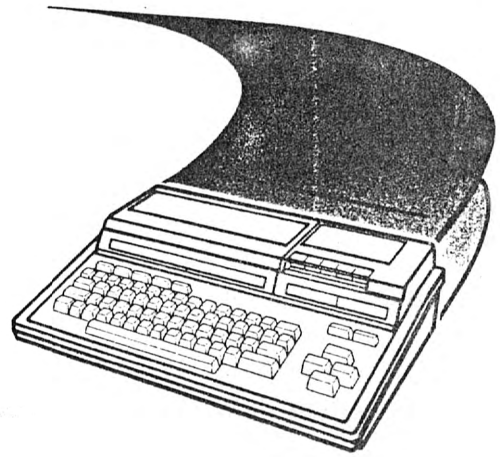
MZ-Verlag
Harald Schicke

Torsten Ziegler

Systemprogrammierung auf MZ-800

Informationswerk für den
Maschinensprachenanwender



ISBN 3-89240-000-8

Verlag: MZ-Verlag Harald Schicke
Postfach, D-2110 Buchholz 5
Tel.: 04187/6533

Autor: Torsten Ziegler

Satz: MZ-Verlag

© 1986 ISBN 3-89240-000-8

Druck: MZ-Verlag Harald Schicke

Inhalt

Einleitung

Dieses Buch, das Sie gerade in Händen halten, wurde in der Absicht geschrieben, Sie über diejenigen Eigenschaften Ihres MZ-800 zu informieren, die im Handbuch verschwiegen werden, um es Ihnen zu ermöglichen, selbst in Maschinensprache zu programmieren.

Dabei soll dieses Buch jedoch kein Handbuch für die Programmierung des Prozessors Z80 sein (derartige Lehrbücher gibt es bereits in ausreichender Anzahl), sondern die Kenntnis der Speicherorganisation, der Portbelegungen usw. vermitteln, wie sie zum effektiven Programmieren nötig ist.

Kapitel 1: Das Displaymoderegister (CE)

Wie Sie schon aus dem BASIC-Handbuch des MZ-800 wissen, kann der MZ-800 in zwei verschiedenen Betriebsarten arbeiten:

Dem MZ-700 Modus, in dem der MZ-800 in allen Einzelheiten wie ein MZ-700 reagiert, und in dem MZ-800 Modus, in dem der Rechner erst richtig seine Fähigkeiten entfaltet.

Die Umschaltung zwischen beiden Betriebsarten nun geschieht nicht etwa, wie man dies nach der Lektüre des MZ-800 Handbuches glauben könnte, durch den "Systemschalter" an der Rückseite des Rechners, sondern über das Displaymoderegister. Der MZ-700/800-"Systemschalter" hat auf die Betriebsart, in der sich der Rechner befinder, keinerlei Einfluß. Er wird lediglich vom ROM nach dem Laden eines Programmes abgefragt. Das Monitor-ROM wählt dann je nach Stellung des Schalters die betreffende Betriebsart.

Der Systemschalter kann vom Benutzer problemlos über Port CEH abgefragt werden. Steht er auf MZ-700-Stellung, so ist das zweite Bit (BIT 1) der aus Port CEH gelesenen Daten rückgesetzt. Steht er auf MZ-800-Stellung, ist es gesetzt.

Die Auswahl eines Displaymodus bzw. der Betriebsart geschieht ebenfalls über Port CEH, jetzt jedoch als Ausgabeorperation. Hierzu zunächst eine Anmerkung: Prinzipiell kann zwischen MZ-700 und MZ-800 Modus gewählt werden. Hierbei ist jede Anzeigeart, die auf die hochauflösende Grafik zugreift, MZ-800 Modus.

An dieser Stelle ist es nun nötig, auf die Organisation des Grafikspeichers in den verschiedenen Auflösungen und Speicherausbaustufen einzugehen.

I. Speicheraufteilung ohne Grafikerweiterung

   In einem MZ-800 ohne Grafikerweiterung stehen 16KB Video-RAM (VRAM) zur Verfügung. Diese werden je nach Betriebsart folgendermaßen organisiert:

   Bei der Auflösung 640x200 Punkte existiert eine Grafikplatte von 16KB Länge, in der ein gesetztes Bit einen gesetzten Grafikpunkt (Palette 1) und ein rückgesetztes Bit einen gelöschten Grafikpunkt (Palette 0) repräsentiert.

Kompliziert sieht dies bei der Auflösung 320x200 Punkte aus: Nun existieren zwei Grafikplatten à 8KB, die durch Überlagerung die entsprechenden Palettencodes erzeugen.

Ein Bit, das weder in Ebene 1 noch in Ebene 2 gesetzt ist, ist ein gelöschter Punkt (Palette 0). Ein Bit, das in Ebene 1 aber nicht zugleich in Ebene 2 gesetzt ist, ist ein Punkt der Palette 1. Ein Punkt, der nur in Ebene 2 jedoch nicht zugleich in Ebene 1 gesetzt ist, codiert Palette 2. Ein Bit, das in beiden Grafikplatten gesetzt ist, codiert Palette 3.

II. Speicheraufteilung mit Grafikerweiterung

Ist der MZ-800 mit einer Grafikerweiterung ausgerüstet, so stehen nun als VRAM 32KB zur Verfügung, die je nach Auflösung folgendermaßen organisiert werden:

Im 620x200 Modus erfolgt nun die Auflösung wahlweise einfarbig wie ohne Grafikerweiterung, wobei nun zwischen zwei verschiedenen Grafikbildern umgeschaltet werden kann, oder in vier Farben, wobei die Verknüpfung der Bits in derselben Weise für den Palettencode maßgebend ist wie im 320x200 Modus ohne Grafikerweiterung.

Auch im 320x200 Modus sind nun zwei Möglichkeiten offen: Entweder der Betrieb mit vier Farben, wobei auch hier die Möglichkeit besteht, zwischen zwei Bildern umzuschalten, oder aber der Betrieb in 16 Farben. In diesem Fall stehen nun vier Grafikplatten bereit. Ihre Kombination erzeugt den Palettencode.

Die Datenübergabe an das Displaymoderegister erfolgt nun auf der Breite von vier Bits. Ihre Kombination erzeugt den Anzeigemodus.

Bit 3	Bit 2	Anzeigemodus
0	0	320x200, hochauflösend, daher MZ-800 Modus
0	1	640x200, hochauflösend, daher MZ-800 Modus
1	0	MZ-700 Modus
1	1	unzulässig

Bit 1	Bit 0	Grafikplattenanzeige	
		320x200	640x200
0	0	Bild A, Platte I+II	Bild A, Platte I
0	1	Bild B, Platte III+IV	Bild B, Platte III
1	0	16 Farben, Platten I-IV	4 Farben, Platten I+III

Wie man sieht, sind mit dem MZ-800 in der Grundversion die beiden untersten betriebsarten nicht möglich, da mangels Speicherplatz die Platten III und IV fehlen. Wählt man sie dennoch, so ist das Ergebnis des Zugriffs nicht gesichert.

Im folgenden nun einige Beispiele, um die Ansprache des Displaymoderegisters zu verdeutlichen:

Beispiel 1: Es soll der MZ-700 Modus gewählt werden. Die an das Displaymoderegister (DMR) auszugebenden Daten wären also
1000B (B=Binär) = 8

Das Maschinenprogramm lautet also:
```
LD A,08H
OUT (0CEH),A
```

Beispiel 2: Auf einem MZ-800 ohne Grafikerweiterung soll die Auflösung 320x200 Punkte gewählt werden. Da hier nur Grafikplatten I+II existieren, muß folgender Wert an das DMR ausgegeben werden:
0000B = 0

Das Maschinenprogramm lautet also:
```
LD A,0
OUT (0CEH),A
```

Beispiel 3: Die betriebsart 640x200 Punkte in vier Farben soll gewählt werden. Somit muß an das DMR ausgegeben werden:
0110B = 6

Das Maschinenprogramm lautet also:
```
LD A,06H
OUT (0CEH),A
```

Die obigen Beispiele sollten ausreichen, um das Arbeiten mit dem DMR zu verdeutlichen. In den kommenden Kapiteln wird nun die Ansprache der Grafikplatten bzw. des MZ-700 Bildschirms beschrieben.

Kapitel 2: Der MZ-700 Modus

Der MZ-700 Modus ist diejenige Betriebsart, in der der Anwender am einfachsten Texte usw. auf dem Bildschirm ausgeben kann. Durch die im MZ-800 implementierte PCG-Grafik (PCG = Programmable Character Generator = Programmierbarer Zeichengenerator) wird zudem selbst in diesem Modus eine gewisse Grafikfähigkeit erreicht.

Im MZ-700 Modus besteht der Speicher des MZ-800 im wesentlichen aus drei Speicherabschnitten:

Der Bereich 0000H-1000H kann wahlweise Monitor ROM oder RAM zur freien Programmierung enthalten,

der Bereich 1000H-CFFFH enthält im MZ-700 Modus immer frei verfügbares RAM und

der Bereich D000H-FFFFH enthält entweder den oberen Teil des Monitor ROM´s und das VRAM oder frei verfügbares RAM.

Wie aus den obenstehenden Ausführungen ersichtlich, besteht die Möglichkeit, den Inhalt bestimmter Speicherbereiche auszutauschen, ein Verfahren, das sofort verständlich wird, wenn man folgende Überlegungen anstellt:

Der MZ-800 besitzt 64KB für den Benutzer frei verfügbares RAM, mindestens 16KB VRAM, 12KB Monitor ROM und 4KB CG-ROM. Summiert man diese Werte auf, so kommt man auf ein Gesamtspeichervolumen von mindestens 96KB. Da der Prozessor des MZ-800 jedoch ein Z80 ist, also ein 8-Bit-Prozessor, der nur maximal 64KB direkt adressieren kann, muß eine andere Möglichkeit gefunden werden, um der CPU den Zugriff auf den Speicherüberhang von 32KB zu erlauben. Dies geschieht sowohl beim MZ-700 als auch beim MZ-800 durch die Methode des Bankswitching: Der Inhalt bestimmter Speicherbereiche kann ausgewechselt werden, so daß die CPU unter derselben Adresse auf zwei verschiedene Speicherbereiche zugreifen kann.

Die Bankswitching-Befehle des MZ-800 im MZ-700 Modus sind hierbei zu den Bankswitching-Befehlen des MZ-700 kompatibel, so daß Systemprogramme des MZ-700 wie z.B. PASCAL, FORTRAN etc. auch auf dem MZ-800 lauffähig sind.

Bankswitching erfolgt beim MZ-700/800 durch Ansprache bestimmter Ports. Hierbei ist es völlig belanglos, welcher Wert an den betreffenden Port ausgegeben wird.

Port	0000H-0FFFH	D000H-FFFFH
E0H	RAM	---
E1H	---	RAM
E2H	ROM (Monitor)	---
E3H	---	VRAM, ROM (Monitor)
E4H	ROM (Monitor)	VRAM, ROM (Monitor)
E5H	---	Zugriff gesperrt
E6H	---	Zustand wie vor der Sperre

Ein Beispiel: Dem Speicherbereich D000H-FFFFH soll VRAM/ROM zugewiesen werden. Der Bankswitching-Befehl lautet OUT (E3H),A. Andere Speicherbereiche werden durch diese Veränderung nicht berührt. Die Bildschirmanzeige ändert sich bei Ein- oder Ausbanken des VRAM´s, also des Bildschirmspeichers, nicht.

Im folgenden soll nun demonstriert werden, auf welche Weise der MZ-700 Bildschirm organisiert ist und wie Zeichen auf ihm codiert werden.

Der Bildschirm im MZ-700 Modus besteht aus zwei Teilen: Dem **Zeichenspeicher**, in dem die auf dem Bildschirm stehenden Zeichen gespeichert sind, und dem **Farbspeicher**, der die den Zeichen zugehörige Farbkennung enthält.

I. Der Aufbau des Zeichenspeichers

Der Zeichenspeicher belegt die Adressen D000H-D7FFH. Davon wird jedoch nur der Bereich D000H-D3F7H (einschließlich) auf dem Bildschirm angezeigt. Der Restspeicher steht zusätzlich zur Verfügung, wenn man zusätzlichen Speicherplatz benötigt, etwa um einen Editor zu schreiben, der zwei Bildschirmseiten umfaßt und zwischen diesen rollen/blättern kann, wie dies z.B. beim MZ-700-BASIC der Fall ist. Auf dem Bildschirm angezeigt werden kann jedoch in jedem Fall nur der Bereich D000H-D3E7H. Hier existiert also nicht wie im MZ-800 Modus eine Umschaltmöglichkeit zwischen zwei Seiten.

Der Bildschirm umfaßt 40x25 Zeichen. Im VRAM stehen diese der Reihe nach von links oben nach rechts unten in zeilenweiser Reihenfolge. Zur Codierung eines Zeichens wird je ein Byte benötigt. Die Codierung erfolgt nach dem **Anzeigecode** (Tabelle siehe Anhang).

Die VRAM-Belegung ist also:

	0	1	2	.....	38	39
0	D000H	D001H	D002H	.....	D026H	D027H
1	D028H	.....	.....	.....	.....	.....
	.....	.....	.....	.....	.....	.....
25	D3C0H	.....	.....	.....	.....	.....

Da der MZ-800 im Monitor immer im MZ-700 Modus arbeitet und das VRAM in den Hauptspeicher eingeblendet ist, kann man die Arbeitsweise des VRAM´s einfach zeigen:

Geben Sie den Monitorbefehl MD000 ein und ändern Sie den Wert dieser Adresse in 01, den Anzeigecode des Buchstabens A. Wie Sie sehen, erscheint in der linken oberen Bildschirmecke ein A.

II. Der Aufbau des Farbspeichers

Der Farbspeicher belegt die Adressen D800H-DFFFH, die Codierung erfolgt dabei folgendermaßen: Adresse D800H enthält den Farbcode des Zeichens in Adresse D000H, D801H codiert die dem Zeichen in D001H zugehörige Farbe usw. usw. Die letzte Adresse im Farbspeicher, die noch auf dem Bildschirm erscheint, ist DBE7H (D800H+03E7H). Der restliche Speicher dient wie beim Zwischenspeicher nur der Speicherung einer eventuellen zweiten Bildschirmseite, kann jedoch nicht angezeigt werden.

Im MZ-700 Modus sind von den 16 Farben des MZ-800 nur acht darstellbar (und zwar, obwohl man dies nicht annehmen sollte, die aufgeblendeten Farben, also die Farben 9-15. Nur Schwarz mit Code 0 ist nicht aufgeblendet. Farbcode 1 im MZ-700 Modus entspricht also Code 9 beim Arbeiten mit HRG, 2 entspricht 10 etc. etc.). Somit reichen zur Codierung einer Farbe 3 Bits aus ($2^3 = 8$).

Die Farbe wird in jedem Farbbyte des VRAM´s nun folgendermaßen codiert: Das untere Nibble des Farbbytes enthält die Hintergrundfarbe und das obere Nibble die Vordergrundfarbe. Der Code 56H bedeutet also, daß das Zeichen Zyan (5) auf Gelb (Code 6) dargestellt wird. Analog entspricht 71H der Farbkennung Weiß auf Blau. 16H ist ist Blau auf Gelb etc.

Während das obere Bit des unteren Nibbles (Bit 3) keine Rolle spielt, bewirkt das oberste Bit des oberen Nibbles die Umschaltung zwischen zwei Zeichensätzen: Der MZ-700 und somit auch der MZ-800 im 700er Modus besitzt zwei Zeichensätze und kann somit nicht nur 256 sondern 512 verschiedene Zeichen darstellen. Ist nun Bit 7 der Farbdaten eines Zeichens rückgesetzt, so erfolgt die Anzeige nach dem ersten Zeichensatz. Ist es gesetzt, wird das Zeichen des zweiten Zeichensatzes angezeigt. Eine Tabelle des zweiten Zeichensatzes finden Sie im Anhang.

Diejenigen Zeichen, die im Monitor angezeigt werden, sind Elemente des ersten Zeichensatzes. Der zweite Zeichensatz enthält vorwiegend für Spiele oder sonstige Anwendungen nützliche Sonderzeichen (verschiedene Monster, Roboterfiguren, griechische Sonderzeichen usw.).

Ein Anwendungsbeispiel: Drücken Sie RESET und dann M. Sie befinden Sie nun im Monitor. Nun soll der Farbcode des in der linken oberen Bildschirmecke stehenden Sternchens derart geändert werden, daß das Zeichen Schwarz auf Weiß dargestellt wird und zudem aus dem

zweiten Zeichensatz entnommen wird. Der Farbcode wäre also 07H. Da der zweite Zeichensatz aktiviert werden soll, muß noch BIT 7 gesetzt werden. Dies entspricht einer Addition von 80H. Die Farbkennung ist also 87H. Geben Sie diesen Wert nun vermittels des Befehls MD800 in Adresse D800H ein. Sie werden sehen, daß sich einerseits die Farbe ändert und andererseits die Zeichendarstellung wechselt: Anstelle des Sternchens sehen Sie nun eine Reihe sich verjüngender Striche.

Ein anderes Beispiel: Ein Programm soll den gesamten Bildschirm auf die Farbkennung Zyan auf Schwarz bringen. Alle Zeichen sind dem ersten Zeichensatz zu entnehmen. Der in den Adressen D800H-D3E7H abzulegende Farbwert ist also 50H (5 = Zyan, 0 = Schwarz). Das Programm lautet:

```
LD HL,D800H     ; HL:=Bildschirmanfang
LD DE,D801H     ; DE:=2. Zeichen auf Bildschirm
LD BC,03E7H     ; BC:=Länge des Farb-RAM
LD (HL),50H     ; Bildanfang auf Farbwert setzen
LDIR            ; Führt aus: (DE):=(HL), HL:=HL+1, DE:=DE+1,
                             BC:=BC-1 bis BC=0
JP EA5EH        ; MZ-800 Monitor Warmstart
```

Im Hexadezimalcode ist das: 21 00 D8 11 01 D8 01 E7 03 36 50 ED B0 C3 5E EA

Geben Sie dieses Programm mit dem M-Befehl z.B. ab Adresse 5000H ein und starten Sie es. Sie sehen, wie sich der Bildschirm entsprechend einfärbt.

Nach den obigen Ausführungen sollte die Arbeitsweise des VRAM´s verstanden worden sein. Sollten aber noch einzelne Punkte unklar sein, so empfiehlt es sich, durch Ausprobieren die Fertigkeiten zu vertiefen.

All diese bis jetzt besprochenen Befehle für Bankswitching und die Speicherorganisation des VRAM´s sind ebenso wie beim MZ-700 aufgebaut. Der MZ-800 bietet jedoch auch im MZ-700 Modus über den zweiten Zeichensatz hinausgehende Grafikmöglichkeiten: PCG-Grafik, also die Möglichkeit, den Character Generator frei zu programmieren. Dies bedeutet in der Praxis, daß der Anwender die Möglichkeit hat, die Zeichenmatrix der auf dem Bildschirm darstellbaren Symbole nach Belieben abzuändern. Diese Grafikmöglichkeit darf nicht mit der hochauflösenden Grafik verwechselt werden.

Hochauflösende Grafik bedeutet, daß über einen Bildschirm hinweg in beliebiger Weise einzelne Punkte gesetzt oder gelöscht werden können.

PCG-Grafik hingegen bedeutet, daß die Zeichenmatrix eines jeden Symbols beliebig umdefiniert werden kann. Ändert man z.B. die Zeichenmatrix des Buchstabens A derartig ab, daß stattdessen ein UFO angezeigt wird, so steht nach dieser Veränderung an jeder Stelle des Bildschirms, an der sich zuvor ein A befand, ein UFO. Der Code im VRAM wurde ja nicht verändert und stellt jetzt auf dem Bildschirm nur ein anderes Zeichen dar.

Zur Realisierung der PCG-Grafik stehen im Rechner zwei Speicher bereit, die per Bankswitching in den Hauptspeicher eingeblendet werden können: Das CG-ROM und das CG-RAM.

Das CG-RAM enthält hierbei immer die Zeichenmatrix, die für die Ausgabe auf dem Bildschirm relevant ist. In ihm müssen also bei Zeichenneudefinitionen Veränderungen vorgenommen werden.

Das CG-ROM hingegen enthält die ursprüngliche Zeichenmatrix, d.h. diejenige Zeichendefinition, die nach RESET oder Einschalten gewählt wird und im Anhang in den Tabellen des Anzeigecodes zu finden ist.

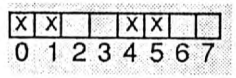
Sowohl CG-ROM als auch CG-RAM umfassen 4KB Speicher. Für die Festlegung der Zeichenmatrix eines Symbols sind 8 Byte erforderlich. Da insgesamt 512 Zeichen definiert sind, müssen auch 512x8=4KB Speicher bereitstehen.

Jedes einzelne Symbol besteht aus einer 8x8 Punkte großen Matrix. Die Verschlüsselung geschieht von oben nach unten, wobei die Matrix einer Zeile in einem Byte (1 Byte = 8 Bit, somit kann die Matrix einer Zeile in einem Byte vollständig aufgenommen werden) zusammengefaßt ist.

Innerhalb einer Zeile wird von links nach rechts verschlüsselt, wobei **Bit 0 links liegt.**

Beispiel für die Verschlüsselung einer Zeile: Eine Zeile einer Zeichenmatrix soll folgendermaßen definiert werden (x=gesetzt). Da Bit 0 links liegt, enspricht diese Matrix dem Wert 00110011B = 33H. In der entsprechenden Adresse des CG-RAM muß also der Wert 33H abgelegt werden.

Nun zur Lage von CG-ROM und -RAM: Beide liegen im mittleren Speicherbereich 1000H-CFFFH, und zwar an den Rändern:



1000H-1FFFH : CG-ROM
C000H-CFFFH : CG-RAM

Das Einblenden dieser Speicherbereiche in den Hauptspeicher geschieht vermittels zweier besonderer Bankswitching-Befehle. Im Gegensatz zu den anderen Banking-Kommandos handelt es sich jedoch hier um Port-Einlese-Befehle:

Befehl	1000H-1FFFH	C000H-CFFFH
IN A,(E0H)	CG-ROM	CG-RAM
IN A,(E1H)	Freies RAM	Freies RAM

Wie der oben stehenden Tabelle zu entnehmen ist, können CG-ROM und -RAM immer nur paarweise, d.h. zusammen ein- bzw. ausgeblendet werden.

Nun muß noch der Frage nachgegangen werden, wie die Zeichenmatrizen im CG-ROM und -RAM definiert sein müssen. Ihre Reihenfolge ist diejenige ihres **Anzeigecodes**, d.h. das Zeichen mit dem Anzeigecode 2 folgt auf das mit dem Code 1, welches wiederum auf Code 0 folgt. Da jedes Zeichen zu seiner Definition 8 Byte benötigt, sieht die Speicheraufteilung im CG-ROM folgendermaßen aus:

1000H-1007H:	Anzeigecode 0 (Space)	1. Zeichensatz
1008H-100FH:	Anzeigecode 1 (A)	1. Zeichensatz
1010H-1017H:	Anzeigecode 2 (B)	1. Zeichensatz
.....:	Anzeigecode.....	1. Zeichensatz
17F8H-17FFH:	Anzeigecode 255	1. Zeichensatz
1800H-1807H:	Anzeigecode 0	2. Zeichensatz
1808H-180FH:	Anzeigecode 1	2. Zeichensatz
.....:	Anzeigecode.....	2. Zeichensatz
1FF8H-1FFFH:	Anzeigecode 255	2. Zeichensatz

Wie aus dem Speicherplan ersichtlich, folgen die Zeichen ohne Lücken aufeinander. Nach den Zeichen des ersten Zeichensatzes folgt der zweite Zeichensatz in derselben Ordnung.

Ebenso wie das CG-ROM ist auch das CG-RAM organisiert, nur daß die Startadresse C000H und nicht 1000H ist. Eine Neuinitialisierung des CG-RAM´s ist also durch einfaches Kopieren des CG-ROM-Inhaltes in das CG-RAM möglich.

Ein Anwendungsbeispiel für PCG-Grafik: Es soll das Zeichen "B" (Anzeigecode 2) so umdefiniert werden, daß es wie folgt aussieht:

Byte 1	XX XX	= 00110011B = 33H
Byte 2	XX XX	= 00110011B = 33H
Byte 3	XX XX	= 11001100B = CCH
Byte 4	XX XX	= 11001100B = CCH
Byte 5	XX XX	= 11001100B = CCH
Byte 6	XX XX	= 11001100B = CCH
Byte 7	XX XX	= 00110011B = 33H
Byte 8	XX XX	= 00110011B = 33H

Die neue Matrix hat also die Codes 33H, 33H, CCH, CCH, CCH, CCH, 33H, 33H (beim Codieren darf nie vergessen werden, daß Bit 0 jedes Bytes im Zeichen links erscheint. Die Binärcodes sind gegenüber der tatsächlichen Matrix also seitenverkehrt).

Die Adresse, ab der die Matrix abgelegt werden muß, berechnet sich nun folgendermaßen: Der Anzeigecode ist 2. Da jedes Zeichen 8 Byte belegt, berechnet sich die Anfangsadresse der Matrix aus 2x8=16=10H. Das Zeichen ist aus dem ersten Zeichensatz. Somit muß C000H (Startadresse des CG-RAM´s) addiert werden. Wäre das Symbol aus dem zweiten Zeichensatz, so müßte C800H addiert werden. Die Adresse ist also 10H+C000H=C010H.

Da die Ansprache des Bankswitching-Befehles IN A,(E0H) nicht nur das CG-RAM sondern auch das CG-ROM einblendet, kann vom Monitor aus nicht einfach auf das CG-RAM zugegriffen werden (der Monitor benötigt den Bereich 1000H-1200H als Arbeitsbereich). Es muß also für die Zeichendefinierung ein spezielles Programm geschrieben werden, wie etwa folgendes:

```
        LD HL,TBL       ; HL auf Start der Tabelle setzen
        LD DE,C010H     ; DE auf Beginn der Matrix im CG-RAM
        LD BC,8         ; BC erhält Länge der Tabelle
                          zugewiesen
        IN A,(E0H)      ; CG-Speicher einbanken
        LDIR            ; Tabelle in CG-Speicher übertragen
        IN A,(E1H)      ; CG-Speicher ausbanken
        JP EA5EH        ; Monitor-Warmstart. Achtung: Jeder
                          Monitor-Kaltstart etwa durch Sprung
                          zu 0000H initialisiert das CG-RAM neu
                          aus dem ROM!!
TBL:DEFB 33H, 33H,CCH,CCH,CCH,CCH,33H,33H
                        ; Tabelle für Matrix
```

Im Hexcode zur Eingabe ab Adresse 5000H lautet das Programm: 21 12 50 11 10 C0 01 08 00 DB E0 ED B0 DB E1 C3 5E EA 33 33 CC CC CC CC 33 33.

Geben Sie dieses Programm ab Adresse 5000H ein und starten es mit J5000, so werden Sie sehen, wie sich das Erscheinungsbild aller B´s auf dem Bildschirm ändert.

Eine Besonderheit des MZ-700 Modus ist es, daß die Kommunikation zu den anderen Bausteinen wie Counter etc. nicht über Ports geschieht, sondern als sogenanntes "Memory mapped I/O" (Ein/Ausgabeoperationen über Speicherstellen wie bei Hauptspeicherzugriff).

Dieses noch vom MZ-80K gewissermaßen "verschleppte" System wurde implementiert, um I/O-Zugriffe auf Tastatur etc. wie bei einem Hauptspeicherzugriff behandeln zu können. Daten können also durch Lade/Lesebefehle übergeben werden und die Ports bleiben für andere Anwendungen frei. Dafür bietet dieses System jedoch den Nachteil, daß zum Zugriff auf diese Speicherstellen das entsprechende RAM eingebankt werden muß (beim MZ-800/700 liegen diese Speicherstellen im Bereich E000H-E008H und werden immer zugleich mit dem VRAM und MZ-800 Monitor ein/ausgeblendet) und daß somit z.B. ein ab Adresse F000H stehendes Programm nicht mit der Tastatur kommunizieren kann, da es sich in diesem Fall selbst ausbanken würde. Die Folge wäre, daß das System abstürzt.

Eine Beschreibung der für I/O-Operationen im MZ-700 Modus verantwortlichen Speicherstellen findet sich im Kapitel 4.

Kapitel 3: Der MZ-800 Modus

Wie schon aus den Ausführungen in Kapitel 1 bekannt, umfaßt der MZ-800 Modus alle Betriebsarten, in denen mit hochauflösender Grafik gearbeitet wird. Im MZ-800 Modus arbeitet der MZ-800 völlig anders als im MZ-700 Modus, so ist z.B. die Speicheraufteilung anders, I/O-Operationen laufen über Ports ab usw.

Wie auch im 700er Modus ist im MZ-800 Modus Bankswitching möglich. Aber obwohl für die Bankswitching-Befehle dieselben Ports wie im 700er Modus verwendet werden, ist die Wirkungsweise der Befehle anders.

Der Hauptspeicher besteht aus folgenden Bereichen:

0000H-0FFFH: Enthält wahlweise Monitor-ROM oder freies RAM
1000H-1FFFH: Enthält entweder CG-ROM oder freies RAM
2000H-7FFFH: Enthält immer freies RAM
8000H-BFFFH: Enthält entweder das VRAM (HRG-RAM) oder freies RAM. Bei Einbanken des VRAM ist zu beachten, daß seine Länge variabel ist:

Im 320x200 Modus benötigt eine Grafikplatte 8KB Speicher, das VRAM geht also von 8000H-9FFFH. Der Bereich A000H-BFFFH bleibt auch bei eingebanktem VRAM freies RAM.

Im 640x200 Modus hingegen wird für eine Grafikplatte ein Speicherbereich von 16KB benötigt. Somit belegt das VRAM beim Einbanken die Speicherplätze 8000H-BFFFH.

C000H-DFFFH: Immer freies RAM
E000H-FFFFH: Wahlweise freies RAM oder Monitor-ROM.

Folgende Bankswitching-Befehle sind möglich:

I. Portansprache per Ausgabebefehl (OUT (n),A)

Port	0000H-0FFFH	1000H-1FFFH	8000H-BFFFH	E000H-FFFFH
E0H	freies RAM	freies RAM	---	---
E1H	---	---	---	freies RAM
E2H	Monitor-ROM	---	---	---
E3H	---	---	---	Monitor-ROM
E4H	Monitor-ROM	CG-ROM	VRAM	Monitor-ROM
E5H	---	---	---	Zugriff gesperrt
E6H	---	---	---	wie vor Sperre

II. Portansprache per Lesebefehl (IN A,(n))

Port	1000H-1FFFH	8000H-BFFFH
E0H	CG-ROM	VRAM
E1H	freies RAM	freies RAM

Bitte beachten Sie, daß die Länge des VRAM variabel ist (es kann von dem Bankswitching-Befehl also auch nur der Bereich 8000H-9FFFH betroffen sein).

Wie im MZ-700 Modus ist es für das Bankswitching belanglos, welchen Wert man an den Port ausgibt. Allein die Ansprache des Ports bewirkt das Umschalten.

Im folgenden soll nun das Arbeiten mit der hochauflösenden Grafik erläutert werden.

Wie schon aus Kapitel 1 bekannt, stellt der MZ-800 die Farbkennung (Palettencode) jedes Grafikpunktes durch Überlagerung mehrerer Grafikplatten dar. Man sollte also erwarten, daß verschiedene Bankswitching-Befehle bereitstehen, mit denen es dann möglich ist, zunächst Grafikplatte I und dann Platte II usw. einzublenden. Ein solches Verfahren hätte für den Benutzer aber eine Reihe von Nachteilen: Wenn zum Beispiel in der Auflösung 320x200 Punkte in 16 Farben ein Punkt in Palette 15 gesetzt werden sollte, so müßte der Programmierer diesen Punkt nacheinander in vier verschiedenen Grafikplatten setzen. Die Folge wäre, daß alle Arbeiten mit HRG-Grafik recht langsam wären.

Um dies zu vermeiden, geht der MZ-800 einen anderen Weg: In zwei Registern, dem READ-Format-Register und dem WRITE-Format-Register, legt der Benutzer fest, auf welche Weise Schreib- und Leseoperationen mit dem VRAM verknüpft werden soll. Da durch diese Methode spezielle Befehle wie Punkte in einer bestimmen Palette zu löschen/setzen schon von der Hardware bereitgestellt werden, ist der Programmierer in seiner Arbeit zudem wesentlich entlastet.

Der Anwender schreibt also, wenn er eine Ladeoperation etwa der Art

```
LD A,FFH
LD (8000H),A
```

ausführt, nicht wie z.B. im 700er Modus direkt in das VRAM, sondern er gibt durch die von ihm gewählte Adresse nur an, mit welcher Adresse des VRAM der Grafik-Schip die von ihm dort abgelegte Bitmatrix verknüpfen soll, wobei die Verknüpfung durch den Inhalt des WRITE-Format-Registers festgelegt ist.

Analog das Verfahren bei Leseoperationen: LD A,(8000H) gibt nicht "den Wert" des VRAM´s an dieser Adresse. Dieser zurückgelieferte Wert ist vielmehr abhängig vom READ-Format-Register: Abhängig von der durch das READ-Format-Register gewählten Lesebetriebsart können aus derselben Adresse verschiedene Werte zurückgeliefert werden.

Der Vorteil dieses Verfahrens liegt auf der Hand: Nicht nur, daß der Zugriff durch durch die Hardware realisierte Anweisungen wie PSET, RESET etc. wesentlich schneller wird. Zusätzlich besteht z.B. die Möglichkeit, problemlos den Inhalt einer Grafikplatte in eine andere zu übertragen, da READ- und WRITE-Format-Register ja getrennt sind.

Das Format der an das READ-Format-Register zu übergebenden Daten ist folgendermaßen:

R/S			A/B	IV	III	II	I
7	6	5	4	3	2	1	0

Port CDH

Hierbei bedeuten die römischen Ziffern die Grafikplatten, die von einem Lesevorgang betroffen sein sollen. Ist das entsprechende Bit gesetzt, d.h. aktiv, so wird die entsprechende Grafikplatte bei Lesevorgängen abgefragt, andernfalls nicht.

Der Zusammenhang zwischen Grafikplattenkombination und Palettencode ist folgender: Der Palettencode, der durch die Kombination von bestimmten Grafikplatten dargestellt wird, ist diejenige Zahl, die sich ergibt, wenn man das Bitmuster nach der für das untere Nibble geltenden Vorschrift bildet.

Ein Beispiel: Welchen Palettencode codieren die Platten I+III? Antwort: Nach obigem Muster ergibt sich 0101B=5. Diejenigen Punkte, die nur in den Platten I+III gesetzt sind, nicht aber in andern Platten, stellen also Palette 5 dar.

Ein anderes Beispiel: Es sollen die Punkte der Palette 2 bei Leseoperationen zurückgeliefert werden. Da 2 dem Binärwert 0010B entspricht, muß das untere Nibble des READ-Format-Registers also 0010B sein.

Das Bit A/B bedeutet: Ist es zurückgesetzt, so wird bei Displaymodi, bei denen zwei Bilder zur Verfügung stehen (z.B. 640x200 Punkte einfarbig, wenn Grafikerweiterung vorhanden) das Bild A genommen. Ist es gesetzt, so werden die Daten aus Bild B gelesen. Achtung: Bei einem MZ-800 ohne Grafikerweiterung muß dieses Bit immer Null sein, da ja keine zweite Ebene existiert.

Das Bit R/S bedeutet Read/Search und ist Teil der Daten, der die Modussteuerung für Leseoperationen festlegt.

Ist dieses Bit zurückgesetzt, also 0, so wird der Wert einer bestimmten Grafikplatte gelesen. Diese Möglichkeit wirft jedoch in manchen Fällen Probleme auf: Da z.B. Palette 3 durch Kombination von Ebene I+II dargestellt wird, ist ein in PAL 3 gesetzter Punkt sowohl in Ebene 1 wie auch in Ebene 2 gesetzt. Würde man also Platte I abfragen,um alle Punkte zu erhalten, die in PAL 1 gesetzt sind, so würde man auch die in PAL 3 gesetzten Punkte erhalten, da diese ja auch in Platte I erscheinen. Um nun alle solchen "Blindgänger" auszuschalten wäre ein zeitaufwendiger Vergleich mit den anderen Grafikplatten erforderlich. Diese Arbeit nimmt der Grafikchip dem Anwender ab, wenn Bit 7 des READ-Format-Registers gesetzt ist: Wird nun

der Wert der Grafikebene I abgefragt, so "weiß" der Rechner, daß nicht Platte 1 sondern Palette 1 gemeint ist und entfernt aus dem Leseergebnis automatisch alle Punkte, die auch in anderen Platten auftauchen.

Bevor wir nun auf das WRITE-Format-Register zu sprechen kommen, ist es empfehlenswert, sich zu verdeutlichen, wie das VRAM aufgebaut ist.

Der Bildschirm für hochauflösende Grafik wird Bit-mapped dargestellt, d.h. ein innerhalb des VRAM's gesetztes Bit repräsentiert auch einen gesetzten Punkt auf dem Bildschirm. Der Bildschirminhalt jeder Grafikplatte ist nun in den Adressen 8000H-9FFFH (640x200: 8000H-BFFFH) von links oben nach rechts unten gespeichert, und zwar wie der MZ-700 Bildschirm in zeilenweiser Reihenfolge. Adresse 8000H enthält also die ersten acht Grafikpunkte mit den Koordinatn 0/0 bis 7/0, Adresse 8001 die Punkte 8/0 bis 15/0 usw. usw.

Innerhalb eines Bytes erfolgt die Verschlüsselung jedes 8er Blocks von links nach rechts, wobei Bit 0 wie bei der PCG-Grafik des MZ-700 Modus immer links liegt.

Nun soll auf das WRITE-Format-Register näher eingegangen werden. Wie schon gesagt, legt das WRITE-Format-Register fest, auf welche Weise die vom Programm in einer Speicherstelle des VRAM abgelegte Daten vom Rechner mit den schon im VRAM stehenden Daten verknüpft werden sollen. Die an das WRITE-Format-Register zu übergebenden Daten haben folgendes Format:

WM2	WM1	WM0	A/B	IV	III	II	I
7	6	5	4	3	2	1	0

Port CCH

Ebenso wie beim READ-Format-Register bezeichnet das untere Nibble die anzusprechenden Grafikplatten. Die Codierung erfolgt analog und soll hier deshalb nicht nochmals erläutert werden.

A/B kennzeichnet bei Betriebsarten, in denen zwei Grafikbildschirme zur Verfügung stehen, wieder das anzusprechende Bild: Ist es gesetzt, so ist Bild B der Adressat. Bei einem MZ-800 ohne Grafikerweiterung muß dieses Bit immer rückgesetzt sein.

Eine wichtige Rolle kommt den Bits WM0 bis WM2 zu: Sie bestimmen die Art und Weise, auf die die Daten, die der Benutzer ablegt, mit dem VRAM verknüpft werden.

Beispiel:
```
LD A,10000010B
OUT (CCH),A
LD A,11001100B
LD (8000H),A
```

Ergebnis: Platte I: (8000H):= 11001100B
Platte II: (8000H):= 0
Je nach Anzeigeart (640x200 in vier Farben etc.) werden andere Platten zurückgesetzt.
Beispiel: In der Betriebsart 320x200 in 16 Farben wird 30H in Platte I unter REPLACE Verknüpfung abgelegt. Platten II, III und IV erhalten den Wert 0.
Beispiel: In der Betriebsart 640x200 in vier Farben wird 30H im REPLACE-Mode in Platte I abgelegt, Platte III erhält den Wert Null.

6 PSET
Ermöglicht das Setzen einzelner Punkte in einer bestimmten Palette. In den spezifizierten Grafikplatten wird jedes in den abgelegten Daten gesetzte Bit gesetzt, in den anderen Platten wird dieses Bit rückgesetzt. In den Daten nicht gesetzte Bits werden nicht verändert.

Beispiel:
```
LD A,11000010B
OUT (CCH),A
LD A,00001000B
LD (8000H),A
```

Ergebnis: Platte I: Bit 3 in 8000H wird gesetzt
Platte II: Bit 3 in 8000H wird rückgesetzt
Ebenso wie bei REPLACE sind die rückgesetzten Platten je nach Betriebsart unterschiedlich.

Im folgenden noch einige wichtige Anmerkungen zu der Handhabung von READ- und WRITE-Format-Register:

1 Die in der Betriebsart 640x200 Punkte in vier Farben zur Verfügung stehenden Grafikplatten heißen I und III und nicht wie vielleicht angenommen I und II.

Um also in dieser Betriebsart in Palette 3 zu schreiben, müssen die Platten I und III als Zielebenen angegeben werden.

2 Bei Anzeigearten mit zwei zur Verfügung stehenden Bildern behalten die Grafikebenen des Bildes B ihre normalen Bezeichnungen:

320x200, 4 Farben: Bild A: Platte I+II, Bild B: Platte III+IV
640x200, 1 Farbe: Bild A: Platte I , Bild B: Platte III

Soll nun z.B. in der Betriebsart 320x200 Punkte auf Bild B geschrieben werden, so muß z.B. bei Schriftpalette 3 Bit B/A gesetzt sein. Zielplatten sind III+IV.

3 Auch im MZ-700 Modus müssen READ- und WRITE-Format-Register gesetzt sein. Dies liegt daran, daß der MZ-800 Platte I doppelt belegt: Im 700er Modus enthält diese Platte CG-RAM, VRAM und FarbRAM. Im 800er Modus dient sie zur Aufnahme von Grafikpunkten.

READ- und WRITE-Format-Register müssen im MZ-700 Modus auf 1 gesetzt werden.

Nach soviel Theorie nun etwas Praxis: Wie schreibe ich ein Programm, das erst den Bildschirm löscht und dann in der linken oberen Bildschirmecke (HOME-Position) den Buchstaben "A" ausdruckt?

Anzeigebetriebsart soll Auflösung 320x200 Punkte in vier Farben sein. Das Zeichen soll in Palette 3 erscheinen.

Das Programm lautet:

```
ORG 5000H       ; Programm-Startadresse ist 5000H
LD A,0
OUT (CEH),A     ; 320x200 in vier Farben
LD A,3
OUT (CCH),A     ; WRITE
OUT (CDH),A     ; READ
LD HL,8000H     ; HRG-RAM Start
LD DE,8001H     ; HL + 1
LD BC,1F3FH     ; VRAM Länge (1F3FH = 7999)
IN A,(E0H)      ; VRAM einbanken
LD (HL),0
LDIR            ; Löschen
LD HL,8000H     ; HRG-RAM-Start
LD BC,40        ; Screen offset (40 Bytes sind eine Zeile)
LD DE,1008H     ; Matrix des Zeichens A im CG-ROM
EXX             ; Vertausche mit Zweitregister
```

```
        LD B,8          ; B´=8 (es müssen acht Byte übertragen
                          werden)
        LD A,83H        ; REPLACE-Mode in PAL 3
        OUT (CCH),A     ; WRITE-Format
LBL1:   EXX             ; Wieder auf Erstregister schalten
        LD A,(DE)       ; Akkumulator erhält Matrixbyte
        LD (HL),A       ; Übertrage in HRG-RAM
        INC DE          ; DE auf nächstes Matrixbyte setzen
        ADD HL,BC       ; HL um 40 erhöhen (nächste HRG-Zeilen)
        EXX             ; Wieder Zweitregister B enthält
                          Schleifenzähler
        DJNZ LBL1       ; Schleifenwiederholung
        LD A,11H        ; PAL 1 auf BLAU
        OUT (F0H),A
        LD A,22H        ; PAL 2 auf ROT
        OUT (F0H),A
        LD A,3FH        ; PAL 3 auf WEIß
OUT(F0H),A XOR A        ; Akkumulator auf Null setzen, also PAL 0 auf
                          Schwarz
        OUT (F0H),A
        HALT            ; Ende
```

Wer beim Lesen des Programmes genau aufgepaßt hat, der hat natürlich gemerkt, daß eine Umschaltung im Programm unnötig ist. Die Initialisierung des WRITE-Format-Registers auf REPLACE-Modus: Da ich in Palette 3 schreibe und somit Single Write auf beide Grafikplatten gleichzeitig einwirkt, besteht keine Gefahr, daß durch Punkteüberlagerung eventuell eine unerwünschte Palette erzeugt wird. Somit könnte weiterhin mit Single Write gearbeitet werden.

Bisher wurde immer nur von der Erzeugung von Palettencodes gesprochen, aber nie davon, auf welche Weise man den einzelnen Palettencodes Farben zuordnet. Dies soll im folgenden erläutert werden:

Wie bekannt, können beim MZ-800 vier Palettencodes beliebige Farbwerte zugewiesen werden, so daß der Anwender bei Betriebsarten mit niedriger Farbauflösung aus den 16 verfügbaren Farben die von ihm gewünschten auswählen kann.

Die Zuweisung von Farb- zu Palettencodes geschieht mittels Port F0H. Die Belegung der Bits der auszugebenden Daten ist folgendermaßen:

	S2	S1	S0	I	G	R/SW1	B/SW0
7	6	5	4	3	2	1	0

Port F0H

Die Bits S1 und S0 codieren den gewünschten Palettencode. Das untere Nibble bezeichnet den zuzuweisenden Farbwert. RGBI steht hierbei für diejenigen Farbkomponenten, aus denen sich die Farben zusammensetzen.

Ein Beispiel:
```
LD A,36H
OUT (F0H),A
```
Setzt Palette 3 auf Farbe 6 (Gelb).

Schwieriger wird die Ansprache im Modus 320x200 in 16 Farben: Hier stehen zwar 16 Paletten zur Verfügung. Von diesen können jedoch immer nur vier auf einen beliebigen Farbwert gesetzt werden. Die anderen Palettencodes codieren ihren Farbwert.

Dies heißt also, daß Palette 0 auch Farbe 0 bedeutet, PAL 1 auch Farbe 1 etc.

Wie schon aus dem BASIC bekannt, können im 320x200 in 16 Farben Modus immer nur vier Paletten vom Anwender spezifiziert werden. Die anderen werden vom System automatisch verwaltet.

Diese Umschaltung zwischen Palettenblöcken, die je vier Paletten umfassen, geschieht vermittels Bit 6 des Palettenregisters: Ist es gesetzt, so bestimmen Bit 0 und 1 der Daten den Palettenblock.

Beispiel:
```
LD A,40H
OUT (F0H),A
```
Schaltet auf Palettenblock 0. Es können also nun den Paletten 0 bis 3 Farbwerte zugewiesen werden.
Achtung: Innerhalb jedes Palettenblocks sind die Paletten ab Null durchnummeriert.

Beispiel: Es soll Palette 5 auf die Farbe 15 gesetzt werden. PAL 5 ist die zweite Palette des zweiten Palettenblocks, also innerhalb von Block 1 PAL 1. Das Programm lautet:

```
LD A,41H                ; Block 1
OUT (F0H),A
LD A,1FH                ; Farbe
OUT (F0H),A
```

Eine der besonders interessanten Eigenschaften des MZ-800 ist es, daß der Video-Chip imstande ist, per Befehl den Bildschirm auf- oder abwärts zu scrollen, also die Fähigkeit zum Hardwarescroll besitzt. Dieser Hardwarescroll ist es auch, der im MZ-800 Demo das weiche Auf- und Abwärtsscrollen des Schriftzuges "MZ-800" ermöglicht.

Per Hardwarescroll können beliebige Bildschirmausschnitte in beliebigen Schritten auf/abwärtsgescrollt werden. Die Datenübergabe geschieht über Ports.

Folgende Register sind für das Hardwarescrolling wichtig:

Scroll start register (SSA): Enthält, ab welcher Bildschirmzeile gescrollt werden soll (einschließlich). Hierbei können nicht HRG-Zeilen sondern nur ganze Bildschirmzeilen (also von 0 bis 24) angegeben werden. Vor Übergabe an das SSA müssen die Werte noch mit 5 multipliziert werden.

Beispiel: Es soll ab Bildschirmzeile 0 gescrollt werden. SSA muß also den Wert 0x5=0 erhalten.

Scroll end register (SEA): Enthält ab welcher Bildschirmzeile einschließlich nicht mehr gescrollt werden soll. Wie beim SSA muß auch hier die Nummer der Bildschirmzeile mit 5 multipliziert werden. Zugelassen für das SEA sind die Zeilen 1-25 (25 um bis Zeile 24 einschließlich zu scrollen).

Beispiel: Es soll bis Zeile 22 einschließlich gescrollt werden. SEA erhält also den Wert 23x5=115.

Scroll width register (SW): Muß die Differenz von SEA und SSA zugewiesen bekommen.

Beispiel: Es sei SEA=115 und SSA=5, SW erhält also 115-5=110

Scroll offset register (SOF): Über dieses Register wird das eigentliche Scroll-Kommando gegeben: Scrolling wird ausgeführt, wenn der Inhalt des SOF geändert wird. Erhöhen des Wertes bewirkt hierbei einen Scroll nach oben, verringern einen Scroll nach unten.

Wie beim SSA erfolgt auch hier die wertzuweisung in 5er Schritten. Eine Veränderung des Wertes im SOF um 5 scrollt um eine **HRG-Zeile**. Wird nicht in 5er Schritten geändert, so erfolgt eine Versetzung des Bildschirms zur Seite in Bildschirm-Vierteln.

Die Ansprache der Scroll Kontrollregister geschieht mittels des indirekten Ausgabebefehles OUT (C),A. Die Portadressen sind:

01CFH	SOF (untere 8 Bit)
02CFH	SOF (obere 8 Bit)
03CFH	SW (7 Bit)
04CFH	SSA (7 Bit)
05CFH	SEA (7 Bit)

Im folgenden ein Anwendungsbeispiel: Es soll ein Programm geschrieben werden, das den Bildschirm langsam nach oben scrollt. Das Scrollen soll jedesmal um eine HRG-Zeile erfolgen. SOF muß also jedemal um 5 erhöht werden. Wäre eine Aufwärtsbewegung um 2 HRG-Zeilen erwünscht, so müßte SOF jedesmal um 10 erhöht werden, bei 3 Zeilen um 15 etc. Beim Abwärtsscroll muß SOF entsprechend verringert werden.

Noch ein Problem gilt es zu lösen: SOF kann nicht beliebig erhöht werden, da dann irgendwann ein Überlauf stattfände. Wie also ist dies zu verhindern? Eine Lösung bietet sich an: Da das Scrollen den Bildschirm nicht verändert (Zeilen, die oben den Bildschirm verlassen, erscheinen wieder unten und umgekehrt), könnte man vor dem drohenden Überlauf einmal statt 1 Zeile hoch 199 Zeilen abwärts scrollen. Der SOF ist dann wieder niedrig und ein Unterschied ist auf dem Bildschirm nicht festzustellen.

Das Programm lautet also:

```
        ORG 5000H
        XOR A           ; LD A,0
        OUT (CEH),A     ;320x200 in 4 Farben
LBL1:   CALL WAIT       ; Verzögerung
        CALL SCROLL     ; Scrollen
        JP LBL1         ; Wiederholung
WAIT:   LD HL,1000H     ; Schleifenlänge
LBL2:   DEC HL          ; Schleifenwert um 1 verringern
        LD A,H
        OR L            ; HL=0?
        JP NZ,LBL2      ; Wiederholung
        RET
SCROLL: LD HL,(SOF)     ; HL erhält SOF
        LD DE,5         ; Offset
        ADD HL,DE       ; Scroll um 5 erhöhen
        PUSH HL         ; SOF auf Stack retten
        LD DE,1000      ; Neuer SOF=1000 (1000=200x5)?
        AND A           ; Reset CF
        SBC HL,DE
        POP HL          ; HL erhält SOF; Flags bleiben erhalten
        JR NZ,LBL3      ; Überspringe, wenn SOF-Rücksetzen nicht
                          nötig
        LD HL,0         ; SOF rücksetzen
LBL3:   LD (SOF),HL     ; Neuen SOF ablegen
        LD HL,DATA      ; HL auf Tabellenadresse
        LD BC,06CFH     ; BC auf anzusprechende Ports
        OTDR            ; Datenausgabe an Video-Chip
        RET
SOF:    DEFW 0          ; Scroll offset
        DEFB 125        ; Scroll width
        DEFB 0          ; Scroll start line
        DEFB 125        ; Scroll end line (25x5)
        END
```


Kapitel 4: I/O-Operationen im MZ-700/800 Modus

Der MZ-800 verwendet eine 8255 zur Kommunikation mit Tastatur, Joystick und Kassettenrekorder.

Abhängig vom Modus erfolgt die Datenübergabe zwischen Baustein und Rechner hierbei entweder Memory-mapped oder über Ports. Ebenso, wie im MZ-800 Modus kein Memory-mapped-I/O möglich ist, können die Ports im 700er Modus nicht angesprochen werden.

Folgende Adressen dienen der Kommunikation mit der 8255:

MZ-700 Modus	MZ-800 Modus	Funktion
E000H	Port D0H	Port A (Ausgabe)
E001H	Port D1H	Port B (Eingabe)
E002H	Port D2H	Port C (Ein- und Ausgabe bitweise)

Die Ports im Einzelnen:

Port	Bit Nr.	Aktiv	I/O	Funktion
PA	PA0-3	High	O	Tastaturabfrage
	PA4	Low	O	Joystick 1 abfragen
	PA5	Low	O	Joystick 2 abfragen
	PA7	Low	O	Cursor blink-timer zurücksetzen
PB	PB0-7	Low	I	Tastaturdatenempfang
PC	PC0	Low	O	8255 Tonausgabe blockieren
	PC1	-	O	Datenaufzeichnung Kassette
	PC2	Low	O	Uhr-Interrupt sperren
	PC3	(x)	O	Motorsteuerung
	PC4	High	I	Zeigt, ob Motor an
	PC5	-	I	Daten lesen von Kassette
	PC6	-	I	Cursor-Blink-Zeitgeber
	PC7	-	I	Vertikal Austastung

Die Spalte "Aktiv" besagt, ob das entsprechende Bit gesetzt (High) oder rückgesetzt (Low) ist, um einen Zustand zu beschreiben.

In der Spalte I/O ist beschrieben, ob das entsprechende Bit zur Ausgabe von Daten an die 8255 dient (O=Output) oder zum Lesen von Daten aus der 8255 (I=Input).

Die Bits im einzelnen:

Port A: Bit 0-3 dienen der Tastaturabfrage: Über diese Bits wird angegeben, welche Tastaturspalte abgefragt werden soll (siehe Anhang: Tastaturmatrix). Zugelassen sind die Werte 0-9.

Bit 4 und 5 dienen zur Joystickabfrage. Zur Aktivierung der Abfrage müssen diese Bits rückgesetzt sein. Vergleiche auch Kapitel 5!

Bit 7 dient dem Rücksetzen des Cursor-Blink-Timers

Port B: Bit 0-7 dienen dem Empfang der Tastendaten derjenigen Tastaturspalte, die über Port PA spezifiziert wurde.

Port C: Durch Rücksetzen von Bit 0 kann die Tonausgabe des 8255 blockiert werden.

Über PC1 werden Daten an den Kassettenrekorder ausgegeben (High/Low).

Rücksetzen von PC2 sperrt den 12-Stunden Uhr Interrupt zum Umschalten von AM/PM.

Über PC3 erfolgt die Motorsteuerung. Durch Ausgabe der Signalfolge Low-High-Low an PC3 wird der augenblickliche Motorzustand invertiert. Lief also der Motor, so ist er nun gestoppt und umgekehrt.

PC4 ist gesetzt, wenn der Motor läuft, andernfalls rückgesetzt.

PC5 gibt die von Kassette gelesenen Daten an (Low/High).

PC6 liefert den Wert des Cursor-Blink-Timers zurück. Dieser Wert pendelt periodisch zwischen Low und High.

PC7 gibt an, ob sich der Rechner in der vertikalen Austastlücke befindet, d.h. ob sich die Bildschirmausleselogik am Ende einer Bildschirmzeile befindet. Diese Bit ist im praktischen Betrieb des MZ-800 ohne Bedeutung. Es wurde noch vom MZ-80K übernommen, wo auf diese Austastlücke gewartet werden mußte, um einen bildstörungsfreien Bildschirmzugriff zu ermöglichen. Da beim MZ-700/800 der Bildschirmzugriff gebremst ist, um Bildstörungen zu verhindern, entfällt hier die Abfrage von VBLINK. Eine Bedeutung hat dieses Bit nur noch für die Kompatibilität des MZ-800 zum MZ-80K.

Im folgenden soll die Abfrage der Tastatur an einem Programmbeispiel verdeutlicht werden.

Ein im 700er Modus lauffähiges Programm soll abfragen, ob die CTRL-Taste gedrückt ist. Ist sie gedrückt, so soll ein "C" auf dem Bildschirm angezeigt werden, andernfalls eine Leerstelle. Das Programm soll solange wiederholt werden, bis die Funktionstaste F1 gedrückt wird.

Das Programm lautet:

```
        ORG 5000H
LBL1:   LD A,8          ; CTRL steht in Spalte 8
        LD (E000H),A
        LD A,(E001H)
        Bit 6,A         ; Bit 6 ist CTRL
        LD A,67         ; 67 ist ASCII für "C", Flags bleiben erhalten
        JR Z,LBL2       ; Wenn Taste gedrückt, dann LBL2
        LD A,32         ; 32 ist ASCII für Space
LBL2:   CALL 0012H      ; Monitorroutine: Ausgabe Akku ASCII
        LD A,9          ; F1 ist Spalte 9
        LD (E000H),A    ; Abfrage Spalte 9
        LD A,(E001H)
        BIT 7,A         ; F1 ist Bit 7
        JP Z,EA5EH      ; Wenn gedrückt, Monitor Warmstart
        JP LBL1         ; Ansonsten Wiederholung
```

Ein anderes Beispiel: Eine Schleife, die den Wert des 16-Bit Registerpaares HL ausgibt und dann um eins erhöht, soll solange ausgeführt werden, bis SHIFT+BREAK gedrückt sind.

Zur Abfrage von SHIFT+BREAK soll jedoch nicht die Monitorroutine 001EH benutzt werden (dies wäre zu einfach).

Das Programm lautet:

```
        ORG 5000H
        LD HL,0          ; Erster Zählwert
LBL1:   CALL 03BAH       ; Monitorroutine: Ausgabe HL hexadezimal
        CALL 0006H       ; Monitorroutine: Cursor auf nächste Zeile
        INC HL           ; Zählwert erhöhen
        LD A,8           ; SHIFT+BREAK sind Spalte 8
        LD (E000H),A
        LD A,(E001H)
        AND 81H          ; Alle Bits bis auf Bit 7 und 0 ausblenden
        JP NZ,LBL1       ; ZF ist gesetzt, wenn Ergebnis des AND Null
                         ; ist. Trifft zu, wenn sowohl Bit 7 als auch
                         ; Bit 0 zurückgesetzt sind, d.h. wenn
                         ; SHIFT+BREAK gedrückt sind (Aktiv=0!!)
        JP EA5EH         ; Monitor Warmstart
```

Ein anderes Programmbeispiel: Es soll durch ein Programm erreicht werden, daß der Motor des Kassettenrekorders gestoppt wird, wenn er noch läuft. Da es nicht möglich ist, durch ein Kommando einfach den Kassettenmotor zu stoppen, muß das Programm also abfragen, ob der Motor überhaupt läuft, und wenn ja, den Motorzustand über PC3 invertieren.

Das Programm lautet:

```
        ORG 5000H
        LD A,(E002H)     ; Port C
        BIT 4,A          ; Motor an?
        RET Z            ; Rückkehr, wenn Motor schon aus
        LD HL,E002H      ; HL:=Port C
        RES 3,(HL)       ; Motor Low
        SET 3,(HL)       ; Signal High
        RES 3,(HL)       ; Signal Low, Motor ist nun aus
        RET              ; Rücksprung
```

Da das Programm mit RET-Befehlen abgeschlossen ist, muß es mit CALL aufgerufen werden, also z.B. mit dem G-Befehl vom Monitor aus.

Ein letztes Programmbeispiel zum Thema I/O-Operationen: Ein Programm soll den Rechner auf "Durchzug" schalten, d.h. alle Tonsignale, die der MZ-800 über den Kassettenrekorder einliest, sollen direkt an den Lautsprecher ausgegeben werden.

Dem Programm liegt folgende Idee zugrunde: Es wird PC5 abgefragt, ob Daten am Rekordertonkopf gelesen werden. Wenn ja, wird ein Ton ausgegeben, andernfalls wird die Tonausgabe gestoppt.

Da ein Kassettenrekorder die Tonfolgen ebenfalls digital aufzeichnet, kann mit dieser Methode der Kassetteninhalt vollständig auf den Lautsprecher übertragen werden. Daß das Ergebnis nicht unbedingt so wie bei einer Stereoanlage klingt, hat – neben der Qualität des MZ-800 Lautsprechers – zwei Gründe: Die Ausführungsgeschwindigkeit des Computers ist nicht hoch genug, um alle Signale verlustfrei zu erfassen, so daß ein Teil der Informationen verlorengeht. Zudem registriert der Kassettenrekorder des MZ-800 nur starke, d.h. laute Signale. Die Folge ist, daß leisere Töne verlorengehen.

Zur Tonerzeugung verwendet das Programm der Einfachheit halber die noch vom MZ-80K/MZ-700 her vorhandene Tonausgaberoutine. Ausgegeben wird der höchstmögliche Ton, da bei niedrigen Frequenzen der Verlust an Informationen am höchsten ist.

Das Programm lautet:

```
        ORG 5000H
        LD DE,E002H      ; Adresse PC
LBL1:   LD HL,0100H      ; Information zur Tonhöhe
        LD A,(DE)        ; Lese PC
        BIT 5,A          ; Tonsignal Low/High?
        JR Z,LBL2        ; Nach LBL2, wenn Low
        CALL 02AEH       ; Tonausgabe nach Information in HL
        JP LBL1
LBL2:   CALL 0047H       ; Tonausgabe stoppen
        JP LBL1
```

Dieses Programm läuft solange weiter, bis der RESET-Knopf an der Geräterückseite gedrückt wird.

An dieser Stelle noch eine Erläuterung zu den Tonroutinen: Verwendet werden die im MZ-800 vorhandenen Monitorroutinen zur Tonausgabe.

Es gibt zwei Routinen:

0044H: Ton starten. Die Frequenz wird durch den Inhalt der Adressen 11A1H/11A2H (16-Bit Wert) bestimmt. Dieses Teilungsverhältnis berechnet sich aus 2MHz/nn, wobei nn die gewünschte Frequenz angibt. Das Teilungsverhältnis muß einen Wert größer als 256 haben, andernfalls wird die Tonausgabe gestoppt.

0047H: Tonausgabe stoppen.

Hier wurde die Routine 02AEH verwendet, die wie 0044H arbeitet, jedoch das Teilungsverhältnis in HL statt in 11A1/A2H verlangt.

Kapitel 5: Joystickabfrage

Wie aus dem MZ-800 Handbuch hervorgeht, können mit dem MZ-800 ATARI-kompatible Joysticks verwendet werden.

Der MZ-800 ist jedoch auf den ATARI-Anschluß nur als Option ausgelegt, d.h. daß zum Abfragen von ATARI-kompatiblen Joysticks erst auf eine andere Betriebsart umgeschaltet werden muß.

Dies liegt daran, daß SHARP für den Joystick alle 7 Pins des Joystickanschlusses nutzt, während ATARI nur 5 Pins nutzt. SHARP hat die andere Belegung gewählt, um einen zusätzlichen Eingang für den zweiten Feuerknopf des Joysticks zu schaffen, während bei ATARI nur ein Feuerknopf je Joystick möglich ist.

Die Umschaltung hat nun folgenden Effekt: Normalerweise, d.h. bei SHARP-Joysticks, läuft die Stromversorgung über den mit GND gekennzeichneten Pin. Bei ATARI hingegen muß ein anderer Pin die Stromversorgung liefern, und zwar der mit COMA/COMB bezeichnete. Dieser Pin kann nun wahlweise aktiviert oder ausgeschaltet werden.

Die Steuerung der zusätzlichen Stromversorgung geschieht über den 8255, Port A (MZ-700 Modus: Adresse E000H, MZ-800 Modus: Port D0H), Bit 4 und 5.

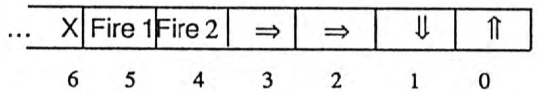
Wird bei einer Ausgabeoperation an diesen 8255-Port das für den Joystick vorgesehene Bit gesetzt, so ist der Pin COMA/COMB (je nach Bit) inaktiv. Ist das Bit rückgesetzt, so ist der Pin aktiv.

Hierbei codiert

Bit 4	Joystick 1	COMA
Bit 5	Joystick 2	COMB.

Abgefragt werden die Joysticks nun über Ports F0H und F1H, und zwar sowohl im 700er als auch im 800er Modus. Hier besteht also zwischen beiden Betriebsarten kein Unterschied.

Die über die Ports F0H/F1H eingelesenen Daten haben folgendes Format:



Ports F0H/F1H

Wie bei den Tastaturdaten bedeutet auch hier ein rückgesetztes Bit, daß die entsprechende Richtung aktiv ist. Ist ein Bit gesetzt, so ist die Richtung nicht aktiviert.

Ein Beispiel:

Ein Programm soll abhängig von der Bewegungsrichtung einen Buchstaben ausdrucken. Bei Druck auf FIRE soll die Ausführung abgebrochen werden.

Das Programm lautet:

```
        ORG 5000H
        XOR A           ; LD A,0
        LD (E000H),A    ; Aktiviere COMA/COMB
LBL1:   IN A,(F0H)      ; Akkumulator erhalt Taten von JOY1
        LD B,A          ; Kopiere in B
        BIT 4,B         ; FIRE?
        JP Z,EA5EH      ; Monitor Warmstart
        LD A,'U'        ; 'Up'
        BIT 0,B         ; Richtung Hoch
        JR Z,LBL2       ; Ausgabe
        LD A,'D'        ; 'Down'
        BIT 1,B
        JR Z,LBL2
        LD A,'L'        ; 'Left'
        BIT 2,B
        JR Z,LBL2
        LD A,'R'        ; 'Right'
        BIT 3,B
        JR Z,LBL2
        XOR A           ; Keine Richtung = Kein Zeichen (Akku = 0)
LBL2:   CALL 0012H      ; Ausgabe Akkumulator ASCII
        JP LBL1         ; Wiederholung
```


Kapitel 6: Quickdisk ROM Unterstützung

Im Gegensatz zur MZ-700 Quickdisk (QD) wird die QD des MZ-800 voll vom ROM unterstützt, d.h. sie kann vom ROM aus wie ein Kassettenrekorder (CMT) angesprochen werden.

Zu diesem Zweck stehen im ROM-Monitor QD-Befehle zur Verfügung, die z.B. Kopieren von CMT auf QD, Kopieren einzelner QD-Dateien etc. ermöglichen, aber leider nicht im Handbuch aufgeführt sind.

Diese Befehle sind:

QC Kopieren einer Datei von einer QD auf eine andere. Es können Dateien beliebiger Kennung (also auch für ROM/BASIC unbekannte wie LOGO/FORTH Dateien) kopiert werden. Kann Dateien bis 48KB kopieren, also auch längere als mit TRANS.

QD Directory. Ist im QD-Handbuch beschrieben. Achtung: Die Directory Routine legt die gelesenen Namen ab Adresse CD90H im Hauptspeicher ab. Daten, die im Bereich CD90H-CFFFH liegen, bleiben also nicht erhalten.

QF QD formatieren

QL Laden von QD. Wie L für CMT.

QS Abspeichern auf QD. Wie S für CMT. Datei erhält die Kennung OBJ.

QX Übertragen einer Datei von Band auf QD. Vorteil gegenüber TRANS: Es können Dateien kopiert werden, die für TRANS zu lang sind.

Ein weiteres Feature bietet das BASIC: LOAD ALL/SAVE ALL. LOAD ALL überträgt den gesamten QD-Inhalt in die RAM-Disk, SAVE ALL überträgt den gesamten RAM-Disk Inhalt auf die QD.

Achtung: Für LOAD ALL muß die RAM-Disk leer sein. Für SAVE ALL benötigt man eine neu formatierte (also leere) QD.

Kapitel 7: Kassettenaufzeichnung

Im MZ-700 Monitor des MZ-800 sind bereits diejenigen Routinen enthalten, die man benötigt, um beliebige Speicherauszüge auf Band abzuspeichern oder von Band zu laden. Es ist jedoch zu beachten, daß die Routinen

1 nur im 700er Modus ablauffähig sind (die Ausgabe erfolgt über Memory-mapped I/O)

2 nur verwendet werden können, wenn die Speicheraufteilung der des MZ-80K entspricht (000H-1000H ROM; 1000H-CFFFH RAM; D000H-FFFFH VRAM/ROM). Die Routinen banken also nicht selbständig die für das Memory-mapped I/O benötigten Speicherstellen ein.

Nun zu der bei den SHARP-Rechnern der MZ-Serie verwendeten Abspeicherungsart: Der MZ-800 zeichnet im sogenannten SHARP PWM-Format (**P**uls**W**eiten**M**odulation) auf, d.h. ein gesetztes Bit wird durch einen langen Impuls dargestellt und ein rückgesetztes durch einen kurzen.

Die Aufzeichnung selbst erfolgt in zwei Blöcken: Dem sogenannten Header, also dem Aufzeichnungskopf, der die dem Programm zugeordneten Daten wie Anfangsadresse, Autostartadresse, Länge, Namen, Programmart usw. enthält, und den eigentlichen Daten.

Der Header hat immer eine Länge von 128 Bytes und wird durch eine eigene Routine aufgezeichnet bzw. gelesen.

Diese Zweiteilung bietet den Vorteil, daß man nach dem Lesen des Kopfes die gelesenen Informationen vom Ladeprogramm aus noch verändern kann, also z.B. ein ab Adresse 1200H adressiertes Programm ab Adresse 4000H laden kann, oder z.B. ein ab 3000H im Speicher stehendes Programm so aufzeichnet, daß es später immer ab 5000H geladen wird.

Folgende Adressen des Monitorbereiches 1000H - 1200H sind für den Header von Bedeutung:

10F0H: Dateikennung. Gibt an, um was für ein Programm es sich handelt:
01: Maschinenprogramm (OBJ)
02: BASIC-Programm (BTX)
03: BASIC-Datei sequentiell (BSD)...

10F1H: Dateiname. Maximal 16 Zeichen plus 0DH ((CR-Code) als Abschluß)

1102H: Dateilänge. Berechnet sich aus ENDADRESSE - STARTADRESSE +1. Die 1 muß addiert werden, da ein Programm von 1200-1201H nicht ein sondern zwei Byte lang ist (Grenzen werden ja einschließlich angegeben!!)

1104H: Startadresse des Programms

1106H: Autostartadresse. Ist diejenige Adresse, an der der Monitor ein Maschinenprogramm nach dem Laden startet (z.B. BASIC)

Folgende Routinen dienen nun zum Laden/Abspeichern eines Programmes:

0021H: Header saven. Schreibt Informationskopf auf Band. Es kommt die Anweisung "! RECORD.PLAY", nach Beginn der Aufzeichnung erscheint die Meldung "WRITING..."

0024H: Programm abspeichern. Der Programmblock wird auf Band geschrieben. Die Adresse, ab der abgespeichert werden soll und die Programmlänge werden den Speicherstellen 1104H-1106H entnommen.

0027H: Header laden. Der nächste Informationsblock wird geladen. Diese Routine kann an jeder beliebigen Stelle auf dem Band aufgerufen werden, da sie sich den nächsten Anfang eines Informationsblockes selbst sucht. Die gelesenen Daten liegen wieder in den Adressen, aus denen sie auch mit 0021H abgesaved wurden.

002AH: Datenblock lesen. Um den auf einen Informationsblock folgenden Datenblock zu lesen, bezieht sich die Routine wieder auf die im Header gelesenen Daten. Der Datenblock wird an der Adresse abgelegt, die in 1104H steht. Es werden soviele Daten gelesen, wie 1102H angibt. Stimmt die im Header angegebene Länge nicht mit der tatsächlich auf dem Band befindlichen überein, so wird ein Fehler erkannt.

Alle oben genannten Routinen liefern nach der Ausführung zurück, ob ein Fehler aufgetreten ist oder SHIFT/BREAK gedrückt wurde.

Trifft eine der Möglichkeiten zu, d.h. die Ausführung der Routine wurde aus einem der beiden Gründe vorzeitig verlassen, so ist das Carry-Flag CF gesetzt. Zudem gibt der AKKU an, ob es sich un eine Unterbrechung durch Fehler (Akku=2) oder durch Drücken von SHIFT/BREAK gehandelt hat (Akku=1).

Um diese recht theoretischen Ausführungen nun anschaulicher zu machen, einige Beispiele:

Beispiel 1: Ein Programm soll den Speicherbereich 1200H-2000H unter der Kennung eines BASIC-Programmes (BTX) abspeichern.

```
ORG 5000H
LD A,3          ; BTX-Code
LD (10F0H),A
LD HL,1200H     ; Startadresse
LD (1104H),HL
LD HL,0E01H     ; Länge: 1200H bis 2000H+1=0E01H
LD (1102H),HL
LD HL,000H      ; Autostart
LD (1106H),HL
LD A,0DH        ; CR-Code
LD (10F1H),A    ; Name ist "" (Leerstring): 1. Zeichen ist Endcode
CALL 0021H      ; WRINF (Write Information)
RET C           ; ERROR
CALL 0024H      ; WRD (Write Data)
RET
```

Wie an den Rücksprungadressen erkenntlich, soll das Programm vom Monitor aus mit dem G-Befehl als Unterprogramm aufgerufen werden.

Beispiel 2: Ein Programm soll ein beliebiges anderes unabhängig von dessen Startadresse kopieren können (also auch, wenn Kopierprogramm und das zu kopierende Programm sich überschneiden). Ein solches Programm muß folgende Probleme lösen: Das Programm muß so in den Speicher geladen werden, daß es sich mit dem Kopierprogramm nicht überschneidet. Dann muß der Header wieder restauriert werden, damit bei einem späteren Ladeprozeß wieder an die Originaladresse geladen wird. Dann wird nach Abspeichern des Headers der Datenblock wieder abgespeichert:

```
        ORG 1200H       ; Programmstart 1200H, damit möglichst viel
                          zusammenhängender Speicher zum Laden frei
                          ist
        LD A,16H        ; Sondercode Bildschirm löschen
        CALL 0012H      ; Zeichen im Akku ausgeben (CLS)
        CALL 0027H      ; Header lesen
        LD DE,MSG1      ; "LOADING" 0027H/002AH geben kein LOADING
                          aus!!!
        CALL 0015H      ; String ab DE ausdrucken
        LD DE,10F1H     ; Dateiname
        CALL 0015H      ; Dateinamen ausdrucken. Gesamt: "LOADING ..."
        CALL 0006H      ; Neue Bildschirmzeile
        LD HL,(1104H)   ; Ladeadresse aus Header in HL
        LD (STADD),HL   ; Ladeadresse retten
        LD HL,NADD      ; HL erhält neue Ladeadresse (PRG-Ende)
        LD (1104H),HL   ; Neue Ladeadresse ablegen
        CALL 002AH      ; Daten lesen (RDD=Read Data)
        LD DE,MSG2      ; String "Change Cass, then press any key"
        CALL 0015H      ; String ausgeben
LBL1:   CALL 001BH      ; GETKEY: Akku erhält gedrückte Taste
        OR A            ; Akku=0? (kürzer als CP 0)
        JR Z,LBL1       ; ZF: Keine Taste gedrückt
        LD HL,(STADD)   ; Alte Ladeadresse in HL
        LD (1104H),HL   ; Header restaurieren
        CALL 0006H      ; NL (New Line)
        CALL 0006H
        CALL 0021H      ; WRINF: Header schreiben
        LD HL,NADD      ; Adresse, ab der Datenblock steht
        LD (1104H),HL   ; Header für WRD präparieren
        CALL 0024H      ; WRD: Write Data
        RET
MSG1:   DEFM "LOADING"
        DEFB 0DH
MSG2:   DEFM "Change Cass, then press any key"
        DEFB 0DH
STADD:  DEFW 0          ; Zwischenspeicher für Startadresse
NADD:                   ; Neue Ladeadresse
```

Wie man sieht, geht das Programm von der recht optimistischen Annahme aus, daß beim Lesen/Schreiben kein Fehler auftritt (es wird nicht auf gesetztes CF nach Aufruf der Bandroutinen abgetastet). Wie alle anderen Beispiele ist auch dieses mit einem RET-Befehl abgeschlossen und wird vom Monitor aus mit dem G-Befehl aufgerufen.

Eine Anmerkung noch zum BOOT-Ladesystem des IPL: Programme/ Sprachen, die vom IPL geladen werden, werden **nicht** ab ihrer Startadresse geladen, sondern ab 1200H und dann zu ihrer Startadresse verschoben.

Dieses Verfahren bietet den Vorteil, daß Programme, die ab 000H den Speicher belegen, auf dem Header direkt ab 000H adressiert werden können, da der IPL vor dem Verschieben dem Speicherbereich 000H-1000H RAM zuweist. Dieses Verfahren nutzt z.B. BASIC, dessen Startadresse mit 000H im Header steht.

Der Nachteil dieses Verfahrens liegt darin, daß von jedem IPL/Monitor Ladevorgang der Bereich ab 1200H betroffen ist, so daß man z.B. keinen Disassembler laden kann, um den Bereich ab 1200H zu untersuchen, da der Speicherbereich, der untersucht werden soll, durch den Ladeprozeß verändert wird.

Umgehen läßt sich dieser Nachteil auf zwei Arten: Entweder indem man in den Monitor springt und ein eigenes Ladeprogramm eingibt, oder aber eleganter, indem man im Monitor J00AD eingibt, um in den MZ-700 Monitor des MZ-800 zu springen, der diese Ladeverschiebung nicht macht, sondern normal ab der Startadresse lädt.

Kapitel 8: QD-Aufzeichnungsroutinen

Beim MZ-800 wird die QD ähnlich wie der Kassettenrekorder voll ROM unterstützt. Die QD-Routinen bieten jedoch gegenübe den CMT-Routinen den Vorteil, daß sie sowohl im 700er als auch im 800er Modus verwendungsfähig sind.

Die Datenübergabe für die Routinen erfolgt ähnlich wie bei Kassettenbetrieb, jedoch in anderen Adressen:

10F0H: Dateikennung (OBJ/RB/GR/BTX/BSD etc.)
10F1H: Dateiname 16 Zeichen & 0DH
1104H: Dateilänge
1106H: Datei-Anfangsadresse (TOP-Adresse)
1108H: AUTO-Startadresse
1130H: Modussteuerung für Routine
1132H: Anfangsadresse (Doppel von 1106H)
1234H: Dateilänge (Doppel von 1104H)
1136H: Anfangsadresse
1138H: Dateilänge

Wie schon auf den ersten Blick auffällt, sind einige Daten dopelt oder gar dreifach im Speicher vorhanden. Dies liegt am Aufbau der Lade/Save-Routinen, die die Daten teilweise in verschiedenen Speicherstellen verlangen.

Folgende Routinen werden für die QD-Ansprache benötigt (aufgesplittet nach LOAD/SAVE/DIR/FORMAT):

1 **Laden von Programmen:**
EB13H: QD angeschlossen? ZF:QD ist angeschlossen
EEECH: QD-Arbeitsbereich initialisieren
EF27H: QD ready? CF:not ready
F25FH: Initialisieren für das Laden
EEF7H: RDINF
E010H: RDD

2 **Abspeichern von Programmen:**
EB13H: QD angeschlossen
EEECH: QD-Arbeitsbereich initialisieren
EFE1H: Initialisieren für Schreiben, QD ready? CF:not ready
EF9FH: WRINF
E010H: WRD

3 **Formatieren einer QD:**
EB13H: QD angeschlossen?
EEECH: Init Arbeitsbereich
EFE1H: QD ready?
E010H: Formatieren

4 **Fehlerbearbeitungsroutine**
E010H: Motor aus
F25FH: Init

5 **DIRectory**
EB13H: QD angeschlossen?
EEECH: QD-Arbeitsbereich initialisieren
EF27H: QD ready?
F25FH: QD INIT
E010H: Read Header/Motor off

Wie man sieht, wird die Routine E010H mehrfach in verschiedenen Zusammenhängen verwendet. Dies liegt daran, daß die Routine über die Adressen der Modussteuerung 1130H gesteuert wird.

Unabhängig vom gewünschten Vorgang muß in jedem Fall vor Verwendung der QD die Routine EEECH zur Initialisierung des QD-Arbeitsbereiches aufgerufen werden.

Folgende Dateikennungen werden vom ROM erkannt/verwendet:

01H:	OBJ	(Maschinenprogramm)
02H:	BTX	(BASIC Text)
03H:	BSD	(Sequentielle BASIC Datei)
04H:	BRD	(Random-Datei, bei QD nicht möglich)
05H:	RB	(Relocatible Binary File des SHARP Assemblers)
07H:	LIB	(Library File)
0AH:	SYS	(Systemfile)
0BH:	GR	(Grafikfile)
sonst:	???	(unbekannt)

Im folgenden werden nun die zum Laden, Saven etc. nötigen Vorgänge beschrieben und in einem Beispiel dargestellt:

1 **Programm laden:**
Das Laden eines Programmes läuft ähnlich dem Laden von Kassette ab: Zunächst wird der Programmkopf gelesen und dann der zugehörige Datenblock.

Beispiel: Laden des Programmes "XYZ"

```
        CALL EEECH      ; INIT
        CALL EF27H      ; QD ready? CF:ERROR
        RET C           ; QD not ready
        LD DE,11A3H     ; Adresse ab der der Name abgelegt
                        ; werden muß
        LD HL,NAME      ; "XYZ"
        LD BC,4         ; Vier Buchstaben
        LDIR            ; Übertragen
        CALL F25FH      ; INIT für Lesen
        CALL EEF7H      ; RDINF
        LD HL,(1106H)   ; Startadresse
        LD (1132H),HL   ; Kopieren für Routine E010H
        LD HL,(1104H)   ; Modus
        LD (1130H),HL
        CALL E010H      ; RDD   CF:ERROR
        RET
NAME:   DEFM "XYZ"
        DEFB 0DH
```

Wie man sieht, geht auch dieses Programm von der optimistischen Annahme aus, daß kein Ladefehler auftritt.

Alle QD-Routinen liefern Fehler durch Setzen des Carry Flags und die Nummer des Fehlers im Akku zurück.

Folgende Fehler sind möglich:

28H:	File not found
39H:	QD: Bad disk error
2EH:	QD: Write protect error
32H:	QD: Not ready
35H:	QD: No file space error
36H:	QD: Unformat error
2AH:	Already exist error
33H:	QD: Too many files error
00H:	Break
sonst:	QD: Hardware error

Nach Auftreten eines Fehlers muß der Motor der QD abgeschaltet werden, da dies nicht automatisch geschieht.

2 Fehlerbearbeitungsroutine:
Zur Fehlerbearbeitung muß erst der nachfolgende Programmteil aufgerufen werden, bevor der Fehlertext auf dem Bildschirm ausgegeben wird:

```
Error:  PUSH AF          ; Fehlernummer retten
        LD A,06H         ; Modus: Motor off
        CALL E010H       ; Motor off
        CALL F25FH       ; QD Init
        POP AF           ; Fehlernummer restaurieren
... Fehlerausgabe
```

3 QD Formatieren:
Die Formatierungsroutine besteht nur aus einem Aufruf der Routine E010H:

```
FORMAT: CALL EEECH       ; Arbeitsbereich initialisieren
        CALL EFE1H       ; QD ready?
        RET C            ; CF: Not ready
        LD A,02H         ; Modus: FORMAT
        LD (1130H),A     ; Modus ablegen
        CALL E010H       ; Kommando ausführen, ERROR: CF=1,
                         ; Akku=Nummer
        RET
```

4 Abspeichern auf QD:
Für das Abspeichern auf QD müssen folgende Werte übergeben werden:

11A3H/10F1H:	Filename 16 Zeichen & 0DH
1106H:	Startadresse (TOP)
1104H:	Länge der Datei
1108H:	Autostartadresse (Exc.)
10F0H:	Dateikennung

Beispiel: Es soll der Speicherauszug von 1200H bis 2000H unter der Kennung SYS (0AH) auf QD abgespeichert werden.

```
        ORG 3000H
        CALL EB13H       ; QD angeschlossen?
        RET NZ           ; ZF: QD ist angeschlossen
        CALL EEECH       ; QD Init
        CALL EFE1H       ; QD ready?
```



```
        RET C            ; Not ready
        LD HL,NAME       ; "SAVE.DEMO"
        LD DE,10F1H      ; Bestimmungsadresse
        LD BC,000AH      ; 10 Zeichen (9+0DH)
        LDIR             ; Übertragen
        LD HL,1200H      ; Dateianfang
        LD (1106H),HL    ; Ablegen
        LD HL,0E00H      ; 2000H-1200H+1
        LD (1104H),HL    ; Ablegen
        LD HL,EA5EH      ; Monitor Warmstart als Autostart =
                         ; kein Autostart
        LD (1108H),HL    ; Ablegen
        LD A,0AH         ; Kennung SYS
        LD (10F0H),A
        CALL EF9FH       ; WRINF möglich? (File space, not too
                         ; many etc.)
        RET C            ; RET if ERROR
        LD HL,(1106H)    ; TOP
        LD (1136H),HL    ; Kopieren
        LD HL,0040H      ; Länge 64 Byte = Länge Header
        LD (1134H),HL    ; Ablegen
        LD HL,10F0H      ; Startadresse Header
        LD (1132H),HL    ; Ablegen
        LD HL,0404H      ; Modus: WRINF+WRD
        LD (1130H),HL    ; Ablegen
        LD HL,(1104H)    ; Dateilänge
        LD (1138H),HL    ; Ablegen
        CALL E010H       ; Write, CF:Error, Akku = Fehlernummer
        RET
Name:   DEFM "SAVE.DEMO" ; Dateiname
        DEFB 0DH
```

5 Directory der QD:
Das Erstellen eines Directorys ist im Vergleich zum Laden/Saven relativ kompliziert. Es beruht darauf, daß nacheinander alle auf der QD befindlichen Header gelesen und im Speicher abgelegt werden. Zudem wird in einem Register mitgezählt, wieviele Header vorhanden waren. Eine Ausgaberoutine wertet die im Speicher abgelegten Daten dann aus. Von den Headern wird auf jeden Fall nicht der gesamte Bereich von 64 Byte benötigt, sondern je nach Wunsch nur die ersten 18 Byte (Dateiname und Kennung) oder die ersten 26 Byte, wenn zudem noch Anfangsadresse, Länge, Endadresse und Autostart ausgegeben werden sollen.

Im folgenden nun eine Directoryroutine, die zusätzlich zu Namen und Kennung auch noch TOP/END/AUTO mit ausgibt.

Die Ausgabe soll nach dem Muster

OBJ Name 0000 1000 0000

erfolgen. (Hier: Maschinenprogramm "Name" von 000H bis 1000H AUTO 0000H)

Das Programm lautet:

```
        ORG 1200H
        CALL EB13H      ; QD angeschlossen?
        RET NZ          ; Keine QD vorhanden
        CALL EEECH      ; QD Init
        CALL EF27H      ; QD ready?
        RET C           ; Not ready
        CALL F25FH      ; QD Init
        LD B,0          ; Counter: Anzahl gelesener Header
        LD HL,C000H     ; Adresse ab der die Header abgelegt
                          werden sollen
LBL1:   LD (1132H),HL   ; Ablege Header Ladeadresse
        PUSH HL         ; Rette Adressenzeiger
        LD HL,0003H     ; Modus
        LD (1130H),HL   ; Ablegen
        LD HL,0040H     ; 0040H=64: Länge Header
        LD (1134H),HL   ; Ablegen
        PUSH BC         ; Rette Counter
        CALL E010H      ; RDINF  CF:ERROR
        POP BC          ; Counter restaurieren
        POP HL          ; Adressencounter restaurieren
        JR C,LBL2       ; Beim Headerlesen Fehler aufgetreten
        INC B           ; Counter um 1 erhöhen: Wieder einen
                          Header richtig gelesen
        LD DE,001AH     ; Benötigte Länge Header: 26 Byte
        ADD HL,DE       ; Adresse für nächsten Header
                          berechnen
        JP LBL1         ; Headerlesevorgang wiederholen
LBL2:   CP 28H          ; Aufgetretener Fehler = File not found?
        RET NZ          ; Wenn nicht: Lesefehler
        LD A,06H        ; Motor off
        LD (1130H),A    ; Modus ablegen
```



```
        PUSH BC         ; Rette Counter
        CALL E010H      ; Motor off
        POP BC          ; Counter restaurieren
        XOR A           ; Akku = 0
        CP B            ; Kein Header gelesen (QD leer?)
        RET NC          ; Rück wenn nicht B größer 0
        LD DE,TXDIR     ; Text "DIRECTORY OF QD:"
        CALL 0006H      ; Neue Zeile
        CALL 0015H      ; Textausgabe
        CALL 0006H      ; Neue Zeile
        CALL 0006H

        LD DE,C000H     ; Adresse, ab der die Header stehen

LBL3:   LD A,(DE)       ; Dateikennung (OBJ/BTX/BSD/...)
        INC DE          ; DE auf Dateinamen
        DEC A           ; Kennung um 1 verringern. OBJ nun
                          00H usw.
        CP 3            ; Kennung größer BSD?
        JR C,LBL4       ; Nicht größer
        LD A,3

LBL4:   ADD A,A         ; A=Ax2
        ADD A,A         ; A nun mit 4 multipliziert. Sinn:
                          Kennungskürzel sind je vier Byte lang
                          (inkl. CR)
        PUSH BC         ; Rette Counter
        LD B,0          ; Highbyte BC=0
        LD C,A          ; Nun BC=A
        LD HL,Kenn      ; Anfang Kennungstabelle
        ADD HL,BC       ; HL steht auf Kennungskürzel
        EX DE,HL        ; Nun DE auf Text
        CALL 000CH      ; Ausgabe Leerstelle " "
        CALL 0015H      ; Kennungsausgabe
        CALL 000CH      ; Leerstelle
        CALL 000CH
        EX DE,HL        ; DE steht auf Dateiname
        CALL 0015H      ; Ausgabe Dateiname
        LD A,18H        ; 18H=24, neue X-Koordinate Cursor
        LD (1171H),A    ; Neue X-Koordinate ablegen
        LD HL,0013H     ; Abstand bis Byte-Länge der Datei
        ADD HL,DE       ; HL indiziert nun Dateilänge
        LD C,(HL)       ; Low-Byte
```

```
        INC HL
        LD B,(HL)       ; High-Byte, BC enthält nun Länge
        INC HL          ; HL steht nun auf Dateibeginn (TOP)
        LD E,(HL)       ; Low-Byte laden
        INC HL
        LD D,(HL)       ; High-Byte laden, DE enthält nun TOP
        INC HL          ; HL steht nun auf AUTOstartadresse
        EX DE,HL        ; Nun enthält HL TOP, DE indiziert
                          AUTO
        CALL 03BAH      ; Ausgabe TOP hexadezimal
                          (Monitorroutine)
        CALL 000CH      ; Leerstelle
        ADD HL,BC       ; HL enthält nun TOP + Länge =
                          Endadresse + 1
        DEC HL          ; HL enthält nun Endadresse
        CALL 03BAH      ; Ausgabe Ende hexadezimal
        CALL 000CH      ; Leerstelle
        EX DE,HL        ; HL indiziert AUTO
        LD E,(HL)       ; Low-Byte laden
        INC HL
        LD D,(HL)       ; High-Byte, DE enthält nun
                          AUTOstartadresse
        INC HL          ; HL steht auf nächster Kennung
        EX DE,HL        ; Vertausche DE,HL. HL nun AUTO, ED
                          indiziert nächste Kennung
        CALL 03BAH      ; Ausgabe AUTO
        CALL 0006H      ; Nächste Zeile
        POP BC          ; Counter wieder vom Stack holen
        RET

TXDIR:  DEFM "DIRECTORY OF QD:"
        DEFB 0DH

Kenn:   DEFM "OBJ"
        DEFB 0DH
        DEFM "BTX"
        DEFB 0DH
        DEFM "BSD"
        DEFB 0DH
        DEFM "???"
        DEFB 0DH
        END
```


Noch einige allgemeine Anmerkungen zu den QD-Routinen: Die QD-Routinen lassen sich sowohl im 700er als auch im 800er Modus nutzen.

Voraussetzung für die Verwendung der Routinen ist jedoch, daß sich im Bereich 1000H-1200H freies RAM als QD-Arbeitsbereich befindet (es ist also z.B. nicht möglich, bei eingebanktem CG-ROM auf QD zu schreiben) und daß im Bereich E000H-FFFFH das ROM, das die Routinen enthält, eingebankt ist (logisch, aber...).

Mit der QD ist es nicht wie z.B. beim Kassettenrekorder möglich, direkt aus dem VRAM/HRG-RAM auf Datenträger zu schreiben bzw. direkt von QD in den Bildspeicher einzulesen, da – wie bekannt – beim MZ-800 der Bildschirmzugriff gebremst ist, um Bildstörungen zu vermeiden und der Rechner somit beim Einlesen/Schreiben nicht schnell genug zugreifen kann. Die Folge sind Schreib- oder Lesefehler.

Kapitel 9: Der MZ-700 Monitor

Wie schon einmal in früheren Ausführungen angesprochen, besitzt der MZ-800 im Bereich 0000H-0FFFH bis auf geringe Anpassungen exakt das Monitor-ROM, das auch der MZ-700 besaß.

Änderungen sind nur in zwei Bereichen vorgenommen worden: Die Kassettenroutinen wurden an die höhere Geschwindigkeit des MZ-800 angepaßt, um sicherzustellen, daß die Aufzeichnungsgeschwindigkeit auch beim MZ-800 1.200 Bauc beträgt und somit zum MZ-700 kompatibel ist. Außerdem wurden die ersten drei Byte in einen JP E800H geändert, so daß beim MZ-800 nach dem Einschalten nach JP 0000H bzw. RESET in den MZ-800 Monitor gesprungen wird. Der MZ-700 Monitor ist also nicht aktiv, sondern dient nur zur Aufrechterhaltung der Kompatibilität zum MZ-700.

Da der MZ-700 Monitor jedoch vollständig vorhanden ist, kann er auch im MZ-800 angesprungen und genutzt werden.

Ein Kaltstart des MZ-700 Monitor über JP 004AH (MZ-700 ROM Kaltstart) ist jedoch **nicht** möglich, da der MZ-700 Monitor in einer noch vom MZ-700 her stammenden Routine geststellt, daß sich im Bereich E800H-FFFFH ROM befindet und dann nach E800H springt (beim MZ-700 diente diese Routine zur Erkennung des Floppy/QD-Betriebssystems, das bei Erwerb eines dieser Geräte nachträglich eingesetzt wurde).

Starten läßt sich der MZ-700 Monitor somit erst nach dem ROM-Check, also an Adresse 00ADH.

Starten Sie einmal den MZ-800 Monitor Ihres MZ-800 und geben dann J00AD ein. Sie befinden sich nun im MZ-700 Monitor.

Im MZ-700 Monitor stehen folgende Befehle zur Verfügung:

Jhhhh	Sprung an die Adresse hhhh. Wie im MZ-800 Monitor.
L	Programm von Kassette laden. Im Gegensatz zum MZ-800 Monitor wird jedoch nicht ab 1200H geladen und dann an die tatsächliche Adresse verschoben, sondern direkt an die im Header stehende TOP Adresse. Nützlich, wenn der Bereich ab 1200H durch einen Ladevorgang nicht betroffen sein soll. Wer nicht in den MZ-700 Monitor springen will, um diesen Ladevorteil zu nutzen, kann die Laderoutine auch vom MZ-800 Monitor aus mit J0111 starten.

F Booten von der Floppy aus.

B Tastaturbeep einschalten. Nach B ertönt bei jeder Tastenbetätigung ein Piepton. Nochmalige Eingabe von B schaltet diesen Ton wieder ab. Da der MZ-700 und MZ-800 Monitor dieselbe Tastaturroutine benutzen, läßt sich dieser Tastaturbeep in den MZ-800 Monitor mit "zurücknehmen", wenn man den MZ-800 Monitor mit JEA5E an seiner Warmstartadresse startet. Bei Benutzung der Tastatureingaberoutine 0003H läßt sich dieser Kontrollbeep immer einschalten, indem man in 119DH den Wert 00H ablegt. Ablegen von FFH schaltet den Ton wieder ab.

Hat dieselbe Wirkung wie das Drücken von CTRL+RESET gleichzeitig: Der gesamte Speicher von 0000H-FFFFH wird freies RAM und PC erhält den Wert 0000H (JP 0000H).

Unterschied zu CTRL+RESET: Der Rechner bleibt unabhängig von der Stellung des Systemschalters immer im MZ-700 Modus (es sei denn, das angesprungene Programm schaltet selbst durch Ansprechen des DMR den Modus um).

P sssssss Dient der Ausgabe eines Strings an einen Drucker. Folgende Formate sind möglich:

P string	Druckt das Wart string (inkl. CR)
P&T	Testkommando für SHARP Plotter
P&C	Plotter: Farbwechsel
P&S	Plotter: Textmodus 80 Zeichen/Zeile
P&L	Plotter: Textmodus 40 Zeichen/Zeile
P&G	Plotter : Grafikmodus

Die letzten fünf Kommandos funktionieren nur in Verbindung mit dem SHARP Plotter. Befindet sich der Plotter im Grafikmodus, so können mit P string Kommandos zum Zeichnen übergeben werden. Beispiel: PM0,0 ist das Kommando MOVE 0,0. Näheres siehe Plotterhandbuch.

Mhhhh Wie im MZ-800 Monitor.

Saaaabbbbcccc Abspeichern des Programmes von aaaa bis bbbb einschließlich unter Autostart cccc. Namenseingabe wird einzeln abgefragt (FILENAME ?). Bei Eingabe des Namens muß beachtet werden, daß aufgrund eines Fehlers im MZ-700 ROM das erste Zeichen der Eingabe verlorengeht. Es empfiehlt sich also, zunächst eine Leerstelle und dann erst den Namen zu schreiben.

V Vergleichen der Bandaufzeichnung mit dem tatsächlichen Speicherinhalt. Wie im MZ-800 Monitor.

Daaaabbbb Memorydump von aaaa bis bbbb. Auch nur Daaaa ist möglich. Wie im MZ-800 Monitor.

Nun noch einige Monitorstartadressen:

0000H MZ-800 Monitor Kaltstart. PCG-, Peripherie- und Arbeitsbereichinitialisierung.

00ADH MZ-700 Monitor Warmstart. Keinerlei Initialisierungen.

0111H Ansprungadresse MZ-700 Bandroutine. Lädt nächstes gefundenes Programm von Band an die tatsächliche Startadresse.

E800H siehe 000H

E906H MZ-800 IPL Warmstart: Keinerlei Initialisierungen. IPL meldet sich mit Wahlmenü. **Kein** selbständiges Booten bei angeschlossener QD/FD!!!

EA5EH MZ-800 Monitor Warmstart. Keinerlei Initialisierungen.

Wer Interesse daran hat, das Innenleben seines MZ-800 genauer kennenzulernen, dem empfehle ich die Anschaffung eines MZ-700 und MZ-800 Monitor-Disassemblerlistings. Ein Listing des MZ-700 Monitors befand sich im Handbuch des MZ-700. Deshalb bietet es sich an für alle, die noch Besitzer eines MZ-700 sind, sich an dieses Listing zu halten. Für weitere Listings siehe Anhang D.

Kapitel 10: Steigerung der Aufzeichnungsrate des Kassettenrekorders

Wie Sie schon aus früheren Aufzeichnungen wissen, enthält der Monitor des MZ-800 Kassettenaufzeichnungsroutinen, die im sogenannten SHARP PWM-Verfahren aufzeichnen.

PWM steht – wie schon erläutert – für **P**ulsweiten**m**odulation und bedeutet, daß ein kurzer Impuls auf dem Band ein rückgesetztes Bit repräsentiert und ein langer ein gesetztes. Sämtliche Kassettenroutinen – unabhängig davon, ob sie nun der Aufzeichnung oder Abspeicherung dienen – nutzen nun dieselben Verzögerungsroutinen, um einen Impuls zu schreiben oder zu erkennen.

Die Idee, die der einfachsten Methode, die Aaufzeichnungsgeschwindigkeit zu steigern, zugrunde liegt, ist nun, diese drei grundlegenden Zeischleifen zu kürzen.

Hierbei muß jedoch beachtet werden, daß die Zeitschleifen nicht beliebig gekürzt werden dürfen, sondern daß die **Kürzung aller drei Schleifen im selben Verhältnis** erfolgen muß. Soll die Aufzeichnungsrate also z.B. verdoppelt werden, so müssen alle drei Schleifen halbiert werden.

Alle Zeitschleifen sind nach diesem Muster aufgebaut:

```
DLY:    LD A,Verz       ; Schleifenwert
LBL1:   DEC A           ; verringern
        JP NZ,LBL1      ; Wiederholung
```

Folgende drei Routinen sind für die Aufzeichnungsrate entscheidend:

```
0759H:  DLY1  Schleifenwert=15H  für 4.000 Baud: 07H
0760H:  DLY2  Schleifenwert=13H  für 4.000 Baud: 09H
09A9H:  DLY4  Schleifenwert=59H  für 4.000 Baud: 1EH
```

Die unter 4.000 Baud angegebenen Werte entsprechen - wie sich unschwer erkennen läßt - nicht ganz der oben angegebenen Berechnungsmethode, da hier im Gegensatz zu den Originalwerten DLY2 größer als DLY1 ist. Dies kommt daher, daß der Wert für 4.000 Baud in der "normalen" Form nicht differenziert genug ist und häufig Lesefehler verursacht. Die Erfahrung hat gezeigt, daß die oben angegebenen Werte besser arbeiten.

```
        LD HL,(1104H)     ; Startadresse (TOP)
        LD (ADDR),HL      ; Retten
        LD HL,Nadd        ; Neue Ladeadresse = Programmende
        LD (1104H),HL     ; Ablegen
        CALL 002AH        ; RDD
        JP C,ERROR        ; Fehler?
        LD DE,MSGSW       ; "4000 bps now, change Cass, then
                            press any key"
        CALL 0015H        ; Ausgabe
        CALL 09B3H        ; Warten auf Eingabe mit blinkendem
                            Cursor, Ergebnis steht in Acc. im
                            Anzeigecode
        CALL 0006H        ; Neue Zeile
        CALL 0006H        ; Leerzeile
        OUT (0E0H),A      ; 0000H-1000H: RAM = 4000 Bps
                            Monitor
        LD HL,(ADDR)      ; Originalstartadresse TOP
        LD (1104H),HL     ; Ablegen
        CALL 0021H        ; WRINF
        JP C,ERROR
        OUT (0E4EH),A     ; 0000H-1000H: ROM etc.
ERROR:  LD DE,MSGER       ; "ERROR!!!"
        CALL 0006H        ; Neue Zeile
        CALL 0015H        ; Ausgabe
        JP EA5EH          ; MZ-800 Monitor

STMSG:  DEFM "1200 to 4000 bps converter"
        DEFB 0DH
MSGLD:  DEFM "Reading "
        DEFB 0DH
MSGSW:  DEFM "4000 bps now, change Cass, then press any key"
        DEFB 0DH
MSGER:  DEFM "ERROR!!!"
        DEFB 0DH
ADDR:   DEFW 0
Nadd:
```


Anhang A: Erläuterung der wichtigsten Fachbegriffe

Bankswitching bezeichnet ein Verfahren, mit dem es ermöglicht wird, daß der Prozessor auf einen größeren Speicherbereich zugreifen kann, als sein Adreßbereich umfaßt. Wird dadurch realisiert, daß durch bestimmte Befehle der Inhalt bestimmter Speicherbereiche ausgetauscht werden kann und der Prozessor somit unter derselben Hauptspeicheradresse verschiedene Speicher anspricht. Beim MZ-800 durch Port-Ansprechbefehle realisiert.

CG-RAM Speicher, der die augenblicklich Matrix jedes auf dem Bildschirm darstellbaren Zeichens enthält. Existiert beim MZ-800 nur im 700er Modus und muß mittels » Bankswitching in den Hauptspeicher eingeblendet werden.

CG-ROM Enthält die Zeichendefinitionen, die dem » CG-RAM bei einem Rechnerkaltstart zugewiesen werden. Wird immer zusammen mit dem CG-RAM in den Hauptspeicher ein- oder ausgeblendet.

CMT Abkürzung für Kassettenlaufwerk.

CPU Von englisch Central Processing Unit = Prozessor (beim MZ-800 eine Z80).

Datei Jede Art Programm- oder Datenfragment, das auf QD oder Kassette abgespeichert wird.

FORTH Äußerst schnelle, vielseitige Programmiersprache. Durch die ungewöhnliche Rechenlogik UPN für Einsteiger nicht leicht zu erlernen.

Header Kopfteil jeder Aufzeichnung auf QD oder CMT. Enthält die die nachfolgende » Datei beschreibenden Daten wie Anfangsadresse, Länge in Byte, Autostartadresse, Dateiart und -name usw.

HiRes » HRG

HRG Von englisch High Resolution Graphic (auch HiRes genannt), bezeichnet eine Grafikschirmbetriebsart mit hoher Auflösung. (Beim MZ-800 im 800er Modus).

I/O-Operationen Von englisch Input/Output. Bezeichnet den Datentransfer zwischen CPU und einem anderen Rechner bzw. Baustein.

LISP Sprache der künstlichen Intelligenz

LOGO Einfach zu erlernende Programmiersprache, in der Struktur (Existenz von Listen etc.) mit » LISP verwandt.

Monitor Im Rechner fest installiertes Programm, das der Realisierung bestimmter Grundfunktionen (Laden/ Saven/ Editieren von Speicherbereichen etc.) dient. Beim MZ-800 existieren zwei Monitore: Der MZ-700 Monitor im Bereich 000H-1000H, der bis auf geringe Anpassungen vollständig mit dem des MZ-700 übereinstimmt (Warmstart: 00ADH), und dem MZ-800 Monitor im Bereich E000H-FFFFH, der beim Einschalten/ RESET-Drücken und bei JP 0000H gestartet wird.

QD Kurzbezeichnung für » Quickdisk.

Quickdisk Sequentiell arbeitende 2,8" Minidiskette. Beim MZ-800 voll » ROM unterstützt.

RAM Von englisch Random Access Memory = Speicher für wahlfreien Zugriff. Speicher der sowohl gelesen als auch beschrieben, d.h. verändert werden kann.

ROM Von englisch Read Only Memory = Nur Lese Speicher. Wird bei der Herstellung fest programmiert und kann vom Prozessor nur noch gelesen aber nicht mehr verändert werden.

RDD Kurzform von Read Data. Bezeichnet die ROM-Routine, die zum Lesen des Datenblocks einer Aufzeichnung dient.

RDINF Kurzform von Read Information. » ROM-Routine, die zum Lesen des » Headers (Information) einer » Datei dient.

VRAM Kurzform für Video » RAM (Bildwiederholspeicher). Derjenige Bereich des Rechners, der das im Moment auf dem Bildschirm dargestellte Bild enthält. Bezeichnet sowohl den Zeichen/ Farbspeicher im 700er Modus als auch den » HRG-Speicher im 800er Modus.

WRD Kurzform von Write Data. » ROM-Routine, die dem Aufzeichnen des Datenblocks einer Datei dient.

WRINF Kurzform von Write Information. » ROM-Routine, die den » Header einer » Datei aufzeichnet.

Anhang B: Codetabellen

■ Anzeige-Kode-Tabelle

Der Anzeige-Kode dient der direkten Anzeige von Zeichen. Die entsprechenden Kodes müssen dazu direkt im Videospeicher abgelegt werden. Die Monitorroutinen PRINT (0012H) und MSG (0015H) verwandeln ASCII-Zeichen zunächst in Anzeige-Kode und speichern diesen dann an der durch den Kursor bestimmten Adresse im Video-RAM. Kodes zwischen C1H und C6H dienen der Kursorbewegung.

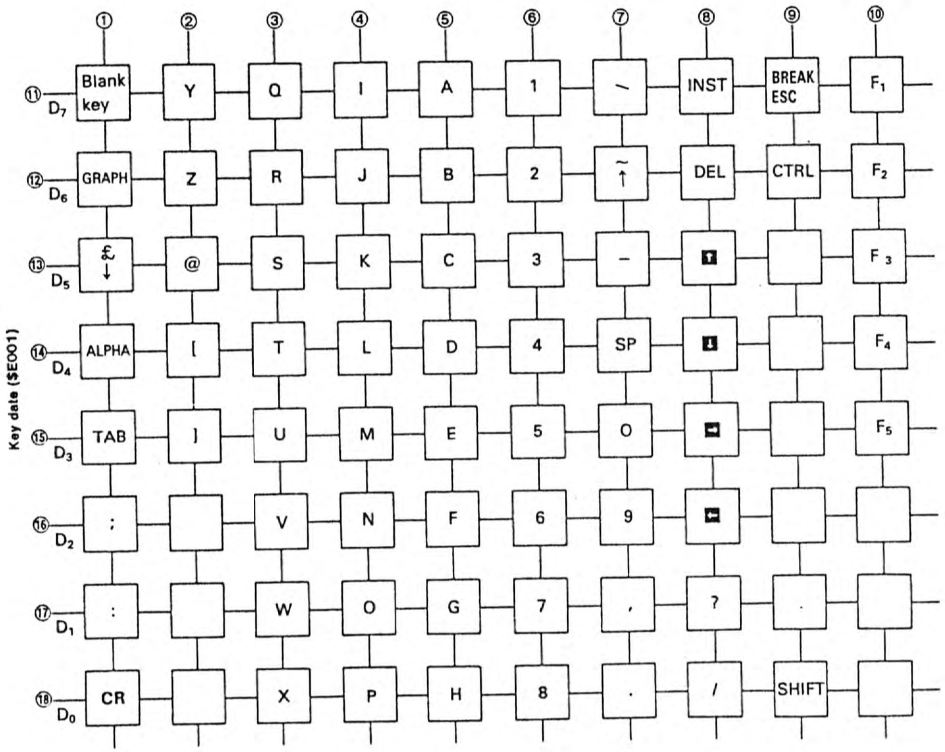
LSD \ MSD	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0 0000	SP	P	0		}	↑	π		|	p			↓			SP
1 0001	A	Q	1		♠	<	!		a	q			↓			
2 0010	B	R	2			[	"		b	r			↑			
3 0011	C	S	3		■	♥	#		c	s			→			
4 0100	D	T	4		◆	]	$		d	t	`		←			
5 0101	E	U	5		←	@	%		e	u	~		H			
6 0110	F	V	6		♣		&		f	v			C			
7 0111	G	W	7		●	>	'		g	w						
8 1000	H	X	8		○	↓	(		h	x			H			
9 1001	I	Y	9		?	\	)		i	y			I			
A 1010	J	Z	—			→	+		j	z	ß					
B 1011	K	£	=				*		k	ä	ü			°		
C 1100	L		;						l		ö	{				
D 1101	M		/						m		Ü		¥			
E 1110	N		.						n		Ä	^				
F 1111	O		,		:				o		Ö	_				

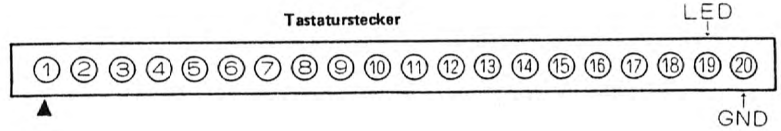
Anzeigecoden-Tabelle (2 kByte der zweiten Hälfte)

LSD \ MSD	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0 0000																
1 0001																
2 0010																
3 0011																
4 0100																
5 0101																
6 0110																
7 0111																
8 1000																
9 1001																
A 1010																
B 1011																
C 1100																
D 1101																
E 1110																
F 1111																

[Tastaturmatrix]

Um feststellen zu können, welche Taste der Tastatur gedrückt wurde, erzeugt man über Port PA des Bausteins 8255 ein Signal und legt es an die unten dargestellte Tastaturmatrix an (Anschlüsse 1 bis 10). Ist nun eine Taste gedrückt, wird dieses Signal an die Anschlüsse 11 bis 18 weitergeleitet und wird dort von Port PB abgenommen. Durch Auswertung von angelegten und abgenommenen Signalen kann dann die gedrückte Taste bestimmt werden.





Anhang C: Assembler Pseudo-Ops

In den vorangegangenen Kapiteln wurden in allen Beispielprogrammen sogenannte Assembler Pseudo-Operationen verwendet, kurz Pseudo-Ops.

Diese Pseudo-Ops sind bestimmte Befehle wie z.B. DEFM/DEFB etc., die bei fast allen Assemblern gleich sind und bestimmte Operationen auslösen. Dabei sind diese Befehle jedoch **keine** Anweisungen des Z80-Prozessors, sondern einzig und allein Kommandos an den Assembler, wie er bestimmte Passagen zu verstehen hat.

Für den Fall, daß der von Ihnen verwendete Assembler die benutzten Anweisungen vielleicht nicht versteht, oder daß Sie vielleicht die Bedeutung der Befehle nicht kennen, finden Sie nachfolgend eine Liste der verwendeten Pseudo-Ops und deren Bedeutung.

DEFB a,b,c,... Legt die nachfolgend spezifizierten Daten direkt im erzeugten Objekt-Code ab.

```
Beispiel:   LD A,00H
            DEFB 0C9H
            LD A,03H
```

erzeugt den Objektcode 3E 00 C9 3E 03 (3E nn = LD A,nn)

DEFM "abcdef..." Legt den nachfolgend spezifizierten Text im ASCII-Code direkt im erzeugten Objektprogramm ab.

```
Beispiel:   LD HL,0000H
            DEFM "01234"
            RET
```

erzeugt den Objektcode 21 00 00 30 31 32 33 34 C9 (30H = ASC("0") etc.)

ORG hhhh Bestimmt die Adresse, ab der ein Objektcode abgelegt werden soll. Bei anderen Assemblern auch REL (von englisch Relate)