses kann aber auch während der Programmausführung geschehen (siehe Abbildung 8.5).

Die Geräte-Handler benutzen I/O-Kontrolldaten, die über die CIO zum ZIOCB gebracht werden. Die Daten im ZIOCB werden beim Ausführen von I/O-Funktionen, wie z.B. OPEN, CLOSE, PUT oder SET verwendet. Nicht alle Geräte-Handler sprechen auf alle I/O-Kommandos an (das PUT-Kommando zur Tastatur würde eine Fehler-146 (Funktion nicht implemetiert) -Meldung erzeugen). Abschnitt 5 des OS-Benutzer-Manuals enthält eine Liste der von den einzelnen Handlern ausführbaren I/O-Funktionen.

SERIELLE I/O-SYSTEM-ROUTINE - SIO

SIO und die Geräte-Handler

Die SIO bearbeitet die Kommunikation zwischen den seriellen Geräte-Handler im Computer und den Devices des seriellen Busses. Dieses geschieht über den Device-Kontrollblock (DCB). Die SIO benutzt die I/0-Kontrolldaten im DCB zum Senden und Empfangen von Daten über den seriellen Bus. Die Aufrufsequenz sieht wie folgt aus:

```
                ; der DCB wurde zur Durchführung der
                ; Operation erstellt.
      JSR SIOV  ; Systemvektor zur SIO
      BMI ERROR ; gesetztes N-BIT zeigt Fehler bei der
                ; Ausführung der I/O-Operation
```

Der DCB enthält Kontrollinformationen für die SIO und muß vor deren Aufruf aufgestellt werden. Abbildung 8.3 zeigt die Inhalte des DCBs für einige I/O-Operationen.

Es ist erforderlich, die Struktur des DCBs zu verstehen, um Kommandos an die SIO senden zu können. Abbildung 8.7 zeigt eine einfache Assembler-Routine, um eine Zeile (durch Erstellen eines DCBs und Aufrufen der SIO) über den Drucker ausgeben zu lassen.

```
0000            05        *= $3000 Beliebiger Startpunkt
                10 ; Diese Routine druckt eine Zeile auf dem
                15 ; Drucker, indem die SIO aufgerufen wird.
E459            20 SIOV         = $E459   SIO-Vektor
009B            30 CR           = $9B     EOL
0040            40 PRNTID       = $40     Drucker-ID f.ser. Bus
004E            50 MODE         = $4E     Normaler Modus
001C            60 PTIMOT       = $001C   Timeout-Speicherstelle
0300            70 DDEVIC       = $300    Geraete-ID f.ser. Bus
0301            80 DUNIT        = $301    Nummer d.ser. Einheit
0302            90 DCOMND       = $302    SIO-Kommando
0303            0100 DSTATS     = $303    SIO-Datenrichtung
0304            0110 DBUFLO     = $304    Niederw.Adr.d.Puffers
0305            0120 DBUFHI     = $305    Hoeherw.Adr.d.Puffers
0306            0130 DTIMLO     = $306    SIO-Timeout
0307            0140 DTIMHI     = $307
0308            0150 DBYTLO     = $308    Pufferlaenge
0309            0160 DBYTHI     = $309
030A            0170 DAUX1      = $30A    Hilfsbyte:Druckermodus
030B            0180 DAUX2      = $30B    Hilfsbyte:nicht
                                                benutzt
                0190 ;
3000 42 45 49   0200 MESS       .BYTE     "BEISPIEL12",CR
3001 53 50 49
3005 45 4C 31
3009 32 9B
                0210 ;
300B A9 40      0220 LDA #PRNTID          Setzen des Bus IDs
300D 8D 00 03   0230 STA DDEVIC
3010 A9 01      0240 LDA #1               Setzen d.Einheits-Nr.
3012 8D 01 03   0250 STA DUNIT
3015 A9 4E      0260 LDA #MODE            Norm. Drucker-Modus
3017 8D 0A 03   0270 STA DAUX1
301A A9 01      0275 LDA #1
301C 8D 0B 03   0280 STA DAUX2            Nichtbenutzt
```

```
301F 8D 07 03  0290 STA DTIMHI          Timeout nach 256 Sek.
3022 A5 1C     0300 LDA PTIMOT          Setzen des SIO-Time-
3024 8D 06 03  0310 STA DTIMLO          outs fuer Drucker
3027 A9 00     0320 LDA #MESS&255
3029 8D 04 03  0330 STA DBUFLO          MESS als Puffer
302C A9 30     0340 LDA #MESS/256       setzen
302E 8D 05 03  0350 STA DBUFHI
3031 A9 80     0360 LDA #$80            Setzen der SIO-Daten-
3033 8D 03 03  0370 STA DSTATS          richtung f. Peripherie
3036 A9 57     0380 LDA #´W             SIO-Kommando schreiben
3038 8D 02 03  0390 STA DCOMND
303B 20 59 E4  0410 JSR SIOV            SIO aufrufen
303E 30 01     0420 BMI ERROR
3040 00        0420 GOOD BRK
3041 00        0440 ERROR BRK
```

Abbildung 8.7.
Aufrufen der SIO zum Drucken einer Zeile auf dem Drucker

SIO-INTERRUPTS

Die SIO benutzt drei IRQ-Interrupts,um über den seriellen Bus mit den Geräten zu kommunizieren. Die Interruts sind:

IRQ	Speicherstelle, Länge	Funktion
VSERIR	($020A,2)	Serielle Ausgabe abgeschlossen
VSEROR	($020C,2)	Serielle Ausgabe benötigt
VSEROC	($020E,2)	Übertragung beendet

Die gesamte Programmausführung wird gestoppt, wenn die SIO den seriellen BUS für die Datenübertragung benutzt. Für diese Ausgabe übergibt die SIO ein Byte an das „serielle Ausgabe-Schieberegister" im POKEY-Chip (SEROUT), welches dann auf den seriellen Bus gesendet wird. Danach wird in eine Warteschleife gesprungen, bis ein IRQ (VSEROR) vom POKEY-Chip empfangen wird und der SIO mitteilt, daß SEROUT für ein anderes Byte frei ist. Der IRQ bewirkt einen Sprung zur SIO-Routine für das Laden von SEROUT mit einem Byte. Diese Schleife wird solange ausgeführt, bis alle in der DCB-Länge festgelegten Bytes gesendet wurden. VSEROR ist ein IRQ, der anzeigt, daß die Übertragung von Bytes über den Bus abgeschlossen ist.

Die SIO-Operation für die Eingabe läuft ähnlich ab. Der POKEY-Chip informiert die SIO, daß ein Byte im seriellen Eingabe-Schieberegister empfangen wurde (SERIN), indem ein IRQ (VSERIN) erzeugt wird. Die SIO speichert das Byte in einem Puffer und geht daraufhin in eine Endlosschleife, die durch den nächsten IRQ d.h. das nächste Byte unterbrochen wird.

Wie man aus den oberen Absätzen ersehen kann, verliert die SIO einige Zeit, während auf das Signal vom POKEY-Chip gewartet wird. Da die Vektoren für die 3 SIO-IRQ-Service-Routinen RAM-Vektoren sind, können sie auch durch andere Handler benutzt werden, um die I/O-Operationen des Systems zu verbessern.

Dieses ist genau der Punkt, der dem 850-Modul ermöglicht, einen eigenständigen I/O durchzuführen. Der Handler des 850-Moduls überschreibt die SIO-IRQ-Vektoren mit den moduleigenen IRQ-Routinen. Dieses geschieht beim eigenständigen I/O, wodurch der 850-Modul-Handler dann Kommandos über den Bus senden kann. Der Modul-Handler gestattet dann, daß ein Programm weiterläuft, während das 850-Modul ein I/O-Kommando ausführt.

INTERRUPTS
Einführung

Im vorangegangenen Abschnitt wurde gezeigt, wie die SIO Interrupts zum Koordinieren von Datenübertragungen über den seriellen Bus benutzt. Der Computer besitzt außerdem noch andere Interrupt-Arten, die ein Programm um leistungsstarke Möglichkeiten erweitern können. Es gibt 2 verschiedene Interrupt-Typen: den maskierbaren Interrupt (IRQ) und den nicht-maskierbaren Interrupt (NMI). Die Interrupts das ATARI ´TM´ Personal Computers sind:

Name (Vektor)	**Typ**	**Funktion**	**Benutzt von****
DISPLAY LIEST			
(VDLIST)	NMI	Graphik-Timing	B
SYSTEM RESET			
(keinen)	NMI	System-Initials.	C
VERTIKAL BLANK			
(VVBLKI,VVBLKD)	NMI	Graphik-Display	C,B
SERIELLE AUSGABE BEENDET			
(VSERIN)	IRQ	Ser. Eingabe	C
SERIELLE EINGABE ABGESCHLOSSEN			
(VSEROR)	IRQ	Ser. Eingabe	C
SERIELLE AUSGABE ABGESCHLOSSEN			
(VSEROC)	IRQ	Ser. Ausgabe	C
POKEY-TIMER 1			
(VTIMR1)	IRQ	Hardware-Timer	B
POKEY-TIMER 2			
(VTIMR2)	IRQ	Hardware-Timer	B
***POKEY-TIMER 4**			
(VTIMR4)	IRQ	Hardware-Timer	B
TASTATUR			
(VKEYBD)	IRQ	Tastendruck	C
BREAK-TASTE			

(keinen)	IRQ	(BREAK)-Tastendruck	C
SERIELLER BUS ABLAUF			
(VPRCED)	IRQ	Geräte-Ablauf	N
SERIELLER BUS INTERRUPT			
(VINTER)	IRQ	Geräte-Interrupt	N

* Dieser IRQ wird im augenblicklichen im Computer vorhandenen OS nicht vektorisiert.

** B=Benutzer; C=Computer; N=Nicht benutzt

Ist der Leser nicht mit Interrupts vertraut, dann sei er auf Abschnitt 6 des OS-Manuals verwiesen, der Information zu diesem Thema enthält. Das Arbeiten mit Interrupts kann kompliziert sein. Wird z.B. der Tastatur-IRQ-Interrupt ausgeschaltet, dann ignoriert der Computer sämtliche Tastendrücke mit Ausnahme der (BREAK)-Taste. Obwohl dieses manchmal von Nutzen sein kann, macht es die Fehlersuche in einem Programm ein wenig schwierig.

DER IRQ-INTERRUPT-HANDLER

Das OS besitzt einen IRQ-Interrupt-Handler, der die verschiedenen IRQs bearbeitet. Dieser Handler hat RAM-Vektoren für alle IRQs, ausgenommen den (BREAK)-Tasten-IRQ. Die IRQ-Vektoren werden während des Einschaltens bzw. beim SYSTEM-RESET auf ihre Werte gesetzt. Die Speicherstellen der IRQ-RAM-Vektoren werden in Abbildung 8.9b gezeigt.

Die IRQ-Vektoren sind:

VIMIRQ Direkter IRQ-Vaktor. Alle IRQs laufen über diese Speicherstelle. VIMIRQ zeigt normalerweise auf den IRQ-Handler. Dem Benutzer ist es aber möglich, diesen Vektor zu ändern, um eigene Routinen für die Interrupts abarbeiten zu lassen.

VSEROR IRQ-Vektor POKEYs zur Beendigung der seriellen Ausgabe (siehe Abschnitt 8.2).

VSERIN IRQ-Vaktor POKEYs zur Beendigung der seriellen Ausgabe (siehe Abschnitt 8.2).

VSEROC IRG-Vaktor POCKEYs zum Abschluss der seriellen Ausgabe (siehe Abschnitt 8.2).

VTIMR1 Timer 1-Vektor POKEYs (siehe Abschnitt 8.7 für Information über die POKEY-Timer).

VTIMR2 Timer 2-Vektor POKEYs.

VTIMR4 Timer 4-Vektor POKEYs.

VKEYBD Tastatur-IRQ-Vektor. Das Drücken einer Taste, ausgenommen der (BREAK)-Taste, verursacht diesen IRQ. Der VKEYBD-Vektor kann zum Vorarbeiten des Tastencodes benutzt werden, bevor er durch das OS in ATASCII-Code umgewandelt wird. Dieser Vektor zeigt normalerweise auf die Tastatur-Routine des OS.

VPRECD IRQ-Vektor für den Peripherie-Ablauf. Diese Signalleitung ist über den seriellen Bus für Peripherie-Geräte zugänglich. Dieser IRQ wird zur Zeit noch nicht benutzt und zeigt daher normalerweise auf einen RTI-Befehl.

VINTER IRQ-Vektor für Peripherie-Interrupt. Diese Signalleitung ist über den seriellen Bus für den Computer zugänglich. VINTER zeigt normalerweise auf ein RTI-Kommando.

VBREAK IRQ-Vektor für 6502-BRK-Anweisungen. Immer, wenn ein $00 Opcode (softwaremäßige BREAK-Anweisung) ausgeführt wird, taucht dieser Interrupt auf. Dieser Vektor kann zum Setzen von Unterbrechungs-Punkten innerhalb eines Maschinensprache-Debuggers benutzt werden. VBREAK zeigt normalerweise auf einen RTI-Befehl.

Die IRQs werden gemeinsam durch die 6502-Instruktion CLI bzw. SEI ein- oder ausgeschaltet. Weiterhin haben die IRQS eigene Ein/Ausschalt-Bits. Ein Abschnitt des Hardware-Manuals führt die IRQs sowie die Ein/Ausschalt-Bits auf.

Das Register IRQEN enthält die meisten der E/A-Bits für die IRGs und ist ein Nur-Schreib-Register. Das OS besitzt eine für den Benutzer zugängliche Schattenadresse. Diese Adresse ist POKMSK und wird während des Vertical-Blanks geschrieben.

BENUTZEN DER IRQS

Viele Programme erfordern eine „gesicherte" Tastatur. Der Benutzer kann also jede Tasten-Kombination drücken, wobei nur die zulässigen akzeptiert und die anderen ignoriert werden. Der Benutzer kann eine Reihe von IRQs verwenden, um diese Sicherung zu erreichen. Das Beispiel in Abbildung 8.8 benutzt den VKEYBD-IRQ-Vektor, um die Kontroll-Taste (CTRL) abzuschalten, d.h. unwirksam zu machen. Die Routine maskiert außerdem die (BREAK)-Taste, indem der VIMIRQ-Vektor geändert und der (BREAK)-Tasten-Interrupt ignoriert wird.

DER NMI-HANDLER

Das OS besitzt einen NMI-Handler zum Bearbeiten der nicht-maskierbaren Interrupts. Im Gegensatz zu den IRQs können die NMIs nicht „maskiert" werden, d.h. sie können nicht über den 6502 ausgeschaltet werden. Alle NMIs ausgenommen des SYSTEM-RESET-Interrupts, können über den ANTIC-Prozessor ausgeschaltet werden.

Zwei NMIs, der Display-List-Interrupt und der Vertical-Blank-Interrupt VBLANK) , besitzen RAM-Vektoren, die vom Programmierer benutzt werden können. Die Vektoren sind:

Name	**Vektor**
SYSTEM-RESET	keinen
DISPLAY-LIST-INTERRUPT	VDSLST($0200)
VERTICAL-BLANK:	
IMMEDIATE	VVBLKI($0222)
DEFERRED	VVBLKD($0224)

Der SYSTEM-RESET-NMI besitzt keinen RAM-Vektor. Ein SYSTEM-RESET resultiert immer in einem Sprung zur Warmstart-Routine des Monitors (siehe Abschnitt 8.5). Der DLI- und der VBLANK-Interrupt werden in Kapitel 5 bzw. in Anhang I besprochen.

```
0000           10 POKMSK = $0010
D209           20 KBCODE = $D209
0208           30 VKEYBD = $0208
D20E           40 IRQEN  = $D20E
D20E           50 IRQST  = $D20E
0216           60 VMIRQ  = $0216
0000           70       *= $0600
0600 78        80 START  SEI            IRQs abschalten
0601 AD 16 02  90        LDA VMIRG      IRQ-Vektor durch
0604 8D 4D 06  0100      STA NBRK+1     einen vom Benutzer
0607 AD 17 02  0110      LDA VMIRQ+1    ersetzen. Alle IRQs
060A 8D 4E 06  0120      STA NBRK+2     gehen dann zu NBRK.
060D A9 45     0130      LDA #IRQ&255
060F 8D 16 02  0140      STA VIMRQ
0612 A9 06     0150      LDA #IRQ/256
0614 8D 17 02  0160      STA VIMRQ+1
0617 58        0170      CLI            IRQs einschalten
0618 AD 08 02  0180      LDA VKEYBD     Tastatur-IRQ zeigt
061C 8D 43 06  0210      STA JUMP+1     nach REP.
061F AD 09 02  0220      LDA VKEYBD+1
0622 8D 44 06  0230      STA JUMP+2
0625 A9 39     0240      LDA #REP&255   Tastatur-IRQ-Vektor
0627 8D 08 02  0250      STA VKEYBD     niederwertiges Byte
062A A9 06     0260      LDA #REP/256   des Vektors
062C 8D 09 02  0270      STA VKEYBD+1
```

```
062F 60        0280      RTS
               0290     *= $0639
0639 AD 09 02  0300 REP  LDA KBCODE     Hierher laufen alle
063C 29 80     0310      AND #$80       Tastatur-IRQs. CTRL-
063E F0 02     0320      BEQ JUMP       Taste sedrueckt? Nein,
0640 68        0330      PLA            dann springe.. andern-
0641 40        0340      RTI            falls CTRL ignorieren.
0642 4C 42 06  0350 JUMP JMP JUMP       Dieses ruft den alten
0645 48        0360 IRQ  PHA            Tasten-IRQ.
0646 AD 0E D2  0380      LDA IRQST      <BREAK>?
0649 10 04     0390      BPL BREAK      Ja-, BREAK-IRQ
064B 68        0400      PLA            Andernfalls alter IRG
064C 4C 4C 06  0410 NBRK JMP NBRK       Alten IRQ-Vektor auf-
064F A9 7F     0430 BREAKLDA #$7F       rufen.
0651 8D 0E D2  0440      STA IRQST      <BREAK> nicht zeigen
0654 A5 10     0450      LDA POKMSK
0656 8D 0E D2  0460      STA IRQEN
0659 68        0462      PLA
065A 40        0464      RTI            Ruecksprung, Wie ohne
065B           0470     *= $02E2        BREAK.
02E2 00 06     0480      .WORD START
```

Abbildung 8.8:
„Sicherung" der Tastatur

DIE SYSTEM-VEKTOREN

Das OS besitzt zwei Arten von Vektoren: ROM- und RAM-Vektoren. ROM-Vektoren sind Speicherstellen, die JMP-Befehle enthalten, welche auf System-Routinen zeigen. Die RAM-Vektoren enthalten 2-Byte-Adressen zu System-Routinen, zu Handler-Einsprung-zeigern (siehe CIO-Abschnitt 8.2) oder zu Initialisierungs-Routinen.

Sowohl die RAM- und ROM-Vektoren werden sich in neuen Versionen des Operating-Systems nicht ändern. Dieses gilt aber nicht für ihre Inhalte, d.h. ein Programm, das auf dem jetzigen und allen zukünftigen Systemen laufen soll, muß auf die Vektoren, anstatt auf die in ihnen befindlichen Adressen zeigen. Die Vektoren, ihre Inhalte und eine kurze Beschreibung ihrer Funktion findet sich in den Abbildungen 8.9a und 8.9b.

ROM-Vektoren

Name	Speicher-stelle	Funktion
DISKIV	$E450	Initialisierung des Disk-Handlers
DSKINV	$E453	Vektor für Disketten-Handler
CIOV	$E456	Vektor für zentrale I/0-Routine
SIOV	$E459	Vektor für die serielle I/O-Routine
SETVBV	$E45C	Routinen-Vektor z.Setzen d. System-Timer
SYSVBV	$E45F	Berechnungen d.Systems f.d. Vert.Blank
XITVBV	$E462	Berechnungen z.Ausgang a.d. Vert.Blank
SIOINV	$E465	Initialisierung des seriellen I/Os.
SENDEV	$E468	Einschaltroutine z.Senden ü.d.ser.Bus.
INTINV	$E46B	Interrupt-Handler-Routine.
CIOINV	$E46E	Initialisierung des zentralen I/0s.
BLKBDV	$E471	Blackboard-Modus (MEMOPAD)-Vektor.
WARMSV	$E474	Warmstart-Einsprungpunkt (SYSTEM-RESET).
COLDSV	$E477	Kaltstart-Einsprungpunkt (Einschalten).
RBLOKV	$E47A	Routine zum Block-Einlegen von Cassette.
CSOPOV	$E47D	Bereitmachen für Eingabe (Cassette).

Ein Beispiel für die Verwendung eines ROM-Vektors wäre:

```
JSR CIOV
```

Abbildung 8.9a:
ROM-Vektoren

Name	Stelle	Inhalt	Funktionen
VDSLST	$0200	$E7B3	Display-List-Vektor (NMI)
VPRCED	$0202	$E7B3	IRG-Vektor Proceed-Leitung-z.Z.unbenutzt
VINTER	$0204	$E7B3	IRG-Vektor Interrupt-Leitung-z.Z.unben.
VBREAK	$0206	$E7B3	IRG-Vektor So+tware-BREAK
VKEYBD	$0208	$FFBE	IRG-Vektor Tastatur.
VSERIN	$020A	$EB11	IRG-Vektor serielle Eingabe beendet.
VSEROR	$020C	$EA90	IRG-Vektor serielle Ausgabe beendet.
VSEROC	$020E	$EAD1	IRQ-Vektor ser. Ausgabe abgeschlossen
VTIMR1	$0210	$E7B3	IRQ-Vektor POKEY-Timer 1
VTIMR2	$0212	$E7B3	IRG-Vektor POKEY-Timer 2
VTIMR4	$0214	$E7B3	IRG-Vektor POKEY-Timer 4.
VIMIRQ	$0216	$E6F6	Direkter IRD-Vektor zum IRG-Handler.
VVBLKI	$0222	$E7D1	Immediate NMI-Vektor Vertical-Blank.
VVBLKD	$0224	$E93E	De+erred NMI-Vektor Vertical-Blank.
CDTMA1	$0226	$xxxx	JSR-Adresse für System-Timer 1.
CDTMA2	$0228	$xxxx	JSR-Adresse für System-Timer 2.
CASINI	$0002	$xxxx	Initialisierung des Cassetten-BOOTs.
DOSINI	$000C	$xxxx	Initialisierung des Disketten-BOOTs.
RUNVEC	$02E0	$xxxx	DUP-File Run-Vektor
INIVEC	$02E2	$xxxx	DUP-File Initialis.-Vektor.
HATABS	$031A	"P"	Drucker-Geräte I.D.
	$031B	$E430	Adr. d. Drucker Einsprungpunkt-Tafel.

$031D	"C"	Cassetten-Geräte I/D.
$031E	$E440	Adr. d. Cassetten Einsprungpunkt-Tafel.
$0320	"E"	Display-Editor I.D.
$0321	$E400	Adr. d. Dis.Edit.Einsprungpunkt-Ta+el.
$0323	"S"	Bildschirm-Handler I.D.
$0324	$E410	Adr. d.B-Handler Einsprungpunkt-Tafel.
$0326	"K"	Tastatur-Handler I.D.
$0327	$E420	Adr. d. Tast.-Handleinsprungpunkt-Ta+el.
$0329	"x"	Unbenutzter HATABS-Eintrag 1
$032B	"x"	 2
$032E	"x"	 3
$0331	"x"	 4
$0234	"x"	 5
$0237	"x"	 6
$032A	"x"	 7
$034D	"x"	 8
$0340	"x"	 9

Ein „X" zeigt einen sich ändernden Inhalt an.
Ein Beispiel für die Benutzung eines RAM-Vektors wäre:

```
JSR CALL
CALL JMP (DOSINI)
```

Abbildung 8.9b:
RAM-Vektoren

```
                0010 ; seschrieben von MiChaei Ekbers

0600            0030 START     = $600
000C            0040 DOSINI    = $0C
02E7            0050 MEMLO     = $2E7
3000            0060 NEWMEM    = $3000 Dieser Wert legt die
                0065 ; Groesse fest.
                0070 ; Diese Routine sichert Speicherplatz
                0080 ; fuer Assembler-Routinen, indem der
                0090 ; MEMLO-Zeiger gesetzt wird. Sie laeuft
                0100 ; als AUTORUN.SYS-File. Sie setzt
                0110 ; ausserdem MEMLO bei <SYSTEM-RESET>
                0120 ; zurueck. MEMLO wird auf den Wert von
                0130 ; NEWMEM sesetzt.
                0135 ;
                0140 ; Dieser Teil ist Permanent, d.h. muss
                0150 ; resident sein. Der DOS-InitVektor
                0160 ; wurde geaendert und in die Speicher-
                0170 ; stelle INITDOS+1 & 2 gebracht. Das
                0180 ; DOS wird initialisiert, worauf
                0190 ; gleiches mit MEMLO geschieht. INITDOS
                0191 ; wird bei <SYSTEM-RESET> ausgefuehrt.
0000            0200           *= START
```

```
                0210 INITDOS
0600 20 0D 06   0220          JSR CYNTHIA     ;DOS INITLIST
0603 A9 00      0230          LDA #NEWMEM&255 ;ausfuehren
0605 8D E7 02   0240          STA MEMLO
0608 A9 30      0250          LDA #NEWMEM/256
060A 8D E8 02   0260          STA MEMLO+1
                0270 CYNTHIA
060D 60         0280          RTS
                0290 ; Dieser Teil wird nur beim Einschalten
                0300 ; ausgefuehrt und kann danach geloescht
                0310 ; werden. Diese Routine speichert die
                0320 ; Inhalte von DOSINI in einem JSR bei
                0330 ; der Speicherstelle INITDOS+1. Danach
                0340 ; wird der Wert von DOSINI ersetzt;
                0350 ; der neue ist die Speicherstelle
                0360 ; INITDOS.
                0390 JACKIE
060E A5 0C      0400          LDA DOSINI      ;DOSINI sichern
0610 8D 01 06   0410          STA INITDOS+l
0613 A5 0D      0420          LDA DOSINI+1
0615 8D 03 06   0430          STA INITDOS+2
0618 A9 00      0440          LDA #INITDOS&255 ;DOSINI setzen
061A 85 0C      0450          STA DOSINI
061C A9 06      0460          LDA #INITDOS/256
061E 85 0D      0470          STA DOSINI+1
0620 A5 00      0480          LDA NEWMEM&255  ;MEMLO setzen
0622 8D E7 02   0490          STA MEMLO
0625 A9 30      0500          LDA #NEWMEM/256
0627 8D E8 02   0510          STA MEMLO+1
062A 60         0520          RTS
062B            0530          *= $2E2
02E2 0E 06      0540          .WORD JACKIE ;RUN-Adresse setzen
```

Abbildung 8.10: MEMLO-Änderer

DER MONITOR

Der OS-Monitor ist ein Programm im ROM, das die Einschalt- und SYSTEM-RESET-Sequenz des Gerätes bearbeitet. Im Bezug auf ihre Funktion sind Einschalt- und SYSTEM-RESET-Sequenz identisch. In Anhang IV befindet sich ein Flussdiagramm der Einschalt- und SYSTEM-RESET-Routinen.

Die Einschalt-Routine (auch als Kaltstart-Routine bezeichnet) ist auf zwei Arten ansprechbar: durch Einschalten des Computers oder einen Sprung nach COLDSV($E477). COLDSV ist der System-Routinen-Vektor zur POWER-UP (Einschalten)-Routine. Wichtige Punkte beim Einschalten sind:

1. Der gesamte Speicher, ausgenommen die Adressen $0000 bis $000F, wird gelöscht.

2. Disketten- und Cassetten-BOOT werden versucht. Das Flag BOOT?($0009) signalisiert den Erfolg oder Misserfolg des BOOTs. Bei einem erfolgreichen Cassetten-BOOT ist Bit 0=1 bei einem erfolgreichen Disketten-BOOT Bit 1=1.

3. COLDST($0244) ist ein Flag, das dem Monitor mitteilt, ob ein POWER-UP oder ein SYSTEM-RESET vorliegt. COLDST=0 bedeutet SYSTEM-RESET, COLDST ungleich 0 bedeutet POWER-UP. Eine interessante Verwendung dieses Flags wäre das Setzen auf einen Wert ungleich Null, während ein geBOOTetes Programm ausgeführt wird. Dieses hat zur Folge, dass ein SYSTEM-RESET zu einem POWER-UP wird. Diese Technik erhöht den Schutz des Programms, da der Benutzer keine Kontrolle über den Computer erlangen kann, während es läuft.

Das Drücken der RESET-Taste verursacht einen SYSTEM-RESET (auch als Warmstart bezeichnet). Wichtige Punkte beim SYSTEM-RESET sind:

1. Die RAM-Vektoren des OS werden während des SYSTEM-RESETs und während des Einschaltens vom ROM übertragen. Möchte der Benutzer einen Vektor ändern, so müssen einige Vorkehrungen getroffen werden, damit SYSTEM-RESET entsprechend bearbeitet werden kann. Siehe Kapitel 9 für die Behandlung das SYSTEM-RESETs.

2. MEMLO, MEMTOP, APPMHI, RAMSIZ und RAMTOP werden beim SYSTEM-RESET zurückgesetzt. Werden diese Zeiger geändert, um Speicherplatz für Assembler-Routinen in BASIC freizumachen, so müssen ebenfalls einige Vorkehrungen getroffen werden, damit SYSTEM-RESET entsprechend bearbeitet werden kann. Abbildung 8.10 ist ein Beispiel hierfür.

DIE SYSTEMVARIABLEN

Die Systemvariablen enthalten viele für den Benutzer interessante Informationen. Mit Assembler-Routinen oder POKE- und PEEK-Befehlen kann direkt auf diese Adressen zugegriffen werden.

Die Systemvariablen liegen auf den RAM-Seiten 0 bis 4 ($0000 bis $03FF). Sie enthalten System-Flags, I/O-Puffer, I/O-Paramter usw. Der Programmierer kann einige dieser Speicherstellen benutzen, um Funktionen anzuwenden, die nicht durch das OS oder ein Programm (z.B. BASIC) bereitgestellt werden. Abbildung 8.11 zeigt einige dieser Elemente:

Die Systemvariabeln

Name	Adresse Größe	Init. Durch	Wert	Funktion
MEMLO	$2E7,2	R,P	x	Anfang des freien Benutzerspeichers
MEMTOP	$2E5,2	R,P	x	Ende des freien Benutzerspeichers
APPMHI	$00E,2	P,R	$00	fr. Ben.spchr.unt.Begr.d.Bildschirms
RAMTOP	$06A,2	P,R	x	Displ.-Handler obere RAM-Adr. (MSB).
RAMSIZ	$2E4,1	P,R	x	obere RAM-Adresse (MSB)
POKMSK	$010,1	P,R	x	OS IRQ-Einschalt-übertragungs-Adr.
BRKKEY	$011,1	P,R	$FF	BREAK-Tasten-Flag. Jeder Wert ungleich 0 bedeutet: Taste nicht gedr.
IRQEN	$D20E	P,R	x	ANTIC-IRQ-Einschalt-Bit-Register.
PTIMOT	$01C	P,R	x	Drucker-Timeout-Wert für SIO.
ZIOCB	$020,16			Zero-Page I/O-Block der CIO.
BOOT?	$009,1	P,R	x	BOOT-Flag. Bit 0 für Kassette, Bit 1 für Diskette
CKEY	$04A,1	P	x	Monitor-Flag f. Kassetten-Boot
CASSBT	$04B	P		Flag für Kassetten-Boot
KBCODE	$D209,1		x	Register für Tastatur-Code
CH	$2FC,1	P,R	$FF	augenblicklicher Tastenwert.
CH1	$2F2,1		x	letzter Tastenwert.
SHLOK	$2BE,1	P,R	$40	Shift-Lock (Verrieglung).
KEYDEL	$2BF1,1		$03	Timer für Tastenentprellung.
SSFLAG	$2FF,1		x	Start/Stop-Flag für Bildschirm
ATRACT	$04D,1			Flag für Attract-Modus
SRTIMR	$22B,1		x	Timer für Attract-Modus-Timeout
COLDST	$224,1			Initialisierungs-Flag
WARMST	$008,1			dito.
CHBAS	$2F4,1	P,R	$E0	Zeiger für Zeichensatz-Basis
CRITIC	$042	P,R		Flag für kritische I/O-Region.

P: POWER-UP = Einschalten
R: SYSTEM-RESET
x: sich ändernder Wert

Abbildung 8.11:
Datenbasis des Systems

Speicher-Zeiger

Das OS benutzt 5 Speicherstellen, um den Benutzer- und Bildschirm-Speicher zu überwachen: MEMLO, MEMTOP, APPMHI, RAMTOP und RAMSIZ. Ihre Zusammenhänge werden in einer einfachen Speicherübersicht in Abbildung 8.12 dargestellt.

MEMLO ist eine 2-Byte Speicherstelle, die das OS verwendet, um anzuzeigen, wo ein Anwendungs-Programm beginnen kann. Wird MEMLO vorsichtig benutzt, so kann es zum Reservieren von Speicherplatz für kleine Assembler-Routinen in BASIC verwendet werden. Der BASIC-Interpreter benutzt den MEMLO-Zeiger um festzulegen, wo ein Programm anfangen kann (siehe Kapitel 7 für die Struktur von BASIC-Programmen). Soll der MEMLO-Zeiger auf eine höhere Adresse gesetzt werden, so muß dieses geschehen, bevor das BASIC-Modul gestartet wird. MEMLO muß außerdem sehr vorsichtig verwendet werden, da diese Speicherstelle beim Einschalten und beim SYSTEM-RESET wieder zurückgesetzt wird.

Wird von einer Diskette gebootet, so kann die AUTORUN.SYS-Möglichkeit benutzt werden, um MEMLO auf einen festgelegten Wert zu setzen. Da das DOS den Zeiger auch bei einem SYSTEM-RESET über den DOSINI-Vektor zurücksetzt, ist es notwendig diesen zu ändern. DOSINI enthält die Adresse des Initialisierungs-Programms für das DOS. Es wird durch den Monitor als ein Teil der Initialisierung des Systems aufgerufen. Der Inhalt von DOSINI muß in eine 2-Byte-Adresse einer JSR-Anweisung übertragen werden, um ein Teil des POWER-UPs zu werden. DOSINI wird dann auf die Adresse der JSR-Instruktion für das Initialisierungs-Programm gesetzt. MEMLO enthält einen festgelegten Wert. Bei einem SYSTEM-RESET wird das neue Initialisierungs-Programm aufgerufen, wobei dessen erste Instruktion JSR OLDDOSINI, wiederum das DOS initialisiert. Das neue Init-Programm setzt MEMLO auf den festqelegten Wert und springt zur alten Init-Sequenz zurück (RTS). Abbildung 8.10 ist hierfür ein Beispiel.

Die oben beschriebene Technik kann ebenso für **MEMTOP** (der Zeiger für das obere RAM-Ende) verwendet werden. Durch Verringern des MEMTOP-Wertes kann Speicherplatz für Assembler-Routinen reserviert werden. Allerdings ergibt sich bei der Verwendung von MEMTOP anstelle von MEMLO ein Problem. MEMTOP ändert sich mit der Größe des freien RAMs und dem Grafik-Modus des Bildschirms. Dieses macht es schwierig seinen Wert zu bestimmen. Aus diesem Grunde müßen die Assembler-Routinen frei verschiebbar sein.

APPMHI ist eine Speicherstelle, die die Adresse enthält, welche wiederum die niedrigste Adresse angibt, an die das Bildschirm-RAM gelegt werden kann. Diese Variable wird gesetzt, um Programme oder Daten vor dem Überschreiben durch den Display-Handler zu schützen.

RAMSIZ kann genau wie MEMTOP zum Sichern von Speicherplatz für kleine Assembler-Routinen benutzt werden. Der Vorteil von RAMSIZ gegenüber MEMTOP liegt darin, daß der reservierte Speicherbereich hinter dem Bildschirm-RAM liegt, der durch MEMTOP gesicherte Speicherbereich liegt unter dem Bildschirmbereich. Letzterer kann sich durch Ändern des Grafik-Modus ausdehnen oder zusammenziehen.

MEMORY MAP

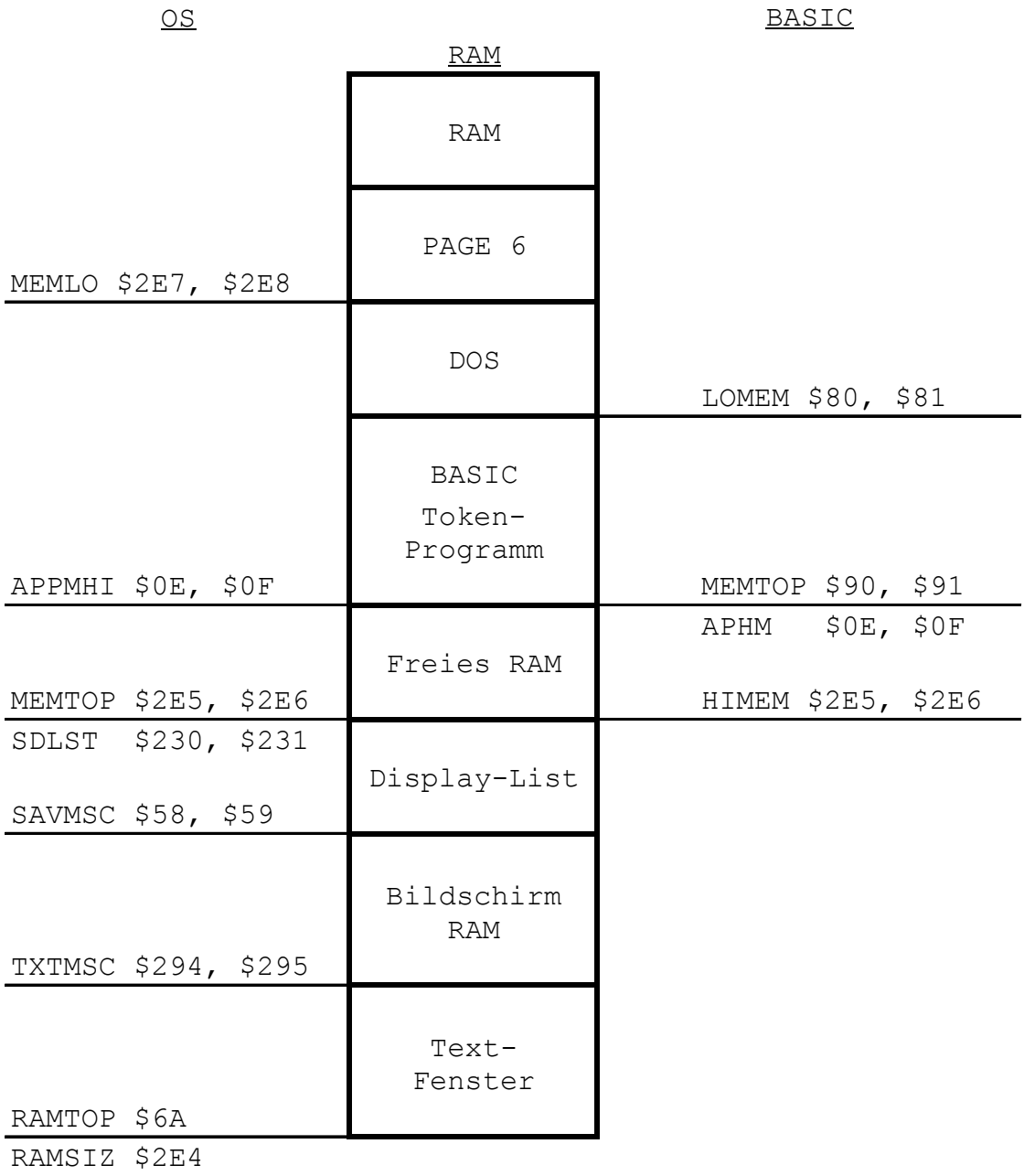


Abbildung 8.12:
OS- und BASIC-Zeiger (mit DOS)

Verschiedenes

BRKKEY ist ein Flag, das gesetzt wird, wenn das OS ein Drücken der BREAK-Taste registriert. Der normale Wert von BRKKEY ist $FF. Ändert er sich, so wurde die BREAK-Taste gedrückt. Der hierfür zuständige IRQ (nicht identisch mit dem IRQ für die softwaremäßige BRK-Anweisung) muß eingeschaltet werden, damit das OS die BREAK-Taste abfragen kann.

Der Timeout-Wert des Druckers wird in der Speicherstelle **PTIMOT** gespeichert. Sie enthält den Wert der Timeaut-Periode für SIO in Sekunden. Diese Zeit kann durch Ändern des Wertes in PTIMOT beliebig verlängert oder verkürzt werden. PTIMOT wird auf 30 Sekungen initialisiert; dies geschieht bei jedem Öffnen (OPEN) des Druckers. Der normale Timeout-Wert für den 825-Drucker beträgt 5 Sekunden.

```
1 POKE 752,1:BOTO 3
3 ? "<":REM Bildschirm loeschen
4 ? "Stunde";:INPUT HOUR:? "Minute";:INPUT MIN:?
"Sekunde";:INPUT SEC
5 CMD=1:GOSUB 65
6 ?" ";HOUR;":";MIN;":";SEC:?" ":?" "
7 CMD=2:GOSUB 65
9 ?"";HOUR;":";MIN;":";SEC;"  ":GOTO 7
10 REM Dies ist ein Beispiel fuer die Erstellung einer
20 REM Echtzeit-Uhr. Diese Routine erwartet die Eingabe
30 REM einer Zeit (Std., Min., Sec.).Danach wird die Echt-
40 REM zeit-Uhr auf 0 gesetzt. Der augenblickliche Wert
50 REM in RTCLOCK wird benutzt, damit durch Addieren zum
60 REM Wert der Echtzeit-Uhr die aktuelle Zeit entsteht.
65 HIGH=1536:MED=1537:LDW=1538
66 REM
67 REM *****Einsprungpunkt*****
68 REM
70 ON CMD BOTO 100,200
80 REM
90 REM *****Uhr initialisieren*****
95 REM
100 POKE 20,0:POKE19,0:POKE18,0
105 DIM CLOCK$(50)
106 CLOCK$=" ":GOSUB 300
110 IHOUR=HOUR:IMIN=MlN:ISEC=SEC:RETURN
197 REM
198 *****Uhr lesen*****
199 REM
200 REM
201 A=USR(ADR(CLOCK$))
210 TIME=((PEEK(HIGH)*256+PEEK(MED))*256+(PEEK(LOW))/50
220 HOUR=INT(TIME/3600):TIME=TIME-(HOUR*3600)
```

```
230 MIN=INT(TIME/60):SEC=INT(TIME-(min*60))
235 SEC=SEC+ISEC:IF SEC>60 THEN SEC=SEC-60:MIN=MIN+l
236 MIN=MIN+IMIN:IF MIN>60 THEN MIN=MIN-60:HOUR=HOUR+l
237 HOUR=HOUR+IHOUR
240 HOUR=HOUR-(INT(HOUR/24))*24
250 RETURN
300 FOR I=l TO 38:READ Z:CLOCK$(I,I)=CHR$(Z):NEXT I:RETURN
310 DATA 104,165,18,141,0,6,165,19,141,1,6,165
320 DATA 20,141,2,6,165,18,205,0,6,208,234
330 DATA 165,19,205,1,6,208,227,165,20,205,2,6,209,220,96
```

Abbildung 8.13:
Echtzeit-Uhr

DIE SYSTEM-TIMER

Es gibt zwei Arten von Timern. Die System-Timer besitzen die Bildschirm-Frequenz. Für nordamerikanische Fernsehgeräte beträgt diese Frequenz 59,923334 Hertz. Europäische Geräte haben eine Frequenz von 50 Hertz. Die POKEY-Timer arbeiten mit durch den Benutzer veränderlichen Frequenzen.

Es gibt 6 System-Timer:

Name	**Speicherstelle**	**Vektor/Flag**
RTCLOK	$0012,3	keine
CDTMV1	$0218,2	CDTMA1 $0226,2
CDTMV2	$021A,2	CDTMA2 $0228,2
CDTMV3	$021C,2	CDTMA3 $022A,1
CDTMV4	$021E,2	CDTMA4 $022C,1
CDTMV5	$0220,2	CDTMA5 $022E,1

Alle System-Timer werden als Teil des Vertical-Blanks VBLANK) aktualisiert. Wenn dieses ausgeschaltet oder umgangen wird, werden die Timer nicht mehr geändert.

Die Echtzeit/Uhr (RTCLOK) und der System-Timer Nr. 1 (CDTMV1) werden während der ersten Stufe des VBLANKs auf den neuesten Stand gebracht. RTCLOK zählt von 0 an aufwärts und ist ein 3-Byte-Wert. Wenn RTCLOK seinen maximalen Wert erreicht (16.777.216), so wird dieser wieder auf Null zurückgesetzt. In Beispiel 8.13 wird eine Anwendung für RTCLOK gezeigt.

Da die System/Timer als ein Teil des VBLANK-Prozesses aktualisiert werden, ist besondere Aufmerksamkeit erforderlich, damit dieses korrekt geschieht. Eine mit SETVBV (.$E45C.) bezeichnete Routine wird benutzt, um sie zu setzen. Der Aufruf dieser Routine sieht wie folgt aus:

Register X & Y : Timer-Wert

```
Register A     : Nummer des Timers: A=l bis 5=> Timer Nr. 1-5

Beispiel:

     LDA #1    ; Setzen des System-Timers Nr. 1
     LDY #0
     LDX $02   ; Der Wert beträgt $200 VBLANKs.
     JSR SETVBV     ;Aufrufen der Routine zum Setzen der Timer


5 ? " ":REM Bildschirm löschen
6 REM Routine zum Setzen einer Metronom-Rate.
7 REM Von M. Ekbers für Carla.
10 X=10:FOR I=l TO 2 STEP 0
20 TOP=l0:FOR J=l TO TOP:NEXT J:REM Verzögerungs-Schleife
50 IF STICK(0)=l3 THEN X=X+1:REM Nach oben = schneller
51 IF STICK (0)=14 THEN X=X-1:REM Nach unten = langsamer
52 IF X<l THEN X=1:REM X darf niemals kleiner als 1 oder
53 IF X>255 THEN X=255:REM grösser als 255 sein.
54 REM
56 ?"";INT(3600/X);" Schläge pro Minute             "
60 POKE 0,X:REM In Speicherstelle $0000 steht die Rate
70 NEXT I:REM für die folgende Assembler-Routine.

-------------------------------------------------------------
40             *=$600
50 ; Metronom-Routine...benutzt $0000 zum Uebersetzen d. Rate.
60 ;
70 AUDF1       = $D200   Audio Frequenz-Register
80 AUDC1       = $D201   Audio Kontroll-Register
90 FREG        = $08     AUDF1-Wert
0100 VOLUME    = $AF     WERT VON AUDC1
0110 OFF       = $A0Lautstaerke ausschalten
0120 SETVBV    = $E45C   Routine zum Setzen des Timer-Wertes
0130 CDTMA2    = $0228   Vektor von Timer 2
0140 ZTIMER    = $0000   VBLANK-Wert des Timers in Zero-Page
0150 ;
0160 START     LDA #10
0170           STA ZTIMER
0180 INIT
0190 ; Setzen des Timer-Vektors
0200 ;
0210           LDA #CNTINT&255
0220           STA CDTMA2
0230           LDA #CNTINT/256
0240           STA CDTMA2+1
0250 ;
0260 ; Setzen dem Timers nach dem Voktor
0270 ;
0290           LDY ZTIMER     Setzen von Timer 2 auf Zaehler
0290           JSR SETIME
0300           RTS
0310 ;
0320 ;
```

```
0330 ;
0340 CNTINT
0350 ;
0360 ; Bereitmachen das Audio-Kanals für "Klick"-
0370 ; Geraeusch
0380           LDA #VOLUME
0390           STA AUDC1
0400           LDA #FREQ
0410           STA AUDF1
0420           LDY #$FF
0430 DELAY
0440           DEY
0450           BNE DELAY
0460           STY AUDC1
0470           JMP INIT
0480 ;
0490 ; Unterroutine zum Setzen das Timers
0500 ;
0510 SETIME
0520           LDX #0          Niemals 256 VBLANKs
0530           LDA #2          Setzen von Timer 2
0540           JSR SETVBV      System-Routine zum Setzen
0550           RTS             der Timer
0560           *=$2E2
0570           .WORD START
0580           .END
```

Abbildung 8.14:
Metronom

Die System-Timer 1 bis 5 sind 2-Byte Zähler. Sie können mit Hilfe der SETVBV-Routine auf einen bestimmten Wert gesetzt werden. Sie werden dann während des VBLANKs durch das OS dekrementiert. Timer 1 wird in der 1. Stufe des immediate VBLANKs dwkrementiert. Bei den Timern 2 bis 5 geschieht dieses während des immediate VBLANKs in der 2. Stufe. Wenn ein bestimmter Timer auf den Wort 0 heruntergezählt wurde, werden verschiedene Aktionen vom OS durchgeführt.

Die System-Timer 1 und 2 besitzen ihnen zugeordnete Vektoren. Erreicht einer dieser Timer den Wert 0, dann simuliert das OS einen JSR-Befehl über den entsprechenden Vektor. In Abbildung 8.9b stehen die Werte der Vektoren für diese beiden Timer.

Die Timer 3 bis 5 besitzen Flags, die normalerweise gesetzt sind, d.h. einen anderen Wert als Null aufweisen. Sobald einer dieser Timer gleich Null ist,löscht das OS das entsprechende Flag, d.h. es setzt es auf 0. Das Flag kann durch den Benutzer geprüft werden, woraufhin dieser dann bestimmte Operationen einleiten kann.

Die Timer 1 bis 5 sind Software-Timer für allgemeine Verwendung. Timer 1 wird z.B.durch die SIO zum Timing von Operationen über den seriellen Bus verwendet. Wird der Wert das Timers auf Null gesetzt bevor eine Bus-Operation abgeschlossen ist, so wird eine "Timeout"-Fehlermeldung erzeugt. Timer 1 wird auf verschiedene Werte gesetzt, abhängig von dem angesprochenen Gerät. Dieses garantiert, daß der Computer nicht endlos auf die Antwort eines nicht vorhandenen Gerätes wartet. Timer 3 wird ebenfalls durch das OS benutzt. Der Cassetten-Handler verwendet diesen Timer zum Festlegen der zum Schreiben und Lesen von Tonband-Headern benötigten Zeit. In Beispiel 8.14 wird Timer 2 zum Simulieren eines Metronoms benutzt. Die Rate des Metronoms kann mit Hilfe des Joysticks auf verschiedene Werte gesetzt werden.

Kapitel 9
Das Disketten-Oparating System (DOS 2.OS)

EINFÜHRUNG

Das Disketten-Oparating-System (DOS) ist eine Erweiterung des OS. Es gestattet den Zugriff auf Disketten / Massenspeicher in Form von Dateien, die wie andere Dateien abgerufen worden können. Es folgt eine Besprechung des DOS und seiner Benutzung.

Das Disketten-Oparating-System besteht aus drei Teilen: dem residenten Disketten-Handler, dem File-Manager (FMS) und der Disketten-Utility (DUP). Der residente Disketten-Handler ist der einzige Teil des OS, der sich im ROM befindet. FMS und DUP befinden sich auf der Diskette und werden in den Computer geladen, sobald dieser eingeschaltet wird (BOOT).

Der residente Disketten-Handler

Der residente Disketten-Handler ist der einfachste Teil des DOS. Der Disketten-Handler folgt nicht, wie andere Handler, der normalen CIO-Aufrufssequenz. Das Verhältnis des Disketten-Handlers zum I/O-Untersystem wird in Abbildung 8.2 in Abschnitt 8.2 dieses Buches dargestellt.

Aus der Abbildung ist zu ersehen, daß der DCB die Kommunikation mit dem Disketten/Handler übernimmt. Die Aufruf-Sequenz für den Disketten-Handler lautet wie folgt:

```
                ;Benutzer muss DCB erstellen
    JSR DSKINV  ;System-Routinen Vektor zum residenten
                ;Disketten-Handler.
    BPL OKAY    ;Bei Erfolg verzweigen, Y-Register=1.
                ;Andernfalls enthält Y-Register den
                ;Fehlercode, der auch in DCBSTA
                ;gespeichert wird.
```

Der Disketten-Handler führt 5 Funktionen aus:

FORMAT - Ausgabe eines FORMAT-Befehls an den Disk-Kontroller
READ SECTOR - Lesen eines bestimmten Sektors
WRITE SECTOR - Schreiben eines festgelegten Sektors

WRITE/VERIFY SECTOR - Schreiben eines bestimmten Sektors mit anschließender Überprüfung
STATUS - Zustand des Disk-Kontrollers abfragen

Der Format/Befehl löscht alle Spuren auf der Diskette, woraufhin die Sektoradressen auf die neuen Spuren geschrieben werden. Durch dieses Kommando wird keine Datei-Struktur auf die Diskette gebracht.

Die 3 I/O-Befehle für Sektoren können zum Lesen oder Schreiben von Sektoren von bzw. auf die Diskette oder zum Aufstellen einer eigenen Datei-Struktur verwendet werden. In Abschnitt 10 des OS-Benutzer-Manuals befindet sich ein Beispiel für die Benutzung des Disketten-Handlers zum Schreiben einer BOOT-Datei.

Die Status-Funktion wird zum Feststellen der Zustände der einzelnen Disketten-Stationen benutzt, kann aber auch für andere Zwecke verwendet werden. Da der Timeout-Wert für den STATUS-Befehl kleiner als für andere Kommandos ist, ist dieser gut geeignet, den Anschluss einer bestimmten Disketten-Station festzustellen. Zeigt der Disketten-Handler eine Timeout-Meldung an, so ist die betreffende Station nicht angeschlossen.

Die DUP (Disk Utility Package) ist ein Satz von Hilfsprogrammen für das Arbeiten mit der Disketten-Station, der auf dem Bildschirm als DOS-Menü erscheint. DUP führt Kommandos aus, indem das FMS über die CIO aufgerufen wird. Die Befehle sind:

A. DIRECTORY (Disketten-Verzeichnis)
B. ROM CARTRIDGE (Kontrolle an das Modul übergeben)
C. COPY FILES (Kopieren von Dateien)
D. DELETE FILES (Löschen von Dateien)
E. RENAME FILE (Umbenennen einer Datei)
F. LOCK FILES (Schützen von Dateien)
G. UNLOCK FILE (Datei-Schutz aufheben)
H. WRITE DOS FILES (Schreiben von DOS-Dateinn)
I. FORMAT DISK (Diskette formatieren)
J. DUPLICATE DISK (Diskette duplizieren)
K. SAVE BIN-FILE (Sichern von Binär-Dateinn)
L. LOAD BIN-FILE (Laden von Binär-Dateien)
M. RUN AT ADRESS (Bei festgelegter Adresse starten)
N. WRITE MEM.SAV FILE (Schreiben einer MEM.SAV-Datei)
0. DUPLICATE FILE (Datei duplizieren)

FMS

Das FMS (File Management System) ist ein nicht-residenter Geräte-Handler, der das normale Interface "Geräte-Handler-CIO" benutzt. Das FMS ist nicht im OS-ROM vorhanden. Es wird beim Einschalten geBOOTet, sofern sich eine Diskette in der Station befindet, auf der das DOS vorhanden ist.

Das FMS erhält, wie die anderen Geräte-Handler, die I/O-Kontrolldaten von der CIO. Das FMS verwendet dann den residenten Disk-Handler, um ein Eingabe/Ausgabe für die Diskette vorzunehmen. Das FMS wird aufgerufen, indem ein IOCB aufgestellt und die CIO angesprungen wird. Das FMS liefert einige spezielle CIO-Funktionen, die bei anderen Handlern nicht verfügbar sind:

FORMAT Das FMS ruft den residenten Disk-Handler auf, damit die Diskette formatiert wird. Nach einer erfolgreichen Ausführung dieser Aktion schreibt das FMS einige Daten für die Datei-Struktur auf die Diskette.

NOTE Das FMS zeigt den augenblicklichen Wert das Datei-Zeigers an.

POINT Das FMS setzt den Datei-Zeiger auf den festgelegten Wert

Disketten-I/O

Der Programmierer kann alle standardmäßigen I/O-Aufrufe für Dateien über die CIO ansprechen. In BASIC werden dazu die I/O-Kommandos, wie z.B. OPEN, CLOSE, GET oder PUT benutzt. In Assembler ist es erforderlich, daß ein IOCB vom Benutzer erstellt und die CIO aufgerufen wird. Es folgt eine Einführung in das DOS, bei welcher der Einfachheit halber BASIC verwendet wird.

Um eine Ein-/Ausgabe über die Diskettenstation durchzuführen, muß als erste Aktion das Bereitmachen einer Datei ausgeführt werden. Das BASIC-Format hierfür lautet wie folgt:

```
OPEN #IOCB,ICAX1,0,"D:MYPROG.BAS"
```

"#IOCB" wählt einen der 7 verfügbaren IOCBs (in BASIC sind es nur 7, da der Interpreter selbst IOCB Nr.0 benutzt).

ICAX1 ist der Code für die Art des OPENs. Die Bits für diesen Code sind:

```
Bit 7 6 5 4 3 2 1 0
    x x x x W R D A
```

Wobei: A: bedeutet APPEND (Anfügen)
D: bedeutet DIRECTORY (Disketten-Verzeichnis)
R: bedeutet READ (Lesen)
W: bedeutet WRITE (Schreiben)
x: bedeutet nicht benutzt

Die verschiedenen Werte von ICAX1 werden in Abschnitt 5 des OS-Manuals behandelt. Einige wichtige Dinge der verschiedenen OPEN-Madi sind:

ICAX1=6 Dieser Wert wird zum Öffnen des Disketten-Verzeichnisses verwendet. Die danach eingelesenen Records sind die Einträge in das Verzeichnis (Directory) der Diskette.

ICAX1=4 READ (=Lese) -Modus.

ICAX1=8 WRITE (=Schreib) -Modus. Jede in diesem Modus bereitgemachte Datei wird gelöscht. Die ersten geschriebenen Bytes befinden sich direkt am Anfang der Datei. (Wird eine Datei mit einem schon vorhandenen Namen geöffnet, so wird die alte Datei gelöscht!)

ICAX1=9 WRITE APPEND (=fortgesetztes Schreiben) -Modus. Die Datei bleibt erhalten. Beschriebene Bytes werden an das Ende der Datei angefügt.

ICAX1=12 UPDATE (=Ergäzungs) -Modus. Dieser Modus gestattet sowohl das Schreiben als auch das Lesen einer Datei. Das Schreiben/Lesen von Bytes wird am Anfang der Datei begonnen.

ICAX1=13 Nicht vorhanden.

Es folgt jetzt eine Behandlung der Datenübertragung. Es gibt zwei Arten von Ein-/Ausgabe, die durch den Programmierer verwendet werden können: record/ und zeichen-orientierter I/0.

Zeichenorientierte Ein-/Ausgabe bedeutet, daß die Daten in einer Datei lediglich Listen von Bytes sind. Das DOS interpretiert diese Listen als Daten (keine eingebetteten

Kontrollzeichen). Ein Beispiel hierfür (Zeichendaten, alle Werte in Hex-Notation):

00 23 4F 55 FF 34 21

Record (Datensatz)-orientierter I/0 bedeutet, daß die Daten in Records zusammengefaßt sind. Ein Datensatz ist eine Reihe von Bytes, die durch ein EOL-Zeichen ($9B) abgeschlossen wird. Das folgende Beispiel zeigt zwei Datensätzec

00 23 4F 55 FF 34 9B **21 34 44 9B**

Datensatz1 **Datensatz 2**

Record- und Zeichen-orientierten I/0 von Files kann in willkürlicher Reihenfolge geschehen. Daten können in Record-Form als Zeichendaten gelesen werden. Gleiches gilt für die umgekehrte Richtung (Zeichendaten können als Records eingelesen werden). Der einzige Unterschied zwichen zeichen- und record-orinntiertem I/O besteht darin, daß Records mit einem $9B-Zeichen abgeschlossen werden. Bei zeichenorientiertem I/O wird dieses EOL-Zeichen als normaler Datenteil gelesen.

In BASIC ist es sehr gut möglich, mit record-orientiertem I/O zu arbeiten. Die Kommandos PRINT und INPUT werden benutzt, um Records von Dateien zu lesen und zu schreiben. Andererseits ist zeichen-orientierter I/O mit dem BASIC-Interpreter nicht so einfach möglich, wie er sein könnte. Die Befehle GET und PUT gestatten das Schreiben oder Lesen eines einzelnen Bytes zur Zeit.

Das OS besitzt die Fähigkeit ganze Zeichenblöcke zu lesen oder zu schreiben. Diese Fähigkeit wird durch den BASIC-Interpreter nicht genutzt. Neben der Möglichkeit, die Länge eines Blockes festzulegen, kann der Benutzer auch die Adresse festlegen. Um diese Möglichkeit vom BASIC aus anzuwenden, muß der Programmierer ein Assembler-Unterprogramm schreiben und mit der USR(X)-Funktion aufrufen. Abbildung 8.6 in Abschnitt 8.2 dieses Buches zeigt ein Beispiel für eine solche Unterroutine.

RANDOM ACCESS (=freier Zugriff)

Eine der wichtigsten Verwendungen von Disketten liegt im freien Zugriff auf gespeicherte Datensätze in beliebiger

Reihenfolge. Die Benutzung der I/O-Befehle in Verbindung mit den speziellen Kommandos NOTE und POINT macht das Erstellen von frei zugreifbaren Dateien möglich.

Die Befehle NOTE und POINT zeigen bzw. ändern den Wert des Datei-Zeigers. Das DOS besitzt einen File-Zeiger für jede geöffnete Datei, die ihm die augenblickliche Position in der Datei mitteilt. Der Datei-Zeiger besteht aus zwei Parametern: dem Sektor- und dem Byte-Zähler. Die Sektor-Nummer (ein Wert von 1-719) teilt dem DOS mit, auf welchen Sektor der Diskette der Datei-Zeiger deutet. Der Byte-Zähler gibt das anzusprechende Byte innerhalb eines Sektors an (das erste Byte im Sektor hat den Byte-Zählerwert, das zweite den Wert 1 usw.). Abbildung 9.1 zeigt das Verhältnis von Datei-Zeiger und Datei. (Alle Werte stehen in hexadezimaler Notation.)

```
 Datei:
  41  42  43  9B  44  45  46  9B  47  48  49  4A  4B  9B  ...  41  42
   A   B   C EOL  D   E   F EOL  G   H   I   J   K EOL       A   B

 Datei-Zeiger:

 Sektor-Nr.:
  50  50  50  50  50  50  50  50  50  50  50  50  50  50  ...  50  51
 Byte-Zähler
  00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  ...  7C  00
```

Die oben gezeigte Datei wurde folgendermaßen mit Hilfe des BASIC erstellt:

```
10 OPEN #1,8,0,"D:DATEI"
20 ? #1;"ABC"
30 ? #1;"DEF"
40 ? #1:"GHIJK"
     :
     :          :REM Füllen des restlichen Sektors
     :
100 ? #1;"AB":REM Dieser Datensatz ueberschreitet die
110 CLOSE #1:REM Begrenzung des Sektors.
```

Abbildung 9.1:
NOTE- und POINT-Werte

Die Sektor-Nr. beträgt in diesem Beispiel 50, weil das DOS diese Datei in Sektor 50 begann. Diese Zahl ändert sich in 51, da die Datei größer als ein Sektor ist. Der Datensatz "AB" überschreitet die Begrenzung des ersten Sektors.

Der Byte-Zählers des Datei-Zeigers beginnt mit dem Wert Null und wird bis zum Ende des Sektors inkrementiert ($7D=125). Das DOS reserviert mindestens 3 Bytes pro Sektor für Header-Daten der Datei. Für Dateien auf der SIO-Station beträgt der maximale Wert des Byte-Zählers 124 (0 bis 124 = 125 Bytes).Wenn die Datei das Ende des Sektors erreicht, wird der Byte-Zähler wieder auf 0 zurückgesetzt.

Der Leser sollte jetzt einen Eindruck davon haben,wie Records auf Diskette gespeichert und wieder abgerufen werden. Abbildung 9.2 zeigt eine Unterroutine, die Datensätze sichert, ihre Position speichert und wieder liest. In Anhang VIII befindet sich ein in BASIC geschriebenes Programm, das einen absolut freien Zugriff gestattet.

```
1000 REM Diese Routine erstellt Dateien & greift auf sie zu.
1001 REM Es gibt folgende Befehle:
1002 REM CMD=l Schreiben das n-ten Datensatzes
1003 REM CMD=2 Lesen das n-ten Datensatzes
1004 REM CMD=3 Aendern des n-ten Datensatzes
1005 REM
1006 REM RECORD$ ist der Ein-/Ausgabe-Datensatz
1007 REM n ist die Nummer des Datensatzes
1008 REM INDEX ist ein zweidimensionaler, durch
1009 REM DIM INDEX(1,RECNUM) dimensionierter Array,
1010 REM INDEX enthaelt alle Note-Werte jedes Datensatzes
1020 REM IOCB1 ist die vorgegebene Daten-Datei
1030 REM
1200 ON CMD GOTO 2000,3000,4000
2000 REM
2100 REM Schreiben des n-ten Datensatzes
2200 NOTE #1,X,Y
2300 INDEX(SEC,N)=X:INDEX(BYTE,N)=Y
2400 ? #1;RECORD$:RETURN
3000 REM
3100 REM Lesen des n-ten Datensatzes
3200 X=INDEX(SEC,N):Y=(BYTE,N)
3300 POINT #l,X,Y
3400 INPUT #l:RECORD$:RETURN
4000 REM
4100 REM Erneuern des n-ten Datensatzes
4200 REM
4300 X=INDEX(SEC,N):Y=INDEX(BYTE,N)
4400 POINT #1,X,Y
```

```
4500 ? #1;RECORD$:RETURN
```

Abbildung 9.2:
Beispiel für NOTE und POINT

Kapitel 10
TON

---

1. Die Hardware-Möglichkeiten den ATARI Computer Systems

Der ATARI Computer besitzt leistungsfähige Tonerzeugungsfähigkeiten. Es gibt 4 Tonkanäle, wobei jeder unabhängig von den anderen kontrollierbar ist. Jeder Kanal besitzt ein Frequenzregister, welches die Tonhöhe bestimmt, sowie ein Kontroll-Register, das die Lautstärke und den Klang festlegt. Einige Optionen gestatten das Einfügen von HI(gh)-Paß-Filtern, das Setzen verschiedener Optionsmodi usw.

Zum besseren Verständnis dieses Kapitels ist es erforderlich, einige spezielle Fachausdrücke zu erklären:

1 Hz (Hertz) entspricht 1 Schwingung pro Sekunde
1 kHz (Kilo-Hertz) entspricht 1.000 Schwingungen pro Sekunde
1 MHz (Mega-Hertz) entspricht 1.000.000 Schwingungen/Sekunde

Ein Impuls bezeichnet hierbei den ruckartigen Anstieg der elektrischen Spannung, dem ein ebenso plötzlicher Abfall derselben folgt; in Verbindung mit einem Lautsprecher entsteht so ein hörbares „PLOPP"-Geräusch.

Eine „Welle" beschreibt die Spannung in Abhängigkeit zur Zeit wodurch die Tonart bestimmt wird. (Die vom ATARI Computer erzeugten Wellen sind Rechteck-Schwingungen (siehe Abbildung 10.2), Blechblasinstrumente erzeugen Sägezahn-Schwingungen und die von einem Sänger erzeugten Schwingungen weden als Sinus-Kurve bezeichnet.) Wird eine elektrische Welle zum Lautsprecher übertragen, so wird sie in eine Schallkurve umgewandelt.

Eine unterbrochene Folge von Rechteck-Impulsen ergibt eine Rechteckwelle.

Im Computer gibt es sogenannte Schieberegister, die für die Tonerzeugung benötigt werden und einem ganz gewöhnlichen Speicherbyte entsprechen in dem Binär-Daten gespeichert werden. Ein solches Schiebe-Register bewegt allerdings alle Bits um eine Position nach rechts, wenn es dazu angewiesen wird. So erhält Bit 5 den Wert von Bit 4, Bit 4 den Wert von Bit 3 usw. Der Wert des am weitesten rechts stehenden Bit wird aus dem Register geschoben und Bit 1 erhält einen neuen Wert:

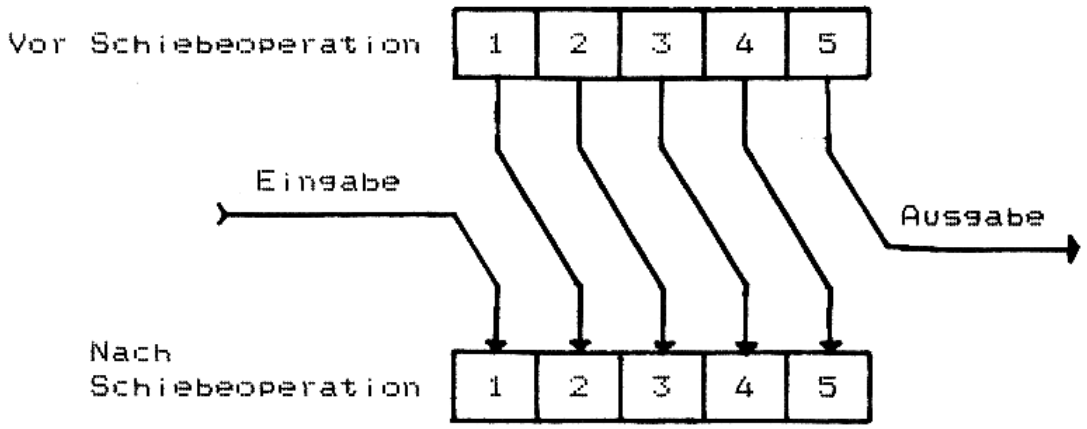


Abbildung 10.1:
Diagramm der Bit-Bewegung in einem Schiebe-Register

Der Ausdruck AUDF1-4 bedeutet: „eines der 4 Audio-Frequenz-Register."
Die Adressen dieser Register sind:
$D201, $D203, $D205, $D207 (53760, 53762, 53764, 53766).

Der Ausdruck AUDC1-4 steht für: „eines der 4 Audio-Kontroll-Register."
Die zugehörigen Adressen sind:
$D201, $D203, $D205, $D207 (53761, 53763, 53765, 53767).

Die Wörter „Frequenz", „Ton" und „Note" werden abwechselnd benutzt. Für dieses Kapitel ist ihre Bedeutung identisch.

„Rauschen" und Verzerrung werden ebenfalls abwechslend in diesem Kapitel benutzt, obwohl ihre Bedeutung in Wahrheit nicht die gleiche ist. „Rauschen" entspricht eher der durch den ATARI 400/800"TM" ausgeführten Funktion.

Der 50-hz-Interrupt, auf den im 2. Teil dieses Kapitels verwiesen wird, kann auch als Vertical-Blank-Interrupt bezeichnet werden.

Alle vorliegenden Beispiele wurden in BASIC geschieben, sofern nicht anders angegeben. Bei der Eingabe dieser Beispiele muß auf die Form geachtet werden, d.h., wenn keine Zeilennummer angegben wurde, darf auch keine eigegeben werden. Stehen mehrere Statements in einer Zeile, so müssen sie auch in einer Zeile stehend eingetippt werden, usw.

Bei Soundmanipulation mittels POKE-Befehl in Basic bzw. in Maschinensprache können kleinere Probleme auftreten, da der POKEY-Chip initialisiert werden muß, bevor er einwandfrei arbeiten kann. In BASIC geschieht dieses mit einer SOUND 0,0,0,0-Anweisung. In Maschinensprache muß eine 0 in AUDCTL ($D208 = 53768) und eine 3 in SKCTL ($D20F 53775) gespeichert werden.

AUDF1-4

Jeder Kanal besitzt ein zugehöriges Frequenz-Register, das die vom Computer zu erzeugende Note kontrolliert.

Das Frequnz-Register enthält einen Wert „N", der in einer "dividiere-durch-"N"-Schaltung verwendet wird. Dieses Teilen ist keine Divsion im mathematischen Sinn, sondern ein einfacheres Zuordnenz: auf jedem N-ten eingehenden Impuls wird ein Impuls ausgegeben. Das folgende Beispiel zeigt eine „dividiere-durch-4"-Funktion:

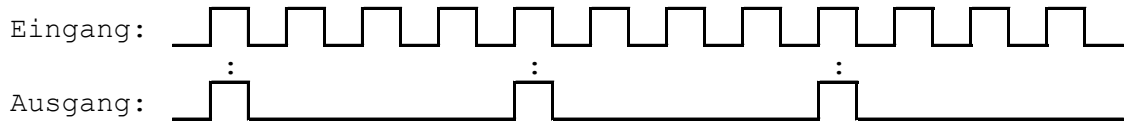


Abbildung 10.2:
„dividiere-durch-4"-Funktion

Wird der Wert von N größer, so sinkt die Zahl der ausgegebenen Schwingungen - der Ton wird tiefer.

Im Zusammenhang mit diesem Kapitel bezeichnet „Frequenz" die Anzahl von Schwingungen innerhalb eines bestimmten Zeitraumes. Ein Teil mit der Frequenz 100 Hz bedeutet z.B., daß in einer Sekunde 100 Schwingungen erzeugt werden. Eine Sopran-Sängerin singt auf einer sehr hohen Frequenz (ungefähr 5 Khz). Eine Kuh erzeugt mit ihrem Muh eine sehr niedrige Frequenz (ungefähr 100 Hz).

AUDC1-4

Jeder Kanal besitzt neben dem Frequenzregister auch ein zugehöriges Kontroll-Register. Diese Register gestatten das Steuern von Verzerrung und Lautstärke für Jeden einzelnen Kanal. Die Bit-Zuordnung von AUDC1-4 lautet wie folgt:

AUDC1-4

Bit:	7	6	5	4	3	2	1	0
	Verzerrung			Nur Laut-stär-ke	Lautstärke			

LAUTSTÄRKE: Die Lautstärken-Kontrolle für die einzelnen Kanäle ist sehr einfach aufgebaut. Die unteren 4 Bit der Audio-Kontroll-Register stellen eine 4 Bit-Zahl dar, welche die Lautstärke 0 angibt. Eine 0 in diesem Nibble bedeutet - nicht hörbar. Eine 15 bedeutet so laut wie möglich. Es gibt 16 Lautstärken.

Die Summe der Lautstärken der 4 Kanäle sollte 32 nicht überschreitenten, da dieses eine Übermodulation der Audio-Ausgabe zur Folge hätte. Der ausgegebene Ton würde nicht mehr die gewünschte Lautstärke besitzen und einem Brummton gleichen.

VERZERRUNG: Jeder Kanal besitzt außerdem noch 3 Bit, welche die Verzerrung kontrollieren. Das Rauschen wird zum Erzeugen spezieller Ton-Effekte benutzt, wenn ein reiner Ton unpassend ist oder nicht gewünscht wird.

Die Anwendung der Verzerrung durch den ATARI"TM"-Computer ist in der Industrie einzigartig. Die Vielseitigkeit und Kontrollierbarkeit gestattet die Erzeugung einer nahezu endlosen Anzahl von Ton-Effekten. Dieses geht vom reinen Ton über Rumpel-, Rassel-, Klick-, und Flüster-Geräuschen zu Quak-Tönen und Hintergrund-Rythmen.

Die vom ATARI"TM" Personal Computer System verwendete Verzerrung entspricht nicht der standardmäßigen Interpretation. So sind z.B. die Begriffe „harmonisch" und „Intermodulations-Verzerrung" bei Stereo-Geräten qualitätsgebend. Sie geben die Stärke einer Schwingungs-Veränderung an, die durch kleine Fehler in der Elektronik auftritt. Die Verzerrung gibt beim ATARI"TM"-Computer nicht die Veränderung von Kurven, sondern die Entfernung der ausgesuchten Schwingungen innerhalb einer Kurve an (es werden vom Gerät immer nur Rechteck-Schwingungen ausgegeben). Diese Technik wird durch

das Wort „Verzerrung” nicht korrekt bezeichnet; ein besserer Begriff wäre „Rauschen”.

Doch bevor uns den „Rausch”-Techniken zuwenden, muß der Leser erst den Begriff „Poly-Zähler” kennen.

Poly-Zähler werden vom ATARI-Computer-System als Quelle für zufällige Schwingungen bei der Rauscherzeugung verwendet. Die Poly-Zähler bestehen zum größten Teil aus Schiebe-Registern (siehe Beschreibung am Anfang dieses Kapitels), die zum schnellen Umschieben (1.79 MHz) verwendet werden. Der Ausgang dieser Schiebe-Register liefert die zufälligen Schwingungen, wobei ihre Eingänge durch Verarbeiten der Werte festgelegter Bits dieser Register bestimmt werden.

Das untere Diagramm zeigt, daß z.B. der Wert von Bit 5 aus dem Register geschoben wird, wodurch die nächste Zufall-Schwingung erzeugt wird. Um das freigewordene Bit 1 neu zu laden, werden die Bits 5 und 3 miteinander verknüpft:

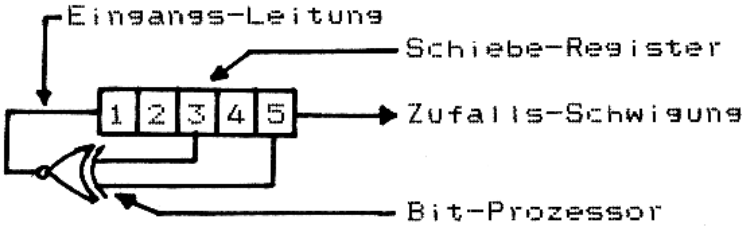


Abbildung 10.4:
5 Bit Poly-Zähler

Der Bit-Prozessor erhält neue Werte, indem bestimmte Bits desselben (im oberen Fall die Bits 3 und 5) auf eine für diese Besprechung unmaßgebliche Weise verknüpft werden. Der Verknüpfungswert wird im Bit 1 des Prozessors gespeichert.

Die Werte, die diese Poly-Zähler ausgeben, sind im Prinzip nicht „zufällig”, da sich die Bitfolge der Ausgabe nach einer bestimmten Zeit wiederholt. Diese Wiederholungs-Rate hängt von der Größe des Registers (Anzahl der Bits) ab; je größer das Register, desto seltener wiederholt sich die Folge.

Nach dieser Einführung in die Funktionsweise der Poly-Zähler können wir uns jetzt der Erzeugung von Verzerrung mit dem ATARI”TM” Personal Computer System zuwenden.

Die Verzerrung wird beim ATARI Computer durch Benutzung dieser zufälligen Schwingungen vom Poly-Zähler erreicht, die sich wiederum in einer „Selektions-Schaltung" befinden. Diese Schaltung ist in Wahrheit ein digitaler Vergleicher, wird aber durch den oben genannten Begriff besser beschrieben. Aus diesem Grunde wird in diesem Kapitel der Ausdruck „Selektions-Schaltung" verwendet.

Die einzigen Schwingungen, die durch die Selektions-Schaltung gelangen, sind die mit den Zufalls-Schwingungen übereinstimmenden. Dadurch werden verschiedene Impulse „zufällig" eleminiert. Die folgende Abbildung veranschaulicht diese Methode. Die gestrichelten Linien zeigen übereinstimmende Impulse:

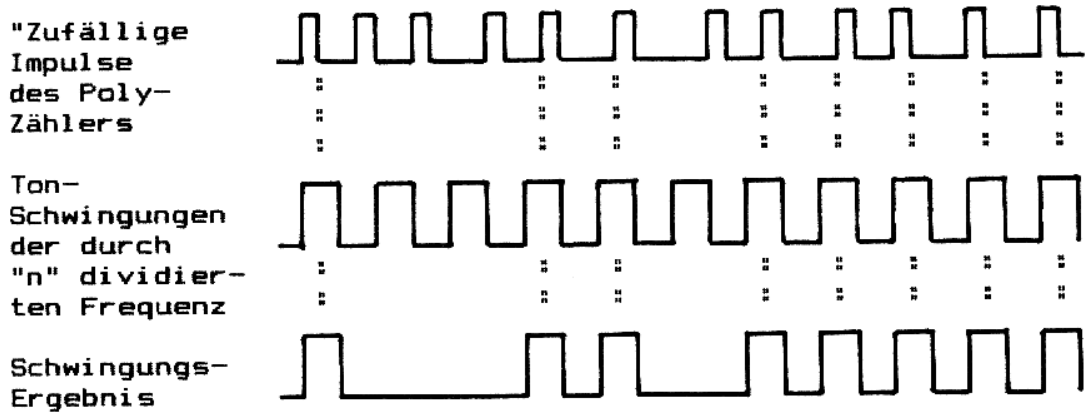


Abbildung 10.5:
Funktion der Selektions-Schaltung zum Einbringen von Verzerrung

Der End-Effekt ist dabei folgender: einige Impulse der Frequenz, die durch eine Schaltung dividiert wird, werden gelöscht (siehe Abschnitt dem Kapitels -AUDF1-4. Offensichtlich ändert sich die Wirkung des Tones hörbar, wenn verschiedene Impulse gelöscht werden. Auf diese Weise wird die Verzerrung eines Tones erreicht.

Da Poly-Zähler ihre Bit-Folge wiederholen, wird auch die Folge der Impulse wiederholt. Außerdem wird eine Note auch dieses Muster besitzen, sofern sie verzerrt und dieser Selektions-Schaltkreis benutzt wird. Diese Methode gestattet dem Programmierer die Erzeugung von Motoren-, Summ- und anderen Geräuschen, die sich wiederholende Folgen besitzen.

Der ATARI Computer besitzt drei Poly-Zähler unterschiedlicher Größe, wodurch verschiedene Zufalls-Stufen vorhanden sind. Die kürzeren Poly-Zähler (4 & 5 Bit lang) wiederholen ihre Sequenz oft genug, um Geräusche zu erzeugen, die rasch ansteigen und abfallen, wogegen der größere Poly-Zähler (17 Bit groß) zu lange braucht, um sich zu wiederholen, so daß im Prinzip kein Muster bei der Verzerrung vorliegt. Dieses 17 Bit-Register kann zur Erzeugung von Rauch-, Explosions- und allen anderen Geräuschen benutzt werden, bei denen ein zufälliges „Knistern" und „Krachen" gewünscht wird. Dieses Register ist ebenfalls unregelmäßig genug zum Erzeugen von weißem Rauschen (ein Ton-Begriff, der einen zischenden Klang beschreibt). Ein Beispiel hierfür wäre:

```
SOUND 0,100,8,
POKE 53768,64
```

Eine Poly-Zähler Option von 9 Bit liefert einen vernünftigen Kompromiss zwischen den kleinen und großen Poly-Registern (siehe Abschnitt I, Besprechung von AUDCTL).

Jeder Audio-Kanal liefert 6 verschiedene Kombinationsmöglichkeiten der drei Poly-Zähler:

AUDC1-4

Bit:	7	6	5	4	3	2	1	0

0 0 0	dividiere Takt durch Frequenz, wähle durch Benutzung von 5-Bit-Poly, danach 17-Bit-Poly, dividiere durch 2
0 x 1	dividiere Takt durch Frequenz, wähle durch Benutzung von 5-Bit-Poly, danach dividiere durch 2
0 1 0	dividiere Takt durch Frequenz, wähle durch Benutzung von 5-Bit-Poly, danach 4-Bit-Poly, dividiere durch 2
1 0 0	dividiere Takt durch Frequenz, wähle durch Benutzung von 17-Bit-Poly, danach dividiere durch 2
1 x 1	dividiere Takt durch Frequenz, danach dividiere durch 2 (keine Poly-Zähler)
1 1 0	dividiere Takt durch Frequenz, wähle durch Benutzung von 4-Bit-Poly, danach dividiere durch 2

Anmerkung: „Takt” bezeichnet die Basis-Frequenz -- siehe Besprechung von AUDCTL in Teil 1.

Ein „x” bedeutet, daß es nicht maßgeblich ist, ob dieses Bit

gesetzt ist oder nicht. Die in 10.7 gezeigte Struktur verdeutlicht, warum dieses der Fall ist

Abbildung 10.6:
Mögliche Kombinationen der Poly-Zähler

Die oberen Bits von AUDC1-4 kontrollieren 3 Schalter im Audio-Schaltkreis, der in der folgenden Abbildung dargestellt wird. Dieses Diagramm verdeutlicht, warum die Tafel in Abbildung 10.6 ihre Form hat:

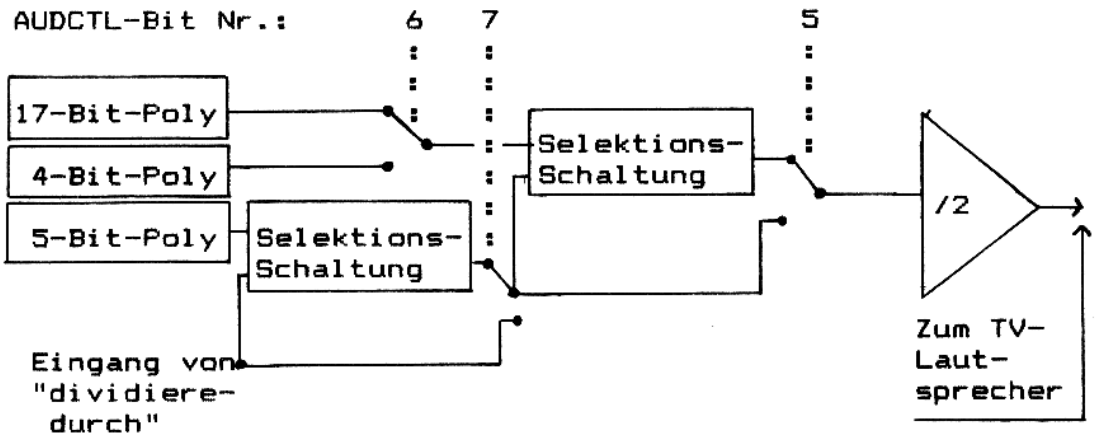


Abbildung 10.7:
AUDC1-4 Block-Diagramm

Jede Kombination der Poly-Zähler liefert einen anderen Klang. Sucht der Programmierer einen bestimmten Sound, sollte er jede Kombination der Poly-Zähler in mehrern Frequenzen durchprobieren, da sich ein bestimmtes Rauschen bei unterschiedlichen Frequenzen ganz anders anhören kann. Die folgende Abbildung zeigt eine grobe Übersicht, damit der Leser einen Anhaltspunkt hatt:

AUDC1-4

7	6	5	4	3	2	1	0

	tiefere Freq.	Mittlere Freq.	Höhere Freq.
0 0 0	Geigerzähler/tobendes Feuer/stürmende Luft/ Dampf		
1 x 1	Masch.-Bew./Wagen im Leerlauf/ E-Motor/Transformator		
0 1 0	ruhiges Feuer/Arbeitendes Auto/Wag. M Fehlzündungen		
1 0 0	zusammenstürz. Gebäude/Radiostörungen/Wasserfall		
1 x 1	reine Töne auf der ganzen Skala		
1 1 0	Flugzeug	Rasenmäher	Rasierapperat

Abbildung 10.9
Durch verschiedenartige Kombination von
Verzerrung und Frequenz erzeugte Geräusche

Zusammenfassung: Ein Schieberegister wird als Hauptbestandteil eines Poly-Zählers benutzt, welche wiederum zum Erzeugen von „zufälligen" Impulsen verwendet wird. Mit den zufälligen Impulsen werden bestimmte Tonschwingungen gelöscht, wodurch der Ausgangston eines Kanals verzerrt wird. Die oberen Bit (5, 6 und 7) von AUDC1-4 verändern 3

Schalter, die zum Auswählen der Poly-Zähler für die Verzerrung benutzt werden.

NUR LAUTSTÄRKE: Das 4. Bit von AUDC1-4 legt einen Nur-Lautstärke Modus fest. Ist dieses Bit gesetzt, werden die Lautstärke-Bits (AUDC1-4 Bits 0 bis 3) als Lautstärke zum Fernseh-Lautsprecher gesendet: durch dieses Bit wird keine Frequenz mehr übertragen.

Um diese Operations-Art zu verstehen, muß der Leser die Funktionsweise eines Lautsprechers kennen, der einen Impuls erhält.

Jeder Lautsprecher besitzt einen beweglichen Metallbolzen. Die Position dieses Bolzens ist jederzeit proportional der Stromstärke, die vom Computer ausgesendet wird. Beträgt die Spannung z.B. Null Volt, so befindet sich der Bolzen in Ruhestellung. Durch diese Bewegung des Bolzens wird die Lautsprecher-Membran in Bewegung gesetzt, was wiederum die Luft zum Schwingen bringt. Das menschliche Ohr erfaßt diese Schwingungen schließlich als Ton.

Die für einen Impuls getroffene Defininition besagt, daß ein solcher aus einem plötzlichen Spannungs-Anstieg, gefolgt von einem plötzlichen Abfall desselben besteht. Wird ein solcher Impuls zum Fernseh-Lautsprecher gesendet, so erzeugt dieser eine einzelne Luftschwingung, die das Ohr als kurzes "PLOPP"-Geräusch wahrnimmt. Die folgende Anweisung produziert einen solchen Impuls, der zum Lautsprecher des Fernsehgerätes gesendet wird:

```
POKE 53761,31:POKE 53761,16
```

Eine Reihe von Impulsen (= Welle), die zum Lautsprecher gesendet werden, erzeugen eine gleichmäßige Bewegung des Bolzens, wodurch ein durchgehendes Summen zu hören ist. Auf diese Weise liefert der Computer Töne und Geräusche.

Ein Impuls fällt nicht automatisch auf Null ab, sondern bleibt konstant, bis der Abfall durch das Programm oder eine Anweisung bewirkt wird. Ein Programm muß, um ein Geräusch oder einen Ton zu erzeugen, den Impuls oft genug verändern. Geben Sie bitte folgende Befehle in den Computer ein und achten Sie nach jedem einzelnen auf das erzeugte Geräusch:

```
POKE 53761,31
POKE 53761,31
```

Bei der Ausführung der ersten Anweisung ist, wie sicherlich vermutet, ein Knacken vernehmbar; der Metallbolzen wird nach

außen gedrückt und es entsteht eine Luftbewegung. Bei der Ausführung des zweiten Befehls ist nichts mehr zu hören, da der Bolzen bereits in der entsprechenden Position steht; die Luft wird nicht mehr bewegt.

```
POKE 53761,16
POKE 53761,16
```

Genau wie vorher ist beim ersten Befehl ein Knacken zu hören, weil der Metallbolzen wieder in den Lautsprecher zurückgezogen wird. Auch hier ist bei der Ausführung der zweiten Anweisung nichts zu vernehmen, da sich der Bolzen bereits in der angesprochenen Position befindet.

Die Lautstärke-Bits gestatten also die absolute Kontrolle über die Bolzenposition. Obwohl die oberen Beispiele nur binärer Natur sind (entweder alles eingeschaltet oder alles ausgeschaltet), ist der Programmierer in der Bearbeitung des Lautsprechers nicht eingeschränkt. Der Lautsprecher kann tatsächlich auf 16 verschiedene Positionen gebracht werden.

Es ist möglich eine Sägezahn-Schwingung zu erzeugen (womit die von Blechinstrumenten hervorgerufenen Schwingungen bezeichnet werden), indem eine Reihe von Lautstärken gesendet wird, die sich ständig wiederholt (z.B. Lautstärken 8, 9, 10, 9, 8, 7, 6, 5, 6, 7 und schließlich wieder 8. BASIC ist für einen solchen Effekt zu langsam.) In diesem Beispiel werden nur 7 von 16 möglichen Bolzenstellungen benutzt. (Das angegebene Programm für die Tonerzeugung mit Maschinensprache würde in der Tat eine Sägezahn-Schwingung liefern, sofern die Tabelle für die Schwingungsdauer nur Einsen enthalten würde).

Durch schnelles Ändern der Lautstärke kann also nahezu jede Schwingungsform erreicht werden. Der Computer könnte klar verständlich „Hallo” sagen.

Hat der Leser sich noch nicht mit Tonerzeugung, Schwingungsformen usw. beschäftigt, so erscheint die Benutzung dieser Lautstärke-Bits ziemlich kompliziert. In einem solchen Fall sollte der Leser sich mit Hilfe von Literatur über diese Dinge informieren.

In Teil II wird die Besprechung dieses Bits fortgesetzt.

AUDCTL

Zusätzlich zu den unabhängigen Kontrollbytes für die einzelnen Kanäle (AUDC1-4) gibt es ein Options-Byte

(AUDCTL), welches alle Kanäle beeinflußt. Jedem Bit in AUDCTL ist eine spezielle Funktion zugeordnet:

AUDCTL ($D208 = 53768)

BIT	Ist dieses Bit gesetzt, dann...
0	wird der Haupttakt von 64 kHz auf 15 kHz umgeschaltet.
1	wird ein High-Paß-Filter in Kanal 2 eingefügt, der durch Kanal 4 getaktet wird.
2	wird ein High-Paß Filter in Kanal 1 eingefügt, der durch Kanal 3 getaktet wird.
3	werden die Kanäle 4 und 3 verbunden, (16 Bit-Dauerstellung)
4	werden die Kanäle 2 und 1 verbunden (16 Bit-Dauerstellung).
5	wird Kanal 3 mit 1,79 MHz getaktet.
6	wird Kanal 1 mit 1,79 MHz getaktet.
7	wird der 17 Bit Poly-Zähler in einen 9 Bit Poly-Zähler umgewandelt.

Abbildung 10.9:
Bit-Zuordnung in AUDCTL

TAKT: Bevor wir mit der Erklärung der AUDCTL-Optionen fortfahren, muß der Leser die Bedeutung des Begriffes Takt verstehen. Im allgemeinen ist ein Takt eine Reihe von Impulsen, die zum synchronisieren der Millionen internen Operationen im Computer benutzt wird. Ein Takt ist eine gleichmäßige Folge von Impulsen, wobei jeder Impuls dem Computer mitteilt, daß er die nächste Operation ausführen kann.

Im Abschnitt zum Begriff Frequenz wurde gesagt, das ein Frequenz-Teiler nur jeden n-ten Eingangs-Impüls einen Impuls wieder ausgibt. Diese Eingangs-Impulse sind der Takt.

In einem Computer werden mehrere Takte benutzt. AUDCTL gestattet dem Benutzer nun, den als Eingang verwendeten Takt zu ändern, wodurch sich eine langsamere oder schnellere Taktgeschwindigkeit ergibt. Durch Ändern des Eingangstaktes ändert sich logischerweise auch der Ausgangstakt proportional.

Man stelle sich einen Takt mit einer Frequenz von 15 kHz vor, wobei das Frequenz-Register so gesetzt ist, daß es durch 5 dividiert.Die durch die Teiler-Schaltung ausgegebene Frequenz wäre 3 kHz. Wird aber nun der Takt (=Eingangs-Frequenz) z.B. auf 40 kHz geändert, ohne daß das

Frequenz-Register geändert wird, so würde der Teiler-Schaltkreis weiterhin bei jedem 5. Eingangsimpuls einen Impuls ausgeben. Die Ausgabe erfolgt also schneller, mit dem Ergebnis, daß die Ausgangsfrequenz auf 8 kHz ansteigt.

Die Formel für die Ausgangs-Frequahz ist sehr einfach:

$$\text{Ausgangs-Frequenz} = \frac{\text{Takt-Frequenz}}{N}$$

Die Ausgangsfrequenz ist also direkt proportional zur Eingangs-Frequenz.

Wird die Taktfrequenz erhöht, so wird alles, was auf diese angewiesen ist, mit geringerer Verzögerung ausgeführt. Im Falle der Audio-Schaltkreise des ATARI 400/8OO"TM" bedeutet eine Erhöhung der Taktfrequenz eine dichtere Folge von Tonimpulsen, die zum Fernseh-Lautsprecher gesendet werden;, der ausgegebene Ton wird höher (siehe Teil 1, Abschnitt über Frequenzen).

Das oben Besagte gilt ebenfalls für den gegenteiligen Fall - wird die Taktfrequenz erniedrigt - so hat dieses eine weniger dichte Reihe von Tonimpulsen zur Folge, wodurch ein tieferer Ton zustande kommt.

15 kHz-Option: Geben Sie bitte folgendes ein:

```
SOUND 0,128,10,9    mittlerer Ton
POKE 53768,1        AUDCTL 15 kHz-Option
```

Wie in diesem Abschnitt zum Takt erklärt, bewirkt eine Änderung der Zeitbasis eine proportionale Änderung der hörbaren Frequenz. In diesem Falle bewirkt das Speichern einer 64 in AUDCTL, daß für Kanal 1 eine 1,79 MHz-Zeitbasis anstelle einer 64 kHz-Zeitbasis verwendet wird. Wie der Leser wahrscheinlich aus dem vorangegangenen Abschitt gefolgert hat, verursacht das POKEn die Erhöhung eines Tones. Dieses ist eine sehr schwerwiegende Änderung, da 1,79 MHz fast 30 mal schneller als 64 kHz ist.

Das Setzen von AUDCTL-Bit 5 zwingt Kanal 3 zur Benutzung vom 1,79 MHz anstelle des 64 kHz-Taktes.

```
SOUND 2,255,10,8    Kanal 3 einschalten, tiefer Ton
```

POKE 53768,32 Setzen von AUDCTL-Bit 5

Diese Optionen erweitern den Bereich der zu erzeugenden Frequenzen bis über die Grenzen hinaus, die das menschliche Ohr wahrnehmen kann (der Mensch kann Frequenzen bis zu 20 kHz hören).

16 BIT-OPTIONEN: Die Bits Nr. 3 und 4 von AUDCTL gestatten das Zusammenfügen von zwei Kanälen zu einem einzigen, der dann eine größere dynamische Reichweite besitzt. Arbeiten die Kanäle unabhängig voneinander, so haben sie jeweils einen Frequenzbereich von 0 bis 255 (8-Bit-dividiert-durch-N-Möglichkeit). Das Zusammenschalten von zwei Kanälen gestattet einen Frequenzbereich von 0 bis 65535 (16 Bit-dividiert-durch-N-Möglichkeit). In diesem Modus ist es möglich die Ausgabe-Frequenz auf einen einzelnen Impuls zu reduzieren, wovon mehrere jeweils durch einige Sekunden getrennt werden. Das folgende Programm benutzt zwei Kanäle im 16 Bit-Modus, wobei zwei Paddles zur Frequenz-Eingabe verwendet werden:

```
10 SOUND 0,0,0,0:REM Sound initialisieren
20 POKE 53768,80:REM Takt von 1,79 für Kanal 1 & 2
30 POKE 53761,160:POKE 53763,168:REM Kanal 1 aus, K. 2 ein
40 POKE 53760,PADDLE(0):POKE 53762,PADDLE(l):REM regeln
50 GOTO 40:REM der Frequenzen mit PADDELS
```

Das rechte Paddle stellt den Ton grob, das linke den Ton fein ein.

Das Programm setzt als erstes die Bits Nr. 4 & 6 von AUDCTL. Dieses bedeutet: "Wähle einen Takt von 1,79 MHz für Kanal 1 und verbinde Kanal 2 mit Kanal l." Sobald dies geschehen ist, werden die beiden 8-Bit-Frequenz-Register der einzelnen Kanäle zusammengezogen, um ein 16 Bit-Frequenz-Register zu bilden. Dieses 16 Bit-Register gibt die Zahl N, die zum Dividieren des Eingangstaktes verwendet wird.

Als nächstes wird die Lautstärke von Kanal 1 auf Null gesetzt, weil die Ausgabe von diesem Kanal nur Bedeutung für Kanal zwei besitzt. Wenn diese Kanäle in den 16 Bit-Modus gebracht werden, verursachen interne Änderungen, daß die zwei Kanäle wie ein einziger (16 Bits) behandelt wird. Diese Änderungen schalten die Ausgabe durch Kanal 1 ab.

Das Frequenzregister von Kanal 1 wird als Fein- bzw. niederwertiges Byte der Tonerzeugung benutzt. Das Frequenzregister von Kanal 2 dient entsprechend als Grob- oder höherwertiges Byte. So bewirkt z.B. das Setzen einer 1 in Frequenzregister von Kanal 1, daß durch 1 dividiert wird. Dagegen hat ein Setzen einer in Frequenzregister von Kanal 2

eine Division durch 256 zur Folge. Setzt man eine 1 in die beiden Register, so wird durch 257 dividiert.

Das 3. Bit von AUDCTL kann zur entsprechend verwendeten Kombination von Kanal 3 und 4 benutzt werden.

Die 16 Bit-Option ist in Programmen nützlich, bei denen eine feinere Kontrolle der Frequenzen gewünscht bzw. erforderlich ist. Diese feinere Kontrolle wird durch Auswahl des 1,79 MHz-Taktes und der 16-Bit-Option, wie in Zeile 20 des oberen Programms gezeigt, erreicht.

Um den Grund für die feineren Abstufungen der einzelnen Frequenzhöhen zu verstehen, muß der Leser wissen, was mit der Ausgabe-Frequenz geschieht, wenn der Wert des Frequenz-Registers geändert wird. Als erstes ein konkretes Beispiel: dividiert das Frequenz-Register durch 1 und beträgt die Eingangs-Frequenz 1 kHz, so liegt die Ausgabe-Frequenz ebenfalls bei 1 kHz. Wenn der Wert des Registers nun um 1 erhöht wird, also durch 2 dividiert wird, dann wird eine Frequenz von 500Hz ausgegeben. Dieses entspricht einem Abfall um die Hälfte. Wird er Wert des Frequenz-Registers nochmals um 1 erhöht, so beträgt die Ausgabe-Frequenz 333 Hz der Abfall nur noch 167 Hz. Die zweite Inkrementierung des Frequenz-Register-Wertes hat also einen geringeren Abfall der Ausgabe zur Folge, als die erste. Daraus läßt sich schließen: fährt man fort, den Wert des Frequenz-Registers zu erhöhen, so hat dieses jedesmal einen geringeren Frequenz-Abfall als beim vorigen Mal zur Folge.

Die Grundidee hierbei ist: je größer die Zahl N im Frequenz-Register ist, umso geringer ist der Unterschied zwischen den einzelnen Ausgabe-Frequenzen, wenn N sich ändert.

Wird nun ein 16 Bit-Kanal mit einem Takt von 1,79 MHz durch einen Wert von mehreren Tausend geteilt, so hat dieses die gleiche Ausgabe zur Folge, wie ein 8 Bit-Kanal des Taktes 64 kHz, der durch mehrere Hundert geteilt wird. Der 16 Bit-Kanal gestattet eine feinere Abstufung zwischen den einzelnen Frequenzen, da jedes Erhöhen des Frequenz-Register-Wertes einen geringeren Effekt hat.

Das folgende Beispielprogramm arbeitet mit 16 Bit-Kanälen. Geben Sie es bitte einmal ein und experimentieren Sie damit, indem Sie die Wertkombinationen der letzten 4 POKE-Befehle ändern.

```
SOUND 0,0,0,0
```

```
POKE 53768,24
POKE 53761,168
POKE 53763,168
POKE 53765,168
POKE 53767,168
POKE 53760,240:REM ändern Sie die folgenden 4
POKE 53764,252:REM Speicherstellen mit den POKE-
POKE 53762,28:REM Befehlen.
POKE 53766,49
```

HIGH-PAß-FILTER: mit den Bits 1 & 2 von AUDCTL werden die High-Paß-Filter der Kanäle 2 und 1 kontrolliert. Ein High-Paß Filter gestattet nur hohen Frequenzen den Durchgang.

Im Falle dieser High-Paß Filter sind hohe Frequenzen als die definiert, welche höher liegen, als der Takt. Der Takt ist hierbei die Ausgabe eines anderen Kanals.

Wenn Kanal 3 z.B. den "MUH"-Laut einer Kuh ausgibt und Bit 2 von AUDCTL gesetzt wurde, dann werden nur Töne ausgegeben, deren Frequenz über der des "MUH"s liegen. Alles tiefer liegende wird ausgefiltert.

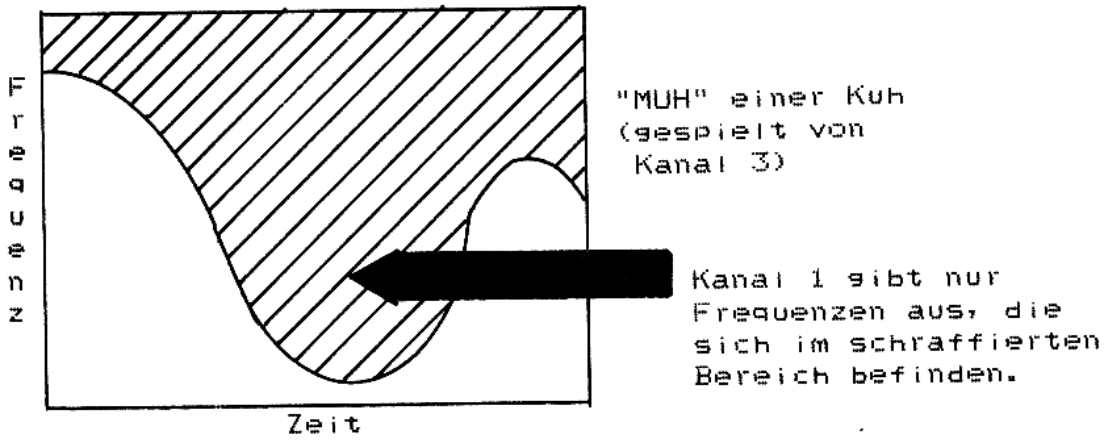


Abbildung 10.10:
Diagramm der Wirkung eines High-Paß Filters der in Kanal 1 eingesetzt und durch Kanal 3 getaktet wird.

Der Filter ist in Realzeit programmierbar, da dieses ein anderer Kanal ist, der jederzeit gändert werden kann. Dem Programmierer wird hierdurch ein großes Feld von Anwendungs-Möglichkeiten eröffnet.

Die Filter werden meistens zum Erzeugen spezieller Toneffekte verwendet. Geben Sie bitte folgendes Beispiel ein:

```
SOUND 0,0,0,0
POKE 53768,4
POKE 53761,168:POKE 53765,168
POKE 53760,240:POKE 53764,127
POKE 53760,240:POKE 53764,127
```

9 BIT POLY-UMWANDLUNG: Bit 7 von AUDCTL schaltet den 17 Bit Poly-Zähler in einen 9 Bit Poly-Zähler um. In dem Abschnitt über diese Zähler wurde erklärt, daß sich eine Verzerrungs-Folge umso öfter wiederholt, je kürzer der Poly-Zähler ist. Daraus kann man schließen, daß die Änderung des 17 Bit Poly-Zählers in einen mit 9 Bits einen klarer zu erkennenden Verzerrungseffekt hat.

Geben Sie bitte folgendes Beispiel zur 9 Bit Poly-Option ein und hören Sie genau hin, wenn der POKE-Befehl ausgeführt wird:

```
SOUND 0,80,8,8:REM 17-Bit-Poly wird benutzt
POKE 53768,128:REM Änderung in 9-Bit-Poly-Zähler
```

II. SOFTWARE-TECHNIKEN FüR DIE GERÄUSCH- UND TONERZEUGUNG

Es gibt zwei Möglichkeiten das Sound-System des ATARI Computers zu benutzen: die statische und die dynamische. Statische Tonerzeugung bedeutet, daß der Programmierer einen oder mehrere Tonkanäle einschaltet, eine gewisse Zeit wartet und sie dann wieder ausschaltet. Dynamisch bedeutet, daß die Tongeneratoren ständig neue Werte erhalten, während das Programm ausgeführt wird. Ein Beispiel hierfür:

```
statisch:
SOUND 0,120,8,8

dynamisch:
FOR X=0 TO 255:SOUND 0,X,8,8:NEXT X
```

Statische Tonerzeugung

Der statische Ton ist in seiner Wirkung sehr beschränkt. Meistens sind die einzigen Töne, die erzeugt werden können,

nur Klick-, Piep- oder Summgeräusche. Natürlich gibt es auch Ausnahmen. Zwei Beispiele sind die in Abschitt I dieses Kapitels angegebenen Programme (Teile über High-Paß Filter und 16-Bit Tonerzeugung). Ein anderes, einfacheres Beispiel ist die Benutzung von 2 Tonkanälen auf die folgende Weise:

```
SOUND 0,255,10,8
SOUND 1,254,10,8
```

Der seltsame Effekt entsteht durch den kleinen Frequenz-Unterschied der Töne, wodurch Schwingungen erzeugt werden, wie im folgenden Diagramm dargestellt. Es zeigt zwei von den 2 Tonkanälen ausgegebene Sinuskurve und ihre Summe. Die Summenkurve zeigt das ungewöhnliche Interferenz-Muster, das durch die Addition der beiden Kanäle entsteht.

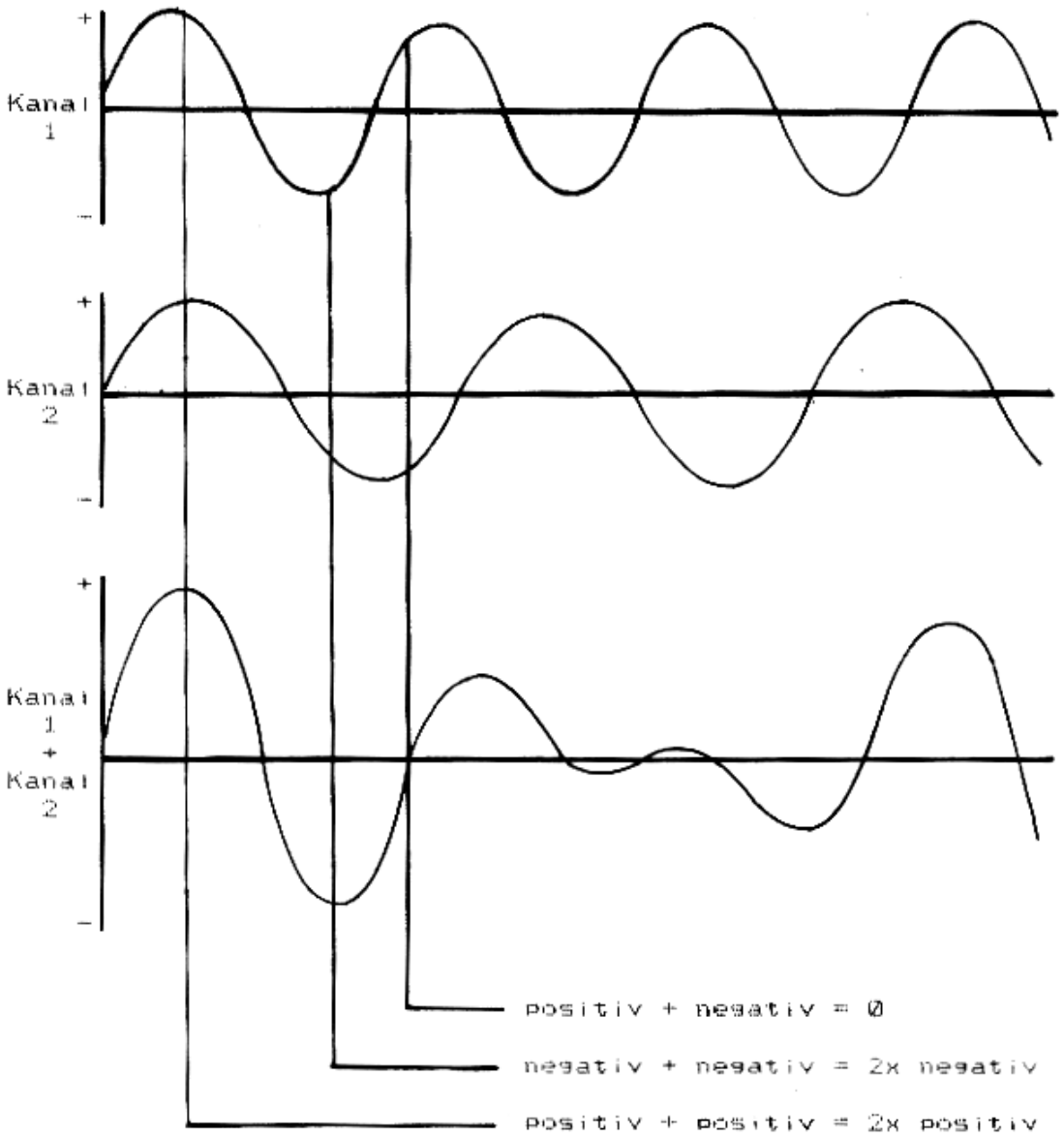


Abbildung 10.11:
Zwei Sinuswellen unterschiedlicher Frequenz und ihre Summe

Bevor die Tonkanäle zum Fernseh-Lautsprecher gesendet werden, werden sie gemischt. Dieser Mischvorgang entspricht dem im menschlichen Ohr; es ist eine einfache Adddition.

Abbildung 10.11 veranschaulicht, daß die Schwingungen sich an einigen Stellen so überlagern, daß sie sich gegenseitig verstärken, wogegen sie sich an einigen Stellen gegenseitig aufheben. Die Addition von Schwingungen, deren Schwingungsrichtung gleich verläuft, bedeutet also eine

Verstärkung, eine Addition zweier gegeneinanderlaufender Schwingungen eine Schwächung der Lautstärke.

Die dargestellte Kurve der Addition zeigt diesen Vorgang. Zum Ende des Graphen hin steigt die Lautstärke an, da sich die beiden Schwingungen von Kanal 1 und 2 ergänzen; die Lautstärke wird nahezu verdoppelt. In der Mitte des Graphen stehen die beiden Schwingungen gegeinander, daraus resultiert eine fast vollständige Löschung des Tones. Ein interessantes Programm wäre, die Schwingungen von 2, 3 oder 4 Tonkanälen und deren Additionsergebnis darzustellen.

Je geringer der Unterschied in der Frequenz zwischen den einzelnen Kanälen ist, umso länger dauert es, bevor sich die Ergebnis-Schwingung wiederholt. Um dieses zu verstehen, sollte der Leser sich noch einige Graphen wie in Abbildung 10.11 zeichnen und die Wechselwirkungen studieren. Geben Sie nun bitte folgendes Beispiel in den Computer ein:

```
SOUND 0,255,10,8
SOUND 1,254,10,8
SOUND 1,253,10,8
SOUND 1,252,10,8
```

Ein anderes Beispiel wäre:

```
SOUND 0,254,10,8
SOUND 1,127,10,8
```

Dynamische Tonerzeugung

Im allgemeinen muß jeder Ton, der nicht ein einfaches Klicken oder Piepen ist, dynamisch erzeugt werden. Dem Programmierer steht die Wahl zwischen drei Arten der dynamischen Tonerzeugung frei: Sound in BASIC, 50-Hz-Interrupt oder Sound in Maschinensprache.

SOUND IN BASIC: BASIC ist in gewisser Hinsicht in den Möglichkeiten der Tonerzeugung beschränkt. Wie der Leser vielleicht schon bemerkt hat, löscht jede SOUND-Anweisung eine geänderte AUDCTL-Einstellung. Dieses Problem kann umgangen werden, indem anstelle des SOUND-Befehls direkt mit POKE gearbeitet wird. Das Programm in Teil I (16 Bit-Optionen von AUDCTL) ist ein gutes Beispiel für diese Technik.

Außerdem besteht in BASIC ein weiteres Problem in der begrenzten Ausführungs-Geschwindigkeit. Sofern das Programm

nicht speziell auf Tonerzeugung ausgelegt wurde, reicht die Prozessorzeit im allgemeinen nicht aus, um mehr als statischen Sound zu erzeugen. Natürlich kann dieses umgangen werden, indem sämtliche anderen Aktionen gestoppt werden und das Gerät nur auf die Tongenerierrung beschränkt wird.

Eine weitere Schwierigkeit tritt dann auf, wenn der Computer Musik auf mehr als einem Kanal gleichzeitig spielen soll. Werden alle 4 Kanäle benutzt, so kann die Zeit, die zwischen Einschalten des 1. und des 4. Kanals vergeht, ausreichen um einen hörbaren Missklang zu erzeugen.

Das folgende Programm wäre eine Lösung für dieses Problem:

```
10 SOUND 0,0,0,0;DIM SIMUL$(16)
20 RESTORE 9999:X=l
25 READ Q:IF Q<>-1 THEN SIMUL$(X)=CHR$(Q):X=X+1:GOTO 25
27 RESTORE 100
30 READ Fl,Cl,F2,C2,F3,C3,F4,C4
40 IF Fl=-l THEN END
50 X=USR(ADR(SIMUL$),Fl,Cl,F2,C2,F3,C3,F4,C4)
55 FOR X=0 TO 150:NEXT X
60 GOTO 30
100 DATA 182,168,0,0,0,0,0,0
110 DATA 162,168,182,166,0,0,0,0
120 DATA 144,168,162,166,35,166,0,0
130 DATA 128,168,144,166,40,166,35,166
140 DATA 121,168,128,166,45,166,40,166
150 DATA 108,168,121,166,47,166,45,166
160 DATA 96,168,108,166,53,166,47,166
170 DATA 91,168,96,166,60,166,53,166
999 DATA -1,0,0,0,0,0,0,0
9000 REM
9010 REM Diese Datenzeilen enthalten das Mgschinen-Sprache
9020 REM Programm, das in SIMUL$ eingelesen wird.
9030 REM
9999 DATA 104,133,203,162,0,104,157,0,210,232,228,203,208,
10000 DATA 246,96,-l
```

SIMUL$ ist ein kleines Maschinensprache-Programm, das die Werte für die 4 Tonkanäle fast gleichzeitig setzt. Jedes Programm, bei dem diese Routine benutzt wird, kann die 4 Tonkanäle gleichzeitig starten.

Jedes Programm kann SIMUL$ aufrufen, indem die Werte für die SOUND-Register in die in Zeile 50 gezeigte USR-Funktion geschrieben werden. Die Parameter müssen wie gezeigt angeordnet werden, wobei der Wert für das Kontroll-Register dem für das Frequenz-Registers folgt. Diese Folge wird 1 bis

4 mal wiederholt und zwar für jeden Kanal, der gesetzt werden soll, einmal.

Das Programm in SIMUL$ hat den Vorteil, daß eine beliebige Anzahl von Kanälen (natürlich bis zu 4) gesetzt werden kann. Hierfür müssen lediglich die nicht zu benutzenden Parameter aus der USR-Funktion gelassen werden. Die folgende Anweisung zeigt die USR-Funktion, bei der nur die ersten beiden Kanäle gesetzt werden:

```
X=USR(ADR(SIMUL$),Fl,Cl,F2,C2)
```

Das SIMUL$-Programm liefert einen weiteren Vorteil gegenüber der SOUND-Anweisung in BAISC: da kein SOUND-Befehl verwendet wird, bleibt der Wert in AUDCTL erhalten.

Es gibt eine weitere Möglichkeit der Tonerzeugung, die über den BASIC-INTERPRETER nicht vollständig ausgenutzt wird. Sie besteht in der Benutzung des "Nur-Lautstärke"-Bits der 4 Kanäle. Geben Sie bitte folgendes Beispiel ein und starten Sie es:

```
SOUND 0,0,0,0
10 POKE 53761,16:POKE 53761,31:GOTO 10
```

Dieses Programm setzt das Nur-Lautstärke-Bit von Kanal 1 und moduliert die Lautstärke von 0 bis 15; dieses geschieht allerdings nur so schnell (oder langsam), wie es in BASIC eben möglich ist. Um es konkret auszudrücken: dieses Programm benutzt 100% der verfügbarn Prozessor-Zeit und erzeugt dabei nur ein tiefes Brummen.

Die beste Methode, um in BASIC komplexe Töne zu erzeugen, ohne daß der Prozessor mit Beschlag belegt wird, ist die Benutzung des 50-Hz-Interrupts. Dieser wird im nachfolgenden Abschnitt besprochen.

DER 50-HZ-INTERRUPT: Die Tonerzeugung über diesen Interrupt ist wahrscheinlich die nützlichste und praktischste Methode, aller auf dem ATARI"TM" verfügbaren.

Präzise jede 50tel Sekunde erzeugt die Hardware des Computers einen Interrupt (=Unterbrechung). Wenn dieses geschieht, verläßt der Computer zeitweilig das eigenliche Programm (z.B. BASIC oder STAR RAIDERS"TM") und führt eine sogenannte "interrupt-Service-Routine" aus. Dieses ist ein sehr kleines Programm, das speziell zum Bearbeiten solcher Interrupts vorgesehen ist. Wurde diese Routine abgearbeitet, dann folgt ein Maschinensprache-Befehl, der den Computer veranlaßt, wieder zum Hauptprogramm zurückzukehren. Alles

dieses geschieht (sofern richtig), ohne das Hauptprogramm zu beeinflussen.

Die augenblicklich noch im ATARI Personal Computer residente Interrupt-Service-Routine erhält die Timer aufrecht, übersetzt die Informationen von Kontrollern und bearbeitet die anderen Dinge, die eine regelmäßige Beachtung erfordern.

Bevor aber die Interrupt-Service-Routine wieder zum eigenlichen Programm führt, kann der Benutzer veranlassen, daß eine weitere, von letzterem geschriebene Routine abgearbeitet wird, z.B. eine Sound-Routine. Diese Möglichkeit ist ideal für die Tonerzeugung, da das Timing präzise kontrolliert wird und ein anderes Programm laufen kann, ohne auf die Sound Generatoren achten zu müssen.

Ein weiterer Vorteil dieser Routine ist ihre Vielseitigkeit. Ein Interrupt zur Tonerzeugung führt auf die gleiche Weise zu jedem Hauptprogramm zurück, egal in welcher Sprache letzteres geschrieben wurde (z.B. Assembler, BASIC, FORTH, PASCAL usw.). In der Tat werden nur geringe oder sogar keine Änderungen nötig, damit das Interrupt-Programm für Tonerzeugung mit einem anderen Hauptprogramm oder einer aneren Sprache arbeitet.

Eine "tabellen-gesteuerte" Routine liefert ein Maximum an Flexibilität und Einfachheit für solche Zwecke. "Tabellen-gesteuert" bezeichnet ein Programm, das auf Daten-Tabellen im Speicher zugreift, um die notwendigen Informationen zu erhalten. Im Falle der Tonerzeugung würde so eine Tafel die Werte für die Frequenz (und evtl. für die Kontroll-) Register enthalten. Die Routine würde einfach die Werte lesen und sie in die zugehörigen Register übertragen. Auf diese Weise könnten die Noten bis zu 50 mal in jeder Sekunde geändert werden, was für die meisten Anwendungen ausreichend sein dürfte.

Die Routine muß in Maschinen-Sprache geschrieben werden, da sie regelrecht zu einem Teil des Operating Systems wird!

Sobald ein solches Programm geschrieben und im Speicher plaziert wurde (z.B. bei Speicherstelle $600 beginnend), muß es zu einem Teil der 50-Hz-Interrupt-Routine werden. Dieses wird durch eine Technik erreicht, die als "Vektor-Änderung" bezeichnet wird und die in Anhang I ausführlich erklärt wird.

Die Speicherstellen $224 und $225 enthalten die Adresse einer kleinen Routine: XITVBL (= EXIT - Vertical - Blank - Interrupt Service-Routine = verlasse Interrupt - Service für

Vertical Blank). XITVBL wird ausgeführt, nachdem sämtliche Arbeiten zum 50-Hz-Interrupt abgeschlossen sind. Sie dient dazu den Computer wieder (wie vorangehend besprochen) auf das Hauptprogramm zurückzuführen.

Um die Sound-Routine einzusetzen, müssen folgende Schritte durchgeführt werden:

1) das Programm wird in den Speicher gebracht;

2) die letzte Anweisung muß ein JMP $E462 sein ($E462 ist die Adresse von XITVBL, damit das Hauptprogramm fortgesetzt werden kann.);

3) das X-Register wird mit dem höherwertigen Byte der Routinen-Adresse geladen (in diesem Fall eine 6);

4) das Y-Register wird mit dem niederwertigen Byte der Routinen-Adresse geladen (in diesem Fall eine 0);

5) im Akkumulator wird eine 7 gespeichert;

6) es wird ein JSR $E45C (zum Setzen der Speicherstellen $224 und $225) ausgeführt.

Die Schritte 3-6 sind lediglich zum Ändern des Wertes in $224 und $225 erforderlich. Die aufgerufene Routine heißt SETVBV (Setze die Vertical-Blank-Vektoren); sie bringt nur die Speicherstellen (siehe Anhang I).

Einmal eingebracht, arbeitet das System wie folgt, sobald ein Interrupt auftaucht:

1) die Interrupt-Routine des Computers wird ausgeführt;

2) es wird zum Benutzerprogramm gesprungen, dessen Adresse in den Speicherstellen $224 und $225 steht;

3) die Benutzer-Routine wird ausgeführt;

4) es wird zu XITVBL gesprungen;

5) XITVBL bringt den Computer wieder zum eigentlichen Programm zurück.

Zum Vergleich hier noch einmal die Aktionen, wenn keine Benutzer-Routine vorhanden ist:

1) die Interrupt-Routine des Computers wird ausgeführt;

2) es wird zur in $224 und $225 stehenden Adresse gesprungen (in diesem Falle XITVBL);

3) XITVBL bringt den Computer wieder zum eigentlichen Programm zurück.

Möchte der Leser keine eigene Routine schreiben, so kann er sich einer schon Vorhandenen bedienen. Ein BASIC-Editor gestattet die Erzeugung und Änderung von Ton-Daten, während der Leser das Ergebnis prüft. Dieser Editor ist mit einem oben beschriebenen Interrupt Sound-Generator kombiniert. Dieses Programm heißt INSONIA (**IN**terrupt **SO**u**N**d **I**nitialiser/**A**lterer = Initialisierer/Änderer für Interrupt-Sound) und wird vom ATARI"TM" auf der Diskette "Utility" angeboten.

TONERZEUGUNG IN MASCHINENSPRACHE: Durch die Benutzung von Maschinen-Sprache eröffnen sich zahlreiche Möglichkeiten der Tonerzeugung. Der Benutzer kann nun z.B. versuchen, bestimmte Musikinstrumente zu simulieren. Der Programmierer muß hierfür als erstes ein Programm schreiben, das der 50-Hz-Interrupt-Routine entspricht und tafel-gesteuert ist. Die Ausgabe dieser Routine sieht wie folgt aus:

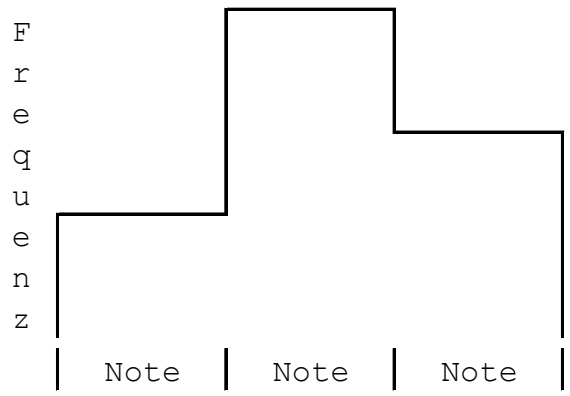


Abbildung 10.12:
Beispiel von 3 Noten, die
von einer normalen Musik-Routine
gespielt werden

Da nun mehr Verarbeitungszeit zur Verfügung steht, können wir einen Schritt weiter gehen und die Frequenz einer Note ändern, während sie gespielt wird. Mit dem Computer ist es möglich, einen beim Anschlagen einer Klaviertaste

entstehenden Klang zu simulieren. Dieses geschieht, indem sehr schnell eine bestimmt Frequenztafel gespielt wird, die z.B. folgendes Aussehen besitzt:

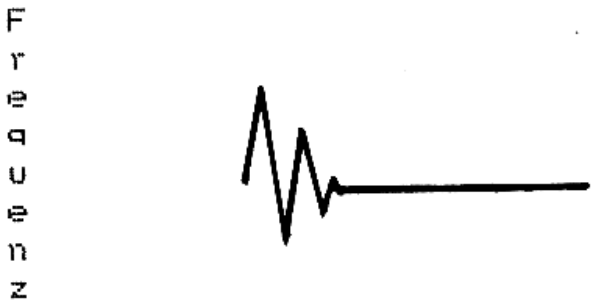


Abbildung 10.13:
Graphisch dargestellte Tafel von Frequenzen,
die dem Klang eines Klaviers nahekommen.

Die obere Tafel ist eine "Modifikations-Tafel" und die in ihr enthaltenen Daten sind eine "Klavier-Modifikation". Um ein Klavier zu simulieren, müßen die Originaltöne des Computers durch Addition der Klavier-Modifikation geändert werden. Die Note wird also während ihres Ertönens leicht verändert. Eine KlavierSimulation für die 3 Töne in Abbildung 10.12 würde wie folgt aussehen:

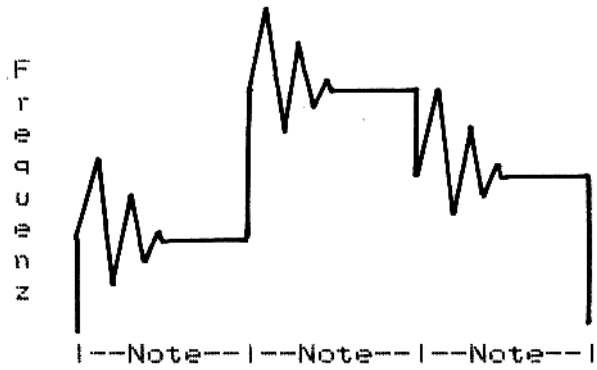


Abbildung 10.14:
Beispiel der drei Noten aus Abbildung 10.12,
die mit einer Klaviermodifikation gespielt werden

Das Ergebnis ist grundsätzlich der gleiche Sound, der auch von einer normalen Musikroutine erzeugt wird. Die Töne erhalten aber jetzt den Klang eines Klaviers anstelle eines einfachen durchgehenden Klanges.

Unglücklicherweise muß für jede Erzeugung dieses Klavierklanges sämtliche andere Verarbeitung geopfert werden. Der Sound wird nicht mehr bei jeder Note erneuert, sondern ungefähr 100 mal innerhalb jeder einzelnen Note.

NUR LAUTSTÄRKE: Es wurde schon vorangehend mit den Nur-Lautsärecke-Bits von AUDC1-4 experimentiert, wodurch ihnen eine große Leistungsfähigkeit zugeschrieben worden sind. Augenscheinlicherweise sind sie aber nicht von großem Nutzen. Dieses ist aber ein Fehlschluß, der daher rührt, daß BASIC nicht schnell genug für die effektive Benutzung dieser Bits ist. Für Maschinen-Sprache trifft dieses aber nicht zu!

Wie schon angesprochen, eröffnen diese Bists eine sehr gute Möglichkeit für fein kontrollierbare Tonerzeugung. Es ist mit diesen Bits die Erzeugung wirklicher Schwingungskurven (mit Rücksicht auf die Lautstärke-Auflösung des Computers) möglich. Anstatt einen Klavierklang zu erzeugen, kann dieser nun regelrecht kopiert werden.

Ein Nachteil des Computers hierbei ist, daß ein Instrument niemals absolut präzise simuliert werden kann. 4 Bit (16 Werte) ist keine weitreichende Lautstärken-Auflösung, was jedoch nicht heißen muß, daß ein Versuch von vornherein ausssichtslos ist. Das folgende Programm benutzt die Nur-Lautstärke Bits. Wenn Sie ein Assembler-Modul besitzen, geben Sie es bitte einmal ein starten Sie es:

```
0100 ;
0110 ; VONLY Bob Fraser 23.7.1981
0120 ;
0130 ;
0140 ; Testroutine (Nur-Lautstärke) für AUDC1-4
0150 ;
0160 ;
0170 AUDCTL=$D208
0180 AUDF1=$D200
0190 AUDC1=$D201
0200 SKCTL=$D20F
0210 ;
0220 ;
0230 *=$B0
0240 TEMPO .BYTE 1
0250 MSC .BYTE 0
0260 ;
0270 ;
0280 *=$4000
0290 LDA #0
0300 STA AUDCTL
0310 LDA #3
```

```
0320 STA SKCTL
0330 LDX #0
0340 ;
0350 LDA #0
0360 STA $D40E      VBIs loeschen
0370 STA $D20E      IRQs loeschen
0380 STA $D400      DMA loeschen
0390 ;
0400 ;
0410 ;
0420 L00 LDA DTAB,X
0430  STA MSC
0440 ;
0450  LDA VTAB,X
0460 L0 LDY TEMPO
0470  STA AUDC1
0480 L1 DEY
0490  BNE L1
0500 ;
0510 ; Dekrementieren
0520 ;
0530  DEC MSC
0540  BNE L0
0550 ;
0560 ; Neue Note
0570 ;
0580  INX
0590  CPX NC
0600  BNE L00
0610 ;
0620 ; Noten-Zeiser einfuegen
0630 ;
0640  LDX #0
0650  BEQ L00
0660 ;
0670 ;
0680 NC .BYTE       28 Noten-Zaehler
0690 ;
0700 ; Tabelle der zu spielenden Toene
0710 ;
0720 VTAB
0730  .BYTE 24,25,26,27,28,29,30,31
0740  .BYTE 30,29,28,27,26,25,24
0750  .BYTE 23,22,21,20,19,18,17
0760  .BYTE 18,19,20,21,22,23
0770 ;
0780 ; Diese Tabelle enthält die jeweils zugehörige Dauer
0790 ;
0800 DTAB
0810  .BYTE 1,1,1,2,2,2,3,6
```

```
0820  .BYTE 3,2,2,2,1,1,1
0830  .BYTE 1,1,2,2,2,3,6
0840  .BYTE 3,2,2,2,1,1
```

Überraschenderweise ist die Geschwindigkeit hier nicht das Problem. Die Kurve besteht aus fast 50 Stufen und könnte sogar in einer noch höheren Geschwindigkeit gespielt werden (ungefähr 10 kHz).

Entfernen Sie bitte die Zeilen 360 und 370 und starten Sie das Programm erneut. Das Ergebnis ist ein sehr ungleichmässiger Ton. Der Grund hierfür ist der 50-Hz-Interrupt, der im vorangegangenen Abschnitt besprochen wurde. Man kann den genauen Zeitpunkt des Interrupts durch Zuhören feststellen, da während der fraglichen Zeit sämtliche Tonerzeugung gestoppt wird.

Die Zeile 380 schaltet den Bildschirm-DMA aus. Daher erhält der Bildschirm eine feste Hintergrundfarbe, sobald das Programm ausgeführt wird. Dieses Ausschalten dient zwei Zweckenz: zum einen wird die Ausführungsgeschwindigkeit erhöht und zum anderen bleibt das Timing konstant, da der DMA in unregelmäßigen Zeitabständen Maschinenzyklen “stiehlt”. Siehe Kapitel 5 für weitere Information zum DMA.

In diesem speziellen Fall ist der erzeugte Ton eine Sinuskurve. Die Schwingung ist gleichmäßig und hört sich tatsächlich nach einer Sinuskurve an. Werden die Daten in einen Graphen übertagen, ergibt sich folgendes Bild:

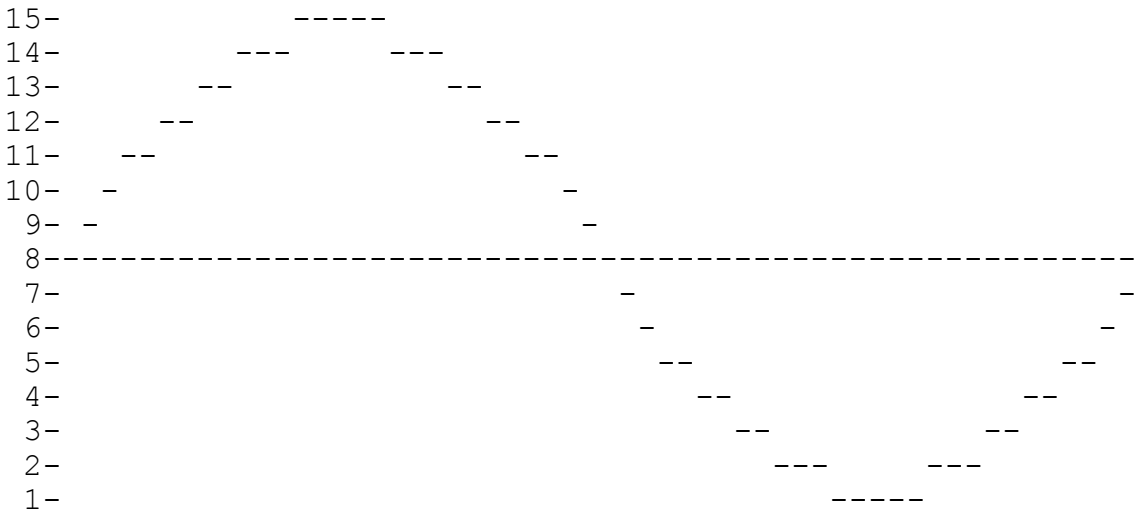


Abbildung 10.15:
Daten der Sinuskurve des vorangegangenen Programms

Mit den Sound-Möglichkeiten des ATARI Computers kann der Programmierer viel erreichen. Die Frage ist "Warum Sound?".

Die Produzenten von Filmen wissen seit langem von der Wichtigkeit der Hintergrundmusik. Das fantastische Weltraumabenteuer von George Lucas ist hierfür das beste Beispiel. Betritt der Schwarze Lord den Raum, wird dieses durch die Hintergrundmusik entsprechend begleitet, so daß ein Gefühl von Furcht und Haß beim Zuschauer erzeugt wird. Gleiches gilt für das Auftreten des Helden, wenn er die Prinzessin befreit. Die Musik erzeugt ein Gefühl der Freude, In Horror-Filmen ist die Musik eines der wichtigsten Mittel, um dem Zuschauer Furcht einzuflößen. Selbst an Stellen, an denen nichts gefährliches geschieht, kann durch die Hintergrundmusik beim Betrachter Angst erzeugt werden.

Das SPACE INVADERS"TM"-Programm hat als Eigenheit ein Pulsgeräusch, das schneller und lauter wird, je weniger Angreifer vorhanden sind. Dieses löst beim Spieler eine gewisse Anspannung aus. Wann ein Zylon bei STAR RAIDERS"TM" ein Photonen-Torpedo abfeuert, reißt der Benutzer den Steuerknüppel herum, um diesem Schuß auszuweichen.

Impressionistische Geräusche und Töne beeinflussen unseren Gemütszustand. Dieses ist möglich, da Töne immer unsere Ohren erreichen, auch wenn wir nicht hinhören. Sound eröffnet dem Programmierer daher die Möglichkeit, direkt das Gemüt des Benutzers zu erreichen.

Selbst ein langweiliges Spiel kann durch Einfügen von speziellen Ton- und Geräuscheffekten interessant gemacht werden. Dieses kann für ein Spiel oder Programm sehr vorteilhaft sein, auch wenn solche Effekte einigermaßen schwer zu entwickeln sind.

Anhang I
VBI
Vertikal-Blank-Interrupts

Der ATARI Computer besitzt eine Vielzahl von Interrupts, die für den Benutzer von großem Wert sein können. Dieser Anhang behandelt die Vertical-Blank-Interrupts. Diese Interrupt-Art ist nicht maskierbar und tritt 50 mal pro Sekunde auf. Dieses geschieht, während der Elektronenstrahl vom unteren Bildschirm zum Oberen zurückläuft.

Bei einem Verical-Blank gibt der ANTIC-Prozessor als erstes ein NMI- Signal an den 6502-Chip weiter, woraufhin dieser zu einer NMI-Service-Routine springt, welche die Ursache für den Interrupt feststellt. Ist der Interrupt ein VBI, dann schiebt der 6502 zunächst das A-, X- und Y-Register auf den Stapel. Danach springt das Programm über den Immediate-(=sofortiger)-Vertical-Blank-Vektor (VVBLKI($0222)). Dieser Vektor zeigt im Normalfall auf die Speicherstelle $E45F. Anschließend erfolgt ein Sprung über den Deferred-(=zurückgestellter)-Vertical-Blank-Vektor (VVBLKD($0224)). Dieser Vektor zeigt normalerweise auf eine Routine zum Abschließen einer Interrupt-Bearbeitung. Diese Routine ist sehr einfach aufgebaut und beginnt bei Speicherstelle $E462. Abbildung I.1 zeigt schematisch den eben beschriebenen Ablauf:

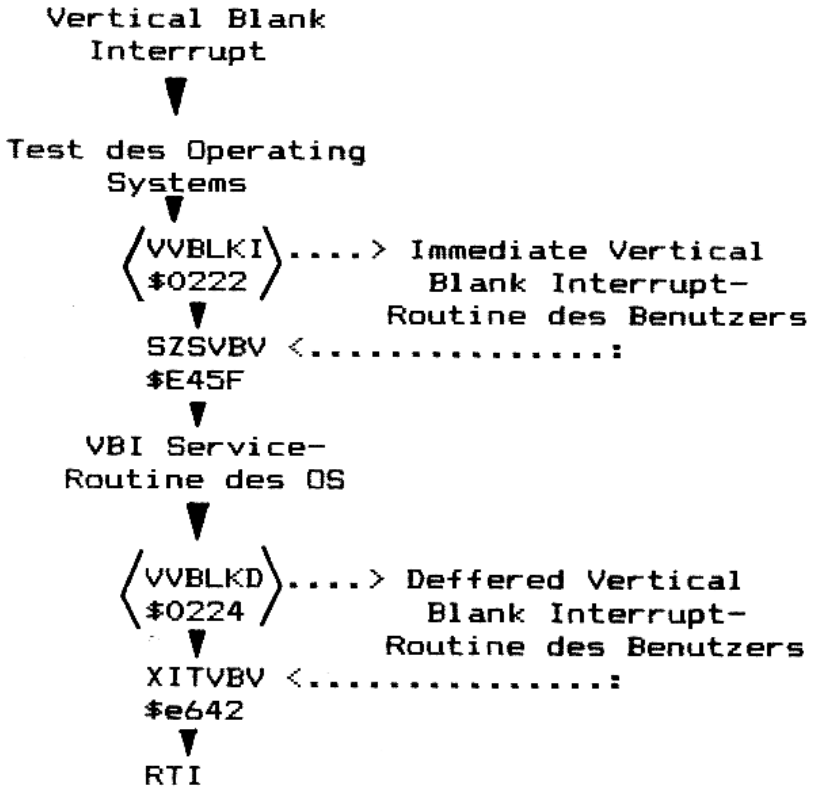


Abbildung I.1:
Normale Bearbeitung des Vertical Blank Interrupts durch das OS (und durch eventuelle Zusätze das Benutzers)

Diese beiden VBI-Vektoren sind im RAM untergebracht, damit der Programmierer die 5O-Hz-Interrupt-Routine des OS ausschalten und diesen Interrupt für seine eigenen Zwecke benutzen kann. Um dieses zu erreichen, müssen folgende Dinge getan werden: der Benutzer muß als erstes entscheiden, ob seine Routine für den Immediate oder für den Deferred VBI sein soll. In den meisten Fällen besteht hierbei kein großer Unterschied zwischen den beiden Möglichkeiten. Es gibt allerdings auch Ausnahmen. Es ist von Bedeutung, wenn die VBI-Routine in ein Register schreibt, das von der VBI-Routine des OS übertragen wird. Die Routine vom Benutzer muß hinter der des OS stehen, damit sie wirksam ist.

Im zweiten entscheidenden Fall nimmt die VBI-Routine des Benutzers soviel Zeit in Anspruch, daß die Routine des OS zeitlich verschoben wird. Werden durch die OS-Routine Graphik-Register geändert, dann kann es in diesem Fall dazu kommen, daß diese Änderung geschieht, während der Elektronenstrahl sich schon wieder in der Bildschirmmitte befindet. Ein unschönes Bild wäre die Folge. Die VBI-Routine

des Benutzers sollte in diesem Falle an der Stelle für den Deferred VBI plaziert werden.

Die maximale Grenze für Immediate VBI-Routine liegt bei 3800 Maschinenzyklen; für Deferred VBI-Routinen liegt sie bei 20.000. Natürlich werden viele dieser 20.000 Zyklen ausgeführt, während sich der Elektronenstrahl auf dem Bildschirm befindet. Graphische Operationen sollten also durch die Deferred VBI-Routine nicht ausgeführt werden. Außerdem ist die Zeit für die Ausführung eines DLIs ein Teil dieser 20.000 Maschinenzyklen. Der Leser sollte sich auch daran erinnern, daß diese 20.000 Zyklen von der gesamten Menge der Prozessor-Zyklen abgezogen werden.

Der dritte Fall, bei dem die Stellung der Benutzer-Routine von Bedeutung ist, liegt dann vor, wenn letzterer mit zeitkritischem I/0 (z.B. Disketten- oder Casseten-I/O) gemischt wird. Die VBI-Routine des OS besteht aus zwei Stufen: einer kritischen und einer unkritischen. Während eines solchen zeitkritischen I/Os schaltet das OS die VBI-Routine nach der ersten Stufe ab. Wenn der Benutzer nicht möchte, daß seine eigene VSI-Routine während des zeitkritischen I/Os abgeschaltet wird, muß er sie als Immediate VBI-Routine definieren. Die dabei aufgebauten Verzögerungen können allerdings diesen I/0 stören, ist nun aber Problem des Programmierers.

Sobald der Benutzer sich entschieden hat, ob seine VBI-Routine sofortiger oder verschobener Art sein soll, wird diese im Speicher plaziert (Page 6 ist hierfür gut geeignet). Der Benutzer muß den Abschluß seiner Routine der normalen VBI-Verarbeitung anpassen und die OS RAM-Vektoren so ändern, daß sie auf diese Routine zeigen.

Die Immediate VBI-Routine des Programmierers muß mit JMP $E45F abschließen; eine Deferred VBI-Routine mit einem $E462-Befehl. Will der Benutzer die VBI-Routine des OS umgehen (z.B. um Prozesearzeit zu gewinnen), muß seine Immediate VBI-Routine mit JMP $E462 abgeschlossen werden.

Ein für 8-Bit-Mikrocomputer typisches Problem taucht auf, wenn Vektoren während eines Interrupts geändert werden sollen. Vektoren sind 2-Byte-Werte, d.h. es sind zwei Speicherbefehle erforderlich, um sie zu ändern. Es besteht eine geringe Wahrscheinlichkeit, daß erst ein Byte eines Vektors geändert wurde, wenn Interrupt auftritt. Diese Situation würde zu einem “Absturz” des Computers führen. Es gibt für diees Problem allerdings auch eine Lösung, die in Form einer OS-Routine geliefert wird. Diese Routine heißt SETVBV und beginnt bei Speicherstelle $E45C. Das Y-Register

dem 6502-Prozessors wird als erstes mit dem niederwertigen Byte für den Vektor, das X-Register mit dem höherwertigen Byte gelanden. Im Akkumulator wird eine 6 für Immediate bzw. 7 für Deferred VBI gespeichert. Durch einen nun folgenden Sprung (JSR SETVBV) zur genannten Routine wird der Interrupt auf sichere Weise eingeschaltet und beginnt 50-mal pro Sekunde zu arbeiten.

Für 5O-Hz-Interrupts gibt es ein weites Anwendungsfeld. Als erstes können Bildschirm-Manipulationen während des Vertical Blanks ausgeführt werden, so daß diese Änderungen garantiert nicht auf dem Bildschirm geschehen. Zum Zweiten können regelmäßig wiederkehrende Bildschirm-Änderungen mit hoher Geschwindigkeit ausführt werden. In vielen Fällen, bei denen Animation erforderlich ist, kann dieses sehr vorteilhaft sein. SO müssen z.B. die Blasen im SCRAM"TM"-Programm mit gleichbleibender Geschwindigkeit bewegt werden. Sie dürfen nicht langsamer oder schneller werden, wenn der 6502 mehr bzw. weniger Aktionen ausführt. Die einzige Möglichkeit um dieses zu erreichen ist, die Animation während eines Vertical-Blanks auszuführen.

Eine andere Anwendungsmöglichkeit von VBIs liegt in der Sound-Modifikaiton. Die Sound-Register des ATARI Computers gestatten eine Kontrolle der Frequenz, der Lautstärke und der Verzerrung, aber nicht der Dauer. Diese kann über einen VBI gesteuert werden, indem zusammen mit dem Aufrufen der Routine ein Parameter gesetzt wird, der die Dauer festlegt. Die Interrupt-Routine dekrementiert dann jeden Wert, der ungleich Null ist. Diese Methode kann zum Steuern der Lautstärke eines Kanals benutzt werden, wodurch die Erzeugung von Anstiegs-, Halt- und Abfallzeit eines Tones möglich ist. Die Kontrolle von Frequenz und Verzerrung ist mit einer erweiterten Form dieser Technik möglich, wodurch interessante Effekte generiert werden können. Aufgrund einer zeitmäßigen Auflösung von einer 50stel Sekunde kann ein VBI nicht nur zur direkten Kontrolle der Lautsprecher-Amplitude eingesetzt werden.

VBIs sind auch zur Bearbeitung von Benutzereingaben sehr dienlich. Solche Eingaben erfordern wenig Berechnung, aber ständige Aufmerksamkeit. Ein VBI gestattet einem Programm 50 mal pro Sekunde eine Eingabe des Benutzers zu überprüfen, was andernfalls das Programm verlangsamt. Dieses ist eine ideale Lösung für die Aufrechterhaltung der Programm-Geschwindigkeit verbunden mit gleichzeitiger Beachtung des Benutzers.

Schließlich werden durch VBIs grobe Formen von sog. MULTI-TASKING (= gleichzeitige Bearbeitung mehrerer

Prozesse) möglich. Ein "Vordergrund"-Programm kann unter einem VBI laufen, während ein "Hintergrund"-Programm als Hauptprogramm abgearbeitet wird. Wie bei jedem Interrupt ist eine Trennung der Datensätze beider Programme erforderlich.

Anhang II
Benutzergerechte Programmierung

Das ATARI"TM" Personal Computer System ist an erster Stelle ein Verbraucher-Computer. Die Hardware wurde so entworfen, daß der Benutzer das Gerät einfach bedienen kann. Es gibt außerdem viele Hardware-Anordnungen, die den Verbraucher (sowie das Gerät) vor möglichen Fehlern schützen. Die für diesen Computer geschriebene Software sollte ebenfalls Rücksicht auf die Schwächen das Benutzers nehmen. Der durchschnittliche Verbraucher ist nicht dumm; er ist nur nicht mit den Konventionen und Traditionen der Computer-Welt vertraut. Sobald er ein Programm versteht, wird er es überwiegend richtig benutzen. Es gibt natürlich auch Ausnahmen, bei denen gelegentlich Fehler unterlaufen. Der Programmierer ist nun dafür verantwortlich, daß er den Verbraucher vor dessen Fehlern schützt.

Zum augenblicklichen Zeitpunkt ist die Ausarbeitung der Software auf einem sehr niedrigen Stand. Es werden viele Programme verkauft, die mögliche Fehler des Benutzers wenig berücksichtigen. Dies trifft zum überwiegenden Teil für Software zu, die von Amateur-Programmierern geschrieben wurde; aber auch Programme von großen Software-Häusern weisen diese Mängel auf.

Die Software-Ausarbeitung ist keine Wissenschaft, sondern eine Kunst. Sie erfordert eine große technische Geschicklichkeit, aber auch Einfühlungsvermögen des Programmierers. Als solche ist sie ein sehr subjektives Feld. Dieser Anhang offenbart daher die persönliche Einstellung und die Ideen das Autors. Eine Betrachtung sämtlicher Möglichkeiten zu diesem Thema hätte den normalen Rahmen diesem Kapitels gesprengt. Außerdem würde eine komplette Darstellung aller Punkte (Erklärungen, Vorteile, Nachteile usw.) den Leser verwirren. Der Autor zog es deshalb vor, das Thema aus seinem persönlichen Blickwinkel zu betrachten, wobei die wichtigsten Punkte berücksichtigt wurden. Das Ergebnis ist trotzdem widersprüchlich genug, um selbst den verständigen Leser zu verwirren.

DER COMPUTER ALS EMPFINDENDES WESEN

Ein Weg, die Probleme der Software-Ausarbeitung verständlich darzustellen, ist den Programmierer als einen Magier anzusehen, der ein intelligentes Wesen beschwört, welches

sich im Computer befindet. Diese Kreatur entbehrt einer physikalischen Verkörperung, ist aber in der Lage intellektuelle Arbeiten, speziell die Organisation und Verarbeitung von Information, durchzuführen. Der Benutzer eines Programms tritt nun in eine Verbindung mit diesem Wesen. Diese beiden Intelligenzen denken unterschiedlich; die Gedankengänge des Menschen sind assoziativ, einander ergänzend und diffus. Das "Denken" des Programms dagegen ist direkt, analytisch und festgelegt. Diese Unterschiede sind grundlegend und produktiv, da der Dämon sehr gut Dinge ausführen kann, zu denen der Mensch nicht in der Lage ist. Unglücklicherweise entstehen durch diese Differenzen auch Kommunikations-Hindernisse zwischen Mensch und Dämon. Sie haben einander sehr viel zu sagen, können aber aufgrund ihrer Unterschiede nicht gut miteinander kommunizieren. Das zentrale Problem bei Entwicklung guter Software muß daher die Lieferung besserer Kommunikations-Möglichkeiten für Benutzer und Computer sein. Bedauerlicherweise verwenden aber die meisten Programmierer den Großteil ihrer Konzentration auf die Erweiterung und Verbesserung der Verarbeitungs-Leistung ihrer Programme. Dieses erzeugt zwar ein “intelligenteres Wesen”, das aber keine Augen zum Sehen und keinen Mund zum Sprechen besitzt.

Aufgrund der Fähigkeiten der heutigen Computer ist es möglich, Programme auf diesen Geräten zu schreiben, die den größten Teil der Verbraucher-Bedürfnisse befriedigen. Die vorrangige Einschränkung der Möglichkeiten ist nicht länger die Taktgeschwindigkeit oder der vorhandene Speicher; es ist die “dünne Leitung”, die den Benutzer mit der “Intelligenz” im Computer verbindet. Jeder der beiden kann schnell und effizient Informationen verarbeiten; dieser Prozess wird nur durch die schmale Brücke zwischen beiden beeinträchtigt.

KOMMUNIKATION ZWISCHEN MENSCH UND MASCHINE

Wie kann nun die Verbindung zwischen den beiden “Denkern” erweitert werden? Wir müssen uns für die Beantwortung dieser Frage als erstes auf die Sprache konzentrieren, in der diese kommunizieren. Wie jede Sprache ist eine Mensch-Maschine-Sprache durch die physikalischen Möglichkeiten des Speichers in ihrer eigentlichen Bedeutung eingeschränkt. Da Mensch und Maschine sich physikalisch unterscheiden, sind auch ihre physikalischen Ausdrucks-Möglichkeiten unterschiedlich. Dieses zwingt uns (den Menschen) zum Entwurf einer Sprache, die nicht in zwei Richtungen arbeitet (wie menschliche Sprachen gewöhnlich

aufgebaut sind). Eine Maschinen-Mensch-Sprache besitzt stattdessen zwei Kanäle: einen für die Eingabe und einen für die Ausgabe. Genau wie wir menschliche Sprache untersuchen, indem wir als erstes die Töne studieren , die ein Kehlkopf erzeugen kann, beginnen wir mit der Überprüfung der physikalischen Komponenten einer Mensch-Maschinen-Brücke.

AUSGABE (VON COMPUTER ZUM MENSCHEN)

Es gibt zwei primäre Ausgabe-Kanäle vom Computer zum Benutzers: der erste ist der Fernsehbildschirm, der zweite der Fernseh-Lautsprecher. Glücklicherweise sind diese beiden Dinge sehr flexibel und eröffnen dadurch ein weites Feld von Ausdrucks-Möglichkeiten. Der Hauptteil dieses Buches beschreibt die vom Standpunkt des Computers aus erreichbaren Möglichkeiten.

Zum besseren Verständnis dieses Anhanges ist es sinnvoller, vorgenannte Aspekte vom menschlichen Standpunkt aus zu betrachten. Von den beiden Ausgabe-Kanälen (Bildschirm und Lautsprecher) ist der Bildschirm unschwer als stärkere Ausdrucks-Möglichkeit zu erkennen. Das menschliche Auge ist besser für Informations-Aufnahme ausgestattet als das Ohr. In elektrotechnischen Worten: Das Auge besitzt eine größere Bandbreite als das Ohr. Das Auge kann drei grundlegende Formen der visuellen Information verarbeiten: Formen (Shapes), Farbe und Animation.

Formen (Shapes)

Shapes sind eine gute Möglichkeit, Information an den Menschen weiterzugeben. Die direkteste Benutzung von Shapes besteht in dar Darstellung von Objekten. Wenn Ihr Programm dem Benutzer etwas mitteilen soll, so zeichnen Sie ein Bild davon. Ein Bild ist direkt und sofort verständlich.

Die zweite Anwendungs-Möglichkeit von Shapes besteht in Symbolen. Einige Dinge der menschlichen Sprache entbehren einer bildlichen Darstellung. So können z.B. Liebe, Unendlichkeit und Richtung nicht mit Bildern dargestellt werden. Sie müssen stattdessen durch Symbole, z.B. ein Herz, eine waagerecht liegende 8 oder einen Pfeil, dargestellt werden. Dies sind einige der vielen Symbole, die wir alle erkennen und verwenden. Oft ist es möglich, innerhalb eines Programms ein solches Symbol sehr schnell in seiner Bedeutung zu erkennen. Symbole sind eine Möglichkeit der kompakten Ideen-Darstellung, sie sollten aber nicht in

Situationen benutzt werden, wo ein Bild den gleichen Zweck erfüllt oder Kompaktheit erforderlich ist. Ein Symbol ist im Gegensatz zu einem Bild eine indirekte Ausdrucksart; ein Bild verkörpert eine Idee eindringlicher.

Die dritte und am häufigsten vorkommende Anwendung von Shapes besteht in der Verbindung mit Text. Ein Buchstabe ist ein Symbol; die einzelnen Buchstaben werden aneinandergereiht, um Worte zu bilden. Die dabei entstehende Sprache ist extrem ausdrucksstark. Es heißt der Wahrheit entsprechend: "Wenn Sie es nicht sagen können, so wissen Sie es auch nicht." Diese Ausdrucksstärke kostet ihren Preis: extreme Indirektheit. Das eine Idee ausdrückende Wort besitzt keine emotionelle oder empfindungsgemäße Verbindung mit der Idee. Der Mensch ist gezwungen weitläufige gedankliche "Verrenkungen" auszuführen, um das Wort zu entschlüsseln. Natürlich tun wir diesem sehr häufig, so daß wir einige Übung darin haben, folgen von Buchstaben in Ideen umzuwandeln. Wir spüren die "Anstrengung" nicht. Der springende Punkt ist, daß die Indirektheit die Ausdruckskraft und Direktheit der Kommunikation verringert.

Es gibt Menschen, die der Ansicht sind, daß Text eine höhere Kommunikationsform als graphische Darstellung ist. Die Grundlage dieser Meinung ist, daß Text die Vorstellungs-Möglichkeiten des Lesers stärker anspricht. Dieses Argument überzeugt den Autor dieses Kapitels allerdings nicht, da bei Benutzung der Vorstellungskraft sich der Leser Dinge vorstellt, die nicht in der Kommunikation selbst eingeschlossen sind. Ein gleichartiges Experiment mit Bildern würde bessere Ergebnisse liefern. Ein überzeugenderes Argument für Text ist, daß durch seine Indirektheit ein relativ großer Informations-Betrag auf kleinem Raum untergebracht werden kann. Der verfügbare Raum jeder Kommunikation erhöht den Wert der Kompaktheit von Text. Trotzdem macht es ihn (nicht(?)) höherwertiger als graphische Darstellungen; Text wird nur ökonomischer. Graphiken erfordern mehr Platz, Zeit, Speicher oder Geld, durch sie ist aber eine bessere Kommunikation möglich als durch Text. Die Wahl zwischen graphischer und Text-Darstellung ist eine Geschmacksfrage, wobei der Geschmack und damit die Entscheidung dem Konsumenten außer Frage steht. Bei einem Vergleich zwischen der Beliebtheit von Fernsehen und Radio oder Kino und Büchern siegt die graphische Darstellung.

Farbe

Farbe ist ein anderes Mittel der Informations-Übertragung. Es ist weniger ausdrucksstark als Shapes und spielt daher nur eine untergeordnete Rolle im Vergleich zu Shapes bei visuellen Darstellungen. Die häufigste Anwendung von Farbe besteht in der Unterscheidung von zwei gleichförmigen Shapes. Sie spielt außerdem eine wichtige Rolle bei Hinweis-Weitergabe an den Benutzer. Eine gute Farbgebung kann aus einem vieldeutigen Bild ein eindeutiges machen. Wenn z.B. ein Baum in einem Zeichen dargestellt werden soll, so stehen hierfür 8*8 Pixel zur Verfügung. Dieses Feld ist zu schmal, um einen erkennbaren Baum zeichnen zu können. Dieser Mangel kann aber dadurch beseitigt werden, indem das Bild grün gefärbt wird: das Bild wird leichter erkennbar. Farben sind auch zum Erregen der Aufmerksamkeit des Betrachters oder zum Signalisieren wichtigen Materials nützlich. Warme Farben ziehen die Aufmerksamkeit auf sich. Farbige Graphiken sind gefälliger als schwarz/weiße Bilder.

Animation

Der Ausdruck "Animation" wird hier zum Bezeichnen einer jeden visuellen Änderung benutzt. Animation schließt demnach sich ändernde Farben und Shapes, sich bewegenden Vorder- und Hintergrund und Objekte ein. Der vorrangige Wert von Animation besteht in der Darstellung von dynamischen Vorgängen. In der Tat ist die graphische Animation die einzige Möglichkeit, um sehr aktive Ereignisse erfolgreich darzustellen. Der Wert der Animation wird am eindringlichsten durch Spiel wie STAR RAIDERS"TM" demonstriert. Können Sie sich vorstellen, wie das Spiel ohne Animation oder sogar nur in Text wirken würde? Der Wert der Animation erstreckt sich aber auch weit über die Anwendung bei Spielen hinaus. Animation gestattet dem Designer die klare Darstellung dynamischer Vorgänge. Sie ist einer der hauptsächlichen Vorteile von Computern gegenüber Papier. Schließlich ist Animation im Bezug auf sinnliche Wahrnehmung sehr wirksam. Das menschliche Auge ist so ausgelegt, daß es stark auf Änderungen des visuellen Feldes reagiert. Animation steigert die Aufmerksamkeit der Augen und erregt die Begeisterung des Benutzers für ein Programm.

Sound

Graphische Darstellungen müssen betrachtet werden, um eine Wirkung zu haben. Sound dagegen erreicht den Benutzer, selbst wenn er nicht auf ihn achtet. Sound besitzt daher

einen großen Wert als ankündigendes oder warnendes Signal. Eine große Anzahl von Piep-, Grunz-Geräuschen und -Tönen können zur Signalisierung einer Rückkopplung an den Benutzer gesendet werden. Korrekte Aktionen können mit einem angenehmen Klingelgeräusch, falsche Aktionen mit einem Rattern beantwortet, Warnbedingungen durch ein Tuten angezeigt werden.

Die Tonerzeugung besitzt eine zweite Anwendungs-Möglichkeit in der Erzeugung realistischer Geräuscheffekte. Qualitative Toneffekte verstärken die Wirkung eines Programms, da durch die Tonerzeugung ein zweiter Informations-Kanal zum Benutzer geöffnet wird, der auch dann wirkungsvoll ist, wenn letzterer visuell beschäftigt ist.

Sound ist nicht für die Übermittlung von Fakten verwendbar: die meisten Menschen besitzen nicht die Fähigkeit kleine Frequenz-Unterschiede von zwei Tönen wahrzunehmen. Sound ist eher zum Ausdrücken von emotionalen Zuständen geeignet, da viele Menschen ein großes Assoziationsfeld zu bestimmten Melodien und Tönen haben. Eine absteigende Notenfolge deutet verschlechternde Umstände an. Ein Explosions-Geräusch steht für Zerstörung. Eine Fanfare verkündet die Ankunft einer wichtigen Person o. ähnl., Notenfolgen weithin bekannter Lieder werden sofort mit bestimmten Gefühlen in Verbindung gebracht. Beispielsweise verwendete der Autor des ENERGY CZAR"TM"-Programms einen Trauermarsch, um anzuzeigen, daß der Spieler die Energie-Situation Amerikas absolut ruiniert hat: ein Abschnitt von "Happy Days Here Again" zeigen seinen Erfolg.

EINGABE-GERÄTE (VOM MENSCHEN ZUM COMPUTER)

Es gibt auf dem ATARI"TM" Personal Computer 3 am häufigsten benutzte Eingabe-Möglichkeiten. Es sind: Tastatur, Joystick und Paddles.

Tastatur

Die Tastatur ist unschwer als leistungsstärkste dem Programmierer verfügbare Eingabe-Möglichkeit zu erkennen. Über 50 direkte Tastendrücke sind sofort zugänglich. Die Benutzung der SHIFT- und CONTROL-Tasten verdoppelt die Anzahl dieser vom Benutzer ausführbaren Eingaben. Durch die CAPS/LOWER- und ATARI-Logo-Taste wird diese Anzahl ebenfalls vergrößert. Der Benutzer kann also durch einen einzigen Tastendruck eines von 125 Kommandos anwählen. Über zwei Tastendrücke ergibt sich eine Auswahl-Möglichkeit von über

15000. Offensichtlich können durch dieses Gerät den Bedürfnissen eines jeden Programms entsprochen werden. Aus diesem Grunde ist die Tastatur die bevorzugte Eingabe-Möglichkeit unter Programmierern.

Während die Stärken der Tastatur unbestreitbar sind, werden ihre Schwächen selten erkannt. Die erste Schwäche ist, daß nicht viele Menschen wissen, wie sie effektiv zu benutzen ist. Programmierer beanspruchen die Tastatur bei ihrer täglichen Arbeit sehr stark und sind konsequenterweise auch schnelle Schreiber, wogegen der normale Verbraucher nicht so gut mit ihr umgehen kann. Es kann sehr leicht die falsche Taste gedrückt werden. Allein die Existenz aller Tasten und das Wissen, daß die Richtige zu drücken ist, erschreckt die meisten Menschen.

Die zweite Schwäche der Tastatur ist ihre Indirektheit. Es ist sehr schwer möglich, der Tastatur eine indirekte Bedeutung zu geben. Sie besitzt keine offensichtliche sinnliche oder emotionelle Bedeutung. Der unerfahrene Benutzer hat große Probleme sich an sie zu gewöhnen. Sämtliche Arbeit mit der Tastatur läuft symbolisch ab: es werden Knöpfe verwendet, denen den jeweiligen Umständen entsprechend verschiedene Bedeutungen zugeordnet werden. Tastaturen leiden außerdem unter ihrer Verbindung mit Text auf Bildschirm: die Schwächen von Text wurden in diesem Kapitel schon besprochen.

Ein weiterer Nachteil von Tastaturen ist, daß der Designer sich ständig über ihre digitale Natur im Klaren sein muß. Die Tastatur ist sowohl Auswahl als auch zeitlich digital aufgebaut. Hierdurch wird natürlich ein gewisses Maß an Schutz vor Fehlern geliefert. Da die Abfrage von Tastendrücken im Bezug auf den zeitlichen Ablauf nicht fortlaufend, sondern digital ist, eignet sich die Tastatur nicht gut für Echtzeit-Programme. Menschen sind Echtzeit-Wesen, daher ist dieses eine Schwäche. Der Designer muß erkennen, daß die Verwendung der Tastatur ihn von Echtzeit-Kommunikation zwischen Computer und potentiellem Benutzer abhält.

Paddles

Paddles sind die einzigen wirklichen analogen Eingabe-Geräte, die vom System gelesen werden können. Als solche leiden sie an einem standardmäßigen Problem, das alle analoge Eingabe-Einheiten haben: Die Forderung, daß der Benutzer Werte Präzise setzen muß, um ein Ergebnis zu erhalten. Ihr Auflösungewinkel ist klein und thermische

Effekte produzieren selbst bei einem nicht berührten Paddle Ausgabe-Ungenauigkeiten.

Paddles besitzen zwei Vorteile. Sie sind erstens gut zum Auswählen von Werten einer eindimensionalen Variablen geeignet. Es ist einfach verständlich daß mit dem Paddle schnell ein Wert ausgesucht und mit dem Auslöser festgelegt werden kann. Zweitens kann der Benutzer durch eine einzige Umdrehung des Reglerknopfes von einem Ende eines Spektrums zum Anderen gelangen. Das gesamte Spektrum ist für den Benutzer also jederzeit verfügbar.

Ein Wichtiger Faktor bei der Benutzung von Paddles ist die Erzeugung einer geschlossenen Ein/Ausgabe-Schleife. Bei den meisten Eingabe-Vorgängen ist es vorteilhaft, die Eingabe sofort auf dem Schirm auszudrücken, so daß sie sofort überprüfbar ist. Dieser Prozess erzeugt eine geschlossene Ein/Ausgabe-Schleife. Die Information geht vom Benutzer zum Computer, der sie auf dem Bildschirm anzeigt und somit wieder an den Benutzer weitergibt. Da Paddles keine absolute Position besitzen ist eine solche Wiedergabe der Eingabe grundlegend.

Jeder Satz von Eingaben kann in einer linearen Bedeutungsfolge platziert werden, die durch ein Paddle adressiert werden kann. So können z.B. Menüs mit einem Paddle angewählt werden. Die Reihenfolge läuft vom oberen zum unteren Menüende. Es ist möglich (allerdings auch unsinnig), mit einem Paddle eine Tastatur zu ersetzen. Mit dem Paddle wird der Buchstabe ausgewählt der gleichzeitig auf dem Bildschirm angezeigt wird. Durch den Feuerknopf wird der Buchstabe festgelegt. Während diese Adressierungs-Methode die Schreibgeschwindigkeit nicht verbessert, ist sie doch sinnvoll bei Lernprogrammen für Kinder.

Joystick

Die Joysticks sind die simpelsten Eingabe-Geräte des Computers. Sie sind sehr stabil und können daher einer relativ hohen Belastung ausgesetzt werden. Sie besitzen nur 5 Schalter, daher wird ihre Leistungs-Fähigkeit oft unterschätzt. Trotzdem sind Steuerknüppel sehr wirkungsvolle Eingabegeräte. Mit einem Steuerknüppel kann über einen Cursor jede Bildschirmposition angewählt werden, wobei der Auslöser als Festlegung fungiert. Durch ein entsprechendes Bildschirm-Layout kann der Joystick ein weites Funktions-Spektrum anwählen. Diese Technik wurde z.B. bei SCRAM"TM" benutzt.

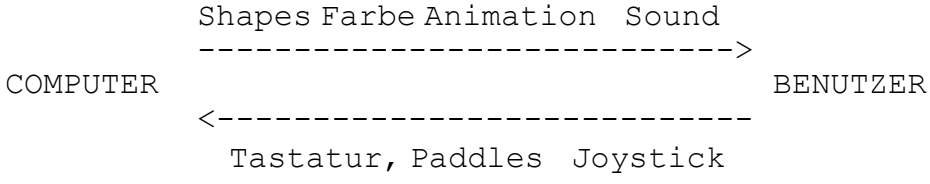
Der Schlüssel zur guten Benutzung des Joystick liegt in der Erkenntnis, daß der kritische Punkt nicht die Auswahl eines Schalters ist, sondern der Zeitraum für den ein Schalter aktiviert wird. Durch Kontrollieren der Zeit, für die ein Schalter gedrückt wird, bestimmt der Benutzer, wie weit sich der Cursor bewegt. Diese Technik erfordert meistens einen Cursor, der sich mit einer konstanten Geschwindigkeit bewegt. Ein Nachteil hierbei ist: bewegt er sich zu schnell, ist der Benutzer nicht in der Lage, ihn genau zu positionieren; bewegt er sich zu langsam, wird der Benutzer ungeduldig, wenn der Cursor größere Entfernungen zu überwinden hat. Eine Lösung dieses Problems wäre ein beschleunigender Cursor. Wenn der Cursor sich zu Anfang langsam bewegt und dann schneller wird, ist sowohl feines Positionieren, als auch ein schnelles Bewegen über größere Distanzen möglich.

Der eigentliche Wert des Joysticks ist eine feine Bedienungs-Möglichkeit. Durch den Steuerknüppel ist dem Benutzer eine direkte und gefühlvolle Eingabe möglich. Die Feinfühligkeit der Tastatur ist emotional nicht bedeutend. Ein Joystick besitzt eine Logik - er wird aufwärts gedrückt, um ein Objekt nach oben zu bewegen, nach unten, um es nach unten zu bewegen. Der Cursor wiederholt diese Bewegung auf dem Bildschirm, so daß der Benutzer in einer gefühlsmäßigen Verbindung mit diesem steht.

Joysticks besitzen ihre Grenzen. Obwohl es möglich ist, den Knüppel in eine diagonale Stellung zu bewegen und diese Stellung korrekt zu lesen, ist es nicht so deutlich, als wenn diagonale Richtungen separat eingegeben würden. Daher sollten diagonale Werte vermieden werden, bis sie in einer rein geometrischen Bedeutung benutzt werden: aufwärts auf dem Joystick bedeutet aufwärts, rechts bedeutet rechts und diagonal bedeutet diagonal.

ZUSAMMENFASSUNG DER KOMMUNIKATIONSELEMENTE

Es wurden eine Anzahl von Möglichkeiten besprochen, die zusammen die Elemente einer Sprache für die Kommunikation zwischen Mensch und Benutzer bilden. Es sind:



AUFBAUEN EINER SPRACHE

Wie werden alle Elemente einer effektiven Sprache zusammengesetzt? Um dieses zu tun, müssen wir als erstes die grundlegenden Züge einer von uns als gut bewerteten Sprache bestimmen. Es sind:

Vollständigkeit

Die Sprache muß alle für die Kommunikation zwischen Computer und Benutzer nötigen Dinge ausdrücken können. Die internen Vorgänge oder die Gedankengänge des Benutzers müssen nicht verdeutlicht werden. So muß z.B. die Sprache des STAR RAIDER"TM"-Programms nur die für die Steuerung des Raumschiffs und die für den Kampf wichtigen Informationen ausdrücken können. Die Gefühle des Spielers oder die Flugabsichten der Zylonen müssen nicht durch sie übermittelt werden. Diese Dinge sind zwar sehr wichtig für die gesamte Funktion des Spiels, müssen aber nicht zwischen Benutzer und Programm ausgetauscht werden.

Vollständigkeit ist die offensichtliche Funktion einer jeden Sprache, eine, die alle Programmierer intuitiv erkennen. Probleme im Zusammenhang mit der Vollständigkeit tauchen meistens dann auf, wenn der Programmierer zusätzliche Funktionen an ein Programm anfügen will, die aber nicht durch die von ihm aufgebaute Sprache geliefert werden können. Dieses kann sehr ärgerlich sein, da in vielen Fällen, diese Zusätzlichen Funktionen im Programm selbst vorhanden sind. Der einschränkende Faktor ist immer die Schwierigkeit beim Anfügen neuer Ausdrücke an die I/0-Sprache.

Direktheit

Jede neue Sprache ist schwierig zu erlernen. Der Benutzer kann seine Zeit nicht mit dem Lernen immer neuer ausgeschmückter Sprachen verschwenden. Die vom Programmierer entworfene Sprache muß direkt und treffend sein. Sie sollte sich an Verabredungen halten, die der Benutzer bereits kennt. Die Befehle müssen eindeutig und offensichtlich in ihrer Bedeutung sein. So wäre ein CONTROL-X-Befehl sehr unklar in seiner Wirkung. Mit ihm könnte etwas kontrolliert werden, wobei X Auslöschung oder Negation bedeutet. Die Anweisung könnte auch heißen, daß etwas überprüft, berichtigt oder ähnliches werden soll. Treffen diese Möglichkeiten nicht zu, so ist diesem Kommando zu indirekt.

Tastaturen sind berüchtigt für die Erzeugung solcher Probleme.

Geschlossenheit

Geschlossenheit ist der Aspekt des Kommunikation-Designs, der die größten Probleme verursacht. Dieses läßt sich an einem Beispiel am besten erklären: ein Benutzer befindet sich am Punkt A und will durch das Programm zum Punkt B gelangen. Ein schlecht ausgearbeitetes Programm entspricht einem Seil, das zwischen diesen beiden Punkten gespannt ist. Wenn der Benutzer ganz genau weiß, was er tun muß, so hat er Erfolg. Es ist aber viel wahrscheinlicher, daß er ausrutscht und abstürzt. Bei einigen Programmen gibt es interne Warnungen oder Manuals, die dem Benutzer sagen, was er tun darf und was nicht. Diese Dinge entsprechen Schildern neben dem Seil auf denen steht "SEIEN SIE VORSICHTIG!"- oder "NICHT HINUNTERFALLEN!". Andere Programme stellen Schilder unter dem Seil auf, auf denen steht, warum der Benutzer ausgerutscht ist.

Eine bessere Art von Programmen besitzt Masken für illegale Eingaben. Sie entsprechen Geländern neben dem Seil. Diese sind natürlich sehr viel günstiger als die vorangegangenen Lösungen , müssen aber so geschickt konstruiert sein, daß der Benutzer sie nicht unterlaufen kann. Andere Programme wiederum erzeugen unangenehme Bell-Geräusche, die den Benutzer davor warnen, bestimmte Eingaben zu machen. Diese Lösung entspricht mürrischen Aufsehern auf dem Schulhof und bewirken nur, dass der Benutzer sich wie ein dummes Kind vorkommt. Das ideale Programm entspricht einem Tunnel durch festes Gestein. Es gibt nur einen einzigen Weg: den Weg zum Erfolg. Dem Benutzer bleibt also keine andere Möglichkeit, als sein Ziel zu erreichen.

Die Grundlage der Geschlossenheit ist das Einschränken der Options-Anzahl, die Löschung von Möglichkeiten, was dem Aufbauen von soliden Mauern um den Benutzer entspricht. Ein gutes Design erfordert nicht den Aufbau einer ungeheuren Anzahl von Möglichkeiten auf eine Grundstruktur. Es müssen vielmehr unwichtige Dinge gestrichen werden.

Diese These widerspricht den Auffassungen mancher Programmierer. Sie bevorzugen eine absolute Freiheit, um ihre Stärke gegenüber dem Computer erproben zu können. Das am häufigsten angebrachte Argument besagt, daß ein Programm ihre Möglichkeiten auf irgendeine Weise einschränkt. Wer sollte so dumm sein und die Leistungsfähigkeit dieses wundervollen Werkzeuges einschränken?

Die Antwort hierauf liegt im Unterschied zwischen dem Benutzer und dem Programmierer. Der Programmierer widmet sein Leben dem Computer; der Verbraucher arbeitet im maximalen Fall ein wenig mit dem Gerät. Der Programmierer benutzt die Maschine so intensiv, daß es die aufzuwendende Zeit lohnt, sie besser kennen zulernen. Der Benutzer hat nicht die Zeit, sich genauer mit dem Gerät auseinanderzusetzen; er möchte so schnell wie möglich zu Punkt B gelangen. Er interessiert sich nicht für die Feinheiten, die das Leben des Programmierers ausmachen. Klingel und Bimmelgeräusche, die vom Programmierer mit Begeisterung entwickelt werden, sind für den Verbraucher selbstverständlich. Programmierer mögen sich nicht viel um die Worte des Verbrauchers kümmern, aber wenn diese ihren Lebensunterhalt verdienen wollen, müssen sie dieses wohl oder übel tun.

Geschlossenheit wird durch Erstellen von Ein- und Ausgaben erreicht, die keine illegalen Werte zulassen. Mit einer Tastatur ist dieses sehr schwer durchzuführen, da über sie immer mehr Eingaben möglich sind, als ein Programm benötigt. Ein Joystick wäre in dieser Hinsicht besser geeignet, da er nur einen sehr geringen Eingabe-Bereich besitzt. Die ideale Lösung wird erreicht, wenn über ein Gerät sämtliche Eingaben (nur die absolut vorhandenen) möglich sind. In diesem Fall kann der Benutzer keine falschen Eingaben machen, da es sie gar nicht gibt. Ein Beispiel hierfür wäre die “Neusprache” in Orwells Roman “1984”: der Benutzer könnte keine falschen Eingaben machen (denken), da es keine Wörter für sie gibt.

Geschlossenheit ist mehr als nur das Ausmaskieren schlechter Eingaben; das Ausmaskieren alleine ist nicht funktional. So könnte z.B. die “M”-Taste eines Keyboards abgeschaltet werden, wenn sie bedeutungslos ist. Der Benutzer sieht aber die Taste immer noch, könnte sie daher auch noch drücken und würde sich wundern, warum nichts geschieht - dieses ist alles überflüssige Anstrengung. Der Benutzer verschwendet sehr viel Zeit durch das Drücken und sich-Wundern. Der Programmierer kann aber diese Möglichkeit ausschalten, indem er sich in die Lage dem Benutzers versetzt. Im Gegensatz dazu könnte ein sauber geschlossenes Eingabegerät nur für das Programm erforderliche Daten erzeugen und weiterleiten. Der Benutzer kann keine Zeit an überflüssigen Handlungen verschwenden, da solche gar nicht mehr vorkommen können.

Die Vorteile einer korrekt aufgebauten Geschlossenheit sind vielfältig. Das Programm ist einfacher und schneller, da keine Eingabe-Überprüfung vorgenommen werden muß. Der

Benutzer benötigt weniger Zeit, um mit dem Programm vertraut zu werden, da er weniger Probleme mit ihm hat.

Das vorrangige Problem der Geschlossenheit ist der arbeitsaufwendige Aufbauprozess um sie zu erreichen. Das gesamte Verhältnis zwischen Benutzer und Programm muß sorgsam analysiert werden, um das Minimum des erforderlichen Vokabulars festzulegen, das für die Kommunikation beider gebraucht wird. Eine große Anzahl von Kommunikations-Mustern muß überprüft und aufgegeben werden, bevor das sinnvollste gefunden wird. Bei diesem Vorgang sind viele Glocken- und Pfeifgeräusche zu eleminieren, die der Programmierer gerne in das Programm einbringen würde. Wenn der Programmierer objektiv den Wert dieser Geräusche und sonstiger Ausstattungen betrachtet, so wird er merken, daß diese Dinge eher überflüssiges Anhängsel als sinnvolle Einrichtungen sind.

FOLGERUNGEN

Das Design einer Sprache zur Kommunikation zwischen Mensch und Computer ist der schwierigste Arbeitsprozess bei der Entwicklung von Verbraucher-Software. Der Programmierer muß sorgfältig die Möglichkeiten der Maschinen gegenüber den Bedürfnissen des Benutzers abwägen. Er muß präzise die Information bestimmen, die zwischen den beiden Beteiligten fließen soll. Die Sprache muß so entwickelt werden, daß die Qualität (nicht die Quantität) der zum Benutzer fließenden Information auf ein Maximum gebracht wird, wogegen die erforderliche Anstrengung des Benutzers zur Bedienung des Gerätes auf ein Minimum beschränkt werden muß. Die Sprache sollte die Möglichkeiten und Peripherie der Maschine effektiv anwenden, wobei Geschlossenheit und Direktheit erreicht werden müssen.

EINIGE ÜBLICHE PROBLEME BEI DER AUSARBEITUNG

Nachdem die Probleme bei der Ausarbeitung theoretisch besprochen wurden, können wir uns jetzt Problemen bestimmter Programme in Hinsicht auf diese hinwenden. Die Liste erhebt keinen Anspruch auf Vollständigkeit; es werden die allgemeinen vorkommenden Probleme angesprochen.

VERZÖGERUNGEN

Viele Programme erfordern langwierige Berechnungen. Tatsächlich gilt dies mehr oder weniger für alle Programme;

jedes führt Kalkulationen aus, die mehr als eine Sekunde benötigen. Was unterhält den Benutzer während diese ausgeführt werden? Zu viele Programme unterbrechen einfach den Dialog in dieser Zeit, so daß letzterer sich einem inaktiven Bildschirm ohne Lebenszeichen vom Computer gegenübersieht. Der Computer reagiert nicht auf Eingaben vom Benutzer. In Hinsicht auf die Ausarbeitung fehlt diese total. Es ist absolut unentschuldbar, daß der Benutzer alleingelassen wird.

Separate Vorgänge

Die beste Möglichkeit, dieses Problem der Nichtbeachtung des Benutzers zu lösen ist, den Eingabe-Prozess vom Kalkulations-Vorgang zu trennen. Der Benutzer muß in der Lage sein, Eingaben zu machen, während der Computer Berechnungen ausführt. Die Technik wird durch die Verwendung des Vertical-Blank-Interrupts (siehe Anhang I) ermöglicht. Der Programmierer kann die Eingabe-Bearbeitung von der hauptsächlichen Verarbeitung trennen. Das wirkliche Hindernis hierbei ist, daß viele Probleme sequenzieller Natur sind. Es ist klar ersichtlich, daß der Benutzer erst einen Wert oder eine Wahl eingeben muß, bevor das Programm den nächsten Schritt ausführen kann. Hierdurch ist es schwierig, Eingabe- und Kalkulations-Bearbeitung voneinander zu trennen. Natürlich ist es durch geschickten Programmentwurf möglich Zwischenberechnungen durchzuführen, so das während die wichtigen Daten eingegeben werden, das Endergebnis schneller erzielt wird. Die Verwendung dieser Technik verkürzt sicherlich die Verzögerungszeiten, bei denen der Benutzer warten muß.

Erhöhen der Programmgeschwindigkeit

Eine andere Lösungsmöglichkeit des oben angesprochenen Probleme besteht in der Erhöhung der Ausführungsgeschwindigkeit des Programms selbst. Oft ist es möglich, ein Programm so neu zu schreiben, daß die Ausführungszeit verringert wird. Die ordentliche Verschachtelung von Schleifen reduziert ebenfalls die Ausführungszeit (die Schleife mit mehr Wiederholungen sollte sich in einer Schleife mit weniger Wiederholungen befinden). Bedeutende Verbesserungen können durch die Umwandlung von BASIC in Assembler-Programme erzielt werden, da Assembler 10 bis 1000 mal schneller als BASIC ist. Dar größte Vorteil in

Assembler-Programmen liegt in der Verschiebung von Speicher, Graphik-Routinen und weniger stark in Fließkomma-Berechnungen. Durch Ausmaskieren von Vertical-Blank-Interrupts kann mehr Prozessorzeit für die Ausführung des Hauptprogramme gewonnen werden. Eine weitere Möglichkeit ist die Reduzierung der von ANTIC für DMA benötigten Zyklen. Dieses kann durch Wechseln zu einem einfachen Graphik-Modus (BASIC-Modus 3 ist hierfür am besten geeignet) oder durch Verkürzen der Display-List geschehen. Das Abschalten des ANTIC-Chips ist keine gute Idee, da der Benutzer sich hierdurch wieder einem schwarzen Bildschirm gegenübersieht.

Beschäftigung des Benutzers

Die dritte Möglichkeit, um das Problem der Verzögerungen zu lösen besteht in der Beschäftigung den Benutzers während der Berechnungsvorgänge. Ein Countdown ist eine solche Methode. Der Benutzer sieht diesen und wird abgelenkt. Der Computer kehrt bei Beendigung dieses Countdowns zum Hauptprogramm zurück. Eine andere Möglichkeit wäre das Zeichnen zufälliger Graphikmuster auf dem Bildschirm. Die Verzögerungszeit sollte immer mit einer Information des Computers beginnen, die den Benutzer letzteres mitteilt. Diese Meldung sollte durch ein Klingel- oder ähnliches Geräusch beendet worden. Der Programmierer darf aber nicht von Anfang an auf diese Technik aufbauen. Sie ist zwar besser, als den Benutzer ganz allein zu lassen, aber sollte nicht als erste Möglichkeit erwogen werden.

BEARBEITEN FALSCHER BENUTZEREINGABEN

Das wichtigste Problem bei momentan erhältlicher Verbraucher-Software ist die schlechte Bearbeitung falscher Benutzereingaben. Gute Programme schließen solche dadurch aus, daß sie gar nicht erst möglich sind. Wie schon erwähnt, wird diesem am einfachsten durch die Verwendung des Joysticks erreicht. Es gibt allerdings auch Programme, welche die Benutzung einer Tastatur erfordern. Selbst mit einem Joystick sind solche falschen Eingaben nicht auszuschließen. Wie können sie aber nun bearbeitet werden, wenn es nicht möglich ist, sie von vornherein zu eliminieren? Die folgenden Absätze stellen einige Lösungs-Möglichkeiten dar. Ein Schutz-System muß im gesamten Programm einheitlich ablaufen. Sobald der Benutzer auf ein solchen System stößt, wird er es immer voraussetzen. Eine Nichtbeachtung diesen Grundsatzes schafft Gruben, in die der

Benutzer mit höchster Wahrscheinlichkeit hineinfällt, da er sich einer nicht vorhandenen Schutzvorrichtung sicher ist.

Anzeigen des Fehlers und Lösung

Die beste Lösungs-Möglichkeit in einer solchen Fehlersituation ist das Anzeigen des Benutzer-Fehlers mit einem Beispiel für eine richtige Eingabe auf dem Bildschirm. In der Antwort des Computers müssen drei Dinge eingeschlossen sein: als erstes muß die Eingabe des Benutzers noch einmal angezeigt werden, um diesem seinen Fehler erkennen zu lassen. Als zweites muß der Fehler deutlich markiert werden, so das dem Benutzer der Fehler ersichtlich ist. Drittens muß schließlich eine alternative korrekte Eingabe-Möglichkeit dargestellt werden, damit der Benutzer nicht das Gefühl hat, er stehe vor einer unüberwindlichen Mauer. Ein Beispiel für eine Antwort auf einen falschen Tastendruck wäre: "Sie haben CONTROL-A gedrückt, was eine Autopsie anfordert. Ich kann keine Autopsien für lebende Leute anfertigen. Ich schlage daher vor, daß Sie das Subjekt vorher kaltmachen."

Diese Technik ist im Bezug auf die Programmgröße und Entwicklungszeit offensichtlich nicht sehr billig; ist aber der Preis für eine gute Lösungs-Methode. Es gibt natürlich auch weniger aufwendige Möglichkeiten, die allerdings auch nicht so effektiv sind.

Ausmaskierung nicht zulässiger Tasten

Eine andere übliche Lösung des Eingabe-Problems bei Tastaturen ist die Ausmaskierung aller falscher Eingaben. Wenn der Benutzer eine nicht zulässige Taste drückt, so geschieht nichts. Es wird kein Klicken erzeugt und kein Buchstabe auf dem Bildschirm ausgedruckt. Das Programm hört nur das, was es hören will. Diese Lösung garantiert zwar, daß ein Programm nicht abstürzt, schützt den Benutzer allerdings nicht vor Verwunderung und Verwirrung. Er würde lediglich eine Taste drücken, wenn sie etwas bedeutet. Die Ausmaskierung korrigiert nicht den Eindruck des Benutzers, er habe etwas falsch gemacht. Diese Möglichkeit führt nur zum Schluß, daß irgendetwas mit dem Computer nicht in Ordnung ist, was wir den Benutzern nicht antun wollen.

Eine Variante dieser Möglichkeit wäre die Erzeugung eines unangenehmen Knarr- oder Summgeräusches, das den Benutzer für seine Dummheit bestraft. Einige Programms gehen sogar

soweit, daß sie Letzteren zurechtweisen, bis hin zu Beleidigungen. Dieses sind mehr fragwürdige Techniken. Es gibt natürlich Fälle, bei denen ein Schutz vor gefährlichen Tastendrücken durch unliebsame Mitteilungen erforderlich sind. Sie sind aber sehr selten. Korrektur-Meldungen sollten immer dem Zivilisations-Standard entsprechen.

Fehler-Meldungen

Eine sehr viel bessere Lösung ist das einfache Ausgeben einer Fehlermeldung auf den Bildschirm. Dem Benutzer wird mitgeteilt, daß er etwas falsch gemacht hat. In vielen Fällen ist aber diese Fehlermeldung verschlüsselt und nützt dem Benutzer daher nicht viel. Ein Beispiel hierfür ist das ATARI"TM" BASIC-Modul, bei denen der aufgetretene Fehler durch eine Zahl angegeben wird. Dieses ist nur gerechtfertigt, wenn das Programm unter extremen Speicherbedingungen laufen muß.

In den meisten Fällen ziehen die Programmierer es vor, die Benutzer-Behandlung, wie z.B. sinnvolle Fehlermeldungen, für zusätzliche technische Möglichkeiten zu vernachlässigen. Wie schon am Anfang dieses Kapitel erklärt, erreichen wir eine Stufe, bei der nicht länger die zusätzliche technische Leistungsfähigkeit sondern die Ausarbeitung die Begrenzung ist. Daher sind solche Abstriche nicht unbedingt vertretbar.

Abstriche beim Schutz zugunsten der Leistung

Ein Bestandteil der Behandlung des Benutzers ist, daß deren Folgen die Arbeitsgeschwindigkeit zwischen Benutzer und Computer verlangsamen. Programmierer können Fragen und Sicherungen wie "ARE YOU SURE?" (=Sind Sie sich sicher?) in Programmen nicht mehr sehen. Eine Lösung dieses Problems wäre die Wahlmöglichkeit des Verhältnisses Schutz/Leistungsfähigkeit. So kann ein Programm mit einem sehr stark geschützten Zustand initialisiert werden, wodurch alle Eingaben sorgsam überprüft und berichtigt werden. Der Benutzer kann nun diesen Schutz zugunsten der Geschwindigkeit zurücksetzen. Diese Option wird nicht auf dem Bildschirm ausgedruckt - sie wird nur in der Dokumentation beschrieben. So kann ein Benutzer intensiv und schnell arbeiten, ein anderer langsam, aber sicher vor Fehlern.

MENÜS UND AUSWAHL-TECHNIKEN

Menüs sind die gebräuchlichste Einrichtung, um den Benutzer seine Wahlmöglichkeiten anzuzeigen. Sie sind besonders bei Anfängern sinnvoll. Kommando-orienterte Schemen, die von Programmierern bevorzugt werden, enttäuschen den Benutzer, der nicht die Zeit zum Lernen sämtlicher Befehle eines solchen Programms hat. Natürlich sind auch mit der Verwendung von Menüs Probleme verbunden; sie werden nachfolgend besprochen.

Menügröße

Wie viele Punkte sollten in einem Menü vorhanden sein? Die Obergrenze wird offensichtlich durch die Bildschirmgröße festgelegt, was im BASIC-Modus 0 bis zu 48 Punkten entspräche (24 Zeilen mit jeweils 2 Eingängen). Eine Menge von 7 Auswahlmöglichkeiten wäre zu vertreten. Hierdurch bliebe noch weiterer Platz des Schirmes für andere Dinge frei, wie einen Menütitel oder eine Computerantwort.

Mehrere Menüs

Des öfteren erfordert ein Programm mehrere Menüs, damit sämtliche Möglichkeiten erfasst werden. Diese Menüs müssen dann in einer übersichtlichen Art und Weise organisiert werden, da der Benutzer sich beim Durchwandern solcher Labyrinthe leicht verirren kann. Eine Lösungsmöglichkeit wäre, das Hauptmenü deutlich zu markieren und jedes untergeordnete mit einer Rücksprungs-Möglichkeit zum ersteren auszustatten. Eine andere Verschachtelungsart wäre eine hierarchisch aufgebaute Struktur. Wird diese Technik verwendet, muß der Programmierer Farbe und evtl. Sound in das Programm einbringen, so daß der Benutzer sich anhand dieser Dinge zurechtfinden kann. Jedes Menü-Niveau sollte eine bestimmte Farbe und Tonart besitzen, wobei letztere in Verbindung mit der Position innerhalb der Hierarchie stehen sollte.

Auswahlmethoden

Wie teilt der Benutzer dem Computer seine Wahl mit, wenn er die Möglichkeiten aufgezeigt bekommen hat? Die am häufigsten verwendete Methode besteht in der Markierung der einzelnen Menüpunkte mit einer Zahl oder einem Buchstaben: der Benutzer drückt zum Auswählen einfach die zugehörige Taste auf der Tastatur. Dieses ist eine etwas ungeschickte

Technik, da sie mit unnötiger Indirektheit verbunden ist. Bessere Methoden verwenden das folgende Grundprinzip: ein beweglicher Zeiger adressiert die Option und ein Auslöser wählt sie aus. Dieses wird bei einigen Programmen durch invertierte Schrift deutlich gemacht. Über die SELECT-Taste wird der Menüpunkt angewählt, wobei vom untersten wieder zum obersten zurückgesprungen wird. Durch die START-Taste wird der Menüpunkt aktiviert. Eine andere Möglichkeit wäre, den Zeiger immer durchlaufen zu lassen, so daß der Benutzer nur eine Taste oder einen Auslöser drücken muß, damit die Auswahl getroffen wird. Diese Methode ist allerdings nicht sehr geeignet, da sie entweder zu schnell am Ziel vorbeiläuft oder zu lange zum Durchlaufen sämtlicher Punkte benötigt. Für die vorangegangene Technik sind Joysticks und Paddles besonders gut verwendbar, da mit ihnen die Auswahl getroffen und über den Auslöser aktiviert werden kann.

BEDIENUNGSANLEITUNGEN CONTRA BILDSCHIRM-ERLÄUTERUNGEN

Ein allgemeines Problem im Zusammenhang mit Menüs, Fehlermeldungen, Computer-Antworten (Prompt) und anderen Mitteilungen ist, daß dieses Material sehr leicht einen großen Speicherplatz verbraucht; Speicher, der gut für andere Dinge benutzt werden könnte. Diesem Material könnte in einem Referenzendokument platziert werden, was sich allerdings negativ im Bezug auf die Ausarbeitung auswirkt. Der Programmierer muß entscheiden, wie viel Material in das Programm eingebracht und wie viel in einem Manual abgedruckt werden soll. Bei Programmen, die auf Disketten-Benutzung basieren, ist es möglich, dieses Material auf Disk zu speichern, obwohl dieses keine optimale Lösung ist.

Betrachtet man das Problem vom Standpunkt der Ausarbeitung und des "Sich-um-den-Benutzer-Kümmern" aus, so fällt eine Entscheidung leicht; sämtliche Informations-Materialien müssen sich im Programm, oder zumindest auf Diskette befinden. Gegen diese Lösung sprechen natürlich die technischen Aspekte. Eine Technologie sollte für die Dinge eingesetzt werden, die sie am besten bearbeiten kann. Ein Computer kann natürlich statischen Text behandeln, seine Stärke ist die dynamische Verarbeitung von Information. In den meisten Fällen ist die Bearbeitung statischer Information besser mit Papier und Feder durchzuführen, als mit dem Computer. Statische Information gehört daher im allgemeinen in ein Manual, wobei das Programm auf diesen Text hinweist. Kritische Information sollte allerdings weiterhin im Programm eingeschlossen sein.

ERFOLGSMAßSTÄBE

Wie kann der Programmierer den Erfolg seiner Ausarbeitungen für den Benutzer feststellen? Hierfür gibt es einige Indikatoren, wie z.B. die Länge eines Manuals; je kompakter und kürzer, desto besser. Wenn Sie die Hintergrund-Informationen ausschließen und nur das für die Beschreibung der Programmbenutzung erforderliche Material berücksichtigen, so ist der Umfang letzterer ein guter Maßstab. Ein gutes Programm benötigt nur wenig Erklärungen, was natürlich kein Argument gegen eine ausführliche Dokumentation sein soll. Eine Anleitung sollte das Programm immer detaillierter beschreiben, als absolut notwendig. Ein langes, kompliziertes Manual ist gut; ein Programm, das ein solches Manual erfordert, nicht.

Ein anderer Maßstab ist der Zeitraum, den ein neuer Benutzer zum Erlernen der effektiven Programmbenutzung braucht. Gute Programme können nach wenigen Minuten erfolgreich angewendet werden.

Man kann auch den erforderlichen Gedankenaufwand des Benutzers als Orientierung verwenden. Ein gutes Programm sollte das Denken des Benutzers nicht strapazieren. Er sollte sich über den Inhalt des Programms und nicht über dessen Technik Gedanken machen. Er muß sich darauf konzentrieren können, was er tut und nicht, wie er es tun muß. Ein gut ausgearbeitetes Programm eliminiert mentale Schranken zwischen dem Benutzer und dem Computer. Die beiden "denkenden Wesen" erreichen eine mentale Synchronisation, eine Intellektuelle Einheit.

Anhang III
Der ATARI"TM" Programmrekorder

Es folgt eine Besprechung das ATARI Programm-Recoders. Die nachstehenden Punkte werden hierbei berücksichtigt:

1. WIE DER PROGRAMMRECORDER ARBEITET - Information zur Herd- und Software, die zum Betreiben des Recorders erforderlich sind.

2. ANWENDUNGS-MÖGLICHKEITEN - Wie Audio- und Digitelinformation zur Erzeugung eines benutzerfreundlichen Programms gemischt werden können.

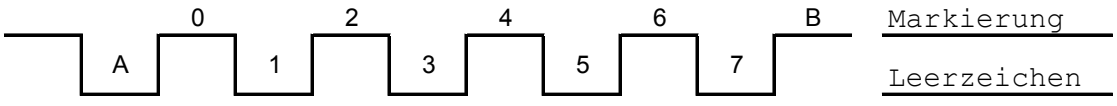
WIE DER PROGRAMMRECORDER ARBEITET

1.1 Die Struktur von Datensätzen

Byte-Definition:

Das OS schreibt Dateien in Blöcke festgelegter Länge mit einer Rate von 600 Baud (Bits/Sekunde). Zum Übertragen von Daten zwischen dem ATARI"TM" Computer und dem ATARI Programmrecorder wird eine asynchrone serielle Übertragung verwendet. Der POKEY-Chip erkennt Datenbytes in der folgenden Reihenfolge: 1 Start-Bit (Leerzeichen), 9 Daten-Bits (0=Leerzeichen, 1=Markierung) und ein Stop-Bit (Markierung). Ein Byte wird mit dem niederwertigsten Bit an der ersten Stelle gesendet/empfangen.

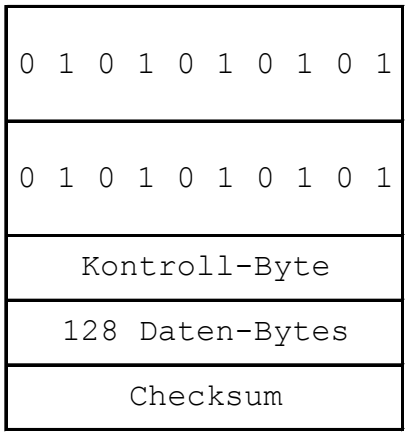
Die Frequenz für eine Markierung liegt bei 5327 Hz; die Frequenz eines Leerzeichens liegt bei 3995 Hz. Das Daten-Byte-Format sieht wie folgt aus:



A = Start-Bit (Leerzeichen)
0..7 = Daten-Byte
B = Stop-Bit (Markierung)

Record-Definition:

Records sind 132 Bytes lang und werden folgendermaßen aufgeteilt: 2 Markierungszeichen zur Geschwindigkeits-Messung, ein Kontrollbyte, 128 Daten-Bytes und ein Checksum (=Prüfsumme)-Byte. Das Record-Format wird nachfolgend dargestellt:



1. und 2. Markierung:

Jede Markierung besitzt den Hex-Wert 55. Inklusive der Start- und Stop-Bits ist jede Markierung 10 Bit lang. Idealerweise sollte sich zwischen den Markierungen und den nachfolgenden Daten kein Leerband befinden.

Geschwindigkeits-Messung:

Der Sinn der Markierungszeichen besteht in dar Justierung der Baud-Rate.

Die Eingabe-Baud-Rate beträgt im Normalfall 600 Baud. Diese Rate wird durch die SIO-Routine justiert, da sich das Band dehnen, der Motor Gleichlauf-Störungen haben kann usw. Sobald die reale Empfangs-Baud-Rate festgestellt wurde, wird die Hardware entsprechend eingestellt. Eine Raten-Änderung auf 318 bis auf 1407 Baud kann theoretisch durch diese Technik bearbeitet werden.

Das OS überprüft die Band-Geschwindigkeit auf folgende Weise: Die Software fragt ständig das serielle Eingab-Bit des POKEY-Chips ab und sucht ein Start (0)-Bit, welches den Beginn eines Record anzeigt. Wird ein solches gefunden, so speichert das OS den augenblicklichen Wert des “Frame” (=Anordnungs) -Zählers, indem der Wert von ANTICs VCOUNT-Register (Vertical Screen Counter = senkrechter

Bildschirm-Zählers) sichert. Das OS fährt weiter mit Abfragen des Eingabe-Bit fort, bis es 20 Bit (Ende der 2. Markierung) gezählt hat. Danach wird VCOUNT benutzt, um die verstrichene Zeit zu bestimmen. Die Baud-Rate wird aus dem Ergebnis errechnet. Dies geschieht für jeden Record.

Kontroll-Byte:

Das Kontroll-Byte enthält einen der drei folgenden Werte:

$FC zeigt an, daß der Record vollständig aus Daten besteht (128 Bytes).

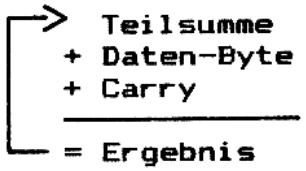
$FA zeigt an, daß der Record zum Teil aus Daten besteht; es wurden weniger als 128 Bytes vom Benutzer angegeben. Dieser Fall taucht nur gegen Ende der Datei auf. Die wirkliche Zahl der Bytes (1-127) wird im letzten Byte vor der Checksum gespeichert (dem 128. Byte).

$FE zeigt an, daß der Record ein End-of-File-Record ist, der von 128 Null-Bytes gefolgt wird.

Checksum:

Die Checksum wird von der SIO-Routine erzeugt und überprüft, befindet sich aber nicht im I/O-Puffer des Cassetten-Handlers CASBUF (.03FD.).

Die Checksum ist die Summe aller Bytes das Records, inklusive der beiden Markierungen, die in einem Byte gespeichert und mit einem Komplement-Übertrag berechnet wird. Bei Addition der einzelnen Bytes wird auch das Carry-Bit (Übertrags-Bit) addiert.



1.2 Timing

1.2.1 Inter-Record-Gap (IRG)

Wie schon in Abschnitt 1.1 angesprochen, besteht jeder Record aus 132 Daten-Bytes inklusive Checksum-Byte. Um ein Record von einem anderen zu unterscheiden, fügt der

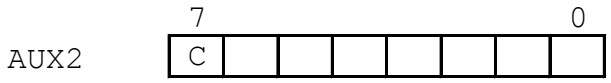
Cassetten-Handler ein sog. "Pre-Record-Write-Tone" (PRWT; Schreibton vor Record) und einen "Post-Record-Gap" (PRG; Lücke nach Record) an. PRWT und PRG sind reine Markierungs-Töne. Der "Inter-Record-Gap" (IRG; Zwischen-Record-Lücke) zwischen jeweils zwei Records besteht aus dem PRG des ersten Record, gefolgt vom PRWT des zweiten Record. Die Anordnung der Records und Lücken sieht wie folgt aus:

PRWT	MARKIERUNG	DATEN	PRG	PRWT	MARKIERUNG	DATEN	PRG
<-----	RECORD 1		----->	<-----	RECORD 2		----->

1.2.2 Normaler und kurzer IRG-Modus

Die Länge von PRWT und PRO sind abhängig vom WRITE-OPEN-Modus. Es gibt zwei Arten von IRG-Modi: normaler und kurzer IRG-Modus.

Bei Öffnen einer Datei legt das höchstwertigste Bit von AUX2 den Modus fest. Bei nachfolgender Ein- und Ausgabe führt der Cassetten-Handler das Schreiben/Lesen in einem der beiden Modi durch, abhängig vom MSB des AUX2-Bytes.



C = 1 zeigt an, daß die Datei im kurzen IRG-Modus geschrieben/gelesen werden soll.
C = 0 zeigt normalen IRG-Modus an.

Normaler IRG-Modus:

Dieser Modus wird für ein Lesen benutzt, das durch Bearbeitung unterbrochen wird; das Band stoppt immer, wenn ein Record gelesen wurde. Stoppt der Computer das Band und führt die Berechnungen schnell genug durch, so kann das Einlesen des nächsten Records so folgen, daß man nur ein kurzes Abbremsen der Kassette erkennt.

Kurzer IRG-Modus:

In diesem Modus wird das Band nicht zwischen den einzelnen Records gestoppt, unabhängig von Schreiben oder Lesen.

Beim Lesen muß das Programm ein READ für jeden Record erhalten, bevor dieser am Lesekopf vorbeikommt. Die einzige übliche Benutzung liegt bisher in der Speicherung von BASIC-Programmen in interner (tokenisierter) Form, wobei der Computer beim Lesen die Daten lediglich im RAM platzieren muß. Die BASIC-Kommandos zum Wählen diesem Modus sind "CSAVE" und "CLOAD".

Hierbei gibt es potentielle Probleme. Die auf das Band schreibende Software muß lange Lücken zulassen, so daß der Anfang der Records beim Lesen gefunden werden kann.

1.2.3 Timing-Struktur

Die Timings für die einzelnen Inter-Record-Gaps lauten wie folgt:

NORMALER IRG PRWT	= 3 Sekunden Markierungston
KURZER IRG PRWT	= 0,25 Sekunden Markierungston
NORMALER IRG PRG	= Bis zu 1 Sekunde unbestimmter Ton
KURZER IRG PRG	= Bis zu N Sekunden unbestimmter Ton, wobei N vom Timing des Benutzer-Programms abhängig ist.

Jeder Record wird mit dem folgenden Timing geschrieben: beim Starten des Motors und nach Schreiben des PRWT hängt die Dauer der Töne von dem oben gezeigten Format ab. Danach werden Record und PRG geschrieben. Der Motor wird im normalen Modus gestoppt, läuft aber im kurzen IRG-Modus weiter, da Markierungen geschrieben werden.

Anmerkung: Im normalen IRG-Modus enthält das Band aufgrund des Motorstops/-starts Teile mit unbestimmten Daten. (Es ist ein Zeitraum von 1 Sekunde möglich abhängig vom Recorder.) Diese unbestimmten Daten können der Rest der vorangegangenen Bandnutzung sein.

1.2.4 Hörbare I/O-Möglichkeit

Die hörbare I/O-Möglichkeit ist zum Bestimmen des Leseerfolges bei Kassetten nützlich, besonders bei CLOAD. Markierungen und Leerzeichen erzeugen unterschiedliche Frequenzen, so daß dar Benutzer schnell zwischen erfolgreichen und erfolglosen Einlese-Operationen unterscheiden lernt.

1.3 Datei-Struktur

Eine Datei besteht aus den folgenden drei Elementen:

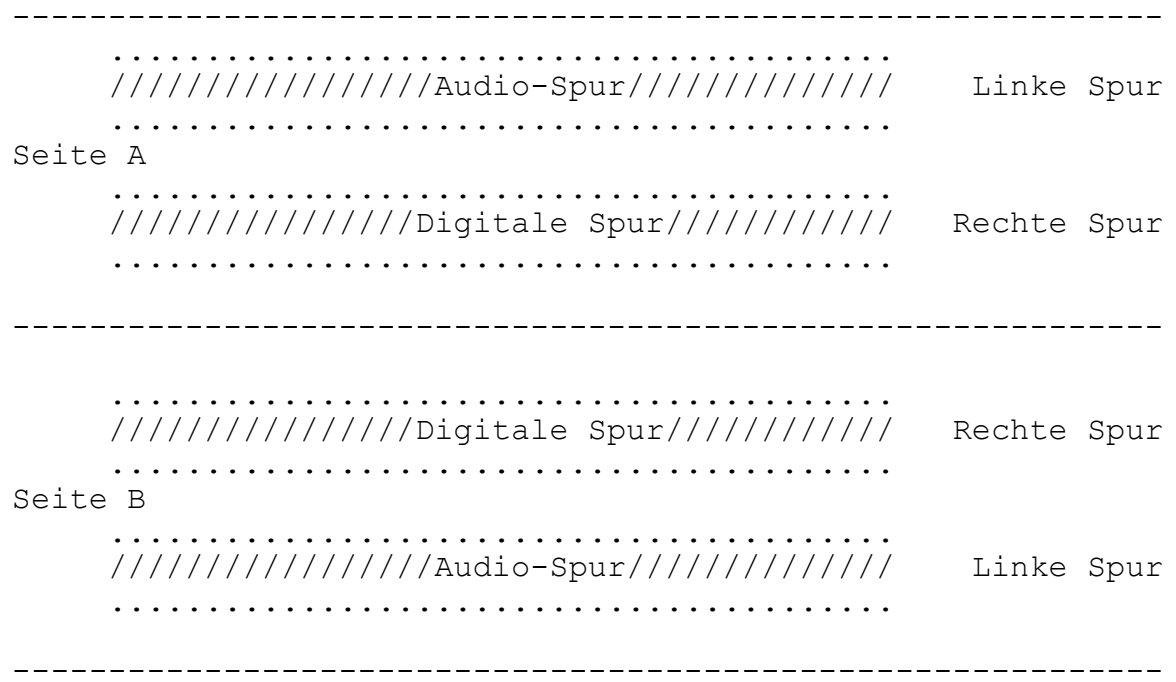
1) Ein 20 Sekunden-Vorspann des Markierungs-Tons.
2) Beliebige Anzahl von Daten-Sätzen.
3) End-of-File.

Wenn eine Datei für eine Ausgabe geöffnet wird, beginnt das OS mit dem Schreiben eines Vorspanns von 20 Sekunden Markierung. Danach springt das OS zurück zum Aufrufer, schreibt aber weiter Markierungen auf das laufende Band.

Der Timeout-Zähler für Schreib/Lese-Operationen wird bei Rücksprung dem OS auf 35 Sekunden gesetzt. Taucht ein Timeout auf, bevor der erste Satz geschrieben wird, so hält das Band an, wobei eine Lücke zwischen dem OPEN-Vorspann und dem Vorspann des ersten Records bleibt.

1.4 Band-Struktur

Es gibt auf jedem Band 2 Seiten. Jede Seite besitzt zwei Spuren, eine für Audio-, die andere für Digitalinformation. Auf diese Weise kann das Band in beide Richtungen beschrieben werden. Die folgende Darstellung zeigt einen Blick auf das Band:



Die Bänder werden im 1/4-Spur-Stereo-Format bei einer Geschwindigkeit von 1 7/8 Inch pro Sekunde (IPS) beschrieben. Dar ATARI Computer benutzt ein Tape-Deck, das einen Stereo-Kopf (nicht Mono- oder Einzel-Kopf) besitzt.

1.5 Cassetten-Boot

Ein Boot-Programm auf Kassette kann beim Einschalten des Systems als Teil der System-Initialisierung gebootet werden.

Bei der System-Initialisierung werden Funktionen wie das Rücksetzen der Hardware-Register, Löschen den RAMs, setzen von Flags usw. ausgeführt.

Nachdem die residenten Handler aktiviert sind, wird, falls die START-Taste gedrückt ist, das Flag CKEY(.004A.) gesetzt. Wenn dieses Flag gesetzt ist, wird ein Cassetten-Boot versucht.

Die folgenden Schritte sind zum Booten von Cassetten erforderlich

1) Der Benutzer muß die START-Taste drücken, wenn dem Gerät eingeschaltet wird.

2) Ein Cassetten-Band mit einer Datei im Boot-Format muß in den Recorder eingelegt und die "PLAY"-Taste gedrückt werden.

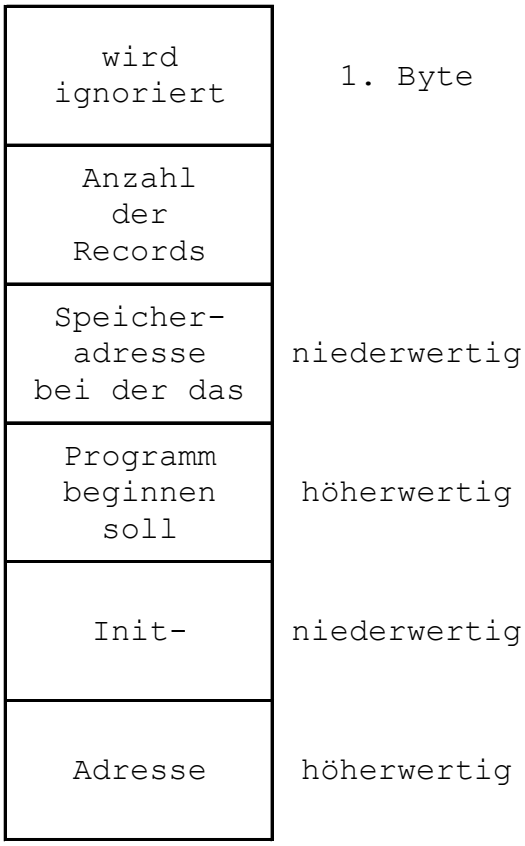
3) Die Cassetten-Datei muß im kurzen IRG-Format erstellt worden sein.

4) Nach Ertönen des Hup-Tons, was die Bereitschaft des Computers anzeigt, muß der Benutzer irgendeine Taste drücken.

Sind alle Bedingungen dieser Liste erfüllt, so liest das OS die Boot-Datei von der Kassette und überträgt nach Abschluss dieser Operation die Kontrolle an die gelesene Software. Die nachfolgende Liste beschreibt den Boot-Vorgang im Detail.

1) Der erste Cassetten-Record wird in den Cassetten-Puffer eingelesen.

2) Die Information der ersten 6 Bytes dieses Records wird analysiert. Diese Bytes einer Cassetten-Boot-Datei besitzen folgendes Format:



1. BYTE: wird nicht für den Cassetten-Boot-Prozess benötigt.

2. BYTE: enthält die Anzahl der Cassetten-Records (128 Bytes), die als Teil des Boot-Vorgangs gelesen werden sollen (inklusive des Record, der diese Information enthält). Diese Zahl bewegt sich zwischen 0 bis 255, wobei 0 = 256 bedeutet.

3. + 4. BYTE: enthalten die Adresse (nieder-, dann höherwertig), an die das erste Byte der Datei geschrieben worden soll. Wird nach Abschluss des Boots die SYSTEM-RESET-Taste gedrückt, so wird ebenfalls hierher gesprungen.

Nach Abschluss von Schritt 2 hat das Cassetten-Boot Programm...

a) Die Anzahl der zu bootenden Records gesichert;
b) die Ladeadresse gesichert;
c) die Initialisierungsadresse in CASINI (.02,03.) gespeichert.

3) Der gerade gelesene Record wird an die festgelegte Lade-Adresse gebracht.

4) Die restlichen Records werden direkt in den Lade-Bereich gelesen.

5) Es wird ein JSR zur Lade-Adresse +6 ausgeführt, wo ein Multiboot-Prozess fortgeführt werden kann. Das Carry-Bit zeigt beim Rücksprung den Erfolg oder Misserfolg der Operation an (Carry gesetzt = Fehler, Carry zurückgesetzt - Erfolg).

6) Es wird indirekt über CASINI gesprungen, um das Programm zu initialisieren. Das Programm muß bei der Initialisierung die Start-Adresse nach DOSVEC (.0A,0B.) bringen.

7) Es wird indirekt über DOSVEC gesprungen, um die Kontrolle an das Programm zu übertragen.

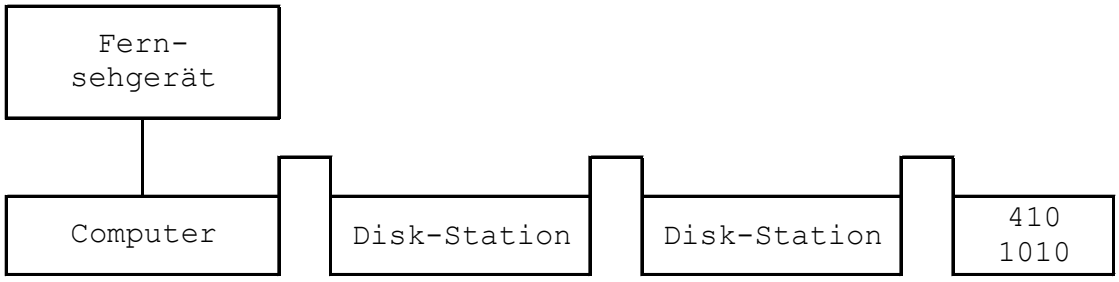
Durch Drücken der SYSTEM-RESET-Taste nach Abschluss der Boot-Operation werden die Schritte 6 und 7 wiederholt.

CASSETTEN-ANWENDUNG

Dieser Abschnitt schildert die Einsatzmöglichkeiten des ATARI"TM"-Cassetten-Systems.

2.1 Anschluss des Cassetten-Systems

Die meisten seriellen Geräte besitzen zwei identische I/O-Anschlüsse: einen seriellen Eingabe-Anschluss und einen Erweiterer des seriellen Busses. Durch diese Anschlüsse können mehrerer Peripherie-Geräte hintereinander geschaltet werden. Dies wird im folgenden Diagramm dargestellt:



Der Programm-Recorder 410 entspricht allerdings nicht dieser Anordnungsfolge. Er muss das letzte Glied der Kette sein, da er keinen Ausgang für den seriellen Bus wie andere Peripherie-Geräte besitzt. Hierdurch wird sichergestellt, daß immer nur ein Recorder an das System angeschlossen werden kann. Das System kann nicht erkennen, ob ein Programm-Recorder angeschlossen ist, daher kann er beliebig angeschlossen oder entfernt werden.

Immer, wenn eine Cassetten-Datei zum Lesen oder Schreiben bereitgemacht werden soll, muß der Benutzer folgende Schritte durchführen:

EINGABE (DATEN VOM RECORDER ZUM COMPUTER): Wird die Kassette für eine Eingabe geöffnet, so wird ein einzelner Hup-Ton vom Computer erzeugt. Ist die Kassette bereit (Strom eingeschaltet, Kabel des seriellen Busses angeschlossen und das Band an den Anfang der Datei gebracht), muß der Benutzer die PLAY-Taste des Programmrecorders und anschließend eine beliebige Taste (ausgenommen <BREAK> auf dem Computer drücken, um den Lesevorgang zu starten.

AUSGABE (DATEN VOM COMPUTER ZUM RECORDER): Wird die Kassette zur Ausgabe bereitgemacht, so werden zwei Hup-Töne vom Computer ausgegeben. Ist die Kassette bereit (siehe oben), muß der Benutzer gleichzeitig die Tasten RECORD und PLAY des Programm-Recorders drücken. Danach muß wieder eine Taste (ausgenommen <BREAK> auf dem Computer betätigt werden, um den Schreibvorgang zu starten.

2.2 Sichern und Laden digitaler Programme

Die folgende Technik sichert digitale Daten direkt vom Computer über den I/0-Anschluss entweder auf den Programm-Recorder oder auf die ATARI-Lab-Machine, die ein 1/4-Zoll-Band mit einer Geschwindigkeit von 7 1/2 Inch pro Sekunde verwendet.

BASIC:

```
Format:   CSAVE
          100 CSAVE
```

Dieses Kommando wird normalerweise im Direkt-Modus benutzt, um ein RAM-residentes Programm auf Kassette zu schreiben. Mit CSAVE wird die tokenisierte Form des Programms gesichert.

Format: CLOAD
100 CLOAD

Dieser Befehl kann sowohl im Direkt-Modus, als auch innerhalb einem Programms verwendet werden, um Programme von Kassette ins RAM zu bringen.

ASSEMBLER:

QUELLPROGRAMM FORMAT: LIST#C:(.,XX,YY.)

Dieses Kommando wird zum Schreiben des Assembler-Quellcodes auf Band benutzt. Die eingeklammerten Werte (.,XX,YY.) stehen für ein optionales Sichern von festgelegten Zeilen. Werden diese Werte nicht angegeben, so wird das gesamte Programm, d.h. alle Zeilen, gesichert.

FORMAT: ENTER#C:

Mit diesem Befehl wird das Quellprogramm von Kassette gelesen.

OBJEKTPROGRAMM:

FORMAT: SAVE#C:<XXXX,YYYY

Der Inhalt des Speicherblocks zwischen den Adressen XXXX und YYYY wird auf Kassette geschrieben.

FORMAT: LOAD#C:

Mit diesem Kommando werden die gesicherten Daten von Band in den Speicher gelesen. Der hierfür benutzte Speicherbereich entspricht dem, aus welchem die Daten mit dem SAVE-Kommando geschrieben wurden.

2.3 Sichern von digitalen Programmen mit Audio-Spur

Konzept:

Mit dieser Technik ist keine Kontrolle des Audio-Kanals durch das Programm möglich. Letzterer fungiert nur als Hintergrund, um die eintönigen Wartezeiten beim Ladevorgang zu überbrücken.

SCHRITT 1:
Folgen Sie den in 2.2 aufgelisteten Anweisungen zum Schreiben eines Digital-Programmes. Dieses Mal kann allerdings nicht

der normale Programm-Recorder (1 7/8 Inch pro Sekunde) benutzt werden, da es mit diesem nicht möglich ist, gewöhnliche Audio-Information aufzunehmen. Es wird statt dessen die ATARI-RECORDING-LAB-MACHINE verwendet, die ein Urband mit einer Geschwindigkeit 7 1/2 Inch pro Sekunde benutzt. Mit der LAB-MACHINE ist es möglich Information auf eine bestimmte Spur zu schreiben.

Auf der LAB-MACHINE wird der Aufnahme-Modus für die rechte Spur eingeschaltet, so daß die digitale Information auf die rechte Spur des 7 1/2 Inch-Bandes gebracht wird.

SCHRITT 2:
Führen Sie Schritt 1 für die Audio-Aufnahme aus. Spulen Sie dabei erst das Band an den Programm-Anfang zurück und schalten Sie dann den Aufnahme-Modus für die linke Spur ein. Auf diese Weise wird die Audio-Information auf die linke Spur des 7 1/2 Inch-Bandes geschrieben.

2.4 Synchronisation von digitalen Programmen, Audio-Information, SYNC-Markierungen und Bildschirmsteuerung

Konzept der Synchronisation

Es gibt keine effiziente Möglichkeit für ein Programm, ein Audio-Segment zu erkennen, wenn das Band läuft. Um das Synchronisations-Problem zu lösen, werden sog. Synchronisations-Markierungen (SYNC-Marks) benutzt. Diese enthalten das Signal, welches dem Programm mitteilt, daß ein Audio-Segment gespielt wurde (ein Audio-Segment kann entweder Musik oder gesprochene Anleitung sein, abhängig vom Programm).

Um es genauer zu formulieren: da die Audio-Daten keine Datei-Struktur besitzen, entspricht die auf der digitalen Spur befindliche SYNC-Mark einer End-of-Record-Markierung der Audio-Spur. So kann ein Programm entscheiden, wenn es eine SYNC-Mark erkennt, ob die Kassette für Berechnungen stoppen oder zum Spielen des nächsten Audio-Segments weiterlaufen soll.

SCHRITT 1:
Der Programmierer entwirft ein "Drehbuch" für das Märchen vom Frosch. Dieses Drehbuch würde wie folgt aussehen:

(MUSIK) HEUTE MÖCHTE ICH DIR DAS MÄRCHEN VON DER PRINZESSIN UND DEM FROSCH ERZÄHLEN. ES IST EINE NETTE GESCHICHTE, ALSO GEH´ NICHT WEG./

(MUSIK) BEVOR ICH MIT DER GESCHICHTE BEGINNE, WÜRDE ICH GERNE WISSEN, ZU WEM ICH SPRECHE. WIE HEIßT DU? GIB DEINEN NAMEN EIN UND DRÜCKE DIE RETURN-TASTE. (PAUSE)

(MUSIK) NUN DENN, FANGEN WIR AN. ES WAR EINMAL, VOR LANGER ZEIT. DA LEBTE EINE SCHÖNE PRINZESSIN MIT DEM NAMEN YYYY AUF EINEM SCHLOSS./

(MUSIK) EINES SCHÖNEN TAGES GING DIE PRINZESSIN .../

Anmerkung:

- "/" bedeutet, daß das Programm nach einer SYNC-Markierung sucht. Der Sprecher sollte für eine halbe Sekunde aufhören zu reden, bevor mit dem nächsten Audio-Segment begonnen wird.

- "Pause" zeigt an, daß der Sprecher für eine Sekunde stoppen sollte, damit der Motor genügend Zeit zum Aus- und Anlaufen hat. Jedes Audio-Segment sollte mindestens 10 bis 30 Sekunden lang sein, da zu viele SYNC-Markierungen auf engem Raum den Computer verwirren können.

SCHRITT 2:
Der Programmierer sollte, bevor er mit der Eingabe beginnt, einen Plan entwerfen, der das Verhältnis zwischen Bildschirm und Audio-Information angibt.

BEISPIEL:
Das folgende Beispiel (siehe nächste Seite) zeigt, wie ein Programmierer eine Kassette aufbauen sollte, die Kontrolle über eine Audio-Spur ausübt. Dieses Beispiel heißt "FROSCH":

SCHRITT 3:
Das Programm FROSCH könnte wie folgt aussehen:

```
10 REM Dieses Programm "FROSCH" soll die Synchro-
20 REM nisation von Audio- und Digital-Information
30 REM mit dem Kassetten-System demonstrieren.
35 REM
40 DIM IN$(20)
50 POKE 54018,52:REM Motor einschalten
60 GRAPHICS 1
```

```
70 PRINT #6;"DIE PRINZESSIN UND DER FROSCH":PRINT #G;....
:REM Aufbauen des Bildschirmes für 2. Teil.
80 GOSUB 1000:REM SYNC-Markierung überprüfen -  Die
Einführung muss abgelaufen sein.
100 POSITION X,Y:PRINT #6;"DEIN NAME?":REM Für Teil 4

105 GOSUB 1000:REM Teil 5
110 POKE 54018,60:REM Motor für Benutzer-Eingabe stoppen.
120 INPUT IN$:REM Name des Benutzers
130 POKE 54018,52
135 PRINT #6;CHR$(125):REM Bildschirm loeschen
140 POSITION X,Y:PRINT #6;IN$:PRINT #6;....:REM Bildschirm
für Teil 10.
150 GOSUB 1000:REM Garantiere, dass Sprache von Teil 10
beendet ist.
160 PRINT #6;....:REM Bereit für Teil 12.
```

Teil	Audio	Bildschirm	Überprüfe SYNC-Markierung	Motor-Modus
1				Ein
2	"Heute möchte ich...	Die Prinzessin und der Frosch (BILD)		
3			Ja	
4	"Bevor ich..."	Die Prinzessin und der Frosch (Bild) Wie heißt Du?XXXX		
5			Ja	
6				Stop
7		Warten bis eine Eingabe erkannt wird		
8				Start
9		Bildschirm löschen		
10	"Nun denn..."	XXXX Bild		
11			Ja	
12	"Eines schönen Tages..."	Bild		
13 :	:	:	:	:

UNTERPROGRAMM ZUM ÜBERPRÜFEN VON SYNC-MARKIERUNGEN:
Auf dem Band wird eine SYNC-Markierung durch ein "Leerzeichen" und eine "Nicht SYNC-Markierung" dargestellt

(Ein Leerzeichen ist eine "0"-Frequenz, d.h. eine niedrigere Frequenz als die einer Markierung mit einer "1"-Frequenz. Wie vorangehend erwähnt liegt eine Markierungs-Frequenz bei 5327Hz und eine Leerzeichen-Frequenz bei 3995Hz). Das Unterprogramm zum Suchen von SYNC-Markierungen wartet auf ein "Leerzeichen" vom seriellen Anschluss und sieht wie folgt aus:

```
1000 IF INT(PEEK(53775)/32+0.5)=INT(PEEK(53775)/32)THEN
RETURN : REM überprüfen des 5. Bits jede ankommenden Bytes.
Wenn eine Null auftaucht, ist das SYNC-Leerzeichen gefunden.
1010 GOTO 1000
```

ROUTINE FÜR DIE MOTORKONTROLLE:
Das Programm ist in er Lage, den Motor, durch Setzen der folgenden Daten an die Stelle 54018, an- und auszuschalten:

```
EIN: POKE 54018,52
AUS: POKE 54018,60
```

SCHRITT 4:
Nachdem das Audio-Drehbuch in Grundzügen festgelegt wurde, muß der Programmierer die für den Programm- und Audio-Teil des Bandes benötigte Zeit (inklusive Pausen) abschätzen. Ist der erforderliche Zeitraum größer als eine Cassettenlänge, so muß entweder das Programm oder das Drehbuch umgeschrieben werden.

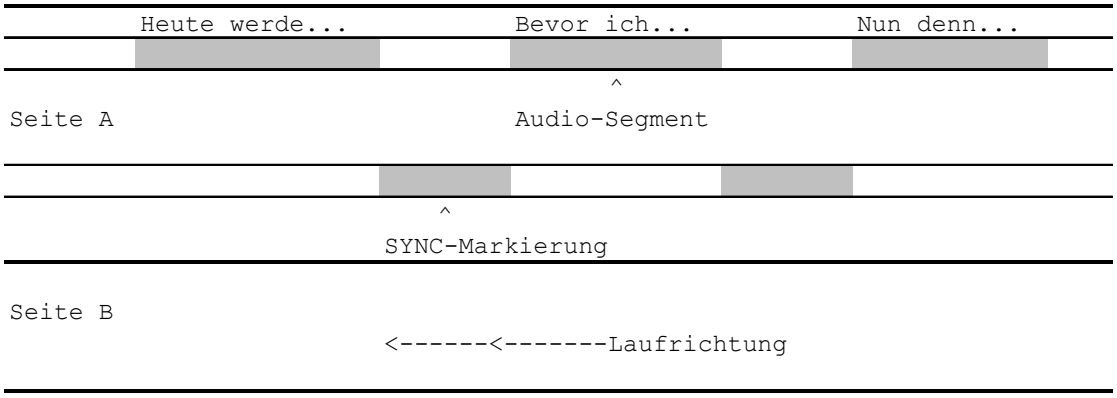
SCHRITT 5:
Sichern Sie das Programm auf ein Master-Tape, z.B. "Master 1".

SCHRITT 6:
Mit Hilfe des Drehbuches wird die Stimme bzw. die Musik (inklusive Pausen) auf ein anderes Master-Tape ("Master 2") geschrieben.

SCHRITT 7:
Nachdem diese beiden Urbänder fertiggestellt sind, werden sie zusammengemischt, um ein weiteres Master-Tape ("Master 3") zu erhalten. Für diesen Vorgang sind 3 RECORDING-LAB-MACHINES erforderlich. Erstellen Sie zwei Kopien von Urband "Master 3".

SCHRITT 8:

Laden Sie des SYNC-Mark-Programm in den ATARI Computer. Mit diesem werden ununterbrochen SYNC-Marks ("0"-Frequenz) auf die digitale Spur geschrieben. Die SYNC-Mark informiert das Programm, daß ein Audio-Segment gespielt wurde. Immer, wenn eine Pause auf der Audio-Spur auftaucht, wird eine SYNC-Mark an dieser Stelle benötigt. Das fertige Band mit Audio- und Digital-Spur würde wie folgt aussehen:



Das SYNC-Mark-Programm sieht folgendermaßen aus:

```
10 REM DRÜCKEN SIE DIE "START"-TASTE, UM EINE
20 REM SYNC-MARKIERUNG AUF DAS BAND ZU SCHREIBEN.
30 REM
40 REM
50 IO=53760:CONSOLE=53279:CASS=54018
100 FOR I=0 TO 8
110 READ J:POKE IO+I,J
120 NEXT I
125 REM DIE FOR-SCHLEIFE SETZT AUDIO-FREQUENZ UND
126 REM KANAL
130 DATA 5,160,7,160,5,160,7,160,0
140 REM
150 REM I/0 IST ERSTELLT: JETZT STARTE CASSETTE
160 POKE CASS,52
200 POKE CONSOLE,8
210 IF PEEK(CONSOLE)<>7 THEN 230:REM CONSOLE=7
 BEDEUTET MARKIERUNG SCHREIBEN.
220 POKE IO+15,11:GOTO 200:REM KEINE TASTE DER
 KONSOLE GEDRUCKT
230 POKE IO+15,128+11:GOTO 200:REM WENN CONSOLE<>7
 SCHREIBE "LEERZEICHEN"
```

SCHRITT 9:
Legen Sie beide "Master 3"-Bänder in zwei unterschiedliche Maschinen und spulen Sie sie zurück. Schließen Sie einen der Recorder an einen ATARI Computer an, in dem sich das SYNC-Mark-Programm befindet. Dieser Recorder wird zur Aufnahme der SYNC-Markierungen auf die Digital-Spur benutzt. Der andere Recorder wird später für die Audio-Aufnahme verwendet.

SCHRITT 10:
Geben Sie "RUN" zum Starten des SYNC-Markierunge-Programms ein. Starten Sie gleichzeitig die beiden Recorder, einen für die Aufnahme, den anderen für die Wiedergabe. Achten Sie auf die Audio-Information und drücken Sie jedesmal die "START"-Taste, wenn dieses durch eine Pause angezeigt wird.

SCHRITT 11:
Das Band ist nun, nach Aufnahme von Programm, Audio-Information und SYNC-Markierungen, fertig. Das Band ist bereit für eine Massenproduktion.

2.5 Verriegeln der BREAK-Taste

Es kann vorkommen, daß der Programmierer die BREAK-Taste verriegeln möchte. Um zu verhindern, daß der Ladevorgang eines Cassetten-Programms unterbrochen wird, wenn er zufällig auf die BREAK-Taste drückt. Das OS kann ein Record nicht wiederfinden, solange der Benutzer nicht in der Lage ist, das Band zu diesem verlorenen Record zurückzuspulen. Die Routine zum Abschalten der BREAK-Taste sieht wie folgt aus:

```
4000 X=PEEK(16):IF X<128 THEN 4020
4010 POKE 16,X-128:POKE 53774,X-128
4020 RETURN
```

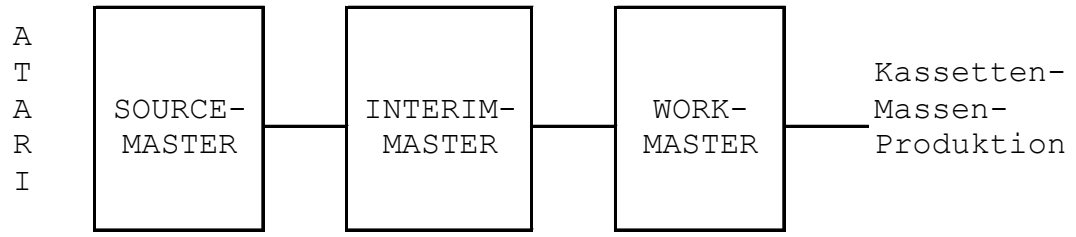
Diese Abschalt-Routine muß nach jedem Öffnen des Bildschirmes und jeder Änderung des Graphik-Modus aufgerufen werden.

2.6 Massenproduktion

Der Programmierer produziert ein oder mehrere Urbänder, entsprechend den in Abschnitten 2.2, 2.3 und 2.4 besprochenen Aufnahme-Techniken. Alle ATARI"TM"-Urbänder werden auf einer offenen 1/4-Spur-Bandspule mit einem

1/4-Zoll-Band bei einer Geschwindigkeit von 7 1/2 IpS aufgenommen. Das “MASTER-TAPE” geht als “SOURCE-MASTER2” an den Duplikator.

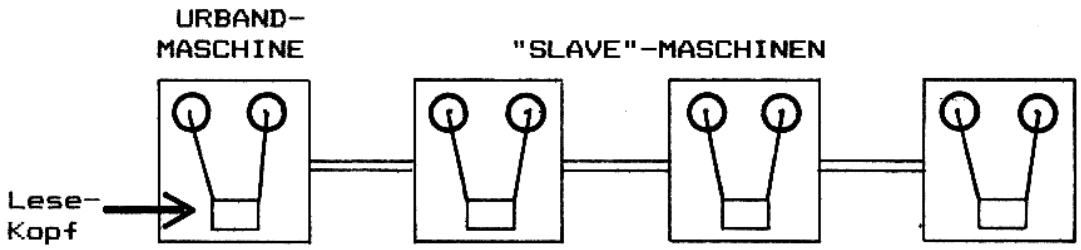
Der Duplikator benutzt das “SOURCEMASTER2”-Band zur Erstellung eines “WORK-MASTER”-Bandes für die endgültige Massenproduktion. Das abgegebene Produkt ist also die dritte Generation. Die folgende Darstellung zeigt diesen Vorgang:

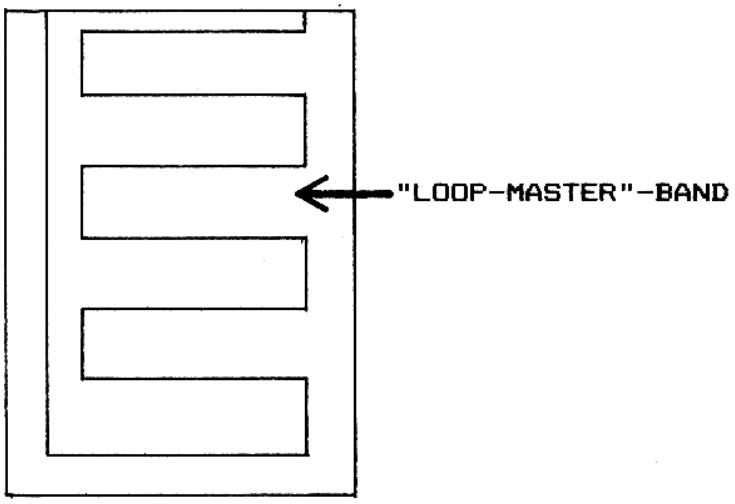


Das “INTERIM-MASTER”-Band ist für den Duplikator erforderlich, da das “WORK-MASTER”-Band zerstört oder durch die häufige Benutzung verbraucht werden kann. Das “SOURCE-MASTER”-Band muß für absolute Notfälle reserviert werden. Das “INTERIM-MASTER”-Band ist die Arbeits-Kopie des “WORK-MASTER”-Bandes.

2.6.1 Massenproduktion von Kassetten

Zum gegenwärtigen Zeitpunkt wird von ARTARI"TM" die “BIN-LOOP”-Methode für die Massenproduktion bevorzugt. Das “WORK-MASTER”-Band wird zur Erzeugung eines “LOOP-MASTER”-Bandes (Schleifen-Urband) kopiert. Letzteres kann jede beliebige Bandbreite besitzen. Das “BIN-LOOP”-Band wird zu einem “CONTINUOUS-LOOP”-Band zusammengefügt, welches einen kurzen leeren Führer-Ton an der Klebestelle besitzt. Diesen Band wird in eine Urband-Maschine gebracht, die mit einer hohen Geschwindigkeit läuft und an die mehrere “SLAVE” (Sklave) - Maschinen angeschlossen sind. Die Anordnung sieht wie folgt aus:





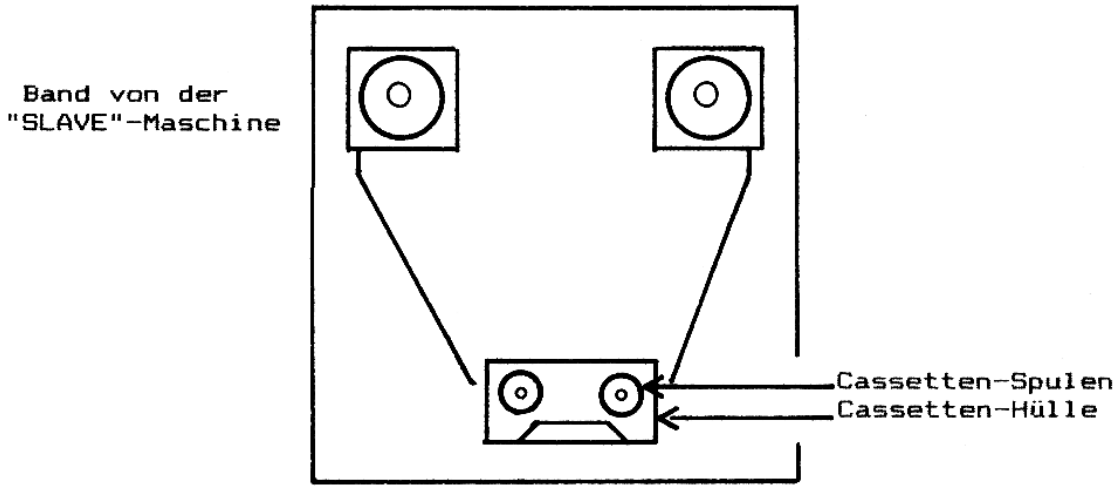
Das "LOOP-MASTER"-Band wird wiederholt gelesen. Wenn z.B. 100 Kassetten produziert werden sollen, wo wird die Länge das Bandes der "SLAVE"-Maschine gemessen, indem das Programm mit 100 multipliziert wird. Auf dar "MASTER"-Maschine befindet sich ein Zähler, der auf 100 gesetzt wird.

Beim Lesen des "LOOP-MASTER"-Bandes werden alle 4 Spuren auf das Band dar "SLAVE"-Maschine geschrieben.

Wird auf den leeren Teil des "LOOP-MASTER"-Bandes gestoßen, so produziert die Maschine einen "SCHNEIDE-TON", der auf eine oder mehrere Spuren der Bänder in den "SLAVE"-Maschinen geschrieben wird. Der Zähler wird danach um 1 erhöht.

Auf jedem fertigen Band der "SLAVE"-Maschine befinden sich 100 Programme und 100 zugehörige "SCHNEIDE-TÖNE". Diese Bänder werden in automatische Lade-Maschinen eingelegt, die erstere in Kassetten-Hüllen spulen. Die Anordnung sieht folgendermaßen aus:

LADE-MASCHINE



Die Kassetten-Hüllen werden mit einem kurzen Stück Leerband geliefert, das auf die Cassetten-Spulen aufgewickelt ist. Die Lade-Maschine zieht dieses Stück aus der Hülle, schneidet es und verklebt es mit dem Bandanfang eines Bandes der "SLAVE"-Maschinen. Danach wird das "SLAVE"-Band in die Kassette gewickelt, bis ein "SCHNEIDE-TON" erscheint. Die Lade-Maschine schneidet an dieser Stelle das Band der "SLAVE"-Maschine erneut und verklebt es mit dem anderen Teil des Leerbandes in der Kassetten-Hülle. Schließlich wird das fertige Band in die Hülle gewickelt und mit der nächsten Kassette begonnen.

Die Kassetten-Hülle wird entweder per Hand oder automatisch in die Lade-Maschine gelegt oder aus ihr genommen, wenn das Band fertig aufgewickelt ist.

2.6.2 Qualitäts-Überprüfung

Jedes mal, wenn eine Produktserie hergestellt wird, müssen einzelne Produkte dieser Serie überprüft werden, bevor erstere ausgeliefert und verkauft werden kann.

Diese Qualitäts-Überprüfung wird im allgemeinen am ersten und letzten Produkt der Serie vorgenommen.

Anhang IV
Beispiel für Fließkomma-Arithmetik

```
0000      20        *= 4000        ;BELIEBIGER STARTPUNKT
DDB6      30   FMOVE = $DDB6
DA60      40   FSUB  = $DA60
0482      50   FTEMP = $0482
DDA7      60   FSTOR = $DDA7
D8E6      70   FASC  = $D8E6
00F3      90   INBUFF= $00F3
D800      85   AFP   = $D800
00F2      90   CIX   = $00F2
0580      0100 LBUFF = $0580
          0110 ;
0098      0120 CR    = $9B
0009      0130 PUTREC= $09
0005      0140 GETREC= $05
E456      0150 CIOV  = $E456
0342      0160 ICCOM = $0342
0344      0170 ICBAL = $0344
0348      0180 ICBLL = $0348
          0190 ; VON CAROL SHAW
          0200 ; DEMO-PROGRAMM ZUR FLIESSKOMMA-ARITHMETIK.
          0210 ; ES WERDEN ZWEI ZAHLEN VOM BILDSCHIRM-
          0211 ; EDITOR EINGELESEN UND IN FLIESSKOMMA-WERTE
          0212 ; UMGEWANDELT, DANACH WIRD DIE ERSTE ZAHL
          0220 ; VON DER ZWEITEN SUBTRAHIERT, SPEICHERT DAS
          0221 ; ERGEBNIS IN FTEMP (DURCH DEN BENUTZER
          0222 ; FESTGELEGTES RESISTER) UND ZEIGT ES AUF
          0230 ; DEM BILDSCHIRM AN.
          0240 ;
          0250 START
4000 205340 0260 JSR GETNUM   ;ERSTE ZAHL VON E: IN F.K.
                               UMWANDELN
4003 20B6DD 0270 JSR FMOVE    ;ZAHL VON FR0 NACH FR1
                               BRINGEN
4006 205340 0280 JSR GETNUM   ;ZWEITE ZAHL VON E: IN F.K.
                               UMWANDELN - AUSLASSEN, WENN
                               NUR EIN ARGUMENT
4009 2060DA 0290 JSR FSUB     ;FR0 <== FR0 - FR1
400C 900A   0300 BCC NOERR    ;VERZWEIGEN, WENN KEIN FEHLER
            0310 ;
            0320 ; FEHLER -- MITTEILUNG ANZEIGEN
            0330 ;
400E A981   0340 LDA #ERRMSG&255
4010 8D4403 0350 STA ICBAL
4013 A940   0360 LDA #ERRMSG/256
```

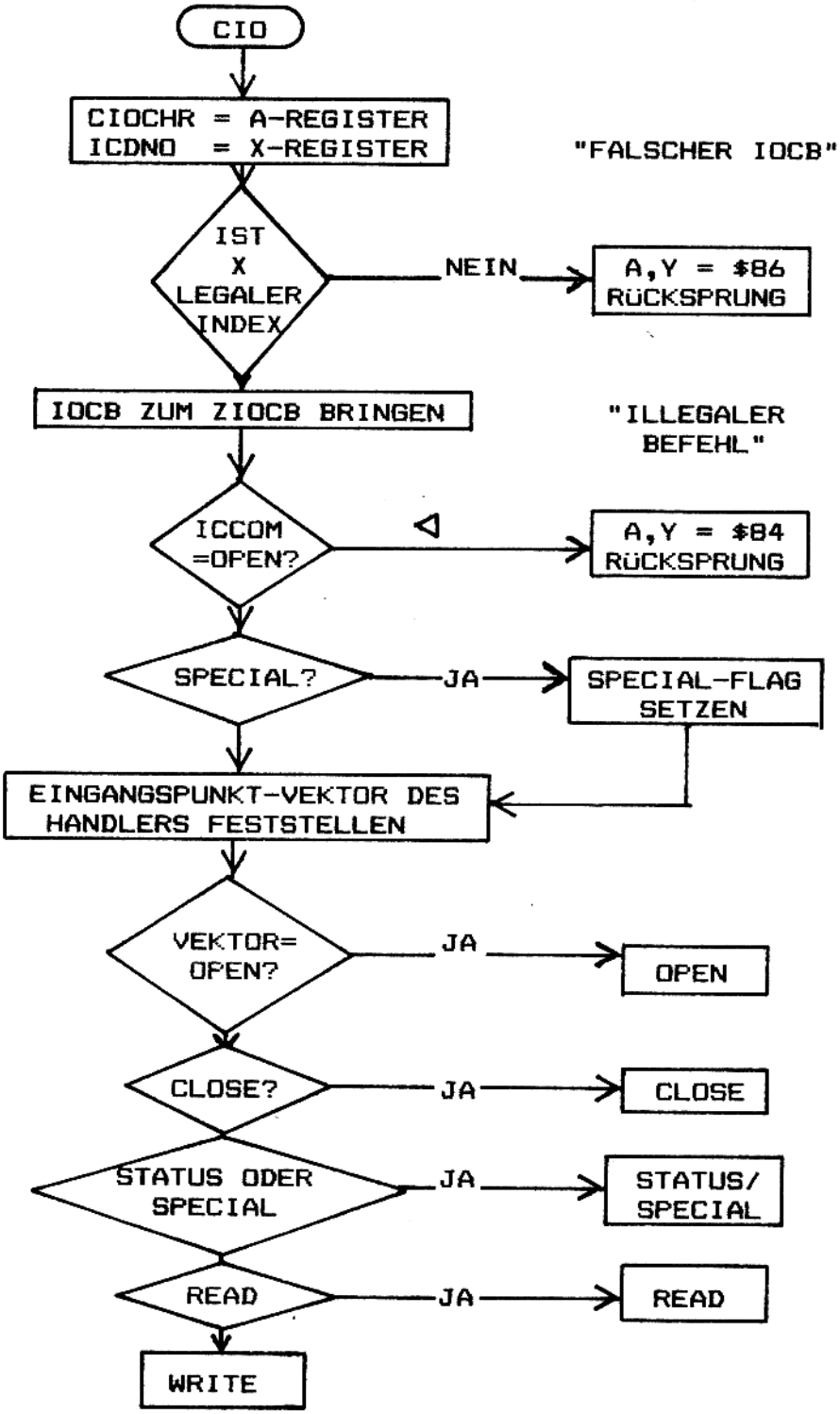
```
4015 4C3940 0370 JMP CONTIN
            0380 NOERR
4018 A282   0390 LDX #FTEMP&255    ;ERGENBIS IN FTEMP
                                    SPEICHERN
401A A004   0400 LDY #FTEMP/256
401C 20A7DD 0410 JSR FSTOR
            0420 ;
            0430 ; ZAHL IN ATASCII-STRING
            0440 ; UMWANDELN. ENDE DES STRINGS
            0450 ; SUCHEN UND NEGATIVE ZAHL IN
            0451 ; POSITIVE UMWANDELN UND ZEILEN-
            0452 ; VORSCHUB ANFÜGEN.
            0453 ;
401F 20E6DB 0460 JSR FASC          ;VON F.K. IN ASCII-
                                    STRING NACH LBUFF
                                    UMWANDELN
4022 A0FF   0470 LDY #$FF
            0480 MLOOP
4024 C8     0490 INY
4025 B1F3   0500 LDA (INBUFF),Y    ;NÄCHTES
                                    BYTE LADEN (WIRD
                                    DURCH INBUFF ANGE-
                                    ZEIGT) POSITIV?
4027 10FB   0510 BPL MLOOP         ;JA - FORTFAHREN
4029 297F   0520 AND #$7F          ;NEIN. NEGATIV -
                                    MSBIT AUSMASKIEREN
402B 91F3   0530 STA (INBUFF),Y
402D CS     0540 INY
402E A99B   0550 LDA #CR           ;ZEILENVORSCHUB SPEICHERN
4030 91F3   0560 STA (INBUFF),Y
            0570 ;
            0580 ;ERGEBNIS ANZEIBEN
            0590 ;
4032 A5F3   0600 LDA INBUFF        ;PUFFER-ADRESSE STEHT IN
                                    INBUFF
4034 8D4403 0610 STA ICBAL
4037 A5F4   0620 LDA INBUFF+l
            0630 CONTIN
4039 8D4503 0640 STA ICBAL+l
403C A909   0650 LDA #PUTREC       ;PUTREC-KOMMANDO
403E 8D4203 0660 STA ICCOM
4041 A928   0670 LDA #40           ;PUFFER-LÄNGE = 40
4043 8D4B03 0680 STA ICBLL
4046 A900   0690 LDA #0
4048 8D4903 0700 STA ICBLL+l
404B A200   0710 LDX #0            ;IOCB NR.-0 (BILDSCHIRM-
                                    EDITOR
404D 2056E4 0720 JSR CIOV          ;CIO AUFRUFEN
4050 4C0040 0730 JMP START         ;VON VORNE ANFANGEN
            0740 ;
```

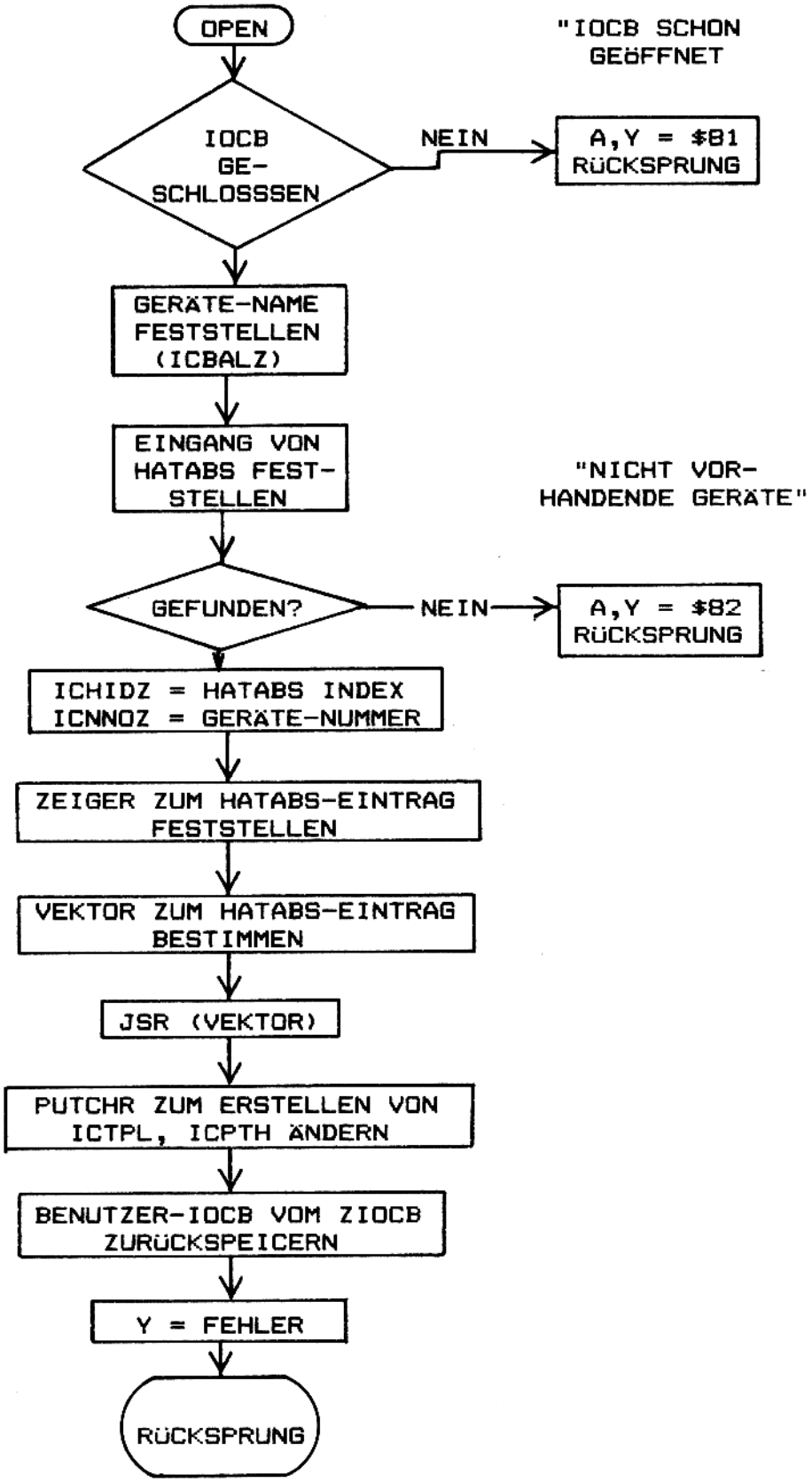
```
           0750 ; GETNUM -- ASCII-STRING VON E: LESEN
           0755 ; UND ZU F.K. NACH FRO UMWANDELN.
           0760 ;
           0770 GETNUM
4053 A905  0780 LDA #GETREC       ;GET RECORD (ENDE MIT CR)
4055 8D4203 0790 STA ICCOM
4058 A980  0800 LDA #LBUFF&255    ;BUFFER-ADRESSE = LBUFF
405A 8D4403 0810 STA ICBAL
405D A905  0820 LDA #LBUFF/256
405F 8D4503 0830 STA ICBAL+1
4062 A929  0840 LDA #40           ;PUFFER-LANGE = 40
4064 8D4803 0850 STA ICBLL
4067 A900  0860 LDA #0
4069 8D4903 0870 STA ICBLL+1
406C A200  0880 LDX #0            ;IOCB NR.-0(BILD-
                                   SCHIRM-EDITOR)
406E 2056E4 0890 JSR CIOV         ;CIO AUFRUFEN
4071 A980  0900 LDA #BUFF&255     ;PUFFER-ADRESSE IM
                                   ZEIGER (INBUFF)
                                   SPEICHERN
4073 85F3  0910 STA INBUFF
4075 A905  0920 LDA #LBUFF/256
4077 85F4  0930 STA INBUFF+1
4079 A900  0940 LDA #0            ;BUFFER-INDEX = 0
407B 85F2  0950 STA CIX
407D 4C00D8 0960 JMP AFP          ;AUFRUFEN DER ASCII ZU
                                   F.K. UND RüCKSPRUNG
4080 60    0970 INIT RTS          ;EINSCHALT (POWER-UP)-
                                   ROUTINE(KEINE AKTION)
4081 45    0980 ERRMSG .BYTE "ERROR",CR
                                  ;ZEIGT GESETZTES
                                   CARRY-BIT BEI
                                   RÜCKSPRUNG VON
                                   F.K.-ROUTINE AN.
4082 52
4083 52
4084 F4
4085 52
4086 9B
            0990 ;
            1000 ; ROUTINEN-START INFORMATION
            1010 ;
4087        1020 *= $2E2
02E0 0040   1030 .WORD START
02E2        1040 .END
```

Routinen der Fließomma-Arithmetik

Name	Adresse	Beschreibung	ung. max. Dauer in (µsec)
AFP	D800	ASCII in Fließkomma	3500
FASC	D8E6	Fließkomma in ASCII	950
IFP	D9AA	Integer in Fließkomma	1330
FPI	D9D2	Fließkomma in Integer	2400
FSUB	DA60	FR0 <= FR0-FR1 Subtraktion	740
FADD	DA66	FR0 <= FR0+FR1 Addition	710
FMUL	DADB	FR0 <= FR0*FR1 Multiplikation	12000
FDIV	DB28	FR0 <= FR0/FR1 Division	10000
FLDOR	DD89	FR0 mit X,Y Fließk-laden	70
FLDOP	DD8D	FR0 mit FLPTR Fließk.-laden	60
FLD1R	DD98	FR1 mit X,Y Fließk-laden	70
FLD1P	DD9C	FR1 mit FLPTR Fließk-laden	60
FSTOR	DDA7	FR0 mit XY Fl.K. speichern	70
FSTOP	DDAB	FR0 mit Fl.K. speichern	70
FMOVE	DDB6	FR0 <= FR1 F.K.-Bewegung	60
PLYEVL	DD40	Polynom-Berechnung	88300
		FR0	
EXP	DDC0	FR0 <= E Exponentialfunktion	115900~.1s
		FR0	
EXP10	DDCC	FR0 <= 10 Exponentialfunktion	108800
LOG	DECD	FR0 <= LOG (FR0) nat. LOG	136000
		E	
LOG10	DED1	FR0 <= LOG (FR0) dek. LOG.	125400
		10	
ZFR0	DA44	FR0 <= 0	80
AF1	DA46	ZERO-PAGE F.K.Register	80
		löschen (6 Bytes)	
Im BASIC-			
Modul			
SIN	BDA7	FR0 <= SIN(FR0)	79400
COS	BDB1	FR0 <= COS(FR0)	77400
ATAN	BE77	FR0 <= ATAN(FR0)	126700
SQR	BEE5	FR0 <= Quadratwurzel(FR0)	131100

Zeitangaben ink. JSR und RTS
Alle Zeiten sin ungefähre Werte:
1 Sekunde = 1.000.000µsec
Durch Abschaltung des DMA (SDMCR)können 30%-40% der Zeit
eingespart werden: POKE 559,0(AUS), POKE 559,34(EIN)


CIO
CIOCHR = A-REGISTER
ICDNO = X-REGISTER
"FALSCHER IOCB"
IST X LEGALER INDEX
NEIN
A,Y = $86
RÜCKSPRUNG
IOCB ZUM ZIOCB BRINGEN
"ILLEGALER BEFEHL"
ICCOM =OPEN?
A,Y = $84
RÜCKSPRUNG
SPECIAL?
JA
SPECIAL-FLAG SETZEN
EINGANGSPUNKT-VEKTOR DES HANDLERS FESTSTELLEN
VEKTOR= OPEN?
JA
OPEN
CLOSE?
JA
CLOSE
STATUS ODER SPECIAL
JA
STATUS/ SPECIAL
READ
JA
READ
WRITE


OPEN
"IOCB SCHON GEöFFNET
IOCB GE-SCHLOSSSEN
NEIN
A,Y = $81 RÜCKSPRUNG
GERÄTE-NAME FESTSTELLEN (ICBALZ)
EINGANG VON HATABS FEST-STELLEN
"NICHT VOR-HANDENDE GERÄTE"
GEFUNDEN?
NEIN
A,Y = $82 RÜCKSPRUNG
ICHIDZ = HATABS INDEX
ICNNOZ = GERÄTE-NUMMER
ZEIGER ZUM HATABS-EINTRAG FESTSTELLEN
VEKTOR ZUM HATABS-EINTRAG BESTIMMEN
JSR (VEKTOR)
PUTCHR ZUM ERSTELLEN VON ICTPL, ICPTH ÄNDERN
BENUTZER-IOCB VOM ZIOCB ZURÜCKSPEICERN
Y = FEHLER
RÜCKSPRUNG

Anhang VI
Freier Zugriff

DEFINITION DES FREIEN ZUGRIFFS

Der freie Zugriff ist als Methode definiert, die Datensätze von oder auf jeden beliebigen Teil einer Datei liest bzw. schreibt, wobei es nicht erforderlich ist, die voranliegenen Datensätze der Datei zu lesen/schreiben.

Bevor ein I/O-Kommando verarbeitet werden kann, muß das Speichermedium des Gerätes physikalisch an der richtigen (Byte-)Stelle positioniert werden. Ein sequentielles Gerät, wie z.B. der ATARI-Programmrecorder, muß per Hand an die richtige Position gebracht werden. In diesem Fall geschieht das über das Zählwerk des Recorders. Soll also auf den 100. Datensatz zugegriffen werden, so müssen die Sätze 1 bis 99 übersprungen werden. Es ist offensichtlich, daß dieses aufgrund der physikalischen Eigenschaften des Gerätes sehr viel Zeit benötigt.

Ein Gerät, wie die ATARI-Diskettenstation, kann durch ein BASIC-Programm an jede Byte-Position gebracht werden. Wenn z.B. der 100. Datensatz gelesen werden soll, so kann dies sofort geschehen, indem der Lesekopf an die entsprechende Stelle positioniert wird. Der freie Zugriff wird z.B. durch das POINT-Kommando in BASIC möglich.

```
OPEN #1,12,0,"D1:--------.---”
SECTOR=63
BYTE=26
POINT #1,SECTOR,BYTE
```

Diese Befehle verursachen, daß die auf Kanal 1 geöffnete Datei auf den Sektor 63 und das 26. Byte positioniert wird. Die Datei muß hierfür bereits in Modus 12 geöffnet und Sektor 63 durch das File-Manager-System (FMS) anerkannt worden sein.

ZIELE

1. Definition der Struktur der Datei, die den freien Zugriff vereinfacht.

2. Maximum der Disketten-Nutzung durch Ausnutzung alles freien Benutzerraumes einer Datei.

3. Vereinfachen von Programmen durch Aufstellen von Unterroutinen, welche die allgemeinen Berechnungen für den freien Zugriff übernehmen.

VORSTELLUNGEN

1. Der freie Zugriff kann auf 2 Arten vereinfacht werden:

Als erstes wird die von POINT-Kommando benötigte Sektor- und Byte-Information in der Datei gespeichert, so daß diese einer Variablen eines Programms zugeordnet werden kann, sobald die Datei geöffnet wurde. Zweitens muß die Programm-Variable so wenig RAM wie möglich verbrauchen.

Ein zweidimensionales numerisches Feld könnte z.B. als Programm-Variable benutzt werden. So würde jeder Datensatz 12 Bytes des Speicher verbrauchen. Eine String-Variable würde weniger RAM benötigen, da die Nummern des Sektors (0-720) in 2 Bytes und die des Bytes (0-127) in einem Byte gespeichert werden können.

2. Die maximale Nutzung der Diskette wird durch Verwendung von Modus 12 (I/O) anstelle von Modus 9 (Anfügen) erreicht.

Ein gibt 2 Probleme, wenn mit Hilfe von Modus 9 auf eine Datei zugegriffen wird. Bei der Verarbeitung eines OPEN-Kommandos legt das FMS als erstes einen neuen Sektor für die Datei fest, gleichgültig ob der vorige bereits vollständig mit Daten gefüllt ist oder nicht. Als zweites arbeitet der POINT-Befehl nicht bei Dateien, die in Modus 9 geöffnet wurden.

In Modus 12 ist es möglich beim Aufbauen der Datei leere Datensätze zu erzeugen.

3. Datensatzerzeuger wird durch jeweils ein Status Byte pro Satz vereinfacht. Ein Wert von 0 zeigt an, daß der Satz frei ist; ein Wert von 1 zeigt an, daß der Satz benutzt wird.

DATEI-STRUKTUR

Eine Datei besteht aus 3 Abschnitten:

DATEIKOPF
DATEIZEIBER (FÜR DEN FREIEN ZUGRIFF)
DATEN-SÄTZE

Der Dateikopf ist der erste Sektor der Datei und ist 125 Bytes lang (124 Datenbytes + 1 Begrenzer).

FILE-HEADER RECORD

BYTE	INHALT
1-2	Sektor-Adresse des Dateikopfes.
3	Byte-Offset des Dateikopfes
4	Nicht benutzt
5-6	Anzahl der Datensätze in der Datei
7-8	Anzahl der Bytes je Datensatz
9-124	Nicht benutzt

Die Dateizeiger folgen dem Dateikopf und beginnen daher im zweiten Sektor der Datei. Die Daten werden als String-Variable in Gruppen von je 4 Bytes gespeichert. Die ersten 4 Bytes werden zur Positions-Festlegung des Zeiger-Strings innerhalb der Datei benutzt. Danach folgen jeweils 4 Bytes für jeden Datensatz in der Datei.

DATEIZEIGER

BYTE	INHALT
1-2	Sektor-Adresse der Zeiger für den freien Zugriff
3	Byte-Offset der Zeiger für den freien Zugriff
4	Nicht benutzt
5-N	4 Bytes für jeden Datensatz
	1-2 Sektor Adresse des Datensatzes
	3 Byte-Offset des Datensatzes
	4 Status des Datensatzes

Die Datensätze folgen den Zeigern und werden als String-Variable gespeichert.

UNTERROUTINEN FÜR DEN FREIEN ZUGRIFF

1. FILEOPEN.SUB

Diese Routine öffnet eine Datei im Modus 12 und initialisiert die Variablen für den freien Zugriff, die in anderen Routinen benutzt werden.

Eingabe-Variablen:

```
FILE$     - wird vom Benutzer dimensioniert und
            angegeben (15 Bytes)
Channel   - IOCB-nummer (1-5)
```

Aufruf:

```
GOSUB 9300
```

Ausgabe-Variablen:

```
FILEMAX   - Anzahl der Datensätze
FILELEN   - Anzahl der Bytes je Datensatz
FILEPTR$  - enthält die Dateizeiger
FILEREC$  - wird für Datensatz I/O dimensioniert
```

Lokale Variablen:

```
FILESEC
FILEBYT     Diese werden im Augenblick nicht
FILESTS     benutzt
```

2. FILEADD.SUB

Diese Routine bestimmt den nächsten verfügbaren Datensatz, indem in FILEPTR$ nach einem Status-Byte mit dem Wert 0 gesucht wird. Wird dieser Wert gefunden, so wird er auf 1 gesetzt und die Nummer des Datensatzes in RECORD gespeichert. Wird RECORD zu 0, dann sind alle Datensätze belegt.

Eingabe-Variablen:

```
FILEMAX
```

Aufruf:

```
GOSUB 9400
```

Ausgabe-Variable:

```
RECORD    - Nummer des nächsten verfügbaren
            Datensatzes
FILEPTR$  - wird mit Status-Byte von 1
            aktualisiert.
```

Lokale-Variablen:
RECORD1
B

3. FILEDEL.SUB

Diese Routine markiert einen belegten Datensatz durch Setzen des entsprechenden Status-Bytes in FILEPTR$ auf 0 in einen freien Datensatz.

Eingabe-Variablen:
RECORD

Aufruf:
GOSUB 9450

Ausgabe-Variablen:
FILEPTR$ - ein Status-Byte wird mit 0 aktualisiert.

Lokale Variablen:
B

4. FILPTR.SUB

Diese Routine aktualisiert die Dateizeiger auf der Diskette.

Eingabe-Variablen:
Keine

Aufruf:
GOSUB 9500

Ausgabe-Variablen:
keine

Lokale Variablen:
S
B

5. FILEPOS.SUB

Diese Routine setzt den Dateizeiger an den Anfang eines Datensatzes.

```
     Eingabe-Variablen:
          RECORD    - Nummer des Datensatzes, auf den
                      gezeigt werden soll.

     Aufruf:
          GOSUB 9600

     Lokale-Variablen:
          S
          B


9200 REM ANHANG UND UNTERROUTINE VON WILLIAM BARLETT
9300 REM "FILEOPEN.SUB" WBB 30.3.81
9305 REM ÖFFNEN EINER DATEI IM MODUS 12 UND DEFINITION
9307 REM ALLER VARIABLEN
9310 OPEN #CHANNEL,12,0,FILE$
9315 DIM FILHED$(124)
9320 FOR I=1 TO 124:GET #CHANNEL,B:FILEHED$(I)=CHRS(B):
NEXT I:GET #CHANNEL,B
9325 FILESEC=ASC(FILEHED$(1)*256+ASC(FILEHED$(2))
9330 FILEBYT=ASC(FILEHED$(3))
9335 FILESTS=ASC(FILEHED$(4))
9340 FILEMAX=ASC(FILEHED$(5))*256+ASC(FILEHED$(6))
9345 FILELEN=ASC(FILEHED$(7))*256+ASC(FILEHED$(8))
9350 DIM FILEPTR$(4+4*FILEMAX),FILEREC$(FILELEN)
9355 FOR l=1 TO 4+4*FILEMAX:GET #CHANNEL,B:FILEPTR$(I)=
CHR$(B):NEXT I:GET #CHANNEL,B
9360 RETURN

9400 REM "FILEADD.SUB" WBB 30.3.91
9405 REM BESTIMMEN DES NÄCHSTEN FREIEN DATENSATZES
9410 RECORD=0
9415 IF FILEMAX=0 THEN RETURN
9420 FOR RECORD1=l TO FILEMAX
9425 B=RECORD1*4+4
9430 IF FILEPTR$(B,B)=CHR$(0) THEN RECORD=RECORD1:RECORD1=
FILEMAX:FILEPTR$(B,B)=CHR$(1)
9435 NEXT RECORD1
9440 RETURN

9450 REM "FILEDEL.SUB" WBB 30.3.Bl
9455 REM FREIMACHEN EINES BELEGTEN DATENSATZES
9460 B=RECORD*4+4
9465 FILEPTR$(B,B)=CHR$(0)
9470 RETURN
```

```
9500 REM "FILEPTR.SUB" WBB 19.3.81
9510 REM FILEPTR$ SCHREIBEN
9515 S=ASC(FILEPTR$(100*256+ASC(FILEPTR$(2))
9520 B=ASC(FILEPTR$*(3))
9525 POINT #CHANNEL,S,B
9530 FOR I=l TO 4+4*FILEMAX:B=ASC(FILEPTR$)1)):PUT
#CHANNEL,B: NEXT I
9535 RETURN

9600 REM "FILEPOS.SUB" WBB 31.3.81
9605 REM ZEIGE FILE ZU RECORD
9610
S=ASC(FILEPTR$(RCORD*4+1))*256+ASC(FILEPTR$(RECORD*4+2))
9615 B=ASC(FILEPTR$(RECORD*4+3))
9620 STS=ACS(FILEPTR$(RECORD*4+4))
9625 POINT #CHANNEL,S,B
9630 RETURN
```

I. EINFÜHRUNG ZU FILE001

Dieses Programm wird zur Erstellung einer neuen Datei auf Diskette und Initialisierung deren Struktur benutzt. Die Struktur des freien Zugriffe wird für das Arbeiten mit den folgenden BASIC-Unterroutinen angelegt

FILEOPEN.SUB
FILEADD.SUB
FILEDEL.SUB
FILEPTR.SUB
FILEPOS.SUB

II. PROGRAMM-STRUKTUR

0001-0999 Hauptprogramm
1000-9999 Unterrautinen

III. PROGRAMM-LOGIK

Die fünf wesentlichen Programmteile sind Programm-Initializierung, Datei-Definition, Bildschirm-Aufbau, File-Festlegung und Schließen.

A. Initialisierung 1000-1495

1015 Variablen dimensionieren

```
1020      Programmnamen anzeigen
1025-1035 Benutzer Programmabbruch gestatten
1040      BLANK$ als String-Füller vorbereiten
```

B. Datei-Definition 1500-1999

```
1510-1535 Benutzer definiert die Datei-Parameter
1540-1550 Benutzer berichtigt die Parameter
1600-1645 Disketten-Verzeichnis überprüfen, ob die
          angegebene Datei bereits vorhanden ist
1700-1730 Überprüfung, ob genügend Sektoren zum
          Erstellen einer Datei vorhanden sind
1800-1825 Dateinamen aufbauen
```

C. Bildschirm-Aufbau 9600-9650

D. Datei-Festlegung 2000-2499

```
2005      Datei aufbauen
2010      Zeiger sichern
2015      Variablen dimensionieren
          FILEHED$ - Dateikopf
          FILEPTR$ - Dateizeiger
          FILEREC$ - Datensatz-I/O-Variable
2100-2180 FILEHED$ erstellen und sichern
2200-2245 FILEPTR$ erstellen und sichern
2305      FILEREC$ auffüllen
2310-2345 Dateizeiger in FILEPTR$ speichern
2350      Bildschirm erneuern
2355      FILEREC$ auf Diskette speichern
2370-2375 Abschließenden FILEPTR$ auf Diskette speichern
```

E. Abschließen 0900-0999

```
10 REM "FILE0001" WBB 12.3.81
100 REM HAUPTPROGRAMM
110 GOSUB 1000
120 IF YN$="N" THEN 900
130 GOSUB 1500
140 GOSUB 9600
150 GOSUB 2000
900 REM ENDE
910 CLOSE
920 GRAPHICS 0
930 END
1000 REM INITIALISIERUNG
```

```
1005 TRAP 9800
1010 GRAPHICS 2
1015 DIM YN$(1),DVC$(3),FILE$(8),EST$(3),FILENAME$(15),
FMS$(16),BLANK$(128)
1020 PLOT 5,4:PRINT #6;"FILE0001"
1025 PRINT "DIESES PROGRAMM INITIALISIERT EINE NEU DATEI."
1030 PRINT "WOLLEN SIE FORTFAHREN(J/N)";
1035 INPUT YN$
1040 BLANK$=" ":BLANK$(128)= BLANK$(2)
1095 RETURN
1500 REM FILE DEFINIEREN
1505 GRAPHICS 2
1510 PRINT #6,"DATEI-DEFINITION"
1515 PRINT "       GERÄT  ";:INPUT
DVC$:DVC$(LEN(DVC$)+1)=":"
1520 PRINT "    DATEINAME  ";:INPUT FILE$
1525 PRINT "  ERWEITERUNG  ";:INPUT EXT$
1530 PRINT "   SATZANZAHL  ";:INPUT FILEMAX
1535 PRINT "    SATZLÄNGE  ";:INPUT FILELEN
1540 PRINT "WOLLEN SIE FORTFAHREN (J/N) ";
1545 INPUT YN$
1550 IF YN$<>"J"" THEN 1500
1600 REM ÜBERPRÜFUNG OB DATEI VORHANDEN
1605 FMS$=DVC$FMS$(LEN(FMS$)+1)"*.*"
1610 OPEN #1,6,0,FMS$
1615 INPUT #1,FMS$
1620 IF FMS$(2,2)<>" " THEN 1700
1625 IF FMS$(3,2+LEN(FILE$))<>FILE$ OR FMS$(11,10+LEN(EXT$))
<>EXT$ THEN 1615
1630 CLOSE #1
1635 PRINT "DIE DATEI IST BEREITS VORHANDEN!";CHR$(253)
1640 GOSUB 9850
1645 BOTO 1500
1700 REM ÜBERPRüFUNG OB GENUGEND FREIRAUM VORHANDEN
1705 CLOSE #1
1710 FREE=VAL(FMS$(1,3)):NEED=FILEMAX*(FILELEN+5)/125+1
1715 IF NEED<FREE THEN 1800
1720 PRINT "KEIN AUSREICHENDER PLATZ VORHANDEN!":PRINT
FREE;" FREI  ";NEED;"BENÖTIGT!";CHR$(253)
1725 GOSUB 9850
1730 OOTO 1500
1900 REM FILENAME VERKETTEN
1805 FILENAME$=DVC$
1810 FILENAME$(LEN(FILENAME$)+1)=FILES
1915 FILENAME$(LEN(FILENAME$)+1)="."
1820 FILENAME$(LEN(FILENAME$)+1)=EXT$
1825 RETURN
2000 REM KOPF-, ZEIGER- UND DATENSATZ-STRINGS INITIALISIEREN
2005 OPEN #1,8,0,FILENAME$
```

```
2010 NOTE #1,FILESEC,FILEBYT
2015 DIM FILEHED$(124),FILEPTR$(4+4*FILEMAX),
FILEREC$(FILELEN)
2100 REM DATEIKOPF
2105 FILEHED$=BALNK$
2110 HI=INT(FILESEC/256)
2115 LO=FILESEC-HI*256
2120 FILEHED$(1,1)=CHR$(HI)
2125 FILEHED$(2,2)=CHR$(LO)
2130 FILEHED$(3,3)=CHR$(FILEBYT)
2135 FILEHED$(4,4)=CHR$(0)
2140 HI=INT(FILEMAX/256)
2145 LO=FILEMAX-HI*256
2150 FILEHED$(5,5)=CHR$(HI)
2155 FILEHED$(6,6)=CHR$(LO)
2160 HI=INT(FILELEN/256)
2165 LO=FILELEN-HI*256
2170 FILEHED$(7,7)=CHR$(HI)
2175 F1LEHED$(8,8)=CHR$(LO)
2180 PRINT #1;FILEHED$
2200 REM DATEI-ZEIGER
2205 FOR I=l 4+4*FILEMAX STEP 128$:FILEPTR$(I)=BLANK$:
NEXT I
2210 NOTE #1,S,B
2215 HI=INT(S/256)
2220 LO=S-HI*256
2225 FILEPTR$(1,1)=CHR$(HI)
2230 FILEPTR$(2,2)=CHR$(LO)
2235 FILEPTR$(3,3)-CHR$(B)
2240 FILEPTR$(4,4)=CHR$(0)
2245 PRINT #1;FILEPTR$
2300 REM DATENSÄTZE
2305 FOR I=1 TO FILELEN STEP 128:FILEREC$(I)=BLANK$:NEXT I
2310 FOR I=1 TO FILEMAX
2315 NOTE #I,S,B
2320 HI=INT(S/256)
2325 LO=S-HI*256
2330 FILEPTR$(I*4+1,I*4+1)=CHR$(HI)
2335 FILEPTR$(I*4+2,I*4+2)=CHR$(L0)
2340 FILEPTR$(I*4+3,I*4+3)=CHR$(B)
2345 FILEPTR$(I*4+4,I*4+4)=CHRS(0)
2350 GOSUB 9700
2355 PRINT #1;FILEREC
2360 NEXT I
2365 CLOSE #l
2370 OPEN #1,12,0,FILENAME$
2375 GOSUB 9510
2380 RETURN
```

```
9500 REM "FILEPTR.SUB" WBB 19.3.81
9510 REM FILEPTR SCHREIBEN
9515 S=ASC(FILEPTR$(1,1))*256+ASC(FILEPTR2,2))
9520 B=ASC(FILEPTR$(3,3))
9525 POINT #1,S,B
9530 PRINT #1;FILEPTR$
9535 RETURN
9600 REM BILDSCHIRM
9605 GRAPHICS 2
9610 PRINT #6,"* INITIALISIERUNG *"
9615 PRINT #6
9620 PRINT #6;"IM AUSENBLIK"
9625 PRINT #6,"       TOTAL"


9630 PRINT #6;"    % BELEGT"
9635 PRINT #6
9640 PRINT #6;"       SEKTOR"
9645 PRINT #6;"         BYTE"
9650 RETURN
9700 REM BILDSCHIRM ERNEUERN
9705 PLOT 14,2:PRINT #6;1
9710 PLOT 14,3:PRINT #6;FILEMAX
9715 PLOT 14,4:PRINT #6;INT(I/FILEMAX*100)
9720 PLOT 14,6:PRINT #6;S;"   "
9725 PLOT 14,7:PRINT #6;B;"   "
9795 RETURN
9900 REM TRAP,SUB
9805 TRAP 9825
9810 PRINT "FEHLER  ";PEEK(195);" BEI ";PEEK(l87)*256+
PEEK(186)
9815 PRINT "BESTÄTIGUNG ";
9820 INPUT YN$
9825 END
9850 REM "DELAY.SUB" WBB 19.3.81
9851 VERZÖSERT AUSFÜHRUNG UM 2,5 SEKUNDEN
9852 REM (SCRATCH-P20)
9860 P20=PEEK(20)+150:IF P20>255 THEN P20=P20-256
9865 IF PEEK(20)<>P20 THEN 9865
9870 RETURN
```

BEISPIEL-PROGRAMM

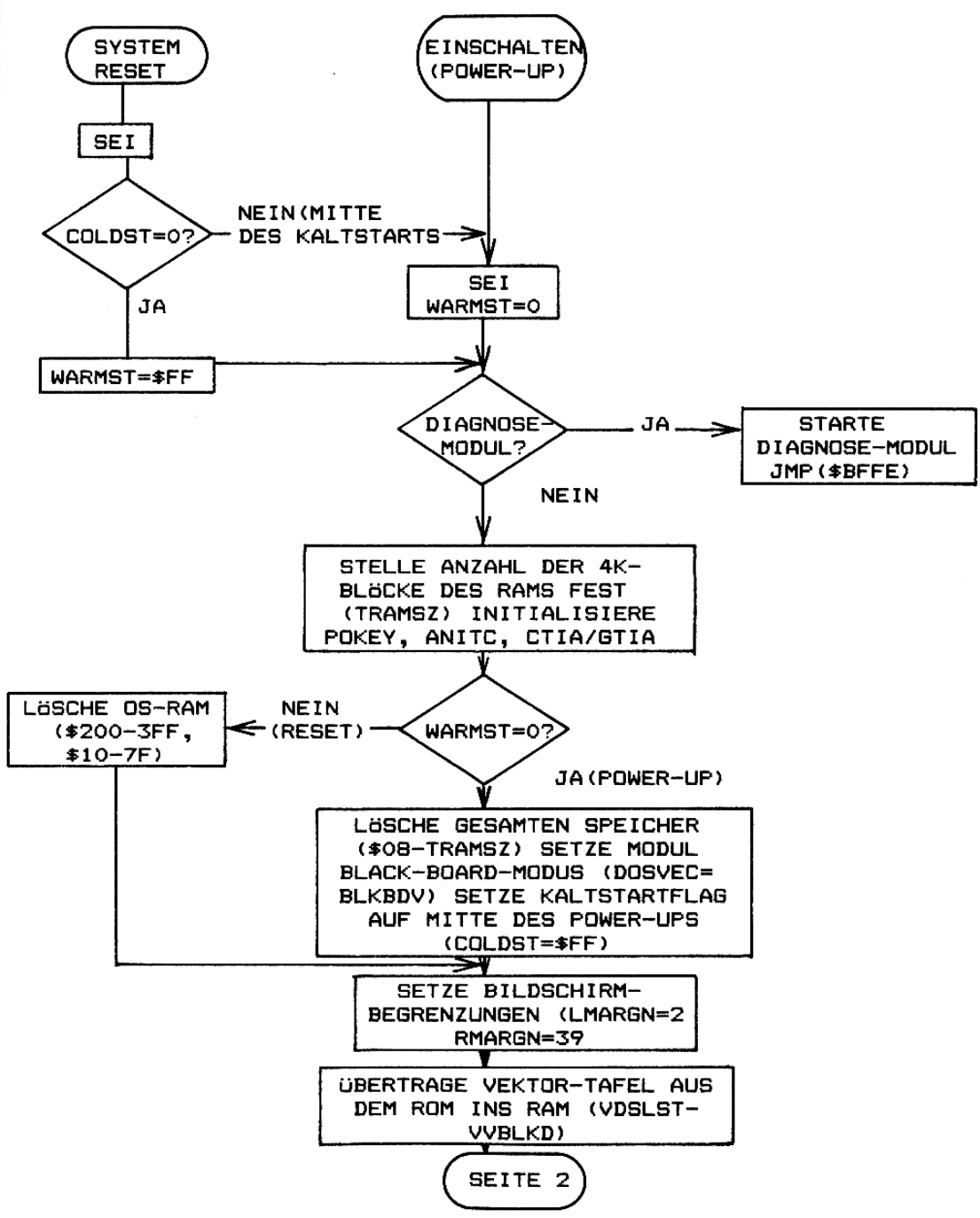
```
10 REM "FILEEX" WBB 31.3.8l
100 REM BEISPIEL DER ROUTINEN FÜR DEN FREIEN ZUGRIFF
101 REM DIESES PROGRAMM VERARBEITET DIE DATEI
102 REM D2:AREACODE.DAT.
103 REM SIE SOLLTE DURCH BENUTZUNG VON "FILE0001"
104 REM BEREITS INITIALISIERT WORDEN SEIN.
```

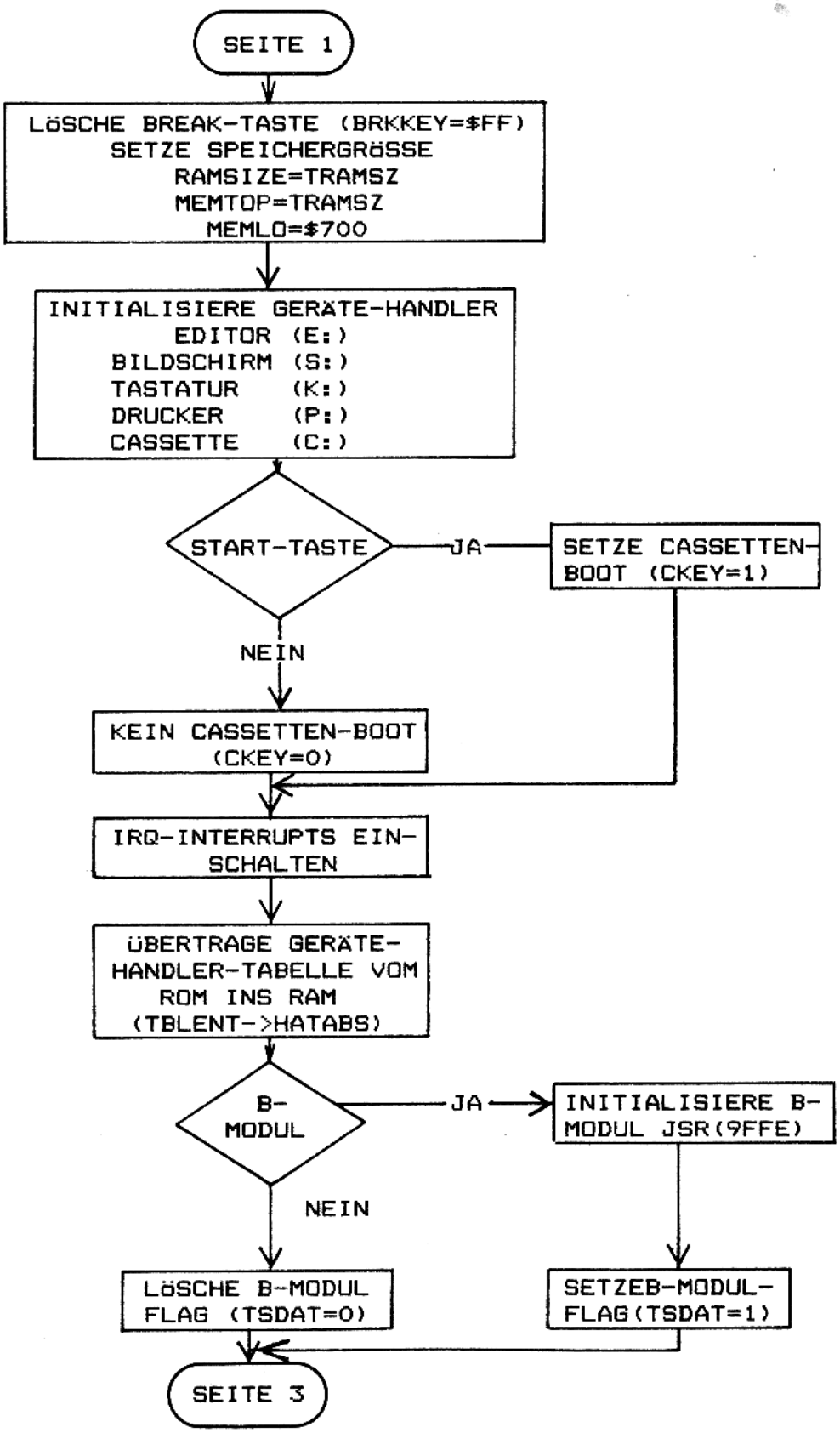
```
105 SIE BESTEHT AUS 24 BYTE-SÄTZEN: 1-3 BEREICH-CODE.
106 REM 4-24 POSITIONS-BESCHREIBUNG
107 REM
110 GRAPHIC 0
120 PRINT "´FILEEX´":PRINT :PRINT "INITIALISIERUNG"
200 REM INITALISIERN DER VARIABLEN
210 DIM FILE$(15),ACODE$(3),LOC$(21),YN$(1)
220 CHANNEL=l
230 FILE$="Dl:AREACODE.DAT"
300 REM DATEI ÖFFNEN
310 PRINT "ÖFFNEN DER DATEI"
320 GOSUB 9300
400 REM BEGINN DER BENUTZER-EINGABE
410 PRINT
420 PRINT "(0=ENDE) BEREICHS-CODE ";:INPUT ACODE$:IF
ACODE$="0" THEN 900
500 REM AKTIVE SÄTZE FÜR PASSENDEN BEREICHS-CODE SUCHEN
510 MATCH=0
520 FOR RECORD=1 TO FILEMAX
530 GOSUB 9600
540 IF STS=1 THEN GOSUB 5000
550 NEXT RECORD
560 IF MATCH=l THEN 400
600 REM NICHTS GEFUNDEN, GESTATTE ANFÜGUNG
510 PRINT "KEINE ÜBEREINSTIMMUNG IN DER DATEI GEFUNDEN!"
620 PRINT "WOLLEN SIE ANFÜGEN (J/N) ";:INPUT YN$
630 IF YN$<>"J" THEN 400
700 REM ANFÜGUNG ERWÜNSCHT
710 GOSUB 9400
720 IF RECORD=0 THEN PRINT "DATEI IST VOLLSTÄNDIG BELEGT,
SATZ WIRD NICHT ANGEFÜGT!":GOTO 400
730 PRINT "STELLE: ";:INPUT LOC
740 FILEREC$=ACODE$
750 FILEREC$(4)=LOC$
800 REM DATEI-ERNEUERUNG AUSFÜHREN
810 GOSUB 9600
820 PRINT #CHANNEL;FILEREC$
830 GOSUB 9500
840 GOTO 400
900 REM LÖSCHUNGEN GESTATTET
910 PRINT
920 PRINT "WOLLEN SIE LÖSCHEN (J/N) ";:INPUT YN$
930 IF YN$<>"J" THEN 1200
1000 REM FESTLEGEN, WELCHES ZU LöSCHEN IST
1010 PRINT
1020 PRINT "(0=ENDE) BEREICHS-CODE ";:INPUT ACODE$:IF
ACODE$"0" THEN 1200
1100 REM SUCHE AKTIVE SÄTZE FUR PASSENDEN BEREICHS-CODE
1110 FOR RECORD=1 TO FILEMAX
```

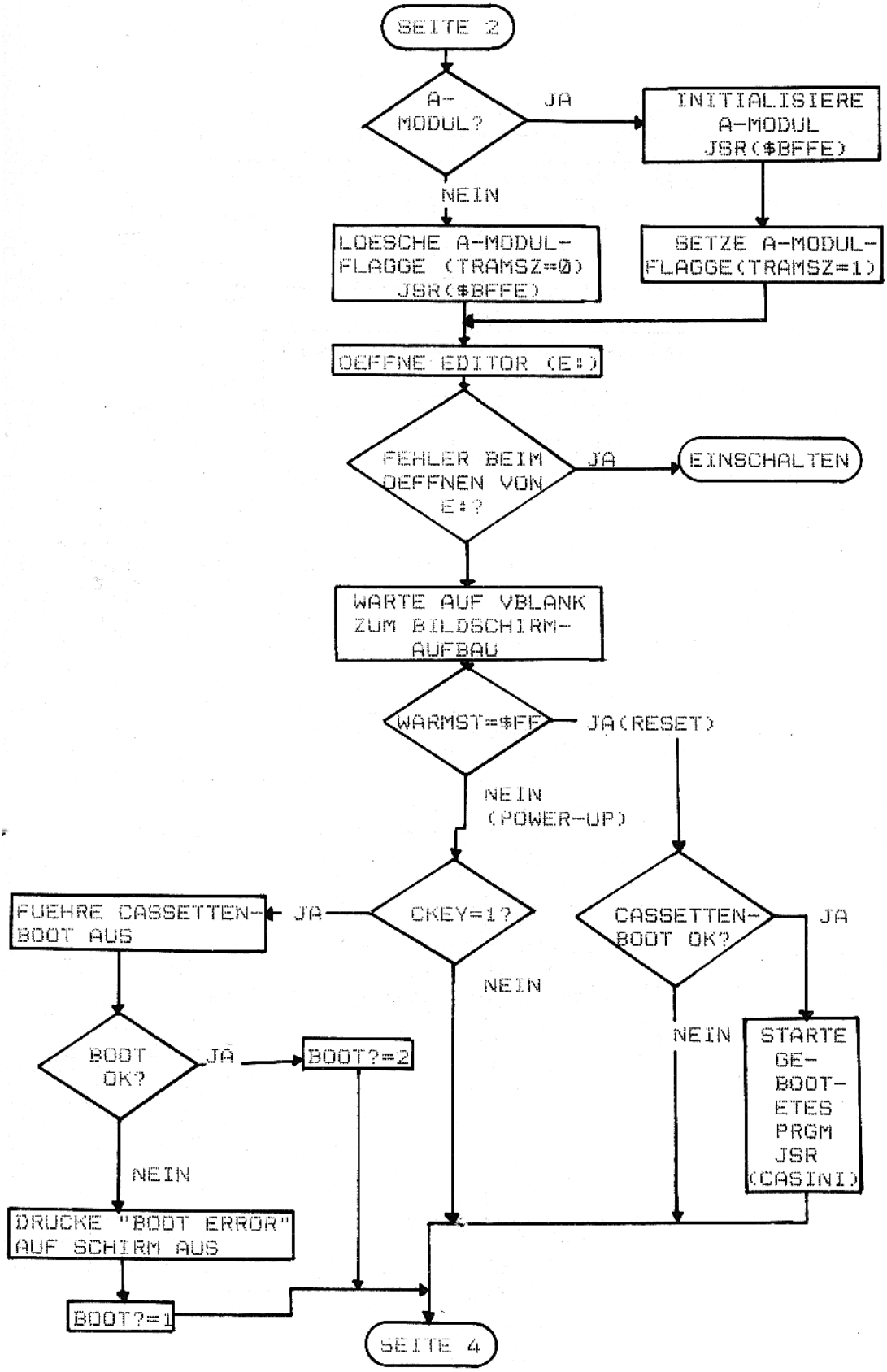
```
1120 GOSUB 9600
1120 IF STS=l THEN GOSUB 5100
1125 NEXT RECORD
1130 GOSUB 9500
1140 GOTO 1000
1200 REM GIB DATEI AUF BILDSCHIRM AUS
1210 PRINT :PRINT "CODE","STELLE":PRINT
1220 FOR RECORD=l TO FILEMAX
1230 GOSUB 9600
1240 IF STS=l THEN GOSUB 5200
1250 NEXT RECORD
1300 REM GESTATTE HARDCOPY
1310 PRINT :PRINT "WOLLEN SIE EINE GEDRUCKTE LISTE
(J/N)";:INPUT YN$
1320 IF YN$<>"J" THEN 4900
1330 LPRINT "CODE","STELLE":PRINT
1340 FOR RECORD=1 TO FILEMAX
1350 GOSUB 9600
1360 IF STS=l THEN GOSUB 5300
1370 NEXT RECORD
1380 GOTO 4900
4900 REM ENDE
4910 CLOSE #CHANNEL
4920 PRINT "ENDE DER AUSFÜHRUNG"
4930 REM
5000 REM VERARBEITE AKTIVEN SATZ/ANZEIGE
5010 INPUT #CHANNEL,FILEREC$
5020 IF FILEREC$(1,3)<>ACODE$ THEN RETURN
5030 MATCH=l
5040 PRINT "STELLE: ";FILEREC$(4)
5050 RETURN
5100 REM VERARBEITE AKTIVEN SATZ/LÖSCHEN
5110 INPUT #CHANNEL,FILEREC$
5120 IF FILEREC$(1,3)<>ACODE$ THEN RETURN
5130 GOSUB 9450
5150 PRINT "GELÖSCHT ";FILEREC$(4)
5160 RETURN
5200 REM VERARBEITE AKTIVEN SATZ/LÖSCHEN
5210 INPUT #CHANNEL,FILEREC$
5220 PRINT FILEREC$(1,3),FILEREC$(4)
5230 RETURN
5300 REM VERARBEITE AKTIVEN SATZ/LPRINT
5310 INPUT #CHANNEL,FILEREC$
5320 LPRINT FILEREC$(1,3),FILEREC$(4)
5330 RETURN

9300 REM "FILEOPEN.SUB" WBB 30.3.81
9305 REM ÖFFNE EINE DATEI IM MODUS 12 UND DEFINIERE ALLE
9307 REM VARIABLEN
9310 OPEN #CHANNEL,12,0,FILE$
```

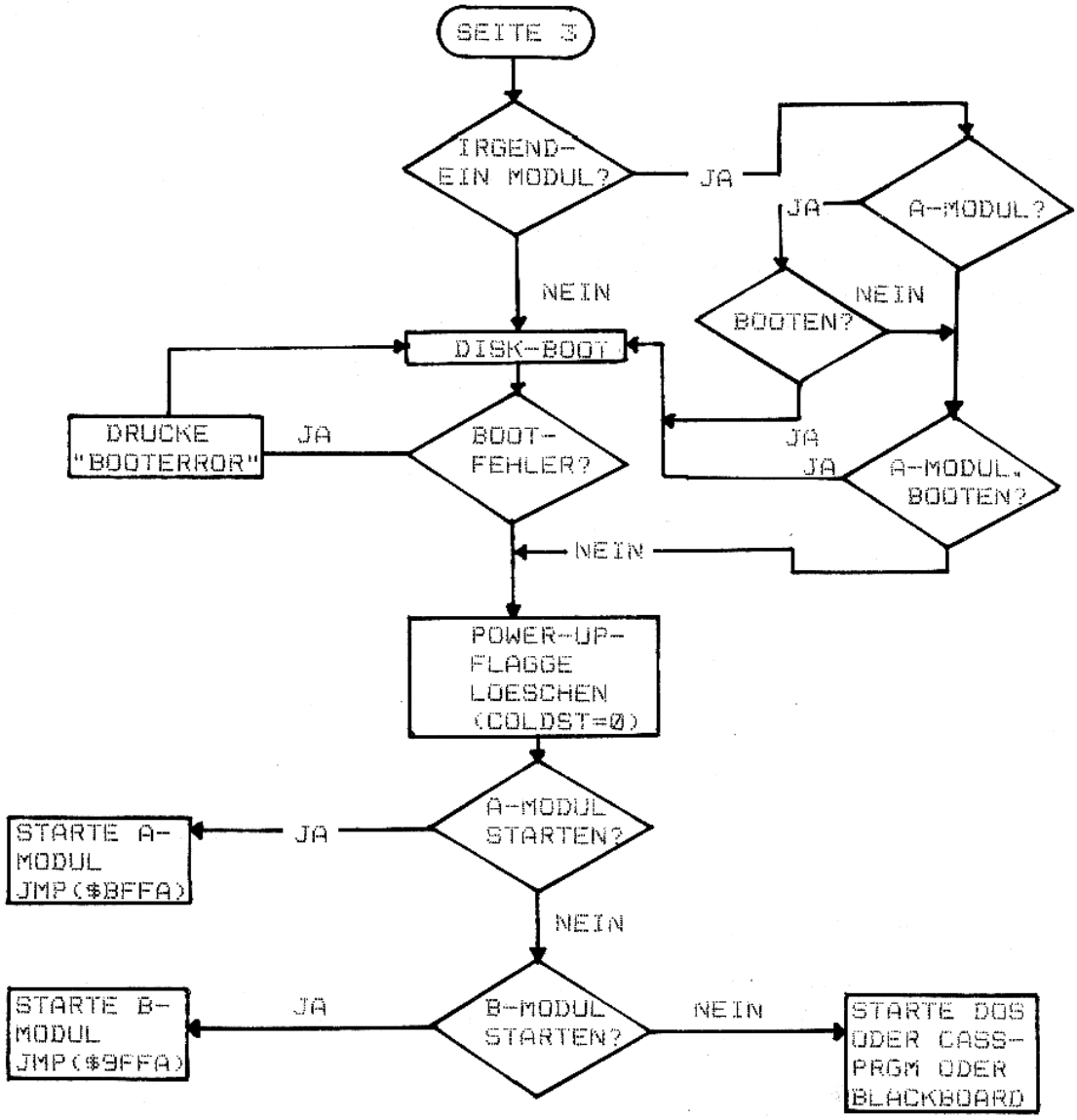
```
9315 DIM FILEHED$(124)
9320 INPUT #CHANNEL,FILEHED$
9325 FILESEC=ASC(FILEHED$(1)*256+ASC(FILEHED$(2))
9330 FILEBYT=ASC(FILEHED$(3))
9335 FILESTS=ASC(FILEHED$(4))
9340 FILEMAX=ASC(FILEHED$(5))*256+ASC(FILEHED$(6))
9345 FILELEN=ASC(FILEHED$(7))*256+ASC(FILEHED$(8))
9350 DIM FILEPTR$(4+4*FILEMAX),FILEREC$(FILELEN)
9355 INPUT #CHANNEL,FILEPTR$
9360 RETURN
9400 REM "FILEADD.SUB" WBB 30.3.81
9405 REM BESTIMME NÄCHSTEN VERFÜGBAREN SATZ
9410 RECORD=0
9415 IF FILEMAX=0 THEN RETURN
9420 FOR RECORD1=1 TO FILEMAX
9425 B=RECORD1*4+4
9430 IF FILEPTR$(B,B)=CHR$(0) THEN RECORD=RECORD1:RECORD1=
FILEMAX:FILEPTR$(B,B)=CHR$(1)
9435 NEXT RECORD1
9440 RETURN
9450 REM "FILEDEL.SUB" WBB 30.3.81
9455 REM LöSCHE EINEN AKTIVEN SATZ
9460 B=RECORD*4+4
9465 FILEPTR$(B,B)=CHR$(0)
9470 RETURN
9500 REM "FILELPTR.SUB" WBB 19.3.91
9510 REM SCHREIBE FILEPTR*
9515 S=ASC(FILEPTR$(1)*256+ASC(FILEPTR$(2))
9520 B=ASC(FILEPTR$(3))
9525 POINT #1,S,B,
9530 PRINT #l;FILEPTR$
9535 RETURN
9600 REM "FILEPOS.SUB" WBB 31.3.81
9605 REM DATEIZEIGER AUF SATZANFANG SETZEN
9610 S=ASC(FILEPTR$(RECORD*4+1))*256+ASC(FILEPTRS
(RECORD*4+2))
9615 B=ASC(FILEPTR$(RECORD*4+3))
9620 STS=ASC(FILEPTR$(RECQRD*4+4))
9625 POINT #CHANNEL,S,B
9630 RETURN
```

Anhang VII
Initialisierung




SEITE 1
LöSCHE BREAK-TASTE (BRKKEY=$FF)
SETZE SPEICHERGRöSSE
RAMSIZE=TRAMSZ
MEMTOP=TRAMSZ
MEMLO=$700
INITIALISIERE GERÄTE-HANDLER
EDITOR (E:)
BILDSCHIRM (S:)
TASTATUR (K:)
DRUCKER (P:)
CASSETTE (C:)
START-TASTE
JA
SETZE CASSETTEN-BOOT (CKEY=1)
NEIN
KEIN CASSETTEN-BOOT (CKEY=0)
IRQ-INTERRUPTS EIN-SCHALTEN
ÜBERTRAGE GERÄTE-HANDLER-TABELLE VOM ROM INS RAM (TBLENT->HATABS)
B-MODUL
JA
INITIALISIERE B-MODUL JSR(9FFE)
NEIN
LöSCHE B-MODUL FLAG (TSDAT=0)
SETZEB-MODUL-FLAG(TSDAT=1)
SEITE 3


SEITE 2
A-MODUL?
JA
INITIALISIERE A-MODUL JSR($BFFE)
NEIN
LOESCHE A-MODUL-FLAGGE (TRAMSZ=0) JSR($BFFE)
SETZE A-MODUL-FLAGGE(TRAMSZ=1)
OEFFNE EDITOR (E:)
FEHLER BEIM OEFFNEN VON E:?
JA
EINSCHALTEN
WARTE AUF VBLANK ZUM BILDSCHIRM-AUFBAU
WARMST=$FF
JA(RESET)
NEIN (POWER-UP)
CKEY=1?
JA
FUEHRE CASSETTEN-BOOT AUS
NEIN
CASSETTEN-BOOT OK?
JA
STARTE GE-BOOT-ETES PRGM JSR (CASINI)
NEIN
BOOT OK?
JA
BOOT?=2
NEIN
DRUCKE "BOOT ERROR" AUF SCHIRM AUS
BOOT?=1
SEITE 4


SEITE 3
IRGEND-
EIN MODUL?
JA
JA
A-MODUL?
NEIN
BOOTEN?
NEIN
NEIN
DISK-BOOT
DRUCKE
"BOOTERROR"
JA
BOOT-
FEHLER?
JA
JA
A-MODUL
BOOTEN?
NEIN
POWER-UP-
FLAGGE
LOESCHEN
(COLDST=0)
A-MODUL
STARTEN?
STARTE A-
MODUL
JMP($BFFA)
JA
NEIN
STARTE B-
MODUL
JMP($9FFA)
JA
B-MODUL
STARTEN?
NEIN
STARTE DOS
ODER CASS-
PRGM ODER
BLACKBOARD

Anhang VIII
GTIA

Der GTIA ist ein neuer Graphik-Chip, der den älteren CTIA ersetzt. Der GTIA unterscheidet sich nicht sonderlich vom CTIA. Er liefert zusätzlich neue Möglichkeiten. So gibt es drei weitere Modi für die Interpretation der vom ANTIC-Chip kommenden Informationen. Der ANTIC braucht hierbei keinen neuen Modus, um sich mit dem GTIA zu verständigen; es wird der hochauflösende Modus $0F (BASIC-Modus 8) verwendet. Ansonsten ist der GTIA-Prozessar absolut kompatibel zum CTIA. Es folgt nun eine kurze Beschreibung der GTIA-Möglichkeiten, so daß die Unterschiede zwischen GTIA und CTIA deutlich werden.

Der CTIA-Chip wird zum Anzeigen der Daten auf dem Fernsehschirm benutzt. Er erzeugt das Playfield, die Player/Missile-Graphiken und fragt Kollisionen einzelner Objekte auf dem Schirm ab. Der CTIA interpretiert die vom ANTIC-Prozessor kommenden Daten als 6 Text- und 8 Graphik-Modi. Bei einem statischen Display werden die von ANTIC gelieferten Informationen zum Anzeigen von Farben und Helligkeiten verwendet (wie in den 4 Farbregistern festgelegt). Durch den GTIA wird dieses Spektrum auf 9 Farbregister (9 Farben + Helligkeiten), 16 Farben in einer Helligkeit oder eine Farbe in 16 Helligkeiten erweitert.

Die 3 "neuen" Graphik-Modi, die durch den GTIA-Chip möglich werden, sind einfach drei neue Interpretationsarten des Hochauflösungs-Modus $0F. Alle drei Modi beeinflussen lediglich das Playfield; die Player/Missile-Objekte können weiterhin benutzt werden. Außerdem können alle Farben und Helligkeiten durch Display-List-Interrupts geändert werden.

Der GTIA benutzt jeweils 4 Bit der von ANTIC kommenden Daten, um ein Pixel festzulegen (daher die Bezeichnung Pixel-Daten). Jedes Pixel ist 2 Color-Clocks breit und eine Scan-Line hoch, ist also viermal so breit wie hoch. Das Display besitzt eine Auflösung von 80 Pixel in der Breite mal 192 Pixel in der Höhe. Jede Zeile benötigt 320 Speicherbits, d.h. 40 Bytes, was der im Modus $0F benötigten Speichergröße entspricht. Ein Programm, das mit den GTIA-Modi arbeiten soll, muß daher einen freien Speicherraum von mindestens 8k zur Verfügung haben.

Die GTIA-Modi werden durch das Prioritäts-Register PRIOR ausgewählt. PRIOR besitzt eine Schatten-Adresse bei $026F

und liegt im OS bei $D01B. Die Bits D6 und D7 sind die für die GTIA-Modi maßgeblichen Kontrollbits. Ist keines dieser Bits gesetzt, wird kein GTIA-Modus benutzt und der GTIA-Prozessor arbeitet wie der CTIA. Wenn Bit D7 auf 0 und Bit D6 auf 1 gesetzt sind, dann wird der GTIA-Modus 9 angesprochen, welcher eine Farbe in 16 verschiedenen Helligkeiten zuläßt. Sind Bit D7 auf 1 und Bit D6 auf 0 gesetzt, dann wird der GTIA-Modus 10 eingeschaltet. Dieser Modus erzeugt 9 Farben + zugehörigen Helligkeiten, indem die 4 Playfield- plus die 4 Player/Mismile plus 1 Hintergrund-Farbregister verwendet werden. Werden in diesem Modus die Player/Missile-Objekte eingesetzt, dann erhalten auch sie ihre Farbdaten aus den ihnen zugeordneten Registern. (im Original:...erhalten sie ihre Farbe auch hier ihre Farbdaten aus...)) Wenn sowohl Bit D7, als auch Bit D6 auf 1 gesetzt sind, dann wird GTIA-Modus 11 eingeschaltet. Dieser Modus liefert 16 Farben in einer Helligkeit. Wie bei Modus 9 können auch hier Play- und Missile-Objekte zum Erweitern der Farb- bzw. Helligkeitsauflösung benutzt werden.

PRIOR

D7 D6 D5 D4 D3 D2 D1 D0

D7	**D6**	**OPTION**
0	0	KEIN GTIA-Modus (Modi 0-8) (CTIA-Operation)
0	1	1 Farbe, 16 Helligkeiten (Modus 9)
1	0	9 Farben + Helligkeiten (Modus 10)
1	1	16 Farben, 1 Helligkeit (Modus 11)

Abbildung VIII.1:
Bit-Muster in PRIOR zum Anwählen der GTIA-Modi

Das Anwählen der GTIA-Modi ist genauso einfach, wie das der anderen CTIA-Betriebsarten: um die GTIA-Modi von BASIC aus aufzurufen, sind lediglich die Kommandos GRAPHICS 9, GRAPHICS 10 oder GRAPHICS 11 für die Modi 9, 10 und 11 erforderlich. In Assembler entspricht der Aufruf dieser Betriebsarten dem Bildschirmöffnen für die anderen Modi. Wird eine neue Display-List aufgebaut, so muß PRIOR zum

richtigen Einschalten der GTIA-Betriebsarten mitbenutzt werden (Siehe Abbildung VIII.1).

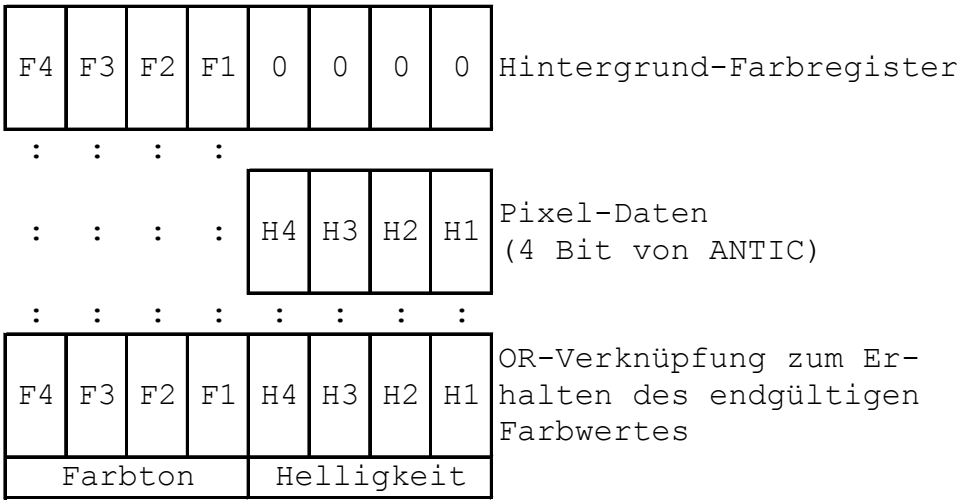
In Modus 9 werden bis zu 16 verschiedene Helligkeitsstufen einer Farbe erzeugt. Der ANTIC-Chip liefert die Pixel-Daten, welche die jeweilige Helligkeit auswählt. Der Farbton wird durch das Hintergrund-Register bestimmt (dieses geschieht in BASIC mittels POKE oder SETCOLOR). Das obere Nybble des Hintergrund-Registers gibt die Farbe an, während das untere Nybble auf Null gesetzt wird. Das Format dieses Kommandos sieht demnach wie folgt aus:

```
SETCOLOR 4,"Farbwert",0
```

Die 4 gibt hierbei an, daß die Farbe ins Hintergrund-Register gebracht werden soll. Der Farbton kann wie gewöhnlich ein Wert von 0 bis 15 sein, wobei die Null den Helligkeitswert des Farbregisters auf 0 setzt. Dieses ist erforderlich, weil die Pixel-Daten von ANTIC mit dem unteren Nybble des Hintergrund-Registers oderiert werden, um die gewünschte Helligkeit auf den Schirm zu bringen. Auch das COLOR-Kommando wird zum Bestimmen der zum Zeichnen auf den Bildschirm verwendeten Helligkeit benutzt. Der Parameter-Wert bei diesem Befehl kann sich zwischen 0 und 15 inklusive bewegen. Ein BASIC-Programm muß also mindestens die folgenden Anweisungen enthalten, damit der GTIA-Modus 9 verwendet werden kann:

```
GRAPHICS 9          GTIA 9 ansteuern
SETCOLOR 4,12,0     Hintergrund-Farbregister auf
                    Grün festlegen.
FOR I=0 TO 15       Beispiel für die Verwendung des
COLOR I             COLOR-Kommandos.
PLOT 4,I+10
NEXT I
```

In Assembler muß die OS-Schatten-Adresse des Hintergrund-Registers bei $2C8 benutzt werden, um den Farbton mit dem oberen Nybble festzulegen. Wird CIO aufgerufen, dann werden die Pixel-Daten im OS-Register ATACHR bei $2FB gespeichert. Hierdurch wird die Helligkeit mit den Werten $0 bis $F ausgewählt. Werden vom Programmierer eigene Anzeige-Daten geliefert, dann gehen diese Pixel-Daten direkt in die linke bzw. rechte Hälfte des Bytes des Anzeige-RAMs.



FARBTON:
Ein konstanter Farbton, der durch das HintergrundRegister bestimmt wird.

HELLIGKEIT:
16 Helligkeitsstufen, die durch die Pixel-Daten ausgewählt werden.

Abbildung VIII.2
OR-Verknüpfung von Hintergrund-Farbregister und Pixel-Daten zum Erhalten des endgültigen Farbwertes.

Modus 11 entspricht dem oben gezeigten Modus 9, außer daß bei diesem nicht 16 Helligkeiten einer Farbe, sondern 16 Farben in einer Helligkeit angezeigt werden können. ANTIC liefert dann die Pixel-Daten zur Auswahl der 16 Farbabstufungen. In BASIC wird das SETCOLOR-Kommando verwendet, um den Helligkeitswert aller Farben im unteren und den Farbwert im oberen Nybble auf Null zu setzen. Das Format für dieses Kommando lautet also:

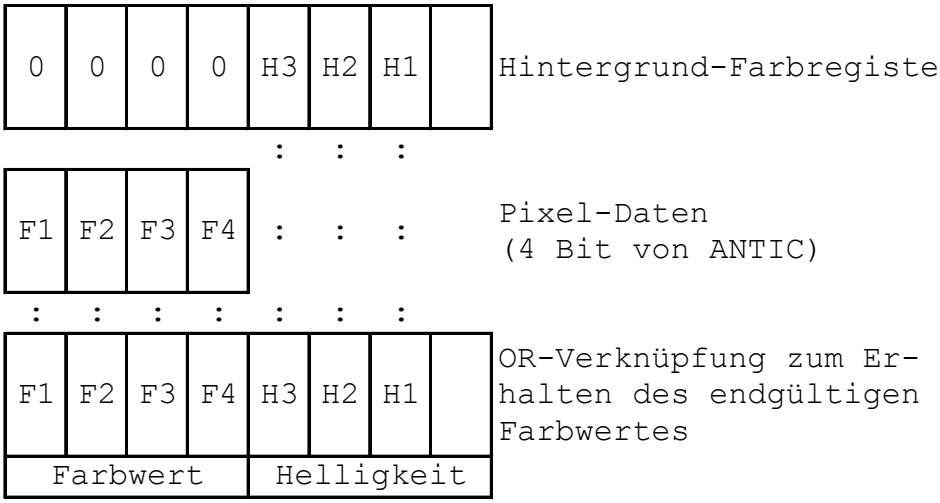
```
SETCOLOR 4,0,"Helligkeit"
```

Die 4 wählt das Hintergrund-Register an. Die Null setzt das obere Nybble auf Null und der Helligkeitswert kann eine Zahl von 0 bis 15 sein. Wie bei anderen Graphik-Modi (ausgenommen Modus 9), ist das erste Bit der Helligkeit maßgebend. Daher sind nur gerade Zahlen und somit 8 Helligkeiten in diesem Modus möglich. Das COLOR-Kommando wird in diesem Modus benutzt, um einen der verschiedenen Farbwerte (0 bis 15) auszuwählen. Die Pixel-Daten von ANTIC werden mit dem oberen Nybble des Hinergrund-Registers oderiert, um den endgültigen

Farbwert festzulegen. Ein BASIC-Programm muß daher mindestens die folgenden Schritte ausführen, um im Modus 11 arbeiten zu können:

```
GRAPHICS 11          Modus 11 anwählen
SETCOLOR 4,0,12      Hintergrund auf eine Helligkeit
                     setzen, in diesem Fall sehr hell.
FOR I=0 TO 15        Beispiel für die Benutzung
COLOR I              des COLOR-Be+ehls
PLOT 4,I+10
NEXT I
```

In Assembler muß die OS-Schatten-Adresse bei $2C8 des Hintergrund-Farbregisters benutzt werden, um die Helligkeit in den unteren vier Bit auf einen Wert zwischen $0 und $F zu setzen. Wird die CIO angesprungen, dann müßen die Pixel-Daten in ATACHR bei $2FB gespeichert werden. Hierdurch wird ein Farbwert von $0 bis $F ausgewählt. Liefert der Programmierer eigene Anzeige-Daten, so gehen die Pixel-Daten direkt in die linke oder rechte Hälfte des Bytes des Anzeige-Rams.



FARBTON:
16 Farbwerte, die durch die Pixeldaten bestimmt werden.

HELLIGKEIT:
Eine konstante Helligkeit, die durch das Hintergrundregister festgelegt wird.

Abbildung VIII.3
OR-Verknüpfung von Hintergrund-Farbregister und Pixel-Daten zum Erhalten des endgültigen Farbwertes

Modus 10 gestattet die gleichzeitige Benutzung aller 9Farbregister auf dem Playfield. In jedes Farbregister wird hierbei eine Kombination aus Farb- und Helligkeitswert gespeichert. Die in diesem Modus von ANTIC kommenden Pixel-Daten werden zur Auswahl eines dieser 9 Register verwendet. In BASIC kann das SETCOLOR-Kommando, wie im BASIC-Referenz Manual beschrieben, zum Setzen der Hintergrund- und Playfield-Farbregister benutzt werden. Die Farbwerte dieser Register können aber auch mit Hilfe des POKE-Befehls für die Adressen 708-712 (Speicherstellen der Register) verwendet werden. Zum Setzen der Player/Missile-Farbregister muß das POKE-Kommando verwendet werden, dessen Speicherstellen zwischen 704 und 707 liegen.

Der COLOR-Befehl wird zum Auswählen des gewünschten Farbregisters verwendet, wobei die bedeutungsträchtigen Werte zwischen 0 und 8 inklusive liegen. Da von ANTIC in diesem Modus 4 Daten-Bits pro Pixel gesendet werden, wäre theoretisch eine Auswahl von 16 Farbregistern möglich. Ein nicht zulässiger Daten-Wert von 9 bis 15 resultiert daher in der Anzeige eines Farbregisters mit einem anderen Stellenwert. Ein BASIC-Programm, das den GTIA-Modus 10 benutzt, muß die folgenden Anweisungen enthalten:

1)
ein GRAPHICS 10-Kommando zum Anwählen des Modus;

2)
eine Reihe von POKE- oder SETCOLOR-Befehlen, um die gewünschten Werte in die Farbregister zu bringen und

3)
ein COLOR-Kommando zur Auswahl des gewünschten Farbregister.

In Assembler werden die Pixel-Daten, wie in Modus 9 und 11, in ATACHR ($2FB) oder direkt im Bildschirm-RAM gespeichert. In diesem Modus besitzen die Pixel-Daten einen Wert von 0 bis 8 und wählen somit eines der 9 Farbregister aus.

WERT DES COLOR-BEFEHLS	BENUTZTES FARB-REGISTER	OS-SCHATTEN-ADRESSE
0	D012	2C0
1	D013	2C1
2	D014	2C2
3	D015	2C3
4	D016	2G4
5	D017	2C5
6	D018	2C6
7	D019	2C7
8	D01A	2C8

Abbildung VIII.4:
Farbregister-Nummern, -Speicherstellen
und COLOR-Befehl-Übersicht

Eine wichtige Frage betrifft die Kompabilität des GTIAs zum CTIA. Der GTIA-Chip ist aufwärts voll kompatibel zum CTIA, so daß weiterhin Player/Missile-Graphiken, Kollisions-Register oder Display-List-Interrupts benutzt werden können. Sämtliche GTIA-Modi können durch das OS bearbeitet werden, daher laufen alle Utilities und Graphik-Kommandos, die mit dem CTIA funktionieren, auch auf dem GTIA-Chip.

Durch den GTIA wird das Anzeigen von noch mehr Farben gleichzeitig möglich. In einer Bildschirmzeile können 16 Farbänderungen erfolgen, ohne daß der 6502-Prozessor beeinflußt wird. Durch Display List Interrupts wäre ein absolutes Maximum von 128 Farbänderungen pro Zeile machbar. Durch den GTIA-Modus können Konturen und räumliche Tiefe besser dargestellt werden, so daß realistische 3-D Graphiken möglich werden.

Es gibt allerdings auch einige Nachteile dieser Graphik-Modi. Diese Betriebsarten sind Map-Modi, d.h. es kann kein Text auf den Bildschirm gebracht werden, wenn sie aktiviert sind. Um Zeichen darzustellen, muß eine neue Display-List aufgebaut werden. Zum Zweiten ist ein Pixel dieser Modi ein langes horizontales Rechteck (ein Verhältnis von 4 zu 1 in Länge zu Höhe), daher können Kurven nicht gut gezeichnet werden. Weiterhin benötigt jedes Pixel 4 Informations-Bits, so daß ein GTIA-Modus fast 8K-RAM zum Arbeiten verbraucht. Schließlich ist der GTIA nicht abwärts kompatibel, d.h. Programme, die auf dem GTIA-Computer laufen, nicht unbedingt auf älteren CTIA-Modellen arbeiten. Graphiken sind zwar erkennbar, aber natürlich nicht so farbig. Ein Programm besitzt keine Möglichkeit

festzustellen, ob es auf einem CTIA- oder GTIA-Computer läuft.

==========

ERGÄNZUNGEN ZUM DE RE ATARI

INFORMATIONEN ZU DEN XL-GERÄTEN

==========

Inhalt

Einführung

Dieser Text ist eine Ergänzung zum DE RE ATARI und behandelt die speziellen Eigenschaften des XL-Systems.

Wie in Abschnitt 3.5 dargestellt, ist die XL-Linie eine Weiterentwicklung der alten 400/800-Linie. Das Betriebssystem der neuen Geräte wurde so geschrieben, daß ein Höchstmaß an Kompatibilität zur Alten gewährleistet wurde.

Da die grundlegende (für die Kommunikation mit dem Benutzer zuständige) Hardware (bis auf die hier beschriebenen Ausnahmen) zum allergrößten Teil kompatibel ist, bleibt das Betriebssystem in etwa das gleiche. Die im OS-Manual für den alten 400/800er angegebenen Daten behalten also auch für die neuen Geräte ihre Gültigkeit.

Dieser Anhang soll den Benutzer über die zusätzlichen Fähigkeiten und Adressen des neuen XL-Computers informieren, wobei auch Details der Peripherie-Charkteristiken behandelt werden.

Für das XL-System gültige Literatur

1.
Das OS-MANUAL für die ATARI"TM"-Home-Computer: Das Buch behandelt das Operating-System des alten 400/800-Computers und ist Grundlage für die in diesem Text beschriebenen Änderungen.

2.
Das Hardware-Manual für die ATARI"TM"-Computer: Das Hardware-Manual enthält Angaben zu den Hardware-Registern, welche die unterschiedlichen Funktionen des ATARI Computers kontrollieren. Soweit sie Teil des Operating-Systems sind, werden in diesem Manual die für die zusätzlichen Funktionen verwendeten Hardware-Register behandelt. Der Benutzer kann also in dem Hardware-Manual nachschlagen, um weitere Informationen zur Hardware zu erhalten.

3.
Das DE RE ATARI: Dieses Textwerk zeigt Möglichkeiten zur effektiven Programmierung des ATARI"TM"-Personal Computer Systems auf.

Vergleich zwischen dem alten und dem neuen System

Die nachstehende Liste gibt die in diesem Kapitel behandelten Fähigkeiten der XL-Geräte an, wobei jeder Punkt einen eigenen Abschnitt besitzt.

In diesem Kapitel werden folgende Punkte besprochen:

1.
Die HELP-Taste

2.
Wie und welche Tastencodes selbst definiert werden können.

3.
Wie die Rate der Tastenwiederholungs-Funktion geändert werden kann.

4.
Die Taste für die Umschaltung zwischen Groß- und Kleinschrift.

5.
Welche Dinge beim Selbsttest ausgeführt werden.

6.
Welche neuen Bildschirmmodi die XL-Geräte benutzen können.

7.
Wie das Feinscrolling des Text-Bildschirmes eingeschaltet werden kann.

8.
Wie der Disketten-Handler zum Erzielen einer besseren Kommunikation geändert wurde.

9.
Ein-/Ausschalten des Tastatur-Klickgeräusches.

10.
Wechsel zwischen internem und internationalem Zeichensatz.

Die in der oberen Liste angegebenen Zahlen entsprechen den jeweiligen Abschnitten. So wird z.B. Punkt 4 in Abschnitt 3.4 behandelt.

3.1 Die HELP-Taste

Das Betriebssystem erkennt bei der Tastatur-Abfrage, ob die HELP (=Hilfe) -Taste gedrückt wurde. Dies geschieht über eine Systemvariable, welche bei Betätigung der Taste gesetzt wird.

Für die HELP-Taste wird kein ATASCII-Code erzeugt, der von einem Programm abgefragt werden könnte. Es wird ausschließlich die Systemvariable beeinflusst. Ein Programm, das diesen Tastendruck erkennen soll, muß also zusätzlich zur normalen Tastatur-Bearbeitung prüfen, ob sich der Inhalt der Systemvarivale HELPFG ("Hilfsflag") geändert hat und damit vom Benutzer Hilfe angefordert wurde.

Die HELPFG-Variable liegt an der Speicherstelle $02DC. Nachfolgend werden die verschiedenen möglichen Inhalte dieser Adresse und deren Bedeutung angegeben.

Hexadezimaler Wert	Bedeutung
00	Das Flag für die HELP-Taste ist gelöscht. Es wird beim Einschalten auf 0 gesetzt und muß, nachdem es entsprechend bearbeitet wurde, vom Programm auf Null gesetzt werden.
11	Nur die HELP-Taste wurde betätigt.
51	Die Tastenkombination SHIFT-HELP wurde niedergedrückt.
91	Dieser Wert wird durch die Tastenkombination CTRL-HELP erzeugt.

3.2 Neudefinierung der Tastatur

Im XL-Computer gibt es eine Tabelle, in der die einzelnen Tasten definiert werden. Das Betriebssystem besitzt eine eigene Datentafel, welche die normale Belegung angibt. Der Benutzer kann nun selbst eine solche Definitionstafel erstellen. Er muß dann nur dem Betriebssystem mitteilen, wo sich letztere im Speicher befindet.

Eine Anwendungsmöglichkeit für diese Definitionstabelle wäre das Ausprobieren anderer, vielleicht besserer Tastenanordnungen, wie z.B. die DVORAK-Tastenbelegung. Ein Beispiel hierfür befindet sich in Anhang A, so daß der Benutzer mit dieser Möglichkeit experimentieren kann. (Im

Laufe der Jahre wurde die QWERTY-Anordnung der Tastatur als Standard angenommen, obwohl es Leute gibt, die der DVORAK-Belegung den Vorzug geben. Der Leser kann nun eine eigene Entscheidung fällen.

INHALT DER TASTATUR-DEFINITIONS-TABELLE

Diese Tabelle gestattet die Erzeugung jedes beliebigen ATASCII-Codes oder einer internen Funktion für nahezu jede Taste. Die Ausnahmen werden am Ende dieses Abschnittes angegeben. Um die Tasten neu zu definieren, muß als erstes ein Speicherbereich gefunden und festgelegt werden, in dem eine 192 Bytes große Tabelle abgelegt werden kann. In dieser Tabelle werden die neuen Tastendefinitionen vom Benutzer gespeichert. Danach wird dem Betriebssystem mitgeteilt, wo sich diese Tafel befindet, so daß sie anstelle der normalen verwendet wird.

Die Tafel besteht aus folgenden Teilen:

Kleinbuchstaben 64 Bytes	Beginnt bei einer, durch den Benutzer festgelegten Adresse 0. Tabellenteil der Kleinbuchstaben
SHIFT + Taste 64 Bytes	Tabellenteil für Großbuchstaben
CTRL + Taste 64 Bytes	Tabellenteil für CTRL-Tasten-kombinationen

Das letzte Byte dieser Tabelle liegt an der Adresse KEYTABLE-START + 191.

Der Grund für die Tabellenaufteilung in 3 Gruppen zu je 64 Bytes liegt in der Hardware, die insgesamt nur 64 Tastencodes erzeugen kann. Diese von 00-63 (=$00-$3F) nummerierten Codes werden als direkter Index innerhalb der Tabellenteile benutzt. Welcher dieser Tafelabschnitte benutzt wird, hängt von Betätigung der SHIFT- oder CTRL-Taste ab.

Anmerkung: Es gibt keinen Tafelteil für gleichzeitiges Drücken der SHIFT- und der CTRL-Taste. Diese Kombination ist unzulässig und wird dementsprechend vom Betriebssystem ignoriert.

Jeder der drei 64 Bytes großen Tabellenteile besitzt die nachfolgend angegebene Form:

Byte 0 enthält die Umwandlung für den Tastencode 00 mit CTRL-, SHIFT-Taste oder ohne diese beiden. Letzteres hängt vom angewählten Tafelabschnitt ab.

Code 00	Byte 0	enthält die Umwandlung für den Tastencode 00 mit CTRL-, SHIFT-Taste oder ohne diese beiden. Letzteres hängt vom angewählten Tafelabschnitt ab.
Code 01	Byte 1	enthält die Umwandlung für den Tastencode 01
:	:	:
Code 3F	Byte 63	enthält die Umwandlung für den Tastencode 3F

Die vom Benutzer in der Tabelle platzierten Codes erzeugen entweder einen ATASCII-Code (für spätere Umsetzung in ein Zeichen oder weisen das System an, eine bestimmte Funktion auszuführen. Speziell die Codes von $80 bis $92 werden besonders vom System behandelt. Diese Umsetzung wird in nachfolgender Darstellung angegeben:

CODES UND IHRE WIRKUNG AUF DAS SYSTEM NACH DER UMSETZUNG

CODE	**EFFEKT (sofern vorhanden)**
$00..$7F	Werden nur als ATASCII-Code benutzt.
$92..$FF	Werden nur als ATASCII-Code benutzt.
$80	Wird ignoriert; ungültige Tastenkombination.
$81	Invertiert die Ausgabe auf dem Bildschirm.
$82	Umschaltung zwischen Groß-/Kleinschrift-Rastung.
$83	Großbuchstaben-Festrastung.
$84	CTRL-Festrastung.
$85	End-of-File (=Dateiende).
$86	ATASCII-Code;
$87	ATASCII-Code.
$89	Ein-/Ausschalten des Tastatur-Klickgeräusches
$8A	Cursor eine Zeile hoch
$8B	Cursor eine Zeile runter
$8C	Cursor eine Spalte nach links

$8D Cursor eine Spalte nach rechts
$8E Cursor in obere linke Bildschirmecke
$8F Cursor an untere linke Bildschirmecke
$90 Cursor an linken Rand
$91 Cursor an rechten Rand

Die nachstehende Tabelle zeigt den zu einzelnen Codes gehörigen Großbuchstaben. Die physikalische Position der einzelnen Tasten innerhalb der Tafel legt den von ihnen erzeugten Code fest. Um den entsprechenden Code zu erhalten, müssen vom Leser die jeweiligen Werte der Spalten und Zeilen addiert werden. Das Ergebnis ist der zum Betriebssystem gelangende Hex-Wert ($00=$3F).

TABELLE DER TASTENDEFINITIONEN

	0	1	2	3	4	5	6	7
00	L	J	;			K	+	*
08	O		P	U	RET	I	-	=
10	V		C			B	X	Z
18	4		3	6	ESC	5	2	1
20	,	SPACE	.	N		M	/	
28	R		E	Y	TAB	T	W	Q
30	9		0	7	BACKS	8	(	)
38	F	H	D		CAPS	G	S	A

Ein Beispiel: Der Großbuchstabe "C" befindet sich in der Tabelle in Zeile 10 und Spalte 2. Die Hardware erzeugt also einen Hardware-Code mit dem Wert $10 + $2 = $12. Der ATASCII-Code dieses Zeichens wird in der Definitionstafel für die Tasten an Stelle $12 gespeichert, wobei dieses für alle möglichen Eingabearten des "C" gilt (C-Taste allein, mit SHIFT oder mit CTRL gedrückt). Der Benutzer kann nun das Durchführen einer Funktion bzw. die Ausgabe eines bestimmten ATASCII-Codes veranlassen, indem der Inhalt der Definitionstafel an Stelle $12 geändert wird.

Nachdem die neue Definitionstafel erstellt und im Speicher untergebracht wurde, muß der Benutzer dem Betriebssystem mitteilen, wo diese sich befindet. Dies geschieht durch Speichern der Adresse des Tabellenbeginns an den Stellen $79 und $7A. Das OS benutzt dann die neue Tafel zur Umwandlung der Tasteneingaben.

Das niederwertige Byte der Definitionstabellen-Adresse muß in die Speicherstelle $79, das höherwertige Byte in Stelle

$7A geschrieben werden. Diese beiden Speicherstellen sind zusammen einer der verschiedenen Systemvektoren des Computers. Er wird beim Einschalten des Gerätes und beim SYSTEM-RESET so zurückgesetzt, daß er auf die Tabelle mit der Original-Definition zeigt.

NICHT NEUDEFINIERBARE TASTEN UND TASTENKOMBINATIONEN

Die folgenden Tasten bzw. Tastenkombinationen sind auf eine spezielle Weise verdrahtet oder werden auf eine besondere Art vom Betriebssystem behandelt und können daher nicht neu definiert werden.

Obwohl ein hardwaremäßig erzeugter Code und ein entsprechender Freiraum in der Umsetzungstafel existiert, ist es nicht möglich, diese Tasten neu zu definieren. Das Operating-System "fängt" den Hardware-Code dieser Eingabe direkt ab, damit eine spezielle Funktion immer ausgeführt werden kann und nicht zum Umsetzungsmodus gesprungen werden muß. Die folgenden Tasten sind von der Neudefinition ausgenommen:

BREAK
Diese Funktion wird gesondert durch das Betriebssystem bearbeitet. Sie wird durch die Hardware erkannt.

SHIFT
Diese Taste ist ein Bestandteil der hardwaremäßigen Codierung aller Tastenbetätigungen.

CTRL
Für diese Taste gilt das Gleiche, wie für die SHIFT-Taste.

OPTION, SELECT, START
Diese drei Tasten sind direkt mit den GTIA-Schaltkreisen verbunden und werden deshalb auch direkt von diesen erkannt.

RESET
Ist direkt mit der RESET-Leitung des 6502-Prozessors verbunden.

HELP
Diese Funktion ist im Betriebssystem festgelegt. Die Bearbeitung der HELP-Funktion wird an anderer Stelle diesem Manual behandelt.

CTRL-1
Diese Taste stoppt und startet die Bildschirmausgabe. Sie kontrolliert die Stop/Start-Funktion beim Listen ("LIST" in BASIC) und wird vom Betriebssystem beim Decodieren der hardwaremäßigen Tasten-Codes "abgefangen".

3.3 DURCH DEN BENUTZER VERÄNDERLICHE RATE DER TASTENWIEDERHOLUNGSFUNKTION

Das Betriebsystem der XL-Serie gestattet dem Benutzer, die Rate festzulegen, mit der eine Taste ihre Eingabe ins System wiederholt, wenn sie längere Zeit niedergedrückt wird. Diese Festlegung kann durch ein Programm geschehen, das die OS-System-Variable KEYREP verändert. Diese liegt bei Speicherstelle $02DA.

Diese Variable legt die Wiederholungsrate fest, indem die Anzahl der aufgetretenen VBLANK (Vertital Blank)-Intervalle gezählt wird. Für das PAL-System liegt er bei 5. Diese Worte haben zur Folge, daß eine Taste bei beiden Systemen 10 mal pro Sekunde wiederholt wird.

Durch Veränderung dieser Variablen wäre eine maximale Wiederholungsrate von 50 Zeichen pro Sekunde beim PAL-System möglich, was der Rate des Bildschirm-Refresh entspricht. Dieser Zustand wird durch einen Wert von 1 in der Variablen erreicht.

Der Benutzer kann außerdem die Zeit festlegen, die verstreichen muß, bevor die Tastenwiederholung einsetzt. Die OS-Systemvariable, die dieses steuert, liegt bei Speicherstelle $02D9 und wird mit KRPDEL bezeichnet.

Diese Variable kontrolliert die Anzahl der auftretenden VBLANKs, bevor die erste Wiederholung auftritt. Danach wird die Wiederholungsrate wie oben beschrieben gesteuert. Der Ausgangswert dieser OS-Variable liegt bei 40 für PAL-Systeme, so daß 0,8 Sekunden verstreichen, bevor eine Tasteneingabe vom System wiederholt wird.

3.4 UMSCHALTUNG ZWISCHEN GROß- UND KLEINSCHRIFT

Die Funktion der CAPS/LOWR-Taste sieht auf den Geräten der XL-Serie wie folgt aus.

TASTENKOMBINATION	AUGENBLICKLICHER ZUSTAND	NEUER ZUSTAND
CAPS	CTRL-Verriegelung	Kleinbuchstaben
CAPS	Großbuchstaben	Kleinbuchstaben
CAPS	Kleinbuchstaben	Großbuchstaben
SHIFT-CAPS	- beliebig -	Großbuchstaben
CTRL-CAPS	- beliebig -	CTRL-Verriegelung
SHIFT-CTRL-CAPS	- beliebig -	keine Änderung

Die Bedeutung der Zustände lautet folgendermaßen:

Kleinbuchstaben: Alle Tasten arbeiten im Kleinschrift-Modus.

Großbuchstaben: Alle alphabetischen Tasten (A-Z) arbeiten im Großbuchstaben-, alle anderen im KleinschriftModus.

CTRL-Rastung: Alle Tasten arbeiten, als wenn die CTRL- zusammen mit der Taste gedrückt würde.

3.5 SELBSTTEST BEIM EINSCHALTEN

Während des Einschaltens führt das Betriebssystem der XL-Serie folgende Überprüfungen aus, die RAM und ROM dem Computers betreffen:

a)
ist es möglich, einen Wert von $FF in alle RAM-Speicherstellen zu schreiben?

b)
ist es möglich, einen Wert von $00 in alle RAM-Speicherstellen zu schreiben?

c)
entspricht die Prüfsumme der beiden ROMs denen der in den jeweiligen ROMs gespeicherten?

Sobald einer dieser Tests negativ ausfällt, überträgt das Betriebssystem die Kontrolle an die Selbsttest-Routine für den Speichertest. Diese führt dann einen intensiveren Test sowohl des RAMs als auch des ROMs durch.

3.6 ZUSÄTZLICHE HARDWAREMÄßIGE BILDSCHIRMMODI

Die Geräte der XL-Serie gestatten im Gegensatz zu den alten 400/800ern den direkten Zugriff auf die speziellen Arbeitsmodi des Anzeige-Prozessors. Die nachfolgende Tabelle

gibt die Modi an, die bei der alten Serie direkt zugänglich waren. Die zweite Tafel zeigt die neuerdings direkt zugreifbaren Bildschirmmodi bei den XL-Computern, sowie die Nummern, über die sie softwaremäßig angesprochen werden können.

Die direkt zugänglichen Bildschirmmodi des ATARI 400/800"TM" Computers.

Software-Modus	**ANTIC-Modus**	**GTIA-Modus**
0 ($00)	2 ($02)	0
1 ($01)	6 ($06)	0
2 ($02)	7 ($07)	0
3 ($03)	8 ($08)	0
4 ($04)	9 ($09)	0
5 ($05)	10 ($0A)	0
6 ($06)	11 ($0B)	0
7 ($07)	13 ($0D)	0
8 ($08)	15 ($0F)	0
9 ($09)	15 ($0F)	1
10 ($0A)	15 ($0F)	2
11 ($0B)	15 ($0F)	3

Zusätzlich direkt ansteuerbare Bildschirmmodi der XL-Serie:

Software-Modus	**ANTIC-Modus**	**GTIA-Modus**
12 ($0C)	4 ($04)	0 (Anm. 1)
13 ($0D)	5 ($05)	0 (Anm. 1)
14 ($0E)	12 ($0C)	0
15 ($0F)	14 ($0E)	0

Anmerkung 1:
Die existierenden Zeichensätze erzeugen keine erkennbaren Zeichen auf dem Bildschirm. Aus diesem Grunde muß der Benutzer einen eigenen Zeichensatz definieren, wenn er diese Modi verwendet. Wie dies durchgeführt wird, steht in Kapitel 3 des DE RE ATARI

In Anhang B dieses Textes steht ein Beispiel für einen solchen speziellen Zeichensatz, der für diese Bildschirmmodi geeignet ist.

3.7 FEINSCROLLING DES TEXTBILDSCHIRMES

Durch den Bildschirm-Editor der XL-Serie wird nun das Feinscrolling der Bildschirmdaten des Text-Bildschirmes als Option möglich. Diese Feinscrolling-Option wird eingeschaltet, wenn die Datenbasis-Variable mit der Bezeichnung FINE (bei Speicherstelle $026E auf einen Wert

ungleich Null gesetzt wird, bevor ein OPEN-Befehl an den Bildschirm-Editor gesendet wird. Entsprechend wird das Feinscrolling abgeschaltet, wenn der Wert auf 0 gesetzt wird, bevor das OPEN-Kommando gegeben wird.

3.8 VERBESSERUNGEN BEI DER DISKETTEN-KOMMUNIKATION

Mit den XL-Computern ist es dem residenten Disketten-Handler möglich, Sektoren zu lesen und zu schreiben, die eine variable Länge von 1 bis 65536 Bytes besitzen. Der Standardwert dieser Sektoren liegt, wie bei den Geräten der alten Serie, bei 128 Bytes. Diese Größe wird sowohl beim Einschalten, als auch beim RESET wieder eingesetzt. Ein Benutzer-Programm kann diese Sektorenlänge über die OS-Systemvariable DSCTLN kontrollieren. Letztere ist eine 2-Byte Variable und liegt an den Adressen $02D5 und $02D6 (niederwertiges Byte in $02D5, höherwertiges in $02D6).

Zusätzlich besitzen die XL-Computer noch die Möglichkeit, einen Sektor ohne nachträgliche Überprüfung auf die Diskette zu schreiben, die beim alten 400/800er immer erfolgte. Dieses Kommando lautet "P" und befindet sich nicht in den alten Ausgaben des Operating-System.

Durch diese Möglichkeit kann der Benutzer entscheiden, ob er das Schreiben (zur Sicherheit) überprüfen möchte oder ob er aufgrund der dann erreichten Geschwindigkeit darauf verzichtet. Der Benutzer sollte einige Tests der Programmsicherung mit und ohne anschließende Überprüfung durchführen, um besser entscheiden zu können.

3.9 EIN-/AUSSCHALTEN DES TASTATUR-KLICKGERÄUSCHES

Der Benutzer kann diese Tasten-Bestätigung über ein Programm ein- oder ausschalten. Es muß hierfür lediglich die gleiche Variable geändert werden, die das Operating-System verwendet, um anzuzeigen, ob ein Klickgeräusch erzeugt werden soll oder nicht. Diese Variable wird mit NOCLIK bezeichnet und liest an der Speicherstelle $02DB.

Beim Einschalten des Gerätes und beim Drücken der RESET-Taste wird diese OS-Systemvariable auf einen Wert von 0 initialisiert. Dieser Wert bedeutet, daß bei Tastendruck ein Klicken ausgegeben wird. Soll das Geräusch abgeschaltet werden, so muß ein Wert von $FF in diese Adresse gespeichert werden.

3.10 WECHSEL ZWISCHEN AMERIKANISCHEM UND INTERNATIONALEM ZEICHENSATZ

Die internationale Version des Zeichensatzes liegt im ROM und beginnt bei Speicherstelle $CC00. Durch Schreiben des Wertes $CC nach Adresse $02F4 (= CHBAS) wird dieser Zeichensatz ausgewählt. Der normale (interne) Zeichensatz liegt ebenfalls im ROM und beginnt ab Adresse $E000. Wird ein Wert von $E0 in CHBAS gespeichert, so wird diese Zeichensatzversion ausgewählt (siehe DE RE ATARI).

Speicheraufteilung des XL-Systems

Die folgende Tafel zeigt, wie der 6502-Prozessor den Adressbereich aufteilt. Der maximale Adressbereich den der 6502-Prozessor mit 16 Bit ansteuern kann, liegt bei $0000 bis $FFFF. Dieser Adressbereich wird folgendermaßen durch die Hardware-Schaltungen aufgespalten.

Anmerkung: Der ATARI 8OOXL "TM" benutzt ebenso wie der erweiterte ATARI 600XL "TM" einen beschreibbaren Speicherbereich von 64-K-RAM. Dieser normalerweise ganz zugreifbare Bereich des RAMs wird durch den Memory-Manager (=Speicher-Verwaltung) so aufgeteilt, daß ROMs, Modul-Einschübe und Peripherien ein Teil des Speichers belegen können.

SPEICHERAUFTEILUNG

HEX-ADRESSE	BELEGT DURCH	ANMERKUNGEN
FFFF-D800	OS-ROM oder RAM, wenn ROM abgeschaltet	1
D7FF-D000	Durch Zugriffe auf diese Speicher-Page werden aktive Chip-Selektierungen für die Peripherie-Chips erzeugt.	
	I/O-Speicheraufteilung (memory-mapped) D000-D0FF GTIA D200-D2FF POKEY D300-D3FF PIA D400-D4FF ANTIC D500-D5FF Jede in diesem Bereich angesprochene Adresse aktiviert die CCNTL-Kontrollschaltung des Modul-Interfaces (wie bei der alten Serie).	
	D100-D1FF Sind für künftige D600-D6FF Belegung reserviert. D700-D7FF	
	OS-ROM physisch vorhanden, kann aber nicht angesprochen werden.	2
CFFF-C000	OS-ROM oder RAMs, wenn ROM abgeschaltet	1
BFFF-A000	RAM oder Modul-Interface, wenn RD5-Leitung durch den Einschub auf +5V an-	

	gesprochen wird.
9FFF-8000	RAM oder Modul-Interface,wenn RD4-Leitung durch den Einschub auf +5V angesprochen wird.
7FFF-5800	RAM
57FF-5000	RAM, solange nicht im Selbsttest-Modus 2
4FFF-000	RAM

Anmerkungen:

1. Der Zugriff auf das OS-Rom kann durch Schreiben eines Wertes von 0 nach Port 8 das PIA, Bit PB0, abgeschaltet werden. Der Zugriff wird normalerweise durch eine 1 in diesem Bit eingeschaltet. (Wird dieses Bit geändert, so dürfen andere Bits des Registers nicht beeinflusst werden.)

2. Der Selbsttest-ROM-Code ist an den Adressen $D000-$7FF im OS-Rom physisch vorhanden. Dieser Bereich wird allerdings für den Zugriff auf die in der Speicherliste aufgeführten I/O-Geräte benötigt. Wird der Selbsttest angesteuert, dann wird das RAM an den Speicherstellen $5000-$57FF abgeschaltet. Der Memory-Manager definiert den Speicherzugriff so, daß die Adressen $D000-$D7FF dem physischen OS-ROMs an den Speicherstellen $5000-57FF angesprochen werden. Er benutzt Port B des PIA, Bit PB7, um festzulegen, ob RAM oder ROM in dem Bereich von $5000-$57FF angesteuert werden soll. Ist das PB7 logisch 1, so wird RAM angesteuert. Andernfalls wird auf das OS-ROM zugegriffen. (Bei Änderungen dieses Bits sollten die anderen Bits des Registers nicht beeinflusst werden).

(Port 9 wurde in der alten 400/800er-Serie zum Bearbeiten der Joystick-Eingänge 3 & 4 benutzt.)

Verbesserungen der XL-Geräte Gegenüber der alten Serie

Die Funktionen des im alten ATARI 400/800"TM" vorhandenen RV.B Operating Systems wurden durch die nachstehenden aufgelisteten Möglichkeiten erweitert:

Drucker-CLOSE mit Daten-Puffer

Der Drucker-Handler fügt, bevor der Puffer bei einem CLOSE gesendet wird, ein EOL (End-of-Line)-Zeichen in den entsprechenden Puffer ein, sofern dieses nicht vorhanden ist. Hierdurch wird garantiert, daß die letzte Zeile sofort, anstatt in einer Sonderzeile, gedruckt wird.

Verarbeitung von Einheits-Nummern der Drucker

Der Drucker-Handler wurde so geändert, daß er nun auch die Nummer der Drucker-Einheit im IOCB verarbeitet. Hierdurch können bis zu 8 Drucker mit den Bezeichnungen Pl bis P8 adressiert werden.

CIO-Bearbeitung von unvollständigen Sätzen beim Lesen

Die CIO platziert jetzt ein EOL im Benutzer-Puffer, wenn ein zu langer Datensatz oder ein EOF beim Einlesen auftaucht. Hierdurch werden alle Datensätze zugreifbar. Auch wenn der Benutzer den Puffer zu klein gewählt hat, wird zumindest soviel vom Datensatz eingelesen, wie in diesem Puffer paßt.

CURSOR-Verarbeitung des Display-Handlers

Der Display-Handler akzeptiert jetzt unabhängig von den X- und Y-Werten der Cursor-Position einen Code zum Bildschirm-Löschen.

Speicher-Löschung des Display-Handler5/Bildschirm-Editors

Der Display-Handler und der Bildschirm-Editor löschen nun nicht mehr den Speicher bis über das durch RAMTOP angegebene Ende des Speichers hinaus. Daher ist es dem Benutzer möglich, das Ende des vom System zu verwendenden

Speicherbereiches anzugeben und Geräte-Handler sowie eigene Maschinen-Programme im Bereich über dem Bildschirm zu schreiben. Durch Ändern des Graphik-Modi tritt dann keine Löschung der hinter dem Bildschirm (und damit dem System zugänglichen Bereich) auf.

Überarbeitung der Fließkomma-Arithmetik

Durch das Operating-System der XL-Serie wird ein Fehler des RV.B OS korrigiert. Beim versuchten Berechnen des LOG oder LOG 10 von Null wird nun ein Fehler-Status erzeugt.

Neuer ROM-Vektor

Der folgende Einsprungpunkt wurde in das ROM der XL-Serie eingefügt:

E480 JMP PUPDIS Einsprung zum Selbsttest.

Andere Änderungen / Allgemeine Informationen

Dieser Abschnitt enthält Punkte, die Änderungen des Operating-Systems betreffen, aber schlecht unter Kategorien zu behandeln sind.

VERBESSERUNGEN DER BEARBEITUNG VON SYSTEM-VARIABLEN

Während der normalen Einschalt-Sequenz (Kaltstart) werden die System-Variablen von $03ED bis $03FF auf Null gesetzt. Beim RESET (Warmstart) geschieht dies NICHT über das OS. Ein zukünftiges Operating-System wird diese Speicherstellen also benutzen können, ohne daß sie nach einer RESET-Oparation neu geladen werden.

Alle Bytes dieses Bereiches sind für zukünftige OS-Versionen reserviert!

TIMING DER NTSC/PAL-VERSIONEN

Es gibt mehrere Timing-Unterschiede zwischen den PAL- (50Hz) und NTSC- (60Hz) Versionen. Um die Notwendigkeit eines speziellen OS-Roms für jede Version auszuschalten, werden die unterschiedlichen Werte für die Timing-Justierung in einem einzigen ROM-Satz bearbeitet.

Um zu bestimmen, mit welchem ROM-System es arbeitet, prüft das OS ein Flag im GTIA-Chip und richtet alle Timing-Erfordernisse daraufhin aus. Dies ist möglich, da der GTIA in verschiedenen Versionen existiert, um die Bildschirme der jeweiligen Fernseh-Systeme bearbeiten zu können. Indem bestimmte Timing-Zustände abhängig von diesem Flag gestaltet wurden, war es möglich, die externen Timing-Erfordernisse von den beiden Systemen unabhängig zu machen.

Die für die Bearbeitung des ATARI 410”TM” Cassetten-Recorders und der Wiederholungs-Rate der Tastatur wichtigen Timings werden in folgender Liste angegeben:

UNABHÄNGIGE CASSETTEN-TIMINGS	**Timing**
Zwischen-Satz-Lücke (IRG) beim Schreiben (lang)	3,0 Sek.
IRG beim Lesen (lang)	2,0 Sek.
IRG beim Schreiben (kurz)	0,25 Sek.
IRG-Verzögerung beim Lesen (kurz)	0,16 Sek.
Dateikopf beim Schreiben	19,2 Sek.

Dateikopf-Verzügerung beim Lesen	9,6 Sek.
Dauer des Piep-Tones	0,5 Sek.
Trennung des Piep-Tones	0,16 Sek.

UNABHANGIGE WIEDERHOLUNGS-FUNKTION	**Timing**
Anfangs-Verzögerung	0,8 Sek.
Wiederholungs-Rate	10 Zeich/Sek.

Anhang A
Beispiel für eine Neudefinition der Tastatur

Wie schon in diesem Text angesprochen, können die einzelnen Tasten des Keyboards umdefiniert werden. Die nachstehende Tabelle gibt die Tastenbelegung nach dem DVORAK-System (auch als vereinfachten amerikanisches System bezeichnet) an. Als die Schreibmaschine im Jahre 1967 erfunden wurde, wählte der Erfinder Christopher L. Shales eine Tastenanordnung, welche die Schnell-Schreiber(in) bremsen und so das Gerät vor Beschädigung schützen sollte. Dieses System der Tasten-Belegung hat sich bis zum heutigen Tag gehalten.

1932 wurde von August Dvorak eine neue Tasten-Anordnung erfunden, bei der die am häufigsten benutzten Zeichen, inklusive der Vokale, in die Grundlinie gelegt wurden. Außerdem wurden die restlichen Buchstaben so umgelegt, daß sich das Anschlag-Verhältnis links-rechts von 65% zu 35% auf etwa 50% zu 50% änderte. Von einigen Herstellern wird diese Dvorak-Tastatur als Option angeboten. Der Benutzer hat beim XL-System nun die Möglichkeit dieses Dvorak-System selbst auszuprobieren, indem er die Tastatur nach den angegebenen Richtlinien neudefiniert.

OBERE TASTENREIHE:

Normal: Qq Ww Ee R T Y U I o P 1/4 1/2
Dvorak: ?/ ,, .. P Y F G C R L ” ´

MITTLERE TASTENREIHE:

Normal: A S D F G H J K L :; ”´
Dvorak: A O E U I D H T N Ss --(Unterstrich)

UNTERE TASTENREIHE:

Normal: Zz x c v b n m , . ?/
Dvorak: :; Q J K X B M Ww Vv Zz

(Anm.d.Übers.: Die Dvorak-Anordnung ist speziell für die englische Sprache ausgelegt, d.h. die Anschlagoptimierung entspricht nicht dem deutschen Verhältnis.)

Anhang B
Vorschlag für einen speziellen Zeichensatz für die neuen Grafik-Modi

---

In diesem Anhang werden die neuen Grahik-Modi 12, 13, 14 und 15 behandelt. Die Bildschirm-Modi 14 und 15 sind reine Graphik-Modi mit einer Auflösung von 160 mal 20 bzw. 160 mal 40 Pixel. Da sie keine Zeichen-Modi sind, bezieht sich die nachfolgende Besprechung nur auf die Modi 12 und 13.

In den Bildschirm-Modi 12 und 13 werden mit dem normalen Zeichensatz keine lesbaren Zeichen erzeugt. Dies wird durch einen Vergleich zwischen Modus 0 und 12 bzw. 13 verständlich:

Modus 0 ist ein 40-Zeichen-Modus, bei dem jeder Buchstabe aus 9 waagerechten Pixeln besteht (kleinste waagerechte Bildschirm-Auflösung). Jedes Pixel besitzt eine Breite von 1/2 Color-Clock.

Die Bildschirm-Modi 12 und 13 besitzen ebenfalls 40 Zeichen pro Zeile, aber jeder Buchstabe besteht aus nur 4 waagerechten Pixeln, die jeweils eine Breite von einem Color-Clock besitzen. Ein Buchstabe ist daher genauso groß wie im Modus 0. Es ist wesentlich schwieriger, mit einer Auslösung von 4x8 Pixeln ein Zeichen darzustellen, als mit 8x8 Pixeln.

Überprüfen wir zunächst, wie ein 4-Pixel-Zeichen im Vergleich zu einem 8-Pixel-Zeichen aufgebaut ist:

Im Modus Null ist eine Wahl zwischen zwei Farben für jedes Pixel möglich. (Im Hardware-Manual werden 1 1/2 angegeben, aber es gibt zum einen die Farbe und Helligkeit des 2. Playfields - wenn sich eine 0 im entsprechenden Bit befindet - oder die Farbe und Helligkeit des 1. Playfields - sofern sich eine 1 an der entsprechenden Bit-Position befindet). Es wird daher nur ein einzelnes Bit zur Definition eines 1/2 Color-Clock breiten Pixels benötigt. Der im OS vorhandene Zeichensatz definiert die Zeichen in einer 8 mal 8 Matrix, die durch jeweils 9 Bytes festgelegt wird. (Jedes Byte gibt das Aussehen des Buchstaben oder Zeichens in einer Scan-Line an).

Modus 12 benötigt ebenfalls 8 Scan-Lines pro Zeichen. Allerdings werden die Bytes der Zeichen-Definition anders verwendet: jedes der von ANTIC eingeholten Datenbytes wird

als eine Einheit von vier 2-Bit Werten behandelt, wobei jedes Bit-Paar die Farbe eines der 2 Color-Clocks breiten Pixel im Zeichen festlegt. Im Modus 13 werden die Datenbytes genauso aufgespalten, aber anders als bei Modus 12 sind die einzelnen Zeichen 16 anstatt 8 Scan-Lines hoch. Ein Datenbyte gibt daher das Aussehen von jeweils zwei Scan-Lines an.

Betrachten wir nun ein normales Zeichen, z.B. das W. Die den Buchstaben darstellenden Bits sind folgendermaßen angeordnet:

```
10000001 Anzeige:   *      *
10000001            *      *
10011001            *  **  *
10011001            *  **  *
10100101            * *  * *
10100101            * *  * *
11000011            **    **
11000011            **    **
10000001            *      *
```

Anmerkung: Dies ist nicht die exakte Darstellung, wird aber hier verwendet, um den Unterschied zwischen der richtigen Bildschirm-Darstellung in Modus 0 und der falschen in den Modi 12 und 13 zu verdeutlichen.

Betrachtet man nun das Ergebnis der Bit-Übertragung auf den Bildschirm, so ergibt sich in der Tat ein lesbares “W”. Die Bits geben die jeweiligen Punkte des Zeichens an.

Im Modus 0 erzeugt jede 1 eine Farbe, eine 0 den Hintergrund, so das ein lesbarer Buchstabe entsteht.

In den Modi 12 und 13 ist dies nicht der Fall, da die 4 (anstatt 8) Pixel wie folgt kontrolliert werden:

Wert des Bit-Paares:	Farbe des Pixels:
00	Hintergrundfarbe
01	Farbe von Playfield 0
10	Farbe von Playfield 1
11	Farbe von Playfield 2 (wenn Bit 7 des Zeichennamens = 0)
11	Farbe von Playfield 3 (wenn Bit 7 dem Zeichennames = 1).

Für das gezeigte Beispiel wäre die 4. Zeile von unten folgendermaßen gefärbt: 4 Pixel in den Farben Playfield 1, 1, 0 und 0. Die Farben der letzten Zeile wären Playfield 1, Hintergrundfarbe, Hintergrund und Playfield 0. Ein solches Zeichen ist verständlicherweise nicht mehr lesbar. (Dieser Buchstabe ist dabei noch spiegelsymmetrisch zur Mittelachse - nichtsymmetrisch Zeichen wären noch schwerer zu erkennen.)

Ein Zeichensatz für diese Bildschirm-Modi 12 und 13 könnte so aufgebaut werden, daß jeweils zwei Datenblöcke von 8 Bytes aufgestellt werden, so daß ein Buchstabe wieder eine Breite von 8 Pixel erhält. Um einen Buchstaben in dieser Form auf den Bildschirm auszugeben, müßte verfahren werden, wie bei der Ausgabe von zwei Zeichen im Modus 0. So könnte ein "W" z.B. wie folgt aufgebaut werden:

Byte-Satz 1:	Byte-Satz 2:
10 00 00 00	00 00 00 10
10 00 00 00	00 00 00 10
10 00 00 10	10 00 00 10
10 00 00 10	10 00 00 10
10 00 10 00	00 10 00 10
10 00 10 00	00 10 00 10
10 10 00 00	00 00 10 10
10 10 00 00	00 00 10 10
10 00 00 00	00 00 00 10

Byte-Satz 1 könnte z.B. an ATASCII-Stelle $57 des neuen Zeichensatzes definiert werden. Byte-Satz 2 an Stelle $D7 (= $57 plus $80). Natürlich bleibt diese Verteilung jedem Benutzer selbst überlassen.

Werden diese beiden Zeichen auf den Bildschirm gedruckt, so erhält man zum einen zwar ein lesbares Zeichen, zum anderen aber passen nur noch 20 Buchstaben in eine Zeile. Die Bit-Kombination 10 entspricht hierbei der 1 im Beispiel für Modus 0, die Kombination 00 der 0. Der entsprechende Buchstabe besitzt also die gleiche Form, wie im Beispiel für Modus 0.

Der Benutzer besitzt auch die Möglichkeit, die Zeichen in einem von einer 8 x 8 Matrix differierenden Gitter zu definieren. (Bei dem gegebenen Beispiel bestünden zwischen den einzelnen Zeichen keine Lücken. Sie würden auf dem Bildschirm ineinander übergehen.)

Durch die Modi 12 und 13 können auch Buchstaben und Zeichen erzeugt werden, die aus mehreren Farben gleichzeitig bestehen. Es ist also, wie im Kapitel 3 des DE RE ATARI angesprochen, die Darstellung mehrfarbiger Karten usw. mit einem Zeichensatz möglich.

Nützliche Systemadressen

ZERO-PAGE-ADRESSEN DES BETRIEBSSYSTEMS

Abk.	Hex	dez	Bytes	Kurzbeschreibung
CASINI	0002	2	2	Kassettenbootinitialisierungsvektor
WARMST	0008	8	1	Warmstart-Flag: $00=Coldstart,$FF=Warmstart
BOOT	0009	9	1	Boot-Flag: Bit0 für Diskettenboot Bit1 für Kassettenboot
DOSVEC	000A	10	2	Diskettenprogrammstartvektor (z.B. DOS)
DOSINI	000C	12	2	Diskettenbootinitialisierungsvektor
APPMHI	000E	14	2	erster frei verfügbarer Speicherplatz
POKMSK	0010	16	1	POKEY-Interrupt-Maske: Bit7 für BREAK-Tasten-Interrupt Bit6 für Tastatur-Interrupt
BRKKEY	0011	17	1	BREAK-Tasten-Flag: 0=BREAK gedrückt
RTCLOCK	0012	l8	3	interne Uhr
SOUNDER	0041	65	1	Ein-/Ausgabe-Geräusch-Flag: 0=leise
CRITIC	0042	66	1	Flag für kritische Operationen
ATRACT	004D	77	1	ATTRACT-Mode-Flag: Falls Wert > 127 Beginnt das automatische Wechseln der Farben. Wird mit dem mittleren Byte der internen Uhr inkrementiert.
LMRGIN	0052	82	1	linker Bildschirmrand
RMRGIN	0053	83	1	rechter Bildschirmrand
ROWCRS	0054	84	1	Zeilennummer des Cursors im Grafikfenster
COLCRS	0055	85	2	Spaltennummer des Cursors im Grafikfenster
CRMODE	0057	87	1	benutzter Grafikmodus
SAVMSC	0058	88	2	obere linke Ecke des Bildschirms
RAMTOP	006A	106	1	Anzahl der RAM-Seiten
KEYDEF #	0079	121	2	Zeiger zur Tastatur-Belegungs-Tabelle

Erklärung der erwendeten Symbole:

* nur bei ATARI 400/800
# nur bei ATARI 600XL/8OOXL

ZERO-PAGE-ADRESSEN DES BASICS

Abk.	Hex	dez	Bytes	Kurzbeschreibung
LOMEM	0080	128	2	Anfang des vom BASIC benutzten Speichers
VNTP	0082	130	2	Zeiger zur Tabelle der Variablennamen
VNTD	0084	132	2	Zeiger zum Ende der Tabelle der Variablen-namen
VVTB	0086	134	2	Zeiger zur Tabelle der Variablenwerte
STMTAB	0088	136	2	Zeiger zur Befehlsliste des eingegebenen Programms
STMCUR	008A	138	2	Zeiger zum augenblicklichen Befehl
STARP	008C	140	2	Zeiger zum Anfang des Feldvariablenwerte-bereichs

Abk.	Hex	dez	Bytes	Kurzbeschreibung
RUNSTK	008E	142	2	Zeiger zum Laufzeit-Stapel
MEMTOP	0090	144	2	Zeiger zum Ende des vom BASIC benutzten Speichers
STOPLN	00B9	186	2	Nummer der Zeile, in der ein STOP gefunden wurde bzw. in der ein Fehler auftrat
ERRSAVE	00C3	195	1	Fehlercode zu STOPLN
PTABW	00C9	201	1	Tabulatorabstand für PRINT-Befehle

ZERO-PASE-ADRESSEN DES GLEITKOMMAPAKETS

Abk.	Hex	dez	Bytes	Kurzbeschreibung
FR0	00D4	212	6	Gleitkommaregister 0
FR1	00E0	224	6	Gleitkommaregister 1
INBUFF	00F3	243	2	Zeiger auf einen ASCII-Text-Puffer
FLPTR	00FC	252	2	Zeiger zu einer Gleitkommazahl

WEITERE SYSTEMVARIABLEN DES BETRIEBSSYSTEMS

Abk.	Hex	dez	Bytes	Kurzbeschreibung
VDSLST	0200	512	2	Display-List-Interrupt-Vektor (DLI)
VPRCED	0202	514	2	Vektor zur IRQ-Routine für die Bearbeitung von Operationen das seriellen Busses
VINTER	0204	516	2	Vektor zur Seriellen-Bus-IRQ-Interrupt-Routine
VBREAK	0206	518	2	BREAK-Interrupt-Vektor
VKEYBD	0208	520	2	Tastatur-Interrupt-Vektor
VSERIN	020A	522	2	Vektor zur Routine zum seriellen Daten-empfang
VSEROR	020C	524	2	Vektor zur Routine zum seriellen Daten-senden
VSEROC	020E	526	2	Vektor zur Routine zum Beenden des seriellen Datensendens
VTIMR1	0210	528	2	POKEY-Timer-1-IRQ-Interrupt-Vektor
VTIMR2	0212	530	2	POKEY-Timer-2-IRQ-Interrupt-Vektor
VTIMR4	0214	532	2	POKEY-Timer-4-IRQ-Interrupt-Vektor
VIMIRQ	0216	534	2	IRQ-Interrupt-Hauptvektor
CDTMV1	0218	536	2	Wert des System-Timers 1
CDTMV2	021A	538	2	Wert des System-Timers 2
CDTMV3	021C	540	2	Wert des System-Timers 3
CDTMV4	021E	542	2	Wert des System-Timers 4
CDTMV5	0220	544	2	Wert des System-Timers 5
VVBLKI	0222	546	2	Immediate-Vertical-Blank-Vektor
VVBLKD	0224	548	2	Deferred-Vertical-Blank-Vektor
CDTMA1	0226	550	2	System-Timer-1-Interrupt-Vektor
CDTMA2	0228	552	2	System-Timer-2-Interrupt-Vektor
CDTMF3	022A	554	1	System-Timer-3-Flag
CDTMF4	022C	556	1	System-Timer-4-Flag
CDTMF5	022E	558	1	System-Timer-5-Flag
SDMCTL	022F	559	1	Register für den direkten Speicherzugriff des ANTICs (Schattenregister von $D400)
SDLSTL	0230	560	2	Zeiger zum Beginn der Display-List
LPENH	0234	564	1	horizontale Lightpen-Position
LPENV	0235	565	1	vertikale Lightpen-Position
COLDST	0244	580	1	Kaltstart-Flag; 1=Kaltstart ausführen, 0=Warmstart ausführen, wenn SYSTEM RESET gedrückt wird

Abk.		Hex	dez	Bytes	Kurzbeschreibung
KEYDIS	#	026D	621	1	Tastatur-Abschalt-Flag
FINE	#	026E	622	1	Fine-Scroll-Flag
SPRIOR		026F	623	1	Priaritätsregister (Schattenregister von $D01B)
PADDL0		0270	624	1	Wert des Paddles 0
PADDLI		0271	625	1	Wert des Paddles 1
PADDL2		0272	626	1	Wert des Paddles 2
PADDL3		0273	627	1	Wert des Paddles 3
PADUL4	*	0274	628	1	Wert des Paddles 4
PADDL5	*	0275	629	1	Wert des Paddles 5
PADDL6	*	0276	630	1	Wert des Paddles 6
PADDL7	*	0277	631	1	Wert des Paddles 7
STICK0		0278	632	1	Wert des Joysticks 0
STICK1		0279	633	1	Wert des Joysticks 1
STICK2	*	027A	634	1	Wert des Joysticks 2
STICK3	*	027B	635	1	Wert des Joysticks 3
PTRIG0		027C	636	1	Paddletrigger 0
PTRIG1		027D	637	1	Paddletrigger 1
PTRIG2		027E	638	1	Paddletrigger 2
PTRIG3		027F	639	1	Paddletrigger 3
PTRIG4	*	0280	640	1	Paddletrigger 4
PTRIG5	*	0281	641	1	Paddletrigier 5
PTRIG6	*	0282	642	1	Paddletrigger 6
PTRIG7	*	0283	643	1	Paddletrigger 7
STRIG0		0284	644	1	Joysticktrigger 0
STRIG1		0285	645	1	Joysticktrigger 1
STRIG2	*	0286	646	1	Joysticktrigger 2
STRIG3	*	0287	647	1	Joysticktrigger 3
TXTROW		0290	656	1	Zeilennummer des Cursors im Textfenster
TXTCOL		0291	657	2	Spaltennummer des Cursors im Textfenster
TXTMSC		0294	660	2	obere linke Ecke des Textfensters
TABMAP		02A3	675	15	Tabulator-Tabelle
INVFLG		02B6	694	1	Invers-Video-Flag: $00=normal,$80=invers
SHFLOK		02BE	702	1	SHIFT-Verriegelungsflag: $00=Kleinbuchstaben, $40=Großbuchstaben, $80=CTRL-Verriegelung, $FF=alle Buchstaben werden ignoriert
BOTSCR		02BF	703	1	Anzahl der Textzeilen (0,4 oder 24)
PCOLR0		02C0	704	1	Farbregister für Player/Mi5sile 0
PCOLR1		02C1	705	1	Farbregister für Player/Missile 1
PCOLR2		02C2	706	1	Farbregister für Player/Missile 2
PCOLR3		02C3	707	1	Farbregister für Player/Missile 3
COLOR0		02C4	708	1	Farbregister für Spielfeld 0
COLOR1		02C5	709	1	Farbregister für Spielfeld 1
COLOR2		02C6	710	1	Farbregister für Spielfeld 2
COLOR3		02C7	711	1	Farbregister für Spielfeld 3
COLOR4		02C8	712	1	Farbregi5ter für Hintergrundfarbe
KRFDEL	#	02D9	729	1	Tastaturverzögerungsrate
KEYREP	#	02DA	730	1	Tastaturwiederholungsrate
NOCLIK	#	02DB	731	1	Tastatur-Klick-Flag
HELPFS	#	02DC	732	1	HELP-Flag
RAMSIZ		02E4	740	1	Anzahl der RAM-Seiten
MEMTOP		02E5	741	2	RAM-Endezeiger des Betriebssystems
MEMLO		02E7	743	2	RAM-Anfangszeiger des Detriebssystems
DVSTAT		02EA	746	1	Geräte-Status
CRSINH		02F0	752	1	Cursor-Ein/Aus-Flag

Abk.	Hex	dez	Bytes	Kurzbeschreibung
CHACT	02F3	755	1	Zeichen-Darstellungs-Register: 0=Darstellung von invers als normal 1=Darstellung von invers als Blanks 2=Darstellung von invers als invers 3=Darstellung von invers als inv. Blöcke Mit 4..7 erzielt man die gleichen Effekte wie mit 0 bis 3 nur die Zeichen stehen auf dem Kopf
CHBAS	02F4	756	1	Zeichensatzanfang-Seitennummer: 204=europäischer Zeichensatz 224=amerikanisEher Zeichensatz (groß) 226=amerikanischer Zeichensatz (klein)
CH	02FC	764	1	Tastatur-Code der letzten gedrückten Taste
FILDAT	02FD	765	1	Farbwert für den Fill-Befehl (XIO 18)
DSPFLG	02FE	766	1	CTRL-Zeichen-Darstellungs-Flag: 0=normal, 1=CTRL-Zeichen darstellbar
SSFLAG	02FF	767	1	Start-/Stop-Flag für Ausgabe: 0=normale Ausgabe, 1=Anhalten der Ausgabe (entspricht CTRL-1)

Abk.	Hex	dez	Bytes	Kurzbeschreibung
DDEVIC	0300	768	1	Geräte-Identifikation
DUNIT	0301	769	1	Gerätenummer
DCOMND	0302	770	1	Kommandobyte
DSTATS	0303	771	1	Gerätestatus
DBUFLO	0304	772	1	Daten-Pufferadresse How Byte)
DBUFHI	0305	773	1	Daten-PuHeracresee (high Byte)
DTIMLO	0306	774	1	Wartezeit für Gerätehandler
DBYTLO	0308	776	1	Bytezahl für die übertragung (low Byte)
DBYTHI	0309	777	1	Bytezahl für die übertragung (high Byte)
DAUX1	030A	778	1	Sektornummer (low Byte)
DAUX2	030B	779	1	Sektornummer (high Byte)
HATABS	031A	794	34	Handler-Adressen-Tabelle
IOCB0	0340	832	16	I/O-Kontroll-Block 0
ICCOM	0342	834	1	Kommandobyte
ICSTA	0343	835	1	Statusbyte
ICBAL	0344	836	1	Puffer-Adresse (low Byte)
ICBAH	0345	837	1	Puffer-Adresse (high Byte)
ICBLL	0348	840	1	Puffer-Länge (low Byte)
ICBLH	0349	841	1	Puffer-Länge (high Bytel
ICAX1	034A	842	1	Hilfs-Byte 1
ICAX2	034B	843	1	Hilfs-Byte 2
ICAX3/6	034C	844	4	Hilis-Bytes 3/6
IOCB1	0350	848	16	I/O-Kontroll-Block 1 (Aufteilung s. IOCB0)
I0CB2	0360	864	16	I/O-Kontroll-Block 2 (Aufteilung s. IOCB0)
IOCB3	0370	880	16	I/O-Kontroll-Block 3 (Aufteilung s. IOCB0)
I0CB4	0380	896	16	I/O-Kontroll-Bleck 4 (Aufteilung s. IOCR0)
IOCB5	0390	912	16	I/O-Kontroll-Block 5 (Aufteilung s. IOCB0)
IOCB6	03A0	928	16	I/O-Kontroll-Block 6 (Aufteilung s. IOCB0)
IOCB7	03B0	944	16	I/O-Kontroll-Block 7 (Aufteilung s. IGCB0)

Abk.	Hex	dez	Bytes	Kurzbeschreibung
HPOSP0	D000	53248	1	(S) horizontale Position von Player 0
M0PF				(L) Kollisionsregister Missile 0 mit Spielfeld
HPOSP1	D001	53249	1	(S) horizontale Position von Player 1
M1PF				(L) Kollisionsregister Missile 1 mit Spielfeld
HPOSP2	D002	53250	1	(S) horizontale Position von Player 2
M2PF				(L) Kollisionsregister Missile 2 mit Spielfeld
HPOSP3	D003	53251	1	(S) horizontale Position von Player 3
M3PF				(L) Kollisionsregister Missile 3 mit Spielfeld
HPOSM0	D004	53252	1	(S) horizontale Position von Missile 0
P0PF				(L) Kollisionsregister Player 0 mit Spielfeld
HPOSM1	D005	53253	1	(S) horizontale Position von Missile 1
P1PF				(L) Kollisionsregister Player 1 mit Spielfeld
HPOSM2	D006	53254	1	(S) horizontale Position von Missile 2
P2PF				(L) Kollisionsregister Player 2 mit Spielfeld
HPOSM3	D007	53255	1	(S) horizontale Position von Missile 3
P3PF				(L) Kollisionsregister Player 3 mit Spielfeld
SIZEP0	D008	53256	1	(S) Größe von Player 0
M0PL				(L) Kollisionsregister Missile 0 mit Player
SIZEP1	D009	53257	1	(S) Größe von Player 1
M1PL				(L) Kollisionsregister Missile 1 mit Player
SIZEP2	D00A	53258	1	(S) Größe von Player 2
M2PL				(L) Kollisionsregister Missile 2 mit Player
SIZEP3	D00B	53259	1	(S) Größe von Player 3
M3PL				(L) Kollisionsregister Missile 3 mit Player
SIZEM	D00C	53260	1	(S) Größe der Missiles
P0PL				(L) Kollisionsregister Player 0 mit Player
GRAFP0	D00D	53261	1	(S) Form von Player 0
P1PL				(L) Kollisionsregister Player 1 mit Player
GRAFP1	D00E	33262	1	(S) Form von Player 1
P2PL				(L) Kollisionsregister Player 2 mit Player
GRAFP2	D00F	53263	1	(9) Form von Player 2
P3PL				(L) Kollisionsregister Player 3 mit Player
GRAFP3	D010	53264	1	(S) Form von Player 3
TRIG0				(L) Joysticktrigger 0
GRAFPM	D011	53265	1	(S) Form aller Missiles
TRIG1				(L) Joysticktrigger 1
COLPM0	D012	53266	1	(S) Farbregister für Player/Missile 0
TRIG2				(L) Joysticktrigger 2
COLPM1	D013	53267	1	(S) Farbregister für Player/Missile 1
TRIG3				(L) Joysticktrigger 3
COLPM2	D014	53268	1	(S) Farbregister für Player/Missile 2
COLPM3	D015	53269	1	(S) Farbregister für Player/Miesile 3
COLPF0	D016	53270	1	(S) Farbregister für Spielfeld 0
COLPF1	D017	53271	1	(S) Farbregister fur Spielfeld 1
COLPF2	D018	53272	1	(S) Farbregister für Spielfeld 2
COLPF3	D019	53273	1	(S) Farbregi5ter für Spielfeld 3
CQLBK	D01A	53274	1	(S) Farbregister fur Hintergrundfarbe

Abk.		Hex	dez	Bytes	Kurzbeschreibung
PRIOR		D01B	53275	1	(S) Prioritätsregister: Bit 7 und Bit 6 für GTIA-Grafikmodi Bit 5 läßt bei überlappenden Playern Mischfarben entstehen Bit 4 läßt alle Missiles zu einem Player mit der Farbe aus COLPF3 zusammenfassen Bit 3 Priorität: PF0,PF1,P0-P3,PF2,PF3,BK Bit 2 Priorität: PF0-PF3,P0-P3,BK Bit 1 Priorität: P0-P1,PF0-PF3,P2-P3,BK Bit 0 Priorität: P0-P3,PF0-PF3,BK
GRACTL		D01D	53277	1	Grafik-Kontroll-Register: Bit 0 für direkten Missile-Speicherzugriff Bit 1 für direkten Player-Speicherzugriff
HITCLR		D01E	53278	1	Kollisionsregister-Löschung
CONSOL		D01F	33279	1	Funktionstasten-Register: Bit 0 für START-Taste Bit 1 für SELECT-Taste Bit 2 für OPTION-Taste
AUDF1		D200	53760	1	(S) Geräusch-Frequenzregister Kanal 1
POT0					(L) Potentiometer (Paddle) 0
AUDC1		D201	53761	1	(S) Geräusch-Kontrollregister Kanal 1
POT1					(L) Potentiometer (Paddle) 1
AUDF2		D202	53762	1	(S) Geräusch-Frequenzregister Kanal 2
POT2					(L) Potentiometer (Paddle) 2
AUDC2		D203	53763	1	(S) Geräusch-Kontrollregister Kanal 2
POT3					(L) Potentiometer (Paddle) 3
AUDF3		D204	53764	1	(S) Geräusch-Frequenzregister Kanal 3
POT4	*				(L) Potentiometer (Paddle) 4
AUDC3		D205	53765	1	(S) Geräusch-Kontrollregister Kanal 3
POT5	*				(L) Potentiometer (Paddle) 5
AUDF4		D206	53766	1	(S) Geräusch-Frequenzregister Kanal 4
POT6	*				(L) Potentiometer (Paddle) 6
AUDC4		D207	53767	1	(S) Geräusch-Kontrollregister Kanal 4
POT7	*				(L) Potentiometer (Paddle) 7
AUDCTL		D208	53768	1	(S) Audio-Kontroll-Register
KBCODE		D209	53769	1	(S) Tastatur-Code-Register
RANDOM		D20A	53770	1	(L) Zufahlszahlengenerator
SEROUT		D20D	53773	1	(S) Ausgaberegister für den seriellen Bus
SERIN					(L) Eingaberegister für den seriellen Bus
IRQEN		D20E	54774	1	(S) POKEY-Interrupt-Maske
PORTA		D300	54016	1	(L/S) Datenregister für Joystickports 1&2 (falls Bit 2 in PACTL=1) (S) Datenrichtungsregister für Joystickports 1&2 (falls Bit 2 in PACTL=1)
PORTB	*	D301	54017	1	(L/S) Datenregister für Joystickports 3&4 (falls Bit 2 in PBCTL=1) (S) Datenrichtungsregister für Joystickports 3&4 (falls Bit 2 in PBCTL=1)
PORTB	#	D301	54017	1	(S) RAM/ROM-Kontrolle: Bit 0 für das Betriebssystem-ROM Bit 1 für das BASIC-ROM Bit 7 für den Selbsttest

Abk.	Hex	dez	Bytes	Kurzbeschreibung
PACTL	D302	54018	1	(S) Kontrollregister für Port A: $4C schaltet Motor des Kassettenrecorders ab $46 schaltet Motor des Kassettenrecorders an
PBCTL *	D303	54019	1	(S) Kontrollregister für Port B

Abk.	Hex	dez	Bytes	Kurzbeschreibung
DMACTL	D400	54272	1	(S) Register für direkten Speicherzugriff des ANTICs: Bit 0 und Bit 1 für die Spielfeldbreite Bit 2 für den Missile-Speicherzugriff Bit 3 für den Player-Speicherzugriff Bit 4 für die vertikale Playerauflösung Bit 5 für die Speicherzugriffsanweisungen
CHACTL	D401	54273	1	(S) Zeichendarstellungs-Register(vgl.$2F3)
DLISTL	D402	54274	1	(S) Zeiger zur Display-List (low Byte)
DLISTH	D403	54275	1	(S) Zeiger zur Display-List (high Byte)
HSCROL	D404	54276	1	(S) horizontales Scroll-Register
VSCROL	D405	54277	1	(S) vertikales Scroll-Register
PMBASE	D407	54279	2	(S) Player/Missile-Bereichs-Adresse
CHBASE	D409	54281	1	(S) Zeichensatzanfang-Seitennummer
WSYNC	D40A	54282	1	(S) Warten auf Horizontalsynchronisation
VCOUNT	D40B	54283	1	(L) Bildschirm-Zeilen-Zähler
PENH	D40C	54204	1	(L) horizontale Lightpen-Position
PENV	D40D	54285	1	(L) vertikale Lightpen-Position
NMIEN	D40E	54286	1	(S) NMI-Interrupt-Maske
NMIRES	D40F	54287	1	(S) NMI-Reset
NMIST				(L) NMI-Status

EINSPRUNSPUNKTE DES GLEITKOMMAPAKETES

Abk.	Hex	dez	Bytes	Kurzbeschreibung
AFP	D800	55296	-	Umwandlung von ASCII- in Gleitkomma-darstellung
FASC	D8E6	55526	-	Umwandlung von Gleitkomma- in ASCII-Darstellung
IFP	D9AA	55722	-	Umwandlung von Ganzzahl- in Gleitkomma-darstellung
FPI	D9D2	55762	-	Umwandlung von Gleitkomma- in Ganzzahl-darstellung
ZFR0	DA44	55876	-	FR0 löschen (FR0=0)
ZF1	DA46	54878	-	Löschen e. Zero-Page-Gleitkommaregisters (Registeradresse muß im X-Register sein)
FSUB	DA60	55904	-	Gleitkommasubtraktion (FR0=FR0-FR1)
FADD	DA66	55910	-	Gleitkommaaddition (FR0=FR0+FR1)
FMUL	DADB	56027	-	Gleitkommamultiplikation (FR0=FR0*FR1)
FDIV	DB28	56104	-	Gleitkommadivision (FR0=FR0/FR1)
PLYEVL	DD40	56640	-	Polynom-Berechnung: $FR0=A_n*FR0^n+A_{n-1}*FR0^{n-1}+...+A1*FR0+A0$ Startadresse der Koeffizientenliste A, (i=n...0) muß im X- und Y-Register stehen Anzahl der Koeffizienten muß im Akku stehen
FLD0R	DD89	56713	-	Laden des Gleitkommaregisters FR0 Startadresse des Wertes muß im X- und Y-Register stehen

Abk.	Hex	dez	Bytes	Kurzbeschreibung
FLD0P	DDSD	56717	-	Laden des Gleitkommaregisters FR0 Startadresse des Wertes muß in FLPTR stehen
FLD1R	DD98	56728	-	Laden des Gleitkommaregisters FR1 Startadresse des Wertes muß im X- und Y-Register stehen
FLD1P	DD9C	56732	-	Laden des Gleitkommaregisters FR1 Startadresse des Wertes muß in FLPTR stehen
FSTOR	DDA7	56743	-	Speichern des Gleitkommaregisters FR0 Startadresse des Wertes muß im X- und Y-Register stehen
FSTOP	DDAB	56747	-	Speichern des Gleitkommaregisters FR0 Startadresse des Wertes muß in FLPTR stehen
FMOVE	DDB6	56758	-	Kopieren von FR0 in FR1 (FR1=FR0)
EXP	DDC0	56768	-	Exponentialfunktion ($FR0=e^{FR0}$)
EXP10	DDCC	56780	-	Exponentialfunktion zur Basis 10 ($FR0=10^{FR0}$)
LOG	DECD	57037	-	natürlicher Logarithmus ($FR0=\log_e(FR0)$)
LOG10	DED1	57041	-	dekadischer Logarithmus ($FR0=\log_{10}(FR0)$)

BETRIEBSSYSTEMVEKTOREN

Abk.		Hex	dez	Bytes	Kurzbeschreibung
DISKIV		E450	58448	-	Disk-Handler-Initialisierungsvektor
DSKINV		E453	58451	-	Disk-Handler-Vektor
CIOV		E456	58454	-	CIO-Vektur
SI0V		E459	38457	-	SIO-Vektor
SETVBV		E45C	58460	-	System-Timer-Aufruf-Vektor
SYSVBV		E45F	58463	-	System-Vertical-Blank-Routinen-Vektor
XITVBV		E462	58466	-	Vertical-Blank-Abschluß-Routinen-Vektor
SIOINV		E465	58469	-	SIO-Initialisierungs-Vektor
SENDEV		E468	58472	-	Aktivierungsroutinen-Vektor zum Senden über den seriellen Bus
INTINV		E46B	58475	-	Interrupt-Handler-Initialisierungs-Vektor
CIOINV		E46E	58478	-	CIO-Initialisierungs-Vektor
BLKBDV	*	E471	58481	-	Memo-Pad-Modus-Einsprung-Vektor
	#				Selbst-Test-Einsprung-Vektor
WARMSV		E474	58484	-	Warmstart-Einsprung-Vektor
COLDSV		E477	58487	-	Kaltstart-Einsprung-Vektor
RBLOKv		E47A	58490	-	Kassetten-Block-Lesen-Einsprung-Vektor
CSOPIV		E47D	58493	-	Vektor zur Kassetten-OPEN-Routine zum Lesen

Erklärung der verwendeten Symbole:

* nur bei ATARI 400/800
# nur bei ATARI 600XL/8OOXL