Dullin
Strassenburg

MSX

Tips & Tricks

EIN DATA BECKER BUCH

Dullin
Strassenburg

MSX

Tips & Tricks

EIN DATA BECKER BUCH

ISBN 3-89011-112-2

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

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

DANKWORT

Das Schreiben eines solchen Buches macht zugegebenermaßen meistens nur den Autoren Spaß. Für Mitbewohner, Freundinnen und Freunde kann aber leider "Frust" auftreten, da wir die Autoren, in einem sog. "Bücherarbeitsrausch und Computerwahn" stecken müssen, um ein Buch zu vollenden. Wir möchten uns bei all denen bedanken, die durch ihre Geduld und Unterstützung dieses Buch möglich gemacht haben. Insbesondere danken wir Birgitt Mikutta, Kristin Grünewald und Andreas Bethmann für die unschätzbare Hilfe bei der Korrektur des Manuskriptes.

VORWORT

Endlich ist er da!

Der MSX Standard für Homecomputer. Für uns ist ein langersehnter Wunsch Wirklichkeit geworden. Es gibt endlich über 20 Rechner, deren Soft- und Hardware kompatibel, d.h. untereinander austauschbar ist. Der Vorteil besteht darin, daß man nun Programme, die auf anderen MSX Rechnern geschrieben wurden, ohne Probleme auf seinem eigenen MSX Computer verwenden kann. Damit dürfte in kurzer Zeit fast unbegrenzt viel Software für den Benutzer zur Verfügung stehen.

Die Buchstaben MSX stehen für "Microsoft Extended BASIC". Damit wird deutlich, daß die renomierte Firma Microsoft das BASIC für diese Rechner entwickelt hat. Das MSX BASIC zeichnet sich durch seine Vielfalt von nützlichen Befehlen aus. Es fehlen weder Befehle zur Unterstützung vom Grafik und Sound, noch Befehle, die das Editieren angenehm erleichtern, wie z.B. >TRON< oder >TROFF<.

Damit Ihnen das Programmieren mit diesen Rechnern noch mehr Freude bereitet, haben wir dieses Buch geschrieben. Hier finden Sie nützliche "Tips & Tricks", die Sie in ihren eigenen Programmen sinnvoll verwenden können. Außerdem erhalten Sie wichtige Informationen zum internen Aufbau und zur Maschinensprache dieser Rechner.

Wir wünschen Ihnen beim Umgang mit diesem Buch viel Spaß und Erfolg!

Die Autoren

INHALTSVERZEICHNIS

KAPITEL III : I/O

KAPITEL IV: SOUND MIT DEM SOUND CHIP

KAPITEL V : MASCHINENSPRACHE

KAPITEL VI: SYSTEMROUTINEN

KAPITEL VII : PEEKS und POKES

KAPITEL VIII: BASIC INTERN

KAPITEL IX : KRAMECKE

KAPITEL X : PROGRAMME

ANHANG

KAPITEL I : D E R M S X S T A N D A R D

1.1 WAS IST EIN STANDARD?

Kein Mensch verschwendet noch Gedanken darauf, ob ein Fernsehgerät, Radiogerät o.ä., daß er gekauft hat, zu Hause an seine Stromversorgung angeschlossen werden kann. Es ist selbstverständlich, daß der Stecker des Elektrogerätes auch in die Steckdose paßt. Überraschungsmomente erlebt man höchstens einmal bei Reisen ins Ausland. Hier kann man feststellen, daß entweder der gelieferte Strom eine andere Spannung, oder der Stecker nicht in die Steckdose paßt. Erst dann wird deutlich, daß wenigstens länderspezifisch eine Absprache über die Entwicklung eines Standards, einer Norm, zwischen Firmen stattgefunden hat oder auch nicht.
Ein Standard ist in anderen technischen Bereichen fast selbstverständlich. Nur im Homecomputerbereich war das, aufgrund der schnellen Entwicklung, aber wohl hauptsächlich aus Konkurrenzgründen nicht der Fall. Hier ist es für den Anwender oft umständlich, wenn nicht gar unmöglich, einen Computer mit Peripherie nach eigener Wahl auszustatten. Denn oft passen nur die Geräte der Firma, die auch den Computer hergestellt hat, ohne große Komplikation zum eigenen Rechner.
Gerade bei Software ist diese Tatsache oft sehr ärgerlich. Man hört von wirklich guten Programmen z.B. Textverarbeitung, die aber auf dem eigenen Computer nicht laufen. Daher muß ein Kompromiß in Bezug auf das zu verwendende Programm eingegangen werden.
Es ist also eine erfreuliche, verbraucherfreundliche Entwicklung, die durch den Zusammenschluß von über zwanzig Firmen zum MSX Standard, stattgefunden hat.

1.2 DER STANDARD

- Zilog Z80A 8-Bit Mikroprozessor
- Texas Instruments TMS 9918A Video Chip
- General Instruments AY-3-8910 Audiochip
- 32 Kilobyte ROM
- Microsoft Extended BASIC, einprogrammiert im ROM
- 8 Kilobyte RAM Minimum, 16 Kilobyte erwünscht
- 40 Zeichen pro Zeile Bildschirmdarstellung
- 16 verschiedene Farben
- Schnittstelle für Kassettenspeicher
- Erweiterungsschacht für RAM-Speichermodul
- Genormter Schacht für Softwaremodule
- Zwei Joystickeingänge
- Centronics-Parallel-Schnittstelle für Druckeranschluß
- Länderspezifisches Keybord mit fünf Funktionstasten (4fach belegt)
- Vier Cursor-Steuertasten
- Genormte Anschlußbelegung aller Ports
- Monitorausgang

Wir wollen nun kurz auf die Funktionen der wichtigsten Bausteine in diesen Rechnern eingehen:

Der Z80 A Prozessor

Wie Sie sicher wissen, besitzt jeder Computer einen Mikroprozessor, den man als das "Gehirn" des Rechners bezeichnen kann. Diesen IC (Integrierter Schaltkreis) nennt man CPU (Central Prozessng Unit) oder Zentraleinheit. Die CPU führt Maschinenbefehle aus, steuert den Ablauf im Rechner und die extern angeschlossenen Geräte (Peripherie). Die Zentraleinheit ist der wichtigste Baustein in einem Computer.
Die MSX Rechner besitzten einen Z80A Prozessor, der auch in vielen anderen Mikrocomputern Verwendung findet. Der Z80A ist eine sehr leistungsfähige Zentraleinheit, welche über 600 Befehle versteht, die bei den MSX Computern mit sehr hoher

Geschwindigkeit verarbeitet werden. Hersteller dieses Mikroprozessors ist die Firma Zilog aus dem amerikanischen Silicon Valley in Kalifornien.

Der Video Chip TMS 9918A von Texas Instruments

Der Video Chip hat die Aufgabe ein Bild auf den Monitor bzw. Fernseher zu geben. Außerdem ist er für die Grafik verantwortlich. Auch die verschiedenen Spriteebenen und das 16K große Video RAM werden durch ihn verwaltet. Durch seine Existenz wird der Z80A Prozessor entlastet, der bei einigen Rechnern teilweise auch diese Aufgaben miterledigen muß.

Der Sound Chip AY-3-8910 von General Instruments

Dieser Chip ist, wie der Name schon sagt, für den "Sound", also für die Musik zuständig. Der Sound Chip ermöglicht einen Tonumfang von 8 Oktaven. Viele Heimorgeln stellen einen wesentlich geringeren Tonumfang zur Verfügung. Außerdem sind in ihm drei regelbare Tonkanäle eingebaut. Damit ist es verhältnismäßig leicht möglich dreistimmig zu spielen. Weiterhin ist dieser Chip für die Klangfarbe zuständig, d.h. es können Instrumente und Geräusche täuschend echt nachgeahmt werden.

Das Microsoft Extended BASIC

Das MSX BASIC stellt weit über 100 Befehle zur Verfügung. Unter ihnen sind ebenso Befehle, die die Eingabe und das Editieren von Programmen komfortabel gestalten, wie Befehle, die speziell den Sound- oder den Video Chip ansprechen. Auch Befehle zum Laden und Speichern auf Kassette, Diskette bzw. zur Druckeransteuerung sind selbstverständlich vorhanden. Besonders zu erwähnen sind die Möglichkeiten der Interruptsteuerung.
Insgesamt gesehen ist das BASIC gut durchdacht. Dadurch ist es angenehm mit ihm zu arbeiten. Umständliche Befehlserweiterung, wie z.B. beim Commodore 64, sind bei den MSX Rechnern nicht notwendig.

8K RAM minimal, 16K RAM erwünscht

Die kleinsten MSX Rechner stellen ein 8K RAM zur Verfügung. In diesem Standard sind 16K RAM erwünscht. Der Trend geht aber eindeutig zu den 64K Versionen, d.h., bei diesen Rechnern sind, mit dem Video RAM, 80K RAM vorhanden.

1.3 DIE SCHNITTSTELLEN DER MSX RECHNER

Die MSX Computer kommen natürlich genausowenig ohne Peripheriegeräte aus, wie andere Rechner. Aus diesem Grund ist es sinnvoll zu wissen, welche Anschlußmöglichkeiten für welche Geräte diese Rechner zur Verfügung stellen.

Fernseher:

Der MSX Standard ermöglicht den Anschluß von Fernsehgeräten über die Antennenbuchse desselben. Der Eingangsport beim Rechner entspricht dabei einer normalen TV Buchse mit 75 Ohm Leitungswiderstand. Leider ist das Bild eines Fernsehers eher durchschnittlicher Qualität.

Monitor

Standardgemäß besitzen die MSX Computer einen Monitoranschluß, der das Bild ohne, die Umsetzung auf eine Trägerfrequenz, auf den Monitorbildschirm bringt. Die entsprechenden Buchsenanschlüsse 1 bis 8 sind zwar genormt, leider aber nicht der Steckertyp. So ist man teilweise auf Adapter angewiesen, die aber ohne Schwierigkeiten im Fachhandel erhältlich sind.
Zusätzlich sind noch sogenannte RGB Anschlüsse vorhanden. Bei dieser Art von Anschluß wird das Bild in seine drei Grundfarben (Rot, Grün, Blau) zerlegt, und erst in einem dafür vorgesehenen Monitor wieder zusammengesetzt. Außer-

dem wird ein RGB Status- sowie ein Luminanz Signal mit übertragen. Dadurch ist eine sehr gute Bildqualität gewährleistet.

Kassettenrekorder

Zur Speicherung und zum Laden von Programmen auf Kassette ist eine 5 polige Dinbuchse auf der Rückseite der Rechner vorgesehen. Dabei werden die vom Computer kommenden Signale auf Pin 4 gelegt. Der "Pin" ist dabei das Gegenstück zu einem Stift, also einem Kontakt innerhalb einer Buchse. Buchse und Stecker bilden eine zusammensteckbare Einheit. Die Übergabe von Daten vom Kassettenrekorder, auch oft als Datasette oder Datenrekorder bezeichnet, findet an Pin 5 statt. Zum Starten und Stoppen des Rekordermotors kann ein entsprechendes Signal von Pin 6 bzw. 7 gesendet werden. Alle anderen Pins liegen auf Masse und haben damit keine besondere Funktion.

Joystick

Die MSX Rechner stellen zwei Anschlüsse zur Verfügung, die sowohl für Joysticks als auch für Paddles vorgesehen sind. Was viele nicht wissen ist, daß diese Anschlüsse auch für andere Geräte, wie z.B. Relaisansteuerung, verwendet werden können. Allerdings erfordert dies einiges elektronisches Wissen.

Disketten

An die Pins der ROM Cartridge werden die Diskettenlaufwerke mittels eines Interfaces angeschlossen. Die meisten MSX Hersteller bieten solche Geräte bereits an. Nach unserer Einschätzung wird sich aber wohl die Floppy von der Firma Sony durchsetzen. Sie ist ein extrem schnelles und solide gebautes 3 1/2 Zoll Laufwerk. Die 3 1/2 Zoll stehen dabei für die Diskettengröße. Diese Disketten passen bequem in die Hosentasche und sind zudem durch einen besonderen Verschluß staubgeschützt.

Da bei Peripheriegeräten die MSX Norm nicht gültig ist, ist es ratsam, sich vor dem Erwerb entsprechender Geräte, genau zu informieren.
Eine besondere Möglichkeit im Zusammenhang mit der Speicherung und des Ladens von Daten bei MSX Rechnern, soll die Bildplatte in absehbarer Zukunft bilden. Da die Bildplatte in reiner Digitaltechnik arbeitet, können auf ihr Unmengen von Daten gespeichert werden. Außerdem kann der Computer ohne große Probleme die Bildplatte ansteuern. Zur Verdeutlichung der Speicherkapazität:
Auf einer Bildplatte haben etwa 200 Mikrofilme Platz.
Viele Firmen arbeiten an der Entwicklung entsprechender Geräte.

Drucker und Plotter

Zum Anschluß von Druckern und Plottern stellt der MSX Standard eine Centronics Parallel Schnittstelle zur Verfügung. Diese Schnittstelle ist schon seit längerem eine Art Normanschluß für Drucker und Plotter. Dadurch gibt es ein großes breitgefächertes Angebot dieser Geräte.
Die Centronics Schnittstelle hat 36 Pole, wobei die meisten Anschlüsse nicht belegt sind. Die Daten zum Drucker bzw. Plotter liegen an Pin 2 bis 9 an (8 Bit), dazu gibt es einige Anschlüsse für Steuersignale. Der Rest der Pins ist entweder nicht angeschlossen, oder liegt auf Masse.

SPEZIALISTEN UNTER DEN MSX RECHNERN

Der MSX Standard hat ein Grundkonzept festgelegt, an das sich Firmen halten müssen, die sich mit zu den Anbietern von MSX Rechnern zählen. Nicht festgelegt aber ist eine Spezialisierung in einzelnen computerspezifischen Bereichen. So kommt es, daß es Rechner dieses Standards gibt, die noch besondere, außerhalb des Standards liegende Fähigkeiten besitzen.

So bietet die Firma Yamaha einen MSX Computer an, der besonders "musikalisch" ist. Dieser Rechner wird im Rahmen eines CX-5 Computer Musik Systems angeboten. Dabei besteht das Basis Set aus dem MSX Rechner, einem FM Modul SFG-01, dem Minikeyboard YK-10 mit 44 Tasten oder dem normalen Keyboard mit 49 Tasten.
Mit diesem Set besitzt man im Grunde genommen einen Preset Synthesizer mit Begleitautomatik, Sequenzer und Manualteilung. Entscheidend aber ist die angebotene Software, die als ROM Erweiterungsmodul vorliegt und in den Geräteschacht gesteckt werden kann. Dieser Schacht befindet sich allerdings am Boden des Rechners und ist nicht MSX kompatibel. Wer sich also eingehender mit Musik beschäftigen will, ist mit diesem Rechner sehr gut beraten. Er besitzt Fähigkeiten, die selbst einige größere Synthesizer nicht zur Verfügung stellen.

Erwähnenswert ist auch der MSX Computer der Firma Sanyo. Für diesen Computer ist ein Lichtgriffel vorgesehen, der mit entsprechender Software erstaunliche Möglichkeiten eröffnet.
Es können Videobilder von Videorekordern, Fernsehgeräten und Videokameras auf den Monitorschirm gebracht und mit Hilfe des Lichtgriffels manipuliert werden. Mit einiger Fantasie und etwas Geschick können damit fantastische Effekte erzielt werden.

Auch der Sony Hit Bit ist ein Spezialist. Er bietet ein fest eingebautes Adressprogramm mit Notizblock und Terminkalender, welches anwenderfreundlich über Menuesteuerung bedient werden kann.
Wie Sie gesehen haben, gibt es einige interessante Entwicklungen im MSX Computerbereich.

KAPITEL II : G R A F I K MIT DEM TMS9918A

2.1 EINFÜHRUNG

Bevor wir uns mit den speziellen Fähigkeiten der MSX Rechner auf dem Gebiet der Grafik beschäftigen, schauen wir uns an, wie Grafiken grundsätzlich von einem Computer verarbeitet werden.

Wie Sie wissen, besteht das Bild eines Monitors bzw. eines Fernsehers aus mehreren tausend einzelnen Punkten. Der Rechner muß jeden Einzelnen dieser Punkte speichern. Die abzuspeichernden Informationen über einen Bildpunkt sind:

- Zustand des Punktes, also gesetzt oder nicht gesetzt

- Farbe des Punktes

Da das Bild ständig verändert wird, müssen diese Informationen im RAM (veränderbarer Schreib-/Lesespeicher) Ihres Computers gespeichert werden.

Wie werden nun üblicherweise die benötigten Informationen abgespeichert?

Um festzustellen, ob ein Bildpunkt gesetzt oder rückgesetzt ist, ist nur ein Bit nötig. Das Bit ist die kleinste Informationseinheit. Elektronisch entspricht ein Bit den Leitungszuständen AN bzw. AUS. Im Computer werden diese Zustände durch die beiden Ziffern des Binär- bzw. Dualsystems, 0 und 1, dargestellt. für jeden Bildschirmpunkt wird eine 1 abgespeichert, wenn er gesetzt ist, und eine 0, wenn er nicht gesetzt ist.

Würde z.B. ein Fernsehbild mit 640*400 Punkten auf diese Weise abgespeichert, so wären dazu 640*400=256000 Bits nötig.

Außerdem soll noch die Farbinformation abgespeichert werden.

Der Platzbedarf für die Farbe hängt direkt mit der Anzahl der zur Verfügung stehenden Farben zusammen. Jeder verfügbaren Farbe wird ein bestimmter Code zugeordnet. Diese Zahl wird dann für jeden Punkt, zusätzlich zu der Information gesetzt/rückgesetzt, abgespeichert. Da der Platzbedarf mit der Größe des Farbcodes, und damit mit der Anzahl der unterschiedlichen Farben zusammenhängt, ist das Speichern der Farbe platzaufwendiger als das der Punktinformation.
Damit das RAM des Rechners nicht vorrangig durch Farbcodes belegt wird, sind meist Einschränkungen sinnvoll. Dazu werden zwei verschiedene Wege beschritten:

1. Bei einer hohen Punktauflösung wird die Anzahl der darstellbaren Farben drastisch vermindert. Oftmals stehen im höchstauflösenden Modus nur noch zwei Farben zur Verfügung: Eine festgelegte Punktfarbe, und eine Farbe für den Hintergrund. Für detailgenaue Zeichnungen, wie zum Beispiel Schaltpläne, ist diese hochauflösende Bildschirmdarstellung sinnvoll. Werden andererseits mehrere Farben benötigt, so wird die folgende Möglichkeit benutzt.

2. Sollen viele verschiedene Farben dargestellt werden, so ist dies, bei gleichem Speicherbedarf, nur auf Kosten der Auflösung möglich. Das bedeutet, daß in dieser Darstellungsart jeweils eine Anzahl der kleinstmöglichen Punkte (Pixel) zu einem großen Punkt zusammengefaßt werden. Dieser "Großpunkt" kann dann nur geschlossen, also alle ihn bildenden Pixel zusammen, gesetzt oder rückgesetzt werden. Der dadurch eingesparte Platz kann nun für das Abspeichern der Farbe eines solchen Großpunktes benutzt werden.

Auch Mischungen aus den beiden oben genannten Möglichkeiten werden benutzt.

Die besprochenen Prinzipien des Speichern von Grafik werden sowohl bei Textdarstellung, als auch bei hochauflösender Grafikdarstellung verwendet. Auf die, für den jeweiligen Bildschirmmodus, im MSX Rechner verwandten Methoden, werden wir in diesem Kapitel eingehen.

Wenden wir uns jetzt den besonderen Grafikeigenschaften der MSX Rechner zu. Wie wir schon im ersten Kapitel erwähnten, wird die Bilderzeugung durch einen speziell dafür konzipierten Chip vorgenommen, den VDP (Video Display Prozessor). Der durch den MSX Standard festgelegte VDP ist der, von der Firma Texas Instruments hergestellte Videoprozessor TMS9918A. Dieser Prozessor zeichnet sich durch hervorragende Eigenschaften aus.

- Auflösung von 256*192 Bildpunkten auf normalem TV

- 15 unterschiedliche Farben und Transparent

- Darstellung von bis zu 32 Sprites

- Durch Überlagerung von 35 Ebenen 3D Simulation möglich

- Adressierung von eigenständigem Video RAM vorgesehen

- 4 verschiedene Bilddarstellungsmodi, und zwar:

- Text Modus: 40 Zeichen Breite, 256 verschiedene Zeichen

- Grafik 1: 32 Zeichen Breite, Sprites und Farben

- Grafik 2: hochauflösende Grafik mit 256*192 Punkten

- Multicoulor Modus: 64*48 Punkte mit 16 Farben

Die 4 Modi werden wir ausführlich besprechen. Außerdem ist den Sprites ein eigener Abschnitt gewidmet. An dieser Stelle gehen wir noch kurz auf Punkt fünf ein:

Jeder MSX Rechner besitzt zusätzlich zum üblichen RAM, der zum Speichern von Programmen etc. benutzt wird, einen Video RAM, abgekürzt auch VRAM. Dieser VRAM dient ausschließlich dem VDP, um die, für den Bildaufbau notwendigen Daten zu speichern. Der Vorteil dieser Methode ist einerseits, daß kein Byte des "kostbaren" RAM für Grafik verbraucht wird und andererseits, daß ausschließlich für das Speichern der Graf-

fiken 16 Kilobyte (=16*1024 Byte= 16*1024*8 Bit) zur Verfügung stehen. Weiterhin wird die interne Verwaltung des VRAM vom VDP selbständig erledigt, wodurch der Prozessor mehr Zeit für andere Aufgaben hat.
Da der Zugriff (lesen oder schreiben) auf die beiden RAM's unterschiedlich ausgeführt wird, gibt es die speziell für den VRAM zuständigen BASIC Befehle:

>VPOKE< : Schreiben von Werten ins VRAM und
>VPEEK< : Lesen von Werten aus dem VRAM.

Mit diesen beiden Befehlen ist eine direkte Manipulation der Inhalte des VRAM, und damit der aktuellen Bildschirmanzeige, möglich.

Geben Sie ein:

```
SCREEN 0:FOR I=0 TO 255:VPOKE I,I:NEXT
```

Bevor wir die verschiedenen Grafikmodi im Einzelnen behandeln, schauen wir uns anhand eines Beispiels die Fähigkeiten der MSX Rechner auf dem Gebiet der Grafik an:

Erstes Grafikprogramm

```
10 SCREEN 2
20 FOR I=0 TO 255 STEP 2:REM Step variieren
30 'LINE(0,96)-(I,96+95*SIN(I/33)):REM 33 variieren
31 'LINE(0,96)-(I,96+95*SIN(I/33)),I MOD 16:REM 16 variier en
32 'LINE(255-I,96)-(I,96+95*SIN(I/33):REM 33 variiren;I MOD
15 anfügen
33 'LINE (255-I,96-95*COS(I/33))-(I,96+95*SIN(I/33):REM 33
bei COS oder SIN aendern
34 'LINE(255-I,96+95*SIN(I/17))-(I,96+95*SIN(I/34)):REM 34
bzw. 17 bei COS oder SIN aendern
35 'LINE(255-I,191)-(I,96+95*SIN(I/33)):REM 33 aendern
36 'LINE(255,96-95*SIN(I/33))-(I,96+95*SIN(I/33)):REM 33 ae
ndern
37 'LINE(255,96+95*SIN(I/33))-(I,96+95*SIN(I/33)),I MOD 16:
REM 33 aendern
```

```
38 'LINE(128-127*SIN(I/33),96)-(I,96+95*SIN(I/33))
39 'LINE(128-127*SIN(I/33),96-95*SIN(I/33))-(I,96+95*SIN(I/
33))
40 NEXT
50 WAIT &HAA,&H40: REM warten auf CAP aus
```

Dieses Programm beeinhaltet verschiedene Variationen von Zeichnungen. Alle Zeilen sind durch den ' als REM Zeilen geschützt. Wollen Sie nun eine dieser Möglichkeiten ausprobieren, müssen Sie nur das REM Zeichen in dieser Zeile löschen. Die gedrückte CAP Taste läßt die Grafik solange auf dem Bildschirm, bis Sie durch nochmaliges Drücken derselben zurück in den Normalmodus springen.

Eine Spezialität des MSX BASIC sind die, bei Grafik und Sound vorkommenden Makro- oder Unterbefehle, die als Strings dargestellt werden können.

```
10 SCREEN 2
20 S=2
30 A$="c=c;s=s;nr10u10nr10e5r10nd10g5d10e5u10"
40 FOR S=1 TO 39 STEP 2
50 C=SMOD13+3
60 PSET(100-S*1.4,120+S*1,4)70 DRAW A$
80 NEXT
90 WAIT &HAA;&H40
```

2.2 DER TEXTMODUS

Der Textmodus bietet die Darstellung von maximal 40 Zeichen pro Zeile bei 24 Zeilen pro Bildschirm an. 256 verschiedene Zeichen, unter anderem "nicht-englische Buchstaben", stehen zur Verfügung. Damit können, ohne eine Änderung des Zeichensatzes, die deutschen Umlaute auf den Bildschirm gebracht werden. Im Textmodus sind nur zwei Farben wählbar: Die Hintergrundfarbe, die zugleich für den Rand benutzt wird, und die Schriftfarbe.

Wie wird nun der Bildschirmtext intern abgespeichert?

Da es sich um einen Textmodus handelt, wird zunächst jedes der 256 möglichen Zeichen für sich abgespeichert. Ein Zeichen dieser Art besteht aus 8 mal 8 Punkten, wobei im Textmodus in horizontaler Richtung nur 6 Punkte angezeigt werden.

Betrachten wir ein Beispiel:

```
1 ***
2*  *
3*  *
4*  *
5* * *
6* *
7 ** *
8
 12345678
```

Ausgehend von der Matrixdarstellung, wird für einen gesetzten Punkt eine 1 und für einen rückgesetzten Punkt eine 0 verwendet. Die so erhaltene Reihe wird als Binärzahl interpretiert (siehe Kapitel 5).

```
&B01110000
&B10001000
&B10001000
&B10001000
&B10101000
&B10010000
&B01101000
&B00000000
```

Auf diese Weise erhalten wir 8 Bytes. ein Byte ist eine 8-Bit Binärzahl. Ihr Wert liegt zwischen 0 und 255. Damit kann exakt das jeweilige Zeichen definiert werden. Jedes der 256 möglichen Zeichen wird auf diese Art durch 8 Bytes dargestellt. Zum Abspeichern des gesamten Zeichensatzes, sind

somit 8*256=2024 Bytes=2K (Kilobytes) notwendig. Diesen Speicherbereich, der beim MSX System im Video RAM liegt, bezeichnet man als Zeichengenerator.

Folgendes Programm liest den Zeichensatz des MSX BASIC aus:

```
10 SCREEN 0
20 ST=BASE(2)
30 FOR I=ST TO ST+256*8-1
40 PRINT RIGHT$("0000000"+BIN$(VPEEK(I)),8)
50 NEXT
```

Zeile 40 ist das Kernstück des Programms:

>VPEEK(I)< liest ein Byte aus dem VRAM. Durch >BIN$< wird das gelesene Byte in eine Binärzahl umgewandelt, und durch >RIGH$< werden eventuell fehlende Nullen am Anfang ergänzt, so daß die Gesamtlänge des auszugebenen Strings genau 8 Zeichen beträgt. >BASE(2)< in Zeile 20 liefert außerdem die Startadresse des Zeichensatzes.

Beim Ablauf des Programms, sehen Sie den gesamten Zeichensatz quasi in Großformat über den Bildschirm laufen. Dabei steht eine 1 für einen Punkt und eine 0 für keinen Punkt.

Wie Sie sehen, sind die einzelnen Zeichen nicht untereinander getrennt, d.h. auf das letzte Byte eines Zeichens folgt sofort das erste des nächsten Zeichens. Um die einzelnen Zeichen voneinander zu unterscheiden, erhält jedes von ihnen eine Zahl, den sogenannten Zeichencode. Der Zeichencode gibt an, welches Zeichen im Zeichengenerator gemeint ist.

Die Bytes 0 bis 7 entsprechen dem Zeichen 0, die Bytes 8 bis 15 dem Zeichen 2, die Bytes 16 bis 23 dem Zeichen 3 usw..

Allgemein bedeutet das:

Das Zeichen mit dem Code N, ist vom (N*8)ten Byte bis zum (N*8+7)ten Byte des Zeichengenerators abgespeichert. Für die

Numerierung gibt es den weitverbreiteten ASCII Code (American Standard Code for Information Interchange).

Dieser Code legt die Bedeutung der Zeichen mit den Codes 0 bis 127 fest. Der MSX Standard benutzt den ASCII Code mit leichten Abweichungen. Die Zeichem mit den Codes 128 bis 255 sind mit MSX eigenen Grafikzeichen und ausländischen Buchstaben belegt. Doch probieren wir das Gesagte einmal aus. Das folgende Programm gibt nach Eingabe des Codes, das dazugehörige Zeichen aus.

```
 5 CLS
10 SCREEN 1
20 ST=BASE(7)
30 LOCATE 3,3:INPUT" Zeichencode: ";ZC
40 LOCATE 0,6
50 FOR I=ST+ZC*8 TO ST+ZC*8+7
60 PRINT RIGHT$("00000000"+BIN$(VPEEK(I),8)
70 NEXT
80 LOCATE 0,16:PRINT CHR$(ZC)
90 GOTO 30
```

Wie Sie sehen, ist das direkt mit >VPEEK< gelesene Zeichen gleich dem, durch >CHR$(ZC)< erzeugten. >PRINT CHR$< gibt das zum jeweilig angegebenen Code gehörige Zeichen aus. Das Gegenstück zum >CHR$< Befehl, ist die >ASC< Funktion. >ASC(A$)< gibt den Code des Zeichens von A$ an.

Damit ist wohl der Aufbau des Zeichengenerators klar geworden. So, wie es mit der >VPEEK< Funktion möglich ist, den Zeichengenerator auszulesen, ist es auch möglich, ihn mit dem >VPOKE< Befehl zu beschreiben. Das bedeutet, daß der Zeichensatz leicht nach eigenen Wünschen und Bedürfnissen geändert werden kann.

Ändern des Zeichensatzes

Das Zeichen, mit der wohl einfachsten Darstellung, ist das Leerzeichen. Da beim Leerzeichen kein Punkt in der 8*8 Matrix gesetzt ist, wird es durch 8 0 Bytes dargestellt. Dieses Zeichen soll nun verändert werden:
Der ASCII Code vom Leerzeichen (oder Space) ist 32, wie man durch >PRINT ASC(" ")< erhält. Es soll nun der Punkt der oberen linken Ecke des Leerzeichens gesetzt werden. Die oberste Reihe wird durch das erste der 8 Bytes dargestellt, also

```
SCREEN 0
VPOKE BASE(2)+32*8, &B10000000
```

Anstelle von &B10000000 können wir auch den Dezimalwert 128, der mit >VAL("&B10000000")< berechnet wird, setzen. Sofort erhalten wir ein verändertes Leerzeichen, bei dem der linke obere Punkt gesetzt ist. Diese Änderung tritt für alle auf dem Bildschirm vorhandene Leerzeichen in Kraft. Der ursprüngliche Zustand kann durch

```
VPOKE BASE(2)+32*8,0
```

wiederhergestellt werden.

```
10 REM deutsch. Zeichensatz
20 READ ZA,QA: REM Ziel-,Quelladresse
30 IF ZA=256 THEN END
40 BA=BASE(2): REM Basisadresse vom Zeichengenerator
50 FOR I=0 TO 7: REM Schleife zum Matrixauslesen
60 VPOKE (BA+I+ZA*8),VPEEK (BA+I+QA*8)
70 NEXT I:GOTO 20
80 DATA 64,34,35,191,94,38,38,47,42,40
90 DATA 40,41,41,61,36,43,123,154,91,129
100 DATA 62,58,60,59,34,142,39,132,58,153
110 DATA 59,148
120 DATA 0,122: REM z in 0 zwischenspeichern
130 DATA 122,121: REM z wird y
140 DATA 121,0: REM y wird z
```

```
150 DATA 0,90: REM Z in 0 zwischenspeichern
160 DATA 90,89: REM Z wird Y
170 DATA 89,0: REM Y wird Z
180 DATA 47,45: REM / vertauschen mit -
190 DATA 45,225: REM - wird zu ß
200 DATA 0,95: REM - in 0 zwischenspeichern
210 DATA 95,63: REM - mit ? vertauschen
220 DATA 63,0: REM ? wird -
300 DATA 256,256: REM Ende Markierung
```

Programmbeschreibung:

Dieses Programm ändert den Standardzeichensatz des Rechners derartig, daß die gleiche Zeichenkonstellation wie auf einer deutschen Schreibmaschinentastatur zur Verfügung steht. Dabei werden die alten Zeichen mit den neuen Zeichen überschrieben. Da bei den MSX Rechnern auch Sonderzeichen (z.B. deutsche Zeichen: ß,ö,ä usw.) vorhanden sind, ist es lediglig notwendig deren Zeichendefinitionen mit denen der Standardzeichen zu vertauschen.
Der ganze Vorgang der Änderung der Zeichenbelegung ist verhältnismäßig einfach in einem Programm zu realisieren.

Die Zeilen 10 bis 70 dürften von ihrer Funktion her verständlich sein.

In den DATA Zeilen stehen die Codes der verwendeten Zeichen und zwar in der Reihenfolge von links nach rechts betrachtet: Standardzeichen ZA, deutsch. Zeichen QA.

Die Zeile 110 besagt also zum Beispiel:
Überschreibe das Standardzeichen ";" mit dem neuen Zeichen "ö". D.h., wenn jetzt die Taste des ursprünglichen Zeichens gedrückt wird, erhalten wir Anstelle von ";" das"ö". In der gleichen Weise wird in den Zeilen von 80 bis 90 verfahren. Man muß aber darauf achten, daß die Reihenfolge der Zeichenänderung so gestaltet wird, daß nicht ein noch benötigtes Zeichen vorzeitig überschrieben wird.

Interessant wird es in den Zeilen ab 120.

Bis jetzt war es immer ohne große Probleme möglich ein Standardzeichen mit einem neuen zu überschreiben, da das ursprüngliche Zeichen später nicht mehrweiterverwendet werden mußte.
Bei den Buchstaben Z,z,Y,y treten aber nun Schwierigkeiten auf, weil die beiden Tasten auf denen diese liegen vertauscht werden müssen, ohne das ein Zeichen verlorengeht. Die Lösung sieht folgendermaßen aus:

Zeile 120:

In dieser Zeile wird das "z" in Zeichencode 0 zwischengespeichert. Das Zwischenspeichern ist notwendig, damit der Buchstabe nicht überschrieben wird und dadurch verloren geht.

Zeile 130:

Dann wird das "z" mit "y" vertauscht.

Zeile 140:

Anschließend wird das ursprüngliche "y" mit dem "z", welches in Zeichencode 0 zwischengespeichert wurde, vertauscht.
Erst durch diese Operation ist es möglich geworden die Buchstaben z und y ohne Verlust eines von ihnen zu vertauschen. Der gleiche Vorgang wiederholt sich bei den anderen Zeichen.

Programm zur Erstellung eigener Zeichensätze

```
10 REM Zeichensatzgenerator
20 SCREEN 0:COLOR 15,4,4:DEFINT A-Z
30 BA=BASE(PEEK(&HFCAF)*5+2):REM Basisadresse
Zeichentabelle
40 XA=5:YA=7: REM Position des Buchstaben
50 EI=0: REM Eingabegerät festlegen
60 ON STRIG GOSUB 460: REM Interruptsprung
definieren
70 STRIG(EI) ON: REM Interrupt einschalten
80 AU$=" ":EI$=CHR$(1)+CHR$(64+10): REM Ausschalt$,
Einschalt$
90 ON KEY GOSUB 700,800: REM Interruptsprünge
definieren
100 KEY 1,"next":KEY (1) ON
110 KEY 2,"Ende":KEY (2) ON: REM Funktionstasten
belegen; Interrupt zulassen
120 REM
130 REM Zeichen ausgeben
140 REM
200 CLS:LOCATE 4,4:PRINT"Zeichen :";
210 A$=INKEY$:IF A$="" THEN 210:REM Eingabe
erwarten
220 PRINT A$: REM Buchstaben ausgeben
230 BZ=BA+ASC(A$)*8:REM Basisadresse Zeichen
240 FOR I=BZ TO BZ+7
250 BY=VPEEK(I)
260 P=128
270 LOCATE XA,YA+I-BZ
280 FOR J=0 TO 7
290 IF (BY AND P)=0 THEN C$=AU$ ELSE C$=EI$
300 PRINT C$;:REM Zeichenmatrix ausgeben
310 P=P/2
320 NEXT
330 NEXT
340 X=0:Y=0
350 LOCATE XA+X,YA+Y,1
360 R=STICK(EI)
```

```
370 IF R=0 THEN 360: REM Cusorsteuerung per
Joystick/Cursortasten
380 IF R=8 OR R=1 OR R=2 THEN Y=Y-1
390 IF R>3 AND R<7 THEN Y=Y+1
400 IF R>1 AND R<5 THEN X=X+1
410 IF R>5 AND R<=8 THEN X=X-1
420 IF X>7 THEN X=X-1:BEEP
430 IF X<0 THEN X=X+1:BEEP
440 IF Y>7 THEN Y=Y-1:BEEP
450 IF Y<0 THEN Y=Y+1:BEEP
460 GOTO 360
470 REM STRIG Interrupt Routine
480 BY=VPEEK(BZ+Y) XOR 2^(7-X)
490 IF (BY AND 2^(7-X))=0 THEN PRINT AU$ ELSE PRINT
EI$:REM Matrix ändern
500 LOCATE XA+X,YA+Y
510 VPOKE BZ+Y,BY
520 RETURN
700 REM Key 1 Interrupt Routine
710 KEY (1) ON:LOCATE,,0:GOTO 200
800 REM Key 2 Interrupt Routine
810 LOCATE ,,0:REM Cursor aus
820 DEFUSR1=&H139D:REM Sprungadresse für original
Keybelegung
830 X=USR1(1)
840 CLS
850 END
```

Programmbeschreibung:

Dieses Programm ist insofern interessant, da es sowohl im >SCREEN 0< als auch im >SCREEN 1< Modus funktioniert. Bemerkenswert ist außerdem noch, daß es interruptgesteuert ist. Wie Sie wissen, ist Interruptsteuerung bei den MSX Rechnern vorgesehen.

Zeile 30:

Hier wird die Basisadresse der Zeichentabelle abhängig vom jeweiligen Modus berechnet.

Zeile 40:

X bzw. Y-Position des Buchstaben wird bestimmt.

Zeile 50:

In EI werden die Eingabegeräte festgelegt.
EI=0 : Tastatur, d.h. Cursortasten + Leertaste(Feuer)
EI=1 : Joystick 1
EI=2 : Joystick 2

Zeile 60:

Diese Zeile definiert die Sprungadresse für einen Interruptsprung, der vom Feuerknopf ausgelöst wird. Wenn also der Feuerknopf gedrückt wird, läuft das Programm in Zeile 470 weiter.

Zeile 70:

Schaltet Interrupt vom Feuerknopf ein.

Zeile 80:

In AU$ (Ausschalt$) wird ein Leerzeichen gespeichert.

In EI$ (Einschalt$) wird ein Sonderzeichen in Form eines inversen Kreises gespeichert. AU$ wird Anstelle eines nicht gesetzten und EI$ Anstelle eines gesetzten Punktes benutzt.

Zeile 90:

Definiert zwei Sprünge nach Interrupt durch Funktionstasten Key 1 und Key 2.

Zeile 100,110:

Die Funktionstaste 1 wird mit "next" belegt und der Interrupt wird angeschaltet.
Die Funktionstaste 2 wird mit "Ende" belegt. Auch hier wird der Interrupt angeschaltet.

Zeile 200:

Hier beginnt der Programmteil zur Zeicheneingabe. Ansonsten ist diese Zeile leicht verständlich.

Zeile 210:

In dieser Zeile wird das Zeichen in A$ gespeichert was auf der Tastatur gedrückt wurde.

Zeile 220:

Gibt das gedrückte Zeichen auf dem Bildschirm aus.

Zeile 230:

In dieser Zeile wird die Basisadresse des Zeichens (BZ) berechnet. Das Zeichen der Basisadresse berechnet sich aus der Basisadresse (BA) und dem ASCII Code der in A$ enthaltenen Buchstaben mal acht. Die Acht setzt den Zähler auf den Anfang eines jeweiligen Zeichens im Zeichengenerator.

Zeile 240:

Diese Schleife ermöglicht das byteweise Auslesen eines Zeichens aus der Zeichentabelle.

Zeile 250:

Liest eine Reihe von 8 Punkten dieses Zeichens in BY (Bytematrix) ein.

Zeile 260:

Hier beginnt die Routine, die das Zeichen vergrößert auf dem Bildschirm ausgibt.
In P (Potenz) wird der Wert 128 für spätere Verwendung gespeichert.

Zeile 270:

Hier wird die aktuelle Position zur Ausgabe des Zeichens auf dem Bildschirm berechnet.

Zeile 280:

Schleife zum bitweise Lesen eines Bytes.

Zeile 290:

Wenn das Bit gesetzt ist, wird das Sonderzeichen in EI$ ausgegeben, ansonsten ein Leerzeichen (AU$).

Zeile 300:

Gibt das jeweilige Zeichen aus.

Zeile 310:

P, dessen Wert immer dem aktuellen Bit entspricht, wird halbiert. So entsteht eine Zahlenreihe, die folgendermaßen aussieht:
128,64,32,16,8,4,2,1
Diese Reihe entspricht dem Wert der Bits von links nach rechts gelesen.

Zeile 350:

Diese Zeile setzt den Cursor auf die aktuelle Position innerhalb des großen, auf dem Bildschirm ausgegebenen Zeichens und schaltet ihn ein.

Zeile 360 bis 460:

Hier wird die Joystickbewegung und Abfrage realisiert. Durch dies Routine kann man sich mit dem Joystick innerhalb des "Großzeichens" frei bewegen. Hier beginnt das eigentliche Hauptprogramm.

Zeile 470,520:

Hier wird bei einem durch Feuerknopf/Leertaste ausgelösten Interrupt hingesprungen. In BY wird nun in der "Großmatrix" durch Knopfdruck entweder ein Zeichen gesetzt oder gelöscht, jenachdem, wie der vorherige Zustand war.

Zeile 500:

Setzt den Cursor wieder auf die oben geänderte Position innerhalb des "Großzeichens".

Zeile 510:

Schreibt das geänderte Bit (=1 Punkt) in die Zeichentabelle.

Zeile 700,710:

Einsprung für durch KEY 1 ausgelöste Interrupts. Bewirkt die Auswahl und den Bildschirmausdruck des nächsten Zeichens.

Zeile 800 bis 850:

Routine, die durch Drücken von Key 2 über Interrupt angesprungen wird. Schaltet den Cursor aus und stellt die Orginalkeybelegung wieder her.
In den folgenden Zeilen wird das Programm beendet.

Zeile 830

Ruft die Routine zur Standardkeybelegung auf.

Nach dem Starten des Programms, legen Sie das zu ändernde Zeichen durch Drücken der dazugehörigen Taste fest. Danach wird das Zeichen in Großformat ausgegeben und der Cursor erscheint auf dem Bildschirm. Nun können Sie durch die Cursortasten bzw. durch den Joystick den Cursor innerhalb der 8*8 Matrix bewegen. Durch das Betätigen der Leertaste bzw. des Feuerknopfes des Joysticks wird, der zum jeweiligen Zeitpunkt unter dem Cursor gelegene Punkt invertiert, d.h. gesetzt, wenn er nicht gesetzt ist und umgekehrt. Die vorgenommene Änderung tritt sofort in Kraft, sowohl in der vergrößerten Darstellung, als auch im Originalzeichen. Mit der Funktionstaste "F1" können Sie weitere Zeichen ändern, mit F2 das Programm beenden. Beachten Sie bei der Änderung von Buchstaben, daß es im Textmodus im Gegensatz zum Grafikmodus 1, nur 6 Punkt breite Zeichen gibt.
Damit haben wir den Zeichensatz umfassend besprochen.

Steuerzeichen

Folgende Besonderheiten bedürfen jetzt noch einer Erwähnung. Die Zeichen mit den Codes von 0 bis 31 können nicht direkt durch einen einzigen >CHR$< Befehl erzeugt werden. Das ist notwendig, da die Codes, die kleiner als 32 sind, normalerweise als Steuerzeichen interpretiert werden. Auch diese Steuerzeichen sind durch den ASCII Code genormt. Alle Steuerzeichen können über die CTRL Taste in Kombination mit bestimmten anderen Taste erzeugt werden. Außerdem sind für wichtige Steuerzeichen eigene Tasten vorhanden.
Mit dem folgenden Programm können Sie die Codes der CTRL+Taste Kombinationen herausfinden:

```
10 A$=INKEY$:IF A$="" THEN 10
20 PRINT ASC(A$):GOTO 10
```

Es gilt folgende Zuordnung:

Code	CTRL+ Taste	Funktion
0	Klammeraffe	-
1	A	Ausgabe von Grafikzeichen
2	B	Wortsprung rückwärts (BACK)
3	C	Abbruch der Eingabe (AUTO ausschalten)
4	D	-
5	E	Löscht alles bis Ende der Zeile
6	F	Wortsprung vorwärts (FORWARD)
7	G	Piepton
8	H	Backspace (BS-Taste)
9	I	Tabulatorsprung (TAB-Taste)
10	J	Zeilenvorschub (Line Feed)
11	K	Cursor Home (HOME-Taste)
12	L	Bildschirm löschen (Shift-HOME-Taste)
13	M	Wagenrücklauf "CR"(Return-Taste)
14	N	Sprung ans Ende der Zeile
15	O	-
16	P	-
17	Q	-
18	R	Inest Modus an/aus (Inest-Taste)
19	S	-
20	T	-
21	U	Zeile komplett löschen
22	V	-
23	W	-
24	X	Select-Taste
25	Y	-
26	Z	-
27	eckige Klammer	Escape (ESC-Taste) (Klammer auf)
28	Û:(Shift)	Cursor links (Cursor Taste)
29	eckige Klammer	Cursor rechts (Cursor Taste)
30	^ Welle	Cursor hoch (Cursor Taste)
31	unterstreichen	Cursor runter (Cursor Taste)
127		Delete Taste

Die Steuerzeichen können also mit CTRL bzw. teilweise mit den Spezialtasten, aber auch durch >PRINT CHR$(...)< erzeugt werden, z.B. durch >PRINT CHR$(7)<. Für Codes kleiner als 32 werden so durch >CHR$< keine Zeichen auf dem Bildschirm ausgegeben. Da diese Zeichen trotzdem vorhanden sind, gibt es einen anderen Weg, sie auf den Bildschirm zu bringen.
Hier ein Beispiel:

Das Zeichen mit Code N (N ist kleiner als 32!) wird durch

```
PRINT CHR$(1);CHR$(64+N)
```

ausgegeben.

Hier ist >CHR$(1)< also ein Steuerzeichen, was dem Rechner mitteilt, daß der nachfolgende Code eines der unteren 32 Zeichen ist. Interessant in diesem Zusammenhang ist, daß das Zeichen mit Code 0 im Zeichensatz nicht belegt, also gleich dem Leerzeichen ist.

Außer dem Code 0 gibt es noch einen zweiten besonderen:

Der Code 255

```
10 SCREEN 0: WIDTH 36
20 LOCATE 0,5
30 FOR I=5 TO 20
40 PRINT STRING$(36,255)
50 NEXT
60 LOCATE 0,0:PRINT"Das Cursorzeichen hat den Code 255"
70 FOR I=0 TO 34:LOCATE I,0,1
80 FOR W=0 TO 150: NEXT W
90 NEXT
```

Die Erklärung des eben gesehenen ist folgende:
Der Cursor wird durch das Zeichen mit dem Code 255 dargestellt. Das Betriebssystem ändert ständig, je nach dem, welches Zeichen sich ursprünglich auf der Cursorposition befand, die Definition des 255sten Zeichens. Sie entspricht in der inversen Darstellung des jeweiligen Zeichens. Durch das Invertieren, werden gesetzte Punkte rückgesetzt und umgekehrt. Die Darstellung der inversen Zeichen werden wir genau im Abschnitt über den Grafikmodus 1 behandeln.

Was ist die Namenstabelle?

Um das komplette Bild zu erzeugen, reicht das Abspeichern des Zeichensatzes allein nicht aus. Es muß weiterhin festgelegt werden, welches Zeichen an welcher Stelle auf dem Bildschirm erscheinen soll.

Diese Aufgabe bewältigt die sogenannte Namens- oder Mustertabelle. Alle möglichen Zeichenpositionen auf dem Bildschirm werden dazu durchnumeriert. Man beginnt in der linken oberen Ecke, und numeriert Zeichen für Zeichen, bis die rechte untere Ecke des Bildschirmes erreicht ist. Durch dieses Verfahren ist jede Bildschirmposition eindeutig bestimmt.
Da das Bild in Modus 1 aus 40 Spalten und 24 Zeilen aufgebaut ist, muß die Namenstabelle 40*24=960 Bytes lang sein. Byte 0 der Namenstabelle enthält den Code des Zeichens, welches an der oberen linken Bildschirmecke steht. Byte 1 enthält den

Code des nächsten Zeichens usw.. Demnach ist der Code des ersten Zeichens der 2ten Reihe im 40ten Byte der Tabelle enthalten. Bei >SCREEN 0< bekommen wir die Startadresse der Namenstabelle mit >BASE(0)<. Im Standardfall ergibt das die Adresse 0.
Damit kann z.B. die Befehlsfolge

```
LOCATE X,Y: PRINT CHR$(64);
```

ersetzt werden durch:

```
VPOKE X+40*Y,64
```

Das gilt nur bei >WIDTH 40<.

Mit Hilfe des >VPOKE< Befehls, können oft Bildschirmausgaben mit höherer Geschwindigkeit ausgeführt werden. Intern werden sämtliche >Print< Befehle letztendlich als >VPOKE< Befehle, die ihrerseits wiederum aus I/0 Befehlen (siehe Kapitel 3) bestehen, ausgeführt.
Fassen wir noch einmal zusammen:

SCREEN 0	
Startadresse der Namenstabelle	BASE(0)=0
Länge der Tabelle	40*24=960 Byte
Startadresse des Zeichengenerators	BASE(2)=&H800
Länge der Tabelle	256*8= 2048 Bytes

SPEICHERAUFTEILUNG

Namenstabelle	&H0000 - &H03BF (knapp 1K)
frei	&H03C0 - &H07FF (über 1K)
Zeichengenerator	&H0800 - &H0FFF (2K)
frei	&H1000 - &H3FFF (12K !!)

Wie Sie sehen, ist weniger als ein Viertel des 16K VRAM genutzt. Um diese brachliegende Kapazität nutzen zu können, beschäftigen wir uns mit den Registern des VDP.

Register des VDP

Der TMS 9918A besitzt 8 Register, 7 Schreibregister (nicht lesbar !) und 1 Leseregister (nicht beschreibbar !).
Ein Register können wir uns, genau wie eine einzelne Speicherstelle, als einen Speicher für ein bzw. mehrere Bytes vorstellen. Eine solche Speicherstelle, die fest einem bestimmten Baustein zugeordnet ist, bezeichnet man oft als Register. Ein Register "registriert" für den jeweiligen Baustein wichtige Informationen.

Der VDP hat die Aufgabe, das Fernsehbild zu erzeugen. Dazu sind z.B. Angaben über die Startadresse des Zeichengenerators und die Startadresse der Namenstabelle notwendig. Befinden wir uns im Textmodus, so reichen diese und die Informationen über die aktuellen Farben aus, um das Bild zu erzeugen.
Damit das korrekte Bild vom VDP produziert wird, müssen unter anderem diese Informationen weitergegeben werden. Sämtliche Register des VDP sind 8-Bit Register, wobei oft nicht alle 8 Bits benutzt sind.

Die Namenstabelle wird in Register 2 abgespeichert. Dazu wird sie zunächst codiert. Die reale Speicheradresse der Namenstabelle im Textmodus, erhalten wir durch >BASE(0)<. Sie ist standardmäßig 0.

(Für den Fall, daß Sie sich nicht mit dem hexadezimalen und dem binären Zahlensystem auskennen, lesen Sie zunächst im Kapitel 5 über die Zahlensysteme nach.)

Grundsätzlich sind Adressen des Videorams im Bereich von 0 bis &H3FFF gespeichert. Um &H3FFF, also die größte VRAM Adresse, abzuspeichern sind 14 Bit (=2+3*4, WARUM ?) notwendig. Von diesen 14 Bit werden jeweils nur die obersten Bits, die mit dem höchsten Stellenwert, in den Registern abgespeichert.

Im Falle der Namenstabelle, werden nur die obersten 4 Bits abgespeichert, also die Bits von 10 bis 13. Das Bit Nr.10 hat den Wert 2^10=1024. Das bedeutet, daß die Adresse der Namenstabelle in Schritten von einem Kilobyte (1 K=1024 Byte) verlegt werden kann.

Im Folgenden werden Sie eine Möglichkeit kennenlernen, die den freien Platz im VRAM durch eine Änderung der VDP Registers ausnutzt. Dadurch können Sie 14 verschiedene Bildschirme gleichzeitig speichern, und wahlweise zwischen ihnen umschalten.
Da die ursprüngliche Adresse der Namenstabelle 0 ist, liefern >BASE(0)< und auch >VDP(2)< den Wert 0. Der Zeichengenerator belegt die Adressen bis &H0FFF. Der darüberliegende Bereich ist frei. Wir wollen jetzt die Namenstabelle nach Adresse &H2000 verlegen.
Dazu können Sie einfach

```
BASE(0)=&H2000
```

oder, wenn Sie direkt das VDP Register 2 verändern wollen

```
VDP(2)= &H2000/2^9
```

eingeben.
Mittels der Division durch 2^9, werden nur die obersten 4 Bit (10 bis 13) berücksichtigt.
Leider erreichen wir durch diese Befehle zwar einen interessanten, aber nicht den gewünschten Effekt. Auf dem jetzt an-

gezeigten Bild, das unter Umständen sehr chaotisch aussieht, können wir weder den Cursor bewegen, noch irgendwelche Angaben tätigen. Schalten Sie also wieder in den Ausgangszustand:

```
VDP(2)=0    bzw.    BASE(0)=0
```

Lassen Sie sich nicht durch die Tatsache, daß Sie nicht erkennen können, was Sie eingeben, stören. Sämtliche Eingaben, die sie gemacht haben, sind auf dem ursprünglichen Bildschirm gelandet und nicht auf dem, welcher angezeigt wurde.

Hierbei wird deutlich, daß der VDP unabhängig vom BASIC arbeitet. Wenn seine Register verändert werden, dann erzeugt er das entsprechende Bild. In unserem Fall "wußte" das Betriebssystem nicht, daß die Namenstabelle verlegt worden war. Folglich erfolgten sämtliche Angaben weiterhin in die alte Namenstabelle, obwohl diese vom VDP gar nicht benutzt wurde. Üblicherweise teilt der >SCREEN< Befehl die notwendigen Informationen dem Betriebssystem mit. Dabei wird die mit dem >BASE< Befehl eingegebene Adresse benutzt.

Geben Sie also ein:

```
BASE(0)=&H2000
SCREEN 0
```

Nun befindet sich die Namenstabelle an der angegebenen Adresse, wie Sie durch >PRINT HEX&(BASE(0))< nachprüfen können. Diese Methode hat den Nachteil, daß der jeweilge Bildschirmbefehl durch den >SCREEN< Befehl gelöscht wird. Sollen die verschiedenen Bildschirme nebeneinander benutzt werden, so muß es möglich sein umzuschalten, ohne daß der Bildschirminhalt verloren geht.

Die Adresse, an der das Betriebssystem die Adresse der Namenstabelle speichert, ist &HF922.
Setzen wir also zusätzlich diese Adresse auf den neuen Wert, so erhalten wir auch wieder den Cursor auf dem neuen Bild.

Folgendes kleines Programm soll Ihnen die Möglichkeit aufzeigen, mehrere Bildsschirme im Direktmodus, also z.B. beim Programmieren, zu benutzen. Damit können Sie unter anderem verschiedene Teile des Listings auf unterschiedlichen Bildschirmen ablegen und je nach Bedarf hin- und herschalten.

```
9999 REM vorher KEY1,"RUN 10000"+CHR$(13) eingeben
10000 REM Seitenumschalter
10010 A$=INKEY$:IF A$="" THEN 10010
10020 A=VAL(A$)
10030 IF A=0 AND A$<>"0" THEN A=(ASC(A$) AND
&B11011111)-55
10040 IF A>13 OR A<0 THEN 10010
10050 POKE &HF923,(A+2+2*(A<2))*4
10060 VDP(2)=A+2+2*(A<2)
10070 LOCATE 0,20
```

Nach Eingabe des Programms und von

```
KEY1, "RUN 10000"+CHR$(13)
```

können Sie durch Drücken von F1 den gewünschten Bildschirm durch Drücken einer der Tasten 0 bis 9 und A, B, C, oder D wählen, wobei 0 der Standardbildschirm ist. Bei der ersten Benutzung eines Bildschirmes müssen Sie ihn meist noch mit SHIFT+HOME löschen.

Programmerklärung:

Zeile 10010: Holt gedrückte Taste
Zeile 10020: Ermittelt Wert falls Zahl
Zeile 10030: Ermittelt Wert falls Buchstabe
Zeile 10040: Prüft auf Zulässigkeit
Zeile 10050: Schreibt die Adresse der Namenstabelle für das Betriebssystem
&HF922 enthält das Low Byte (=0), und
&HF923 enthält das High Byte der Adresse.

Zuordnung von A zur Adresse:

A	Na.tab.Adr.	A	Na.tab.Adr.
1	&H0400	7	&H2400
2	&H1000	8	&H2800
3	&H1400	9	&H2C00
4	&H1800	10(A)	&H3000
5	&H1C00	11(B)	&H3400
6	&H2000	12(C)	&H3800
		13(D)	&H3C00

Zeile 10060: Video Register 2 mit dem jeweiligen Wert laden. Dabei erzeugt +2*(A<2) eine -2, wenn A<2 ist, wodurch die Zuordnung erreicht wird.

Zeile 10070: Setzt den Cursor an das Ende des Bildschirms.

Natürlich können Sie diese Routine auch in Ihren Programmen verwenden. Sollten die gleichzeitig mit unserem Programm, im Speicher liegenden BASIC Programme, die Zeilen ab 10000 belegen, so kann das Programm auch anders numeriert werden.

Beachten Sie bei der Veränderung der Basisadressen, den Unterschied vom >VDP<- zum >BASE< Befehl:
Der >VDP< Befehl schreibt ausschließlich das jeweilige Register mit dem angegebenen Wert. Die Änderung wird durch den >SCREEN< Befehl zurückgesetzt und wieder in den Ausgangszustand gebracht. Hingegen werden mit dem >BASE< Befehl verursachte Änderungen nicht nur in die Register des Video Chip geschrieben, sondern auch im RAM gespeichert und so beim nächsten >SCREEN< Befehl als Ausgangszustand angenommen. Eine Änderung über den >BASE< Befehl gilt solange, bis sie durch einen weiteren >BASE< Befehl verändert wird. Die Änderungen mit >VDP< werden durch >SCREEN< und auch durch >BASE< überschrieben.

Wie Sie wissen, kann durch >VDP< ein Regisster nicht nur beschrieben, sondern auch gelesen werden. Da das beim TMS9918A aber nicht möglich ist, außer bei Register 8, werden die aktuellen Registerwerte immer im RAM mitgespeichert. Ab Adresse &HF3DF stehen die Registerwerte des Video Chips beginnend mit 0, d.h. für

```
PRINT VDP(N)
```

kann man schreiben

```
PRINT PEEK(&HF3DF+N)
```

Im Textmodus sind noch weitere VDP Register von Bedeutung: In Register 4 werden die 3 MSB Bits (Most Significant Bits= Höchstwertige Bits) der Startadresse des Zeichengenerators abgespeichert. Es besteht also auch die Möglichkeit, zwischen verschiedenen Zeichensätzen oder sogar zwischen den Zeichensätzen und dem Bildschirm hin und her zu schalten. Dieses werden wir noch genau im Kapitel über den Modus Grafik I beschreiben.

Farben im Textmodus

Nun möchten wir näher auf die Farben eingehen:
Die Farben werden im Register 7 abgespeichert. Da es 16 verschiedene Farben gibt, können nur 2 Farben in einem Byte gespeichert werden. Dabei bestimmen die 4 MSB's (Bit 4 bis 7) des Registers 7 die Hintergrundfarbe, und die 4 LSB's (Least Significant Bits=Niedrigwertige Bits, Bit 0 bis 3) die Schriftfarbe.

Im Textmodus ist die Rahmenfarbe immer gleich der Hintergrundfarbe. Damit sind Sie im Textmodus nicht in der Lage, verschiedene Farben auf den Bildschirm zu bringen. Auch eine inverse Darstellung von Zahlen oder Buchstaben ist nicht möglich. Da das aber sehr nützlich für eine übersichtliche Bildgestaltung ist, z.B. bei Menüs, werden wir Ihnen ein Programm vorstellen, das die inverse Darstellung von Zeichen implementiert.

Dazu wird zunächst der Zeichensatz geändert. Die Zeichen mit den Codes von 0 bis 127 bleiben erhalten. Die Codes von 128 bis 255 entsprechen den unteren 128 Zeichen, nur sind sie invertiert. Soll nun ein Zeichen invers dargestellt werden, so wird einfach 128 zu seinem Code hinzuaddiert und dieser Code angegeben. Doch zunächst das Programm, das den Zeichensatz in besprochener Weise manipuliert.

```
10 BA=BASE(2)+128*8
20 FOR I=BA TO BA+128*8-1
30 VPOKE I,VPEEK(I-128*8) XOR 255
40 NEXT
```

Nun können inverse Zeichen durch >VPOKE< ausgegeben werden. Die normale Ausgabe mit

```
LOCATE X,Y:PRINT CHR$(Z);
```

muß ersetzt werden durch

```
VPOKE BASE(0)+Y+X*40,Z+128
```

um das inverse Zeichen zu erhalten. Da dieses Verfahren grundsätzlich zu aufwendig wäre, um ganze Wörter oder gar Sätze zu schreiben, haben wir die folgende Maschinenroutine geschrieben. Mit Hilfe dieser Maschinenroutine, wird eine Ausgabe mit dem >PRINT< Befehl möglich, nur ist alles invers dargestellt.

```
10 REM BASIC Lader Patch Invers
20 CLEAR 200, &HF370
30 FOR I=&HF370 TO &HF37A
40 READ A$
50 POKE I,VAL("&H"+A$)
60 NEXT
70 DATA E1,F1,FE,20,38,02,F6,80,F5,E5,C9
80 POKE &HFDA5,&H70
90 POKE &HFDA6,&HF3
100 REM Aktivieren mit POKE &HFDA4,&HC3
110 REM Ausschalten mit POKE &HFDA4,&HC9
120 REM nicht im Direktmodus benutzen
```

Dieses Programm lädt eine Systemerweiterung, die in den Patch Bereich des MSX Systems geschrieben wird. Durch >POKE &HFEE4,&HE1< wird die inverse Zeichenausgabe eingeschaltet, und durch >POKE &HFEE4,&HC9< ausgeschaltet. Der Zeichensatz muß vor der Benutzung dieses Programms natürlich entsprechend "behandelt" worden sein, d.h. die Codes 128 bis 255 müssen die inversen Matrizen der Zeichen mit den Codes 0 bis 127 enthalten

Für "Freaks" hier das Assemblerlisting des Patches:

```
FDA4 C370F3     JP   &HF370 ; Patch Ausgabe auf Bildschirm

F370 E1         POP  HL ; Rücksprungadresse
F371 F1         POP  AF ; Zeichencode
F372 FE20       CP   &H20 ; Steuerzeichen ?
F374 3802       JR   CN &HF378 ; Ja,dann nicht umwandeln
```

```
F376 F680       OR   &H80 ; Code=Code+128
F378 F5         PUSH AF ; Zeichencode auf Stapel
F379 E5         PUSH HL ; Rücksprungadresse auf Stapel
F37A C9         RET     ; Weiter im ROM Programm
```

Hier ein Beispiel:

```
10 POKE &HFDA4,&HC3:PRINT"Invers"
20 PRINT "Noch immer invers"
30 POKE &HFDA4,&HC9:PRINT"Jetzt normal"
40 PRINT "bleibt normal, bis POKE &HFEE4,&HC3 erfolgt"
```

Da wir uns gerade mit einer anspruchsvollen Bildschirmgestaltung beschäftigen, wollen wir noch ein wenig bei diesem Thema bleiben.

"Windows" mit MSX

Obwohl das MSX System keine Windows vorsieht, können sie recht gut erzeugt werden. Ein Window (engl.= Fenster) ist ein fest definierter Ausschnitt aus dem gesamten Bildschirm. Auf das Window kann unabhängig vom restlichen Bildschirm ausgegeben werden. Es kann jede beliebige rechteckige Form annehmen.

Wie kann nun eine Art Window für den MSX Rechner definiert werden?

Zuerst könnte natürlich der >WIDTH< Befehl benutzt werden. Er legt die Breite des Ausgabebereiches fest. Allerdings wird bei der Ausführung dieses Befehls, automatisch ein CLS (Clear Screen) ausgeführt. Dadurch würden Inhalte anderer Fenster gelöscht werden.
Das automatische Löschen des Bildschirmes nach dem >WIDTH< Befehl kann jedoch mit einem bestimmten >POKE< Befehl verhindert werden. Die Stelle, an der die aktuelle Bildschirmbreite für das System abgespeichert ist, hat die Adresse &HF3B0. Durch eine Änderung dieser Speicherstelle mit >POKE<, kann die Bildschirmzeilenlänge eingestellt werden, ohne das ein

CLS ausgeführt wird. Damit ist der erste Schritt in Richtung Windowtechnologie getan.

Um den festgelegten Bildschirmbereich weiter einzugrenzem, beschäftigen wir uns mit der Anzeige der Keybelegung auf dem Bildschirm. Die Keybelegungsanzeige kann vom BASIC aus an- und abgeschaltet werden.
Die Zeile, auf der die Angabe erfolgt, ist in Speicherstelle &HF3B1 abgelegt. Der hier gespeicherte Wert, entspricht der letzten, von der Bildschirmausgaberoutine benutzten Zeile, wenn KEY off (aus) ist. Ist KEY on (an), so ist der Wert, die Nummer der Zeile, die unter der letzten benutzten Zeile liegt. Probieren Sie:

```
POKE &HF3B1,15
```

Zunächst ändert sich gar nichts. Wenn Sie jedoch mit dem Cursor in Richtung des untereh Bildschirmrandes laufen, so werden Sie feststellen, daß Sie auf der 15ten (Wenn KEY off, sonst auf der 14ten) Zeile "steckenbleiben". Damit ist der jeweils betroffene untere Bildschirmteil geschützt. Auch beim Scrolling bewegt sich nur noch der obere Teil des Bildschirms. Sie können auf diese Weise, die letzte Zeile eines Windows festlegen.
Doch das ist noch nicht alles. Es gibt noch eine weitere Speicherstelle, die das Bildschirmformat maßgeblich beeinflußt. Diese Speicherstelle hat die Adresse &HF3DE. Vordergründig wird dieses Byte als Flag (engl.= Flagge, Anzeigen) benutzt, um festzuhalten ob die Keyanzeige an- oder ausgeschaltet ist.

```
KEY OFF:PRINT PEEK(&HF3DE)
```

ergibt 0 als Wert und

```
KEY ON:PRINT PEEK(&HF3DE)
```

ergibt 255

Wird diese Stelle direkt durch >POKE< auf 0 bzw. auf 255 gesetzt, tritt der jeweilige Zustand erst nach Eingabe von CLS ein.
Mit dem >POKE< Befehl, ist man jedoch auch in der Lage, andere Werte als 0 und 255 abzuspeichern. Probieren Sie:

```
CLS:POKE &HF3B1,20:KEY ON
```
und dann
```
POKE &HF3DE,250
```

Wenn Sie jezt mit dem Cursor nach unten laufen, werden Sie nicht weiter, als bis zur 5ten Zeile über die Keyausgabe gelangen. Geben Sie hingegen

```
POKE &HF3DE,2
```

ein, so können Sie bis zu zwei Zeilen unter die Zeile der Keyausgabe laufen.
Also gilt:

&HF3B1- Zeile der Keyausgabe
&HF3DE- Key on/off Flag Differenz im Zweierkomplement zur Zeile der Keyausgabe

Die effektiv letzte Bildschirmzeile, erhalten wir durch

```
PRINT PEEK(&HF3B1)+PEEK(&HF3DE)+256*(PEEK(&HF3DE)>127)
```

Durch geschickte Manipulationen dieser Speicherstellen, ist eine Bildschirmgestaltung mit Windows möglich.

2.3 DER MODUS "GRAFIK I"

In diesem Modus werden im Gegensatz zum Textmodus alle Punkte der 8*8 Matrix des Zeichengenerators angezeigt. Dadurch wird die maximale Anzahl von Zeichen pro Zeile auf 32 verringert. Die Anzahl der Zeilen beträgt weiterhin 24. Zusätzlich sieht dieser Modus jedoch die Benutzung von allen 16 Farben gleichzeitig vor. Auch können die Hintergrund- und die Rahmenfarbe unterschiedlich sein. Ein weiteres wichtiges Merkmal für diesen Modus ist, daß die Benutzung von Sprites vorgesehen ist.

Die Basisadresse des Zeichengenerators erhalten wir für Grafik I durch >BASE(7)<; die Adresse der Namenstabelle mit >BASE(5)<. Diese Tabellen sind analog zu den vom Textmodus (siehe dort) aufgebauten. Die Namenstabelle muß natürlich dem neuen Anzeigeformat angepaßt werden.
Ihre Länge ist also 32*24=768=&H300 Bytes.

Die VDP Register 2 und 4 werden genau wie im Textmodus benutzt.

Wir gehen nun zunächst auf die Behandlung der Farben ein:
Mit dem Befehl >COLOR<, lassen sich Schrift-, Hintergrund- und Rahmenfarbe unabhängig voneinander bestimmen. Das VDP Register 7 speichert die Rahmenfarbe in den 4 unteren Bits. Die Hintergrundfarbe und die Schriftfarbe werden im Grafikmodus I mit Hilfe einer Farbentabelle bestimmt. Die oberen 4 Bit des Registers 7 sind also nicht benutzt.
Die Startadresse der Farbtabelle läßt sich mit >BASE(6)< ermitteln. Die obersten 8 Bit der Farbtabellenadresse sind im VDP Register Nummer 3 abgespeichert. Damit ist die Farbtabelle in Schritten von &H40=2^7=2^(14-7) zu verlegen.

Wie ist nun die Farbtabelle aufgebaut?

Beim Einschalten des Rechners, erhalten Sie höchstens je eine Schrift- und eine Hintergrundfarbe zur Zeit. Es ist aber möglich, z.B. alle 26 Farben gleichzeitig zu benutzen. Dazu ent-

hält jedes Byte der Farbtabelle Informationen über Hintergrund- und Schriftfarbe. Wie im Textmodus bei der Eintragung ins VDP Register 7, sind die unteren 4 Bit für die Hintergrund- und die oberen 4 Bit für die Schriftfarbe zuständig. Sämtliche möglichen Zeichen des Zeichengenerators, sind in Gruppen zu je acht aufeinanderfolgenden Zeichen eingeteilt. Einer solchen Achtergruppe kann nun durch die Eintragung der entsprechenden Werte in die Farbtabelle, eine eigene, von den anderen Gruppen unabhängige Farbe, gegeben werden. Die Farbe des Zeichens mit den Codes 0 bis 7 wird also durch das erste Byte der Farbtabelle bestimmt. Die Codes 8 bis 15 durch das zweite Byte usw.. Allgemein gilt:
Die Farbe des Zeichens mit dem Code N, wird durch das 1+INT(N/8)te Byte der Farbtabelle bestimmt. Damit ist die Farbtabelle 32 Bytes lang.

Wird die Farbe vom BASIC aus mit dem >COLOR< Befehl festgelegt, so werden alle 32 Bytes der Farbtabelle mit demselben Wert beschrieben z.B. >COLOR 15,4,4<. Mit >VPOKE< besteht die Möglichkeit einzelne 8. Gruppen unterschiedlich zu färben. Als Beispiel betrachten wir die Vorhebung aller Großbuchstaben durch inverse Darstellung.
Die Großbuchstaben belegen die Codes von &H41 (=A) bis &H5A (=Z). Also müssen die Bytes von INT(&H41/8)+1 bis INT(&H5A/8)+1, also die Bytes 9 bis 12 mit den inversen Farbwerten beschrieben werden.

```
FOR I=BASE(6)+8 TO BASE(6)+11: VPOKE I,15+4*16:NEXT
```

Natürlich kann auch hier wieder zwischen mehreren Farbtabellen hin- und hergeschaltet werden.

Die Aufteilung des VRAM nach dem Einschalten ist Folgendermaßen:

```
Zeichengenerator : &H0000 - &H07FF
            frei : &H0800 - &H17FF
     Namenstabelle : &H1800 - &H1AFF
   Sprite Atribut : &H1B00 - &H1B7F
            frei : &H1B80 - &H1FFF
    Farbentabelle : &H2000 - &H201F
            frei : &H2020 - &H37FF
    Sprite Muster : &H3800 - &H3FFF
```

(Die Sprite Tabellen werden in Kapitel 2.6 behandelt.)

Leider ist ein Umschalten mit dem >BASE< Befehl im Gafikmodus nicht ohne weiters möglich:
Die Routine, die den >BASE< Befehl ausführt, hat bei unserem Sony Rechner einen kleinen Fehler. Schreiben Sie zum Test eine zweite Farbtabelle ab &H2040:

```
FOR I=&H2040 TO &H105F:VPOKE I,&H4F:NEXT
```

Schalten Sie zunächst mit dem >VDP< Befehl in die neue Farbtabelle

```
&H2040/&H40=129,also

VDP(3)=129
```

Sie erhalten nun die neuen Farben. Mit

```
VDP (3)=128
```

kann zurückgeschaltet werden. Das selbe Ergebnis sollte durch

```
BASE (6)=&H2040
```

zu erreichen sein. Dies ist jedoch nicht der Fall !! Folgender Trick schafft Abhilfe:

```
DEF USR 9= &H7B
```

Nach dieser Eingabe, können alle >BASE< Befehle, die den Grafikmodus I betreffen, durch das Nachstellen von >X=USR9(1)< richtiggestellt werden. Das korrekte Ergebnis erzielen wir durch

```
BASE(6)=&H2040:X=USR9(1)
```

Das Gleiche gilt auch für die Befehle >BASE(5)=....< bis >BASE(9)=...<.
Die Ausführung der >BASE< Funktion, also in der Form >...=BASE(..)< ist für alle Werte korrekt, und es darf nicht >X=USR9(1)< nachgestellt werden!

Das Umschalten mit >BASE< oder >VDP<, kann jetzt benutzt werden, um verschiedene Bilder, wie beim Textmodus, nebeneinander zu speichern und wahlweise anzuzeigen.

Es sollen zwei verschiedene Bilder, mit unterschiedlichen Farbtabellen und als Besonderheit auch mit verschiedenen Zeichengeneratoren, erzeugt werden. Dadurch können wir wahlweise verschiedene Schrifttypen benutzen.
Überlegen Sie sich einen Speicherbelegungsplan, in dem diese Informationen untergebracht werden können. Beachten Sie dabei, daß

```
-Die Namenstabelle       in &H400 Schritten
-Der Zeichengenerator    in &H800 Schritten
-Die Farbtabellen        in &H040 Schritten
-Die Sprite Attributtab. in &H080 Schritten
-Die Sprite Mustertab.   in &H800 Schritten
```

verlegt werden muß.

Vorschlag:

```
&H0000 - &H0800 Zeichengenerator 1
&H0800 - &H1000 Zeichengenerator 2
&H1000 - &H1300 Namenstabelle 1
&H1300 - &H1320 Farbtabelle 1
```

```
        &H1320 - &H1400
        &H1400 - &H1700 Namenstabelle 2
        &H1700 - &H1720 Farbtabelle 2
        &H1720 - &H1B00 Frei
        &H1B00 - &H1B80 Sprite Attribut
        &H1B80 - &H3800 Frei
        &H3800 - &H4000 Sprite Muster

10000 A$=INKEY$: IF A&="" THEN 10000
10010 A=VAL(A$): IF A>1 THEN 10000
10020 DEFUSR 9=&H7B
10030 BASE(7)=&H800*A: X=USR9(1): REM Zeichengenertor
10040 POKE &HF925,&H8*A: REM High Byte
10050 BASE(5)=&H1000+&H400*A: X=USR9(1): REM Namenstabelle
10060 POKE &HF923,&H10+&H4*A: REM High Byte
10070 BASE(6)=&H1300+&H400*A: X=USR9(1): REM Farbtabelle
```

Das Progamm funktioniert analog zu dem Programm im vorhergehenden Kapitel. Die BASIC Adressen des Zeichengenerators sind

&HF924 (LOW) und &HF925 (HIGH).

An diese Adresse muß die Startadresse des Zeichengenerators zusätzlich geschrieben werden.
Natürlich muß zuvor, der zweite Zeichensatz definiert werden.

Probieren Sie folgendes kleines Programm:

```
1 VDP(2)=VDP(2) XOR 1: GOTO 1
```

XOR 1 bewirkt hierbei, daß das Bit 0 des VDP Registers 2 invertiert wird. Dadurch wird ständig zwischen der aktuellen Namenstabelle und der darüberliegenden Namenstabelle (Abstand &H400) umgeschaltet. Allerdings funktioniert dieses Umschalten nicht schnell genug, um eine Überlagerung der beiden Bilder zu erzeugen.

Um dies zu erreichen, müssen wir uns in die interne Interruptroutine "einhängen". Diese Routine wird 50 mal in der Se-

kunde aufgerufen. Sie ermöglicht u.a. die Programmierung von Interrupts im BASIC.
Damit wir unsere eigenen Routinen einbauen können, benutzen wir den zweiten Patch der Interruptroutine. Seine Adresse ist &HFD9F:

```
FD9F C360F3      JP    F360 ; Patch Interrupt
F360 F5          PUSH  AF ; Akku retten
F361 21F3E1      LD    HL,&HF3E1 ; Adresse VDP Register 2
Inhalt
F364 7E          LD    A,(HL) ; Wert VDP Register 2 lesen
F365 EE01        XOR   01 ; Bit invertieren,d.h. umschalten
F367 47          LD    B,A ; neuer Wert
F368 DE02        LD    C,2 ; soll nach Register 2 geladen
werden
F36A CD6047      CALL  &H9947 ; führt VDP(C)=B am
F36D F1          POP   AF ; Akku holen
F36E C9          RET   ; weiter im ROM
```

Der BASIC Lader dieses Programms ist:

```
10 REM Interrupt Bildumschalter
20 CLEAR 200,&HF300
30 FOR I=&HF360 TO &HF36E
40 READ A$:POKE I,VAL("&H"+A$):NEXT
50 DATA F5,21,E1,F3,7E,EE,01,47
60 DATA 0E,02,CD,47,00,F1,C9
70 POKE &HFDA0,&H60
80 POKE &HFDA1,&HF3
90 REM Aktiv mit POKE &HFD9F,&HC3
100 REM Aus mit POKE &HFD9F,&HC9
```

Durch dieses Programm erfolgt eine Umschaltung der Namenstabelle. Natürlich können auch alle anderen Tabellen umgeschaltet werden. Dazu sind folgende Änderungen nötig:

Die jeweilige Registernummer muß als 2ter Code in Zeile 60 eingegeben werden (dort 02).
Der 2te bis 4te Code von Zeile 50 stellt den Maschinenbefehl

LD HL,&HF3E1 dar. Das ist die RAM Adresse, an der der aktuelle Wert des VDP Register 2 gespeichert ist. Es ist notwendig, alle Inhalte der Schreibregister des VDP zusätzlich im RAM zu speichern, da die Register des VDP nicht direkt gelesen werden können. Geben Sie also mit

```
PRINT VDP(2)
```

den Wert des Registers 2 aus, so wird dieser Inhalt intern durch Lesen der Speicherstelle &HF3E1 ermittelt. Entsprechend dazu, ist der Wert des Registers 0 an Adresse &HF3DF und allgemein der Wert des Registers N an Adresse (&HF3DF+N) abgespeichert.

Daraus folgt, daß die 3te und 4te Hexzahl der Zeile 50 entsprechend der Registeradresse geändert wird. Dabei wird zuerst das Low Byte, in unserem Fall also &HE1 und dann das High Byte &HF3 abgespeichert.

Durch geschickte Programmierung, können Sie erreichen, daß z.B. verschiedene Schriftformen gleichzeitig auf dem Bildschirm erscheinen, u.a. kursiv oder Fettschrift. Auch ein Unterstreichen von Worten ist auf diese Weise möglich.
Die gleiche Methode läßt sich außerdem dazu benutzen, mehr als 32 (nämlich 64!) verschiedene Sprites gleichzeitig darzustellen.
Nun aber vorerst genug mit der "Hin- und Herschalterei!" Wir möchten Ihnen zum Abschluß des Kapitels ein Programm vorstellen, mit dem Sie eine Hardcopy des aktuellen Bildschirminhaltes erzeugen können:

```
20000 REM SUB Texthardcopy
20010 DEFINT A-Z
20020 SN=PEEK(&HFCAF)
20030 IF SN>1 THEN RETURN
20040 BA=BASE(SN*5)
20050 AS=40-8*SN
20060 FOR Z=0 TO 23
20070 FOR S=0 TO AS-1
20080 BY=VPEEK(BA+Z*AS+S)
```

```
20090 IF BY>31 THEN LPRINT CHR$(BY);
20100 NEXT
20110 LPRINT
20120 NEXT
20130 RETURN
```

Das Besondere an diesem Programm ist, daß es sowohl in >SCREEN 0<, als auch in >SCREEN 1< läuft. Das Programm stellt automatisch fest, welcher Modus eingeschaltet ist. Dazu wird die Speicherstelle &HFCAF abgefragt. Sie enthält die Nummer des zur Zeit eingeschalten Bildschirmodus.
Wir haben extra dieses Programm als Unterprogramm ausgelegt, damit Sie es sofort in Ihre eigenen Programme einbauen können.
In Zeile 20 wird die aktuelle >SCREEN< Nummer mit >PEEK(&HFCAF)< gewonnen. Das ist nicht die einzige Möglichkeit, den Modus festzustellen. Da der VDP das Bild erzeugt, muß auch in den VDP Registern die Modus Kennziffer abgespeichert sein.
Unter Anderem werden für diese Aufgabe die VDP Register 0 und 1 benutzt. Sie werden auch als Kommandoregister bezeichnet. Für die Auswahl des Modus sind Bit 3 und Bit 4 von Register 1 und Bit 6 von Register 0 zuständig. Die Bits 3 und 4 werden auch als M1 und M2 und das Bit 6 von Register 0 wird auch als M3 bezeichnet.

So gilt dann folgendes:

	M1	M2	M3
Textmodus	1	0	0
GrafikI	0	0	0
GrafikII	0	0	1
Multicoulor	0	1	0

M1 : Bit 3 von Register 1
M2 : Bit 4 von Register 1
M3 : Bit 6 von Register 0

2.3 DER GRAFIKMODUS II

Der Grafikmodus II bietet die höchstmögliche Auflösung von 256*192 Punkten. Dabei können alle 16 Farben gleichzeitig und noch zusätzlich Sprites benutzt werden. Prinzipiell ist der Grafik II Modus dem Grafik I Modus ähnlich. Es ist jedoch beim Grafikmodus II ein größerer Zeichengenerator vorgesehen, so daß für jede der 768 (=32*24) Zeichenpositionen auf dem Bildschirm ein eigenes, von allen anderen Zeichen zu unterscheidenes, Zeichen definiert werden kann. Das bedeutet, daß jeder einzelne der 49152 Bildpunkte, gesetzt bzw. rückgesetzt werden kann. Außerdem besteht die Möglichkeit, für ein 8*8 Zeichen, unterschiedliche Farben zu bestimmen. Für jedes Byte (=8 Punkte) dieser aus 8 Bytes bestehenden Zeichen, können zwei ausgewählte Farben bestimmt werden. Damit sind der Zeichengenerator und die Farbtabelle &H1800 Bytes lang.

VRAM Aufteilung

Wie auch im Grafikmodus I, besteht die Namenstabelle im Grafikmodus II aus 768 Eintragungen, die genau den 768 Zeichenpositionen des Bildschirms entsprechen. Da die Grafik I Modus Musternamen genau 8 Bit lang sind, können maximal 256 verschiedene Zeichendefinitionen in der, im letzten Kapitel besprochenen Weise, adressiert werden.

Im Grafikmodus II, sollen nun jedoch 768 verschiedene Zeichendefinitionen adressiert werden. Dazu wird der Bildschirm in drei gleiche Teile mit je 256 Zeichenpositionen unterteilt. In derselben Weise, werden auch die Namens-, Farb- und Zeichentabellen unterteilt. Dann entsprechen jeweils das obere Drittel der Tabellen, dem oberen Drittel des Bildschirms; das mittlere Drittel der Tabellen, dem mittlerem Teil des Bildschirms, und entsprechend wird bei dem unteren Drittel verfahren. Mit Hilfe dieses Tricks, ist es nun möglich, 768 verschiedene Zeichen darzusstellen. Der Nachteil dieses Verfahrens liegt darin, daß die Drittel voneinander unabhängig sind. Es kann also eine Zeichendefinition des ersten Drittels nicht direkt in den anderen Dritteln des

Bildschirms dargestellt werden. Wie wir jedoch sehen werden, spielt das in den meisten Fällen keine besonders große Rolle.

Betrachten wir ein Beispiel:

In der Namenstabelle steht an 300ster Stelle, also im zweiten Drittel, der Wert 10. Folglich wird auf der 300sten Bildposition (Zeile 9, Spalte 12) das Zeichen mit dem Code 10, d.h. das 10te Zeichen des mittleren Drittels (!) des Zeichengenerators angezeigt. Die Farben des Zeichens werden durch die Bytes 10*8 bis 10*8+7 des zweiten Drittels der Farbtabelle bestimmt.

	NAMENSTAB.	ZEICHENGENERATOR	FARBTABELLE	BILDSCHIRM
1	0 : 256	0 : 2047	0 : 2047	0 Zeile : : : 7 Zeile
2	256 : 300=(10) : : 511	2048 : -->2047+10*8 bis 2047+10*8+7 : 4095	2048 : ->2047+10*8 bis 2047+10*8+7 : 4095	8 Zeile : auf Zeile 9 Spalte 12 wird ent- sprechendes Zeichen in
3	512 : 767	4096 : 6143	4095 : 6143	entsprech- enden Far- ben ange- zeigt

Zu je 8 Punkten, also einem Byte, des Zeichengenrators gehören 2 Farben (2*4 Bit=1Byte) der Farbtabelle. Dabei bestimmt der Wert der unteren 4 Bit, die Farbe eines gesetzten Punktes, der Wert der oberen 4 Bit, die Farbe eines rückgesetzten Punktes.
Schauen Sie sich nun das folgende Programm an, das die Zusammenhänge verdeutlicht:

```
10 SCREEN 2
20 READ X,Y:REM Position
30 NT=BASE(10): REM Namenstabelle
40 FT=BASE(11): REM Farbtabelle
50 ZG=BASE(12): REM Zeichengenerator
60 BN=(32*Y+X)*8: REM Bytenummer in Farb- und Zeichentab.
70 FOR I=0 TO 7
80 READ B,VF,HF: REM Byte,Vordergrundfarbe,Hintergrundfarbe
90 VPOKE ZG+BN+I,B: REM Zeichendefinition
100 VPOKE FT+BN+I,HF*16+VF: REM Farben
110 NEXT I
120 N=(32*Y+X) MOD 256: REM Zeichenname
130 FOR I=NT TO NT+767
140 VPOKE I,N
150 NEXT I
160 IF INKEY$="" THEN 160
170 DATA 9,12: REM Position
180 DATA &B00010000,3,15
190 DATA &B00101000,3,15
200 DATA &B01000100,3,15
210 DATA &B10010010,3,13
220 DATA &B01000100,3,15
230 DATA &B00101000,3,15
240 DATA &B00010000,3,15
250 DATA &B00010000,11,3
```

Die Werte der Datazeilen 170-250, können Sie beliebig ändern. Beobachten Sie was passiert. Zunächst wird durch die Schleife von Zeile 70 bis Zeile 110 die Zeichendefinition ins >VRAM< übertragen. Beachten Sie, daß hierbei nicht die Namenstabelle benutzt werden muß. Sobald der Modus >SCREEN 2< eingeschaltet wird, numeriert die entsprechende Systemroutine jedes Drittel der Namenstabelle von 0 bis 255 durch.
Von diesem Zeitpunkt an, bleibt die Namenstabelle unverändert. Das bedeutet, daß alle Änderungen am Bildschirm, direkt im Zeichengenerator vorgenommen werden. Das ist der grundlegende Unterschied vom Grafik II Modus zum Text oder Grafik I Modus. Bei letzterem wird im Normalfall mit einem

festen Zeichensatz und einer festen Farbtabelle gearbeitet. Lediglich die Namenstabelle wird ständig geändert.

Im Grafik II Modus wird nun meist die Namenstabelle unverändert belassen und sämtliche Änderungen im Zeichengenerator und in der Farbtabelle selbst durchgeführt. Das eine Änderung der Namenstabelle möglich ist, und wie sie sich auswirkt, zeigen die Zeilen 120 bis 150 des obigen Programms. Die gesamte Namenstabelle wird mit dem Namen des vorher definierten Zeichens gefüllt. Dadurch bleiben das obere und untere Drittel leer, da die Definition nur für das mittlere Drittel gilt. Dieses wird hingegen vollständig mit dem eben definierten Zeichen gefüllt.
Probieren Sie folgendes Programm:

```
5 REM Zerfetzer
10 SCREEN 2
20 NT=BASE(10): REM Namenstabelle
30 S=7:GOSUB 210: REM Zeichnen
40 GOSUB 170: REM Durcheinander
50 S=20:GOSUB 210
60 GOSUB 120: REM Nummerieren
70 S=11:GOSUB 210
80 GOSUB 170
90 GOSUB 120
100 IF INKEY$="" THEN 100
110 END
120 FOR J=0 TO 2
130 FOR I=0 TO 255
140 VPOKE NT+J*256+I,I
150 NEXT I,J
160 RETURN
170 FOR I=NT TO NT+767
180 VPOKE I,RND(1)*256
190 NEXT I
200 RETURN
210 FOR I=10 TO 80 STEP S
220 CIRCLE (128,96),I
230 NEXT
240 RETURN
```

Überlegen Sie sich anhand der Abläufe auf dem Bildschirm, was intern geschieht. Wie Sie sehen, gehen die BASIC Grafik Befehlsroutinen immer von einer ordnungsgemäß numerierten Namenstabelle aus. Die Reihenfolge der Namenstabelle wird nie verändert. Machen wir das, wie z.B. mit der Routine ab Zeile 170, so wird zunächst das bestehende Bild durcheinandergewürfelt. Wird dann ohne neu zu numerieren weitergezeichnet, so entstehen keinesfalls Kreise (Zeile 50). Erst nach erneutem Numerieren (Zeile 60) wird das Bild richtig sichtbar.
Es ist also nicht ratsam etwas an der Namenstabelle zu ändern, wenn BASIC Routinen benutzt werden sollen. Alle Änderungen sollten direkt im Zeichengenerator und in der Farbtabelle durchgeführt werden.

Die Standardeinteilung des VRAM im hochauflösenden Modus ist:

&H0 - &H1800	Zeichengenerator
&H1800 - &H1B00	Namenstabelle
&H1B00 - &H1B80	Sprite Attribut
&H1B80 - &H2000	Frei
&H2000 - &H3800	Farbtabelle
&H3800 - &H4000	Sprite Muster

Die Namenstabellenbasisadresse wird wie üblich durch das VDP Register 2 bestimmt. Für die Zeichengenerator- und Farbtabellenbasisadresse gilt abweichend folgende Regelung:
Da beide Tabellen 6K groß sind, können sie in 8K Schritten verschoben werden. D.h., sie beginnen entweder an Adresse 0 oder an Adresse &H2000. Zur Unterscheidung zwischen diesen beiden Möglichkeiten wird jeweils das höchste benutzte Bit verwendet. 1 bedeutet dabei ab Adresse &H2000 und 0 ab Adresse 0. Bei der Farbtabelle ist das Bit Nr.7 von Register 3, beim Zeichengenerator Bit Nr.7 von Register 4. Alle niederwertigen Bits müssen außerdem auf 1 gesetzt sein, ansonsten treten sehr merkwürdige Effekte auf, die nur in den seltensten Fällen von Nutzen sind.
Mit der Standardeinteilung ist der VRAM fast vollständig genutzt. Es ist jedoch möglich z.B. zwei Namens- oder sogar mehrere Attributtabellen unterzubringen. Wird auf die Benut-

zung von Sprites verzichtet, so erhöht sich die Zahl entsprechend. Eine Verlegung ist, wie gehabt, durch Änderung der VDP Register mit dem >VDP< Befehl oder aber durch den >BASE< Befehl möglich. Leider ist auch hier wieder ein Fehler zu finden. Damit >BASE< bei eingeschaltetem >SCREEN 2< korrekt ausgeführt wird, muß sofort danach die Systemroutine ab Adresse &H007E aufgerufen werden. Sie setzt die VDP Register auf die richtigen Werte. Also:

```
DEF USR8=&H7E
BASE(...)=....: X=USR8(1)
```

Die Benutzung dieses Umweges ist nur notwendig, wenn >SCREEN 2< eingeschaltet ist, und der aktuelle Bildschirminhalt nicht verändert werden soll. Wird hierauf keinen Wert gelegt, so ist auch

```
BASE(..)=.....:SREEN 2 möglich.
```

Die Adressen des BASIC Interpreters für Namenstabelle und Zeichengenerator sind &HF922/3 und &HF924/5.
Damit Sie die Grafikfähigkeiten Ihres Rechners noch besser verstehen und nutzen lernen, stellen wir Ihnen im Folgenden einige Anwenderprogramme vor.

Grafikeditor

Zunächst folgt ein Programm, mit dem Sie ohne Programmieraufwand direkt am Bildschirm Grafiken erstellen können. Die Benutzung des Programms ist nach dem Maus - Konzept aufgebaut. Das bedeutet, daß Sie ständig ein Symbol, bei diesem Programm unter anderem einen Pfeil, mit Hilfe des Joystick oder der Cursortasten auf dem Bildschirm bewegen. Sämtliche Funktionen des Programms werden aufgerufen, indem Sie das Symbol auf eine markierte Stelle setzen und den Feuerknopf (bzw. Leertaste) drücken. Am unteren Rand des Bildschirms sehen Sie alle 16 Farben. Wählen Sie die aktuelle Schriftfarbe mit dem Pfeil aus. Beim Drücken des Feuerknopfes wird die Farbe gewählt, auf die die Spitze des Pfeils zeigt.

Neben der Farbskala befinden sich die Kürzel Cu. und und Ra. Sie sind in der jeweils aktuellen Farbe angezeigt. Durch das Anwählen von Cu. in der bekannten Weise ("hinlaufen, feuern!"), erhalten Sie anstelle des Pfeils ein C. Durch wählen einer Farbe mit dem C (=Cursor), nimmt der dann wieder erscheinende Pfeil die gewählte Farbe an. Bei der Wahl des Kürzels Ra., erhalten Sie ein R als Symbol für die Rahmenfarbe und können die Rahmenfarbe einstellen.

Am oberen Ende des Bildschirms erhalten Sie folgende Ausgabe:

Pun/Lin/Rec/Pai/Tex/

Pun- Punktmodus

Dieser wird durch den Pfeil gekennzeichnet. Befinden Sie sich im Zeichenfeld, so wird im Punktmodus beim Drücken des Feuerknopfes ein Punkt in der aktuellen Farbe gesetzt.

Lin - Linienmodus

Gehen Sie mit dem Symbol auf Lin und betätigen den Feuerknopf, so erhalten Sie als Symbol einen Pfeil mit einem Loch in der Spitze. Damit können Linien gezeichnet werden. Markieren Sie durch Druck auf den Feuerknopf den Anfangs- und den Endpunkt der Linie. Beim zweiten Knopfdruck werden die beiden Punkte durch eine Linie verbunden. Die einzelnen Modi bleiben solange eingeschaltet, bis ein neuer Modus gewählt wird.

Rec - Rechteck

Der Rec Modus wird durch das Rechtecksymbol gekennzeichnet. Analog zu Lin werden zwei Punkte markiert. In diesem Modus wird jedoch das Rechteck, welches die gedachte Linie zwischen den Punkten als Diagonale besitzt, gezeichnet.

Pai - Paint

Nach dem Anwählen von Pai erhalten Sie ein ausgefülltes Rechteck. Ein durch die aktuelle Farbe begrenztes Gebiet wird mit der aktuellen Farbe gefüllt. Achten sie darauf, daß die zu färbenden Gebiete wirklich geschlossen sind, da ansonsten der gesamte Bildschirm gefärbt wird.

Tex - Textmodus

Nach der Wahl von Tex erhalten Sie ein T als Cursor, mit dem Buchstaben oder Zeichen ausgegeben werden können. Nach dem Drücken des Feuerknopfes wird auf einen Tastendruck einer normalen Taste gewartet. Das jeweilig dazugehörende Zeichen wird dann an der aktuellen Position ausgegeben.

```
10 DEFINT A-Z
20 EI=1:REM JOY
30 CC=1:REM SPRITE CURSOR COLOR
40 OPEN "grp:" FOR OUTPUT AS #1
50 SCREEN 2,0:COLOR 15,4,4
60 FOR I=0 TO 2
70 READ CH$:CH=ASC(CH$):BA=&H1BBF+8*CH
80 FOR J=0 TO 7
90 A$=A$+CHR$(PEEK(BA+J)):NEXT J
100 READ N:SPRITE$(N)=A$:A$=""
110 NEXT I
120 FOR I=0 TO 7:READ A:A$=A$+CHR$(A):NEXT
130 SPRITE$(0)=A$
140 A$=CHR$(ASC(A$))+CHR$(&H48)+CHR$(&H48)+RIGHT$(A$,5)
150 SPRITE$(3)=A$
160 SPRITE$(4)=CHR$(&H7F)+STRING$(6,&H41)+CHR$(&H7F)
170 SPRITE$(5)=STRING$(8,&H7F)
180 X=120:Y=80:XA=X:YA=Y
190 READ XL,XR,YO,YU
200 READ JO,JU
210 LINE (XL,YO+JO)-(XR,YU-JU),2,B
220 FOR I=0 TO 15:LINE (16+I*8,YU-1)-(22+I*8,YU+7),I,BF:
NEXT
230 SF=15:HF=4:GOSUB 780
240 ON STRIG GOSUB 370,370,370,370,370:STRIG(EI) ON
250 PSET (16,YO):PRINT#1,"Pun/Lin/Rec/Pai/Tex/"
260 PUT SPRITE 0,(X,Y),CC,M
270 RI=STICK(EI):IF RI=0 THEN 270
280 IF RI>1 AND RI<5 THEN X=X+XP:XP=XP+2 ELSE XP=1
290 IF RI>3 AND RI<7 THEN Y=Y+YP:YP=YP+2 ELSE YP=1
300 IF RI>5 AND RI<9 THEN X=X-XM:XM=XM+2 ELSE XM=1
310 IF RI=1 OR RI=2 OR RI=8  THEN Y=Y-YM:YM=YM+2 ELSE YM
=1
320 IF X>XR THEN X=XR
330 IF X<XL THEN X=XL
340 IF Y>YU THEN Y=YU
350 IF Y<YO THEN Y=YO
360 GOTO 260
```

```
370 REM strig
380 YM=0:YP=0:XM=0:XP=0
390 IF Y>YO+JO AND Y<YU-JU THEN 610
400 IF Y<=YO+JO THEN 520
410 PO=INT((X-16)/8)
420 IF PO<0 THEN RETURN
430 IF PO>15 THEN 490
440 IF M<>1 AND M<>2 THEN SF=PO:GOSUB 780:RETURN
450 IF M=1 THEN COLOR ,,PO
460 IF M=2 THEN CC=PO
470 GOSUB 780
480 M=MA:RETURN
490 IF PO=16 THEN RETURN
500 IF PO<23 THEN MA=M:M=INT((PO-14)/3):RETURN
510 RETURN
520 REM obere Befehlsreihe
530 PO=INT((X-16)/32)
540 NF=0:RF=0:LF=0:PF=0:TF=0
550 IF PO<1 THEN NF=-1:M=0:RETURN
560 IF PO<2 THEN LF=-1:M=3:RETURN
570 IF PO<3 THEN RF=-1:M=4:RETURN
580 IF PO<4 THEN PF=-1:M=5:RETURN
590 IF PO<5 THEN TF=-1:M=6:RETURN
600 RETURN
610 REM Punktsetzen
620 IF M=1 OR M=2 THEN RETURN
630 IF NF THEN 700
640 IF TF THEN GOSUB 720:RETURN
650 IF LF=1 THEN LINE (XA,YA)-(X,Y),SF:LF=-1:RETURN
660 IF LF=-1 THEN XA=X:YA=Y:LF=1
670 IF RF=1 THEN LINE (XA,YA)-(X,Y),SF,B:RF=-1:RETURN
680 IF RF=-1 THEN XA=X:YA=Y:RF=1
690 IF PF THEN PAINT (X,Y),SF:RETURN
700 PSET (X,Y),SF
710 RETURN
720 REM Textaus
730 POKE &HF3F8,PEEK(&HF3FA):POKE &HF3F9,PEEK(&HF3FB)
740 PSET(X,Y),0
```

```
750 A$=INKEY$:IF A$="" THEN 750
760 PRINT #1,A$;
770 RETURN
780 REM Ausgabe shbc
790 LINE (150,184)-(198,192),HF,BF
800 COLOR SF,HF
810 PSET (152,184),SF:PRINT#1,"Ra.Cu."
820 RETURN
830 DATA R,1,C,2,T,6
840 DATA &B01111100
850 DATA &B01111000
860 DATA &B01111000
870 DATA &B01111100
880 DATA &B01001110
890 DATA &B00000111
900 DATA &B00000011
910 DATA &B00000000
920 DATA 0,249,0,185
930 DATA 10,4
```

Programmbeschreibung

Zeile 40:

Grafikausgabe: Eröffnet Daten für die Ausgabe von Text auf dem Grafikbildschirm.

Zeile 60:

Schleife zur Definition von drei Sprites die Buchstaben darstellen.

Zeile 70:

Das Zeichen wird in CH$ gelesen und der dazugehörige ASCII Code wird in CH gespeichert. BA enthält die Basisadresse des aktuellen Zeichens im ROM Zeichengenerator (Startadresse &H1BBF).

Zeile 80:

Schleife zum Auslesen von 8 Bytes pro Zeichen.

Zeile 90:

Umwandeln der Zeichendefinition in einen String (A$) für eine Spritedefinition.

Zeile 100:

Spritenummer lesen, Sprite definieren und A$ löschen.

Zeile 120,130:

Einen Pfeil als Sprite definieren.

Zeile 140,150:

Erzeugt den Pfeil mit einem Loch in der Spitze, der "Lin" anzeigt.

Zeile 160:

Erzeugt Das leere Quadrat welches "Rec" anzeigt.

Zeile 170:

Erzeugt das volle Quadrat für "Pai".

Zeile 180:

X,Z: Startposition des Sprites
XA,YA: alte Position zum Zeichnen von Linie/Rechteck.

Zeile 190:

Liest Bildschirmbegrenzung : XL=links, XR=rechts, YO=oben, YU=unten

Zeile 200:

Differenzen vom Gesamtbildschirm zum Zeichenfeld.
JO=oben, JU=unten

Zeile 210:

Zeichnet den Rahmen.

Zeile 220:

Zeichnet die farbigen Quadrate am unteren Bildschirmrand.

Zeile 230:

SF=Schriftfarbe und HF=Hintergrundfarbe werden auf ihre Standardwerte gesetzt.

Zeile 240:

Interruptdefinition für Feuerknopf oder Leertaste und Zulassen des Interrupts.

Zeile 260:

M= aktuelle Spritenummer
CC=Cursorfarbe
Anzeigen des jeweiligen Cursorsprites.

Zeile 270 bis 360:

Steuerung des Cursorsprites, wobei XP,YP,XM und YM die Geschwindigkeit für die jeweilige Bewegungsrichtung verändern. Die Geschwindigkeit steigt bei längerer Beibehaltung einer Steuerrichtung.

Zeile 370:

Interruptroutine zur Behandlung des "Strig".

Zeile 380:

Geschwindigkeit des Cursorsprites auf 0 zurücksetzen.

Zeile 390:

Abfrage, ob der Cursorsprite im Zeichenfeld ist.

Zeile 400:

Abfrage, ob der Cursorsprite oberhalb des Zeichenfeldes ist, dann zur Befehlsreiheninterpretation verzweigen.

Zeile 410:

Position in der unteren Reihe feststellen.

Zeile 420:

Wenn links von den Farbfeldern dann ungültig.

Zeile 430:

Wenn rechts von den Farbfeldern weiter in 490.

Zeile 440:

Setzen der Schriftfarbe durch Cursorsprite. (SF=PO ,Schriftfarbe=Farbfeld). Sprung zur Ausgabe shbc.

Zeile 450:

Rahmenfarbe setzen

Zeile 460:

Cursorspritefarbe setzen

Zeile 470:

Zur Farbausgabe springen

Zeile 480:

Alten Sprite aufrufen

Zeile 490:

Wenn ungültige Position im unteren Befehlsfeld, dann passiert nichts.

Zeile 500:

Befehlserkennung für "Cu und RA".

Zeile 530:

Position innerhalb der oberen Befehlsreihe feststellen.

Zeile 540:

NF= Normalmodus; LF= Linienmodus; RF= Rechteckmodus; PF= Paintmodus; TF= Textmodus

Zeile 550:

Wenn der Spritecursor auf Pun steht dann Punktsetzen.

Zeile 560 bis 590:

Wenn auf "Lin" dann Spritenummer 3 aufrufen ; Modusflag setzen.

Zeile 620:

Sprites, die nicht erlaubt sind ausschließen (C und R).

Zeile 630:

Bei NF=-1 nach 700 gehen. Wenn man im Normalmodus ist, wird sofort ein Punkt gesetzt.

Zeile 640:

Wenn man im Textmodus ist, dann nach 720 springen.

Zeile 650:

Zweiten Punkt der Linie X,Y mit erstem Punkt XA,XY verbinden.

Zeile 660:

Registriert den ersten Punkt und setzt LF auf 1 damit beim nächsten Mal die Verbindung hergestellt wird.

Zeile 670,680:

Gleiche Technik wie in den Zeilen 650,660.

Zeile 690:

Fläche färben

Zeile 700:

Punkt setzen

Zeile 720:

Textausgabe

Zeile 730:

Zeichenpuffer löschen

Zeile 740:

aktuelle Position festlegen

Zeile 750:

Tastaturabfrage

Zeile 760:

Ausgabe per Datei auf dem Bildschirm

Zeile 780:

Ausgaben von Schrift und Hintergrundfarbe und Rahmenfarbe

Zeile 800:

Farbe setzen

Zeile 810:

Die unteren Befehlsworte "Ra.Cu." ausgeben.

Zeile 830 bis 930:

Spritedefinition für den Pfeil

3D GRAFIK

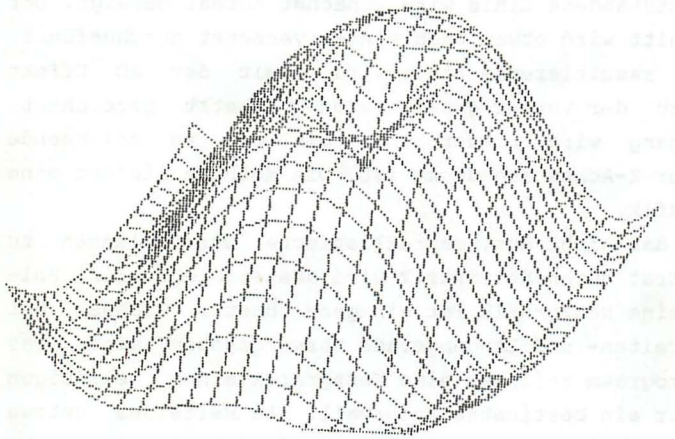
Eine der schönsten und spannendsten Anwendungen der Grafik ist das Darstellen von dreidimensionalen Körpern oder Funktionen. Leider dauert das Erstellen dieser Bilder bei vielen Programmen oft Stunden oder sogar Tage.

Das folgende Programm benötigt in der Grundversion nur wenige Minuten, um ein komplettes Bild zu zeichnen.
Der Darstellung von 3D Funktionen liegt die Idee zugrunde, sie mit Hilfe von Schnitten auf eine zweidimensionale Darstellung zurückzuführen. Die 3D Funktion wird in regelmäßigen Abständen entlang der XY-Achse zerschnitten. Die beim Schneiden entstandene Linie wird zunächst normal gezeigt. Der nächste Schnitt wird etwas nach hinten versetzt durchgeführt. Die daraus resultierende Linie wird, damit der 3D Effekt entsteht, zu der vorher gezeichneten versetzt gezeichnet. Dieser Vorgang wird wiederholt, bis der zu zeichnende Abschnitt der Z-Achse abgedeckt ist. Die Methode liefert eine 3D Liniengrafik.
Oft reicht das nicht aus, um realistische Darstellungen zu erzielen. Erst durch Netzgrafik wird das erreicht. Ein Beispiel für eine Netzgrafik ist ein gezeichneter Globus, bei dem die Breiten- und Längengerade hervorgehoben sind. Das folgende Programm zeichnet eine Netzgrafik einer beliebigen Funktion für ein bestimmtes Intervall. Die Weite des Netzes bestimmt maßgeblich die Zeichengeschwindigkeit. Ein sehr grobes Netz wird schnell erstellt, erzeugt aber auch nur ein genähertes Bild der Funktion. Das liegt in der Methode der Netzgrafik begründet:

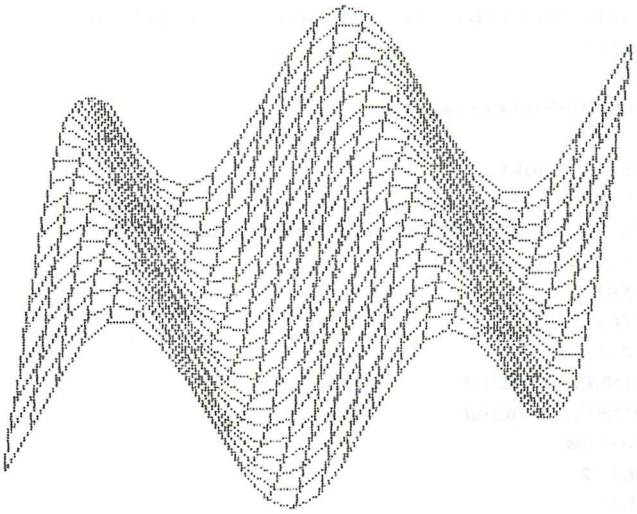
Bei der Netzgrafik werden nicht alle Punkte der Kurve berechnet, daher der immense Geschwindigkeitsvorteil gegenüber der herkömmlichen Zeichenmethode. Vielmehr werden jeweils nur die Knotenpunkte des Netzes berechnet und dann durch Linien verbunden. Prinzipiell können beliebige Funktionen gezeichnet werden. Es muß jedoch meist der Wertebereich, d.h. die Grenzen auf den Achsen zwischen denen gezeichnet werden soll,

entsprechend korrigiert werden, um vernünftige Bilder zu erhalten.

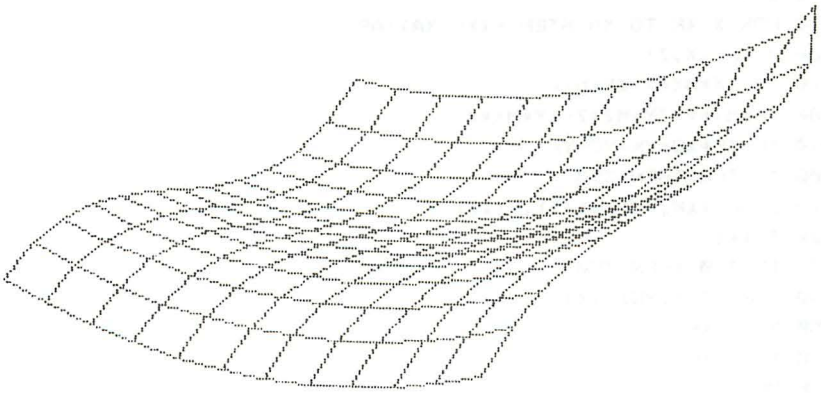
Bei vielen Funktionen wird es sich als störend erweisen, daß die gesamte Funktion, also inklusive der der eigentlich verdeckten Bereiche, gezeichnet wird. In der Realität sehen wir ja z.B. auch immer nur, die uns zugewandte Seite eines Globus, nicht aber die Rückseite, es sei denn, er ist aus Glas gebaut. D.h., Sie erhalten ein Bild der aus Glas nachgebauten und mit dem Netz überzogenen Funktion. Oft kann diese Überlagerung durch ein Ändern der Blickwinkel zumindest teilweise beseitigt werden. Eine grundsätzliche Lösung des Hinterschneidungsproblems ist jedoch mit unseren Mitteln nicht möglich.



```
40 DEFFNF(X,Z)=SIN(SQR(X*X+Z*Z))
90 DATA -5.5,-1.9,1.9,-2.4
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM WINKEL X-ACHSE
140 WZ=52:REM WINKEL Z-ACHSE
```



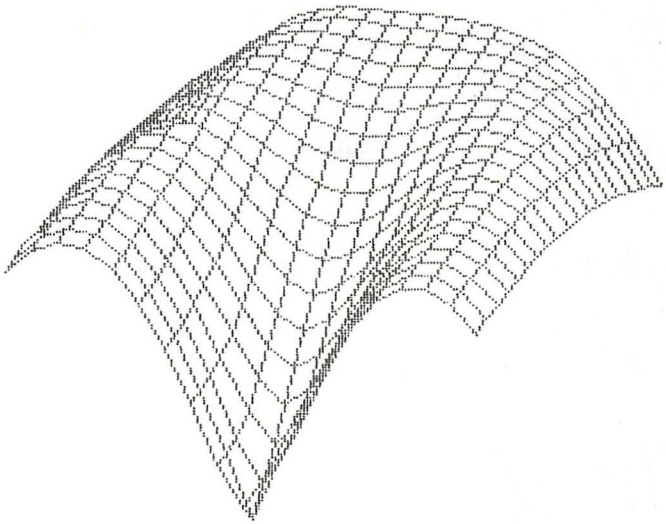
```
40 DEFFNF(X,Z)=SIN(X-Z)
90 DATA -5,5,-1.9,1.9,-2,2
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=12:REM WINKEL X-ACHSE
140 WZ=62:REM WINKEL Z-ACHSE
```



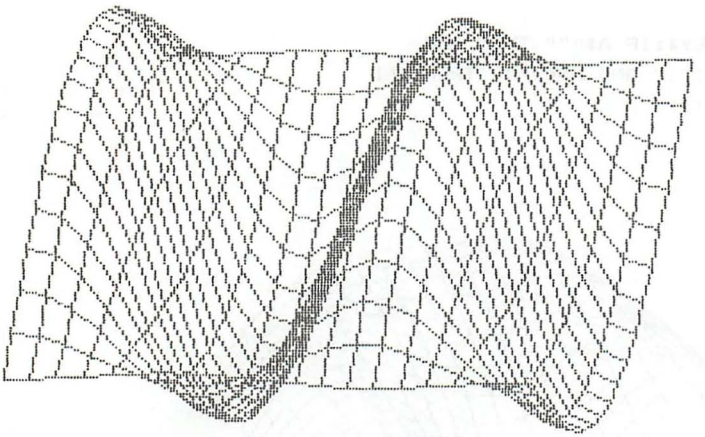
```
40 DEFFNF(X,Z)=(X-Z)^2+2*X-.05*Z*Z*Z+3
90 DATA -400,400,-500000,500000,-170,170
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=12:REM WINKEL X-ACHSE
140 WZ=32:REM WINKEL Z-ACHSE
```

```
10 ' 3-D Netzgrafik Funktionszeichner von Holger Dullin
20 CLEAR 500,&HEFFF
30 DEFUSR1=&HF000
40 DEFFNF(X,Z)=SIN(SQR(X*X+Z*Z))
50 SCREEN 2
60 AP=18:REM Anzahl Punkte
70 DIM X(AP+1),Y(AP+1)
80 READ XA,XE,YA,YE,ZA,ZE:REM Achsenbegrenzungen
90 DATA -2.1,2.1,0,1.3,-1.6,.1
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM WINKEL X-ACHSE
140 WZ=52:REM WINKEL Z-ACHSE
150 DE=3.141529#/180
160 ZX=COS(WZ*DE)^2
170 XX=COS(WX*DE)^2
180 ZY=SIN(WZ*DE)^2
190 XY=SIN(WX*DE)^2
200 XO=XA/(XA-XE)*256
210 YO=YE/(YE-YA)*192
220 J=0
230 FOR Z=ZE TO ZA STEP -(ZE-ZA)/AP
240 I=0
250 FOR X=XE TO XA STEP -(XE-XA)/AP
260 Y=FNF(X,Z)
270 XK=XX*MX*X-ZX*MZ*Z
280 YK=MY*Y-ZY*MZ*Z-XY*MX*X
290 XK=XK+XO:YK=YO-YK
300 IF I=0 THEN 320
310 LINE (XK,YK)-(X(I),Y(I))
320 I=I+1
330 IF J=0 THEN 350
340 LINE (XK,YK)-(X(I),Y(I))
350 X(I)=XK
360 Y(I)=YK
370 NEXT X
380 J=J+1
```

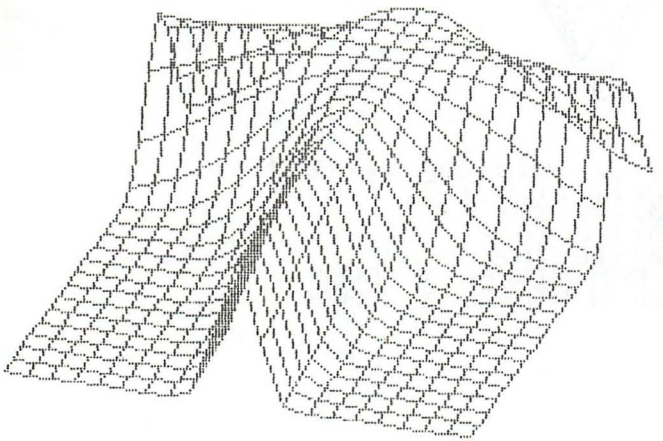
```
390 NEXT Z
400 A$=INKEY$:IF A$="" THEN 400
410 IF A$<>"j" AND A$<>"J" THEN END
420 X=USR1(1)
430 END
```



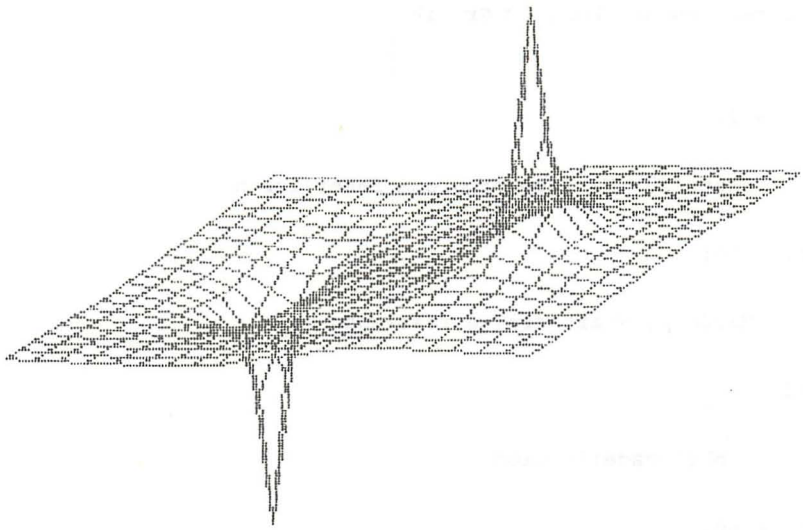
```
40 DEFFNF(X,Z)=SIN(SQR(X*X+Z*Z))
90 DATA -2.1,2.1,0,1.3,-1.6,.1
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM WINKEL X-ACHSE
140 WZ=52:REM WINKEL Z-ACHSE
```



```
40 DEFFNF(X,Z)=SIN(X)*COS(Z)
90 DATA -4.8,4.8,-1,1,-1.6,1.6
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=170/(ZE-ZA)
130 WX=10:REM WINKEL X-ACHSE
140 WZ=62-WX/2:REM WINKEL Z-ACHSE
```



```
40 DEFFNF(X,Z)=EXP(-X*X*Z*Z)
90 DATA -3,3,-1.2,1.2,-.5,3
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM WINKEL X-ACHSE
140 WZ=52:REM WINKEL Z-ACHSE
```



```
40 DEFFNF(X,Z)=X/SQR((1-X*X)^2+
   (2*X*Z/300)^2+1E-03)
90 DATA -2,2,-7,7,-300,300
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=12:REM WINKEL X-ACHSE
140 WZ=42:REM WINKEL Z-ACHSE
```

Programmbeschreibung 3D Grafik

Zeile 20:

Platz für Hardcopyprogramm reservieren

Zeile 30:

Hardcopy Startadresse festlegen

Zeile 40:

Funktionsdefinition

Zeile 60:

AP bestimmt die Anzahl der Netzquadrate, die entlang einer Achse gebildet werden sollen.
Große Werte in AP (bis max.30) bringen ein besseres Bild. dafür dauert die Zeichnung länger.
Kleine AP Werte (bis min.10) liefern sehr schnell einen groben Überblick über den Funktionsverlauf.

Zeile 70:

Reservierung der Feldvariablen, die jeweils die Koordinaten der zuletzt gezeichneten Netzpunkte speichern.

Zeile 80,90:

Hier werden die Achsenbegrenzungen aus der darunterstehenden DATA Zeile eingelesen. Sie bestimmen direkt den Ausschnitt der Funktion, die dargestellt wird.
Kommt kein befriedigendes Bild zustande, so müssen oft diese Werte geändert werden. Falls Sie bei einer unbekannten Funktion keine Vorstellung über die Auswahl dieser Werte haben, benutzen Sie zunächst recht große Ausschnitte.

Zeile 100 bis 120:

Maßstabsfaktor für die Umrechnung in Bildschirmkoordinaten für X-, Y- und Z- Achse. Die angegebenen Zahlen brauchen (und sollten) nur in Ausnahmefällen geändert werden.

Zeile 130:

WX ist der Winkel, den die X-Achse in der Darstellung mit der Waagerechten einschließt. Dieser Winkel ist mitentscheidend für das Aussehen des Bildes.

Zeile 140:

WZ benutzt den Winkel, den die Z-Achse mit der Waagerechten einschließt. Auch dieser Winkel ist sehr entscheidend für das Aussehen des Bildes.

Zeile 150:

Umrechnungfaktor von Grad in Bogenmaß

Zeile 160 bis 190:

In Abhängigkeit des Winkel WZ und WX der Achsen werden hier die Stauchungs- bzw. Streckfaktoren für die einzelnen Achsenrichtungen bestimmt. Sie bewirken, daß Abstände, die in der Realität unter einem spitzen Winkel betrachtet werden in der Zeichnung entsprechend verkürzt dargestellt werden.

Zeile 200,210:

XO und YO sind Summanden, die benutzt werden um die realen Koordinaten in Bildschirmkoordinaten umzurechnen. Dabei wird berücksichtigt, wie die Grenzen gewählt wurden.

Zeile 220:

I=0 bedeutet, daß zur Zeit die erste Linie in X-Richtung gezeichnet wird.

Zeile 230:

Durch die Z-Schleife wird das reale Bild sozusagen in Scheiben geschnitten. Die Linie, die dann beim Schnitt entsteht wird gezeichnet.

Zeile 240:

I ist der Zähler, der die Nummer des aktuellen Netzquadrates enthält.

Zeile 250:

Durch die X-Schleife wird je eine Linie des "Z-Schnitts" gezeichnet.

Zeile 260 bis 290:

Berechnung des Funktionswertes.
Umrechnung von 3D auf 2D.
Umrechnung in die Bildschirmkoordinaten.

Zeile 300:

Wenn Linienanfang (I=0) keine Verbindungslinie zeichnen (waagerecht).

Zeile 310:

waagerechte Verbindungslinie zeichnen

Zeile 330:

Wenn erste Linie, dann keine senkrechte Verbindungslinie zeichnen.

Zeile 340:

Senkrechte Verbindungslinie zeichnen.

Zeile 350,360:

Netzpunktkoordinaten zeichnen.

Zeile 390 bis 430:

Wenn Eingabe von J, Hardcopy (Hardcopyprogramm muß vorher geladen sein !), ansonsten Ende.

Grafik Hardcopy

Damit Sie die mit dem Grafikeditor oder mit dem 3D Funktionsplotter erstellten Zeichnungen auch auf das Papier bringen können, stellen wir Ihnen nun noch ein Hardcopy Programm vor. Das Programm läuft ohne Änderung auf dem EPSON FX-80 Drucker und kompatiblen. Zur Anpassung an andere Druckertypen muß nur die Steuersequenz, die den Drucker auf Grafikmodus schaltet, geändert werden, vorausgesetzt, daß ein 8-Punkt Bitmustermodus existiert.
Für die Hardcopy wird direkt der Zeichengenerator im hochauflösenden Modus ausgelesen. Wie Sie wissen, sind dort je 8 aufeinanderfolgende Bytes ein Zeichen, und die nächsten 8 Bit das nächste Zeichen usw.. Das Problem besteht nun darin, daß der Drucker jeweils 8 untereinander und nicht wie beim Bildschirmspeicher nebeneinanderliegende Punkte druckt.

Beispiel:

Bildschirmspeichercodes

```
00010000    = 16
00101000    = 40
01000100    = 68
```

```
 10111010    =186
 01000100    = 68
 00101000    = 40
 00010000    = 16
 11111110    =254
 !!!!! !!
.!! ! 17!
17!147! !
  !! !! !
 41!85! 0    Druckercode
   !  !
  85  41
```

Das Zeichen wird durch

```
FOR I=32 TO 39: READ A
VPOKE I,A:NEXT
DATA 16,40,68,16,68,40,16,254
```

auf dem Bildschirm ausgegeben. Für die Druckerausgabe müssen die spaltenweise gebildeten Werte gesendet werden.

```
LPRINT CHR$(27);"K";CHR$(8);CHR$(0);
FOR I=0 TO 7:READ A
LPRINT CHR$(A);:NEXT
DATA 17,41,85,147,85,41,17,0
```

Der erste >LPRINT< Befehl ist die Steuercodefolge für den EPSON FX-80, die die Ausgabe von 8 Grafikcodes im Bitmustermodus ankündigt.

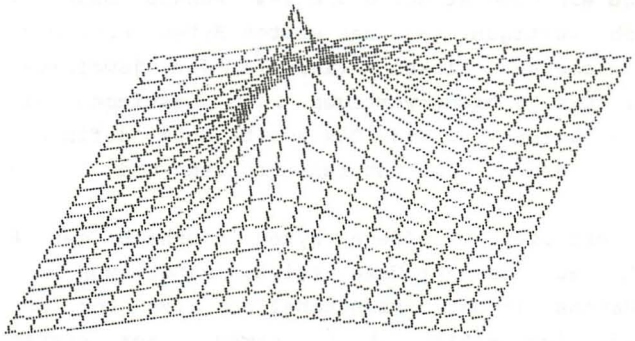
Das Hardcopy Programm muß nun den gesamten Bildschirmzeichensatz Schritt für Schritt nach obiger Vorschrift umwandeln, um die "Spaltenwerte" senden zu können. Da das im BASIC einige Stunden dauern kann, ist hier nur eine Lösung in Maschinensprache sinnvoll.

```
 F000             10                    ; Hardcopy
 F000             20  PRINT   EQU   &H00A5
 F000             30          ORG   &HF000
?F000 210000      40          LD    HL,TABLE1 ; Steuerzeich
entabelle
 F003 7E          50          LD    A,(HL)
 F004 47          60          LD    B,A ; Anzahl Steuerzeic
hen
 F005 23          70  NESTE1  INC   HL
 F006 7E          80          LD    A,(HL)
 F007 CDA500      90          CALL  PRINT
 F00A 10F9       100          DJNZ  NESTE1
 F00C 210000     110          LD    HL,&H0000 ; Namenstabel
le
 F00F 0618       120          LD    B,24 ; Zeilenzaehler
 F011 C5         130  NEZEIL  PUSH  BC
 F012 E5         140          PUSH  HL
?F013 210000     150          LD    HL,TABLE2 ; Steuerzeich
entabelle
 F016 7E         160          LD    A,(HL)
 F017 47         170          LD    B,A ; Anzahl Steuerzeic
hen
 F018 23         180  NESTE2  INC   HL
 F019 7E         190          LD    A,(HL)
 F01A CDA500     200          CALL  PRINT
 F01D 10F9       210          DJNZ  NESTE2
 F01F E1         220          POP   HL
 F020 0620       230          LD    B,32 ; Spaltenzaehler
 F022 C5         240  NESPAL  PUSH  BC
 F023 E5         250          PUSH  HL ; Namenpointer
 F024 CDA50B     260          CALL  &H0BA5 ; Zeichendefinit
ion ins RAM Kopieren
 F027 0608       270          LD    B,8 ; Bytezaehler
 F029 C5         280  NEBYTE  PUSH  BC
 F02A 2118FC     290          LD    HL,&HFC18
 F02D 0608       300          LD    B,8 ; Bytezaehler
 F02F 97         310          SUB   A ; Akku loeschen
 F030 CB06       320  NEBIT   RLC   (HL) ; Bit ins Carry
 F032 17         330          RLA    ; Carry in Akku
 F033 23         340          INC   HL
 F034 10FA       350          DJNZ  NEBIT
 F036 CDA500     360          CALL  PRINT
 F039 C1         370          POP   BC ; Bytezaehler
 F03A 10ED       380          DJNZ  NEBYTE
 F03C E1         390          POP   HL ; Namenstabellenpoin
ter
 F03D 110800     400          LD    DE,8
 F040 19 '       410          ADD   HL,DE
```

```
 F041 C1          420          POP  BC ; Spaltenzaehler
 F042 10DE        430          DJNZ NESPAL
 F044 3E0A        440          LD   A,10 ; Line Feed
 F046 CDA500      450          CALL PRINT
 F049 3E0D        460          LD   A,13
 F04B CDA500      470          CALL PRINT
 F04E C1          480          POP  BC ; Zeilenzaehler
 F04F 10C0        490          DJNZ NEZEIL
 F051 C9          500          RET
 F052             510           ; Steuerzeichtabellen
**** Zeile  40   : TABLE1=&HF052
 F052 03          520  TABLE1 DB   3 ; Laenge der Tabelle
 F053 1B          530          DB   27 ; ESC Code
 F054 41          540          DM   "A" ; Zeilenvorschub
 F055 08          550          DB   8 ; 8/72 inch
**** Zeile  150   : TABLE2=&HF056
 F056 05          560  TABLE2 DB   5 ; Laenge der Folgende
n Sequenz
 F057 1B          570          DB   27 ; ESC Code
 F058 2A          580          DM   "*" ; Bit Image Selecti
on
 F059 04          590          DB   4 ; CRT Graphics
 F05A 0001        600          DW   256 ; 256 Bytes pro Zei
le
------------------------------------

Programm :harcop
Start : &HF000     Ende : &HF05B
Lnge : &H5C Bytes
Fehler : 0
Variablentabelle :
PRINT  00A5    NESTE1 F005
NEZEIL F011    NESTE2 F018
NESPAL F022    NEBYTE F029
NEBIT  F030    TABLE1 F052
TABLE2 F056
```

```
10 CLEAR 200,&HEFFF:MAXFILES=1
20 FOR I=&HF000 TO &HF05B
30 READ A$:W=VAL("&H"+A$)
40 S=S+W:POKE I,W:NEXT
50 IF S<>8670 THEN BEEP:PRINT"Fehler in DATAs":END
60 PRINT"Alles klar !"
70 DEFUSR1=&HF000
80 DATA 21,52,F0,7E,47,23,7E,CD
90 DATA A5,00,10,F9,21,00,00,06
100 DATA 18,C5,E5,21,56,F0,7E,47
110 DATA 23,7E,CD,A5,00,10,F9,E1
120 DATA 06,20,C5,E5,CD,A5,0B,06
130 DATA 08,C5,21,18,FC,06,08,97
140 DATA CB,06,17,23,10,FA,CD,A5
150 DATA 00,C1,10,ED,E1,11,08,00
160 DATA 19,C1,10,DE,3E,0A,CD,A5
170 DATA 00,3E,0D,CD,A5,00,C1,10
180 DATA C0,C9,03,1B,41,08,05,1B
190 DATA 2A,04,00,01
```



```
40 DEFFNF(X,Z)=EXP(-SQR(X*X+Z*Z))
90 DATA -3,3,-0.9,0.9,-2,2
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM WINKEL X-ACHSE
140 WZ=52:REM WINKEL Z-ACHSE
```

DER MULTICOULOR MODUS

Dieser Modus bietet eine Anzeige von 64*48 farbigen Quadraten. Dabei besteht jedes Quadrat aus 4*4 Punkten. Ein Zeichen in >SCREEN 3< besteht also aus 4 Quadraten. Die Farbe jedes dieser Quadrate kann eine der 16 möglichen sein. Damit können alle 16 Farben gleichzeitig benutzt werden. Die Sprites stehen vollständig zur Verfügung.

Die Namenstabelle ist dieselbe, wie bei den beiden Grafik Modi. Sie besteht aus 768 Eintragungen, wobei der Name nicht mehr in eine Farbtabelle, sondern nur noch in den Zeichengenerator zeigt. Die Farbe wird also durch den Zeichengenerator bestimmt. Da üblicherweise eine Zeichendefinition 8 Bytes lang ist, zeigt ein Name auf ein 8 Byte Teilstück des Zeichengenerators.

Nur 2 dieser 8 Bytes bestimmen jedoch jeweils die Bildschirmanzeige. Diese beiden Bytes bestimmen 4 Farben, wobei jede Farbe ein Quadrat der Größe 4*4 Punkte bedeckt. Die 4 MSB's (höherwertigen, 4-7) des ersten Bytes bestimmen die Farbe des oberen linken Viertels der jeweiligen Zeichenposition. Die 4 niederwertigen Bits bestimmen die Farbe des rechten oberen Viertels. Das zweite Byte definiert entsprechend die Farben des unteren rechten und linken Viertels.

Die Positionen der 2ten betroffenen Bytes innerhalb des 8 Bytes Teilstücks auf das der Name zeigt, hängt von der Position des Namens in der Namenstabelle, also von der betroffenen Bildschirmposition ab. Für Namen der ersten Zeile (Namen 0-31) werden die ersten beiden Bytes des 8 Byte-Teilstücks benutzt. Die nächste Zeile von Namen (32-63, entspricht Bildschirmzeile 2) benutzt die Bytes 3 und 4. Die nächste Zeile benutzt die Bytes 5 und 6 und schließlich die letzte Zeile die Bytes 7 und 8. Dieses Schema wird für den gesamten Bildschirm fortgesetzt.

Die Belegung der VDP Register und die Benutzung des >BASE< Befehls ist analog zum bisher gelernten. Ansonsten gibt es zu diesem Modus nicht viel zu sagen. Die Auflösung ist hier viel zu grob, als das mit ihm interessante Grafiken erzeugt werden könnten. Allenfalls in Verbindung mit den Sprites kann der

Multicoulor Modus als Hintergrund dienen. Beschäftigen wir uns also mit den Sprites.

2.6 SPRITES

Die Möglichkeiten der Darstellung von Sprites sind beim MSX System besonders gut. Bis zu 32 Sprites können programmiert werden. Mit Hilfe von Sprites können auf hervorragende Weise Bewegungen auf dem Bildschirm dargestellt werden.
Prinzipiell ist ein Sprite nur ein überdemensionales Zeichen, das über spezielle Befehle auch auf dem Grafikbildschirm angezeigt werden kann. Ein großer Sprite besteht aus 16*16 Punkten. Analog zu den Zeichen wird ein Sprite durch Eintragungen in einer sogenannten Sprite Mustertabelle definiert. Die Position des Sprites entspricht den Koordinaten seiner linken oberen Ecke. Durch das Verändern der Positionskoordinaten kann das Spritmuster über den gesamten Bildschirm bewegt werden. Eine Darstellung von ruhigen, gleitenden Bewegungen, die sonst in hochauflösender Grafik aus Zeitgründen schwierig ist, kann leicht programmiert werden.

Bei MSX-Rechnern sind damit jedoch die Spritemöglichkeiten noch nicht erschöpft. Die Anzeige der Sprites geschieht auf den sogenannten Spriteebenen. Auf jeder der 32 hintereinanderliegenden Ebenen kann ein Sprite angezeigt werden. Das Besondere an dieser Verfahrensweise drückt sich im "Hintereinanderliegenden" aus: Ein Sprite, der auf einer hinteren Ebene angezeigt ist, wird von einem, auf einer davorliegenden Ebene angezeigten überdeckt, wenn sich die Muster überlappen. Dadurch können Darstellungen in pseudo-3D (ohne wirkliche Perspektive) erzeugt werden. Der Vordergrund, wie z.B. bewegliche Figuren, Fahrzeuge etc. wird auf den vorderen Ebenen (=kleine Zahlen) angezeigt. Damit überlappen diese Sprites alle anderen, was der Realität entspricht. Bilder, die in Wirklichkeit weiter vom Beobachter entfernt sind, werden auf weiter hinten liegenden Spriteebenen angezeigt. Auf der hintersten Ebene (=31) werden z.B. oft Wolken o.ä. dargestellt. Hinter der letzten Spriteebene befindet sich die

eigentliche Bildschirmebene. D.h. Sprites überdecken grundsätzlich die Standardbildschirmanzeige.
Die Benutzung von Sprites ist in Mode 1, 2 und 3 identisch. Im Modus 0 (=Textmodus) können keine Sprites benutzt werden. Die Ebene, auf der der Sprite angezeigt werden soll, wird durch den ersten Parameter des >PUT SPRITE< Befehls bestimmt. Sprites können in vier verschiedenen Modifikationen benutzt werden. Es besteht, wie schon erwähnt, die Möglichkeit 16*16 Punkte große Sprites oder 8*8 Punkte große Sprites zu verwenden. Unabhängig von der Punktzahl können Sprites normal (=Originalgröße) und vergrößert (=doppelte Größe) dargestellt werden. Bei Vergrößerung erhöht sich nicht (!) die Anzahl der verschiedenen Punkte. Lediglich wird jeder Punkt als 4mal so groß wir vorher angezeigt. Die Wahl des Anzeigemodus geschieht mit dem zweiten Parameter des >SCREEN< Befehls. Dabei bedeutet:

Parameter	Punkte	Vergrößerung	reale Größe
0	8*8	normal	8*8
1	8*8	vergrößert	16*16
2	16*16	normal	16*16
3	16*16	vergrößert	32*32

Die Definition eines 8*8 Sprites haben wir im Prinzip schon im Kapitel über den Textmodus kennengelernt. Ein Sprite wird als Bitmatrix geschrieben, wobei jede horizontale Linie von je 8 Punkten ein Byte (=8 Bit) darstellt. Der Wert des Bytes ist gleich dem Wert der Binärzahl, die man erhält, wenn man für Punkte "Einsen" und für nichtgesetzte Punkte "Nullen" schreibt.
Damit wird ein 8*8 Sprite (wie ein Zeichen) durch eine Folge von 8 Zahlen charakterisiert. Bei Zeichendefinitionen mußten wir diese Zahlen direkt ins VRAM poken. Bei den Sprites gibt es hierfür einen BASIC Befehl: >SPRITE $(...)=...<.
Vielleicht wundern Sie sich, warum es sich hier aufeinmal um einen String Befehl handelt. Für die interne Verarbeitung sind jedoch Strings und Zahlenketten gleich, d.h. ein String wird als Zahlenkette abgespeichert. Z.B. wiird "A"= CHR$(65) intern als Byte mit dem Wert 65 abgespeichert.

Der Vorteil, die Spritedefinitionen über Strings durchzuführen ist, daß zur Stringbehandlung eine Menge Befehle zur Verfügung stehen, die alle benutzt werden können, um dann mit >SPRITE$< das Spritemuster zu manipulieren.
Betrachten wir folgenden Sprite:

```
1 ********      &HFF
2 *** ***       &HEE
3 ******        &HFC
4 * ***         &HB8
5 ******        &HFC
6 *** ****      &HEF
7 **   **       &HC6
8 *    * *      &H85
  12345678
```

Dieser Sprite soll als Sprite Muster 1 definiert werden:

1.Möglichkeit:

```
10
SPRITE$(1)=CHR$(&HFF)+CHR$(&HEE)+CHR$(&HFC)+CHR$(&HB8)+CHR$(&
HFC)+CHR$(&HEF)+CHR$(&HC6)+CHR$(&H85)
```

2.Möglichkeit:

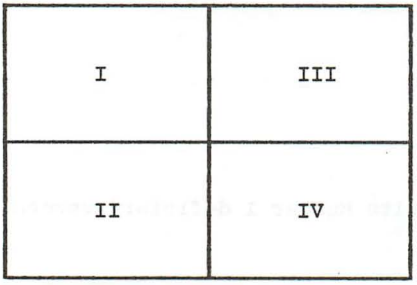
```
10 FORI=0TO7:READB$:A$=A$+CHR$(VAL("&H"+B$)):NEXT
20 SPRITE$(1)=A$
30 DATA FF,EE,FC,B8,FC,EF,C6,85
```

Die zweite Möglichkeit, die zunächst umständlicher aussieht, hat den Vorteil, daß die Sprite Definition im A$ gespeichert ist, also weiterhin zur Verfügung steht und evtl. später verändert werden kann.
Bei 8*8 Sprites können bis zu 256 (!!) (Nummern 0-255) verschiedene Spritemuster definiert werden. Von diesen 256 können dann je 32 beliebige Sprites gleichzeitig angezeigt werden.

Beachten Sie, das analog zur ersten Möglichkeit auch so etwas wie:

SPRITE$(1)="1AXz1,Bz" funktioniert.

Die 16*16 Sprites werden prinzipiell genauso definiert. Dazu wird die 16*16 Matrix in vier 8*8 Matrizen zerlegt, die dann hintereinander angegeben werden. Es gilt folgende Reihenfolge:



Insgesamt können 64 verschiedene 16*16 Sprite Muster definiert werden. Damit das Entwerfen von Sprites nicht mühsam auf dem Papier erledigt werden muß, stellen wir Ihnen hier einen Spriteeditor vor.

Spriteeditor

Die Funktion des Spriteeditors ist gleich der des Zeichengenerators. Der wesentliche Unterschied besteht in der Unterscheidung von 8*8 und 16*16 großen Sprites, und in den zusätzlichen Benutzerfunktionen, die über die Funktionstasten abrufbar sind.

Zunächst wählen Sie zwischen 8*8 und 16*16 Punkten, dann zwischen normal und vergrößert aus.
Danach können sie sich, wie üblich, mit dem Joystick (Cursortasten) innerhalb einer Matrix bewegen und durch Drücken des Auslösers (Leertaste) Punkte setzen bzw. zurücksetzen.

Neben der Definitionsmatrix sehen Sie die zum jeweiligen Muster dazugehörigen Werte angezeigt. Diese müssen Sie dann in der Reihenfolge spaltenweise von oben nach unten für die Spritedefinition in Ihrem Programm benutzen. Dabei stellt die linke Spalte die Werte für den ersten und zweiten Quadranten und die rechte Spalte für den dritten und vierten Quadranten (es liegt das 16*16 Spritemuster auf der vorigen Seite zugrunde) in hexadezimaler Form dar.
Diese Werte müssen Sie entweder in >CHR$< Form (Möglichkeit 1: ein paar Seiten weiter vorn) oder in >DATA< Zeilen (Möglichkeit 2: an der gleichen Stelle) eingeben. Wenn Sie nicht mit Hexadezimalzahlen vertraut sind, schauen Sie sich das Kapitel über Maschinensprache an. Hier finden Sie unter der Überschrift Zahlensysteme die von Ihnen benötigten Informationen.

Den Sprite in Originalgröße sehen Sie über der Editiermatrix angezeigt.

Hier folgt nun die Funktionstastenbelegung:

Key 1: Norm (Normalmodus)
Wenn Sie diese Funktionstaste gedrückt haben, können Sie mit dem Joystick einzelne Punkte innerhalb der vorher ausgewählten Matrix durch Druck auf den Feuerknopf löschen oder setzen.

Key 2: Lion (Line on Modus)
Durch das Drücken der 2ten Funktionstaste schalten Sie den Line on Modus ein. In diesem Modus könen Sie durch das Drücken des Feuerknopfes innerhalb der Spritematrix Linien zeichnen. Sie sind also nicht mehr gezwungen einzelne Punkte zu setzen, sondern zeichnen geschlossene Linien.

Key 3: Liof (Line off Modus)
Diese Taste bewirkt das Einschalten des Line off Modus. Dieser ist genau das Gegenteil von "Lion", d.h., Sie löschen geschlossene Linien

Key 4: prnt(Printer Modus)
Wenn Sie diese Taste gedrückt haben, wird der von Ihnen gezeichnete Sprite mit seinen Werten (&H) ausgedruckt.

Key 5: Ende
Diese Taste beendet das Programm.

```
10 DEFINT A-Z
20 SCREEN 2
30 A$="J"
40 INPUT"8x8 Sprite  (j/n) ";A$
50 IF (ASC(A$)AND&B11011111)=ASC("J") THEN M=0 ELSE M=2
60 A$="N"
70 A$="N":INPUT"vergroessert (j/n)";A$
80 IF (ASC(A$)AND&B11011111)=ASC("J") THEN M=M+1
90 SCREEN 1,M:COLOR 15,4,4
100 MK=INT(M/2)
110 BA=BASE(9)
120 XA=1:YA=6:QZ=1
130 EI=1
140 ON STRIG GOSUB 530,530,530,530,530
150 STRIG(EI) ON
160 AU$=" ":EI$=CHR$(42)
170 ON KEY GOSUB 700,620,660,850,740
180 KEY 1,"norm":KEY (1) ON
190 KEY 2,"Lion":KEY (2) ON
200 KEY 3,"Liof":KEY (3) ON
210 KEY 4,"prnt":KEY (4) ON
220 KEY 5,"Ende":KEY (5) ON
230 CLS:PUT SPRITE 0,(24,0)
240 FOR QZ=1 TO 1+3*MK
250 FOR Y=0 TO 7
260 GOSUB 580
270 NEXT Y,QZ
280 GOSUB 700
290 X=0:Y=0:QZ=1
300 LOCATE XA-8*(QZ>2)+X,YA-8*(QZ=2 OR QZ=4)+Y,1
310 R=STICK(EI)
320 IF R=0 THEN 310
330 STRIG(EI) OFF
340 IF R=8 OR R=1 OR R=2 THEN Y=Y-1
350 IF R>3 AND R<7 THEN Y=Y+1
360 IF R>1 AND R<5 THEN X=X+1
370 IF R>5 AND R<=8 THEN X=X-1
380 IF X<8 THEN 410
```

```
390 IF M<2 THEN X=7:BEEP:GOTO 410
400 IF QZ=1 OR QZ=2 THEN X=0:QZ=QZ+2 ELSE X=7:BEEP
410 IF X>=0 THEN 440
420 IF M<2 THEN X=0:BEEP:GOTO 440
430 IF QZ=3 OR QZ=4 THEN X=7:QZ=QZ-2 ELSE X=0:BEEP
440 IF Y<8 THEN 470
450 IF M<2 THEN Y=7:BEEP:GOTO 470
460 IF QZ=1 OR QZ=3 THEN Y=0:QZ=QZ+1 ELSE Y=7:BEEP
470 IF Y>=0 THEN 500
480 IF M<2 THEN Y=0:BEEP:GOTO 500
490 IF QZ=2 OR QZ=4 THEN Y=7:QZ=QZ-1 ELSE Y=0:BEEP
500 IF LF=0 THEN STRIG(EI) ON:GOTO 300
510 IF LF=-1 THEN BY=VPEEK(BA+(QZ-1)*8+Y) OR 2^(7-X):GOS
UB 560:GOTO 300
520 BY=VPEEK(BA+(QZ-1)*8+Y) AND (NOT 2^(7-X)):GOSUB 560:
GOTO 300
530 REM Strig
540 BY=VPEEK(BA+(QZ-1)*8+Y) XOR2^(7-X)
550 LOCATE ,,0
560 IF (BY AND 2^(7-X))=0 THEN PRINT AU$; ELSE PRINT EI$
;
570 VPOKE BA+(QZ-1)*8+Y,BY
580 LOCATE XA+8*(1+MK)+3-3*(QZ>2),YA+Y-8*(QZ=2 OR QZ=4)
590 PRINTRIGHT$("00"+HEX$(BY),2);
600 LOCATE XA-8*(QZ>2)+X,YA-8*(QZ=2 OR QZ=4)+Y,1
610 RETURN
620 REM Lion: Linie setzen
630 LF=-1
640 A$="Linie Zeichnen":GOSUB 800
650 RETURN
660 REM liof: Linie loeschen
670 LF=1
680 A$="Linie Loeschen":GOSUB 800
690 RETURN
700 REM Normalbetrieb
710 LF=0
720 A$="Normalbetrieb ":GOSUB 800
730 RETURN
```

```
740 REM Ende
750 LOCATE ,,0
760 DEFUSR1=&H139D
770 X=USR1 (1)
780 CLS
790 END
800 XP=POS(0):YP=CSRLIN
810 LOCATE 10,1,0
820 PRINTA$;
830 LOCATE XP,YP,1
840 RETURN
850 REM Print
860 LOCATE ,,0:TB=BASE(5)
870 FOR Z=YA TO YA-1+8*(1+MK)
880 FOR S=0 TO 31
890 LPRINT CHR$(VPEEK(TB+32*Z+S));
900 NEXT S
910 LPRINT:NEXT Z
920 LOCATE ,,1:RETURN
```

Programmbeschreibung:

Der prinzipielle Aufbau des Spriteeditors ist gleich dem des Zeichengenerators. Der wesentliche Unterschied besteht in der Unterscheidung 8*8, 16*16 Spritegröße und in den zusatzlichen Funktionen, die die Benutzung sehr komfortabel gestalten.

Anhand der zusätzlichen Variablen werden die Unterschiede deutlich.
M enthält den Spriteanzeigemodus (0 bis 3) und MK ist 0 bei 8*8 und gleich 1 bei 16*16 Spritematrix.
Wichtig ist die Variable QZ. Sie enthält die Nummer des 8*8 Unterblocks bei 16*16 Spritegröße.
Anhand von QZ, X und Y kann immer die reale Bildschirmposition ermittelt werden.

Teilweise ergeben sich relativ komplizierte Strukturen, wie >...+8*(QZ>2)...<. Falls Ihnen diese Benutzung von arithmetischen Ausdrücken unbekannt ist, probieren Sie einmal folgendes aus:

```
PRINT 1=3 oder PRINT 4>2
```

Eine wahre Aussage liefert den Wert -1, eine falsche den Wert 0.
Aufbauend auf diesen Trick funktioniert die Unterscheidung der Darstellung von 8*8 und 16*16 Sprites. Machen Sie sich mit dieser Programmiertechnik vertraut. Sie kann oft sehr nützlich sein und Programme erheblich verkürzen.

Zur Unterscheidung der verschiedenen Betriebsarten wird die Variable LF benutzt. 0 bedeutet normal, 1 Zeile löschen und -1 Zeile setzen.
In allen Betriebsarten außer "normal", ist der STRIG Interrupt ausgeschaltet, da er nicht benötigt wird.

Wie wird nun ein Sprite bewegt?

Die Position eines Sprites wird durch die Koordinaten seiner linken oberen Ecke festgelegt. Diese Koordinaten werden mit >PUT SPRITE< festgelegt. Das komplette Format des Befehls ist:

PUT SPRITE Ebene, (x-Koordinaten, Y-Koordinaten) Farbe, Musternummer

Die Musternummer ist die bei
>SPRITE$(Musternummer)=...< zugeordnete Zahl.

Die Erzeugung von ganzen Bildern mit Sprites ist nun möglich. Ein Demoprogramm hierfür währe aufgrund der vielen Spritedefinitionen zu umfangreich. Bedenken Sie bei Ihren Programmen folgendes.

TIPS:

- Bildelemente, die größer als 16*16 Punkte sein sollen, können durch meherere nebeneinander angezeigte Sprites erzeugt werden. Bei Bewegung eines Riesenprites müssen dann natürlich mehrere >PUT SPRITES< Befehle hintereinander ausgeführt werden.

- Mehrfarbige Sprites sind prinzipiell durch die Überlappung von mehreren verschiedenen Farben darzustellen. Auch dabei müssen dann z.B. für die Bewegung eines dreifarbigen Sprites (besteht aus 3 genau übereinanderliegenden Sprites) auch drei gleichartige >PUT SPRITE< Befehle ausgeführt werden. Dabei brauchen die Koordinaten für den Fall, daß die Befehle direkt hintereinander stehen, nur beim ersten Befehl angegeben werden. Stimmen außerdem Musternummer und Ebenennummer überein, so kann einfach >PUT SPRITE n< geschrieben werden, wobei n die Muster- bzw. Ebenennummer ist.

- Auf einer horizontalen Linie können maximal nur 4 verschiedene Sprites angezeigt werden. Bei 5 und mehr werden jeweils die mit den 4 niedrigsten Ebenennummmern angezeigt.

- Eine interessante Anwendung der Sprites ist die "Kinosimulation". Bei 64 verschiedenen 16*16 Sprites können bei geschickter Programmierung sehr viele Bewegungsabläufe hervorragend dargestellt werden.

Hierzu haben wir ein Programm erstellt, das einen laufenden "Elefanten" erzeugt. Dabei wird nicht nur ein Muster bewegt, sondern durch Hintereinanderschalten verschiedener Muster, der Eindruck einer Schrittfolge (Prinzip des Zeichentrickfilms) vermittelt.

Programm: Laufende Elefantenherde

Spritedefinitionen (16*16)

```
    Sprite 1:
                        1234567812345678=8+8=16
          &H00    1 0000000000000000      &H00
          &H0C    2 0000110000000100      &H04
          &H12    3 0001001000000010      &H02
  A$(1)   &H13    4 0001001111110001      &HF1  A$(3)
          &H1A    5 0001101000001101      &H0D
          &H3E    6 0011111000000011      &H03
          &H40    7 0100000010000001      &H81
          &H89    8 1000100110000001      &H81
          &H56    1 0101011010000001      &H81
          &H90    2 1001000010000001      &H81
          &H57    3 0101011100001001      &H09
  A$(2)   &H94    4 1001010001001011      &H4B  A$(4)
          &H52    5 0101001010111010      &HBA
          &HF2    6 1111001010001010      &H8A
          &H02    7 0000001010001010      &H8A
          &H07    8 0000011110011110      &H9E
```

```
Sprite 2:

                    1234567812345678=8+8=16

       &H00     1 0000000000000000      &H00
       &H0C     2 0000110000000100      &H04
       &H12     3 0001001000000010      &H02
B$(1)  &H13     4 0001001111110001      &HF1   B$(3)
       &H1A     5 0001101000001101      &H0D
       &H3E     6 0011111000000011      &H03
       &H40     7 0100000010000001      &H81
       &H89     8 1000100110000001      &H81
       &H56     1 0101011010000001      &H81
       &H90     2 1001000010000001      &H81
       &H57     3 0101011100001001      &H09
B$(2)  &H94     4 1001010001001010      &H4A   B$(4)
       &H52     5 0101001010111010      &HBA
       &HF2     6 1111001010001010      &H8A
       &H05     7 0000010100010100      &H14
       &H0F     8 0000111100111100      &H3C


Sprite 3:

                    1234567812345678=8+8=16

       &H00     1 0000000000000100      &H04
       &H0C     2 0000110000000010      &H02
       &H12     3 0001001011100010      &HE2
C$(1)  &H17     4 0001011100010001      &H11   C$(3)
       &H1A     5 0001101000001101      &H0D
       &H1E     6 0001111000000011      &H03
       &H20     7 0010000000000001      &H01
       &H49     8 0100100110000001      &H81
       &H56     1 0101011010000001      &H81
       &H91     2 1001000100000001      &H01
       &H96     3 1001011000001010      &H0A
C$(2)  &HA4     4 1010010001001010      &H4A   C$(4)
       &HA2     5 1010001010111010      &HBA
       &HF2     6 1111001010001001      &H89
       &H01     7 0000000101000101      &H45
       &H03     8 0000001111001111      &HCF
```

Programm:

```
10 DEFINT A-Z
20 SCREEN 2,2
30 FOR I=1 TO 4:FOR J=0 TO 7:READ A$:A=VAL("&H"+A$)
40 A$(I)=A$(I)+CHR$(A):NEXT J,I
50 DATA 00,0C,12,13,1A,3E,40,89
60 DATA 56,90,57,94,52,F2,05,0F
70 DATA 00,04,02,F1,0D,03,81,81
80 DATA 81,81,09,4A,BA,8A,14,3C
90 SPRITE$(1)=A$(1)+A$(2)+A$(3)+A$(4)
100 B$(1)=A$(1)
110 B$(2)=LEFT$(A$(2),6)+CHR$(&H2)+CHR$(&H7)
120 B$(3)=A$(3)
130 B$(4)=A$(4):MID$(B$(4),4,1)=CHR$(&H4B):B$(4)=LEFT$(B$(4),
6)+CHR$(&H8A)+CHR$(&H9E)
140 SPRITE$(2)=B$(1)+B$(2)+B$(3)+B$(4)
150 SPRITE$(0)=B$(1)+B$(2)+B$(3)+B$(4)
160 FOR J=0 TO 31:READ A$:A=VAL("&H"+A$)
170 C$=C$+CHR$(A):NEXT J
180 DATA 00,0C,12,17,1A,1E,20,49
190 DATA 56,91,96,A4,A2,F2,01,03
200 DATA 04,02,E2,11,0D,03,01,81
210 DATA 81,01,0A,4A,BA,89,45,CF
220 SPRITE$(3)=C$
230 '
240 'Bewegung Sprite
250 '
260 X=256:Y=20
270 FOR N=0 TO 3:X=X-1
280 FOR Z=1 TO 15 STEP 2
290 PUT SPRITE Z,(((X+20*Z)MOD 287)-31,Y+(Z MOD
4)*30-2*(N=2)),Z,N
300 NEXT Z,N
310 IF X>-31 THEN 270
320 GOTO 260
```

Programmbeschreibung

Zeile 30:

Schleifen zum Auslesen der DATA Zeilen in der Reihenfolge A$(1) bis A$(4).
Lesen der DATA Zeilen mit automatischer "&H" Eingabe

Zeile 40:

Verknüpfung der in den DATA Zeilen angegebenen Werte zu A$(1) bis A$(4).

Zeile 50 bis 80:

DATA Zeilen Sprite 1

Zeile 90:

SPRITE$(1) zusammenbauen

Zeile 100 bis 130:

Sprite 2 und 0 aus Teilen von Sprite 1 zusammenbauen.

Zeile 140:

Sprite 2 aus den gelesenen Strings definieren

Zeile 150:

Sprite 0 definieren

Zeile 160:

Schleife zum Auslesen von Sprite 3.
Lesen der DATA Zeilen mit automatischer "&H" Eingabe.

Zeile 170:

Verknüpfung der in den DATA Zeilen angegebenen Werte zu C$(1) bis C$(4).

Zeile 180 bis 210:

DATA Zeilen zu Sprite 3

Zeile 220:

Mit SPRITE$(3)=C$ Sprite 3 definieren

Zeile 260:

X- und Y-Koordinate zur Positionierung auf dem Bildschirm bestimmen.

Zeile 270:

Reihenfolge des Hintereinanderfolgen der verschiedenen Sprites in einer Schleife bestimmen.
X bestimmt die Bewegungsrichtung auf dem Bildschirm.

Zeile 280:

8 Sprites auf den Bildschirm in verschiedenen Farben hintereinander ausgeben.

Zeile 290:

Ruft die Sprites auf. Bestimmt mit Hilfe der in Zeile 270 und 280 angegebenen Schleifen Bewegungsrichtung, Anzahl und Reihenfolge des Auftretens der Sprites.

Zeile 310:

Wenn der Bewegungsrichtungszähler X größer als -31 ist, weiter in Zeile 270. D.h., wenn linker Bildschirmrand überschritten.

Zeile 320:

Sonst in Zeile 260 fortfahren.

Sprites Intern

Abschließend behandeln wir noch kurz die interne Verarbeitung der Sprites im VRAM und die damit zusammenhängenden VDP Register.

Genauso, wie es eine Zeichen- oder Mustertabelle gibt, existiert auch eine Spritemustertabelle. Sie ist 2048 (=2K Byte) lang und kann in 2K Byte Schritten verschoben werden. Es besteht die Möglichkeit zwischen mehreren Spritetabellen per Interrupt umzuschalten siehe Kapitel Grafik Modus I.
Die Spritemustertabelle ist in 256 Blöcke zu je 8 Byte (=ein 8*8 Spritemuster) unterteilt. Bei 8*8 Sprites ist die Musternummer gleich der Nummer des in der Tabelle stehenden 8ter Blocks, der die Definitionen enthält. Bei 16*16 Sprites muß die Musternummer mit 4 multipliziert werden, um die Blocknummer der Definition zu erhalten.

Prinzipiell können Sie also Sprites auch durch >VPOKE< definieren. Die Anfangsadresse der Spritemustertabelle wird durch das VDP Register 6 bestimmt. Dieses Register enthält die 3 MSB's der Adresse der Spritemustertabelle.

Die Startadresse kann auch durch >BASE(PEEKC(&HFCAF)*5+4)< für alle Modi ermittelt werden. Beim Einschalten des Rechners belegt die Spritemustertabelle in allen möglichen Modi die Adresse &H3800 bis &H3FFF. Damit ist der Wert vom VDP Register gleich 6. Die zweite Tabelle ist die sogenannte Sprite Attribut Tabelle (SAT). Sie enthält die Informationen, die durch >PUT SPRITE< festgelegt werden. Da 32 Ebenen existieren ist die SAT in 32 Blöcke unterteilt. Jeder Block ist 4 Bytes lang. Die Blocknummer (0-31) korrespondiert mit der jeweiligen Ebenennummer, d.h. Eintragungen in Block 5 gelten für Ebene 5.

Die SAT ist also in 128er Schritte verlegbar. VDP Register 5 enthält die 7 oberen Bits der 14-Bit Adresse der Tabelle.Mit >BASE< kann die SAT-Startadresse in allen Modi mit >BASE (PEEK(&HFCAF)*5+3)< ermittelt werden.

Beim Einschalten ergibt dies den Wert &H1B00. Die Tabelle belegt also den Bereich von &H1B00 bis &H1BFF, VDP Register 5 enthält &H1B00/2^(14-7)=&H36

Die Eintragungen innerhalb eines Blockes zu je 4 Byte haben folgende Bedeutung:

Byte 0 - Y Koordinaten
Byte 1 - X Koordinaten
Byte 2 - Musternummer
Byte 3 - u.a. Farbe

Die Koordinaten beziehen sich grundsätzlich auf die linke obere Ecke des Sprites.

Auch negative Y-Koordinaten sind möglich (Darstellung im 2er Komplement:256- Betrag der Zahl), was bedeutet, daß der obere Teil der Sprites nicht mit angezeigt wird. Er ist sozusagen hinter dem Rahmen verschwunden. Damit ist es möglich Sprites langsam in den Bildschirm hineinlaufen zu lassen. Bei >PUT SPRITE< können folglich Y-Koordinaten zwischen -31 und +192 angegeben werden. Bei der X-Koordinate ist dies nicht so einfach möglich, da schon alle 256 möglichen Werte eines Bytes als normale Koordinaten benutzt werden. Um das zu ermöglichen, wird Bit 7 des 4ten Bytes eines Blocks benutzt. Es heißt das EC (Early Clock) Bit. Ist es Null, herrscht der normale Zustand. Hat es den Wert 1, dann sind alle X-Positionen der Sprites um 32 Punkte nach links verschoben. Dann kann durch X-Koordinaten zwischen 0 und 32 erreicht werden, daß der Sprite von links in den Bildschirm läuft.

Beim >PUT SPRITE< Befehl können direkt Werte zwischen -32 und +255 eingegeben werden.

Das 3te Byte enthält einen Zeiger in die Spritemustertabelle. Bei 8*8 Sprites ist sie gleich der Spritenummer des BASIC. Die Bits 0-3 des 4ten Bytes schließlich enthalten den Code für die Spritefarbe. Um den Spriteanzeigemodus zu bestimmen, sind 2 Bits des VDP Register 1 vorgesehen:
Bit 0 Mag - Magnification - Vergrößerung
Bit 1 Sizw- Size - Größe

	0	1
MAG SIZE	normal 8*8	vergrößert 16*16

Treten mehr als 4 Sprites in einer horizontalen Linie auf, so wird im VDP Register 8, das üblicherweise als Statusregister (read only) bezeichnet wird, das Bit 6 gesetzt. In den Bits 0 bis 4 wird dann die Nummer des 5ten Sprites, der auf der ersten nicht mehr angezeigten Ebene liegt, gespeichert.

Schließlich gibt es noch das sogenannte Coincidence Flag, das anzeigt, ob sich mindestens zwei Sprites überlappen. Das C-Flag ist das Bit 5 des Statusregisters. Über dieses Bit ist die Basic Interrupt Funktion "ON SPRITE" möglich. Damit können Spritekkollisionen sofort verarbeitet werden.

KAPITEL III: I/O

3.1 Allgemeine Überlegungen zum I/O

Was wäre ein Rechner ohne Peripherie, d.h. ohne Bildschirm, ohne Tastatur, Kasette, Drucker usw.?

Die Möglichkeiten eines Computers werden erst nutzbar, wenn eine entsprechende Peripherie, besonders Tastatur und Bildschirm, vorhanden und anschließbar sind. Dazu besitzt ein Rechner verschiedene sogenannte externe Schnittstellen. Beim MSX System sind das unter anderem:

Monitorbuchse
RGB-Monitor
Drucker-Schnittstelle
Joystick Ports
Cartridge Slots

Zunächst werden wir noch einmal kurz ein MSX-Peripheriegerät vorstellen:

Den Datenrecorder:

Zur grundsätzlichen Bedienung der Kassette ist nicht viel zu sagen.
Die Standardbefehle >CSAVE< und >CLOAD< dienen zum Speichern/Laden von BASIC Programmen.
Die Befehle >BLOAD< und >BSAVE< sind zum Abspeichern von Speicherinhalten gedacht. Diese Anwendung kommt meisten bei der Benutzung von Maschinenprogrammen vor.

Interessant sind die Befehle >SAVE< und >LOAD<.

Es sind allgemeine Befehle, die zur Datenübertragung mit Hilfe von Dateien dienen.
Die Anzahl der gleichzeitig zu benutzenden Dateien wird durch >MAXFILES< festgelegt. Durch diesen Befehl werden im RAM Bereiche zur Verwaltung der Dateien reserviert.

Das Besondere an >LOAD< und >SAVE< ist, daß sie sich nicht nur auf den Datenrecorder beziehen. Vielmehr kann "theoretisch" jedes Peripheriegerät als Ziel- bzw. Quelladresse der zu übertragenden bzw. zu empfangenen Daten dienen. Dafür ist es möglich im Argument dieser Befehle das Ausgabegerät mit anzugeben. Sämtliche Dateibehandlungsbefehle (>OPEN, PRINT#, CLOSE<) können in der oben beschriebenen Weise benutzt werden.
Durch das Definieren eines anderen Zieles wird dann der gesamte Datenstrom auf das jeweilig angegebene Gerät geleitet. An folgende Geräte kann in dieser Form ausgegeben werden:

CAS - Kassette
CRT - Bildschirm
GRP - Grafik Bildschirm
LPT - Drucker

Die Technik der Datenausgabe über >PRINT#< haben wir schon oft in den Grafikprogrammen vom vorigen Kapitel verwendet.

Obwohl Peripherie ein so wichtiges Thema ist, wird die Behandlung oft vernachlässigt. Die I/O-Programmierung ist sehr Hardware bedingt, und damit schwer zugänglich. Viele Ideen sind nur mit Hilfe der Maschinensprache zu verwirklichen. Im folgenden behandeln wir die Grundzüge der I/O Programmierung bezogen auf MSX Rechner.

Bildschirmschnittstellen

Das Signal, das über den normalen Antennenstecker abgerufen werden kann, wird durch einen eingebauten Modulator aus den, vom VDP kommenden Informationen, zusammengesetzt. Das Soundsignal vom PSG (Programmable Sound Generator) ist hier mit eingearbeitet.

Der Monitorausgang benutzt dieselben Grundinformationen, nur wird das Signal nicht moduliert, d.h. es kann nur von einem Monitor, nicht aber von einem Fernseher verarbeitet werden.
Der RGB Ausgang liefert ein Signal für hochauflösende Farb-

monitore. Es wird wiederum aus denselben Grundinformationen aufgebaut.
Interessanterweise enthält der RGB Ausgang ein Stereosignal. Der rechte und linke Kanal sind einzeln abrufbar. Da es hier mit einfachen Mitteln nichts zu programmieren oder zu verändern gibt, wenden wir uns dem zweiten wichtigsten Peripheriegerät zu, der Tastatur:

Tastatur

Elektronisch gesehen ist die Tastatur ein rechteckiges Netz, aus gekreuzten Leitungen. Auf jedem Knotenpunkt sitzt eine Taste. Man spricht in diesem Zusammenhang von einer Tastaturmatrix. Die Tastaturabfrage wird 50mal pro Sekunde durch den internen Interrupt durchgeführt. Dazu wird nacheinander jeweils durch eine waagerecht laufende Leitung ein Strom geschickt (Bit=1). An den senkrecht verlaufenden Leitungen wird dann geprüft, ob ein Strom anliegt. Führt eine dieser Leitungen Strom (Bit=1), so ist die auf dem Knotenpunkt dieser und der jeweilig waagerecht verlaufenden Leitung liegende Taste gedrückt.
Das bedeutet, daß der Stromkreis geschlossen ist. Auf diese Weise wird ständig ermittelt, welche Tasten gedrückt sind.

Wie ist dieser Vorgang nun zu programmieren?

Zunächst wird ein Ausgabebefehl gebraucht (OUT) der je eine waagerechte Leitung unter Strom setzt, und sofort danach ein Eingabebefehl (>INP<), der zurückgibt welche Taste gedrückt ist. Der IC, der die Tastaturabfrage ausführt, ist der PPI (Programmable Periphal Interface). Der MSX Standard schreibt den PPI 8255 von der Firma Intel vor.

Der Ausgabeport, der den Strom in die Tastatur schickt, ist Port C (der PPI). Um die verschiedenen Ausgänge zu unterscheiden, ist jedem I/O Port eine Nummer zugeordnet.

Port C des PPI hat die Nummer &HAA.
Die unteren 4 Bits vom Port C des PPI bestimmen die waagerechte Leitung, durch die der Strom gesendet wird. Die oberen 4 Bits sollen unberührt bleiben. Damit sieht der Programmanfang folgendermaßen aus:

```
10 CLS
20 Z=INP(&HAA):REM PORT C Lesen
30 Z=Z1 AND &HF0:REM oberen 4 Bits erhalten und untere 4 Bits löschen
40 LOCATE 0,0
```

Nun folgt die Abfrage der 9 möglichen Leitungen:

```
50 FOR I=0 TO 8
60 OUT &HAA,Z+I:REM Ite Leitung abfragen
```

Nun wird der Wert der senkrechten Leitung mit >INP< gelesen. Die von der Tastatur kommenden Leitungen bilden den Port B des PPI. Er hat die Nummer &HA9.

```
70 PRINT RIGHT$("0000000"+BIN$(INP(&HA9)),8)
80 NEXT I
90 GOTO 20
```

Lassen Sie das Programm laufen, und Sie erhalten das Bild der Tastaturmatrix auf dem Bildschirm. Eine 0 zeigt jeweils eine gedrückte Taste an.
Fügen Sie jetzt noch Zeile 45 und 46 ein

```
45 A$=INKEY$:IF A$="" THEN PRINT " " ELSE PRINT A$
46 LOCATE 0,1
```

und Sie können die Tastaturmatrix füllen. Es ergibt sich folgendes Bild:

7	6	5	4	3	2	1	0
8	9	-	=	Û			;
'	\	,	.	/		A	B
C	D	E	I	G	H	I	J
K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z
SHIFT	CTRL	GRAPH	CAP	CODE	F1	F2	F3
F4	F5	ESC	TAB	STOP	BS	SELECT	RETURN
SPACE	HOME	INS	DEL				

Beachten Sie, daß die durch >INKEY$< abgefragte Taste oft nur beim Drücken einer einzigen Taste mit der Matrix übereinstimmt.
Das bedeutet, das Sonderzeichen mit >SHIFT, CTRL, GRAPH usw.< von der Abfrageroutine speziell erkannt und interpretiert werden. Wollen wir also unsere eigene Tastaturabfrage programmieren, so müssen wir auf die mit dem obigen Programm ausgelesenen Daten zurückgreifen. Da, wie schon erwähnt, diese Abfrage auch intern durch den Interrupt ausgeführt wird, können wir auf diese Daten zurückgreifen, die im sog. NEWKEY - Puffer gespeichert sind. Unser Programm vereinfacht sich damit zu:

```
10 CLS
20 LOCATE 0,0
30 FOR I=&HFBE5 TO &HFBED
40 PRINT RIGHT$("0000000"+BIN$(PEEK(I)),8)
50 NEXT I:GOTO 20
```

Wie verwenden wir jetzt diese Bitmatrix so, daß wir alle gedrückten Tasten herausbekommen?
Das Programm, das eine solche Tastaturabfrage erledigt, werden wir im nächsten Kapitel benutzen, um einen Polyphonen (d.h. mehrstimmigen) Klang zu programmieren.

Innerhalb des ROM's sind alle Tabellen, die zur Tastaturdecodierung notwendig sind enthalten. Betrachten wir die Tabelle der Tastaturbelegung ohne >SHIFT< oder andere Spezialtasten. Sie liegt ab Adresse &H0DA5. Ab dieser Adresse finden wir die ASCII Codes der Zeichen wieder, die durch einfaches Drücken einer Taste erzeugt werden:

```
FOR I=0 TO 5:FOR J=7 TO 0 STEP-1:PRINT CHR$(PEEK(&HDA5+I*8+J)
);:NEXT:PRINT:NEXT
```

Diese Tabellen werden intern benutzt, um den zu den jeweilig gedrückten Tasten gehörenden ASCII Code herauszufinden. Die ASCII Codes werden dann im sog. KEYBUFfer abgelegt und stehen dort zur Verfügung.
Folgendes kleines Unterprogramm liest den Keybuffer aus und bereitet ihn zur sofortigen Neueingabe vor.

```
10 KB=VAL("&HFBF0"):REM Startadresse KEYBUF
20 GOTO 50: KEYBUF Initialisation
30  A=PEEK(&HF3F8)-&HF0:IF A=0 THEN 30:REM Anzahl  gedrückter
Tasten
40   FOR   I=KB   TO   KB+A-1:PRINT   CHR$(PEEK(I));:NEXT:REM
Pufferinhalt ausgeben
50 POKE &HF3F8,&HF0:POKE &HF3F9,&HFB:REM Puffer Init
60 POKE &HF3FA,&HF0:POKE &HF3FB,&HFB
70 POKE &HF3F7,1:REM Wiederholungswartezeit abschalten
80 PRINT:GOTO 30
```

Interessant ist der >POKE< Befehl in Zeile 70:

Wird die Zeile weggelassen, so wird ein "Gedrückt lassen" der Tasten nicht registriert. Normalerweise wird, bis die Dauerfunktion einer Taste in Kraft tritt, eine gewisse Wartezeit berücksichtigt. Der Wartezeitzähler (REPCNT-Repeat Counter) ist der Wert an Adresse &HF3F7.
Wird er immer wieder auf den Wert 1 gesetzt, ist die Tastenwiederholung mit minimaler Wartezeit möglich.
Soviel zum I/O Gerät Tastatur.

Weiter oben haben wir die vier oberen Bits von PORT C nicht berücksichtigt. Betrachten wir zunächst Bit 6.

Bit 6 von Port C der PPI gibt den Zustand der Capdiode wieder. Die Bedeutung von Bit 6 ist dabei folgende:

Bit 6=1 : Die CAP Diode leuchtet nicht
Bit 6=0 : Die CAP Diode leuchtet

Probieren Sie:

Cap Diode Ausschalten und dann:

```
PRINT HEX$(INP(&HAA))
```

Cap Diode Anschalten und dann:

```
PRINT HEX$(INP(&HAA))
```

Ein Ausschalten der Capdiode wird mit

```
OUT &HAA,INP(&HAA) OR 216
```

oder einfacher mit

```
OUT &HAB,13
```

erreicht.

Die zweite Version stellt einen Steuerbefehl über den Steuerport &HAB des PPI dar. Der Befehl lautet:
"Setze Bit 6 von Port C auf 1."
Ein Anschalten der CAP Diode wird mit

```
OUT &HAA,INP(&HAA) AND NOT 2^6
```

oder durch den Steuerbefehl

```
OUT &HAB,12
```

der "Setze Bit 6 von Port C auf 0" bedeutet, erreicht.

Beachten Sie, daß dadurch lediglich die Leuchtdiode betroffen ist. Die Umschaltung von Groß- zu Kleinbuchstaben verändert sich nicht. Soll diese Umschaltung vom Programm aus vorgenommen werden, so muß folgendermaßen vorgegangen werden.

1) Simulation "CAP Taste drücken" per Programm.

```
DEF USR1=&HF36:X=USR1(0)
```

Dieser Maschinenprogrammaufruf wechselt bei jedem Aufruf den Status von Cap, d.h. er entspricht genau dem Drücken der Captaste.

2) Test, ob Groß- bzw. Kleinschrift eingeschaltet ist:

```
F=PEEK(&HFCAB):IF F=0 THEN PRINT"KLEINSCHR." ELSE PRINT"
GROSSBUCHST."
```

3) Ordnungsgemäßes Einschalten von Großschrift und der Capdiode:

```
POKE &HFCAB,0:OUT &HAB,13
```

Im Zusammenhang mit der Captaste ist z.B. auch eine Warteschleife über den >WAIT< Befehl möglich. Der >WAIT< Befehl führt praktisch ein >INP< Befehl aus, und er vergleicht den erhaltenen Wert mit dem angegebenen. Es wird ein >XOR< mit dem 2ten Wert, und dann ein >AND< mit dem 1ten angegebenen Wert durchgeführt. Ist das Ergebnis 0, so wird weitergewartet, ansonsten wird das Programm fortgesetzt.
Vereinbaren wir nun, daß eine leuchtende Capdiode den Wartezustand anzeigt. Durch

```
WAIT &HAA,2^6
```

wird das Programm solange gestoppt, bis die Captaste gedrückt wird, was die Leuchtdiode ausschaltet.

Soll dagegen gewartet werden, bis die Diode angeschaltet wird, so muß das entsprechende Bit durch >XOR< invertiert werden.

```
WAIT &HAA,2^6,2^6
```

Soviel nun zur Captaste.

Betrachten wir jetzt die Funktion des Bit 7 von Port C.

Sicherlich kennen Sie das sogenannte Tastaturbetätigungsgeräusch ("Knacken"). Mittels des >SCREEN< Befehls oder durch

```
POKE &HF3DB,0
```

kann dieses Geräusch ausgeschaltet werden.

Dieses Knacken wird ausnahmsweise, obwohl es sich um einen "Ton" handelt, nicht vom PSG (Sound Chip) erzeugt. Vielmehr besteht die Möglichkeit über den PPI ein Knacken auf den Audioausgang zu geben. Das geschieht ganz einfach über das Setzen und Rücksetzen einer Leitung, die mit diesem Ausgang verbunden ist. Der Übergang von einem zum anderen Zustand bewirkt nun, daß die Lautsprechermembran bewegt wird, also ein Knacken zu hören ist.
Interessant ist nun die Möglichkeit, nicht nur dieses Knacken zu erzeugen, sondern durch eine sehr schnelle Aufeinanderfolge des Setzens bzw. Rücksetzens der Leitung, den Lautsprecher derartig in Schwingungen zu versetzen, daß ein Ton zu hören ist.

Schalten Sie zuerst das Tastaturbetätigungsgeräusch mit >SCREEN,,0< oder mit >POKE &HF3DB,0< aus. Dann geben Sie folgendes ein:

```
OUT &HAB,15
```

Obwohl das Knackgeräusch ausgeschaltet ist, hören Sie beim Betätigen der Returntaste ein Knacken. Dieses Geräusch zeigt

uns an, daß die Lautsprechermembran nun "gespannt" ist. Bei nochmaliger Ausführung desselben Befehls hören Sie nichts, da sich der Zustand der Membran nicht ändert. Durch

```
OUT &HAB,14
```

wird die Spannung am Lautsprecher wieder auf Null gesetzt. Da die Membran durch den >OUT &HAB,15< Befehl gespannt war, ist auch dabei ein Knacken zu hören.
Den aktuellen Zustand dieser sogenannten Software Sound Leitung erhalten wir durch einen >INP< Befehl. Der Zustand der Leitung entspricht Bit 7 von Port C des PPI. Dieser Port hat die Adresse &HAA. Um ausschließlich Bit 7 zu betrachten, verwenden wir die logischen Befehle:

```
PRINT INP(&HAA) AND 2^7
```

Erhalten Sie nach der Eingabe der obigen Zeile 0, so war die Leitung 0. Wird dagegen 128 ausgegeben, so befand sich die Leitung im Zustand 1.

Da ein Ton durch Schwingungen z.B. einer Saite oder einer Lautsprechermembran klingt, bewirkt ein schnelles Setzen bzw. Rücksetzen der Software Sound Leitung die Erzeugung eines Tons.

```
10 FOR I=0 TO 100
20 OUT &HAB,15
30 OUT &HAB,14
40 NEXT
```

Durch das Einfügen einer Zeile 25, in der z.B. eine Warteschleife ein >REM< oder ein >PRINT< Befehl steht, kann die Geschwindigkeit mit der das "Knacken" aufeinanderfolgt, verlangsamt werden. Dadurch wird die Frequenz, mit der die Lautsprechermembran schwingt, kleiner. Wenige Schwingungen, also geringe Frequenz, bedeutet einen niedrigeren Ton. Viele Schwingungen, also eine höhere Frequenz, bedeutet einen höheren Ton.

Die beiden letzten Bits von Port C sind im Zusammenhang mit dem Kasettenrekorder wichtig:

Bit 4 steuert den Kasettenmotor:

```
OUT &HAB,8 :Motor an
OUT &HAB,9 :Motor aus
```

Bit 5 schießlich wird benutzt, um Informationen auf Kassette zu speichern. >OUT &HAB,11< schaltet die Magnetisierungsspannung ein und >OUT &HAB,10< wieder aus.

3.2 Speicheraufteilung

Eine der wichtigsten Aufgaben des PPI im MSX System ist die Organisation der Speicheraufteilung.
Wie Sie wissen gibt es unterschiedliche Rechnerversionen innerhalb des MSX Standards. Unterschiedliche RAM Kapazitäten und/oder zusätzliche ROM's mit fertigen Anwenderprogrammen oder BASIC Erweiterungen für verschiedenste Spezialaufgaben sind im Handel. Außerdem besteht grundsätzlich die Möglichkeit die ROM oder RAM Kapazität des Rechners über Steckmodule zu erweiten.
Um alle diese Möglichkeiten zu stanardisieren und kompatibel zu machen, wurde eine variable Speicherplatzzuordnung eingerichtet.
Die Programmierung der unterschiedlichen Möglichkeiten erfolgt über Port A vom PPI.
Der Adressbereich der CPU deckt die Adressen von 0 bis &HFFFF ab. Diese 64K sind in vier "Pages" (=Seiten) zu 16K unterteilt. Jeder dieser Seiten kann ein eigener Speicherbereich zugeordnet werden, d.h. z.B. ROM für die Adressen 0 bis &H4000, Disketten Cartridge &H4000-&H8000 usw.. Bei der Zuordnung eines Speichers zu einer Seite gibt es je vier Möglichkeiten. Diese werden als die vier Slots des Rechners bezeichnet (wegen: Cartridge Slot = Steckmodulbuchse).
Das bedeutet:

Slot 0 umfaßt auf jeden Fall den eingebauten BASIC ROM (mindestens 32K).

Existieren weitere fest eingebaute ROM's, so sind diese meist auch Slot 0 zugeordnet. Ansonsten befindet sich im oberen Slot 0 der RAM Bereich für den Benutzer.

Slot 1 ist der in einem evtl. eingestecktem Modul vorhandene Speicher.

Slot 2 ist ein RAM Bereich; der zumindest im unteren Bereich bis &H8000 von allen 64K Versionen benutzt wird. Oft sind auch die oberen Adressen von Slot 2 mit RAM Speicher belegt.

Slot 3 ist ähnlich wie Slot 1 ein Erweiterungsmodul Speicherbereich. Slot 3 wird meist für die Diskettenbenutzung gebraucht.

Jeder 16K Seite des CPU Adressenraumes kann, unabhänig von den anderen Seiten ein Slot zugeordnet werden.

Adresse	Slot 0	Slot 1	Slot 2	Slot 3
0-&H3FFF	Betriebssystem ROM	Steck-platz 1	(RAM)	Steck-platz 2
&H4000-&H7FFF	BASIC ROM		(RAM)	
&H8000-&HBFFF	Personalprogramm ROM (Sony)		RAM	
&HC000-&HFFFF	(RAM)		RAM	

Was können wir mit diesen Informationen anfangen?

Zuerst ist es möglich die aktuelle Speicherkonfiguration Ihres Rechners zu ermitteln.
Geben Sie dazu:

```
PRINT BIN$(INP(&HA8))
```

ein. Unterteilen Sie die erhaltene Binärzahl von rechts ange-

fangen in Gruppen zu 2 Bits. Ermitteln Sie den Wert von jeder 2 Bit Gruppe (der Wert liegt zwischen 0 und 3).
Zum Beispiel beim Sony Hit Bit:

```
10 10 00 00
 2  2  0  0
```

Das bedeutet, daß im Bereich &HFFFF-&H8000, also den zwei oberen Seiten RAM aus Slot 2, und von &H8000-&H0, also den zwei unteren Seiten ROM aus Slot 0 (Systemslot) selektiert ist. Port A gibt also ständig über die Speicherkonfiguration Auskunft.
Interessant ist die Möglichkeit über >OUT< Befehle an Port A die Speicherkonfiguration auch vom BASIC aus manipulieren zu können. Hierzu sind sinnvollerweise 32K Versionen, aber besser noch 64K Versionen notwendig.

Da ein Ausschalten der ROM's vom BASIC aus den sofortigen Absturz des Computers zur Folge hat (z.B.>OUT &HA8,&HCC<), machen wir uns zunächst einen Experimentierbereich frei. Wir verlegen den Basicanfang nach &HC000 (quasi 16K Version) und haben damit die Seite 3 (&H8000-&HC000) für unsere Experimente zur Verfügung.
Die Verlegung des BASIC Bereichs geschieht durch ein Ändern der Adresse &HF677, die den BASIC Start enthält.

```
POKE &HC000,0:POKE &HF677,&HC0:NEW
```

Die Veränderung ist zunächst nicht bemerkbar. Nur sehr lange Programme können jetzt nicht mehr geladen werden. Zum Anschauen der Speicherinhalte folgt hier ein Minimonitor. Natürlich kann auch der Monitor aus Kapitel 5 benutzt werden.

```
10 FOR I=&H8000 TO &HB100 STEP 8
20 PRINT RIGHT$("000"+HEX$(I),4);" ";
30 FOR J=0 TO 7
40 BY=PEEK(I+J)
50 PRINT RIGHT$("0")+HEX$(BY),2);" ";
60 BY=BY AND 127
70 IF BY<32 OR BY>126 THEN BY=46
```

```
80 A$=A$+CHR$(BY)
90 NEXT J
100 PRINT A$;
110 A$="":NEXT I
```

Nun können wir also dem Bereich von &H8000 bis &HC000 (Seite 3) einen anderen Slot zuordnen.

```
OUT &HA8,(INP(&HA8) AND &HCF) OR (3*16)
```

Dabei steht die 3 für die gewünschte Slotnummer.
Wie schon erwähnt, liegt z.B. beim Sony Hit Bit, das eingebaute Programm für die "Personal Data Bank" im System Slot 0.
Speicherbelegung für andere Rechner entnehmen Sie bitte den mitgelieferten Handbüchern.
Schalten Sie also für Seite 3 auf Slot 0 um. Ändern Sie dann z.B. die Adressen in Zeile 10 zu &H8170 und &H8470 um. Als Speicherinhalt erhalten Sie die Meldungen der Personal Data Bank. Auch ein Auslesen der Diskettenerweiterung ist auf ähnliche Weise möglich.

Nun noch einiges, was nur mit den 64K RAM Versionen möglich ist. Vielleicht haben Sie sich schon gefragt, wozu Sie eigentlich einen 64K Rechner gekauft haben, wo doch nur ca. 28K (Einschaltmeldung) für BASIC Programme frei sind.

Wo sind denn nun die fehlenden Kilobytes?

Sie werden ständig vom System Slot 0, also von den ROM's überlagert. D.h. vom BASIC aus sind diese 32K RAM, die die Adressen von &H0 bis &H8000 in Slot 2 belegen nicht zugänglich. Trotzdem war Ihre Entscheidung für 64K richtig. Der Betrieb eines Diskettenlaufwerks ist nur mit 64K RAM möglich. Außerdem kann mit Maschinensprache der gesamte Bereich auch genutzt werden.
Schließlich besteht noch die Möglichkeit die 32K ROM, die nicht veränderbar sind in das darunterliegende RAM zu kopieren. Dann wird auf Slot 2, also RAM umgeschaltet, und der Rechnerbetrieb kann normal weiterlaufen. Der entscheidende

Vorteil dabei ist jedoch, daß das gesamte BASIC und das Betriebssystem über >POKE< Befehle nach Belieben geändert werden kann, da es jetzt im RAM stehen. Damit können wir uns unser eigenes BASIC herstellen bzw. das vorhandene nach eigenen Wünschen modifizieren.

Zunächst also das Programm, welches den ROM Inhalt von Slot 0 ins RAM von Slot 2 kopiert.

```
F000 210080     10        LD   HL,&H8000
F003 16A0       20        LD   D,&HA0
F005 1EAA       30        LD   E,&HAA
F007 0EA8       40        LD   C,&HA8
F009 F3         50        DI
F00A 2B         60  NEXT  DEC  HL
F00B ED51       70        OUT  (C),D
F00D 7E         80        LD   A,(HL)
F00E ED59       90        OUT  (C),E
F010 77        100        LD   (HL),A
F011 7C        110        LD   A,H
F012 B5        120        OR   L
F013 20F5      130        JR   NZ,NEXT
F015 C9        140        RET
```

```
Programm :ramrom
Start : &HF000    Ende : &HF015
Länge : &H16 Bytes
Fehler : 0
Variablentabelle :
Next  F00A
```

Der BASIC Lader:

```
10 CLEAR 200,&HEFFF
20 FOR I=&H7000 TO &HF015
30 READ A$:A=VAL("&H"+A$):POKE I;A:NEXT
40 DEF USR1=&HF000
50 X=USR1(1)
60 END
70 DATA 21,00,80,16,A0,1E;AA,DE
80 DATA A8,F3,2B,ED,51,7E,ED,59
90 DATA 77,7C,B5,20,F5,C9
```

Dieses Programm geht von folgender Konfiguration aus (Sony Hit Bit):

	Slot 0	Slot 2
&H0000 bis &H7FFF	ROM	RAM
&H8000 bis &HFFFF	——	RAM

Für andere Aufteilungen müssen die Werte entsprechend geändert werden.

Seite	0	1	2	3
A0 entspricht:	ROM Slot0	ROM Slot0	RAM Slot2	RAM Slot2
AA entspricht:	RAM Slot2	RAM Slot2	RAM Slot2	RAM Slot2
			Benutzer	RAM

Nachdem Sie das Programm laufen gelassen haben, findet keine sichtbare Veränderung statt. Doch der Unterschied ist gravierend:
Schalten Sie zunächst mit >OUT &HA8,&HA0< die alte Konfiguration wieder ein. Probieren Sie dann die Tastaturbelegungstabelle zu ändern >POKE &HDA5,32<.
Adresse &HDA5 ist der ASCII Code, der der "0" Taste zugeordnet ist. Jedoch ändert sich durch die obige Eingabe nichts an der Tastaturbelegung! Schalten Sie nun auf RAM um >OUT &HA8,&HAA< und probieren Sie wieder >POKE &HDA5,32<. Nun drücken Sie noch die "0" Taste.
Sie erhalten ein Leerzeichen (ASCII Code 32). Durch >POKE &HDA5,49< wird die "1" auf die "0" gelegt usw..
Nun ist es viel einfacher möglich die deutsche Schreibmaschinentastatur (falls noch nicht, wie bei einigen Modellen vorhanden) zu erzeugen. Experimentieren Sie ein wenig mit den Tastaturmatrixtabellen herum:

&HDA5 - &HDD4	ohne Shift
&HDD5 - &HED4	mit Shift
&HE05 - &HE34	mit Graph
&HE35 - &HE5D	mit Graph Shift
&HE65 - &HE94	mit Code
&HE95 - &HEC4	mit Code Shift
&H1033 - &H104A	Steuerzeichen; die 3 untersten Reihen der Tastaturmatrix
&H104B - &H105A	Tabelle für Ziffernblock falls vorhanden
&H1061 - &H1066	Tabelle der Tasten, auf die die Sonderzeichentaste wirkt
&H1067 - &H106C	Erzeugtes Zeichhen bei Sondertaste und einer der erlaubten Tasten
&H106D - &H1072	wie oben mit Shift Sondertaste
&H1073 - &H1078	wie oben mit Code Sondertaste
&H1079 - &H107E	wie oben mit Code Shift Sondertaste
&H107F - &H109D	Tabelle der Sonderbuchstaben, die auch durch CAP Feststellung betroffen werden
&H109E - &H10BC	Ergebnis von Sonderbuchstaben bei angeschaltetem CAP

Sollten Sie sich beim Umprogrammieren der Tastatur einmal hoffnungslos verstrickt haben, so können Sie immer mit >OUT &HA8,&HA0< ins ROM zurückspringen und der usrsprüngliche Zustand ist wiederhergestellt.
Aber nicht nur die Tastaturtabelle kann jetzt direkt geändert werden, auch die original Zeichendefinitionen stehen natürlich im ROM. Das Problem bisher war, daß jeder geänderte Zeichensatz durch ein >SCREEN< Befehl wieder gelöscht wurde. Bei >SCREEN< wird normalerweise der ROM Inhalt wieder ins VRAM kopiert. Die ROM Zeichentabelle steht ab Adresse &H1BBF bis &H23BE.
Wir wollen in das Leerzeichen einen Strich poken:

```
POKE &H1BBF+32*8+7,255
```

Bisher ist noch nichts geschehen. Wir müssen dem System noch mitteilen, daß jetzt die Kopie in Slot 2 steht. Diese Information steht an Adresse &HF91F.

```
POKE &HF91F,2
```

Die Adresse der Kopie steht an den Adressen &HF929/21. Sie ist in unserem Fall korrekt, d.h. noch immer &H1BBF.
Geben Sie jetzt >SCREEN 0< ein, und Sie erhalten den gewünschten Strich im Leerzeichen. Mit Hilfe der Adresse &HF91F und &HF920/21 kann auch, wenn das ROM eingeschaltet ist, eine alternative Zeichensatztabelle ins VRAM kopiert werden.

Vielleicht stört Sie auch manchmal das Fragezeichen beim >INPUT< Befehl.
Jetzt können wir es einfach durch ein anderes beliebiges Zeichen, z.B. Space (ASCII Code=32) überschreiben:

```
POKE &H23D0,32
```

Probieren Sie : INPUT A$
An die Stelle des Fragezeichens ist ein Leerzeichen getreten. Interessant ist auch >POKE &H23D0,29<. Damit beginnt die Eingabe exakt auf der aktuellen Position.

3.3 DER VDP ALS I/O GERÄT

Vom BASIC aus ist der Zugriff auf die VDP Register und auf den Video RAM mit >VDP<, >VPOKE< und >VPEEK< möglich. Intern wird der VDP und der VRAM jedoch als I/O Gerät behandelt. Die Port Adressen &H98 und &H99 beziehen sich auf den VDP. Die folgenden Informationen sind besonders für den Maschinensprache Programmierer interessant, da sie den Zugriff auf den VRAM mit maximaler Geschwindigkeit ermöglichen.
Es gibt vier grundsätzliche Operationen:

1 - Schreiben von Daten in den VRAM
2 - Lesen von Daten aus dem VRAM
3 - Schreiben von VDP Registerinhalten
4 - Lesen des Statusregisters

Zu 1: Der Datentransfer von der CPU in den VRAM über die VDP benutzt ein sogenanntes 14-Bit sich selbst erhöhendes (autoincrement) Adressenregister. Bei jedem Zugriff, also Lesen oder Schreiben des VRAM, wird die Adresse dieses VDP internen Registers erhöht. Um die Adresse auf einen bestimmten Wert zu setzen, sind zwei Datenübertragungen notwendig. Zuerst werden die 5 niederwertigen Bits der Adresse (das Low Byte) über Port &H99 ausgegeben. Dann folgen sofort danach die restlichen Bits (das High Byte). Bei der zweiten Übertragung muß, um anzuzeigen, daß eine VRAM Adresse festgelegt wird, Bit 6=1 und Bit 7=0 sein.

Beispiel:

Adresse &H2030 aus dem VDP übertragen:
Low Byte : &H30
High Byte: &H20

```
OUT &H99,&H30
OUT &H99,&H20 OR 2^6
```

Nachdem die Adresse übermittelt ist, kann über Port &H98 der Wert an dieser Adresse geschieben (>OUT<) werden. >OUT &H98,20< schreibt z.B. den Wert an die aktuellen Adressen. Durch den Zugriff über Port &H98 ist nun die Adresse automatisch auf &H2031 erhöht worden. Ein Erneutes >OUT &H988,...< würde also den Wert an diese Adresse schreiben. Müssen sehr viele Daten an aufeinanderfolgenden Adressen ins VRAM geschrieben werden, so bringt die Autoincrement Funktion des VDP einen erheblichen Geschwindigkeitsvorteil. Anstelle von jeweils drei Bytes braucht nur noch ein Byte übertragen zu werden.

zu 2: Der Vorgang des Adressenschreibens ist ähnlich, nur müssen diesmal Bit 6 und Bit 7 bei der zweiten Übertragung auf 0 gesetzt sein. Für >BY=VPEEK(&H1234)< kann man folgendes schreiben:

```
OUT &H99,&H34
OUT &H99,&H12
BY=INP(&H98)
```

Auch hier wurde durch >INP(&H98)< automatisch die Adresse auf &H1235 erhöht.

zu 3: Zuerst werden über Port &H99 die in den jeweiligen Registern zu schreibenden Daten übetragen. Dann wird über denselben Port die Registernummer übertragen, wobei Bit 7 gesetzt sein muß.

Für VDP(e)=4 kann man schreiben:

```
OUT &H99,4
OUT &H99,3 OR 2^7
```

zu 4: Zur Abfrage des Statusregister braucht nur ein >BY=INP(&H99)< Befehl ausgeführt werden. Dieser entspricht dem >BY=VDP(8)< Befehl.

Die Benutzung dieser Operationen vom BASIC aus ist möglich, führt aber manchmal zu Komplikationen, da der Interrupt die Datenübertragung stören kann

Drucker:

Für die Printerkommunikation sind die Ports &H90 und &H91 verantwortlich.
Zunächst werden die Daten mit >OUT< über Port &H91 geschickt. Über >INP(&H90)< wird gewartet bis das LSB 0 ist, was bedeutet, daß der Drucker empfangsbereit ist. Dann wird auch über Port &H90 das Strobe Signal an den Drucker geschickt.

PSG:

Auch der >SOUND< Befehl läßt sich als Folge von >OUT/INP< Befehlen darstellen. >OUT &HA0,Register< bestimmt das jeweilig betroffene Register.
>OUT &HA1,Wert< schreibt den Wert in das vorher bestimmte Register. Außerdem besitzt der PSG noch zwei I/O Ports, über die u.a. die Joysticks angeschossen sind.
Diese Ports werden über die Registernummern 14 (Port A) und 15 (Port B) selektiert. Dann kann über Portadresse &HA2 ihr Inhalt gelesen werden.

KAPITEL IV : SOUND MIT DEM SOUND- CHIP

4.1 EINFÜHRUNG

Um dem Soundgenerator der MSX Rechner Musik zu entlocken, ist einiges Grundwissen erforderlich. Dieses Wissen wollen wir Ihnen möglichst kurz und prägnant vermitteln:

Der Schall

Unter Schall verstehen wir alle hörbaren Schwingungsvorgänge. Sie sind für uns hörbar, wenn sie in dem Bereich von etwa 16 bis 20000 Hz liegen. Unterhalb des Hörfeldes liegt der Infraschall, oberhalb der Ultraschall. Der Schall läßt sich in Geräusche, Tongemische, Klänge und Töne gliedern. Ein Bereich des Schalls, mit einer nach Kultur und Epoche unterschiedlichen Verteilung, der oben genannten Erscheinungsformen, ist Musik.

Der Ton

Unter Ton versteht man in der Physik den reinen Sinuston, der aber, außer in der elektronischen Musik (synthetischer Musik), in der musikalischen Praxis nicht vorkommt. Was wir in unserer Sprache als Ton bezeichnen, gilt in der Akustik (Lehre vom Schall) als Klang.

Der Klang

Ein Klang entsteht durch Überlagerung einer Grundschwingung mit Schwingungen "harmonischer" Obertöne, die zur Grundschwingung im Verhältnis eines ganzzahligen Mehrfachen stehen.
Ein Instrument, wie z.B. das Klavier oder die Gitarre, erzeugt niemals einen Ton, so wie wir ihn vorher definiert haben, sondern immer nur Klänge. Dabei entsteht die typische Klangfarbe eines Instrumentes durch die Obertöne. Diese Ober-

wellenstruktur ist oftmals sehr kompliziert, auch deshalb, weil sie sich während des Spielens ändert. So ist z.B. der Anschlag eines Klaviers obertonreich, dagegen beim Abschwellen obertonarm.

Wie wir schon erwähnten, kann man mit elektronischen Geräten, und um ein solches handelt es sich beim Computer, reine Schwingungen, also einen physikalisch reinen Ton, erzeugen. Solche Töne klingen aber oft etwas "dünn". Um einen möglichst "breiten" Klang zu produzieren, stellt der Soundchip einige Funktionen zur Verfügung, die es uns ermöglichen, den Ton so zu verändern, daß er dem Hörgefühl eines Klanges ähnlich wird. Folgende Beeinflussungen stellt der PSG (Programmable Sound Generator) zur Benutzung zur Verfügung.

Veränderung der Tonhöhe

Es kann ein Ton erzeugt werden, der vom unteren Höhrbereich, ca. 27Hz, bis in den Ultraschallbereich (unhörbar) geht.

Veränderung der Geräuschfrequenz

Veränderung der Lautstärke

Klangeffekte

Klangeffekte werden durch verschiedene Muster der Lautstärkeänderung erzeugt.

Drei Tonkanäle

Es können drei Töne und Geräuschkanäle aufeinmal gespielt werden.

Zum Programmieren des Sound-Chip stehen folgende Befehle zur Verfügung:

4.2 Der >Play< Befehl

Mit diesem Befehl können Musikstücke nach Vorgabe der Tonhöhe, Tempo, Dauer, Laustärke, Pause und Form abgespielt werden. Die Beeinflussung der einzelnen Parameter wird durch sogenannte "Unterkommandos" erreicht.

Beispiel: Hey Jude/Beatles

```
10 T=80
20 T$="t=t;18":PLAY T$,T$+"14",T$
30 A1$="o5c4"
40 B1$="o5c"
50 C1$="r4"
60 A2$="o4a2r8ao5cdo4g2r4ga"
70 B2$="o3fffacccc"
80 C2$="o3fcfcfcfcecececef"
90 A3$="b-4o5f4r8fecdc16o4b-16a2r8o5c"
100 B3$="eb-b-gfccc8f8"
110 C3$="gcb-cb-cb-cfcfcfcfc"
120 A4$="dd4dg16fe16e16f16dc2o4fgao5d"
130 B4$="o4b-b-b-b-ffcf"
140 C4$="o2b-fb-fb-fb-fo3cfcfcfcf"
150 A5$="dcr8co4b-aeff4c2."
160 B5$="b-b-ecaaa2"
170 C5$="o3ccccccccfcfco2f2"
180 PLAY A1$,B1$,C1$
190 PLAY A2$,B2$,C2$
200 PLAY A3$,B3$,C3$
210 PLAY A4$,B4$,C4$
220 PLAY A5$,B5$,C5$
```

Programmbeschreibung

Zeile 10:

In dieser Zeile wird das Tempo mit der das Musikstück abgespielt wird festgehalten. Dabei kann >T< Werte von 32 bis 255 enthalten. Die Werte von 32 bis 255 sind mit der Anzahl der Viertelnoten, die in einer Minute gespielt werden gl ichzusetzen.

Zeile 20:

Zum Variabelhalten des Unterbefehls >Tn< wird innerhalb des Strings T=T gesetzt. Nach dieser Eingabe muß ein Semikolon stehen.
>L8< und >L4< sind Unterbefehle, die für die Länge der zu spielenden Noten stehen. Im Unterkommando >Ln< können Werte zwischen 1 und 64 enthalten sein. Diese Werte haben folgende Bedeutung:

1 = eine ganze Note
2 = eine halbe Note
3 = eine drittel Note
4 = eine viertel Note
.
.
.

Da in dem Musikprogramm viele Viertel- und Achtelnoten vorkommen, haben wir die Unterkommandos >L4< (Viertel) und >L8< (Achtel) in T$ gespeichert. Damit erreicht man eine weniger umständliche Eingabe in A$, B$ und C$. Dadurch, daß die oft benötigten Längen (>Ln<) in der Variablen T$ festgelegt sind, werden diesbezügliche Befehle in den Stimmenstrings nicht mehr benötigt. Dieser Trick ist insofern nützlich, da er die Stringzeilen übersichtlicher und die Eingabe einfacher gestaltet.

Zeile 30 bis 50:

In diesen Zeilen ist der Auftakt zu allen drei Stimmen enthalten. Die Variable >O< steht dabei für die Oktave. Der Unterbefehl >On< kann Werte von 1 bis 8 enthalten. O4 ist die Oktave vom sog. "Schlüsselloch C" beim Klavier an aufwärts. In der Musik nennt man sie auch die "eingestrichene Oktave".
Der Unterbefehl >C4< steht für den Ton C mit einer Tondauer von einem Viertel.
In Zeile 50 steht >R4< für eine Viertelpause.

Zeile 60 bis 170:

Enthalten in A$ die Ober-, in B$ die Mittel- und in C$ die Unterstimme. Um die Synchronisation der verschiedenen Stimmen zu gewährleisten, ist es ratsam, für jeden Takt eine Zeile zu verwenden.

Zeile 180 bis 190:

Hier werden die in den Stringvariablen enthaltenen Töne nach der erwünschten Reihenfolge aufgerufen.

4.3. Der >SOUND< Befehl

Mit diesem Befehl ist man in der Lage direkt in das PSG Register zu schreiben. Dadurch sind spezielle Geräusch- und Toneffekte zu erzielen. Nur mit dem >SOUND< Befehl ist es möglich die vollen Klangmöglichkeiten des PSG auszunutzen. Ein Vorteil des >SOUND< Statements liegt darin, daß der erzeugte Ton bzw. das erzeugte Geräusch solange klingt, bis das entsprechende Register mit einer neuen >SOUND< Anweisung wieder ausgestellt wird.

Der PSG (Programmable Sound Generator) stellt insgesamt 13 Register zur Verfügung:

Bit	7	6	5	4	3	2	1	0
Register 0	Low Byte von Kanal A							
Register 1	0	0	0	0	High Byte Ka. A			
Register 2	Low Byte von Kanal B							
Register 3	0	0	0	0	High Byte Ka. B			
Register 4	Low Byte von Kanal C							
Register 5	0	0	0	0	High Byte Ka. C			
Register 6	0	0	0	Geräuschfrequenz				
Register 7	0	0	GC	GB	GA	TC	TB	TA
Register 8	0	0	0	La	Lautstärke A			
Register 9	0	0	0	La	Lautstärke B			
Register 10	0	0	0	La	Lautstärke C			
Register 11	LB Frequ.Laut.änderungsmuster							
Register 12	HB Frequ.Laut.änderungsmuster							
Register 13	0	0	0	0	Laut.muster			

Abkürzungen:

LB - Low Byte (siehe Kapitel:Maschinensprache)
HB - High Byte
GC,GB,GA - Geräusch einschalten für Kanal A,B,C
La - Bei Einstellung von dem Wert 16: Änderung des Lautstärkemusters anhand der eingestellten Hüllkurve
Frequ. - Frequenz

Register 0 bis 5:

In diesen Registern kann die Frequenz der Töne in der Reihenfolge Low Byte (niederwertiges Byte=Register 0), High Byte (höherwertiges Byte=Register 1) gespeichert werden.
Um die Frequenz der drei Register zu berechnen, wird die Taktfrequenz des Rechners (3.5MHz) zugrunde gelegt.

```
3 579 545              (Hz)
-------------------------- =  256*(Daten von Register 1,3,5)
32 * Ausgabefrequenz (Hz)    + (Daten von Register 0,2,4)
```

Die Formel läßt sich folgendermaßen vereinfachen:

```
111 860.8         (Hz)
---------------------- =  256* (Daten von Register 1,3,5)
Ausgabefrequenz (Hz)      +(Daten von Register 0,2,4)
```

Wenn wir z.B. den Kammerton "A"(440Hz) erzeugen wollen, findet folgende Berechnung statt:

```
111860,8 (Hz)
-------------= 254= 256*0+254
   440   (Hz)
```

Wir setzen nun diese errechneten Werte jeweils in die beiden Register, die für die Tonfrequenz eines Kanals zuständig sind, z.B. Register 0 und 1. Die Werte betragen 0 für das Register 1 und 254 für das Register 0.
Mit dem >Sound< Befehl können wir nun diese beiden Werte übergeben.

```
Sound 0,254
Sound 1,0
```

Es sind Frequenzen vom unteren Hörbereich bis zum Ultraschallbereich möglich.
In den Register 0,2,4 können Werte von 0 bis 255 enthalten sein.
Die Register 1,3,5 können Werte von 0 bis 15 enthalten.

Register 6:

Dieses Register bestimmt die Geräuschfrequenz. Sie läßt sich durch folgendes Verfahren berechnen:

$$\text{Datenwert von Register 6} = \frac{3\,579\,545 \text{ (MHz)}}{32 * \text{Geräuschfrequenz (Hz)}}$$

Der Datenwert kann 0 bis 31 betragen.
Vereinfacht sieht die Berechnung so aus:

$$\text{Datenwert} = \frac{111\,860.8 \text{ (Hz)}}{\text{Geräuschfrequenz (Hz)}}$$

Wenn wir eine Geräuschfrequenz von z.B. 8000Hz erzeugen wollen, müssen wir in dieser Weise vorgehen:

$$\frac{111\,860.8 \text{ (Hz)}}{8000 \text{ (Hz)}} = \text{ca.14 (Datenwert)}$$

In das Register 6 muß ein Wert von 14 geschrieben werden, um ein Geräusch mit der Frequenz von 8000Hz zu erhalten.
Register 6 darf Werte zwischen 0 und 31 enthalten. Damit sind Frequenzen von 3500 bis 111860Hz möglich.

Register 7

Dieses Register wählt einen Kanal für die Ton- und Geräuscherzeugung. Es können Werte von 0 bis 63 übergeben werden.

	Geräusch			Ton		
Kanal	C	B	A	C	B	A
Wert	32	16	8	4	2	1

Wenn wir z.B. Kanal A mit Ton, Kanal B nur mit Geräusch und Kanal C mit Ton und Geräusch belegen wollen, müssen wir folgende Berechnung anstellen:

Max.-(Kanal+Kanal+Kanal)= Datenwert

Max.= maximaler, in diesem Register darstellbarer Wert (63).

Für unser Beispiel:

63-(1+16+32+4)= 10

Diesen Wert schreiben wir nun ins Register 7.

```
>SOUND 7,11<
```

Für Fortgeschrittene bleibt noch zu erwähnen, daß dieses Register Low orientiert ist. Das heißt, der Zustand 1 eines Bits bedeutet AUS und 0 bedeutet EIN.
Im Gegensatz dazu gibt es die High orientierten Register, bei denen der Zustand 1 EIN bedeutet und der Zustand 0 AUS. Die High orientierte Registeransteuerung ist dabei die Normalform.

Register 8-10:

Diese Register ermöglichen eine Regulierung der Lautstärke im Wertebereich von 0 bis 15.

Register 8 = Kanal A
Register 9 = Kanal B
Register 10 = Kanal C

Wird der Wert 16 in eines der Register geschrieben, dann wird mit Hilfe von Register 11, 12 und 13 ein bestimmter Lautstärkeverlauf bestimmt

Register 11 und 12:

Hier wird die Frequenz für die Änderung des Musters der Lautstärke angegeben. Dadurch kann man z.B. einen Ton so mit dem Muster überlagern, daß ein Schwingungszustand entsteht, dessen Klang "überirdischen" Charakter bekommt. Der Ton erweckt den Eindruck zu schweben.
Zur Berechnung der Periode der Lautstärkeveränderung wird folgender Ausdruck verwendet:

$$\frac{3\,579\,545\ (\text{Hz})}{532*\text{Periode}\ (\text{Hz})} = 256*(\text{Daten von R12})+(\text{Daten von R11})$$

R - Register

Vereinfacht lautet der Ausdruck:

$$\frac{6728.5\ (\text{Hz})}{\text{Periode}\ (\text{Hz})} = 256*(\text{Daten von R12})+(\text{Daten von R11})$$

Es ist ein Frequenzbereich von 0,1 bis 7000Hz möglich.
Wenn wir z.B. das Aufeinanderfolgen der Lautstärkeänderungsmuster mit einer Frequenz von 20Hz realisieren wollen, muß folgende Berechnung angestellt werden.

$$\frac{6728.5\ (Hz)}{20\ \ (Hz)} = Ca.336 = 256*1+80$$

In das Register 11 wird also 80 und in Register 12 eine 1 geschrieben.

Zum besseren Verständnis der zur Tonerzeugung erforderlichen Parameter, folgt ein Beispielprogramm.
Geben Sie bitte folgendes ein und starten das Programm mit >RUN<:

Programm:Sinuskurve

```
10 REM Sinuskurve
20 SCREEN 2:COLOR 1,15,7
30 OPEN"grp:" FOR OUTPUT AS #1
40 LINE (15,10)-(245,180),1,B
50 LINE (235,95)-(20,95),8
60 LINE (20,20)-(20,150),1
70 PSET (90,2),0
80 PRINT#1,"Sinuskurve"
90 PSET(23,20),0
100 PRINT#1,"y"
110 PSET (237,95),0
120 PRINT#1,"X"
130 PSET (30,160),0
140 PRINT#1,"X=Zeit"
150 PSET (30,150),0
160 PRINT#1,"Y=Elongation"
170 PSET (80,30),0
180 PRINT#1,"Wellenlaenge"
190 PSET(210,70),0
200 PRINT#1,"Amp"
210 PSET (30,170),0
220 PRINT#1,"Amp=Amplitude"
230 LINE (60,40)-(60,100),3
240 LINE (185,40)-(185,100),9
250 LINE (60,40)-(185,40)
260 LINE (220,50)-(220,65),10
270 LINE (220,80)-(220,95),10
280 PSET(20,125),0
290 FOR X=20 TO 225 STEP 3
300 Y=90+SIN(X/20)*40
310 LINE -(X,Y),2
320 NEXT X
330 IF INKEY$="" THEN 330
```

Programmbeschreibung:

Nachdem Sie das Programm gestartet haben, wird eine Sinuskurve auf dem Bildschirm gezeichnet. Wie wir schon erwähnten, besteht ein Ton aus Schwingungen. Eine solche Schwingung soll das Bild auf dem Monitor darstellen.

Begriffserklärung:

Elongation:

Der Ausschlag auf der Y-Achse vom Nullpunkt aus wird als Elongation bezeichnet.

Tondauer:

Die Tondauer wird durch die Werte auf der X-Achse angegeben.

Amplitude:

Die Amplitude stellt den größtmöglichen Schwingungsausschlag einer Kurve dar. Je größer die Amplitude einer Schwingung ist, desto lauter ist der Ton und umgekehrt.

Wellenlänge:

Eine Wellenlänge ist ein bestimmtes Schwingungsmuster das sich periodisch wiederholt.

Register 13:

Dieses Register dient der Auswahl eines bestimmten Musters zur Lautstärkeänderung. Diese Muster nennt man auch Hüllkurven, da sie einen Grundton mit einer bestimmten Schwingung und Ampitudengröße sozusagen einhüllen, und seine Amplitudengröße beeinflussen. Damit ändert sich die Lautstärke in einer durch das Muster der Hüllkurve vorgeschriebenen Form.

Register 13 erlaubt uns nun die Auswahl von insgesamt acht verschiedenen Hüllkurven, die alle in etwas anderer Form den Lautstärkeverlauf beeinflussen. Die verschiedenen Formen der Hüllkurven finden Sie in Ihrem Handbuch.
Geben Sie nun folgendes Programm ein und starten es mit >RUN<:

Programm:Enthüllung

```
10 MAXFILES=1:REM Enthuellung
20 OPEN "grp:" FOR OUTPUT AS #1
30 DEFINT X,Y:DEFSNG S
40 SCREEN 2: COLOR 1,15,7
50 LINE (15,10)-(245,180),1,B
60 LINE (20,15)-(20,170),8
70 PRESET (30,180)
80 PRINT#1,"Huellkurve (Saegezahn)"
90 PRESET (21,15)
100 PRINT#1,"y=Lautstaerke"
110 PRESET (150,170)
120 PRINT#1,"x=Zeit"
130 PRESET (238,95)
140 PRINT#1,"X"
150 LINE (236,95)-(20,95),8
160 LA=50:REM Wellenlaenge der Huellkurve
170 AN=200/LA: REM Anzahl der Zaehne auf der x-Achse
180 FY=60/LA:REM Streckfaktor y Achse
190 FOR I=1 TO AN STEP 2
200 LINE -(20+LA*I,95-LA*FY),2:REM Flanke
210 LINE -(20+LA*(I+1),95),2:REM Diagonale
220 NEXT I
230 FOR X=20 TO 236 STEP 2
240 R=(X-20) MOD (LA*2):REM Rest zu LA * 2
250 AM=R*SGN(LA-.1-R)-LA*2*(R>=LA)
260 SY=SIN((X-20)/3):REM Sinusamplitude
270 Y=95-SY*AM*FY:REM y Gesamtamplitude
280 LINE -(X,Y),1
290 NEXT X
300 CLOSE #1
310 IF INKEY$="" THEN 310
```

Programmbeschreibung

Dieses Programm zeigt Ihnen, wie die Hüllkurve funktioniert. Wir haben eine Dreiecksschwingung gewählt (14te Möglichkeit mit Unterkommando im Handbuch).

Wenn Sie folgende Zeilen ändern, erhalten Sie ein Muster wie die 12te Möglichkeit im Handbuch:

```
120 LINE -(20+LA*(I-1),95-LA),2:REM Flanke
130 LINE -(20+LA*II,95),2:REM Diagonale
160 AM=LA-(X-20)MOD LA:REM aktuelle Amplitude der Hüllkurve
```

Bei Veränderung folgender Zeilen erhalten Sie die 8te Möglichkeit:

```
130 LINE -(20+LA*I,95-LA*FY),2:REM Flanke
140 LINE -(20+LA*(I+1),95),2:REM Diagonale
170 R=(X-20) MOD (LA*2):REM Rest zu LA*2
180 AM=R*SGN(LA-.1-R)-LA*2*(R>=LA)
```

Probieren Sie auch verschiedene Werte für folgende Variablen:

LA - Wellenlänge der Hüllkurve
AN - Anzahl der Zähne auf der X-Achse
FY - Streckfaktor Y-Achse
SY - Sinusamplitude
Y - Gesamtamplitude

Durch die Hüllkurven sind einige Manipulationen der Töne möglich. Versuchen Sie doch durch Ausprobieren einige interessante Töne oder Geräusche zu erzeugen.

Programm: Orgel

Das folgende Programm realisiert die Tastatur einer Orgel. Dabei gilt:

Taste drücken : A W S E D F G Y H U J K

Ergibt folgende Töne : C Cis D Dis E F G Gis A B H C

Das Besondere an diesem Programm ist, daß Akkorde (3 Töne gleichzeitig) gespielt werden können.

```
10 DEFUSR1=&HD12
20 SCREEN 0,,0
30 DEFINT A-Z
40 KB=&HFBF0
50 PP=&HF3F8
60 GP=&HF3FA
70 DIM F(25)
80 FOR I=65 TO 90
90 READ F(I-65):NEXT
100 SOUND 7,63:FOR I=8 TO 10:SOUND I,10:NEXT
110 P=10:GOSUB 190
120 A=PEEK(PP)-&HF0-P:IF A=0 THEN GOSUB 190:GOTO 170
130 IF A>3 THEN A=3
140 GOSUB 190
150 FOR I=0 TO A-1:T=(PEEK(KB+I)AND&B11011111)-65
160 SOUND 2*I,F(T)MOD256:SOUND 2*I+1,F(T)ö256:NEXT
170 SOUND 7,64-2^(A)
180 GOTO 120
190 P=10-P
200 POKE PP,&HF0+P:POKE PP+1,&HFB
210 POKE GP,&HF0+P:POKE GP+1,&HFB
220 POKE &HF3F7,1
230 FOR W=0 TO 3:X=USR1(1):NEXT
240 RETURN
250 REM A-H
260 DATA 1710,0,0,1357,1439,1281,1141,1016
270 REM I-P
280 DATA 0,906,855,1,0,0,1,1
290 REM Q-Z
300 DATA 0,0,1524,1209,960,0,1615,0,1078,0
```

Programmbeschreibung: Orgel

Zeile 10:

Startadresse der Routine, die die Tastaturabfrage ausführt. Wird normalerweise automatisch per Interrupt aufgerufen.

Zeile 40:

KB ist die Startadresse des Keypufferspeicher

Zeile 50:

PP ist die sog. Adresse Put Point, an der die Adresse des aktuellen Endes des Keypuffers steht.

Zeile 60:

GP ist die Adresse Get Point, an der die Adresse des aktuellen Anfangs des Keypuffers steht.

Zeile 70:

F enthält die, der jeweiligen Taste zugeordnete Frequenz.

Zeile 80,90:

In diesen Zeilen wird den oben beschriebenen Tasten die jeweilige Frequenz zugeordnet (DATA Zeile 260 bis 300). 0 steht für nicht belegte Taste.

Zeile 100:

Alle Kanäle an und Lautstärke für alle Kanäle auf 10.

Zeile 110:

P ist die Differenz vom ersten zum zweiten benutzen Puffer. Zwei Keypuffer sind notwendig, damit eine neue Tastaturbetätigung in einem Puffer registriert wird, währen der andere noch angelassen wird. Das Unterprogramm ab Zeile 190 initialisiert einen Keypuffer und ruft die Tastaturabfrage auf.

Zeile 120:

Beginn des Hauptprogramms.
A enthält die Anzahl der gedrückten Tasten.
Wenn A=0, dann Tastaturabfrage (GOSUB 1900 und alle Kanäle aus (GOTO 170).

Zeile 130:

Die nächsten 3 gedrückten Tasten berücksichtigen.

Zeile 140:

Tastatur abfragen

Zeile 150:

Die gedrückte Taste der Reihe nach auslesen.

Zeile 160:

Entsprechend der jeweiligen Taste die Frequenz setzen.

Zeile 170:

und die entsprechenden Kanäle anschalten

Zeile 180:

Zurück zum Anfang des Hauptprogramms.

Zeile 190:

Unterprogramm Tastaturabfrage.
Bei jeder Abfrage wechselt P von 10 nach 0 und umgekehrt. Dadurch wird ständig zwischen den beiden Puffern gewechselt.

Zeile 200,210:

Keypuffer löschen und auf Anfangsadresse setzen.

Zeile 220:

Wiederholungsverzögerung ausschalten.

Zeile 230:

3 mal Tastaturabfragen

Programm: Syntheziser

Abschließend folgt ein Synthesizerprogramm, mit dem Sie sämtliche >SOUND< Möglichkeiten der MSX Rechner voll ausschöpfen und auf einfache Weise ausprobieren können.

Das Programm wird sehr komfortabel über Joysticksteuerung bedient. Es ist möglich in dem auf dem Bildschirm erscheinenden Menue frei herumzulaufen. Wenn Sie einen der Werte ändern wollen, drücken Sie den Feuerknopf. Nun können Sie ebenfalls über den Joystick die Werte durch Bewegung nach rechts erhöhen oder durch Bewegung nach links erniedrigen. Es erscheint zum Anzeigen dieses Modus ein Stern links über dem gewählten Begriff. Um wieder in den Normalmodus zu gelangen, müssen Sie nochmals den Feuerknopf betätigen. Zur Anzeige des Verlassens des Eingabemodus verschwindet der Stern sofort vom Bildschirm.

```
10 REM Synthesizer
20 DEFINT A-Z
30 DEFSNG W,D,M,F
40 DIM W(7,3)
50 DIM X(8,4),Y(8,4)
60 SCREEN 1:WIDTH 29
70 TF#=1789772.5#
80 MF(2)=&HFFF:MF(5)=&H1F:MF(6)=2^16-1
90 PRINT"     Kanal 1  Kanal 2  Kanal 3"
100 PRINT"Ton: Aus       Aus       Aus"
110 FOR I=1 TO 3:W(1,I)=1:NEXT:REM Aus
120 PRINT:PRINT"Fre:"
130 FOR I=1 TO 3:W(2,I)=300:NEXT:REM Vorgabefrequenz
140 PRINT:PRINT"Ger: Aus       Aus       Aus"
150 FOR I=1 TO 3:W(3,I)=1:NEXT:REM Aus
160 PRINT:PRINT"Vol:"
170 FOR I=1 TO 3:W(4,I)=10:NEXT:REM Vorgabelautstaerke
180 PRINT:PRINTSTRING$(29,"-")
190 PRINT"         Geraeusch"
200 PRINT:PRINT"Fre:"
210 W(5,1)=10:REM Vorgabegeraeuschfrequenz
220 PRINT:PRINT"         Huellkurve"
230 PRINT:PRINT"Fre:"
240 W(6,1)=10
250 PRINT:PRINT"Num:"
260 W(7,1)=9:REM Huellkurvennummervorgabe
270 FOR I=1 TO 4
280 FOR J=1 TO 3
290 X(I,J)=-4+9*J
300 Y(I,J)=2*I
310 NEXT J,I
320 X(5,1)=5:Y(5,1)=14
330 X(6,1)=5:Y(6,1)=18
340 X(7,1)=5:Y(7,1)=20
350 FOR I=5 TO 7
360 FOR J=2 TO 3
370 X(I,J)=0:Y(I,J)=0
380 NEXT J,I
```

```
390 Y=4
400 FOR X=1 TO 3
410 LOCATE X(Y,X)-1,Y(Y,X),0
420 PRINTW(Y,X);
430 NEXT
440 X=1:Y=7:LOCATE X(Y,X)-1,Y(Y,X),0
450 PRINTW(Y,X);
460 SOUND 13,W(7,1)
470 Y=2
480 FOR X=1 TO 3
490 GOSUB 1120:NEXT
500 X=1:Y=5:GOSUB 1190
510 SOUND 7,63:R7=63
520 FOR X=1 TO 3:SOUND 7+X,W(4,X):NEXT
530 X=1:Y=6:GOSUB 1250
540 X=1:Y=1
550 ON STRIG GOSUB 700,700,700,700,700
560 EI=1
570 STRIG(EI) ON
580 GOTO 600
590 IF DX=0 AND DY=0 THEN 620
600 LOCATE X(Y,X),Y(Y,X),1
610 DX=0:DY=0
620 R=STICK(EI)
630 IF R=8 OR R=1 OR R=2 THEN DY=-1
640 IF R>3 AND R<7 THEN DY=1
650 IF R>1 AND R<5 THEN DX=1
660 IF R>5 AND R<=8 THEN DX=-1
670 IF X(Y+DY,X)=0 THEN DY=0
680 IF X(Y,X+DX)=0 THEN DX=0
690 X=X+DX:Y=Y+DY:GOTO 590
700 REM Strig
710 IF Y<>1 AND Y<>3 THEN 790
720 W(Y,X)=1-W(Y,X)
730 LOCATE X(Y,X),Y(Y,X),0
740 IF W(Y,X)=0 THEN PRINT"Ein"; ELSE PRINT"Aus";
750 BI=X-(Y=3)*3-1
760 R7=(R7 AND (NOT2^BI)) OR (W(Y,X)*2^BI)
```

```
770 SOUND 7,R7
780 GOTO 1100
790 IF Y<>4 AND Y<>7 THEN 950
800 IF LF THEN LF=0:RETURN ELSE LF=-1
810 LOCATE X(Y,X)-1,Y(Y,X)-1,0:PRINT"*";
820 FOR WA=0 TO 1000:NEXT
830 A=PEEK(&HF3E8)
840 IF (A AND 16) =0 THEN 1100
850 R1=STICK(EI):IF R1=0 THEN 830
860 IF R1>1 AND R1<5 THEN D=1
870 IF R1>5 AND R1<9 THEN D=-1
880 W=W(Y,X)+D
890 IF W>(16+2*(Y=7)) OR W<0 THEN 830
900 W(Y,X)=W
910 IF Y=4 THEN SOUND 7+X,W ELSE SOUND 13,W
920 LOCATE X(Y,X)-1,Y(Y,X),0
930 PRINTW;
940 GOTO 830
950 REM Frequenzen
960 IF VF THEN VF=0:RETURN ELSE VF=-1
970 LOCATE X(Y,X)-1,Y(Y,X)-1,0:PRINT"*";
980 FOR WA=0 TO 1000:NEXT
990 DP=-1:DM=1:D=0
1000 A=PEEK(&HF3E8)
1010 IF (A AND 16) =0 THEN 1100
1020 R1=STICK(EI):IF R1=0 THEN DP=-1:DM=1:D=0:GOTO 1000
1030 IF R1>1 AND R1<5 THEN D=D+DP:DP=DP*2
1040 IF R1>5 AND R1<9 THEN D=D+DM:DM=DM*2
1050 W=W(Y,X)+D
1060 IF W<1 OR (W>MF(Y)) THEN 990
1070 W(Y,X)=W
1080 IF Y=2 THEN GOSUB 1120 ELSE IF Y=5 THEN GOSUB 1190
ELSE GOSUB 1250
1090 GOTO 1000
1100 LOCATE X(Y,X)-1,Y(Y,X)-1,0:PRINT" ";
1110 LOCATE X(Y,X),Y(Y,X),1:RETURN
1120 REM Freqyenz Reg Calc.
1130 W=W(2,X)
```

```
1140 F=TF#/16/W
1150 SOUND (X-1)*2,W-INT(W/256)*256
1160 SOUND (X-1)*2+1,INT(W/256)
1170 GOSUB 1310
1180 RETURN
1190 REM Geraeusch Reg Calc.
1200 W=W(5,1)
1210 F=TF#/16/W
1220 SOUND 6,W
1230 GOSUB 1310
1240 RETURN
1250 REM Huell Freq Reg Calc.
1260 F=TF#/256/W
1270 SOUND 11,W-INT(W/256)*256
1280 SOUND 12,INT(W/256)
1290 GOSUB 1310
1300 RETURN
1310 A$=LEFT$(STR$(F)+"      ",7)
1320 LOCATE X(Y,X)-1,Y(Y,X),0
1330 PRINTA$;
1340 RETURN
```

Programmbeschreibung:

Zeile 70:

FF# enthält die halbe Taktfrequenz des Rechners.

Zeile 80:

MF enthält die Maximalregisterwerte für

Tonfrequenz	MF(2)
Geräuschfrequenz	MF(5)
Hüllkurvenfrequenz	MF(6)

Zeile 90 bis 260:

Bildschirmaufbau und Definition der zu den jeweiligen Feldern gehörenden Daten.
W(I,J) enthält die Daten der Iten Eingabezeile und der Jten Spalte.
Bei Ein/Aus bedeutet der Wert 1 aus, der Wert 0 ein. Bei allen anderen Werten entspricht W(I,J) den jeweiligen Registerinhalten.

Zeile 270 bis 380:

Um das einfache Hin- und Herspringen mittels Joystick zwischen den Eingabefeldern zu ermöglichen, wird hier jedem Eingabefeld Ite Eingabezeile, Jte Eingabespalte die dazugehörige Position X(I,J),Y(I,J) auf dem Bildschirm zugeordnet. 0 bedeutet ungültiges Feld.

Zeile 390 bis 530:

Hier werden die beim Einschalten vorgegebenen Werte von W(I,J) in die Frequenzwerte bzw. Lautstärkewerte umgerechnet und auf dem Bildschirm angegeben.

Zeile 390 bis 430:

Ausgabe der Lautstärke

Zeile 440 bis 460:

Ausgabe und Setzen der Hüllkurvennummer

Zeile 470 bis 490:

Ausgabe und Setzen der Tonfrequenzen

Zeile 500:

Geräuschfrequenz setzen und ausgeben

Zeile 510

Kanäle anschalten

Zeile 520:

Lautstärkewerte setzen

Zeile 530:

Hüllkurvenfrequenzen setzen und ausgeben

Zeile 540:

Koordinaten des ersten Eingabefeldes

Zeile 600:

Cursor anzeigen

Zeile 620:

Ist Joystick bewegt? Nein, dann warten.

Zeile 630 bis 660:

feststellen der Richtung

Zeile 670 bis 680:

Prüfen, ob Eingabefeld in die angeforderte Richtung erlaubt ist. Wenn nicht X(..)=0, dann Bewegung nicht ausführen.

Zeile 690:

Neues Eingabefeld setzen und von vorne beginnen

Zeile 710:

Wenn nicht Eingabe Zeile 1 (Ton Ein/Aus) oder Zeile 3 (Geräusch Ein/Aus) dann weiter.

Zeile 720:

Wert (Zustand Ein/Aus bzw. 0/1) umdrehen

Zeile 730:

Cursor ausschalten

Zeile 740:

Neuen Zustand angeben

Zeile 750:

Bitnummer für das >SOUND< Register berechnen.

Zeile 760:

Byte für Register 7 berechnen

Zeile 780:

Ende Strig

Zeile 790:

Wenn nicht Eingabe Zeile 4 (Volumen) oder 7 (Hüllkurvennummer)

Zeile 800:

Wenn Interrupt durch das "zweite Mal feuern" ausgelöst, LF zurücksetzen und Interrupt nicht beachten. Ansonsten LF setzen, damit nächster Interrupt das Ende des Eingabemodus bewirkt.

Zeile 810:

Stern ausgeben als Kennzeichen für Eingabe

Zeile 820:

Warten, bis Feuerknopf losgelassen wird

Zeile 830,840:

Feuerknopf zum zweiten Mal gedrückt? Wenn ja, dann Ende der Eingabe.

Zeile 850 bis 880:

Stickrichtung bestimmen

Zeile 890:

Test auf Grenzen, 0-16 bei Volumen und 0-14 bei Hüllkurvennummer

Zeile 900 bis 930:

Neuen Wert setzen und ausgeben

Zeile 950:

Frequenzveränderung annehmen

Zeile 960 bis 1090:

Analog wie Zeile 800 bis 930.
Unterschied ist, daß die Änderungsgeschwindigkeit bei Beibehalten einer Richtung immer schneller wird. Dazu werden DP bzw. DM jeweils mit 2 multipliziert.

Zeile 1100:

Ende der Strig Routine. Der Stern wird gelöscht

Zeile 1110:

und der Cursor wird wieder auf dem Eingabefeld angezeigt.

Die folgenden Zeilen enthalten Unterprogramme. Ihre Funktion ist anhand der REM Zeilen verdeutlicht.

KAPITEL V : M A S C H I N E N S P R A C H E

5.1 Warum eigentlich Maschinensprache?

Die meisten Homecomputer sind mit BASIC ausgerüstet. Wie Sie sicher gemerkt haben, ist diese Sprache nicht schwer zu erlernen. Besonders das MSX BASIC fällt durch seine Vielzahl von Befehlen auf. Es ensteht der Eindruck, daß mit diesem BASIC keine Wünsche offen bleiben und alle Programmierprobleme damit gut gelöst werden können.
Geben Sie bitte einmal folgendes ein, und beachten Sie die Zeit.

```
5 SCREEN 2
10 HL=&H2000
20 A=1
30 VPOKE HL,A
40 HL=HL+1
50 IF HL<&H3000 THEN 20
60 IF INKEY$="" THEN 60
70 RETURN
```

Starten Sie das Programm mit >RUN< und schauen sich an was geschieht. Wenn Sie zurück zur Eingabe wollen, drücken Sie irgendeine Taste.

Das nächste Programm lädt das Maschinenprogramm mit der gleichen Aufgabe, wie das BASIC Programm:

```
10 CLEAR 200,&HEFFF
20 FOR I=&HF000 TO &HF014
30 READ A
40 POKE I,A
50 NEXT I
60 DEFUSR1=&HF000
70 END
80 DATA &HCD,&H72,&H00,&H21,&H00,&H20,&H3E,&H01
90 DATA &HCD,&H4D,&H00,&H23,&H3E,&H30,&HBC
100 DATA &H20,&HF5,&HCD,&H9F,&H00,&HC9
```

Laden Sie nun das Maschinenprogramm mit >RUN<, rufen dann das so geladene Maschinenprogramm mit >X=USR1(1)< auf und wundern sich!

Wie Sie gesehen haben, läuft das:

- BASIC Programm : ca.43 Sekunden
- Maschinenprogramm : weniger als 1 Sekunde

Die Länge beträgt für das:

- BASIC Programm : 97 Bytes
- Maschinenprogramm : 21 Bytes
  nämlich von &HF000 bis &HF014.

Zur Analogie der Programme:

BASIC		Assemblersprache
5 SCREEN 2	-	CALL &H0072
10 HL=&H2000	-	LD HL,&H2000
20 A=1	-	LD A,1
30 VPOKE HL,A	-	CALL &H004D
40 HL=HL+1	-	INC HL
50 IF HL<&H3000 THEN 20	-	LD A,&H30
	-	CP H
	-	JR NZ,-11
60 IF INKEY$="" THEN 20	-	CALL &H009F
70 RETURN	-	RET

ERKLÄRUNG:

Zeile 5: CALL &H0072 ruft die Routine auf, die den Grafikmodus >SCREEN 2< einschaltet.

Zeile 10: Hier wird der Wert für die VARIABLE HL bzw. das REGISTER HL auf den Anfang des Farbspeichers gesetzt. (LD=engl.load=laden)

Zeile 20: In dieser Zeile wird in A der Wert der auszugebenden Farbe (1=schwarz) gespeichert.

Zeile 30: Hier wird der Wert von A in den Bildschirmspeicher geschrieben, was bewirkt, daß ein schwarzer Strich angezeigt wird.

Probieren Sie doch einfach einmal unterschiedliche Werte für die Adresse HL im Farbspeicher (HL darf zwischen &H2000 und &H3800 liegen !!) und für A den Code für die Farbe einzusetzen.

Zeile 40: Hier wird die Variable HL, die die Adresse im Farbspeicher enthält, um eins erhöht, damit nach und nach der ganze Bildschirm vollgeschrieben wird (engl.:INCrease:erhöhen).

Zeile 50: Abfrage ob HL größer als &H3000 ist, also, ob das Ende des Farbspeicherbereichs erreicht ist. Diese Abfrage muß in Maschinensprache in drei Befehle aufgeteilt werden:
LD (engl.load=laden); laden von A mit dem Vergleichswert (=High Byte von &H3000).
CP H: Vergleich mit H, dem High Byte von HL (CP: engl. compare:vergleiche).
JR (engl.jump relativ: relativer Sprung); NZ (engl.non zero: nicht Null). Man kann also sagen: "Springe, wenn nicht Null" um die angegebene Distanz.

Zeile 60: Tastaturabfrageroutine mit CALL &H009F aufrufen. Wenn eine Taste gedrückt wird, erfolgt Rücksprung.

Zeile 70: RET beendet das Unterprogramm.

Im Folgenden zeigen wir das Assemblerlisting, um Ihnen ein Beispiel zu geben:

ASSEMBLERLISTING zum Maschinenprogramm

```
Adresse Code     Zeilen-Nr. Assemblerbefehl  Kommentar

F000 CD7200       5         CALL &H0072 ; Routine zum
Einschalten des Grafikmodus >SCREEN 2<
F003 210020       10        LD   HL,&H2000 ; Start
Farbspeicher
F006 3E01         20        LD   A,&H01 ; Zeichenfarbe schwarz
(=1)
F008 CD4D00       30        CALL &H004D ; Routine zur Ausgabe
eines Zeichens auf dem Bildschirm
F00A 23           40        INC  HL ; Farbspeicheradresse
erhöhen
F00B 3E30         50        LD   A,&H30
F00D BC           60        CP   H ; H>&H30 ?
F00E 20F5         70        JR   NZ,&HF003 ; nein,dann noch
mal
F011 CD9F00       80        CALL &H009F ; Routine zur
Tastaturabfrage
F014 C9           90        RET ; Zurück zum BASIC
```

Die Vorteile der Maschinensprache liegen also klar auf der Hand. Es sind die Geschwindigkeit mit der Programme ablaufen und der sparsame Verbrauch von Speicherplatz. Einige Probleme lassen sich fast nur mit Maschinensprache effektiv lösen z.B. Textverarbeitung, schnelle Bildfolge bei Grafikbetrieb (Spiele,Konstruktion usw.). Auch eigene mathematische Routinen, speziell für extrem genaue Berechnungen, sind nur in Maschinensprache zu realisieren.
Das waren nur einige Vorteile der Maschinensprache.

Diese Beispiele sollten genügen, um die Notwendigkeit der Maschinenspracheprogrammierung, auch bei Rechnern mit sehr gutem BASIC, wie bei den MSX Computern, darzustellen. Es muß aber gesagt werden, daß diese Sprache auch einen großen Nachteil hat.

Maschinensprache ist die Sprache des Mikroprozessors (CPU) des Computers und damit die am weitesten maschinenorientierte Sprache. Eine starke Maschinenorientierung hat aber für den Programmierer zur Folge, daß er, um diese Sprache zu verstehen, sehr abstrakt denken muß. Das liegt darin begründet, daß die CPU grundsätzlich nur Zahlen versteht, d.h. ein Maschinenprogramm ist einfach eine Reihe von Zahlen und nicht eine Folge von Begriffen. In dieser Form wäre die Programmierung in Maschinensprache bei umfangreicheren Programmen beinahe ein Ding der Unmöglichkeit. Deshalb wurde eine Art Zwischensprache entwickelt, die Maschinenprogramme anschaulicher und verständlicher macht. Diese Sprache nennt man Assembler. Die Assemblersprache ordnet jedem Maschinencode (also einer Zahl) eine Reihe von Symbolen zu. Diese Symbole bestehen aus:

1. Befehlswort, d.h. meist einer Abkürzung des englischen Wortes für den Befehl, auch Mnemonic genannt.

2. Operanden, der z.B. Adressen, Konstanten o.ä. (das Befehlswort betreffend) angibt.

Damit vereinfacht sich das Erstellen eines Maschinenprogramms auf das Schreiben in Assemblersprache. Diese Assemblersprache wird von einem sogenannten Assemblerprogramm automatisch in den Maschinencode übersetzt. Außerdem gibt es noch den Disassembler, ein Programm, das die Zahlenreihen der Maschinensprache in Assemblersprache übersetzt. Im vorhergehenden Beispiel haben Sie ein Assemblerlisting kennengelernt.

5.2 ZAHLENSYSTEME

Da ein Computer intern immer nur mit Zahlen arbeitet und zwar mit den Zahlen 0 und 1 (Digitalcomputer), ist es für den Maschinenspracheprogrammierer notwendig ein wenig über Zahlensysteme zu wissen. Folgende Zahlensysteme sind für uns von Bedeutung:

1. Das Dezimalsystem
2. Das Dualsystem
3. Das Hexadezimalsystem

Zahlensysteme sind nach einem bestimmten Prinzip aufgebaute Ordnungsschemata der Ziffern. Jede Zahl kann in andere Zahlensysteme umgerechnet werden. In allen Zahlensystemen steigt der Stellenwert einer Ziffer von links nach rechts.
Um die anderen Zahlensysteme zu erklären, gehen wir von dem bekannten Dezimalsystem aus.

Das Dezimalsystem

```
Tausender  Hunderter  Zehner  Einer - Stellenwert
    7          3         5       6   - Ziffern
  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
     Der Stellenwert steigt von rechts nach links !
```

Potenz	Zahl	Bezeichnung
10^0	1	E-iner
10^1	10	Z-ehner
10^2	100	H-underter
10^3	1000	T-ausender
10^4	10000	Zehntausender
10^6	1000000	Million

Die Dezimalzahl 1335 kann man auch folgendermaßen schreiben:

1335 bedeutet: 1T + 3H + 3Z + 5E - Der niedrigste Stellenwert(Einer) steht am
435 bedeutet: 4H + 3Z + 5E - weitesten rechts.

1335 ist : $1*1000 + 3*100 + 3*10 + 5*1$
$\qquad\qquad\qquad\quad 3 \qquad\quad 2 \qquad\quad 1 \qquad\quad 0$
1335 ist auch: $1*10 + 3*10 + 3*10 + 5*10$

Man definiert eine Potenz mit dem Exponenten 0 als 1.
$\qquad 0 \qquad 0 \qquad 0$
Z.B.: $10 = 1$, $2 = 1$, $x = 1$

Das Dualsystem

Das Dualsystem ist nach dem gleichen Prinzip aufgebaut. Der Unterschied besteht nur darin, daß der Stellenwert der einzelnen Ziffern nicht durch Zehnerpotenzen, sondern durch Zweierpotenzen dargestellt wird.
Die Basis des Dualsystems ist 2.

Binär 10101101 = Dezimal 173

```
  7   6   5   4   3   2   1   0
  2   2   2   2   2   2   2   2   - Stellenwert

  1   0   1   0   1   1   0   1   - Ziffer
```

```
      7     6     5     4     3     2     1     0
173 = 1*2 + 0*2 + 1*2 + 0*2 + 1*2 + 1*2 + 0*2 + 1*2

173 = 1*128 + 0*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 1*1
```

Bis jetzt haben Sie die Umrechnung vom Dualsystem in das Dezimalsystem kennengelernt. Dieser Vorgang läßt sich natürlich auch umkehren. Zur Erläuterung der Umkehrung, betrachten wir die oben errechnete Dezimalzahl 173.
Wir überlegen, welche 2er Potenz gerade noch in dieser Zahl enthalten ist. Zur Hilfe: Im Prinzip kann man das Dualsystem auf n-stellige Zahlen anwenden. Im Computerbereich werden hauptsächlich 8-stellige Binärzahlen verwendet. Folgende Potenzen von 2 können vorkommen.

	7	6	5	4	3	2	1	0
Potenzen von 2	2	2	2	2	2	2	2	2
umgerechnete Werte	128	64	32	16	8	4	2	1

In diesem Fall ist also 2^7=128 die höchste vorkommende 2-er Potenz. Jetzt bilden wir die Differenz zwischen 173 und 128. Das Ergebnis lautet 45. Bei diesem Rest wird nun in gleicher Weise wie oben verfahren. Wir suchen also wieder die höchste Potenz von 2, die in diesem Wert steckt. Anhand der Tabelle läßt sie sich leicht ermitteln und beträgt 2^5=32. Anschließend bilden wir wieder die Differenz: (45-32=13).

Das beschriebene Verfahren wird solange angewandt, bis der Rest Null beträgt.

```
2^3=8    (13-8=5)
2^2=4    ( 5-4=1)
2^0=1    ( 1-1=0)
```

Wir haben folgende Potenzen von 2 ermittelt:
2^7 , 2^5 , 2^3 , 2^2 und 2^0
Unter jede vorkommende 2er Potenz schreiben wir eine Eins und unter die fehlenden eine Null:

```
  7  6  5  4  3  2  1  0
2 2  2  2  2  2  2  2  2

1  0  1  0  1  1  0  1  = 173
```

Die Dezimalzahl 173 wird also im Dualsystem durch 10101101 dargestellt. Im Folgenden wollen wir Binärzahlen durch das Voranstellen von &B kennzeichnen.

z.B. 173= &B 10101101

Bit und Byte

Ein BIT ist die kleinste Informationseinheit, aus der alle anderen Informationen zusammengesetzt sind. BIT ist die Abkürzung für "binary digit", was soviel heißt, wie Binärziffer. Es wird von einem gesetzten BIT gesprochen, wenn das BIT den Zustand 1, oder von einem rückgesetzten BIT, wenn es den Zustand 0 hat.
Die MSX Rechner haben einen 8-BIT Prozessor, d.h. er kann 8-BIT lange Dualzahlen verarbeiten, was den Dezimalwerten von 0 bis 255 entspricht.

Binärzahl:

```
1 0 1 1 0 1 1 1

g r g g r g g g      g=gesetztes BIT; r=rückgesetztes BIT

7 6 5 4 3 2 1 0      Nummer des BITs
```

Jedem Bit, jeder Ziffer, einer Binärzahl ist eine Bitnummer zugeordnet. Das Bit mit dem niedrigsten Stellenwert, d.h. daß am weitesten rechts stehende, hat die Nummer 0. Von rechts nach links wird dann fortlaufend numeriert. Die Bitnummer entspricht dem Exponenten der Zweierpotenz, die den jeweiligen Stellenwert darstellt.
Beim Computer ist es oft sinnvoll, sich die BIT-Zustände als einen Schalter vorzustellen.

```
SCHALTER EIN = 1
SCHALTER AUS = 0
```

Bei einer Zahl von 8 Schaltern lassen sich Werte von 0-255, also 256 Schaltzustände darstellen.
Acht Schalter (BITs) zusammengefaßt nennt man ein BYTE. Ein Byte kann vom Computer in einer Speicherstelle abgelegt werden. Wie werden aber Zahlen gespeichert, die größer als 255 sind? Zu diesem Zweck teilt man die Zahl in zwei Hälften, nämlich dem LOW Byte (engl. low: niedrig; niederwertiges Byte) und dem HIGH Byte (engl. high: hoch; höherwertiges

Byte). Diese Bytes werden nun in zwei aufeinanderfolgenden Speicherzellen abgelegt.

Das HIGH und LOW Byte läßt sich folgendermaßen berechnen:

Zahl dividiert durch 256=(HIGH Byte)+Rest
Der Rest der Division entspricht dem LOW Byte.

Zur Erinnerung: Die Zahl 255 ist der maximal darstellbare Wert in einem Byte, da es sich aus 8 BITs zusammensetzt.

Beispiel: Die Zahl 34065 soll in ein LOW und ein HIGH Byte zerlegt werden.

```
34065 / 256=133 Rest 17
34065      =133*256+17
```

133=High Byte
17=Low Byte

Die allgemeinen Formeln, in BASIC geschrieben, lauten:

```
1. HB=INT(Zahl/256)        HB=High Byte
   LB=Zahl-HB*256          LB=Low Byte
```

Diese Formel ist bei Zahlen beliebiger Größe anzuwenden.

```
2. HB=Zahl\256              HB=High Byte
   LB=Zahl MOD 256         LB=Low Byte
```

Die zweite Formel ist bei Zahlen, die in dem Bereich von -32768 bis 32767 liegen, anzuwenden.

Damit benötigt eine Zahl, die im Bereich von 256 bis 65535 liegt und im Speicher abgelegt wird, 2 Bytes.
Zur vereinfachten Darstellung von Zahlen, die in dieser Form im Speicher abgelegt sind, ist die Einführung eines weiteren Zahlensystems sinnvoll.

Das Hexadezimalsystem

Die Basis des Hexadezimalsystems ist 16.
Zur Erinnerung:

Die Basis des Dezimalsystems ist 10.
Die Basis des Dualsystems ist 2.

Zur Darstellung von Ziffern, deren Wert größer als 10 ist, werden im Hexadezimalsystem die Buchstaben A bis F verwendet.

Dezimalsystem:
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,...

Hexadezimalsystem:
0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F,10,11,12,...

Zuerst wandeln wir Hexadezimalzahlen in Dezimalzahlen um:

Potenz	Wert
0	
16	1
1	
16	16
2	
16	256
3	
16	4096

```
&H3ABF=3*16^3 + 10*16^2 + 11*16^1 + 15*16^0
&H3ABF=3*4096 + 10*256  + 11*16   + 15*1
&H3ABF=12288  + 176     + 15
&H3ABF=15039
```

Noch ein Beispiel:

```
&H1A3E=1*16^3 + 10*16^2 + 3*16^1 + 14*16^0
&H1A3E=1*4096 + 10*256  + 3*16   + 14*1
```

```
&H1A3E=4096    + 2560     + 48      + 14
&H1A3E=6718
```

Jetzt folgt die Umrechnung von Dezimalzahlen in Hexadezimalzahlen:

Der Vorgang der Umrechnung gleicht dem auf den vorigen Seiten beschriebenen. Nehmen wir an, die Dezimalzahl 45380 soll im Hexadezimalsystem dargestellt werden:

1. Schritt: Wir überlegen, welche größtmögliche Potenz von 16 in dieser Zahl vorkommen kann (zur Verfügung stehen die Zahlen in obiger Umrechnungstabelle: Hexadezimal in Dezimalzahlen).

2. Schritt: Wir teilen unsere oben genannte Zahl (45380) durch diesen Wert (4096) und wandeln die so erhaltene Dezimalzahl in eine Hexadezimalzahl um.

```
45380/4096=11 Rest 324
Dezimal=11 => Hexadezimal B
```

3. Schritt: Nun folgt der gleiche Vorgang mit dem Rest (324). Diesen dividieren wir nun durch die nächst kleinere passende Zahl aus der Tabelle, nämlich 256.

```
324/256=1 Rest 68
Dezimal=1  => Hexadezimal 1
```

Die oben beschriebenen Berechnungen werden so lang weitergeführt, bis der Rest der Division 0 ergibt.

```
68/16=4 Rest 4
Dezimal=4  => Hexadezimal 4
4/1=4 Rest 0
Dezimal=4  => Hexadezimal 4
```

Unsere umgewandelte Zahl heißt &HB144.

Der Vorteil des Hexadezimalsystems liegt darin, daß man das

Low und das High Byte direkt ablesen kann.
Für &H3ABF gilt:

- das High Byte setzt sich aus den ersten beiden Hexadezimalziffern (3 und A) zusammen. Es hat den Dezimalwert (3*16^1+10*16^0)=58.
- das Low Byte setzt sich aus den letzten beiden Hexadezimalziffern (B und F) zusammen. Es hat den Dezimalwert (11*16^1 + 15*16^0)=191.

Geben Sie einmal folgendes ein:

```
PRINT PEEK(&H1D),PEEK(&H1E)
```

An den beiden Adressen &H1D und &H1E steht die Sprungadresse, an die das Betriebssystem verzweigt, wenn eine Routine, z.B. in einem Steckmodul, aufgerufen werden soll. Für eine Sprungadresse ist ein Wert von 0 bis 65535 (also bis &HFFFF) möglich. Diese Zahl ist mit Hilfe von High Byte und Low Byte abgespeichert. Wir wollen die Sprungadresse nun berechnen. Mit dem obigen BASIC Befehl erhalten wir an Adresse &H1D den Wert 23 und an Adresse &H1E den Wert 2. Dezimal ergibt sich die Sprungadresse also aus 2*256+23=535.
Nun wollen wir im Hexadezimalsystem die gleiche Rechnung durchführen:
23=&H17 und 2=&H2, wie Sie leicht nachprüfen können. Den Wert der Sprungadresse erhalten wir einfach durch das Hintereinanderschreiben von High Byte und Low Byte: 535=&H217
Es ist also genauso einfach eine Hexadezimalzahl in High Byte und Low Byte zu zerlegen, wie sie aus High Byte und Low Byte zusammenzusetzen. Im Allgemeinen steht das Low Byte einer Zahl an der niedrigeren Speicheradresse, darauf folgt dann das High Byte.
Hiermit haben Sie den ersten Vorteil des Hexadezimalsystems kennengelernt. Außerdem läßt sich die Umwandlung vom Dualsystem in das Hexadezimalsystem sehr leicht durchführen. Dazu unterteilt man eine Dualzahl in zwei Blöcke zu je 4 Bit. Den Block vom 0ten bis 3ten Bit nennt man Low Nibble und den anderen Block vom 4ten bis 7ten Bit High Nibble. Jedes Nibble entspricht genau einer Hexadezimalziffer. Das ist leicht ein-

sichtig, da eine 4 Bit Dualzahl maximal den Wert 15 annehemen kann (15=8+4+2+1). Alle Werte von 0 bis 15 können aber auch durch eine Hexadezimalziffer (0, 1,...9, A, B, C, D, E, F) dargestellt werden. Betrachten wir ein Beispiel:

```
1 1 0 1  1 0 0 1
High N.  Low Nibble
8+4+1    8+1
  13       9
  &HD      &H9
```

Also: &B11011001=&HD9

Mit einiger Übung können Sie direkt aus einer 4-Bit Zahl die dazugehörige Hexadezimalziffer und umgekehrt ablesen. Dabei soll Ihnen folgende Tabelle helfen:

Dualsystem	Hexadezimalsystem	Dezimalsystem
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Entsprechend läuft die Umwandlung von Hexadezimal nach Dual. Jede Hexadezimalziffer wird durch die entsprechende vier Bit Kombination ersetzt, z.B. &HC7=&B1100 0111.

Das Verstehen der Umwandlung zwischen den unterschiedlichen Zahlensystemen ist eine Grundlage für die Programmierung in Maschinensprache und auch bei der fortgeschrittenen BASIC Programmierung unerläßlich.

5.3 DER Z80A PROZESSOR

Aufbau der CPU

Die MSX Computer besitzen eine Z80A CPU (Zentraleinheit). Wir erinnern uns, daß die CPU als "das Gehirn" des Rechners bezeichnet werden kann. Damit ist die Bedeutung dieser MPU (MPU: engl. Micro Prozessing Unit- Mikroprozessor) keine Frage.
In diesem Kapitel wollen wir uns mit dem Aufbau und der Funktion der einzelnen, in der CPU enthaltenen, Bausteine befassen.

1. CU (CU:engl.Control Unit- Kontrolleinheit)
   Alle Abläufe in einem Computer werden durch die CU kontrolliert und gesteuert.

2. Kontrollbus
   Der Kontrollbus ist der "lange Arm" der CU. Durch ihn werden Bausteine außerhalb der CPU gelenkt und überwacht.

3. Stapelzeiger SP (SP:engl. Stack Pointer)
   Mit Hilfe des SPs werden Daten und Unterprogrammrücksprungadressen im RAM zwischengespeichert. Da im SP Adressen gespeichert werden, ist er ein 16-Bit Register.

4. Programmzeiger PC (PC:engl.Programm Counter- eigentlich Programmzähler)
   Der PC zeigt auf die Speicheradresse, an der der jeweils zu verarbeitende Befehl steht.

5. Register B bis L (Register registrieren)
   Die CPU besitzt mehrere Register, in denen Daten gespeichert werden.

6. Flags (Flag:engl.flag- Flagge, Fahne; hier besser Kennzeichen)
   Flags dienen als Anzeiger für bestimmte Ereignisse, die bei Rechenoperationen in der CPU enstehen. Flags können gesetzt (Flagge oben) oder nicht gesetzt bzw. rückgesetzt (Flagge unten) sein.

7. Akkumulator (lat.akkumulieren:ansammeln)
   Der Akkumulator (Akku) ist das wichtigste Register der CPU. Man kann ihn auch als das Rechenregister bezeichnen.

8. ALU (ALU:engl.Arithmetical Logical Unit- Arithmetische Logische Einheit, Recheneinheit, Rechenwerk)
   Die ALU führt sämtliche arithmetischen und logischen Operationen durch. Abhängig vom Ergebnis der Operationen werden die Flags beeinflußt.

11. Schieber
    Der Schieber führt die Rotier- und Schieberoutinen aus.

Wie in Punkt 5 schon erwähnt, enthält die CPU mehrere Register. Zum Verständnis ihrer Funktionen haben wir sie in fünf Gruppen eingeteilt.

1. Der Akkumulator
2. Die Flags
3. Die "verknüpfbaren sechs" 8 Bit Register
4. Die "unzertrennlichen vier" 16 Bit Register
5. Interrupt-/Refreshregister

DER AKKUMULATOR

Der Akku bzw. das A Register ist das wichtigste Register des Z80. Die meisten arithmetischen und logischen Befehle benutzen dieses Register. Bei der Ausführung eines Vergleichbefehls wird grundsätzlich mit dem Inhalt des Akkus verglichen. Wie alle Register, bis auf SP, PC, IX und IY ist das A Register ein 8-Bit Register.

DIE FLAGS

Das F- bzw. Flag Register ist 8 Bit breit (wie A,B,C,D,E,H und L). Es hat jedoch andere Funktionen als diese. Im Flag Register werden die einzelnen Bits als Anzeiger für bestimmte Ereignisse, die bei Operationen des ALUs (Rechenwerk) entstehen, benutzt. Die einzelnen Bits des F Registers haben folgende Bedeutung:

```
S   Z       H     P/V  N   C  -Flagbezeichnung.
7   6   5   4   3   2   1   0  -Bitnummer
```

- C - Carry-Übertrag
- N - Subtraktion
- P/V - Parität/Überlauf
- H - Halbübertrag
- Z - Zero-Null
- S - Sign bzw. Vorzeichen

C Flag (Bit 0)
Tritt bei einer Addition oder Subtraktion ein Übertrag auf, wird dieses Bit gesetzt, sonst rückgesetzt.

N und H Flag (Bit 1, Bit 4)
Diese Flags werden intern vom Z80 benutzt. Sie haben für unsere Zwecke keine Bedeutung.

P/V Flag (Bit 2)
Dieses Flag hat eine doppelte Funktion:
Es wird gesetzt, wenn ein Überlauf (V) (engl.:overflow) auf-

tritt, sonst rückgesetzt. Weiterhin zeigt es die Parität (P) eines Bytes an.

Z Flag (Bit 6)
Dieses Flag wird gesetzt, wenn das Ergebnis einer Subtraktion Null ist, sonst rückgesetzt. Bei einem Vergleich wird dieses Bit gesetzt, wenn eine Gleichheit vorliegt.

S Flag (Bit 7)
Ist das Ergebnis einer Addition bzw. Subtraktion größer als 127, wird dieses Bit gesetzt. Wie wir später sehen werden, bedeuten bei der Arithmetik der CPU Bytes, die größer als 127 sind, negative Zahlen.

Die Bits 3 und 5 des Flag Registers sind ungenutzt.

DIE "VERKNÜPFBAREN SECHS" 8-BIT REGISTER

Zu dieser Gruppe gehören sechs 8-Bit Register:
B, C, D, E, H, L
Diese Register sind in der Lage, Registerpaare zu bilden, um ein 16-Bit breites Register darzustellen. In C, E, L wird jeweils das Low und in B, D, H das High Byte gespeichert.

B/C (Byte Counter)
Das B Register bzw. BC-Registerpaar wird häufig als Zähler z.B. für Schleifen verwendet.

Das DE Registerpaar ist frei verfügbar.
Dieses Registerpaar wird oft zur Zwischenspeicherung von Adressen oder Daten verwendet.

H/L (High/Low)
Das Registerpaar HL wird oft zur Speicherung von Adressen verwendet.

Eine Gewöhnung an die Benennung der Register in dieser Weise ist sinnvoll, da einige Befehle die Register in der oben be-

schriebenen Weise benutzen. Prinzipiell kann man natürlich auch das L oder E Register als Zähler verwenden.
Eine Besonderheit des Z80 ist, daß alle oben genannten Register mit gleicher Funktion noch einmal vorhanden sind. Dieser Zweitregistersatz steht uns zur Verfügung. Allerdings kann immer nur ein Satz zur Zeit benutzt werden.

DIE "UNZERTRENNLICHEN VIER" 16-BIT REGISTER

Zu dieser Gruppe gehören vier 16-Bit Register:
SP, PC, IX, IY
Das SP Register ist ein festes 16-Bit Register, d.h. es kann nicht in zwei 8-Bit breite Register zerlegt werden. Der Stack Pointer (SP) zeigt auf die jeweilige Adresse im Speicher, an der Rücksprungadressen oder zwischengespeicherte Daten stehen. Die Adresse bezieht sich auf eine Speicherstelle, die in einem Bereich des RAMs liegt, den man Stack oder Stapel nennt. Die Benutzung des Stacks zur Datenspeicherung geht folgendermaßen vor sich:
Beim Einschalten des Rechners wird der SP auf die höchste Adresse im Stack gesetzt (&HF000). Soll nun ein Byte auf den Stack gelegt werden, so wird SP automatisch um eins erniedrigt und dieses Byte in der Adresse, die SP dann anzeigt, abgespeichert. Er zeigt also immer auf die letzte Eintragung im Stapel. Beim "Holen vom Stack" läuft der Vorgang umgekehrt ab. Erst wird das Byte an der Adresse, auf die SP zeigt, gelesen, dann wird SP um eins erhöht. Auf diese Weise ist es möglich, Unterprogrammaufrufe beliebig ineinander zu verschachteln.

Der PC ist ein besonderes Register. Er kann vom Programm aus weder beschrieben noch geändert werden. Der PC wird intern verwaltet und zeigt immer auf die Adresse des aktuellen Befehls.

IX/IY Register werden hauptsächlich zur Speicherung von Adressen bzw. relativen Adressen benutzt. Auch diese beiden Register gehören, wie alle unter 2.5 aufgeführten, zu den 16-Bit Registern. Bei diesen ist es nicht möglich, getrennt

auf High bzw. Low Byte (wie bei BC, DE, HL) zuzugreifen. Die Benutzung der Indexregister ist der des HL Registerpaares ähnlich. Den Unterschied werden wir bei der indizierten Adressierung kennenlernen.

INTERRUPT- UND REFRESH-REGISTER

Diese beiden Register sind der CU zugeordnet.

I- bzw.Interruptregister
(engl.interrupt: unterbrechen)
Tritt ein Interrupt auf, d.h. eine Programmunterbrechung, so enthält dieses 8-Bit Register den oberen Teil der Adresse, an die verzweigt werden soll. Der untere Teil wird von dem Baustein des Computers geliefert, der den Interrupt ausgelöst hat.

R bzw. Refreshregister (engl.:refresh:auffrischen)
Dieses Register wird von der Hardware als Zähler benutzt, um in regelmäßigen Abständen den Inhalt der dynamischen Speicher aufzufrischen. Damit soll verhindert werden, daß gespeicherte Informationen verlorengehen. Durch ständiges Neuladen des gleichen Speicherinhaltes innerhalb sehr kurzer Zeit, wird ein Verlust der Daten verhindert.

Eine Befehlsausführung durch die CPU sieht dann folgendermaßen aus:
Das Byte, an der Adresse, auf die der PC zeigt, wird gelesen, und der PC wird um eins erhöht, d.h. er zeigt auf das nächstfolgende Byte. Das gelesene Byte wird als Befehl interpretiert. Dann werden eventuell zu dem Befehl gehörende Daten gelesen (PC wird dann wieder erhöht). Danach erfolgt die Ausführung des Befehls, und der Vorgang beginnt von Neuem.

Nachdem wir nun die Z80A CPU kennengelernt haben, werden wir uns jetzt mit der Eingabe von Maschinenprogrammen beschäftigen.

5.4 EINGABE VON MASCHINENPROGRAMMEN

Damit wir die Befehle des Z80 gleich ausprobieren können, müssen wir uns zuerst darüber Gedanken machen, auf welche Weise ein Maschinenprogramm vom BASIC aus eingegeben und abgespeichert wird. Ähnlich wie beim BASIC, wo eine Zeilennummer einem Befehl zugeordnet ist, wird jedem Maschinenbefehl eine Adresse zugeordnet.

BASIC		Maschinensprache		
Zeilennr.	Befehl	Adresse	Befehl	Code
9	HL=HL+1	&HF009	INC HL	&H23
10	RETURN	&HF00A	RET	&HC9

- Beim BASIC wird eine Zeilennummer einem Befehl zugeordnet.
- Bei der Maschinensprache gehört zu jedem Befehl eine Adresse.

Ein Maschinenprogramm ist damit eine Folge von Befehlscodes, die in aufeinanderfolgenden Adressen im Speicher stehen.
Vom BASIC aus haben wir die Möglichkeit, mit Hilfe des >POKE< Befehls die Codes an die entsprechenden Adressen zu schreiben. Ein Aufruf der Maschinenprogramme geschieht dann mit dem >USR< Befehl. Vorher muß die Startadresse des Maschinenprogramms mit >DEFUSR< festgelegt werden. Die Startadresse ist meistens die Adresse des Speicherplatzes, der den ersten Maschinencode enthält. Damit unser Maschinenprogramm nicht versehentlich überschrieben wird, müssen wir seinen Speicherbereich mit dem >CLEAR< Befehl reservieren. Wir werden durch >CLEAR 200,&HEFFF< oft den Bereich von &HF000 bis &HF37F reservieren, damit stehen also &H380 Bytes (entspricht ca.1K) für Maschinenprogramme zur Verfügung. Ein typisches BASIC Programm, zum Laden von Maschinenprogrammen hat folgenden Aufbau:

```
10 CLEAR 200,&HEFFF
20 FOR I=Startadresse TO Endadresse
30 READ A
40 POKE I,A
50 NEXT I
60 DEFUSR1=Startadresse
70 END
80 DATA .....
90 DATA .....
    .
    .
    .
```

In den DATA Zeilen stehen die Codes, die das eigentliche Maschinenprogramm bilden werden. Die Endadresse (V=Variable; diese Abkürzung werden wir in Zukunft immer hinter Wörter schreiben, die Variablen sind), muß natürlich größer als &HEFFF und Startadresse (V) kleiner als &HF380 sein. Der Aufruf des geladenen Programms erfolgt mit >X=USR1 (Parameter)<. Normalerweise werden wir &HF000 als Startadresse benutzen. Endadresse (V) ergibt sich aus Startadresse (V) plus Länge des Programms in Bytes minus 1. Die Länge eines Programms entspricht der Anzahl der Eintragungen in den DATA Zeilen.

Mit Hilfe der Befehle >DEFUSR Nr.< und >USR Nr.(Parameter)<, ist es beim MSX BASIC möglich, bis zu 10 verschiedene Maschinenprogrammaufrufe zu definieren. Wir werden zunächst immer >USR1< benutzen. Der hinter dem >USR< Befehl in Klammern stehenden Parameter, hat für uns keine Bedeutung. Hier muß formell eine beliebige Zahl oder Variable dort eingesetzt werden.

Für die Eingabe von kleinen Programmen ist folgendes BASIC Programm sinnvoll:

```
10 CLEAR 200,&HEFFF:CLS
20 INPUT "Startadresse &H";A$
30 ST=VAL("&H"+A$):IF ST<&HEFFF THEN BEEP:GOTO 20
40 AD=ST
50 IF AD>=&HF380 THEN BEEP: PRINT
"Bereichsüberschreitung":END
60 PRINT"Adresse &H";HEX$(AD);" : ";
70 INPUT"Wert &H";A$
80 IF A$="" THEN DEFUSR1=ST:END
90 W=VAL("&H"+A$):A$="":IF W<0 OR W>255 THEN BEEP:GOTO 50
100 POKE AD,W:AD=AD+1:GOTO 50
```

Sie geben die Hexadezimalcodes direkt ein, und das Programm wird das "Poken" für Sie erledigen. Das Hexzeichen (&H) brauchen Sie nicht mit einzugeben.

Nachdem wir nun die Eingabe von Maschinenprogrammen kennengelernt haben, wollen wir uns die Befehle des Z80 ansehen.

Anmerkung: Bei der Befehlserklärung werden wir oft mit Analogien zu den BASIC Befehlen arbeiten. Dazu stellen wir uns ein Register im BASIC als eine Variable mit demselben Namen vor (Register HL in Maschinensprache entspricht Variable HL in BASIC).

5.5 DIE BEFEHLE

Die Befehle des Z80 lassen sich in 5 Gruppen unterteilen:

1. Transfer von Daten
2. Bearbeitung von Daten und Tests
3. Sprünge
4. Steuerbefehle
5. Ein- und Ausgabe

TRANSFER VON DATEN

Diese Befehle dienen der Übertragung von Daten.
Daten können übertragen werden von:

a) Register zu Register
Das entspricht einer Zuweisung im BASIC, wie z.B. A=B oder SP=HL. Der Maschinenbefehl hat folgendes Format:
LD A,B (LD- lade)

b) Register zur Speicherstelle
Bei der Übertragung vom Register zur Speicherstelle ist der BASIC Befehl >POKE Speicheradresse, Variable<, z.B. >POKE &HF000,HL< entsprechend dem Maschinensprachebefehl LD (&HF000),HL.

c) Speicherplatz zu Register
Die Datenübertragung vom Speicher in ein Register, z.B. LD H,(&HF005), entspricht dem BASIC Befehl: >H=PEEK (&HF005)<.

BEARBEITUNG VON DATEN UND TESTS

Die Befehle zur Bearbeitung von Daten kann man wiederum in 5 Gruppen einteilen:

- arithmetische Operationen (z.B. ADDition, SUBtraktion)
- logische Operationen (z.B. AND, OR)
- Zählbefehle (INCrease = erhöhen, DECrease = erniedrigen)
- Bitmanipulation (SET, RESet)
- Vertauschen und Schieben von Bits (Rotate = rotieren, Shift = schieben)

Bei der Ausführung dieser Befehle werden Register- oder Speicherinhalte (im RAM) verändert. Viele Befehle sind denen des BASIC ähnlich:

Assembler	BASIC
SUB A,B (SUBtraktion)	A=A-B
ADD HL,BC (ADDition)	HL=HL+BC
AND C	A=A AND C
OR HL	A=A OR PEEK(HL)

Getestet werden entweder einzelne Bits in Registern bzw. Speicherstellen (BIT Befehl), oder es werden Register- oder Speicherinhalte mit dem Akku verglichen (CP Befehl=compare). Je nach dem Ausgang dieser Tests werden von der ALU die jeweiligen Flags im F Register gesetzt oder gelöscht.

SPRÜNGE

Mit Hilfe dieser Befehle ist es möglich, Verzweigungen in Maschinenprogramme einzubauen.
Man unterscheidet drei Sprungarten:

- direkter Sprung an eine 16-Bit Adresse (JP=Jump)
- relativer Sprung zur aktuellen Adresse (JR=Jump relativ)
- Unterprogrammsprünge (CALL und RET-Rücksprünge)

Man bezeichnet einen Sprung als bedingt, wenn die Entschei-

dung darüber, ob gesprungen wird, vom Status eines Flags abhängt. Ein bedingter Sprung, d.h. einer, bei dem der Sprung vom Status eines Flags abhängt, ist z.B. JR NZ,&HF003.

Analogien:

Assembler	BASIC
JP	GOTO
CALL	GOSUB
RET	RETURN
JR	—— (ähnlich der FOR-NEXT Schleife)

STEUERBEFEHLE

Mit diesen Befehlen kann beispielsweise ein Programm unterbrochen werden. Auch Interruptsteuerung ist mit diesen Befehlen möglich.

EIN/AUSGABEBEFEHLE (Input/Output)

Die I/O-Befehle dienen der Kommunikation mit Ein/Ausgabegeräten. Abhängig von der jeweilig angesprochenen I/O-Port Adresse werden die verschiedensten Aufgaben mit diesen Befehlen bewältigt. Häufig über I/O Befehle ansprechbare ICs sind:

PPI (Programmable Peripheral Interface),
PSG (Programmable Sound Generator),
VDP (Video Display Controller) und damit die Peripheriegeräte: Tastatur, Lautsprecher, Monitor, Drucker und Cassettenrecorder.

5.6 EIN BEISPIEL FÜR MASCHINENPROGRAMMIERUNG

Wir wollen nun Schritt für Schritt mit Ihnen ein Maschinenprogramm entwickeln. Das Programm soll die gleiche Aufgabe haben, wie das im Kapitel über Grafik beschriebene BASIC Programm zur inversen Darstellung eines Zeichens.
Ziel dieses Maschinenprogramms ist das Invertieren der ersten 128 Zeichen im Zeichengenerator und zwar so, daß diese normal und invers dargestellt vorhanden sind. Dieses Problem wollen wir zuerst wieder in BASIC lösen aber in einer Form, die der Maschinenspracheprogrammierung nahe kommt.
Zur Lösung des Problems benötigen wir folgende Informationen:

Basisadresse des Zeichengenerators:

Die Basisadresse erhalten wir mit >PRINT BASE(2)<. Sie lautet 2048 oder hexadezimal &H0800.

Anfangsadresse des ersten Zeichens, das invers darzustellenden ist (Leerzeichen):

Da wir den ab dem Leerzeichen alle Zeichen bis Code 128 behandeln wollen, benötigen wir seine Adresse. Wie Sie aus dem Grafikkapitel her wissen, berechnet man die Position innerhalb des Zeichensatzes mit:

```
Position=ASCII*8
```

Der ASCII Code des Leerzeichen ist 32. Die Berechnung lautet:

```
32*8=256
```

Dieser Wert wird nun zur Basisadresse addiert um die Adresse an der das Leerzeichen steht zu erhalten.

```
2048+256=2304=&H900
```

Somit haben wir die aktuell Speicheradresse, ab der wir

auslesen müssen. Diese Adresse speichern wir in einer Variablen (HL).

```
HL=2304 oder HL=&H900
```

Adresse der Speicherstelle an der das erste inverse Zeichen gespeichert werden soll (inverses Leerzeichen).

Die inversen Matrixen sollen die Codes des normalen Zeichens+128 haben. Das inverse Leerzeichen hat also den Code 128+32=160. Damit ist die Startadresse des ersten Leerzeichens:

```
2048+160*8=3328 oder &HD00
```

Der Wert &HC00 wird auch in einer Variablen (DE) gespeichert:

```
DE=3328 oder DE=&HD00
```

Zähler für die Anzahl der umzuwandelnden 96 Zeichen:

Die Zeichen mit den Codes 32-127 sollen invertiert werden. Das sind 96 Zeichen.
Da jedes Zeichen durch 8 Bytes dargestellt wird, müssen wir also 96*8=768 mal die Schleife zum Lesen und Schreiben durchlaufen. Diesen Wert speichern wir in einer weiteren Variablen (BC) ab.

```
BC=768 oder BC=&H300
```

Die ersten Zeilen unseres BASIC Programm lauten:

```
10 HL=&H900
20 DE=&HD00
30 BC=&H300
```

Zur Maschinenprogrammierung:

Innerhalb der Maschinenprogrammierung werden die oben errechneten Parameter nicht in Variablen sondern Registern gespeichert. Leider sind in dieser Programmiersprache nur einige wenige Register für unseren Zweck geeignet, nämlich die Register HL, DE und BC. In die Register können wir nun die oben errechneten Werte speichern.
Der Befehl dafür heißt:

```
LD  Register,Wert
```

(LD=engl. load: lade)

Lade das Register x mit dem Wert y.
Unsere Zeilen haben in Assemblersprache folgendes Format:

```
10 LD   HL,&H900
20 LD   DE,&HD00
30 LD   BC,&H300
```

Jetzt werden Sie sich sicher fragen, was das mit den Werten, die in den DATA Zeilen stehen, gemein hat. Ein Maschinenprogramm ist ja, wie gesagt, eine Reihe von Zahlen. Für die Übersetzung der Assemblerbefehle in die Maschinencodes gibt es zwei Möglichkeiten. Die komfortable Methode ist die automatische Übersetzung:
Der Assembler übersetzt die für den Programmierer leichter verständliche Assemblersprache in die Codes. Ist jedoch kein Assembler vorhanden, kann man verhältnismäßig einfach mit Hilfe von Tabellen die Assemblersprache in die entsprechenden Maschinencodes umwandeln. In Form der Codes wird ein Maschinenprogramm in DATA Zeilen abgespeichert.

z.B.:Zeile 10 des Maschinenprogramms:

```
10 LD   HL,&H900      ergibt  die Codes 21 00 09
```

Die entsprechende DATA Zeile dazu lautet:

```
100 DATA &H21,&H00,&H09
```

Dabei steht der Code 21 für den Assemblerbefehl
LD HL,Wert
Wenn wir die letzten beiden Werte in der DATA Zeile in der üblichen Weise, erst Low Byte und dann das High Byte lesen, erhalten wir den Startwert des Standardzeichensatzes.

```
LB HB HB LB                         LB=Low Byte
00 09=09 00=&H0900=2304             HB=High Byte
```

Das gleiche Verfahren wird auf die anderen beiden Zeilen angewandt:

```
20 LD  DE,&HD00  ergibt 11 00 0D
```

Dabei steht der Code 11 für: LD DE,Wert
Der Code 000D für &HD00

```
30 LD  BC,&H300  ergibt 01 00 03
```

Dabei steht der Code 01 für: LD BC,Wert
Der Code 0003 für &H300

Um den nächsten Programmierschritt zu vollziehen, gehen wir zurück zum BASIC:

Wir lesen mit >A=VPEEK(HL)< eine Punktreihe in Form eines Bytes der Standardzeichen aus, und schreiben sie, nachdem wir sie mit >A=A OR 255< invertiert haben, mit >VPOKE DE,A< an die für die inversen Zeichen vorgesehene Adresse.

Die nächsten BASIC Zeilen:

```
40 A=VPEEK(HL): REM lesen
50 A=A XOR 255: REM invers
60 VPOKE DE,A: REM schreiben
```

In Maschinensprache benötigen wir zum Schreiben und Lesen einer Speicherstelle zwei Maschinenroutinen. Routinen ruft man mit

```
CALL Adresse
```

auf. Die Adressen und die Funktion dieser Routinen muß man natürlich kennen. Im Kapitel Systemroutinen können Sie einige solcher Adressen und Funktionen finden. Die nächsten Programmzeilen ergeben sich zu:

```
40 CALL  &H004A  entspricht den Codes CD004A
50 XOR   255     entspricht den Codes EEFF
60 EX    DE,HL   entspricht den Codes EB
70 CALL  &H004d  entspricht den Codes CD004D
80 EX    DE,HL   entspricht den Codes EB
```

Die Zeile 40 ruft die Routine auf, die ein Byte aus dem VRAm liest. Die Adresse des zu lesenden Bytes wird durch den Inhalt des HL Registers bestimmt. Die Routine speichert den gelesenen Wert im Akkumulator (A). D.h. in der nächsten Zeile (Zeile 50) steht der Wert, der im BASIC durch >VPEEK(HL)< gelesen wurde im Akku zur Verfügung.

Die Zeile 50 in Maschinensprache ist mit der BASIC Zeile fast identisch. Sie stellt invertiert ein Byte, was für eine Reihe von acht Punkten der Zeichendefinition steht.

Zeile 60: Da für die in der nächsten Zeile aufgerufenen Routine auch die Adresse des zu beschreibenden Bytes im HL Register übergeben werden muß, wir aber an die Zieladresse schreiben wollen, die in DE gespeichert ist, müssen DE und HL vorher vertauscht werden. Das bewirkt der EX Befehl (wie >SWAP<).

In Zeile 70 wird die Routine zum Schreiben eines Bytes in den VRAM aufgerufen. Wiederum enthält HL die Adresse der zu beschreibenden Speicherstelle. Der Wert, der an dieser Speicherstelle gespeichert werden soll, muß beim Aufruf der Routine im Akku (A) enthalten sein. Das ist in unserem Programm der Fall.
Alle logischen Befehle betreffen grundsätzlich den Akku, d.h. XOR 255 invertiert den Akkuinhalt, auch wenn A nicht explizit angegeben ist.

Zeile 80: Um wieder den alten Zustand für den nächsten Aufruf der VPEEK Routine (Zeile 40) herzustellen, wird nochmals mit EX vertauscht.

Die nächsten Programmschritte erhöhen die gespeicherten Werte in HL und DE um jeweils 1. Dadurch wird erreicht, daß nach und nach alle für unser Programm wichtigen Speicherstellen gelesen und beschrieben werden.
Da wir aber nur 96 Zeichen verändern wollen, muß noch ein Zähler vorhanden sein, der gegen Null zählt und über eine Abfrage das Programm beendet, wenn alle Zeichen verändert wurden. Dieser Zähler ist BC. BC wird jeweils um den Wert 1 erniedrigt.
Die letzte Zeile stellt die Abfrage dar, ob BC=0 ist.

```
70 HL=HL+1: REM nächste Speicherstelle
80 DE=DE+1: REM nächste Speicherstelle
90 BC=BC-1: REM nächstes Zeichen
100 IF BC<>0 THEN 40: REM Wenn alle Zeichen gelesen Ende
```

Zur Maschinensprache:

In Maschinensprache erhöht der Befehl

```
INC  Register
```

einen Registerinhalt um den Wert 1. INC steht dabei für engl. increase= erhöhen.
Der Befehl

```
DEC  Register
```

macht das Gegenteil. Er erniedrigt einen Registerinhalt um 1. DEC steht dabei für engl. decrease=erniedrige. Die nächsten Zeilen des Maschinenprogramms haben also folgendes Format:

```
90 INC  HL    entspricht dem Code 23
100 INC  DE   entspricht dem Code 13
110 DEC  BC   entspricht dem Code 0B
```

Die nächsten Befehle realisieren die Abfrage und damit die Schleife in Zeile 100 unseres BASIC Programms.

```
LD  A,B
```

Lade den Akku (A) mit dem Wert von B, also mit dem High Byte von Register BC.

```
OR  C
```

OR C bezieht, sich wie alle logischen Befehle (siehe oben), auf den Akku. OR C bedeutet eigentlich:
Verknüpfe den Akkuinhalt mit dem Inhalt vom C Register durch "oder" und speichere das Ergebnis wieder in A:

```
A=A OR C
```

Der ursprüngliche Inhalt von A ist wegen des vorhergehenden Befehls der Inhalt von B. Also wird eigentlich B mit C über "oder" verknüpft. Der Umweg über den Akku ist notwendig, da ja die logischen Befehle immer den Akku benutzen.
Die "oder" Verknüpfung hat die Eigenschaft, daß alle Bits im Ergebnis gesetzt sind, die in einem der beiden verknüpften Werte gesetzt waren. Das bedeutet, daß das Ergebnis der Verknüpfung nur dann Null ist, wenn beide verknüpften Werte (hier B und C) Null sind. Wenn also der Akku nach OR C gleich 0 ist, ist auch BC=0, also ist die Schleife zu beenden.
Die arithmetisch/logischen Befehle beeinflussen alle das Z-Flag. Dieses Flag zeigt an, ob das Ergebnis eines Befehls Null war (Z=zero) oder nicht (NZ=non zero). Damit kann nun ein bedingter Sprung erfolgen:

```
JR  NZ,Sprungziel oder JR NZ,Distanz
```

(JR= Jump Relativ= relativer Sprung)
(NZ= Non Zero= nicht Null)
Dieser Sprungbefehl bedeutet soviel wie "Springe, wenn nicht Null."
Das Sprungziel wird durch die Differenz der Adresse des auf den Sprungbefehl folgenden Befehls zur Zieladresse angegeben. Negative Zahlen werden im 2er Komplement angegeben (d.h. 256-Betrag des Wertes). Auf die Berechnung dieses Wertes gehen wir nicht ein. Für unseren Fall beträgt das Sprungdistanzbyte &HEF. Ist die Bedingung nicht erfüllt, hier also Z-Flag gesetzt, dann wird der nächste Befehl verarbeitet.
Das ist der das Programm beendende RET (wie RETURN) Befehl, der eine Rückkehr ins BASIC bewirkt.

```
120 LD  A,B       entspricht dem Code 78
130 OR  C         entspricht dem Code B1
140 LR  NZ,$-17   entspricht den Codes 20EF
```

Das komplette Listing des BASIC Programms:

```
10 HL=2304              oder 10 HL=&H900
20 DE=3328              oder 20 DE=&HD00
30 BC=778               oder 30 BC=&H300
40 A=VPEEK(HL)
50 A=A XOR 255
60 VPOKE DE,A
70 HL=HL+1
80 DE=DE+1
90 BC=BC-1
100 IF BC<>0 THEN 40
```

Und nun den BASIC Lader, der das Maschinenprogramm mit der gleichen Aufgabe lädt, wie das oben aufgeführte:

```
5 SCREEN 0
10 CLEAR 200,&HEFFF
20 FOR I=&HF000 TO &HF01A
30 READ A$
40 POKE I,VAL("&H"+A$)
60 NEXT
70 DEF USR1=&HF000
80 LOCATE 0,7:END
90 DATA 21,00,08,11,00,0C,01,00
100 DATA 04,CD,4A,00,EE,FF,EB,CD
110 DATA 4D,00,EB,,13,0B,78,B1
120 DATA 20,EF,C9
```

Starten Sie das Programm mit >RUN< und rufen es mit >X=USR1(1)< auf.

Das Assemblerlisting sieht folgendermaßen aus:

```
Adresse Code     Zeilen-Nr. Assemblerbefehl  Kommentar

F000 210009      10         LD   HL,&H900 ; Startadresse Ze
ichengenerator
F003 11000D      20         LD   DE,&HD00 ;
F006 010003      30         LD   BC,&H300 ;
F009 CD004A      40         CALL &H004A ;Byte von Adresse HL
aus VRAM lesen
F00C EEFF        50         XOR  255 ; Invers
F00E EB          60         EX   DE,HL ; Zieladresse nach HL
F00F CD004D      70         CALL &H004D ; schreiben eines B
ytes an Adresse HL ins VRAM
F013 EB          80         EX   DE,HL ; alten Zustand wied
er herstellen
F014 23          90         INC  HL ; HL um 1 weiterzählen
F015 13          100        INC  DE ; DE um 1 weiterzählen
F016 0B          110        DEC  BC ; BC um 1 erniedrigen
F017 78          120        LD   A,B ; High Byte von BC in
den Akku laden
F018 B1          130        OR   C ; Mit Low Byte BC oderie
ren
F019 20EF        140        JR   NZ,$-17  ; Wenn nicht Null
,dann weiter
F01A C9          150        RET ; sonst zurück zum BASIC
```

Erklärung zum Assemblerlisting

Adresse:

Unter dieser Spalte sind die Adressen der Speicherstellen geschrieben, an die die jeweiligen Codes des Maschinenprogramms gespeichert werden.
Die Adressen sind, wie auch die nebenstehenden Codes in hexadezimaler Form geschrieben.

Code:

Hier stehen die Codes der einzelnen Befehle ebenfalls in hexadezimaler Schreibweise. Immer 2 Ziffern zusammen ergeben ein Byte. Die Anzahl der Bytes eines Befehls läßt sich so leicht feststellen.

z.Beispiel:Zeile 10 und 20:

Adresse Code

F000 210009

F003 11000D

Die Anzahl der Codes in Zeile 10 beträgt 3 nämlich hexadezimal 21=Byte 1, 00=Byte 2 und 09=Byte drei. Demnach benötigt dieser Befehl drei Bytes. Die Adresse der nächsten Zeile zeigt uns das auch an (F003). Die Codes finden Sie auch in den DATA Zeilen des BASIC Programms wieder.

Zeilen Nr.

Hier steht die Zeilennummer nocheinmal in dezimaler Form.

Assemblerbefehl:

Unter diesem Begriff stehen nun die Mnemonics (Befehlsworte) mit den entsprechenden Operanden (z.B. Adressen, Konstanten o.ä).

z.B.: CALL &H0072

CALL = Mnemonic
&H0072 = Operand (in diesem Fall eine Adresse)

Kommentar:

Hier kann man zum besseren Verständnis der Maschinenprogramme Erklärungen aufschreiben.

5.7 DER MONITOR

Ein unentbehrliches Hilfsmittel zur Erstellung von Maschinenprogrammen ist ein sogenanntes Monitorprogramm. Ein Monitor dient zum Anschauen und Ändern von Speicherinhalten. Speicherbereiche können auf Kasette gespeichert bzw. von Kassette geladen werden.
Bei professionellen Monitoren sind noch weitere Funktionen enthalten:

- Disassembler, damit können Speicherinhalte, die Programme darstellen, in Assemblersprache übersetzt werden.
- Such Funktion, mit der nach bestimmten Bytefolgen im Speicher gesucht werden kann.
- Start Funktion, mit der Maschinenprogramme zu Testzwecken aufgerufen werden können.
- Verschiebungsfunktion, die Speicherbereiche verschieben und/oder kopieren können.
- Miniassembler, mit deren Hilfe einzelne Assemblerbefehle in ihre Codes übersetzt werden können.

Im folgenden Programm sind einige dieser Möglichkeiten enthalten.
Nach dem Start des Programms zeigt ein Stern an, daß eine Eingabe erwartet wird. Sämtliche Möglichkeiten des Monitors sind durch Eingabe eines Buchstaben abzurufen:

```
m - Monitor: Speicherbereich anzeigen
S - save
L - Load
a - Ändern von Speicherinhalten
g - Maschinenprogramm starten
r - Slot selektieren
```

Die Funktionen im Einzelnen:

Monitor (m)

m ruft die Grundfunktion des Monitorprogramms auf, nämlich das Anzeigen von Speicherinhalten. Direkt hinter dem m müssen die Start- und Endadresse des Bereiches der ausgegeben werden soll, eingegeben werden. Die Eingabe jeder Zahl wird durch RETURN abgeschlossen. Die Ausgabe der Speicherinhalte erfolgt in dem üblichen Format für einen HEX DUMP (Ausgabe der Speicherinhalte in hexadezimaler Form).

An erster Stelle steht die aktuelle Adresse, darauf folgen die an dieser und den 7 folgenden Adressen stehenden Werte in hexadezimaler Form.
Am Ende der Zeile steht die Zeichendarstellung der acht letzten Codes, wobei nach den ASCII Codes interpretiert wird.
Codes, die größer als 127 sind, werden um 128 erniedrigt.
Steuerzeichen werden als Punkte ausgegeben.

Save (S)

Hier muß hier die Start- und Endadresse des zu speichernden Bereichs eingegeben werden. Dann muß der Name eingegeben werden unter dem gespeichert werden soll.

Load (L)

Hinter dem L muß der Name der zu ladenden Daten angegeben werden.

Aufruf Maschinenprogramm (g)

Nach Eingeben des Buchstaben g wird die Startadresse des aufzurufenden Programms eingegeben.

Ändern von Speicherinhalten (a)

Die Adresse des ersten zu ändernden Bytes muß eingegeben werden. Dann wird jeweils die aktuelle Adresse ausgegeben und Sie müssen den gewünschten Wert eingeben.
e beendet die Eingabe.

Slotwahl (r)

Die Nummer des gewünschten Slots muß eingegeben werden (von 0 bis 3). Ab sofort werden durch ihn die Inhalte des eingegebenen Slots angezeigt. Der Bereich ab &HF000 darf nur mit Slot 2 ausgesesen werden.

```
10 CLEAR 200,&HEFFF:MAXFILES=2
20 OPEN "CRT:" FOR OUTPUT AS #1
30 OPEN "LPT:" FOR OUTPUT AS #2
40 DEFSNG A-Z
50 KN=1:REM Kanalnummer
60 AN=6
70 FOR I=1 TO AN:READ BF$(I):NEXT
80 DATA m,a,s,l,g,r
90 FOR I=&HF000 TO &HF022:REM Mapro Lesen fuer Slot Read
100 READ A$:POKE I,VAL("&H"+A$):NEXT
110 DATA CD,8A,2F,EB,C1,E1,E5,C5,D5,CF,2C,CD,1C,52,FE,04
120 DATA D2,5A,47,E3,CD,0C,00,CD,CF,4F,E1,C1,D1,E5,C5,21
130 DATA F6,F7,C9
140 DEFUSR1=&HF000
150 SCREEN 0:WIDTH 38
160 LOCATE 10,2:PRINT"M o n i t o r"
170 ON KEY GOSUB 880,920,960,1000
180 KEY 1,"D.an":KEY(1) ON
190 KEY 2,"Help":KEY(2) ON
200 KEY 3,"Ende":KEY(3) ON
210 KEY 4,"Stop":KEY(4) ON
220 LOCATE 0,5
230 Y=CSRLIN:FOR I=1 TO AN:PRINT BF$(I);" : ":NEXT
240 LOCATE ,Y
250 LOCATE 5:PRINT"Speicherbereich anzeigen"
260 LOCATE 5:PRINT"Speicherbereich aendern"
270 LOCATE 5:PRINT"Speicherbereich abspeichern"
280 LOCATE 5:PRINT"Speicherbereich laden"
290 LOCATE 5:PRINT"Maschinenprogramm starten"
300 LOCATE 5:PRINT"Slots selectieren"
310 PRINT
320 PRINT"Key 1 : Druckausgabe an/aus"
330 PRINT"Key 2 : Anzeige aller Befehle"
340 PRINT"Key 3 : Programm Ende"
350 PRINT"Key 4 : Aktuellen Vorgang Stoppen"
360 PRINT
370 PRINT:LOCATE 0,CSRLIN-1:PRINT"* ";
380 A$=INKEY$:IF A$="" THEN 380
```

```
390 PRINTA$;" ";
400 FOR I=1 TO AN:IF (ASC(A$)OR2^5)<>ASC(BF$(I)) THEN NE
XT
410 ON I GOSUB 440,610,690,740,780,830,870
420 GOTO 370
430 A$=""
440 REM Monitor
450 GOSUB 1040:REM Eingaben holen
460 IF (SN<>2) AND ((E-2^16*(E<0))>=49152!) THEN RETURN
470 FOR I=S TO E STEP 8
480 PRINT#KN,RIGHT$("000"+HEX$(I),4);" ";
490 A$=""
500 FOR J=0 TO 7
510 REM Byte Lesen
520 B=USR1(I+J),SN
530 PRINT#KN,RIGHT$("0"+HEX$(B),2);" ";
540 C=B AND 127
550 IF C<32 OR C=127 THEN C=46
560 A$=A$+CHR$(C)
570 NEXT J
580 PRINT#KN,A$
590 NEXT I
600 RETURN
610 REM Aendern
620 INPUT S
630 PRINT"&H";RIGHT$("000"+HEX$(S),4);" : ";
640 INPUT W$
650 IF (ASC(W$)OR2^5)=ASC("e") THEN RETURN
660 W=VAL(W$)
670 POKE S,W
680 S=S+1:GOTO 630
690 REM Speichern
700 GOSUB 1040
710 INPUT"Name : ";N$
720 BSAVE "CAS:"+N$,S,E
730 RETURN
740 REM Laden
750 INPUT"Name : ";N$
```

```
760 BLOAD "CAS:"+N$
770 RETURN
780 REM Start Programm
790 INPUT S
800 DEFUSR2=S
810 S=USR2(1)
820 RETURN
830 REM Slot wahl
840 INPUT "Slotnummer : ";SN
850 IF SN>3 THEN 840
860 RETURN
870 BEEP:LOCATE 0:PRINT"     ";:RETURN
880 REM Drucker an/aus
890 IF KN=1 THEN KN=2 ELSE KN=1
900 IF KN=1 THEN KEY 1,"D.an" ELSE KEY 1,"D.aus"
910 RETURN
920 REM Help
930 FOR I=1 TO AN+4:PRINTCHR$(13):NEXT
940 LOCATE ,CSRLIN-AN-3
950 RETURN 230
960 REM Ende
970 DEFUSR1=&H3E:X=USR1(1):REM Standarkeybelegung
980 CLS
990 END
1000 REM Stop
1010 PRINT
1020 BEEP
1030 RETURN 370
1040 REM Eingabe Start/Ende
1050 Y=CSRLIN
1060 INPUT S
1070 LOCATE 14,Y:INPUT E
1080 RETURN
```

Programmbeschreibung:

Der Großteil des Monitors ist durch REM Zeilen dokumentiert. Das in Zeile 90 bis 130 eingelesene Maschinenprogramm, wandelt den >USR< Befehl in einen >PEEK< Befehl um, wobei zusätzlich die Slotnummer angegeben werden kann. Ein Auslesen der Adressen ab &HC000 ist nur bei Slot 2 möglich. Das Format des Befehls ist:

USR1(Adresse),Slotnummer

Das Assemblerlisting für die Erweiterung finden Sie in der Kramecke.

Die Hauptschleife des Programms bilden die Zeilen von 370 bis 420. Weitere Eingaben sind per Interrupt über die Funktionstasten möglich.
Wichtig ist noch, daß sämtliche Zahlen sowohl in hexadezimaler als auch in dezimaler Form eingegeben werden können.

KAPITEL VI : SYSTEMROUTINEN

6.1 DIE BENUTZUNG VON SYSTEMROUTINEN

Das folgende Kapitel soll Ihnen die Möglichkeit geben Maschinenspracheroutinen zu verwenden. Anschließend an diesen Text finden Sie eine Liste von wichtigen Systemroutinen. Viele von diesen Routinen können auf einfache Weise benutzt werden, andere erfordern gute Kenntnisse in Maschinenspracheprogrammierung. In diesem Zusammenhang sei auf das "Maschinensprachebuch zu MSX" hingewiesen, das eine leicht verständliche Einführung in die Maschinensprache bietet und auch dem Fortgeschrittenen viele Hinweise und Tips zur Programmierung in dieser Sprache bietet.

Im einfachsten Fall kann eine Systemroutine direkt aufgerufen werden:

Beispiel: Warten auf Taste

Die Adresse der Routine ist &H009F. Da keine Parameter übergeben werden müssen, braucht nur ein >USR< Befehl mit dieser Startadresse belegt werden, also:

```
DEFUSR1=&H009F
```

Soll nun auf eine Taste gewartet werden, so muß die Routine nur noch aufgerufen werden:

```
X=USR1(1)
```

Wenn an eine Routine Parameter übergeben werden sollen, ist ein Minimaschinenprogramm notwendig, das den Wert einliest.

Beispiel: LD (HL),E mit Slotwahl

Diese Routine ist notwendig, um z.B. in verdeckte RAM Bereiche (&H0 bis &H7FFF) Werte zu schreiben, ohne diesen Bereich

durch >OUT< zu selektieren. Drei verschiedene Werte müssen übergeben werden:

HL - Adresse
E - Wert
A - Slotnummer

Das geschieht mit dem Miniassemblerprogramm:

```
LD HL,Adresse
LD E,Wert
LD A,Slot
CALL &H0014
RET
```

Anhand der Befehlslisten der Ladebefehle des Z80 können Sie leicht die Codes ermitteln.

```
LD HL,nn     Code: 21 n  n
LD E,n       Code: 1E n
LD A,Slot    Code: 3E n
CALL &H0014  Code: CD 14 00
RET          Code: C9
```

Der Code des CALL Befehls ist &HCD und der des RET Befehls &HC9.
Der BASIC Lader hat folgende Form:

```
10 CLEAR 200,&HEFFF
20 FOR I=&HF000 TO &HF00A:READ A:POKE I,A:NEXT
30 DEFUSR1=&HF000
40 DATA &H21,0,0&H1E,0,&H3E,0,&HCD,&H14,&H00,&HC9
```

Die zu übergebenden Werte (Adresse, Wert, Slot) sind zunächst durch Nullen gegeben.
Der Aufruf der Routinen sieht dann so aus:

AD enthält die Adresse
WE enthält den Wert
SL enthält die Slotnummer

```
100 POKE &HF001,AD-INT(AD/256)*256
110 POKE &HF002,INT(AD/256)
120 POKE &HF004,WE
130 POKE &HF006,SL
140 X=USR1(1)
```

Durch die >POKE< Befehle werden die jeweils aktuellen Werte genau an die richtigen Stellen im Maschinenprogramm geschrieben. Beim Aufruf des Maschinenprogramms werden sie dann entsprechend den Assemblerbefehlen geladen und übergeben.

Soll ein Wert, der durch eine Maschinenroutine ermittelt wurde, zurück ans BASIC gegeben werden, so wird er an einer vereinbarten Adresse gespeichert.

Die STRIG Routine gibt im Akku 0 oder 255 zurück. Nach dem Aufrufen der Routine wird der Akku z.B. an Speicherstelle &HF200 (in den mit >CLEAR< reservierten Bereich) geschrieben. Dazu dient der Befehl

```
LD (Adresse),A
```

der den Code 32 hat. Bei der vereinbarten Übergabeadresse &HF200 also

```
LD (&HF200),A      Code: 32 00 F2
```

Danach wird mit RET ins BASIC geschaltet, wo mit >PEEK(&HF200)< der ermittelte Wert geladen werden kann.

6.2 DIE ROUTINEN

Adresse &H0000: RESET

Der Aufruf dieser Routine bewirkt das gleiche, wie ein An/Ausschalten des Rechners bzw. wie ein Drücken der RESET Taste.

Adresse &H0008: RST &H08- Test auf nachfolgendes Byte

Es wird geprüft, ob das an Adresse HL stehende Byte, gleich dem, hinter dem RST &H08 Befehl stehende Byte ist. Bei Ungleichheit wird ein "Syntax Error" ausgegeben. Ansonsten wird danach zur RST &H10 Routine ab Adresse &H4666 verzweigt.

Adresse &H000C: LD A,(HL) mit Slotwahl

A enthält bei Übergabe die gewünschte Slotnummer (0-3). Als Ergebnis erhält man im Akku und im E Register den, an Adresse HL im gewünschten Slot stehenden, Wert.

Adresse &H0014: LD (HL),E mit Slotwahl

Bei Übergabe enthält A die Slotnummer, HL die Adresse und E den zu schreibenden Wert.

Adresse &H0018: RST &H18- Zeichenausgabe

Gibt das im Akku enthaltene Zeichen auf das aktuelle Gerät aus. Standardgemäß ist das der Bildschirm. Durch das Laden der Speicherstelle &HF416 mit einem Wert <> 0 wird der Drucker selektiert.

Adresse &H0020: RST &H20- Vergleich von HL mit DE

Der DE Registerinhalt wird vom HL Register abgezogen und entsprechend dem Ergebnis werden die Flags beeinflußt. HL und De werden nicht (!) verändert.

Adresse &H0028: RST &H28- Variablentyp Test

Ermittelt den Typ der aktuellen Variablen, wobei nach der Rückgabe gilt:

Carry=0	(NC)	Typ 8	DEL
Carry=1	(C)	Typ 2,3 oder 4,	und zwar
Sign Flag=1	(M)	Typ 2	INT
Zero Flag=1	(Z)	Typ 3	String
Sign Flag=0	(P)	Typ 4	SNG

Adresse &H0038: RST &H38- Interruptsprung bei INT MODE

Dies ist der Einsprungspunkt für die Routine, die durch den Standardinterrupt 50 mal pro Sekunde aufgerufen wird.

Adresse &H003E: Standardkeybelegung

Diese Routine belegt die Funktionstasten mit ihren ursprünglich, beim Einschalten vorhandenen, Worten.

Adresse &H0047: VDP Register Write

Das VDP Register mit der im C Register angegebenen Nummer wird mit dem im B Register enthaltenen Wert beschrieben, also VDP(C)=B.

Adresse &H004A: Video RAM- Read

Das an Adresse HL im Video RAM stehende Byte, wird durch diese Routine in den Akku geladen BASIC: A=VPEEK(HL).

Adresse &H004D: Video RAM- Write

Der Wert des Akku wird durch diese Routine an Adresse HL des Video-RAMs gespeichert.
BASIC: VPOKE HL,A

Adresse &H005F: Select SCREEN

Hier wird der SCREEN Modus auf den im Akku enthaltenen Wert (0, 1, 2 oder 3) umgeschaltet.
BASIC: SCREEN A

Adresse &H0093: PSG Register Write

Diese Routine schreibt den, im E Register enthaltenen Wert, in das PSG Register mit der Nummer A. Die Programmierung des PSG in Maschinensprache ist für die Erzeugung komplexer Klänge sehr wichtig. Im BASIC entspricht diese Routine dem >SOUND A,E< Befehl.

Adresse &H0096: PSG Register Read

Nach Aufruf dieser Routine enthält der Akku den Wert des PSG Registers mit der Nummer, die der Akku vor dem Aufruf dieser Routine enthält.

Adresse &H009F: Warten auf Tastendruck

Diese Routine wartet so lange, bis eine Taste gedrückt wird. Der der Taste entsprechende ASCII Code wird registriert und im Akku gespeichert. Dann erfolgt der Rücksprung.

Adresse &H00AE: Eingabe bis CR ab Zeilenanfang

Durch Aufruf dieser Routine erhalten Sie die Eingabe einer ganzen Zeile zurück. Da eine Eingabezeile bis zu 255 Zeichen lang sein kann, muß sie im RAM gespeichert werden. Das geschieht ab Adresse &HF55E. Im Bereich von &HF55E bis &HF65D ist also die letzte Eingabe abgespeichert. Bei der Rückkehr von dieser Routine, enthält HL die Startadresse dieses Eingabepuffers minus 1.

Adresse &H00C0: BEEP

Gibt eine BEEP aus.
BASIC: BEEP

Adresse &H00C3: Clear Screen

Löscht den gesamten Bildschirmanzeigebereich in allen Modi.
BASIC: CLS

Adresse &H00C6: Cursor setzen

Der Cursor wird durch diese Routine auf Position HL gesetzt. Dabei gilt:

H- Zeile
L- Spalte

BASIC: LOCATE L,H

Adresse &H00CC: KEY OFF

Schaltet die KEY Anzeige aus.

Adresse &H00CF: KEY ON

Schaltet die KEY Anzeige an.

Adresse &H00D5: STICK-Abfrage

Die Routine liefert nach Übergabe von A in der üblichen Weise im Akku den Wert der jeweiligen Richtung (siehe Handbuch).

0- Tastatur
1- Joystick 1
2- Joystick 2

BASIC: A=STICK(A)
Obwohl diese Routine genau dem BASIC Befehl entspricht, ist sie wichtig, da gerade für Spiele eine schnelle Abfrage des Joysticks sehr wichtig ist.

Adresse &H00D8: STRIG Abfrage

Diese Routine liefert nach Übergabe von A (siehe oben):
0 im Akku wenn "Feuerknopf" (Space) nicht gedrückt.
255 im Akku wenn "Feuerknopf" (Space) gedrückt.

Adresse &H0132: CAP An/Aus

Schaltet CAP an bzw. aus.

A=0 CAP an
A<>0 CAP aus

Adresse &H0135: Software Sound

Diese Routine setzt die EXT SOUND Leitung, die direkt zum Lautsprecher führt, auf low oder auf high:

A=0 Leitung auf low
A<>0 Leitung auf high

Ein schnelles Hintereinanderausführen von An- und Ausschalten, führt zur Ausgabe eines Tons. Ansonsten wird durch den Aufruf der Routine ein Knacken ausgegeben.

Adresse &H013E: VDP- Status Read

Diese Routine gibt im Akku den aktuellen VDP Statusregisterwert zurück.

Adresse &H2F8A: CINT

Konvertiert FAC ins "Integerformat" und prüft auf Größe. Diese Routine gibt in HL den Wert, und im Akku die Typenkennziffer aus (=2).

Adresse &H2FB2: CSNG

Konvertiert FAC ins Format "einfacher Genauigkeit".

Adresse &H2F99: HL in FAC kopieren

Lädt den Wert des HL Registers in den FAC und lädt die Typenkennziffer mit 2.

Adresse &H303A: CDBL

Konvertiert FAC ins Format "doppelter Genauigkeit".

Adresse &H4055: Syntax Error (ERR=2)

Adresse &H4067: Overflow (ERR=6)

Adresse &H406A: Missing Operand (ERR=24)

Adresse &H406D: Typ Mismatch (ERR=13)

Adresse &H406F: Fehlerausgabe

Gibt den Fehler, der der im E Register übergebenen Zahl entspricht, aus.

Adresse &H400B: GET PAR

BASIC Interpreterroutine, die eine 16 Bit Adresse und ein darauffolgenden, mit Komma abgetrennten, 8 Bitwert liest. Ausgabe:

DE - 8 Bitwert
BC - 16 Bitwert

Adresse &H521C: GET BYTE

Liest einen 8 Bitwert, der im Akku und im E Register zurückgegeben wird.

Adresse &H542F: GET ADR

Liest einen 16 Bitwert, der im DE Register zurückgegeben wird.

Adresse &H5439: GET ADR in Klammern

Liest einen in Klammern stehenden 16 Bitwert.

Adresse &H0103: Punktkoordinatengröße Text

Bei Übergabe der Punktkoordinaten für >SCREEN 2 oder 3<, wobei die X-Koordinate ins DE Register übergeben wird, prüft diese Routine, ob es sich um zulässige Koordinaten handelt.
C=0 : Koordinaten unzulässig
C=1 : Koordinaten zulässig

Ist >SCREEN 3< (Multicoulor) eingeschaltet, werden die Koordinaten durch 4 geteilt, da in diesem Modus 1 Punkt 4*4 Punkten in >SCREEN 2< entspricht.

Adresse &H0111: Berechnung Punktadresse

Bei Übergabe von X-Koordinaten in C und Y-Koordinaten in E wird die Punktadresse berechnet und ins HL Register und an Adresse &HF92A/B übergeben. Der Stellenwert des Bits, das dem zu setzenden Punkt entspricht, wird an Adresse &HF02C gespeichert.

Adresse &H0114: Punktadresse lesen

Die Adresse des zuletzt berechneten Punktes wird ins HL Register und der Bitwert des Punktes in den Akku geladen.

Adresse &H0117: Punktadresse Schreiben

Der derzeitige HL Wert wird als Punktadresse (&H92A/B) und der Akkuinhalt als Bitwert eines Punktes abgespeichert (&HA92C??).

Adresse &H011A: Farbcode Test

Es wird geprüft, ob der im Akku enthaltene Farbcode zulässig (<16) ist. Wenn ja, wird C rückgesetzt und die Farbe als aktuelle Punktfarbe an Adresse &HF3F2 gespeichert.

6.3 SYSTEMVERÄNDERUNGEN

Abschließend werden wir noch eine Möglichkeit kennenlernen das Betriebssystem bzw. den BASIC Interpreter zu verändern. Die Programmierer, die das MSX Betriebssystem entworfen haben, sahen dafür eine Technik vor, die man eingedeutscht als "patchen" (engl. to patch: flicken, überdecken) bezeichnet.

Das grundsätzliche Problem bei einer Veränderung des Systems besteht darin, daß ein ROM Speicher nicht veränderbar ist. Ein Eingriff ist nur in den Teilen möglich, die im RAM liegen. Damit Eingriffe an vielen Stellen möglich sind, wird von den wichtigen ROM Routinen aus, vor der eigentlichen Routine, mit einem Unterprogrammsprung ins RAM verzweigt. Befindet sich das System im Ausgangszustand, sind alle im RAM angesprungenen Adressen mit dem Code &HC9, dem Code für RET, geladen. Das bedeutet, daß der Aufruf dieser Adresse keinen Einfluß hat, und ohne Veränderungen das ROM Programm fortgesetzt wird. Dieser "Patchbereich" liegt im System RAM von &HFD9A bis &HFFC9.
Wollen wir eine Routine verändern, so besteht die Möglichkeit auf die zu der Routine gehörenden RAM Sprungadressen den "Patch" zu legen. D.h., daß der RET Befehl durch die Codes eines anderen Befehls, z.B. eines Sprungbefehls, ersetzt ("überdeckt") wird. Für jeden Patcheinsprung stehen 5 Bytes zur Veränderung zur Verfügung. Mehr dürfen auf keinen Fall geändert werden, da sonst schon die nächste Routine manipuliert wird. Um dieses Verfahren der Systemänderung zu verdeutlichen, betrachten wir ein Beispiel:

Aufgabe ist es, den >INPUT< Befehl des BASIC so zu verändern, daß kein Fragezeichen ausgegeben wird. Die Inputroutine beginnt an Adresse &H23CC. Übersetzten Sie ab Adresse &H23CC mit dem Disassembler:

```
23CC  CDE0F0     CALL  &HFDE0
23CF  3E3F       LD    A,&H3F
23D1  DF         RST   &H18
23D2  3E20       LD    A,&H20
23D4  DF         RST   &H18
23D5  .
  .   .
  .   .
```

Als ersten Befehl sehen Sie den Sprung in den RAM Patchbereich. Wollen wir die Inputroutine modifizieren, so stehen uns die Adressen &HFDE0 bis &HFDE4 zur Verfügung. Der Code für "?" wird im nächsten Befehl geladen und durch RST &H18 ausgegeben. Diese beiden Befehle sollen übersprungen werden. Das Programm soll an Adresse &H23D2 fortgesetzt werden. Der Patch sieht dann folgendermaßen aus:

```
Startadresse des Patch: &HFDE0
POP AF ; Rücksprungadresse holen
JP  &H23D2 ; Fragezeichen überspringen

Die Codes ergeben sich zu: F1,C3,D2,23
```

Folgendes Miniprogramm initialisiert den Patch:

```
10 POKE &HFDE1,&HC3
20 POKE &HFDE2,&HD2
30 POKE &HFDE3,&H23
40 REM Aktivierung:
50 POKE &HFDE0,&HF1
60 REM Abschalten mit &HFDE0,&HC9
```

Wichtig ist, daß die Anfangsadresse des Patches, in diesem Beispiel &HFDE0, als letztes mit dem neuen Wert geladen wird. Um die Veränderung rückgängig zu machen und wieder den Ausgangszustand herzustellen, muß nun wieder RET, also &HC9 an diese Stelle geschrieben werden.

Das folgende Programm gibt mit >PRINT< auf Drucker und Bildschirm gleichzeitig aus.

Das Programm modifiziert die RST &H18 Routine derart, daß sie zweimal aufgerufen wird. Einmal zur Bildschirm- und ein zweites Mal zur Druckerausgabe.

```
10 CLEAR 200,&HF300
20 POKE &HFEE5,0
30 POKE &HFEE6,&HF3
40 REM INIT mit POKE &HFEE4,&HC3
50 REM Ausschalten mit POKE &HFEE4,&HC9
60 REM Nicht im Direktmodus benutzen
70 FOR I=&HF300 TO &HF30B
80 READ A$
90 POKE I,VAL("&H"+A$)
100 NEXT
110 DATA F5,3A,61,F6,32,15,F4,F1
120 DATA CD,63,1B,C9
```

Assemblerlisting:

```
FEE4   JP &H3000; RST &H18 Patch

F300 F5      PUSH AF ; ASCII Code retten
F301 3A61F6 LD   A,(&HF661) ; POS nach
F304 3215F4 LD   (&HF415),A ; LPOS laden
F307 F1      POP  AF ; ASCII Code holen
F308 CD631B CALL &H363 ; Ausgabe auf Drucker
F30B C9      RET ; weiter mit Bildschirmausgabe
```

KAPITEL VII : PEEKS UND POKE

7.1 EINLEITUNG

In den vorhergehenden Kapiteln haben wir schon des öfteren PEEK's und POKE's benutzt. Im Folgenden soll kurz erklärt werden, was PEEK's und POKE's eigentlich prinzipiell sind. Dann folgt eine Liste der wichtigsten POKE Adressen mit ihrer jeweiligen Funktion und Anwendung.

Wie Sie wissen besitzen MSX Rechner 32K ROM. Dort ist das Betriebssystem enthalten, daß es Ihnen möglich macht auf so einfache Weise zu programmieren. Beim Ablaufen dieser internen, komplett in Maschinensprache geschriebenen Programme, fallen eine Menge Informationen an, die abgespeichert werden müssen. Das sind zum Beispiel:

- welcher Bildschirmmodus ist gewählt
- welche Farben sind gewählt
- Informationen über eigeschaltete BASIC Interrupts (>ON STRIG< etc.)
- Bildschirmzeilenlänge >WIDTH<

Aus der Notwendigkeit, diese Informationen zu speichern, ergeben sich folgende Konsequenzen:

- Da im ROM (Festwertspeicher) keine Informationen gespeichert werden können, muß ein Teil des RAM's hierfür verwendet werden. Beim MSX BASIC ist dies der RAM Bereich von &HF380 bis &HFFFF.

- Da es auf der Ebene der Maschinensprache keine Variablen in dem uns vom BASIC bekannten Sinne gibt, werden jeweils ein oder mehrere Speicherzellen des RAM's zur Speicherung der Informationen verwendet.

- Diese vom System festgelegten Speicherstellen werden vom System selbst benutzt und unter Umständen verändert. Da sie aber im RAM liegen, können wir sie mit >PEEK< auslesen und auch mit >POKE< verändern.

Unter Zuhilfenahme der PEEK's und POKE's sind oft Dinge möglich, die vom BASIC aus nicht vorgesehen sind. Da ein >POKE<, ein >PEEK< hingegen nicht, direkt in den Systemablauf eingreift, kann eine unvorsichtige Benutzung dieses Befehls leicht den Absturz oder zumindest ein Fehlverhalten des Rechners zur Folge haben.

7.2 DIE BEFEHLE

>PEEK Adresse< liest den Wert (1 Byte) der an der angegebenen Adresse steht.

>POKE Adresse,Wert< schreibt den angegebenen Wert (1 Byte) an die angegebene Adresse.

Oftmals bilden zwei aufeinanderfolgende Speicherstellen ein Paar, das zusammen gelesen/beschrieben werden soll. Dabei enthält die Speicherstelle mit der niedrigeren Adresse das Low Byte (siehe Kapitel 4) und die Speicherstelle mit der höheren Adresse das High Byte. Um den Wert der beiden Speicherstellen zu ermitteln, muß der Wert des High Bytes (HB) mit 256 multipliziert und der Wert des Low Bytes (LB) zum Ergebnis addiert werden. Für die Zerlegung einer 2-Byte Zahl schauen Sie bitte im Kapitel 4 nach.
Hier folgt nun die Liste einiger wichtiger POKE Adressen ihre Funktion und Anwendung.

7.3 DIE ADRESSEN

F3AE LINL40 LINien Länge Textmodus (max. 40 Zeichen)

Wird >SCREEN 0< gewählt, so wird >WIDTH< auf diesen Wert eingestellt.

Beispiel: >POKE &HF3AE,20:SCREEN 0<

F3AF LINL32 LINien Länge im Grafik I Modus (max.32 Zeichen)

Wird >SCREEN 1< gewählt, so wird >WIDTH< auf diesen Wert eingestellt.
Interessant ist, daß bei den beiden letztgenannten Adressen auch Werte größer als die Maximalwerte möglich sind. Die Eingabe eines solchen Wertes hat zur Folge, daß man mit dem Cursor seitlich aus dem Bildschirm laufen kann.

F3B0 LINLEN LINien LENgth (Länge)

Enthält die aktuelle Länge einer Bildschirmzeile (>WIDTH<). Dieser Wert kann direkt durch >POKE< geändert werden, was dem >WIDTH< Befehl ohne (!) >CLS< entspricht. Durch >SCREEN< wird die durch >WIDTH< geänderte Bildschirmeinstellung wieder auf die obigen Standardwerte gesetzt.

F3B1 CRTCNT Cathode Ray Tube CouNT- Bildschirmzeilenzähler

Anzahl der Zeilen pro Bildschirm. D.h. gleichzeitig, daß auf der letzten Zeile, deren Nummer hier gespeichert ist, die Ausgabe der Keybelegung erfolgt.

F3B2 TABCNT TABulator CouNTer

Enthält die Anzahl der Leerzeichen, die bei TAB ausgegeben werden.

F3B3/4 TXTNAM TeXT NAMenstabelle

Standardadresse der Namenstabelle im Textmodus

F3B7/8 TXTCGP TeXT Character Generator Patten

Standardadresse des Zeichenmuster Generators im Textmodus

F3BD/E T32NAM Textmodus (max.32 Zeichen) NAMenstabelle

Standardadresse der Namenstabelle in Modus Grafik I

F3BF/C0 T32COL Textmodus (max.32 Zeichen) COLor Tabelle

F3C1 T32CGP Textmodus (max.32 Zeichen) Character Generator Patten

F3C3/4 T32ATR Textmodus (max.32 Zeichen) Sprite ATtRibuttabelle

F3C5/6 T32PAT Textmodus (max.32 Zeichen) PATterntabelle

F3C7/8 GRPNAM GRaPhik NAMenstabelle

F3C9/A GRPCOL GRaPhik COLortabelle

F3CB/C GRPCGP GRaPhik Charakter Generator Patten

F3CD/E GRPATR GRaPhik ATtRibuttabelle

F3CF/D0 GRPPAT GRaPhik PATterntabelle

F3D1/2 MLTNAM MuLTicoulor NAMenstabelle

F3D5/6 MLTCGP MuLTicoulor Charakter Generator Pattern

F3D7/8 MLTATR MuLTicoulor ATtRibuttabelle

F3D9/A MLTPAT MuLTicoulor PATterntabelle

F3DB CLIKSW CLIcK SWitch

Tastaturbetätigungsgeräusch Flag 0 bedeutet aus.

F3DC CSRY CurSoR Y-Position

F3DD CSRX CurSoR X-Position

Die Cursor Positionen beziehen sich immer auf die aktuelle >WIDTH< Definition

F3DE FNDFLG FuNction Display FLaG

Benutzung siehe Kapitel Grafik

F3DF bis F3E6 RG0SAV-RG7SAV ReGister n SAVe

Hier wird der Inhalt der VDP Register 0 bis 7 gespeichert. Die VDP Register 0 bis 7 sind "nur Lese Register". Damit ihr aktueller Wert abgefragt werden kann, ist er hier gespeichert. Diese Speicherstellen sollten nicht durch >POKE< verändert werden, da das Verwirrung stiftet.

F3E8 TRGFLG TRiGger FLag

Bit 4 dieses Bytes ist gleich 0, wenn der Feuerknopf gedrückt wird

F3E9 FORCLR FORground CoLoR

Aktuelle Schriftfarbe

F3EA BAKCLR BAKground CoLoR

F3EB BDRCLR BorDeR CoLoR

F3F2 ATRBYT ATtRiBute BYTe

Aktuelle Farbe für hochauflösende Grafik

F3F6 SCNCNT SCaN CouNTer

Zähler, der jeweils von 3 bis 0 beim Durchlauf des Interrupts zählt. Bei 0 werden Interrupts von der Tastatur und von STRIG abgefragt.

F3F7 REPCNT REPeat CouNTer

Wiederholungszähler für die Tastaturrepeatfunktion. Normalerweise auf 13 gesetzt, wenn Taste gedrückt. Verhindert, daß eine Taste mehrmals registriert wird. Wird sofortige Repeatfunktion gewünscht REPCNT mit 1 laden. Siehe Programm Tastaturabfrage im Kapitel über I/O.

F3F8 PUTPNT PUT PoiNTer

Zeigt auf die Adresse, unter der die neue Tastatureingabe zwischengespeichert werden soll.

F3F7 GETPNT GET PoiNTer

Zeigt auf den aktuellen Anfang des Tastaturpuffers seit der letzten Abfrage des Puffers. Die Differenz von PUTPNT und GETPNT gibt die Anzahl der (inzwischen) gedrückten Tasten.

F414 ERRNUM ERRor NUMber

Nummer des zuletzt aufgetretenen Fehlers

F415 LPTPOS Line PrinTer POSition

Position des Druckkopfes in der aktuellen Zeile

F416 PRTFLG PRinT FLaG

Enthält PRTFLG eine 1, so gibt die allgemeine Ausgaberoutine OUTDO, die durch RST &H18 aufgerufen wird, auf den Drucker aus.

F417 NTMSXP NoT MSX Printer

Entspricht der beim >SCREEN< Befehl möglichen Eingabe über MSX oder nicht MSX Drucker

F418 RAWPRT RAW PRinT

Ist RAWPRT ungleich 0, so erfolgt die Druckausgabe ohne vorherige Aufbereitung (TAB,LPOS etc.).

F41C/D CURLIN CURrant LINe

Aktuelle Zeilennummer

F663 VARTYP VARiablen TYPenkennziffer

F676/7 BASSTA BASic STArtadresse

F6AA AUTOFL AUTO FLag

1 bedeutet AUTO ist angeschaltet
0 bedeutet AUTO ist ausgeschaltet.

F6AB/C aktuelle AUTO Zeile

F6AD/E Schrittweite für AUTO

F5B3 ERRLIN ERRor LINe

Enthält die Zeile, an der der letzte Fehler auftrat

F6C2 VARTAB VARiablen TABelle Startadresse

F6C4 VTBTAB Variablen TaBellen (Arrays-Felder) TABelle Start-adresse

F7C4 TROFLG TROn FLaG

0-TROFF; 1-TRON

F85F MAXFIL MAXFILes Anzahl

F91F CGSLOT Character Generator SLOT
Slotnummer in welcher die Kopie des Zeichensatzes steht

F929/1 CGPNT Character Generator PoiNTer

Adresse der CG-Kopie

F922/3 NAMBAS NAMenstabelle BASisadresse aktuell

F924/5 CAPBAS Character Generator BASisadresse aktuell

F926/7 PATBAS Sprite PATtern BASisadresse aktuell

F928/9 ATRBAS Sprite ATtRibur Tabelle BASisadresse

FBB0 ENSTOP ENable STOP

Ist ENSTOP=1, so ist ein Warmstart per Tastatur möglich, d.h. eine Unterbrechung, die sofort zum normalen Eingabemodus führt.
Schreiben Sie ein Programm, das gegen jede Unterbrechung geschützt ist (>ON STOP< etc.) und drücken Sie bei laufendem Programm SHIFT+CODE+GRAF+CTRL gleichzeitig sind Sie im Eingabemodus !
Schutz davor durch >POKE &HFBB0,0<

FBB1 BASROM BASicprogramm im ROM

Wenn BASROM <>0 ist, bedeutet das normalerweise, daß ein ROM BASICprogramm abläuft und deshalb jede Unterbrechung durch CTRL-C und CTRL-STOP unmöglich ist.
Dieser Schutz kann auch für eigen BASICprogramme benutzt werden >POKE &HFBB1,1<.

FBCC GODSAV CODe SAVe

Enthält den Code des Zeichens auf dem der Cursor steht

FC9E TIMER

entspricht der Variablen TIME

FCA9 CSRSW CurSoR ShoW- Cursor an/aus

FCAA CSTYLE Cursor STYLE

halbierter (insert-) Cursor an/aus

FCAB CAPST CAP STatus

Groß-/Kleinschrift an/aus

FCAC Sondertaste für Umlaute wurde gedrückt/nicht gedrückt

FCAF SCRMOD SCReen MODus

aktueller Screenmodus

FCB0 OLDSCR OLD SCRee

alter Modus (Text), wenn Grafik eingeschaltet

KAPITEL VIII : BASIC INTERN

8.1 EINLEITUNG

Im folgenden Kapitel werden wir uns mit der internen Abspeicherung des BASIC und der Variablen befassen. Die Beschäftigung mit diesem Thema stellt ein "Mittelding" zwischen Maschinensprache und reiner BASIC Programmierung dar.

Maschinensprache insofern, da der BASIC Interpreter in Maschinensprache geschrieben ist, und die internen Strukturen von BASIC Programmen und Variablen vom Interpreter verstanden und verarbeitet werden.

BASIC natürlich dadurch, daß es dabei um den Aufbau der BASIC Programme geht. Für einfaches Programmieren in BASIC ist die Kenntnis der internen Strukturen nicht unbedingt notwendig. Wie schon so oft, können jedoch durch die Ausnutzung dieser Strukturen oft Programme geschrieben werden, die "eigentlich gar nicht möglich" sind. Dazu gehören zum Beispiel:

- Erzeugen von BASIC Zeilen vom BASIC aus. Damit sind sozusagen "sich selbst schreibende" Programme möglich.

- Programmierhilfen, wie DUMP (Ausgabe aller Variablen und ihrer Werte), XREF (Verzeichnis aller Variablen und der Programmzeile, in denen Sie benutzt werden) können realisiert werden.

- Ein REM Killer kann programmiert werden. Ein solches Programm löscht alle Kommentare in einem Programm und fügt unter Umständen kurze Zeilen zu längeren zusammen. Dadurch können fertige Programme oft sehr beschleunigt werden.

Diese Beispiele sollen genügen, um zu zeigen, was mit den folgenden Informationen angefangen werden kann.

8.2 ABSPEICHERUNG VON BASIC ZEILEN

Da es sich bei BASIC Programmen um veränderliche Daten handelt, werden sie im RAM gespeichert. Die Startadresse des BASIC RAM's ist bei 32K und mehr Versionen &H8001, bei 16K Versionen &HC001. Grundsätzlich steht die Startadresse des BASIC RAM an den Adressen &HF676/77 als Low und High Byte. Die folgende BASIC Zeile gibt die Startadresse aus:

```
PRINT HEX$(PEEK(&HF676)+256*PEEK(&HF677))
```

Um den internen Aufbau eines BASIC Programms anzuschauen, benötigen wir ein Programm, das uns den Inhalt von Speicherplätzen übersichtlich ausgibt. Dieses Programm nennt man Monitor (siehe Kapitel 5). Damit Sie die folgenden Überlegungen nachvollziehen können, sollten Sie unbedingt das Monitorprogramm abtippen. Geben Sie als erste Zeile des Monitors zusätzlich folgende Zeile ein:

```
1 REM Data Becker Buch
```

Diese Zeile soll nun untersucht werden. Starten Sie den Monitor und geben Sie als Startadresse &H8000 und als Endadresse &H8017 ein. Sie erhalten folgenden Bildschirmausdruck:

```
8000 00 18 80 01 00 8F 20 44 ..... D
8008 61 74 61 20 42 65 63 6B ata Beck
8010 65 72 20 42 75 63 68 00 er Buch.
```

Die Bedeutung der einzelnen Bytes ist folgendermaßen:

Adresse	Byte	Bedeutung
&H8000	&H00	Nullbyte kennzeichnet den BASIC Anfang.
&H8001	&H18	
&H8002	&H80	Die beiden nächsten Bytes bestimmen die Speicheradresse, an der die nächste Zeile beginnt. Da zuerst immer das Low Byte steht, bedeutet dies in unserem Fall, daß die nächste Zeile ab Adresse &H8018 gespeichert ist (Verkettungsadresse).
&H8003	&H01	Auf die Verkettungsadresse folgt
&H8004	&H00	wieder in zwei Byte Darstellung die Zeilennummer, hier also 1.
&H8005	&H8F	Hier beginnt nun die erste Programmzeile. &H8F ist die verschlüsselte Darstellung (Token) für den REM Befehl.
&H8006	&H20	Nun folgt die Bemerkung nach dem REM. Sie beginnt mit einem Leerzeichen (Code &H20).
&H8007	&H44	Buchstabe D
&H8008	&H61	Buchstabe a
&H8009	&H74	Buchstabe t
&H800A	&H61	Buchstabe a
&H800B	&H20	Leerzeichen
&H800C	&H42	Buchstabe B
&H800D	&H65	Buchstabe e
&H800E	&H63	Buchstabe c
&H800F	&H6B	Buchstabe k
&H8010	&H65	Buchstabe e
&H8011	&H72	Buchstabe r
&H8012	&H20	Leerzeichen
&H8013	&H42	Buchstabe B
&H8014	&H75	Buchstabe u
&H8015	&H63	Buchstabe c
&H8016	&H68	Buchstabe h
&H8017	&H00	Dieses Nullbyte bedeutet, daß hier die aktuelle Zeile zuende ist.

Löschen Sie nun Zeile 1 und probieren noch eine andere Zeile:

```
2 PRINT "Data Becker"
```

Geben Sie wieder &H8000 und &H8017 für den Monitor ein. Der Aufbau ist ähnlich.

&H800 ist das Startbyte. Darauf folgt die 2 Byte Verkettungsadresse, dann die Zeilennummer (2 Byte) hier also 2, danach folgt &H91, das Token für PRINT. Anschließend kommen die Anführungszeichen (&H22) und dann der auszugebende Text (Data Becker). Nach dem Text folgen wiederum Anführungszeichen. Am Ende der Zeile steht, wie immer, das Nullbyte.
Die darauffolgenden 2 Bytes stellen die Verkettungsadresse der folgenden Zeile dar. Das Low Byte steht dabei an Adresse &H8014. Das ist genau die erste Verkettungsadresse (siehe oben). Anhand dieser Adressen kann man sich also durch das BASIC Programm hangeln. Damit das Prinzip klar wird folgt hier ein kleines Programm:

```
10 VA=&H8001: REM Startadresse BASIC
20  ZN=PEEK(VA+2)+256*PEEK(VA+3):REM nächste  Verkettungsadre
sse
30 VA=PEEK(VA)+256*PEEK(VA+1):REM nächste Verkettungsadress e
40 IF VA=0 THEN END
50 PRINT ZN: REM Zeilennummer ausgeben
60  GOTO 30:REM Vorgang mit neuer Verkettungsadresse  wiederh
olen
```

Falls bei Ihrem Rechner das BASIC nicht ab Adresse &H8001 beginnt, müssen Sie entsprechend Zeile 10 ändern.
Als Ergebnis erhalten Sie der Reihe nach alle Zeilennummern (wie bei List, allerdings ohne Inhalt) ausgegeben.
Zeile 30 liest die auf die Verkettungsadresse folgende Zeilennummer.

In Zeile 40 wird anhand der alten Verkettungsadresse, die ja immer auf die nächste zeigt, diese ausgelesen. Erhält man für den Wert einer Verkettungsadresse 0, so bedeutet dies, daß das BASIC Programm an dieser Stelle zuende ist. Ein Programm wird grundsätzlich durch 3 Nullbytes beendet. Dabei zeigt das erste Nullbyte das Ende der letzten Programmzeile an, und die beiden folgenden stellen die Verkettungsadresse 0 dar.
Das Ende des Programms wird in Zeile 50 festgelegt.
Damit haben wir bereits das "Gerüst" aller BASIC Programme kennengelernt:

```
1 Nullbyte     als BASIC Angangskennzeichen
2 Bytes        erste Verkettungsadresse
2 Bytes        erste Zeilennummer
.
.              Zeileninhalt
.
1 Nullbyte     als Zeilenendekennzeichen
2 Bytes        zweite Verkettungsadresse
2 Bytes        zweite Zeilennummer
.
.              Zeileninhalt
.
1 Nullbyte     Zeilenende
2 Nullbytes    für Ende des Programms
```

Beim Zeileninhalt spielen die sogenannten Token eine große Rolle.

8.3 TOKEN

Ein Token ist ein Code, der für einen BASIC Befehl steht. D.h., daß intern nie die Befehle selbst wörtlich abgespeichert werden - das wäre viel zu platzaufwendig. Aus diesem Grund werden nach beendeter Eingabe/Änderung einer Zeile sämtliche Befehlswörter erkannt und durch die dazugehörigen Codes (Tokens) ersetzt.

Es gibt einen einfachen Trick, um die Zuordnung zwischen Befehlswörtern und Token herauszufinden. Geben Sie wieder ein:

```
1 REM Hallo
```

Wie wir mit Hilfe des Monitors festgestellt haben, steht das Token für REM direkt hinter der Zeilennummer, also an Adresse &H8005(BASIC Start= &H8000) im Speicher. >PRINT PEEK(&H8005) ergibt 143=&H8F. Ersetzen wir dieses Token doch einfach durch ein anderes >POKE &H8005,&H91<. Wenn Sie jetzt >LIST 1< eingeben, so erhalten Sie:

```
1 PRINT Hallo
```

>POKE &H8005,&H84< ergibt

```
1 DATA Hallo
```

Also ist &H84 das Token für den >DATA< Befehl. Damit Sie nicht alles mühsam ausprobieren müssen, folgt nun die Liste der Token aller BASIC Befehlswörter.

DEZ	HEX	BEFEHLE	DEZ	HEX	BEFEHLE
129	81	END	130	82	FOR
131	83	NEXT	132	84	DATA
133	85	INPUT	134	86	DIM
135	87	READ	136	88	LET
137	89	GOTO	138	8A	RUN
139	8B	IF	140	8C	RESTORE
141	8D	GOSUB	142	8E	RETURN
143	8F	REM	144	90	STOP
145	91	PRINT	146	92	CLEAR
147	93	LIST	148	94	NEW
149	95	ON	150	96	WAIT
151	97	DEF	152	98	POKE
153	99	CONT	154	9A	CSAVE
155	9B	CLOAD	156	9C	OUT
157	9D	LPRINT	158	9E	LLIST
159	9F	CLS	160	A0	WIDTH
161	A1	ELSE	162	A2	TRON
163	A3	TROFF	164	A4	SWAP
165	A5	ERASE	166	A6	ERROR
167	A7	RESUME	168	A8	DELETE
169	A9	AUTO	170	AA	RENUM
171	AB	DEFSTR	172	AC	DEFINT
173	AD	DEFSNG	174	AE	DEFDBL
175	AF	LINE	176	B0	OPEN
177	B1	FIELD	178	B2	GET
179	B3	PUT	180	B4	CLOSE
181	B5	LOAD	182	B6	MERGE
183	B7	FILES	184	B8	LSET
185	B9	RSET	186	BA	SAVE
187	BB	LFILES	188	BC	CIRCLE
189	BD	COLOR	190	BE	DRAW
191	BF	PAINT	192	C0	BEEP

DEZ	HEX	BEFEHLE	DEZ	HEX	BEFEHLE
193	C1	PLAY	194	C2	PSET
195	C3	PRESET	196	C4	SOUND
197	C5	SCREEN	198	C6	VPOKE
199	C7	SPRITE	200	C8	VDP
201	C9	BASE	202	CA	CALL
203	CB	TIME	204	CC	KEY
205	CD	MAX	206	CE	MOTOR
207	CF	BLOAD	208	D0	BSAVE
209	D1	DSKO$	210	D2	SET
211	D3	NAME	212	D4	KILL
213	D5	IPL	214	D6	COPY
215	D7	CMD	216	D8	LOCATE
217	D9	TO	218	DA	THEN
219	DB	TAB(	220	DC	STEP
221	DD	USR	222	DE	FN
223	DF	SPC(	224	E0	NOT
225	E1	ERL	226	E2	ERR
227	E3	STRING$	228	E4	USING
229	E5	INSTR	230	E6	'(REM)
231	E7	VARPTR	232	E8	CSRLIN
233	E9	ATTR$	234	EA	DSKI$
235	EB	OFF	236	EC	INKEY$
237	ED	POINT	238	EE	>
239	EF	=	240	F0	<
241	F1	+	242	F2	-
243	F3	*	244	F4	/
245	F5	^	246	F6	AND
247	F7	OR	248	F8	XOR
249	F9	EQV	250	FA	IMP
251	FB	MOD	252	FC	Ü

In dieser Liste tauchen auch einige Befehle auf, die Sie wahrscheinlich nicht kennen. Dabei handelt es sich um Befehle, die erst angewandt werden können, wenn Sie eine Floppy an Ihren Rechner anschließen.

Wie Sie sehen, sind alle Token größer als &H80=128. Fast alle Zahlen bis 255 sind belegt, es fehlen aber noch einige Funktionen z.B. >PEEK<.
Geben Sie >POKE &H8005,255:POKE &H8006,&H97< und >LIST< ein.

Anstelle des >REM< Befehls, der ursprünglich in diesen Zeilen stand, ist nun die >PEEK< Funktion getreten. D.h. die Funktionen des MSX BASIC werden durch 2 Token verschlüsselt. Zuerst das Token mit dem Wert 255, es zeigt an, daß jetzt eine Funktion folgt. Dann folgt das eigentliche Token der Funktion. Im Folgenden die Liste der Funktionstoken (Zuvor muß immer 255 stehen!)

DEZ	HEX	BEFEHL	DEZ	HEX	BEFEHL
129	81	LEFT$	153	99	SPACE$
130	82	RIGHT$	154	9A	OCT$
131	83	MID$	155	9B	HEX$
132	84	SGN	156	9C	LPOS
133	85	INT	157	9D	BIN$
134	86	ABS	158	9E	CINT
135	87	SQR	159	9F	CSNG
136	88	RND	160	A0	CDBL
137	89	SIN	161	A1	FIX
138	8A	LOG	162	A2	STICK
139	8B	EXP	163	A3	STRIG
140	8C	COS	164	A4	PDL
141	8D	TAN	165	A5	PAD
142	8E	ATN	166	A6	DSKF
143	8F	FRE	167	A7	FPOS
144	90	INP	168	A8	CVI
145	91	POS	169	A9	CVS
146	92	LEN	170	AA	CVD
147	93	STR$	171	AB	EOF
148	94	VAL	172	AC	LOC
149	95	ASC	173	AD	LOF
150	96	CHR$	174	AE	MKI$
151	97	PEEK	175	AF	MKS$
152	98	VPEEK	176	B0	MKD$

Wie Sie sehen, ist bei den Funktionstoken noch platz. Mit einiger Probiererei kann man durch die Benutzung der "leeren" Token interessante Sachen machen.

Listschutz

Wir haben ein Token gefunden, mit dem sich ein einfacher Listschutz erzeugen läßt. Gehen wir wieder von Zeile >1 REM Hallo< aus. Geben Sie folgendes ein:

```
POKE &H8005,255
POKE &H8006,52
```

Nun listen Sie das Programm:
Beim Start des Listens wird automatisch ein >CLS< ausgeführt. Wenn in regelmäßigen Abständen derartig manipulierte >REM< Zeilen innerhalb eines Programms eingebaut sind (besonders nach wichtigen Stellen), ist es sehr aufwendig ein zusammenhängendes Listing auf den Bildschirm zu bekommen. Diese Zeilen dürfen aber nicht angesprungen werden, da das einen Syntax Error zur Folge hat. Man überspringt diese Zeilen z.B. durch:

```
RENUM
5 GOTO 20
```

Der Programmknacker kann aber diese Zeilen nun einfach löschen. Diesen Fall behandeln wir in der Kramecke (Kapitel IX).

Hier noch eine Anwendung

Programm: DATA Zeilengenerator

Per Tastatur eingegebene Daten (hier Worte) sollen fest mittels >DATA< Zeilen ins Programm geschrieben werden. So könnte z.B. eine Adressdatei geschrieben sein, deren Daten nicht (!) auf dem Kasettenrecorder sondern direkt im Programm gespeichert sind.
Die zu verändernde(n) >DATA< Zeile(n) müssen vorher im Programm vorhanden sein. Nehmen wir an, daß die >DATA< Zeile die Nummer 1000 hat und 15 Punkte als provisorische Daten enthält. Dann funktioniert folgendes Programm.

```
10 INPUT"Wort";A$
20 L=LEN(A$)
30 IF L>15 THEN 10
40 VA=&H8001
50 ZN=PEEK(VA+2)+256*PEEK(VA+3)
60 IF ZN=1000 THEN 100
70 VA=PEEK(VA)+256*PEEK(VA+1)
80 IF VA=0 THEN PRINT"Zeile 1000 fehlt":END
90 GOTO 50
100 AD=VA+4:REM Startadresse der DATA Zeile 1000
110  IF PEEK(AD)<>&H84 THEN PRINT "Zeile 1000 ist keine  DATA
Zeile":LIST 160
120 FOR I=1 TO L
130  IF PEEK(AD+I)<>46  THEN PRINT"Zu wenig  Punkte  in  DATA
Zeile 1000":LIST 1000
140 POKE AD+I,ASC(MID$(A$,I,1))
150 NEXT
160 FOR I=L+1 TO 15
170 IF PEEK(AD+I)<>46 THEN 130
180 POKE AD+I,32:NEXT
1000 DATA ...............
```

Nach Ablauf des Programms steht das von Ihnen eingegebene Wort im >DATA< Zeile 1000. Die Zeilen bis 90 haben wir besprochen.
Zeile 110 prüft auf das >DATA< Token.
In der ersten Schleife (Zeile 120 bis 150) wird das Wort Buchstabe für Buchstabe in die Zeile gepoked (Zeile 140). Vorher wird geprüft, ob an der jeweiligen Stelle noch ein Punkt steht (Zeile 130).
Die Schleife in den Zeilen 160 bis 180 füllt entsprechend den Rest der Zeile mit Leerzeichen.

Mit den Tokens sind alle möglichen Bytes größer als &H80 besprochen worden.
Bytes mit Werten zwischen 32 und 127 stellen in der internen BASIC Speicherdarstellung immer ihre ASCII Codes dar, d.h. sie stehen anstelle des jeweiligen Zeichens. Besonders zu beachten sind also wieder die Steuerzeichencodes von 0 bis 31. Nicht alle dieser Codes erfüllen eine bestimmte Funktion. Hier eine Zusammenstellung:

Code	Aufgabe
11	Kennzeichen für Octalzahlen (&O)
12	Kennzeichen für Hexadezimalzahlen (&H)
17	Ziffer 0
18	Ziffer 1
19	Ziffer 2
20	Ziffer 3
21	Ziffer 4
22	Ziffer 5
23	Ziffer 6
24	Ziffer 7
25	Ziffer 8
26	Ziffer 9
27	Zahl 10
28	Kennzeichen für 2 Byte Zahlen (INT)
29	Kennzeichen für 4 Byte Zahlen (SGN)
31	Kennzeichen für 8 Byte Zahlen (DBL)

Alle diese Codes sind in irgendeiner Weise mit der Speicherung von Zahlen verbunden. Mit dem Abspeichern von Zahlen und Strings im Programm und in Variablen werden wir uns jetzt befassen.

8.4 ZAHLENDARSTELLUNG

Es gibt vier verschiedene Arten der Darstellung von Daten im Computer:

1. Integer INT - Ganzzahl
2. Single Precision SNG - einfache Genauigkeit
3. Double Precision DBL - doppelte Genauigkeit
4. Strings STR - alphanumerisch

Integer

Zuerst wollen wir uns nur mit den ersten drei Arten, d.h. mit der Darstellung von Zahlen beschäftigen. Die erste Art, die der Ganzzahldarstellung, haben Sie bereits kennengelernt. Eine Integerzahl wird in Low Byte und High Byte zerlegt. Bit Nummer 7 des High Bytes wird als Vorzeichen benutzt. Null bedeutet positive und 1 negative Zahlen. Die negativen Zahlen werden im Zweierkomplement dargestellt, d.h. der Betrag der Zahl wird komplementiert und 1 addiert. Damit können INTEGER Konstanten oder Variablen ganzzahlige Werte von -32768 bis +32767 erhalten.

dezimal	binär	hex
-32768	1 000 0000 0000 0000	80 00
-32767	1 000 0000 0000 0001	80 01
-32766	1 000 0000 0000 0010	80 02
-32765	1 000 0000 0000 0011	80 03
	...	
-2	1 111 1111 1111 1110	FF FE
-1	1 111 1111 1111 1111	FF FF
0	0 000 0000 0000 0000	00 00
1	0 000 0000 0000 0001	00 01
2	0 000 0000 0000 0010	00 02
	...	
32766	0 111 1111 1111 1110	7F FE
32767	0 111 1111 1111 1111	7F FF

Um intern die verschiedenen Darstellungsarten zu unterscheiden, ist jeder von ihnen eine Typenkennziffer zugeordnet. Diese Kennziffer ist gleich der Anzahl der Bytes, die zur Abspeicherung des jeweiligen Typs notwendig sind. Zum Speichern einer Integerzahl sind 2 Bytes nötig; also ist die Typenkennziffer für Integerkonstanten bzw. Variablen 2. Das Kennzeichen der aktuellen Variablen ist immer an Adresse &HF663 abgespeichert.

Fließkommadarstellung

Die Fließkommazahlen werden im MSX Betriebssystem in einer, für diese Klasse von Rechnern, ungewöhnlichen Art gespeichert: Dem BCD Format.

Beim BCD Format (Binär Codiert Dezimal) werden die einzelnen Ziffern einer Dezimalzahl abgespeichert. Der Vorteil liegt in einer genau festgelegten Anzahl von Stellen, die verarbeitet werden. Auch der Wertebereich ist beim MSX System größer (10^-64 bis 10^62), als bei der üblichen Speichermethode (10^-32 bis 10^31). Leider können Zahlen im BCD Format intern

nicht so schnell verarbeitet werden. Aus diesem Grund gibt es zwei verschiedene Genauigkeiten: SGN mit 6 Stellen und DBl mit 14 Stellen.
Die Darstellung mit einfacher Genauigkeit (SGN) wird in fast allen Fällen den erforderlichen Ansprüchen genügen.

Wie wird nun eine Zahl im BCD Format abgespeichert?

Bevor wir uns mit der Ziffernspeicherung beschäftigen, besprechen wir die Exponentialdarstellung von Zahlen, wie sie z.B. vom Taschenrechner bekannt ist. Soll eine Zahl in der Exponentialschreibweise dargestellt werden, so wird zuerst festgestellt, wie oft die Basis des Dezimalsystems, also die Zehn, in der Zahl als Faktor enthalten ist. Anschließend wird die Zahl in zwei Anteile zerlegt. Der eine Teil enthält alle Ziffern der Zahl (Mantisse), wobei das Komma immer hinter der ersten Ziffer steht. Der zweite Teile gibt nun an, wie oft die Zehn in die urspüngliche Zahl hineinpaßt oder, anders ausgedrückt, um wieviele Stellen das Komma im ersten Teil verschoben werden muß (Exponent). Man schreibt folgendermaßen:

```
   27=2.7*10^2
3956=3.956*10^4
```

Mit der Vereinbarung, daß negative Exponenten eine Verschiebung des Kommas in die entgegengesetzte Richtung, also nach links bedeutet, können alle Zahlen dargestellt werden:

```
  0.21=2.1*10^-1
0.0051=5.1*10^-3
    -9=-9.0*10^0
```

Mit dieser Vereinbarung läßt sich jede Zahl in eine Mantisse (Ziffernteil, vor dem Komma nur eine Stelle) und einen Exponenten zerlegen. Die Exponentialschreibweise hat den Vorteil, daß auch sehr große und sehr kleine Zahlen mit relativ kleinem Aufwand gespeichert werden können:

```
0.00000000000000735=7.35*10^-15
   6390000000000000=6.39*10^15
```

Beim Rechnen mit diesen Zahlen gelten die üblichen Regeln der Exponentialrechnung.

Addition/Subtraktion

Addiert werden können nur Zahlen mit gleichen Exponenten. Sind die Exponenten der Summanden verschieden voneinander, so wird die Zahl mit dem kleineren Exponenten auf den größeren umgeschrieben. Sind die Exponenten dann gleich, können einfach die Mantissen addiert bzw. subtrahiert werden.

Beispiel:

```
          57 + 0.31
5.7*10^1 + 3.1*10^-1
5.7*10^1 + 0.031*10^1
    5.731*10^1
    57.31
```

Multiplikation/Division

Bei den Punktrechnungsarten werden die Mantissen multipliziert bzw. dividiert und dann die Exponenten addiert bzw. subtrahiert.

Beispiel:

```
          0.13 * 20
     1.3*10^-1 * 2.0*10^1
       1.3*2.0 * 10^1(-1+1)
           2.6 * 10^0
           2.6


            25 / 0.05
     2.5*10^1 / 5.0*10^-2
       2.5/5.0 * 10^(1-(-2))
           0.5 * 10^3
           5.0 * 10^2
           500
```

Wollen wir dieses Verfahren auf einen Mikroprozessor übertragen, so stellt sich die Aufgabe, Zahlen in diesem Format abzuspeichern. Beim BCD Format wird jede Ziffer der Mantisse einzeln abgespeichert.
Eine Ziffer des Dezimalsystems hat Werte von 0 bis 9=&B1001. Also sind vier Bits zum Speichern einer Ziffer notwendig. Mit einem Byte = 8 Bits können demnach zwei Ziffern codiert werden, wobei das High Nibbel (Bit 4 bis 7) der ersten und das Low Nibble (Bit 0 bis 3) der zweiten Ziffer entspricht.

Beispiel:

```
Dezimal     : 27
High Nibble: 2 =&B10
Low Nibble :  7=&B111
BCD Format : &B0010 0111=&H27=39
```

Aufgrund der besonderen Eigenschaften des Hexadezimalsystems, daß 4 Bit je einer Hexziffer entsprechen, ist immer genau der Wert, der als Hexzahl interpretierten Dezimalzahl, der BCD codierte Wert.

BCD Wert von 63 ist &H63=&B0110 00111=99

Die Anzahl der zu speichernden Ziffern ist theoretisch unbegrenzt. Sie wird durch die Anzahl, der zur Abspeicherung verwendeten Bytes, festgelegt. Bei der Darstellung von Zahlen mit einfacher Genauigkeit (SNG), beträgt die Anzahl der Bytes, die zur Speicherung der Ziffern der Mantisse benutzt werden, drei. Damit haben Zahlen dieses Typs eine Genauigkeit von 6 Ziffern, nämlich 2 pro Byte.
Die Zahl 123794 wird im BCD Format als Folge von 3 Bytes gespeichert:

&H12, &H37, &H94

Wird die doppelte Genauigkeit gewählt, so stehen 7 Bytes zur Verfügung, was eine Stellenzahl von 14 ermöglicht.
Damit ist klar, wie die Mantisse von Fließkommazahlen in Exponentialdarstellung intern abgespeichert wird.
Bei der Codierung des Exponenten ist man einen anderen Weg gegangen: Er wird direkt durch den Wert eines Bytes dargestellt. Allerdings müssen noch Informationen über

- Vorzeichen der Mantisse und
- Vorzeichen des Exponenten

in diesem Byte untergebracht werden. Bit 7 kennzeichnet in üblicher Weise das Vorzeichen der Zahl (der Mantisse):

0- positiv
1- negativ

Die restlichen 7 Bit stellen den Wert des Exponenten dar, wobei zu dem realen Wert &H41 addiert wurde:

Wert Bit 6-0	Wert Exponent
&H7F	&H3E = 62
&H7E	&H3D = 61
.	.
.	.
.	.
&H42	&H01 = 1
&H41	&H00 = 0
&H40	-&H01 =-1
.	.
.	.
.	.
&H02	-&H3F =-63
&H01	-&H40 =-64
&H00	bedeutet Zahl = 0

Also kann der Exponent Werte von -64 bis +62 annehmen, ein Zahlenbereich, der für jede Berechnung ausreicht. Vereinbarungsgemäß wird bei einer Zahl mit dem Wert 0, der Exponent auf 0 gesetzt.

Testen wir diese Form der Darstellung mit einem BASIC Programm. Dabei wird der Befehl >VARPTR< verwendet. Diese Funktion gibt uns die Adresse im Speicher, an der der Wert der Variablen in der oben beschriebenen Form codiert, steht. Zur Verwaltung der Variablen, folgt direkt auf das BASIC Programm, ein Bereich, wo zu jeder benutzten Variablen ihr Name und Wert gespeichert ist. An der Adresse, die die >VARPTR< Funktion angibt, steht das erste Byte der codierten Zahl, welches vereinbarungsgemäß der Exponent ist. Darauf folgen, je nach Typ, die 3 (SNG) bzw. 7 (DBL) Bytes, die die Ziffern darstellen.

```
10 X!=43546!
20 AD=VARPTR(X!)
30 PRINT HEX$(PEEK(AD))
40 FOR I=AD+1 TO AD+3
50 PRINT RIGHT$("0"+HEX$(PEEK(I)),2);
60 NEXT
```

```
10 X#=.012342349#
20 AD=VARPTR(X#)
30 PRINT HEX$(PEEK(AD))
40 FOR I=AD+1 TO AD+6
50 PRINT RIGHT$("0"+HEX$(PEEK(I)),2);
60 NEXT
```

Setzen Sie verschiedene Zahlen für X ein und beobachten Sie die jeweilige Ausgabe der Programme.

Nachdem wir jetzt die Zahlen besprochen haben, schauen wir uns einen weiteren, vollkommen anderen Variablentyp an, die Stringvariable:

8.5 STRINGS

In einem String werden alphanumerische Daten, d.h. Zeichen, wie Buchstaben oder Ziffern gespeichert. Jedem Zeichen ist dabei ein Code zugeordnet. Die MSX Zeichencodes entsprechen, für die Codes von 0 bis 127, weitgehend dem sog. American Standard Code for Information Interchange (ASCII).

Zum Speichern eines Zeichens ist also genau ein Byte notwendig. Ein "String" (engl.string: Schnur, Kette), ist nun eine Kette von aufeinanderfolgenden Zeichencodes. Da ein Zeichen einem Byte entspricht, wird ein String in einer Reihe aufeinanderfolgender Speicherstellen im RAM gespeichert. Um einer bestimmten Variablen einen bestimmten String zuzuordnen, sind zwei Informationen notwendig, die den sog. String Descriptor bilden:

1. Die Adresse der ersten Speicherstelle, die den ersten Zeichencode des Strings enthält.

2. Die Länge des Strings, also die Anzahl der Bytes, die die Zeichenkette bilden.

Diese beiden Daten werden zusammen mit dem Variablennamen im BASIC Variablenbereich abgespeichert. Die eigentliche Zeichenkette steht an anderer Stelle, entweder im Programm selbst oder in dem speziell dafür reservierten Stringbereich. Die Größe dieses Bereichs kann durch den >CLEAR< Befehl festgelegt werden.
Im BASIC können wir wieder den >VARPTR< Befehl benutzen, um die Adresse des Stringdescriptors einer Variablen zu lesen. Der Stringdescriptor besteht aus drei Bytes:

1. Byte : Länge des Strings
2.+3. Byte : Startadresse des Strings

Die >VARPTR< Funktion gibt die Adresse des ersten Bytes des Stringdescriptors aus.
Probieren Sie folgendes Programm aus:

```
10 X$="ZZZeichenkeTTTTe"
20 AD=VARPTR(X$)
30 LA=PEEK(AD)
40 ST=PEEK(AD+1)+256*PEEK(AD+2)
50 FOR I=ST TO ST+LA-1
60 PRINT CHR$(PEEK(I));
70 NEXT
```

8.6 VARIABLENTABELLE DES BASIC

Abschließend besprechen wir kurz, wie die BASIC Variablen mit Hilfe der bereits erwähnten Variablenseite, die auf das Programm folgt, verwaltet werden.
An der Adresse &HF6C2/3 steht die Startadresse der Variablentabelle.
Es gilt folgendes Schema in der Variablentabelle:

1.Byte: Typenkennziffer (gleichzeitig Anzahl der Bytes)
2.Byte: Variablenname
3.Byte: in ASCII Code Darstellung
ab
4.Byte: Wert der Variablen

Im Einzelnen gilt folgendes:

Typ INT

02 Kennzeichen INT
.. Variablenname
.. in ASCII Codes
Low Byte
High Byte

Typ SGN

04 Kennziffer SGN
.. Name
..
4 Bytes für Wert

```
Typ DBL

08  Typenkennziffer DBL
..  Name
..
8 Bytes für Wert

Typ String

03  Typenkennziffer String
..  Name
..
3 Bytes Stringdescriptor
       1tes Byte: Länge
       2tes und 3tes Byte: Adresse
```

Damit können wir uns ein einfaches Programm zur Ausgabe aller benutzten Variablen schreiben.

Dieses Programm können Sie an Ihre eigenen Programme anhängen und zu Testzwecken aufrufen. Eine Erweiterung des Programms insofern, daß auch die Werte der Variablen angegeben werden, dürfte kein Problem sein.

```
1000 REM Variablen DUMP
1010  AD=PEEK(&HF6C2)+256*PEEK(&HF6C3):REM Anfang  Variablent
abelle
1020  AE=PEEK(&HF6C4)+256*PEEK(&HF6C5):REM Ende  Variablentab
elle
10030 TY=PEEK(AD):REM Typenkennziffer
10040 REM Namen ausgeben
10050 PRINT CHR$(PEEK(AD+1));CHR$(PEEK(AD+2));
10060 REM Typ ausgeben
10070  IF TY=2 THEN PRINT"%";ELSE IF TY=3 THEN  PRINT"$";ELSE
IF TY=4 THEN PRINT"!";ELSE PRINT"#";
10080 PRINT
10090 AD=AD+3:REM Start des Variablenwertes
10100 AD=AD+TY:REM Start der naechsten Variablen
10110 IF AD>AE THEN END ELSE 10030
```

KAPITEL IX : KRAMECKE

9.1 DER DEEK BEFEHL

In einigen BASIC "Dialekten" gibt es den Befehl >DEEK<. >PEEK< liest den 2 Byte Wert von zwei aufeinanderfolgenden Speicherstellen. Er entspricht damit den Befehlen >PEEK (Adresse)+256*PEEK(Adresse+1)<. Der >DEEK< Befehl soll jetzt mit Hilfe des >USR< Befehls in das MSX BASIC implementiert werden. Die Adresse des Low Bytes, der zu lesenden Speicherstellen soll übergeben werden. Am Anfang des Programms wird geprüft, ob der übergebene Parameter vom richtigen Typ ist. Wenn dies nicht der Fall ist, soll zur Ausgabe von "Typ Mismatch Error" gesprungen werden.
Dazu wird das E Register mit der Nummer des jeweiligen Fehlers geladen und nach &H406F (Routine zur Fehlerausgabe verzweigt.

```
CP 2 ; Typ 2
JR Z,OK ; Ja, dann OK
LD E,13 ; Nein, dann
JP &H406F ; Typ mismatch Error
OK ...
```

Allerdings gibt es für jede Fehlermeldung auch eine eigene Einsprungadresse, die das Laden des E Registers erledigt. Für den "Typ Mismatch Error" ist das die Adresse &H406D. Damit vereinfacht sich das Programm zu:

```
CP 2 ; Typ 2 ?
JP NZ,&H406D ; Nein, dann "Typ Mismatch Error"
   .
   .
```

Auf diese Weise hat die Lösung der Typenkontrolle einen Nachteil: Gibt der Benutzer vom BASIC aus eine Zahl richtiger Größe, aber von falschem Typ ein, z.B. kann 1000 sowohl als Integer als auch als SGN- oder DBL-Typ gespeichert werden, so

würde das zu einem "Typ Mismatch Error" führen. Es ist jedoch möglich, eine solche Zahl in eine Integerzahl zu verwandeln. Im BASIC erledigt das die Funktion >CINT<.
Wenn wir also am Anfang des Maschinenprogramms die >CINT< Routine aufrufen, so wird diese Umwandlung automatisch durchgeführt. Ein Fehler der oben beschriebenen Art tritt dann nur noch bei Eingabe von Strings auf. Außerdem prüft die >CINT< Routine, ob es sich bei der Eingabe um eine Zahl der richtigen Größe handelt, sonst wird die Fehlermeldung "Overflow" ausgegeben.
Benötigen Sie Variablen anderen Typs, so benutzen Sie die Routinen CSNG (&H2FB2) und CDBL (&H303A). Beachten Sie die Tabelle am Ende des Buches, in der alle behandelten und außerdem noch viele weitere Systemroutinen aufgeführt sind.

Nun zur Ausführung des >DEEK< Befehls.

```
10 ' CALL &H2F8A ; CINT konvertiert nach INT
20 ' LD HL,(&HF7F8) ; übergebener Parameterwert=Adresse
30 ' LD E,(HL) ; Low Byte
40 ' INC HL
50 ' LD D,(HL) ; High Byte
```

In Zeile 20 wird der zu übergebende Parameterwert, also die Adresse, ab der der 2-Bytewert gelesen werden soll, von den Übergabeadressen &HF7F8 und &HF7F9 gelesen. In den Zeilen 30 bis 50 wird der 2-Bytewert, der an diesen Adressen steht, ins DE Register geladen.
Nun müssen wir den ermittelten Wert wieder ins BASIC zurückübergeben. Dazu muß der zu übergebende Wert an die Adresse &HF7F8/9 geladen werden. Außerdem muß der Akku die Typenkennziffer enthalten, und HL muß mit der Adresse &HF7F6 geladen werden.

```
60 ' LD (&HF7F8),DE ; Ergebnis von "DEEK"
70 ' LD HL,&HF7F6
80 ' RET
```

Assemblieren Sie das Programm und probieren Sie die neue Funktion aus:

```
PRINT USR1(2)
```

ergibt 370, dasselbe Ergebnis wie

```
PRINT PEEK(2)+256*PEEK(3).
```

Das bedeutet also, daß ab Adresse 2 der 2-Bytewert 370 oder &H207 steht.

```
 F000             10                ; DEEK Befehl
 F000             20                ; gibt 2-Byte-Wert ab
uebergebener Adresse
 F000 CD8A2F      30          CALL &H2F8A ; CINT
 F003 2AF8F7      40          LD   HL,(&HF7F8) ; uebergebe
ne Parameterwert=Adresse
 F006 5E          50          LD   E,(HL) ; Low Byte lesen
 F007 23          60          INC  HL
 F008 56          70          LD   D,(HL) ; High Byte lese
n
 F009 ED53F8F7    80          LD   (&HF7F8),DE ; Ergebnis
von DEEK
 F00D 21F6F7      90          LD   HL,&HF7F6
 F010 C9         100          RET
---------------------------------------

Programm :deek
Start : &HF000    Ende : &HF010
Lnge : &H11 Bytes
Fehler : 0
```

```
10 REM DEEK Befehl
20 CLEAR 200,&HEFFF
30 FOR I=&HF000 TO &HF010:READ A$
40 POKE I,VAL("&H"+A$):NEXT
50 DEFUSR1=&HF000
60 DATA CD,8A,2F,2A,F8,F7,5E,23
70 DATA 56,ED,53,F8,F7,21,F6,F7,C9
```

9.2 DER UPPER BEFEFEHL

Mit Hilfe der Maschinensprache ist es möglich, z.B. den, in einigen BASIC Dialekten bekannten, >UPPER< Befehl zu realisieren. Der >UPPER< Befehl wandelt alle Kleinbuchstaben eines Strings in Großbuchstaben um.

```
10 CLEAR 200,&HEFFF:MAXFILES=1
20 FOR I=&HF000 TO &HF021:READ A$
30 W=VAL("&H"+A$):S=S+W
40 POKE I,W:NEXT
50 IF S<>3814 THEN PRINT"Fehler in DATAs":END
60 PRINT"Alles klar !"
70 DEFUSR1=&HF000
80 DATA FE,03,C2,6D,40,1A,B7,C8
90 DATA 47,62,6B,23,7E,23,66,6F
100 DATA 7E,FE,7B,30,06,FE,61,38
110 DATA 02,E6,DF,77,23,10,F1,3E
120 DATA 03,C9
```

```
 F000              10                ; UPPER - BEFEHL
 F000              20          ORG   &HF000
 F000              30  TYPERR  EQU   &H406D ; Typ mismatch E
rror
 F000 FE03         40          CP    3 ; Stringvariable ?
 F002 C26D40       50          JP    NZ,TYPERR ; nein, dann
Typ mismatch Error
 F005 1A           60          LD    A,(DE) ; Laenge des Str
ings
 F006 B7           70          OR    A ; Z-Flag setzen wenn
A=0
 F007 C8           80          RET   Z ; Fertig wenn Laenge=
0
 F008 47           90          LD    B,A ; Laenge als Schlei
fenzaehler
 F009 62          100          LD    H,D ; Deskriptoradresse
 F00A 6B          110          LD    L,E ; nach HL
 F00B 23          120          INC   HL
 F00C 7E          130          LD    A,(HL) ; Adresse
 F00D 23          140          INC   HL ; des
 F00E 66          150          LD    H,(HL) ; Stringanfangs
 F00F 6F          160          LD    L,A ; nach HL
 F010 7E          170  SCHLEI  LD    A,(HL) ; wenn ASCII Cod
e
 F011 FE7B        180          CP    123 ; >= als ASCII("z")
+1
?F013 30FE        190          JR    NC,OK ; dann nicht umwa
ndeln
 F015 FE61        200          CP    97 ; wenn < ASCII("a")
?F017 38FE        210          JR    C,OK ; dann nicht umwan
deln
 F019 E6DF        220          AND   &B11011111 ; Umwandlung
**** Zeile  190  : OK=&HF01B Offset 6
**** Zeile  210  : OK=&HF01B Offset 2
 F01B 77          230  OK      LD    (HL),A ; ASCII wieder S
peichern
 F01C 23          240          INC   HL ; Adresse des naechs
ten Codes
 F01D 10F1        250          DJNZ SCHLEI ; bis Stringende
 wiederholen
 F01F 3E03        260          LD    A,3 ; Typenkennziffer f
uer String
 F021             270                ; DE enthaelt alte Des
kriptoradresse
 F021 C9          280          RET
-------------------------------------

Programm :upper
Start : &HF000     Ende : &HF021
Lnge : &H22 Bytes
Fehler : 0
Variablentabelle :
TYPERR 406D    SCHLEI F010
OK     F01B
```

9.3 DER SLOT PEEK BEFEHL

Im folgenden die "Theorie" zum Slot PEEK Befehl vom Monitor. Bei dem anschließendem Text handelt es sich um einen Auszug aus dem "Maschinensprachebuch zu MSX" von DATA Becker.

Ein BASIC Programm wird nach der Eingabe vom Programmierer in einen Zwischencode übersetzt. Es werden z.B. die Befehlsworte nicht als solche abgespeichert, sondern durch einen Kurzcode (1 Byte lang; >128) ersetzt, das sog. Token. Erkennt der Interpreter ein Token, so springt er in die, diesem Token (=Befehl) zugeordnete, Routine. Dort werden dann die, eytl. hinter dem Befehl stehenden, Parameter eingelesen und auf Richtigkeit geprüft. Damit wollen wir uns nun genauer beschäftigen:

Greifen wir als Beispiel den >POKE< Befehl heraus. Hinter dem >POKE< Befehl stehen zwei durch ein Komma getrennte Parameter; die Adresse und der Wert. Nehmen wir einmal an, daß das Token für >POKE< (=152) erkannt wurde und daß das Programm zur >POKE< Routine (&H5423) verzweigt. Grundsätzlich zeigt bei der BASIC Interpretation das HL Register immer auf die aktuell bearbeitete Stelle im BASIC Programm. HL dient als BASIC Programm Pointer. Es muß immer gewährleistet sein, daß dieser Pointer nicht "verlorengeht", da sonst die Interpretation nicht korrekt fortgesetzt wird.
Für das Einlesen oben genannter Parameter, gibt es im Interpreter spezielle Routinen, die wir natürlich auch benutzen können.
Die beim >POKE< Befehl zunächst aufgerufene Routine, heißt GETADR. Sie liest einen Integerparameter. Bei Aufruf dieser Routine, muß HL mit dem aktuellen BASIC Pointer geladen werden. Von dieser werden Ausdrücke wie, 1237, oder komplexe Ausdrücke, wie

```
ABS(INT(VAL(RIGHT$(STRING$(4,"0")+HEX$(x),4))))
```

gleichermaßen ausgewertet.
Der ermittelte 2-Bytewert wird im DE Register zurückgegeben.

Der BASIC Pointer HL zeigt nach dem Ende der Routine in unserem Fall auf das folgende Komma. Die GETADR Routine hat die Adresse &H542F.
Als nächstes wird geprüft, ob jetzt ein Komma folgt. Auch hierfür gibt es natürlich eine Systemroutine: Der RST &H08. Dieser Restart dient zum Prüfen auf ein beliebiges Zeichen. Der ASCII Code des Zeichens muß dazu mit DB direkt hinter dem RST &H08 Befehl stehen. Für den >POKE< Befehl also:

```
        .
        .
RST &H08 ; Test auf ","
DB  &H2C ; Code für ","
        .
        .
```

Wird das Zeichen nicht gefunden, so wird ein "Syntax Error" ausgegeben, ansonsten erfolgt der Rücksprung.
Die Routine, die dann aufgerufen wird, um den 8-Bitwert einzulesen, nennen wir GETBYT. Der 1-Bytewert wird im Akku zurückgegeben. Nachdem diese Routine abgearbeitet ist, zeigt das HL Register als BASIC Pointer auf das Ende des Befehls. Mit Hilfe dieser Routinen ist es möglich, den >USR< Befehl so zu erweitern, daß beliebig viele Parameter übergeben werden können. Das folgende Programm benutzt sie, um eine modifizierte >PEEK< Funktion zu erzeugen, bei der zusätzlich zur Adresse , die gelesen werden soll, noch die Slotnummer angegeben wird. Damit besteht die Möglichkeit, Erweiterungs- oder auch eingebaute überlappte Module auszulesen.
Das eigentliche Ausführen wird durch die Routine ab &H000C erledigt (SLOT RD). Wird im Akku die Slotnummer, und im HL Register die Adresse übergeben, so gibt diese Routine im Akku den Wert des gelesenen Bytes zurück.
Der BASIC Pointer kann in einem selbsterstellten Programm durch zweimaliges "POP" vom Stapel gelesen werden. Wichtig ist, daß der Stapel, insbeseondere die Rücksprungadresse, immer wieder richtig rekonstruiert wird.
Hier folgt nun das Assemblerlisting:

```
 F000              10                    ; Slot PEEK
 F000              20                    ; Format: USR(Adresse)
,(Slot)
 F000              30          ORG   &HF000
 F000              40   ILLQUA EQU   &H475A
 F000              50   CINT   EQU   &H2F8A
 F000              60   GETBYT EQU   &H521C
 F000              70   SLOTRD EQU   &H000C
 F000              80                 ;
 F000 CD8A2F       90          CALL CINT ; convertiert nach
 INTEGER und laedt Wert nach HL
 F003 EB          100          EX   DE,HL
 F004 C1          110          POP  BC ; Ruecksprungadresse
 F005 E1          120          POP  HL ; BASIC Pointer
 F006 E5          130          PUSH HL ; Stack wieder-
 F007 C5          140          PUSH BC ; herstellen
 F008 D5          150          PUSH DE ; Wert der Adresse
 F009 CF          160          RST  &H08 ; Test auf
 F00A 2C          170          DB   &H2C ; ASCII(",")
 F00B CD1C52      180          CALL GETBYT ; Holt Slot Nr.
 F00E FE03        190          CP   3 ; Slot Nr. >= 3 ?
 F010 D25A47      200          JP   NC,ILLQUA ; ja, dann Il
legal Quantity
 F013 E3          210          EX   (SP),HL ; zu lesende Ad
resse mit BASIC Pointer Tauschen
 F014 CD0C00      220          CALL SLOTRD ; Slot Read
 F017 CDCF4F      230          CALL &H4FCF ; Akku in FAC la
den
 F01A E1          240          POP  HL ; BASIC Pointer
 F01B C1          250          POP  BC ; Ruecksprung
 F01C D1          260          POP  DE ; alter BASIC Pointe
r
 F01D E5          270          PUSH HL ; neuer BASIC Pointe
r
 F01E C5          280          PUSH BC ; Ruecksprung
 F01F 21F6F7      290          LD   HL,&HF7F6 ; Startadress
e FAC
 F022             300                  ; A enthaelt bereits T
yp 2 (siehe Routine ab &H4fcf)
 F022 C9          310          RET
-----------------------------------------

Programm :sltrd
Start : &HF000    Ende : &HF022
Lnge : &H23 Bytes
Fehler : 0
Variablentabelle :
ILLQUA 475A   CINT   2F8A
GETBYT 521C   SLOTRD 000C
```

9.4 DER CLEAR BEFEHL

Bei unserem MSX Rechner funktionierte der CLEAR Befehl nicht einwandtfrei:

```
10 CLEAR 200,&HF000
20 INPUT B$
30 FOR I=32 TO 110
40 FOR J=32 TO Ii
50 A$=A$+CHR$(J)
60 NEXT J
70 PRINT B$
80 A$=""
90 NEXT I
100 END
```

Dieses Programm sollte nach Eingabe eines Wortes (mind. 2 Zeichen) eigentlich immer wieder das Wort ausgeben (Zeile 70). Sollten bei Ihrem Rechner auch Schwierigkeiten auftreten (z.B. aus "Baum" wird aufeinmal "mmmm"), dann sollten Sie hinter jeden >CLEAR< Befehl ein >MAXFILES< Befehl stellen, also:

```
10 CLEAR 200,&HF000:MAXFILES=1
```

Nun sollte alles richtig ablaufen.
Der Fehler liegt darin, daß nicht alle Zeiger für den BASIC Stringvariablenbereich korrekt gesetzt werden. Durch >MAXFILES< wird das korrigiert.

9.5 LISTSCHUTZ

Einen einfachen Listschutz erhält man durch die Blockade des List Patches:

```
POKE &HFF89,&HDD
POKE &HFF8A,&HE1
```

Die Codes bedeuten POP IX, d.h. der Rücksprung zur Listroutine wird einfach vom Stapel geholt. Beim folgenden RET Befehl wird sofort in die Interpreterschleife zurückgesprungen.
Durch >POKE &HFF89,&HC9< wird der Listschutz wieder aufgehoben.

Ein weiterer Listschutz, der besonders in Verbindung mit dem in Kapitel 8 erwähnten wirksam ist, ist die Abfrage der Länge eines Programms. Wird versucht die Zeile, die ständig beim Listen das CLS bewirkt zu listen, so verändert sich die Länge des Programms und das kann festgestellt werden. Daraufhin erfolgt die Selbstzerstörung des Programms. Dazu ist folgende Zeile nötig:

```
1 A=FRE(0):IF A<>&H1111 THEN END
```

Der korrekte Wert für die Länge, der Anstelle von &H1111 gesetzt wird, kann durch einen Probelauf ermittelt werden:

RUN eingeben und PRINT HEX$(A)

Der erhaltene Wert wird dann für &H1111 eingesetzt.
Verändern Sie nun irgendetwas in den folgenden Zeilen des Programms, so wird das festgestellt und das Programm sofort beendet.

Noch wirkungsvoller wird der Schutz, wenn das END durch ein NEW ersetzt wird (oder sogar durch einen RESET Sprung).

9.6 DEUTSCHES BASIC

Nun folgen noch ein paar Tips, die nur dann eine Funktion haben, wenn das ROM ins RAM kopiert wurde.

```
POKE &HD4A,1
```

Diese Zeile bewirkt dann, daß die Wiederholungsfunktion der Tasten ohne Verzögerung eintritt. Auch andere Werte als 1 sind möglich. 13 ist der Standardwert.

Im RAM können wir sogar alle BASIC Befehlsworte und Fehlermeldungen in deutscher oder einer anderen Sprache realisieren.
Die Tabelle der Befehlsworte steht ab Adresse &H3AF"

Durch:

```
POKE &H3AB6,ASC("A")
POKE &H3AB7,ASC("D")
POKE &H3AB8,&HC5 ASC("E")+128
```

wird CLOAD zu CLADE.
Um ein Programm zu laden muß nun statt CLOAD CLADE eingegeben werden. CLOAD erzeugt einen Syntax Error. Auch durch List erhalten Sie jetzt immer CLADE anstelle von CLOAD.
Zum Code des letzten Buchstaben eines Keywords muß immer 128 addiert werden. Die Keyword Tabelle ist alphabetisch geordnet, wobei der jeweilige erste Buchstabe nicht angegeben ist. Anstelle des ersten Buchstaben steht das Token. Ein 0 Byte bedeutet den Beginn des nächsten Anfangsbuchstaben. Schauen Sie sich die Struktur der Tabelle mit dem Monitor (Start &H3A72) an.

Die Tabelle der Fehlermeldungen liegt ab Adresse &H3D75. Hier sind die Fehlermeldungen durch ihre ASCII codes abgespeichert. Die einzelnen Meldungen sind durch Nullbytes abgetrennt.
Geben Sie folgendes ein:

```
POKE &H3D8D,ASC("f")
POKE &H3D8E,ASC("e")
       .        h
       .        l
       .        e
       .        r
```

Nun erhalten Sie anstelle von "Syntax Error" einen "Syntaxfehler" ausgegeben.

9.7 INPUT OHNE ?

Wenn der ROM ins RAM kopiert ist, war dieses Problem gelöst. Es gibt auch die Möglichkeit bei eingeschaltetem ROM das Fragezeichen mit einem Patch zu überschreiben.

```
10 FOR I=&HFDE0 TO &HFDE3
20 READ A
30 POKE I,A
40 NEXT
50 DATA &H21,&HD2,&H23,&HE3
```

Assemblerlisting:

```
FDE0 210223      LD   HL,&H23D2 ; Input Patch &H23D2
ist neue Rücksprungadresse
FDE3 E3          EX   (SP),HL ; auf Stapel
FDE4 C9          RET
```

Der alte Zustand kann durch

```
POKE &HFDE0,&HC9
```

wieder hergestellt werden.

KAPITEL X : PROGRAMME

10.1 DER MENUEGENERATOR

Der Menuegenerator gibt Ihnen die Möglichkeit Programme, bei denen viel mit Menues gearbeitet wird, einfacher zu gestalten.
Der eigentliche Menuegenerator ist in den Zeilen 50 bis 480 und Zeilen 10000 bis zum Ende enthalten.
Die DATA Zeilen am Ende des Programms können Sie nach Ihrem Bedarf ändern. Das Ausrufezeichen (!) kennzeichnet die letzte DATA Zeile.
Die erste DATA Zeile (10270) muß immer Hauptmenue sein, gefolgt von Zeile 10280 DATA 0. Ab dieser Zeile können Ihre eigenen Menuepunkte eingegeben werden. Die Reihenfolge ist folgendermaßen:

Nach DATA 1 steht das Untermenue zum ersten Punkt des Hauptmenues (hier:drucken).
Nach DATA 2 steht das Untermenue des 2ten Unterpunktes usw..
Ist dann jedem Punkt vom Hauptmenue ein Untermenue zugeordnet, so folgt das Untermenue (2te Ebene) des ersten Punktes der ersten Ebene (hier: zu Format gehört DATA 6 und die folgenden) usw..

Das Besondere ist, daß Sie gar nicht auf die Längen der einzelnen Menues achten müssen. Die Eingabe in in der oben beschriebenen Weise reicht vollkommen aus.

Wenn Sie Ihr eigens Menue eingegeben haben, müssen Sie natürlich noch die jeweiligen Routinen dazu schreiben. Dazu dienen die Zeilen 1000-9999.
Die Integration des Menuegenerators ist schematisch immer gleich (siehe Zeile 1010 bis 1040). An den Zielzeilen in >ON GOSUB< Befehl steht dann entweder der Aufruf des nächsten Menues (genau nach dem gleichen Schema), oder eine ausführende Routine. Diese sollte dann mit >RETURN< abgeschlossen sein.

Das Programm beruht auf der Verwaltung der Menuepunkte durch ein Zeigerfeld MD(MP,I). MD(,) enthält Informationen über:

0 - RUCK: Nummer des letzten Menues
1 - NEXT: Nummer des nächsten Menues
2 - GRÖSSE: Anzahl der Eintragungen im Menue
3 - DEFAULT: zuletzt in diesem Menue gewählter Wert

Um die jeweiligen Points abzurufen, wird der Menuepointer MP und die Kennzahl der Information (0-3) für MD als Zeiger benutzt. Die Menuestring und die Zeigerfelder werden sämtlich durch Zeilen 10200 bis 10260 erzeugt.

```
10 REM Menuegenerator
20 CLEAR 200,&HEFFF:MAXFILES=0
30 KEY OFF:SCREEN 0:WIDTH 38
40 DEFINT A-Z
50 GOSUB 10000 : REM Initialisierung
60 GOSUB 1000 : REM Hauptmenue
70 CLS: END
170 '
180 REM Unterprogramme fuer Menue
190 '
200 REM Rahmen
210 CLS:PRINT"Menuegenerator              ";M$(MP)
220 PRINTSTRING$(38,"-")
230 LOCATE 0,21:PRINTSTRING$(38,"-");
240 PRINT"   >RETURN< fuer Auswahl des inversen"
250 NU=MD(MP,RU):IF NU>1 THEN NU=1
260 PRINT"oder >BS< fuer ";EX$(NU+1);
270 RETURN
290 '
300 REM Menue Aufbauen
310 GOSUB 200: REM Rahmen
320 WA=MD(MP,DL):REM Auswahl ist Defaulot
330 LOCATE 0,4
340 FOR I=1 TO MD(MP,GR):IF I=WA THEN POKE &HFDA4,&HC3:R
EM gewaehlten Punkt invers darstellen
350 PRINTI;:POKE IN,AU:PRINT" - ";:IF I=WA THEN POKE IN,
AN
360 PRINT M$(I+MD(MP,NX)-1):POKE IN,AU:REM Menuepunkt au
sgeben
370 NEXT
380 PRINT:PRINT:PRINT"Ihre Wahl :";:POKE IN,AN
390 PRINTWA;:POKE IN,AU
400 LOCATE POS(0)-3
410 EI$=INKEY$:IF EI$="" THEN 410
420 IF EI$=ZR$ THEN RETURN
430 IF EI$=CHR$(13) THEN MD(MP,DL)=WA:MP=MD(MP,NX)+WA-1:
RETURN
440 IF EI$=CHR$(28) OR EI$=CHR$(31) OR EI$=CHRPRINT$(32)
```

```
 THEN WA=WA+1+(WA=MD(MP,GR))*MD(MP,GR):GOTO 480
450 IF EI$=CHR$(29) OR EI$=CHR$(30) THEN WA=WA-1-(WA=1)*
MD(MP,GR):GOTO 480
460 N=VAL(EI$):IF (N<1) OR (N>MD(MP,GR)) THEN 410
470 WA=N
480 POKE IN,AN:PRINTWA;:POKE IN,AU:GOTO330
990 '
1000 REM Hauptmenue
1010 GOSUB 300:REM Menue anzeigen
1020 IF EI$=ZR$ THEN RETURN
1030 ON WA GOSUB 1050,1310,1400,1490,6000
1040 GOTO 1010
1050 REM 1 Untermenue 1. Ebene
1060 GOSUB 300: REM Menue Aufbauen
1070 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1080 ON WA GOSUB 1100,1170,1240
1090 GOTO 1050
1100 REM Untermenue 2. Ebene
1110 GOSUB 300
1120 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1130 ON WA GOSUB 1140,2160,1150:RETURN:REM Ende dieses M
enuezweiges
1140 REM
1150 REM
1160 RETURN
1170 REM Untermenue 2. Ebene
1180 GOSUB 300
1190 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1200 ON WA GOSUB 1210,2260,1220:RETURN:REM Ende dieses M
enuezweiges
1210 REM
1220 REM
1230 RETURN
1240 REM Untermenue 2. Ebene
1250 GOSUB 300
1260 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1270 ON WA GOSUB 1280,2360,1290:RETURN:REM Ende dieses M
enuezweiges
```

```
1280 REM
1290 REM
1300 RETURN
1310 REM 2 Untermenue 1. Ebene
1320 GOSUB 300: REM Menue Aufbauen
1330 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1340 ON WA GOSUB 1360,3200,3300
1350 GOTO 1310
1360 REM Untermenue 2. Ebene
1370 GOSUB 300
1380 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1390 ON WA GOSUB 3150,3160,3170:RETURN:REM Ende dieses M
enuezweiges
1400 REM 2 Untermenue 1. Ebene
1410 GOSUB 300: REM Menue Aufbauen
1420 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1430 ON WA GOSUB 1450,4200,4300
1440 GOTO 1400
1450 REM Untermenue 2. Ebene
1460 GOSUB 300
1470 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1480 ON WA GOSUB 4150,4160,4170:RETURN:REM Ende dieses M
enuezweiges
1490 ' usw.
1500 ' alle Programmzeilen von 1000 -
1510 ' 10000 sind nur Beispiel
1520 ' Innerhalb dieser Zeilen muss
1530 ' dann das Eigentliche Programm
1540 ' erstell werden
1550 ' Das jetzt vorhandene Progr.
1560 ' stellt nur das Geruest fuer
1570 ' beliebige Programme, die um-
1580 ' fangreiche Menues benutzen, dar
10000 REM Init
10010 REM Inversen Zeichensatz laden
10020 BA=BASE(2)+128*8
10030 FOR I=BA TO BA+128*8-1
10040 VPOKE I,VPEEK(I-128*8) XOR 255:NEXT
```

```
10050 REM Invers Patch laden
10060 FOR I=&HF370 TO &HF37A:READ A$
10070 POKE I,VAL("&H"+A$):NEXT
10080 DATA E1,F1,FE,20,38,02,F6,80,F5,E5,C9
10090 POKE &HFDA5,&H70:POKE &HFDA6,&HF3
10100 IN=&HFDA4:AN=&HC3:AU=&HC9:REM Variablen fuer Inver
se an/aus
10110 ZR$=CHR$(8):REM Zurueck Tasten Code
10120 FOR I=0 TO 2:READ EX$(I):NEXT
10130 DATA Ende Programm
10140 DATA Hauptmenue
10150 DATA Letztes Menue
10160 DIM M$(100),MD(100,3)
10170 RU=0:NX=1:REM md(i,ru) zeigt auf rueckwaertiges Me
nue, md(i,nx) auf das naechste
10180 GR=2:DL=3:REM md(i,gr) Groesse Menue, md(i,dl) zul
etzt gewaehlter punkt (Default
10190 N=-1:I=-1:REM Zaehler
10200 I=I+1:READ M$(I):REM Menuepunkt lesen
10210 MD(I,RU)=N:MD(I,DL)=1:REM letztes menue und Defaul
t
10220 IF M$(I)="!" THEN I=I-1:MD(MD(I,RU),GR)=I-MD(MD(I,
RU),NX)+1:RETURN:REM Ende aller Menues feststellen
10230 IF LEN(M$(I))>2 THEN 10200:REM wenn noch nicht neu
es Menue (m$=zahl)
10240 N=VAL(M$(I)):MD(N,NX)=I:I=I-1:REM werte fuer naech
stes Menue setzen
10250 IF MD(I,RUCK)>=0 THEN MD(MD(I,RUCK),GR)=I-MD(MD(I,
RU),NX)+1:REM Groesse des letzten Menues
10260 GOTO 10200
10270 DATA "Hauptmenue
10280 DATA 0
10290 DATA drucken
10300 DATA zeigen
10310 DATA editieren
10320 DATA Suchen
10330 DATA Dienst
10340 DATA 1
```

```
10350 DATA Format
10360 DATA Seiten
10370 DATA Normal
10380 DATA 2
10390 DATA formatiert
10400 DATA 40 Zeichen
10410 DATA 32 Zeichen
10420 DATA 3
10430 DATA Block
10440 DATA loeschen
10450 DATA 4
10460 DATA Suchwort
10470 DATA Seitenzahl
10480 DATA 5
10490 DATA Floppy
10500 DATA Casette
10510 DATA 6
10520 DATA Standartformat
10530 DATA Spezielle Eingabe
10540 DATA Druck ohne Formular
10550 DATA !
```

10.2 MINITEXTOMAT

Eine der wichtigsten Anwendungen des Computers ist die Textverarbeitung. Um Ihnen einen Eindruck der Fähigkeiten der MSX Rechner auf dem Gebiet der Personalcomputer zu geben, haben wir ein Textprogramm für MSX geschrieben.

Die Bedienung des Programms ist zunächst grundsätzlich gleich dem Arbeiten auf dem BASIC Bildschirm (beim Programmieren). Das heißt, >INS<, >DEL< und >BS< wirken beim BASIC immer nur auf eine Zeile, im Textprogramm aber auf einen Absatz.
Ein Absatz wird durch das gleichzeitige Drücken von >SHIFT< und >RETURN< bewirkt. Ein kleiner Pfeil zeigt das Absatzende an. Sollen Leerzeichen im Text auftauchen, so müssen Sie ebenfalls durch >SHIFT< und >RETURN< erzeugt werden. Die gleiche Tastendruckkombination löscht immer den Rest einer Zeile.

Die Cursortasten und >RETURN< (ohne >SHIFT<) dienen dazu sich im Text frei zu bewegen. Mit >CTRL<+Cursor springen Sie Bildschirmweise durch den Text.

Am Ende des Textes sollte immer >GRAPH"+P< eingegeben werden (muß allein auf einer Zeile stehen). Dieses Zeichen, das das gleiche Aussehen wie der Cursor hat, beendet beim Drucken die Textausgabe.

Mit >ESC< verlassen Sie das Programm. Wenn Sie beim Wiederanfang des Programms bei der Löschfunktion "nein" eingeben, ist der alte Text noch erhalten.
Weiterhin können über die Funktionstasten Routinen Druck/ Save/ Load aufgerufen werden.

Mit Save/Load können Texte geladen/gespeichert werden.
Beim Drucken wird das Formular, das den Seitenaufbau bestimmt durch LR,AS,OB,AZ gegeben (Zeile 2510,2520). Ggf. müssen diese Werte geändert werden.

Im Anschluß an das BASIC Listing finden Sie die Assemblerlistings der im Textprogramm benutzten Maschinenroutinen.

```
10 ' Minitextomat fuer MSX von H. Dullin
20 CLEAR 200,&HE6FF:REM Speicherbereich fuer Texte reser
vieren
30 GOSUB 2000:REM Initialisierung
40 INPUT "Speicher loeschen ?";A$
50 IF (ASC(A$)OR2^5)=ASC("j") THEN X=USR7(1):CLS ELSE X=
USR1(TB)
60 A$=""
70 LOCATE 0,0,1
80 ' Haupteingabeschleife
90 A$=INKEY$:IF A$="" THEN 90
100 GOSUB 120:REM Eingabe Auswerten
110 GOTO 80 :REM zur neuen Eingabe
120 REM Auswertung Eingabe
130 AF=0:REM Scrollflag
140 IF A$>=" " THEN 290:REM Standard Zeicheneingabe
150 A=ASC(A$):REM Steuercode
160 IF A>26 THEN ON A-26 GOTO 230,390,430,510,580:REM Es
cape und Cursortasten
170 IF A=13 THEN 650:REM Return
180 IF A=11 THEN LOCATE 0,0:RETURN:REM Home
190 IF A=12 THEN X=USR1(TB):ZU=0:LOCATE 0,0,1:RETURN:REM
 Shift Home
200 IF A=8 THEN A$=CHR$(29):GOSUB 430:X=USR3(TB+(ZU+CSRL
IN)*SM+POS(0)+1):X=USR1(TB+ZU*SM):RETURN:REM Back Space
210 IF A=18 THEN POKE &HFCAA,1+(PEEK(&HFCAA)=1):LOCATE ,
,1:RETURN:REM Insert Taste
220 RETURN: REM wieder zur Eingabe
230 REM Ende Programm
240 CLS:LOCATE 0,0,0
250 ON ERROR GOTO 0:REM On Error ausschalten
260 POKE &HFCAA,0:REM Insert Flag zuruecksetzen
280 END
290 REM Standardbuchstabeneingabe
300 IF ASC(A$)=127 THEN X=USR3(TB+(ZU+CSRLIN)*SM+POS(0)+
1):X=USR1(TB+ZU*SM):RETURN: REM Delete
310 IF POS(0)=SM-1 AND CSRLIN=23-ZO THEN AF=-1:REM Fests
tellen ob Scrolling erforderlich
```

```
320 IF PEEK(&HFCAA)=1 THEN GOSUB 750: REM Wenn Insert an
, dann 1600
330 POKE TB+(ZU+CSRLIN)*SM+POS(0),ASC(A$): REM Buchstabe
ncode Speichern
340 PRINTA$;:REM Buchstaben ausgeben
350 IF POS(0)<>0 THEN RETURN:REM Fertig wenn kein Scroli
ng
360 IF ZU+CSRLIN+1>=ZM THEN BEEP:AF=-1:ZU=ZU-1:REM Test
Speicher voll
370 IF AF THEN ZU=ZU+1:X=USR1(TB+ZU*SM): REM Scrolling
380 RETURN
390 REM Cursor rechts
400 IF POS(0)=SM-1 AND CSRLIN=23-ZO THEN AF=-1:A$=CHR$(1
3)+CHR$(10):REM wenn scrolling
410 PRINTA$;
420 GOTO 350
430 REM Cursor links
440 IF POS(0)=0 AND CSRLIN=0 THEN LOCATE SM-1:GOTO 460:R
EM  wenn rueckwaerts Scrolling
450 PRINTA$;:RETURN
460 REM Scroling rueckwaerts
470 IF ZU=0 THEN LOCATE 0:BEEP:RETURN
480 ZU=ZU-1:REM Uebertragzeilenzaehler erniedrigen
490 X=USR1(TB+ZU*SM):REM Bild anzeigen
500 RETURN
510 REM Cursor hoch
520 IF (PEEK(&HFBEB)AND2)=0 THEN 550:REM Ist CTRL gedrue
ckt ?
530 IF CSRLIN=0 THEN 460:REM rueckwaerts Scrolling
540 PRINTA$;:RETURN
550 REM ctrl curs. Hoch
560 ZU=ZU-24+ZO:IF ZU<0 THEN ZU=0:REM wenn Textanfang
570 X=USR1(TB+ZU*SM):RETURN
580 REM Cursor runter
590 IF (PEEK(&HFBEB)AND2)=0 THEN 620:REM CTRL ?
600 IF CSRLIN=23-ZO THEN AF=-1:PRINTCHR$(10);:GOTO 360:R
EM Scroling
610 PRINTA$;:RETURN
```

```
620 REM CTRL CUrs. Runter
630 J=ZU+24-ZO:IF J<ZM-24+ZO THEN ZU=J ELSE ZU=ZM-24+ZO:
REM Textende ?
640 X=USR1(TB+ZU*SM):RETURN
650 REM Return => Cursor runter
660 IF (PEEK(&HFBEB)AND1)=0 THEN 690:REM SHIFT gedr%Ylm:
 ?
670 IF CSRLIN=23-ZO THEN AF=-1:PRINTA$;CHR$(10);:GOTO 36
0:REM Scrolling ?
680 PRINTA$;CHR$(10);:RETURN
690 REM Shift RETURN
700 I=TB+(ZU+CSRLIN)*SM:REM Stardadresse der Aktuellen Z
eile im Speicher
710 FOR J=I+POS(0) TO I+SM-1:POKE J,32:NEXT:REM Rest der
 Zile loeschen
720 POKE I+POS(0),208:REM Shift Return Zeichen (Pfeil) S
peichern
730 GOSUB 670:X=USR1(TB+ZU*SM):REM Return ausfuehren und
 text anzeigen
740 RETURN
750 REM INSERT
760 X=USR2(TB+(ZU+CSRLIN)*SM+POS(0)-1):REM Im Speicher P
latz freimachen
770 X=USR1(TB+ZU*SM):REM verschobenen Text anzeigen
780 RETURN:REM zur normalen Zeichenausgabe
1000 REM Druckausgabe
1010 CLS
1020 Y$=STRING$(08,10):REM String fuer Zeilenvorschub Ze
itenanfang
1030 AA=TB:REM Speicheradresse
1040 ZZ=0:REM Zeilenzaehler
1050 X$=STRING$(LR,32):REM String fuer Linekn Rand erzeu
gen
1060 REM Seitenanfang
1070 LPRINT Y$;:REM Zeilenvorschub
1080 REM Zeilenausgabe
1090 Z$="":REM aktuelle Zeile
1100 VP=VARPTR(Z$)
```

```
1110 POKE VP,AS:REM Laenge der Zeile
1120 POKE VP+1,AA-INT(AA/256)*256:REM Adresse der Zeile
im Textspeicher
1130 POKE VP+2,INT(AA/256)-256*(AA<0)
1140 AA=AA+AS:REM Adresse auf naechste Zeile
1150 PO=INSTR(Z$,E$):REM GRAPH P ?, bedeutet Textende
1160 IF PO<>0 OR AA>&HF1FF THEN LPRINT STRING$(PZ-OB-AZ,
10):POKE &HF3DE,257-ZO:X=USR1(TB):RETURN 80:REM wenn Tex
tende
1170 PO=INSTR(Z$,CR$):REM SHIFT RETURN in der Zeile ?
1180 IF PO=0 THEN 1240: REM Nein
1190 REM Shift Return
1200 POKE VP,PO-1:REM Druckzeile bis SHIFT RETURN
1210 AA=AA-AS+PO:REM Adresse auf Zeichen nach Shift Retu
rn
1220 IF PEEK(AA)<>32 THEN 1310 ELSE AA=AA+1:GOTO 1220:RE
M folgende Leerzeichen ueberlesen
1230 GOTO 1310
1240 REM Leerzeichen vor naechstem Wort suchen
1250 IF PEEK(AA)=32 THEN AA=AA+1:GOTO 1310
1260 I=0:REM Zaehler bis Leerzeichen
1270 AA=AA-1:I=I+1:REM rueckwaerts in aktueller Zeile su
chen
1280 IF PEEK(AA)<>32 THEN 1270:REM kein Leerzeichn, dann
 weitersuchen
1290 POKE VP,PEEK(VP)-I:REM Zeile bis zum letzen "draufp
assenden" Wort drucken
1300 AA=AA+1:REM Adresse auf Anfang naechstes Wort
1310 LPRINTX$;Z$:REM linken Rand und Zeile ausgeben
1320 ZZ=ZZ+1:REM Zeilenzaehler erhoehen
1330 IF ZZ=AZ THEN LPRINT STRING$(PZ-OB-AZ,10):GOTO 1060
:REM Seitenende
1340 GOTO 1080:REM naechste Zeile
1350 REM Speichern
1360 X=USR4(1):REM Keys loeschen
1370 POKE &HF3DE,0:POKE &HF3B1,25
1380 LOCATE 0,22,1
1390 INPUT"Name :";N$
```

```
1400 BSAVE "CAS:"+N$,TB,&HF1FF
1410 LOCATE 0,0,1
1420 POKE &HF3DE,257-ZO:POKE &HF3B1,23
1430 X=USR5(1):REM Keys anzeigen
1440 RETURN 80
1450 REM laden
1460 X=USR4(1):REM Keys loeschen
1470 POKE &HF3DE,0:POKE &HF3B1,25:TEXTWIDOW VERGROESSERN
1480 LOCATE 0,22,1
1490 INPUT"Name :";N$
1500 BLOAD "CAS:"+N$
1510 LOCATE 0,0,1
1520 POKE &HF3DE,257-ZO:POKE &HF3B1,23
1530 X=USR5(1):REM Keys anzeigen
1540 RETURN 80
2000 REM Initialisierung
2010 SCREEN 0
2020 COLOR 14,4,4
2030 DEFINT A-Z
2040 TB=&HE700:REM Textbasisadres
2050 SM=36:WIDTH SM:REM Zeilenbreite
2060 ZM=INT((&HF1FF-TB)/SM):REM maxiamle Zeilenzahl
2070 ZO=3:REM Zeilenoffset unterer Rand
2080 POKE &HF3B1,23:REM letzte Bildschirmzeile (Key ausg
abe)
2090 ON STOP GOSUB 230
2100 STOP ON
2110 ON ERROR GOTO 230
2120 REM Mapro fuer Textanzeige laden
2130 FOR I=&HF300 TO &HF326:READ A$:POKE I,VAL("&H"+A$):
NEXT
2140 DATA 23,23,5E,23,56,3A,B0,F3
2150 DATA 4F,06,00,CD,32,0C,21,01,01,F5
2160 DATA E5,D5,CD,F2,0B,D1,C5,CD
2170 DATA 45,07,C1,E1,23,F1,3D,20
2180 DATA EE,CD,ED,09,C9
2190 DEFUSR1=&HF300
2200 REM Mapro fuer Insert laden
```

```
2210 FOR I=&HF330 TO &HF371:READ A$:POKE I,VAL("&H"+A$):
NEXT
2220 DATA 23,23,7E,23,66,6F,E5,11
2230 DATA FF,F1,EB,B7,ED,52,E3,C1,E5
2240 DATA 3E,D0,ED,B1,D1,E5,B7,ED
2250 DATA 52,E3,7E,FE,20,28,1A,E5
2260 DATA 21,FF,F1,3A,B0,F3,5F,16
2270 DATA 00,E5,19,D1,EB,03,ED,B8,47,23,3E,20
2280 DATA 77,23,10,FC,E1,C1,54,5D
2290 DATA 1B,EB,ED,B8,C9
2300 DEFUSR2=&HF330
2310 REM Mapro fuer Delete/Backspace laden
2320 FOR I=&HF2B0 TO &HF2F0:READ A$:POKE I,VAL("&H"+A$):
NEXT
2330 DATA 23,23,7E,23,66,6F,E5,11
2340 DATA FF,F1,EB,B7,ED,52,E3,C1,E5
2350 DATA 3E,D0,ED,B1,D1,B7,ED,52
2360 DATA 44,4D,62,6B,1B,ED,B0,2B
2370 DATA 36,20,E5,3A,B0,F3,47,7E
2380 DATA 23,FE,20,20,11,10,F8,EB,21
2390 DATA FF,F1,B7,ED,52,44,4D,E1
2400 DATA 00,EB,ED,B0,C9,E1,C9
2410 DEFUSR3=&HF2B0
2420 REM Mapro Speicher loeschen laden
2430 FOR I=&HF280 TO &HF28D:READ A$:POKE I,VAL("&H"+A$):
NEXT
2440 DATA 21,00,E7,36,20,11,01,E7
2450 DATA 01,FF,1A,ED,B0,C9
2460 DEFUSR7=&HF280
2470 ON KEY GOSUB 1000,1350,1450
2480 KEY 1,"Druck":KEY (1) ON
2490 KEY 2,"Save":KEY (2) ON
2500 KEY 3,"load":KEY (3) ON
2510 LR=10:AS=60:REM Linker Rand,Anzahl Spalten fuer Dru
ck
2520 OB=5:AZ=60:PZ=72:REM oberer Anfang Formula, Anzahl
Druckzeilen, Anzahl Zeilen pro Zeite fuer Druck
2530 KEY ON:CLS:LOCATE 0,0,1:POKE &HF3DE,256-ZO+1:REM Of
```

```
fset Bildende zur Keyzeile
2540 E$=CHR$(219):CR$=CHR$(208):REM Ende Zeichen (Graph
P) und Shift Return Zeichen
2550 DEFUSR4=&HB15:REM Keys loeschen
2560 DEFUSR5=&HB2B:REM Keys anzeigen
2570 RETURN
```

```
 F000              10                    ; Insert Routine
 F000              20  ENDE    EQU  &HF1FF
 F000              30                    ; Uebergabe : RAM Adre
sse
 F000 23           40          INC  HL
 F001 23           50          INC  HL
 F002 7E           60          LD   A,(HL)
 F003 23           70          INC  HL
 F004 66           80          LD   H,(HL)
 F005 6F           90          LD   L,A
 F006 E5          100          PUSH HL ; aktuelle RAM Adres
se
 F007 11FFF1      110          LD   DE,ENDE ; Textende
 F00A EB          120          EX   DE,HL
 F00B B7          130          OR   A ; Carry loeschen
 F00C ED52        140          SBC  HL,DE ; Anzahl bis Text
ende
 F00E E3          150          EX   (SP),HL
 F00F C1          160          POP  BC ; Anzahl
 F010 E5          170          PUSH HL ; RAM Adresse
 F011 3ED0        180          LD   A,208 ; Shift Return
 F013 EDB1        190          CPIR  ; suchen
 F015 D1          200          POP  DE ; RAM Adresse
 F016 E5          210          PUSH HL ; Adresse von Shift
Return +1 = Endzieladresse
 F017 B7          220          OR   A ; Carry = 0
 F018 ED52        230          SBC  HL,DE ; Anzahl zu versc
hiebender Bytes
 F01A E3          240          EX   (SP),HL
 F01B 7E          250          LD   A,(HL)
 F01C FE20        260          CP   32 ; Space ??
?F01E 28FE        270          JR   Z,OK ; ja, dann ist noc
h Platz
 F020 E5          280          PUSH HL ; Endzieladresse
 F021 21FFF1      290          LD   HL,ENDE
 F024 3AB0F3      300          LD   A,(&HF3B0) ; Width
 F027 5F          310          LD   E,A
 F028 1600        320          LD   D,0
 F02A E5          330          PUSH HL ; Textende
 F02B 19          340          ADD  HL,DE ; plus Width
 F02C D1          350          POP  DE
 F02D EB          360          EX   DE,HL
 F02E 03          370          INC  BC
 F02F EDB8        380          LDDR  ; gesamten Resttext ve
rschieben
 F031 47          390          LD   B,A
 F032 3E20        400          LD   A,&H20
```

```
 F034 77         410  NEXT   LD   (HL),A ; freigewordene
Zeile loeschen
 F035 23         420         INC  HL
 F036 10FC       430         DJNZ NEXT
 F038 E1         440         POP  HL
**** Zeile  270  : OK=&HF039 Offset 19
 F039 C1         450  OK     POP  BC
 F03A 54         460         LD   D,H
 F03B 5D         470         LD   E,L
 F03C 1B         480         DEC  DE
 F03D EB         490         EX   DE,HL
 F03E EDB8       500         LDDR  ; Zeile bis Shift Retu
rn um 1 verschieben
 F040 C9         510         RET
------------------------------------

Programm :insert
Start : &HF000     Ende : &HF040
Lnge : &H41 Bytes
Fehler : 0
Variablentabelle :
ENDE    F1FF    NEXT    F034
OK      F039
```

```
 F000              10                ; Delete Routine
 F000              20                ; UEBERGABE : RAM ADRE
SSE
 F000              30  ENDE    EQU   &HF1FF
 F000 23           40          INC   HL
 F001 23           50          INC   HL
 F002 7E           60          LD    A,(HL)
 F003 23           70          INC   HL
 F004 66           80          LD    H,(HL)
 F005 6F           90          LD    L,A
 F006 E5          100          PUSH  HL ; RAM Adresse
 F007 11FFF1      110          LD    DE,ENDE
 F00A EB          120          EX    DE,HL
 F00B B7          130          OR    A
 F00C ED52        140          SBC   HL,DE ; Anzahl bis Text
ende
 F00E E3          150          EX    (SP),HL
 F00F C1          160          POP   BC ; Anzahl
 F010 E5          170          PUSH  HL ; RAM Adresse
 F011 3ED0        180          LD    A,208 ; Shift Return
 F013 EDB1        190          CPIR  ; Suchen
 F015 D1          200          POP   DE
 F016 B7          210          OR    A
 F017 ED52        220          SBC   HL,DE ; Anzahl zu versc
hiebender Bytes
 F019 44          230          LD    B,H
 F01A 4D          240          LD    C,L
 F01B 62          250          LD    H,D
 F01C 6B          260          LD    L,E
 F01D 1B          270          DEC   DE ; -1 = Zieladresse
 F01E EDB0        280          LDIR
 F020 2B          290          DEC   HL ; -1 = alte Adresse
von Shift Return
 F021 3620        300          LD    (HL),&H20 ; loeschen
 F023 E5          310          PUSH  HL
 F024 3AB0F3      320          LD    A,(&HF3B0) ; Width
 F027 47          330          LD    B,A
 F028 7E          340  PRUEF   LD    A,(HL)
 F029 23          350          INC   HL
 F02A FE20        360          CP    32 ; Leerzeichen ?
?F02C 20FE        370          JR    NZ,ZRUCK ; nein, dann E
nde
 F02E 10F8        380          DJNZ  PRUEF ; ganze Zeile pru
efen
 F030 EB          390          EX    DE,HL ; wenn Zeile nur
mit Leerzeichen, dann loeschen
 F031 21FFF1      400          LD    HL,ENDE
```

```
 F034 B7         410          OR   A
 F035 ED52       420          SBC  HL,DE
 F037 44         430          LD   B,H
 F038 4D         440          LD   C,L ; Anzahl
 F039 E1         450          POP  HL ; Adresse altes Shif
t Return
 F03A 2B         460          DEC  HL
 F03B EB         470          EX   DE,HL
 F03C EDB0       480          LDIR  ; evtl. freigewordene
Zeile loeschen
 F03E C9         490          RET
**** Zeile  370  : ZRUCK=&HF03F Offset 11
 F03F E1         500  ZRUCK   POP  HL
 F040 C9         510          RET
-------------------------------------

Programm :delete
Start : &HF000     Ende : &HF040
Lnge : &H41 Bytes
Fehler : 0
Variablentabelle :
ENDE    F1FF    PRUEF   F028
ZRUCK   F03F
```

```
 F000              10                ; Textanzeige Routine
 F000              20                ; UEBERGABE : RAM Adre
sse des ersten anzuzeigende Buchstaben
 F000              30  ENDE    EQU   &HF1FF
 F000 23           40          INC   HL
 F001 23           50          INC   HL
 F002 5E           60          LD    E,(HL)
 F003 23           70          INC   HL
 F004 56           80          LD    D,(HL)
 F005 3AB0F3       85          LD    A,(&HF3B0) ; Width
 F008 4F           90          LD    C,A
 F009 0600         100         LD    B,0 ; BC ist Spaltenzae
hler
 F00B CD320C       110         CALL &H0C32 ; holt Anzahl de
r Zeilen pro Bildschirm in Akku
 F00E 210101       120         LD    HL,&H0101 ; Spalte/Zeil
e
 F011 F5           130 NEXT    PUSH AF
 F012 E5           140         PUSH HL
 F013 D5           150         PUSH DE
 F014 CDF20B       160         CALL &H0BF2 ; VRAM Adresse a
us Spalte/Zeile (HL) berechnen
 F017 D1           170         POP   DE ; Adresse RAM
 F018 C5           180         PUSH BC ; Zeichen pro Zeile
 F019 CD4507       190         CALL &H0745 ; Blockladerouti
ne RAM -> VRAM
 F01C C1           200         POP   BC ; Anzahl pro Zeile
 F01D E1           210         POP   HL ; Cursorposition
 F01E 23           220         INC   HL ; auf naechste Zeile
 F01F F1           230         POP   AF ; Zeilenzaehler
 F020 3D           240         DEC   A ; erniedrigen
 F021 20EE         250         JR    NZ,NEXT ; noch nicht Nu
ll, dann weiter
 F023 CDED09       260         CALL &H09ED ; Cursor wieder
Anzeigen
 F026 C9           270         RET   ; zurueck ins BASIC
------------------------------------

Programm :anzeig
Start : &HF000     Ende : &HF026
Lnge : &H27 Bytes
Fehler : 0
Variablentabelle :
ENDE    F1FF    NEXT    F011
```

```
 F000             10                ; Textbereich loeschen
 F000 2100E7      20          LD    HL,&HE700 ; Anfang Text
bereich
 F003 3620        30          LD    (HL),32 ; loeschen
 F005 1101E7      40          LD    DE,&HE701 ; Zieladresse
 F008 01FF1A      50          LD    BC,&H1AFF ; Anzahl = &H
f200-&He701
 F00B EDB0        60          LDIR  ; Bereich loeschen
 F00D C9          70          RET
----------------------------------------

Programm :loesch
Start : &HF000     Ende : &HF00D
Lnge : &HE Bytes
Fehler : 0
```

UMRECHNUNGSTABELLE DEZIMAL - HEXADEZIMAL - BINÄR

dezimal	hex	binär	dezimal	hex	binär
0	&H00	&B00000000	26	&H1A	&B00011010
1	&H01	&B00000001	27	&H1B	&B00011011
2	&H02	&B00000010	28	&H1C	&B00011100
3	&H03	&B00000011	29	&H1D	&B00011101
4	&H04	&B00000100	30	&H1E	&B00011110
5	&H05	&B00000101	31	&H1F	&B00011111
6	&H06	&B00000110	32	&H20	&B00100000
7	&H07	&B00000111	33	&H21	&B00100001
8	&H08	&B00001000	34	&H22	&B00100010
9	&H09	&B00001001	35	&H23	&B00100011
10	&H0A	&B00001010	36	&H24	&B00100100
11	&H0B	&B00001011	37	&H25	&B00100101
12	&H0C	&B00001100	38	&H26	&B00100110
13	&H0D	&B00001101	39	&H27	&B00100111
14	&H0E	&B00001110	40	&H28	&B00101000
15	&H0F	&B00001111	41	&H29	&B00101001
16	&H10	&B00010000	42	&H2A	&B00101010
17	&H11	&B00010001	43	&H2B	&B00101011
18	&H12	&B00010010	44	&H2C	&B00101100
19	&H13	&B00010011	45	&H2D	&B00101101
20	&H14	&B00010100	46	&H2E	&B00101110
21	&H15	&B00010101	47	&H2F	&B00101111
22	&H16	&B00010110	48	&H30	&B00110000
23	&H17	&B00010111	49	&H31	&B00110001
24	&H18	&B00011000	50	&H32	&B00110010
25	&H19	&B00011001	51	&H33	&B00110011

UMRECHNUNGSTABELLE DEZIMAL - HEXADEZIMAL - BINÄR

dezimal	hex	binär	dezimal	hex	binär
52	&H34	&B00110100	78	&H4E	&B01001110
53	&H35	&B00110101	79	&H4F	&B01001111
54	&H36	&B00110110	80	&H50	&B01010000
55	&H37	&B00110111	81	&H51	&B01010001
56	&H38	&B00111000	82	&H52	&B01010010
57	&H39	&B00111001	83	&H53	&B01010011
58	&H3A	&B00111010	84	&H54	&B01010100
59	&H3B	&B00111011	85	&H55	&B01010101
60	&H3C	&B00111100	86	&H56	&B01010110
61	&H3D	&B00111101	87	&H57	&B01010111
62	&H3E	&B00111110	88	&H58	&B01011000
63	&H3F	&B00111111	89	&H59	&B01011001
64	&H40	&B01000000	90	&H5A	&B01011010
65	&H41	&B01000001	91	&H5B	&B01011011
66	&H42	&B01000010	92	&H5C	&B01011100
67	&H43	&B01000011	93	&H5D	&B01011101
68	&H44	&B01000100	94	&H5E	&B01011110
69	&H45	&B01000101	95	&H5F	&B01011111
70	&H46	&B01000110	96	&H60	&B01100000
71	&H47	&B01000111	97	&H61	&B01100001
72	&H48	&B01001000	98	&H62	&B01100010
73	&H49	&B01001001	99	&H63	&B01100011
74	&H4A	&B01001010	100	&H64	&B01100100
75	&H4B	&B01001011	101	&H65	&B01100101
76	&H4C	&B01001100	102	&H66	&B01100110
77	&H4D	&B01001101	103	&H67	&B01100111

UMRECHNUNGSTABELLE DEZIMAL - HEXADEZIMAL - BINÄR

dezimal	hex	binär	dezimal	hex	binär
104	&H68	&B01101000	130	&H82	&B10000010
105	&H69	&B01101001	131	&H83	&B10000011
106	&H6A	&B01101010	132	&H84	&B10000100
107	&H6B	&B01101011	133	&H85	&B10000101
108	&H6C	&B01101100	134	&H86	&B10000110
109	&H6D	&B01101101	135	&H87	&B10000111
110	&H6E	&B01101110	136	&H88	&B10001000
111	&H6F	&B01101111	137	&H89	&B10001001
112	&H70	&B01110000	138	&H8A	&B10001010
113	&H71	&B01110001	139	&H8B	&B10001011
114	&H72	&B01110010	140	&H8C	&B10001100
115	&H73	&B01110011	141	&H8D	&B10001101
116	&H74	&B01110100	142	&H8E	&B10001110
117	&H75	&B01110101	143	&H8F	&B10001111
118	&H76	&B01110110	144	&H90	&B10010000
119	&H77	&B01110111	145	&H91	&B10010001
120	&H78	&B01111000	146	&H92	&B10010010
121	&H79	&B01111001	147	&H93	&B10010011
122	&H7A	&B01111010	148	&H94	&B10010100
123	&H7B	&B01111011	149	&H95	&B10010101
124	&H7C	&B01111100	150	&H96	&B10010110
125	&H7D	&B01111101	151	&H97	&B10010111
126	&H7E	&B01111110	152	&H98	&B10011000
127	&H7F	&B01111111	153	&H99	&B10011001
128	&H80	&B10000000	154	&H9A	&B10011010
129	&H81	&B10000001	155	&H9B	&B10011011

UMRECHNUNGSTABELLE DEZIMAL - HEXADEZIMAL - BINÄR

dezimal	hex	binär	dezimal	hex	binär
156	&H9C	&B10011100	182	&HB6	&B10110110
157	&H9D	&B10011101	183	&HB7	&B10110111
158	&H9E	&B10011110	184	&HB8	&B10111000
159	&H9F	&B10011111	185	&HB9	&B10111001
160	&HA0	&B10100000	186	&HBA	&B10111010
161	&HA1	&B10100001	187	&HBB	&B10111011
162	&HA2	&B10100010	188	&HBC	&B10111100
163	&HA3	&B10100011	189	&HBD	&B10111101
164	&HA4	&B10100100	190	&HBE	&B10111110
165	&HA5	&B10100101	191	&HBF	&B10111111
166	&HA6	&B10100110	192	&HC0	&B11000000
167	&HA7	&B10100111	193	&HC1	&B11000001
168	&HA8	&B10101000	194	&HC2	&B11000010
169	&HA9	&B10101001	195	&HC3	&B11000011
170	&HAA	&B10101010	196	&HC4	&B11000100
171	&HAB	&B10101011	197	&HC5	&B11000101
172	&HAC	&B10101100	198	&HC6	&B11000110
173	&HAD	&B10101101	199	&HC7	&B11000111
174	&HAE	&B10101110	200	&HC8	&B11001000
175	&HAF	&B10101111	201	&HC9	&B11001001
176	&HB0	&B10110000	202	&HCA	&B11001010
177	&HB1	&B10110001	203	&HCB	&B11001011
178	&HB2	&B10110010	204	&HCC	&B11001100
179	&HB3	&B10110011	205	&HCD	&B11001101
180	&HB4	&B10110100	206	&HCE	&B11001110
181	&HB5	&B10110101	207	&HCF	&B11001111

UMRECHNUNGSTABELLE DEZIMAL - HEXADEZIMAL - BINÄR

dezimal	hex	binär	dezimal	hex	binär
208	&HD0	&B11010000	234	&HEA	&B11101010
209	&HD1	&B11010001	235	&HEB	&B11101011
210	&HD2	&B11010010	236	&HEC	&B11101100
211	&HD3	&B11010011	237	&HED	&B11101101
212	&HD4	&B11010100	238	&HEE	&B11101110
213	&HD5	&B11010101	239	&HEF	&B11101111
214	&HD6	&B11010110	240	&HF0	&B11110000
215	&HD7	&B11010111	241	&HF1	&B11110001
216	&HD8	&B11011000	242	&HF2	&B11110010
217	&HD9	&B11011001	243	&HF3	&B11110011
218	&HDA	&B11011010	244	&HF4	&B11110100
219	&HDB	&B11011011	245	&HF5	&B11110101
220	&HDC	&B11011100	246	&HF6	&B11110110
221	&HDD	&B11011101	247	&HF7	&B11110111
222	&HDE	&B11011110	248	&HF8	&B11111000
223	&HDF	&B11011111	249	&HF9	&B11111001
224	&HE0	&B11100000	250	&HFA	&B11111010
225	&HE1	&B11100001	251	&HFB	&B11111011
226	&HE2	&B11100010	252	&HFC	&B11111100
227	&HE3	&B11100011	253	&HFD	&B11111101
228	&HE4	&B11100100	254	&HFE	&B11111110
229	&HE5	&B11100101	255	&HFF	&B11111111
230	&HE6	&B11100110			
231	&HE7	&B11100111			
232	&HE8	&B11101000			
233	&HE9	&B11101001			

MSX-Computer haben zwei ganz elementare Vorzüge: zum einen ein hervorragendes Preis-/Leistungs-Verhältnis, zum anderen darüber hinaus außergewöhnliche Grafik- und Soundfähigkeiten. Das vorliegende Buch behandelt gerade diese Möglichkeiten der MSX-Rechner, umfassend und ausgezeichnet dargestellt. Viele nützliche Beispielprogramme, die den Text gelungen abrunden.

Lüers
MSX Grafik & Sound
ca. 250 Seiten, DM 39,–
ISBN 3-89011-051-7

Von den Grundlagen der Maschinensprache-programmierung über die Arbeitsweise des Z-80-Prozessors und einer genauen Beschreibung seiner Befehle bis zur Benutzung von Systemroutinen wird in diesem Buch jeder Bereich aufgegriffen und ausführlich – mit vielen Beispielen – erklärt. Ein fundierter Einstieg in die Maschinensprache, und dabei wirklich leichtgemacht! Wenn Sie mehr aus Ihrem MSX machen wollen, dann ist dieser Weg optimal.

Dullin/Straßenburg
MSX Maschinensprachebuch
ca. 300 Seiten, DM 39,–
ISBN 3-89011-109-2

DAS STEHT DRIN:

Dieses Buch enthält eine beispiellose Fülle an Tips & Tricks für alle Rechner mit dem neuen MSX - Standard. Es liefert nicht nur fertige Rezepte, sondern auch die nötigen Grundlagen dazu.

Aus dem Inhalt:

- Der MSX - Standard
- Grafikprogrammierung
- Zeichengenerator
- Windows
- Text- und Grafikhardcopy
- 3d-Grafik
- Spriteeditor
- Input/Output-Bausteine
- Soundprogrammierung
- Miniorgel und Synthesizer-Programmierung
- Maschinensprache
- Monitor
- Benutzung von Systemroutinen
- Variablendump
- Menuegenerator
- Minitextomat

UND GESCHRIEBEN HABEN DIESES BUCH:

Holger Dullin und Hardy Strassenburg sind Studenten der Informatik und Biologie, professionelle Programmierer und erfolgreiche Sachbuchautoren (z.B. Maschinensprachebuch zu MSX und CPC).

ISBN 3-89011-112-2