Folge 1 13.4.89

Hallo Spectrum User!

Nach gut einem Jahr Mitgliedschaft im Club möchte ich auch endlich mal versuchen einen Beitrag zu leisten. Wie ich gerade wieder dem April-Info entnahm, gibt es doch wohl eine Menge Beta-Disk-User, für die jedoch bisher recht wenig an Information erschienen ist. Zwar wurde schon mehrfach eine Beta-Ecke angeregt, aber geworden ist leider nie was draus. Deshalb versuche ich heute mal den Anfang zu machen, in der Hoffnung, daß auch andere User ihre Kenntnisse und Erfahrungen mit dem Beta-Systemen beisteuern!

Ich arbeite nun seit gut 2 Jahren mit der Beta-Disk-Version 5.03, einem 48 kB Spectrum, 2 NEC 3.5" Laufwerken (Typ 1036 A), Iso-Monitor-ROM für Beta 5.03 und dem Kempston E Centronics-Interface und bisher gab es noch keine Probleme damit.

Ich möchte kurz für alle, die diese Version nicht kennen, eine Beschreibung vornehmen. Meines Wissens handelt es sich bei der Version 5.03 um die letzte angebotene Ausführung, die sowohl für den 48 er und den 128 er Spectrum geeignet ist! Parallelbetrieb mit IF 1 ist möglich. Ich kann allerdings nur über meine Erfahrungen mit ersterem berichten!

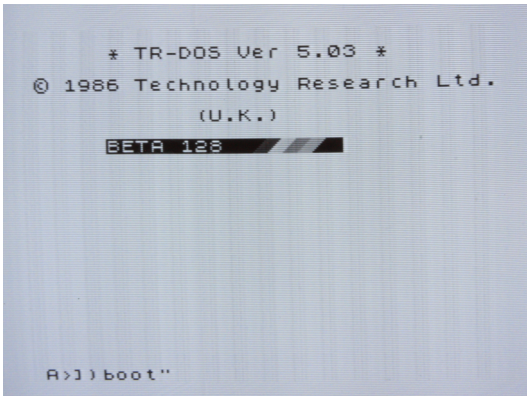
Bis zu 4 Laufwerke können verwendet werden, auch unterschiedliche (3", 3.5", 5 1/4 "), ein- oder doppelseitige, für 40 und/oder 8O Tracks. Auch Kombinationen aller Varianten sind möglich! Das System ist also sehr flexibel, die Laufwerke müssen nur einem Shugart-Bus besitzen um anschließbar zu sein! Sie werden automatisch beim ersten Aufruf geprüft und die Version des jeweiligen Laufwerks dann gespeichert. Die Disketten werden je nach Laufwerk oder Einstellung mit 40 oder 80 Tracks ein- oder doppelseitig formatiert, wobei jeder Track in 16 Sektoren unterteilt wird. Da jeder Sektor 256 Bytes groß ist, kann platzsparend gespeichert werden.

Track 0, Sektor 0 - 7 ist automatisch reserviert für den Katalog und in Sektor 8 ist der sogenannte Organisationsektor abgelegt, Sektor 9 - 15 wird bei der Version 5.03 normalerweise nicht benutzt. Dies soll angeblich bei älteren Beta-Disk-Ausführungen anders sein, aber dazu kann ich nichts Näheres sagen. Aus der Anzahl der übrigen Tracks läßt sich nun die max. Speicherkapazität pro Diskette errechnen:

```
4O Tracks einseitig = 39  * 16 = 624 Sektoren * 256 = 156 kB
40   "     doppels.  = 79  * 16 = 1264   "     * 256 = 316 kB
80   "     einseitig = 79  * 16 = 1264   "     * 256 = 316 kB
80   "     doppels.  = 159 * 16 = 2544   "     * 256 = 636 kB
```

Das Betriebssystem der Beta-Disk 5.03 ist in einem Eprom 27128 (16 kB) untergebracht. Es liegt im gleichen Adressbereich wie das Spectrum-ROM. Das Interface besitzt einen vollständig durchgeschleiften Spectrum-Bus, einen 'System-Schalter' (für Reset, 48 kB, 128 kB), einen 'Magic Button' (zum Saven des gesamten Speicherinhalts auf Knopfdruck), eine Buchse für die Spannungsversorgung und einen 34 poligen Shugart-Bus!

Nach dem Einschalten oder einem Reset meldet sich der Rechner nun mit dem folgenden Bild (beim 48 er Spectrum):



Das Laufwerk 'A' ist nun als aktuelles Laufwerk festgelegt und der Rechner erwartet einen Befehl! Dazu werden die Befehlsworte benutzt. Mit einem Trick könnte man auch die gleichen Worte in Einzelbuchstaben eingegeben, wahlweise in Groß- oder Kleinschrift. Nach 'REM' dazu den gewünschten Befehl in Einzelbuchstaben eintippen 'REM' löschen und den Befehl mit >Enter< übergeben! Und siehe da, er wird genauso ausgeführt, wie die normalen Befehlsworte. Es wird zwar

kein User seine Befehle beim 48 er Spectrum so umständlich eingeben, aber man sieht, das beide Varianten vom Betriebssystem verstanden werden! Die folgenden Befehle sind möglich:

```
*"a:"       = Laufwerk 'A' zum Hauptlaufwerk machen
*"b:"       = Laufwerk 'B' zum Hauptlaufwerk machen
*"c:"       = Laufwerk 'C' zum Hauptlaufwerk machen
*"d:"       = Laufwerk 'D' zum Hauptlaufwerk machen
40          = Laufwerk auf 40 Tracks umstellen, wenn möglich
80          = Laufwerk auf 80 Tracks umstellen, wenn möglich
CAT         = Disk-Katalog auf Bildschirm ausgeben
CAT #x      = Disk-Katalog auf Drucker ausgeben (x = Kanalnummer)
LIST        = Detailierten Disk-Katalog auf Bildschirm ausgeben
LIST #x     = Detailierten Katalog auf Drucker ausgeben (x = Stream)
OPEN #x     = Seriellen oder Random-File öffnen (x = Streamnr.)
CLOSE #x    = Seriellen oder Random-File schließen (x = Streamnr.)
INPUT #x    = Seriellen oder Random-Access-File einlesen
PRINT #x    = Seriellen oder Random-Access-File speichern
COPY        = Files von einem zum anderen Laufwerk kopieren
COPY s      = Einzelne Files kopieren mit nur einem Laufwerk
COPY b      = Gesamte Disk kopieren mit nur einem Laufwerk (Backup)
ERASE       = File-Namen im Katalog der Diskette löschen
LOAD        = Programm, Code oder Datas von Diskette laden
RUN         = Programm von Diskette laden und starten
MERGE       = Basic-Prog. im RAM mit einem von Diskette kombinieren
SAVE        = Programm, Code oder Datas auf Diskette abspeichern
VERIFY      = Gesavetes Programm auf Richtigkeit prüfen
MOVE        = Files 'verdichten', durch Entfernen gelöschter Files
NEW         = Vorhandenen File-Namen auf Diskette ändern
PEEK        = Einen bestimmten Sektor von Diskette ins RAM laden
POKE        = Daten aus RAM in bestimmten Disk-Sektor schreiben
GOTO        = Per 'Magie Button' gasavten Code laden und starten
FORMAT      = Diskette im event. eingestellten Format formatieren
RETURN      = Rückkehr aus dem TRDOS ins normale Betriebssystem
```

Ins Beta-Betriebssystem (im folgenden kurz 'TRDOS' genannt) geht's bei Direkteingabe mit RANDOMIZE USR 15616

Es erscheint nun entweder das oben abgedruckte Bild oder ein leerer Screen mit dem Laufwerkssymbol 'A>' (oder B-D) und die Befehlseingabe kann erfolgen. Falscheingaben werden mit *ERROR* quittiert und können korrigiert werden. Außerdem gibt es noch einige Fehlermeldungen im Klartext (z.B. 'No files', 'File exists', 'No space' usw.). Aber dazu später einmal mehr!

Mit 'RUN' kann nun ein Programm mit dem Namen 'boot' geladen werden. Das gleiche geschieht automatisch auch nach einen Reset oder dem Einschalten, wenn sich eine Disk mit einem Programm 'boot' im Laufwerk befindet (Autoboot). Dies soll bei Spectrum 128 nicht funktionieren.

Aus dem Basic wird das TRDOS anders aufgerufen (Beispiel -> Katalog einer Disk auflisten):

```
100 RANDOMIZE USR 15619: REM: CAT
```

Diese Zeile wird ausgeführt im Laufe des Programms und dann normal die nächste Zeile abgearbeitet. Anders sieht's jetzt allerdings mit den Fehlermeldungen aus. Sie werden im Interesse eines ungestörten Programmablaufs nun unterdrückt! Soll dies verhindert werden, ist eine Errorabfrage einzubauen! Bei Syntaxfehlern erscheinen die normalen Spectrum-Fehlermeldungen.

Jedes Laufwerk kann als permanentes (siehe oben) oder temporäres verwendet werden. Beispiel aus dem TRDOS heraus für letzteres:

```
LOAD "b: Programm" CODE
```

Und schon wird der gewünschte Code von einer Diskette im Laufwerk 'B* geladen, egal welches Laufwerk als Hauptlaufwerk definiert ist. Daran ändert sich auch nichts!

Noch etwas wichtiges: Jeder Befehl beansprucht eine eigene Basic-Zeile, die immer mit 'RANDOMIZE USR 15619: REM:' eröffnet werden muß! Trennung per Doppelpunkt ist nicht möglich. Beim Umschreiben von Kassettenprogrammen sind deshalb manchmal

zusätzliche Zeilennummern erforderlich! Die Zahl 15619 kann naturlich durch eine numerische Variable ersetzt werden, ebenso ein Programmname <String-Var.). Hier sind auch Kombinationen erlaubt, wie 'SAVE "1"+a$', was gut verwertbar ist, falls mehrere Programmteile mit dem gleichen Namen abgespeichert werden sollen. Wenn beim Laden das Gleiche gemacht wird, gibt's keine Probleme mehr! Natürlich dürfen auch die Parameter wie Startadresse und Prog.-Länge durch numerische Variablen ersetzt werden.

So, das war für's Erste die Kurzbeschreibung. Es gibt noch eine Menge dazu zu sagen, aber vermutlich ist das meiste für alte Beta-Disk-Hasen nichts Neues! Besitzer älterer Versionen können vielleicht in etwa erkennen, was hier anders ist.

Nun zum Betriebssystem der Version 5.03! Seit mehr als l Jahr habe ich mich intensiv damit beschäftigt. Dazu mußte ich zuerst einmal alles disassemblieren. Das ergab einen Haufen bedrucktes Papier, so ca. 60 Seiten DIN A4 mit etwa 90 Zeilen pro Seite. Und da habe ich mich dann, wenn Zeit vorhanden war, daran gemacht und versucht, die ganze Sache so allmählich zu verstehen und erst einmal mit handschriftlichen Kommentaren zu versehen. Zuerst ging's sehr mühsam voran, aber so nach und nach findet man sich immer besser darin zurecht. So konnte ich bis heute zumindest einen Teil der Routinen entschlüsseln und am Rest wird weitergearbeitet! Nun habe ich vor, so nach und nach vor allen Dingen allgemein brauchbare und interessante Routinen ausführlicher kommentiert als Assemblerlisting im Info zu beschreiben« Vielleicht besteht ja beim Einen oder Anderen Interesse an sowas!

Jetzt wird sich mancher fragen, wie kommt man ohne Hilfsmittel wie Epromer oder ähnliches überhaupt an das Betriebssystem heran? Ganz einfach geht's! Nur aus dem TRDOS heraus folgenden Befehl eingeben: 'SAVE "Beta5.O3" CODE 0,16384', und Sekunden später hat man den gesamten Eprom-Inhalt auf Diskette. Nun einen Disassembler laden und mit 'LOAD "Beta5.03" CODE xxxxx' den gespeicherten Code in einen geschützten Adressbereich laden und schon kann man alles auslesen!

Da ich aber natürlich sicher gehen wollte, daß das Betriebsystem auch wirklich vollständig und korrekt nach oben beschriebenen Verfahren wiedergegeben wurde, habe ich auch noch das ausgebaute Eprom ausgelesen und disassembliert! Und schon gab's eine Überraschung: Einzelne Codes stimmten nicht überein mit der vorher gewonnenen Version und es ergaben sich unsinnige Befehlsfolgen! Die Ursache war schnell gefunden! Bei allen Zahlen bei denen Bit O oder 7 gesetzt war, trat die Unstimmigkeit auf. Der Grund: vermutlich aus rein praktischen Gründen hat man die Datenleitungen für Bit 0 und 7 beim Eprom vertauscht, was ja belanglos ist, weil die Hardware dafür ausgelegt ist! Nur beim Auslesen in der üblichen Reihenfolge gibt's dann Unsinn. Zur Erklärung ein Beispiel:

	Bit 7 6 5 4 3 2' 10				
	0 0 0 0 0 0 0 1	= Hex.	01	= Dez.	1
Bits vertauscht	1 0 0 0 0 0 0 0	= "	80	= "	128
	1 1 1 1 1 1 1 0	= "	FE	= "	254
Bits vertauscht	0 1 1 1 1 1 1 1	= "	7F	= "	127

Nach dieser Erkenntnis läßt sich nun ein kleines Basicprogramm schreiben, was beim Auftreten der entsprechenden Zahlen korrigiert und mit den Werten der ersten Version vergleicht. Dazu müssen natürlich Beide gleichzeitig an unterschiedlichen Adressen im Speicher stehen (Programm-Beispiel siehe unten! ! Der Test ergab völlige übereinstimung beider Versionen.

Somit steht dem Disassemblieren der 1. Version nichts mehr im Wege. Wer es nachvollziehen will, sollte möglichst einen Disassembler benutzen, der die Verwendung der Originalstartadr. 0 erlaubt, auch wenn der Code an beliebiger Stelle steht im Speicher. Das verbessert die Übersichtlichkeit des Listings. Günstig ist außerdem, wenn man die Ausgabe der Spectrum-ROM spezifischen Datenbytes bei RST $08 bzw. RST $28 Befehlen abschalten kann, da diese im TRDOS hier nicht existieren (dafür jedoch beim RST S2O, dazu demnächst mehr!).

Am Besten beginnt man damit, den gesamten Code als Hexdump mit

gleichzeitiger ASCII-Zeichendarstellung durchzusehen. So lassen sich schnell Text-, Tabellen- und leere Eprom-Bereiche (enthalten $FF) finden. Die gefundenen Anfangs- und Endadressen bitte notieren! Nun kann man die übrigen Bereiche als Disassembler-Listing und die Tabellen usw. als Hex-Dump ausdrucken und erhält so ein zum größten Teil korrektes Listing! Diese Vorgehensweise empfiehlt sich immer beim Prüfen unbekannter MC-Programme.

Nun noch eine Auflistung der Bereiche, die als Hex-Dump auszugeben sind (soweit mir bisher bekannt):

Anfangs-	Endadresse	Inhalt
$0033	$0037	leer = ($FF)
$0360	$03AB	Titeltext (* TR-DOSVer 5.03 * usw.)
$1000	$1017	Text (Interface one fitted)
$10A5	$1150	Titeltext für LIST-Befehl
$16F9	$170F	Leer = ($FF)
$1FB9	$1FC9	Tabelle $01-$10 (Zweck noch unklar!)
$2766	$283B	Fehlerme1dungs- und sonstige Texte
$2880	$28D7	Funktionsnummern mit jeweil. Einsprungadr.
$29B3	$2A34	Fehlermeldungstexte, CAT-Titeltext
$2FF3	$3031	Befehlscodes mit jeweiliger Einsprungadr.
$30FD	$3107	Befehlwort-Tabelle für 128er/48er Spectrum
$3108	$31F2	Befehlscodes zu vorstehender Tabelle
$31FD	$3000	Leer ($FF)
$3C0r<	$3CF9	Leer ($FF)

Alles sonstige wird normal disassemb1iert. Einen Punkt muß man noch beachten: hinter einem $E7 => RST $20 - Befehl folgen im TRDOS immer zwei Bytes, die eine Adresse für eine Routine im normalen Spectrum-Betriebssystem ergeben. Wie das funktioniert, werde ich vielleicht ein anderes Mal erklären. Für heute soll’s erst mal reichen. Bis zum nächsten Mal!

MfG
Wilhelm

```
    1 REM Ab 30000 = > Eprom-Version
    2 REM Ab 47000 => Disk-Version
    5 PRINT "Eprom";TAB 6;"Orig.";TAB 11;"Korrig."; TAB 21;"Disk"; TAB 28;"Code"
   10 FOR n=30000 T0 46384
   20 LET x=n+17000
   30 G0 SUB 100
   40 NEXT n
   50 STOP
  100 PRINT n;TAB 6;PEEK n;TAB 11;
  110 IF INT (PEEK n/2)=PEEK n/2 AND PEEK n>=128 THEN PRINT INVERSE 1;PEEK n-127;
INVERSE 0;: G0 T0 140
  120 IF INT (PEEK n/2)<>PEEK n/2 AND PEEK n<= 127 THEN PRINT INVERSE 1;PEEK n+127;
INVERSE 0;: G0 T0 140
  130 PRINT PEEK n;
  140 PRINT TAB 21;x;,TAB 28; PEEK x
  150 RETURN
 9998 STOP
 9999 RANDOMIZE USR 15619: REM : SAVE "Betatest"
```

Hallo Spectrum User!

Heute will ich mal wieder fortfahren mit der Beschreibung der BETA-Disk Version 5.03 und zwar soll's diesmal hauptsächlich um den vom TRDOS benötigten Puffer für die zusätzlichen System- variablen gehen. Wie im Handbuch beschrieben, werden dafür 112 Bytes im RAM benutzt von Adresse $5CB6 - $5D25 (23734 – 23845). Da ab $5D26 noch die verschobenen Kanalinformationen folgen, kann ein Basicprogramm nun bei Adresse 23867($5D3B) beginnen. Die neuen Systemvariablen werden beim ersten Einschalten, oder falls zwischenzeitlich gelöscht, auch durch die RANDOMIZE USR 15616 (oder ...15619: REM:...) initialisiert!

Nach dem Einschalten arbeitet der Rechner zuerst im BETA-ROM einige Routinen ab, die den Originalen ROM-Routinen stark ähneln. Wer ein SPECTRUM-ROM-Listing hat, kann mal vergleichen, (Im TRDOS von Adr. 0 - $00F9 findet man weitgehend im ROM bei Adr. 0-4 und $11CC - $1287 wieder!) Es wird im Wesentlichen der Speicher gecheckt, die UDG's kopiert und Systemvariablen gesetzt!

Durch eine gegenüber dem SPECTRUM-ROM modifizierte Kopierroutine ($00A5-$00B2 im BETA-ROM) werden nun zuerst die normalen Kanalinformationen ins RAM ab $5CB6 - $5CCA kopiert. Nachdem weitere normale Systemvariablen mit ihren Ausgangswerten besetzt wurden, erfolgt bei Adr. $0114 ein JP $3D31 (Zeile 710)! Zum Verfolgen des weiteren Geschehens dient das folgende Assemb1er1isting:

```
31F3          0010       0R6 $31F3
              0020 ;
31F3 2A4F5C   0030       LD HL,(5C4F) ;$5C4F = CHANS -)Adr. von Kanalinform. ins HL-Reg.
              0040 ;
              0050 ;Wenn Bereich noch nicht erweitert, steht hier die Adr. 55CB6 (23734),
              0060 ;sonst gewoehnlich $5D26 (= 23734*112=238461.
              0070 ;
31F6 B7       0080        0R  A          ;Flags loeschen!
31F7 01255D   0090        LD  BC,$5D25   ;Endadr. des erweiterten Bereichs ins BC-Reg. laden
31FA ED42     0100        SBC  HL,BC     ;Subtr. HL-BC unter Beruecksichtigung des Uebertrag-Flags!
              0110 ;
              0120 ;Je nachdem, ob in HL $5CB6 oder $5025 steht, wird bei der Subtraktion
              0130 ;das C-(Uebertrag)-Flag gesetzt ($5CB6-$5D25) oder nicht ($5D26-$5D25) 0140 ;
31FC C9       0150       RET            ;Sprung zurueck nach $3D24
              0160 ;
              0170 ;Der Bereich von $31FD - $3C00 im Eprom ist leer ($FF)
              0180 ;

3C01          0200       ORG $3C01
              0210 ;
3C01 1803     0220        JR  L3C06      ;Direktaufruf der Version 5.xx landet hier
3C03          0230        DB $FF
3C04 1803     0240        JR  L3C09      ;Aufruf aus Basic bei Version 5.xx landet dagegen hier
3C06 C3003D   0250L3C06   JP  $3D00      ;Sprung  erfolgt nach 15616
3C09 C3033D   0260L3C09   JP  $3D03      ;Sprung erfolgt nach 15619
              0270 ;
              0280 ;Der Bereich von $3C0C - $3CF9 im Eprom ist leer ($FF)
              0290 ;in diesem Bereich landen Aufrufe der Beta-Version 4.xx!
              0300 ;

 3CFA         0320       ORG $3CFA
              0330 ;
 3CFA C3F120  0340   L3CFAJP  $20F1      ;Aufruf einer Routine zum Testen auf  IF  1
 3CFD C33C28  0350   L3CFDJP  $283C      ;Sucht eine im C-Reg. stehende Funktionsnr. bei MC-Aufruf
 3D00 00      0360 L3D00 NOP
 3D01 182E    0370        JR  L3D31      ;Hier landet RANDOMIZE USR 15616!
 3D03 00      0380 L3D03  NOP
 3D04 1814    0390        JR  L3D1A      ;Hier landet RANDOMIZE USR 15619!
 3D06 00      0400        NOP
 3D07 C3EF25  0410       JP   $25EF      ;Bedeutung momentan noch ungeklaert!
 3D0A C34A24  0420 L3D0A  JP  $244A      ;Desgleichen!
 3D0D 00      0430        NOP
 3D0E 18FA    0440        JR  L3D0A
 3D10 00      0450        NOP
 3B11 18E7    0460        JR  L3CFA
 3D13 00      0470        NOP
 3D14 18E7    0480       JR   L3CFD      ;Hier erfolgt meist der Aufruf des TRDOS aus MC-Prog.!
              0490 ;
              0500 ;Es muessen neben Funktionsnr. noch div. Parameter uebergeben werden.
              0510 ;Beispiele folgen spaeter!
              0520 ;
```

```
3D16 00        0530         NOP
3D17 C3692F    0540         JP $2F69          ;Alle Fehlmeldungen laufen über diese Routine ab $2F69
3D1A CD213D    0550 L3D1A   CALL   L3D21      ;Hier zum Erweiterung testen, nach RANDOMIZE USR. 15619
3D1B E5        0560         PUSH   HL         ;HL.-Reg. enthaelt hier Ergebnis der Subtr. von Adr. $31FA (=$5CC2)
3D1E C36C01    0570         JP $016C          ;Bei $016C weitermachen (nach Aufruf aus Basic)
3D21 CDF331    0580 L3D21   CALL   $31F3      ;Testen, ob Puffer auf 112 Bytes erweitert ist
3D24 00        0590         NOP
3D25 00        0600        NOP

3D26 DC4C3D     0610        CALL   C,L3D4C    {Wenn nicht erweitert (C-Flag gesetzt!), nach $3B4C
3D29 21C25C     0620        LD   HL,$5CC2     »Sonst HL-Reg. mit $5CC2 laden !
                0630 ;
                0640 ;Dient spaeter als Ruecksprungadresse und enthaelt einen RET-Befehl.
                0650 ;
3D2C C9         0660        RET               ;Zurueck nach $3D1D oder $3D34, je nachdem, woher Aufruf kam
3D2D 00         0670        NOP
3D2E 00         0680        NOP
3D2F 00         0690        NOP
3D30 C9         0700        RET
3D31 CD213D     0710  L3D31 CALL L3D21        ;Testen, ob Erweiterung auf 112 Bytes schon geschehen.
                0720 ;
                0730 ;Hierher erfolgte der Sprung bei Adr. $0114 im Beispiel oder nach
                0740 ;Aufruf mit RANDOMIZE USR 15616!
                0750 ;
3D34 E5         0760        PUSH HL           ;HL-Reg. enthaelt $5CC2 als Ruecksprungsdr. (enthaelt RET)
3D35 C33902     0770        JP $0239          ;Hier geht's anschliessend weiter!
                0780 ;
                0790 ;
                0800 ;Der folgende Bereich $3D38-$3D4B) wird spaeter in den MEMBOT-Bereich
                0810 ;kopiert und gestartet.(Test ob RS-232 oder ZX-Printer angeschlossen)
                0820 ;
3D38 AF         0830 L3D38  XOR   A           ;A-Reg. loeschen.
3D39 D3F7       0840        OUT   ($F7),A
3D3B DBF7       0850        IM   A,($F7)      ;Serielle Schnittstelle (Port $F7) abfragen?
3D3D FE1E 0860              CP $1E            ;RS-232-Ausgabe moeglich?
3D3F 2803       0870        JR   Z,L3D44      ;Dann weiter bei $3D44!
3D41 FE1F       0880        CP   $1F          ;ZX-Printer-Ausgabe?
3D43 C0         0890        RET   NZ          ;Wenn nicht, zurueck
3D44 CF         0900        L3D44 RST $08     ;Bei IF 1 wird dies aufgerufen und der nachfolgende
3D45            0910        DB   $31          ;Anhaengebefehl 'Variablen initialisieren' ausgefuehrt!
3D46 3E01       0920        LD   A,$01        ;A-Reg. mit 1 laden als  'Merker'fuer initialisierte RS-232
3D48 32EF5C     0930        LD   ($5CEF),A    ;Schnittstelle und dies in $5CEF retten!
3D4B C9         0940        RET               ;Zurueck!
                0950 ;
3D4C AF         0960  L3D4C XOR  A            ;A-Reg. mit 0 laden.
3D4D D3FF       0970        OUT  ($FF).A      ;???
3D4F DBF6       0980        IN   A,($F6)      ;???                    .
3D51 21383D     0990        LD   HL,L3D38     ;Anfangsadr. des zu kopierenden Bereichs ins HL-Reg.
3D54 11925C     1000        LD   DE,$5C92     ;Adr.von MEMBOT als Zieladr. ins DE-Reg.
3D57 011400     1010        LD   BC,$0014     ;und Zeichenanzahl BC laden!                    .
3D5A EDB0       1020        LDIR              ;Nun die 20 Zeichen ab $3D38-$304B nach $5C92 kopieren!
3D5C 21673D     1030        LD   HL,$3D67     ;Adr.   wird nach Ausfuehrung des JP bei $3D64 angesprungen!
3D5F E5         1040        PUSH  HL          ;Dazu   auf dem Stapel ablegen.
3D60 212F3D     1050        LD   HL,$3B2F     ;Diese Adr. enthaelt   einen RET-Befehl.
3D53 £5         1060        PUSH  HL          ;Auch   auf den Stapel als Ruecksprungadr.!
3D64 C3925C     1070        JP   $5C92        ;Aufruf der kopierten Routine in MEMBOT im RAM!
                1080 ;
                1090 ;Nach Rueckkehr aus der Routine geht's nach $3D2F, was ein RET bewirkt
                1100 ;im Beta-ROM nach $3D67!
3D67 21902F     1120        LD   HL,$2F90     ;HL mit Startadr. fuer Routine 'Defaultwerte setzen' laden
3D6A E5         1130        PUSH HL           ;und auf den Stapel!
3D6B 212F3D     1140        LD   HL,$3B2F     ;Adr. zeigt auf RET-Befehl.
3D6E E5         1150        PUSH HL           ;Ebenso auf  den Stapel!
3D6F 215516     1160        LD   HL,$1655     ;Adr. einer  ROM-Routine zum Reservieren von Speicherplatz
                1180 ;Die Bytezahl dazu wird in BC uebergeben, HL zeigt auf die Stelle,
                1190 ;hinter der Platz geschaffen werden soll!
                1200 ;
3D72 E5         1210        PUSH  HL          ;$1655 auf Stapel ablegen.
3D73 21FF5B     1220        LD    HL,$5BFF    ;Adr. des Druckerpufferendes (bei Spectrum  48)in HL!
3D76 E5         1230        PUSH  HL          ;Auch retten!
3D77 36C9       1240        LD    (HL),$C9    ;Nun in $5BFF ,(im  RAM!) ein RET schreiben.
3D79 21B55C     1250        LD    HL,$5CB5    ;P-RAMT = letztes   Byte der normalen Systemvariablen.
                1260 ;
                1270 ;Dahinter werden durch ROM-Routine $1655 die 112 Bytes reserviert!
                1280 ;
3D7C 017000     1290        LD    BC,$0070    ;BC-Reg. mit 112 1= Byte-Anzahl) laden.
3D7F C9         1300        RET               ;Zurueck nach $5BFF im RAM, was erneut ein RET bewirkt
                1310 ;
                1320 ;und nun geht's weiter im ROM bei $1655. Bei Rueckkehr wird dann $3D2F
                1330 ;im Einschaltbereich des Beta-ROM erreicht! Erneutes RET ist die Folge
                1340 ;nach $2F90 auch im Beta-ROM. Nach Ende wird nun $3D29 auf dem Stapel
                1350 ;,gefunden, damit ist Pufferinitialisierung beendet! Dort Weiter!
3D80            1360        END

#     61A8 L3CFA     3CFA   L3CFD  3CFD   L3D00    3D00 L3D03    3003 L3D0A 3D0A L3D1A 3D1A L3D21 3D21 L3D31 3D31
L3D38   3D38 L3DB44   3D44   L3D4C  3D4C
```

Jetzt noch die Routine (ab $2F90), die den geschaffenen Pufferbereich mit den wichtigsten Ausgangswerten füllt:

```
2F90          0010       ORG $2F90
              0020 ;
              0030 ;Im Folgenden wird der reservierte 112 Byte-Bereich mit den wichtigsten
              0040 ;Default-Werten gefuellt. Im weiteren Verlauf werden diese durch die
              0050 ;endgueltigen Werte ersetzt!
2F90  21FFFF  0060       LP   HL,$FFFF
2F93 22FA5C   0070       LD   ($5CFA),HL  ;Die folgenden Adr. mit $FF als Defaultwert laden. Hier
2F96  22FC5C  0080       LD   ($5CFC),HL  ;stehen spaeter Informationen zu den jeweils verwendeten
2F99 22C85C   0090       LD   ($5CC8),HL  ;Laufwerken.
2F9C 22CA5C   0100       LD   ($5CCA),HL
2F9F AF       0110       XOR  A           ;A-Reg. mit 0 laden.
2FA0 32175D   0120       LD   ($5D17),A   ;$00 -> ganzen Screen, $AA -) untere 2 Zeilen loeschen
2FA3 32195D   0130       LD   ($5D19),A   ;Laufwerk 0 (= A1 setzen,moeglich sind Werte von 0- 3!
2FA6 32185D   0140       LD   (S5D18),A   ;$00 -) Kein IF 1, $FF ->IF 1 angeschlossen!
2FA9 320F5D   0150       LD   ($5DOF),A   ;Fehlernummer
2FAC 321F5D   0160       LD   ($5D1FI,A   ;???
2FAF 3EFF     0170       LD    A,$FF      ;A-Reg. mit $FF laden.
2FB1 D3FF     0180       OUT   ($FF),A    ;???
2FB3 323A5C   0190       LD   ($5C3A),A   ;ERR NR mit $FF fuer Fehlermeldung 'OK' laden!
2FB6 32165D   0200       LD   ($5D16),A   ;Defaultwert fuer Laufwerkscode  setzen.
              0210 ;
              0220 ;Hier stehen spaeter fuer die Laufwerke A-D die Werte $3C-$3F, wenn
              0230 ;Seite 0, und $2C-$2F, wenn Seite 1 der betreffenden Diskette
              0240 ;angesprochen wird!

2FB9 320C5D    0250      LD     ($5D0C),A ;$00 ->Puffer-Erweiterung >$5225 vorhanden,$FF ->nein!

2FBC 3EC9     0260       LD   A,$C9       ;A-Reg. mit IC9 (= RET1 laden
2FBE 32C25C   0270       LD   ($5CC2),A   ;und an diese Adr. schreiben (siehe Bemerkung bei $3B29)!
2FC1 3ED0     0280       LD   Ä,$D0       ;A-Reg. mit 208 laden.
2FC3 D31F     0290       OUT   ($1F),A    ;???
2FC5 C9       0300       RET              ;Zurueck nach $3D29 und dort weitermachen!
```

Auch wenn dieses vielleicht nur für einige wenige User wirklich interessant war, geben die Routinen doch einen Einblick, wie trickreich und mit wie vielen Umwegen das Zusammenspiel zwischen BETA-ROM, SPECTRUM-ROM und RAM funktioniert, und das alles in Sekundenschnelle! Ähnliches spielt sich ja auch bei anderen nachgerüsteten Interfaces ab.

Vielleicht noch eine Erläuterung: Für das Verständnis der Routinen speziell Zeile 990 - 1360) muß man wissen, das nur Aufrufe des Bereichs von $385F (=14431) bis $3DFF (15871) das BETA-ROM einschalten. Soll vom BETA-ROM das SPECTRUM-ROM aufgerufen werden, ist ein Umweg über das RAM notwendig! Das gleiche gilt natürlich auch im umgekehrten Fall. Besonders elegant geht das eben, wenn wie gesehen, die anzuspringenden Adressen auf dem Stapel deponiert werden und vorhandene RET-Befehle im jeweiligen Bereich zum Abheben der Adressen benutzt werden!

Für heute genug! Nächstes mal nähere Informationen dazu, welche Daten der Puffer im Einzelnen enthält und weitere Routinen. Bis dann !

MfG
Wilhelm

Folge 3 15.7.89

Hallo Spectrum User!

```
Adr.   Inhalt (Hexadez.)   ASCII-Code

5CB6   F4 09 AS 10 4B F4   ....K.
5CBC   09 C4 15 53 81 0F   ...S..
5CC2   C9 15 52 F4 09 C4   ..R...
5CC8   83 FF FF FF 00 00   ......
5CCE   00 00 00 00 00 00   ......
5CD4   00 00 00 DB 5C 3D   ....ö=
5CDA   5D 00 00 62 6F 6F   ü..boo
5CE0   74 20 20 20 20 42    t   B
5CE6   00 00 00 00 00 00   ......
5CEC   00 00 00 00 00 00   ......
5CF2   00 00 01 00 00 00   ......
5CF8   00 00 08 FF FF FF   ......
5CFE   80 00 25 5D 3D 5D   ..%ü=ü
5D04   C6 31 09 00 00 00   .1....
5D0A   00 00 FF 00 00 00   ......
5D10   00 3C 5D 54 FF 00   .<UT...
5D16   3C AA 00 00 CB 02   <.....
5D1C   50 FF 00 00 FE OD   P.....
5D22   80 00 00 FF F4 09   ......
5D28   A8 10 4B F4 09 C4   ..K...
5D2E   15 53 81 OF C9 15   .S....
5D34   52 FC OE C4 15 50   R....P
5D3A   80 80 F9 CO 36 31   ....61
```

Nun weiteres aus dem BETA-Disk 5.03-Betriebssystem! Hier ein Beispiel dafür, wie der Puffer aussehen kann (hier wurde das TRDOS sofort mit 'RETURN' verlassen und der Puffer ausgelesen!) Die wichtigsten Parameter sind schon gesetzt, das aktuelle Laufwerk (=A) als 2*80 Tracks DS identifiziert und alles vorbereitet für's Laden eines Basic-Programms mit Namen "boot". Es fehlen noch programmspezifische Parameter. Diese können aber erst in den Puffer gelangen, wenn der Name auf der eingelegten Diskette auch gefunden wurde. War beim Einschalten schon eine Diskette im Laufwerk oder wurde mit RUN gestartet, wird ein Programm "boot" sofort geladen und ausgeführt!

Nun soweit mir inzwischen bekannt, die Bedeutung der Pufferadr. für das BETA-Disk-Betriebssystem:

5CB6 Enthält $F4, wenn kein IF 1 angesch1ossen ist. Sonst ???
5CB7- Entspricht meist weitgehend den ursprüngl. Kanalinforrn.
5CC7
5CC2 Enthält $C9 (=ret) (Wird für Rücksprung ins TRDOS benutzt)
5CC8- $FF, wenn entsprechendes Laufwerk nicht initialisiert ist,
5CCB sonst Code für Laufwerkstyp (genaueres dazu später einmal)
5CCC Zähler für zu ladenden Sektoren
5CCD Zähler für zu ladenden Track
5CCE $FF = Schreiben, $00 = Lesen eines Sektors
5CCF Zeigt auf Anfang der Kopie des Organisations-Sektors oder bei MOVE/COPY auf Anfangsadr. des 4 kB-Puffers
5CD0 ???
5CD1- Speichert Autostartadr. eines Basic-Prog. (Low-Byte)
5CD2 ″ " ″ " (High-Byte)
5CD3- ???
5CD5 ???
5CD6 Merker für Prog.-Typ?? Basic =03, Code =00, Code mit neuer Startadresse =01?
5CD7- Bei FORMAT: Anzahl freie Sektoren; Zeitweise Trackzahl.
5CD8 Sonstige Verwendung noch unbekannt!
5CD9- ???
5CDA ???
5CDB- Bei SAVE: Prog.- Länge (wie 5CE9)? LINE: Zei1ennummer?
5CDC OPEN#, CAT, LIST: Kanal-Nr.? LOAD: Blocklänge
5CDD- Enthält gewöhnlich den Programmnamen (Im Beispiel: 'boot')
5CE4 (max. 8 Zeichen lang)
5CE5 Filetyp (Im Beispiel: $42 = B = Basic-Prog.)
5CE6- Startadresse des Programms (Low-Byte)
5CE7 ″ " (High-Byte)
5CE8- Blocklänge des Programms (Low-Byte)
5CE9 ″ ″ (High-Byte)
5CEA Fileanzahl des Programms
5CEB Startsektor des Programms
5CEC Starttrack des Programms
5CED- Dieser Bereich dient bei COPY zum Zwischenspeichern der
5CF3 Daten von 5CE6-5CEC? Sonstige Verwendung noch unbekannt.
5CF4 Nächste zu bearbeitende Sektornummer
5CF5 ″ ″ Tracknummer
5CF6 Laufwerksnummer 0 = "A″, 1 = "B", 2 = "C", 3 = ″D"
5CF7 ???

```
5CF8 Bei COPY: Zwischenspeicher für Laufwerks-Nr. (0-3);
       Sonst: bei $FF = Zusatzpuffer > $5D25 bleibt erhalten,
              ”   $00 =         "        ”  wird gelöscht
5CF9  Vermutlich $00, wenn Autostart-Programm, sonst $FF
5CFA- $FF, wenn entsprechendes Laufwerk nicht initialisiert ist;
5CFD  sonst $04 bei 40-Track-, bzw. $08 bei 80-Track-Laufwerk
5CFE  FCD-Kommandocode
5CFF  Augenblicklich bearbeitete Sektornummer
5D00  Adresse des jeweiligen Pufferendes
5D01- ???
5D07  ???
5D08  Bei ERASE: Zwischenspeicher für’s erste Namenszeichen
5D09  Bei OPEN#: Fileart (RND, R, W)
5D0A- ???
5D0B  ???
5D0C  $00, wenn Puffererweiterung > $5025; $FF, wenn nicht
5D0D- ???
5D0E  ???
5D0F  Speicherplatz für Fehlernummer
5D10  ???
5D11- Augenblicklich bearbeitete Basic-Adresse (Low-Byte)
5D12          ”               ”               ”  (High-Byte)
5D13- Speicher für ERROR-Stapeladr. (Low-Byte)
5D14      ”                   "      (High-Byte)
5D15  $FF, bei Aufruf des TRDOS aus Basic; $00, falls direkt
5D16  Bei Disk-Seite 0 -> $3C - $3F; Disk-Seite 1 -> $2C - $2F
5D17  $AA: dann nur die unteren 2 Zeilen des Screens löschen;
      $00: dann ganzen Screen löschen
5D18  $FF, falls IF 1 angeschlossen, sonst $00
5D19  Nummer des aktuellen Laufwerks
5D1A- ???
5D1B  ???
5D1C- Zwischenspeicher für Stape1adresse (Low-Byte)
5D1D          "                       ”    (High-Byte)
5D1E- ???
5D22  ???
5D23- Bei MOVE und COPY: zur Verfügung stehender Speicherp1atz
5D24  (mind. 4 kB; mehr, falls möglich?)
5D25  Ende dieses 112-Zeichenpuffers. Hiernach können bei
      Bedarf Puffererweiterungen eingefügt werden!
5D26- Hier stehen nun die Kanalinformationen, wenn nur der 112
5D3A  Zeichenpuffer existiert.
5D3B  Dann steht hier $80. Dahinter beginnt ein Basic-Programm!
```

Diese Liste zusätzlicher System-Variab1en erhebt keinen Anspruch auf Vollständigkeit und absolute Richtigkeit! Es können auch noch andere Bedeutungen als die genannten existieren.
Wie schon angedeutet, wird oft über diesen 112-Zeichenpuffer hinaus noch weiterer Platz vom TRDOS beansprucht! Das ist z.B. der Fall bei jedem Zugriff auf eine Disk. Dann wird erst einmal zusätzlicher Raum ab $5D25 geschaffen (mind. 256 Bytes). Die Kanalinformationen werden dann ebenso wie ein eventuell im Speicher stehendes Basic-Prog. vorübergehend um die nötige Bytezahl nach oben verschoben und die Systemvariablen angepaßt. In Adr. $5D00 wird das neue Pufferende notiert und in $5D0C kommt eine $00 als Merker für’s TRDOS, daß der Puffer erweitert wurde! Gewöhnlich wird dann in den geschaffenen Raum erst mal der Organisations-Sektor von Track 0 / Sektor 8 kopiert. Was der enthält wurde schon mal in der ’CK’ Heft 12/87-1/88 beschrieben, trotzdem hier noch einmal:

Byte	Inhalt
0 - 224	Ist leer!
225	Nummer des nächsten freienSektors
226	Nummer des nächsten freienTracks
227	Diskformat (hier gab’s einen Fehler in der ’CK’) Bit 0 = 1 -> 40 Tracks Bit 0 = 0 -> 80 Tracks Bit 1 = 0 -> einseitig (SS) Bit 1 = 1 -> doppelseitig (DS)

i

					Hex	Binär
In der Praxis steht hier bei:	80	Tracks	DS	->	$16	(00010110)
	40	Tracks	DS	->	$17	(00010111)
	80	Tracks	SS	->	$18	(00011000)
	40	Tracks	SS	->	$19	(00011001)

228	Anzahl aller Files incl. der gelöschten auf der Disk
229-230	Anzahl der freien Sektoren auf der Diskette
231	Anzahl der Sektoren pro Track (immer $10 = Dezimal 16)
232-233	Enthalten immer 0
234-242	8 Zeichen für Paßwort bei Version 3.xx
243	Enthält immer 0
244	Anzahl der gelöschten Files
245-252	8 Zeichen für den Diskettennamen

Bei jedem Schreib- und Löschvorgang wird dieser Sektor aktualisiert. Des weiteren wird der erweiterte Puffer benötigt als Zwischenspeicher, wenn im Katalog z.B. ein Programmname gesucht werden soll.

Dazu wird ein Sektor nach dem anderen in den Puffer geladen und auf den Namen hin durchsucht. Falls nicht gefunden, geht's mit dem nächsten Sektor weiter bis die Suche Erfolg hat oder der Katalog zu Ende ist!

Wie in der erwähnten 'CK' zu lesen war, wird bei den älteren BETA-Disk-Versionen der Printerpuffer bei solchen Operationen als Zwischenspeicher verwendet, dessen ursprünglicher Inhalt dazu auf Track 0 / Sektor 9 zwischengespeichert wird. Dieser Weg kann bei der Version 5.xx nicht beschritten werden, da diese auch mit dem SPECTRUM 128 laufen. Meines Wissens wird nämlich im 128er-Modus der Printerpufferbereich mit zusätzlichen Systemvariablen belegt, sodaß dieser nicht als Zwischen-speicher zur Verfügung steht.

Übrigens, wer hat sich nicht schon mal darüber gewundert, daß bei ziemlich vollem Speicher zwar ein CAT möglich war, aber bei LIST die Fehlermeldung 'Out of memory' erscheint? Das hängt u.a. mit der bei LIST angezeigten Autostartadresse zusammen! Da diese ja nirgends im Katalog auftaucht, muß sie irgendwie anderweitig ermittelt werden und dafür benötigt das TRDOS weiteren Platz!

Die Gründe dafür: Beim LIST- wird im Gegensatz zum CAT-Befehl nach jedem vollen Screen der Kopftext mit allen Parametern wie Diskname, Filezahl, Anzahl gelöschter Files, Disketten-Version und Anzahl freier Sektoren erneut ausgegeben. Da der in den Bereich von $5D25 - $5E25 kopierte Org.-Sektor die dazu erforderlichen Angaben enthält, jedoch nach dem ersten Auslesen mit den Katalog-daten überschrieben würde, muß er vorher woanders untergebracht werden. Dazu schafft eine Routine im TRDOS Platz für 546 Zeichen (= 2 Files zu je 256 Bytes und 34 Bytes Org.- Sektordaten usw) im Workspace (Speicheraufteilung siehe SPECTRUM-48-Handbuch Seite 165) und kopiert die Org.-Sektordaten an den Beginn dieses Bereichs um.

Jedesmal wenn nun die Daten eines Basic-Programms gelistet werden, errechnet das TRDOS die Track- und Sektornummer der letzten beiden Sektoren dieses Programms und lädt diese Sektoren (=512 Zeichen) in den reservierten Workspace-Bereich oberhalb der Org.-Sektorkopie! Anschließend wird er auf die Code-Folge $8O, $AA durchsucht! Falls sie gefunden wird, steht dahinter die gesuchte Autostartadresse (die Suche danach wird im TRDOS im Adressbereich von $131B-$135F abgewickelt).

Warum muß aber gerade diese Code-Folge gesucht werden? Die Autostartadr. ist beim SAVE in den Bereich von E-LINE geschrieben worden und wenn man sich im SPECTRUM-Handbuch (Seite 165) die Speicherorganisation betrachtet, sieht man, das der VARS- und E-LINE-Bereich durch ein $80 getrennt ist. Dahinter schreibt beim SAVE eines Basic-Prog. mit Autostartadr. eine Routine im TRDOS ($1B7D $1B89) zuerst den Code $AA und dahinter die ermittelte Autostartadr.!

Wer beim LIST genau aufpaßt, wird die Suche bemerken, da jedesmal eine kleine Verzögerung auftritt (man hört das Laufwerk arbeiten) wenn beim Listen eines Basic-Prog. die Autostartadr. ermittelt wird.

Wenn man sich jedoch mal die aufgebaute Erweiterung ansehen will, muß man zu kleinen Tricks greifen! Also, erst einmal ein Disassembler-Prog. laden. Jetzt eine Diskette einlegen, die möglichst viele Basic-Prog. enthält und dann ohne vorangestellte Zeilennummer eingeben:

'RANDOMIZE USR 15619: REM: LIST'!

Nun erscheint die Auflistung der auf der Diskette befindlichen Progamme mit zugehörigen Parametern. Wenn jetzt irgendwann die letzte gelistete Zeile bevor die Frage 'scroll?' (Bild 1) erscheint ein Basic-Programm mit Autostartadresse enthält, einfach die 'BREAK'-Taste drücken und per 'RAMDOMIZE USR xxxxx' den Disassembler aufrufen.

Dann kann man sich einen Hex-Dump des Bereichs ab Adr. $5CB6 ausgeben lassen, und den Systemvariablen (=112 Zeichen-> Puffer) und den nur beim Suchen der Autostartadr. benutzten Bereich (liegt hinter einem Basic-Prog.!) mit einem Abschnitt des durchsuchten Programms ansehen. In Letzterem wird man dann auch irgendwo die oben erwähnte Code-Folge $80, $AA finden und dahinter die gesuchte Adresse als Hexadezimal-Zahl!

Dazu ein Beispiel: Als Code-Folge wird im beiliegenden Listing bei Adr. $6025 gefunden:

'$80, $AA, $48, $26'.

Die Codes $48 (=72) und $26 (=38) stellen hier die Autostartadresse dar in Low/High-Notierung. Berechnung: 38 * 256 + 72 = 9800. Dieser Wert muß also als letzte gefundene Autostartadr. gelistet worden sein!

Der Bereich mit der Katalog-Kopie ist bei dieser Methode leider nicht mehr vorhanden. Deshalb habe ich einen mit Hilfe des ISO-Monitor-ROMs (NMI-Taster) gewonnenen Ausdruck beigefügt, und ihn halbwegs restauriert !

Leider bleiben die Bereiche z.B. durch den nachfolgenden Disassembleraufruf nicht ganz unbeeinflußt (z.B. wird der Workspace dadurch verschoben)! Aber immerhin kann man sich so leicht ein Bild davon machen, wie das TRDOS arbeitet! Mir hat es jedenfalls viel geholfen, wenn ich nach der Analyse der entsprechenden Routinen das Theoretische in der Praxis bestätigt fand! Wie schon erwähnt, kann der Abbruch immer nur erfolgen, wenn z.B. eine Meldung wie 'scroll?' erscheint, denn dann ist vorübergehend das SPECTRUM ROM eingeschaltet und man gelangt ins Basic. Solange das BETA-ROM arbeitet, ist gewöhnlich kein Ausstieg ins Basic möglich.

```
Title: Disk 11      Disk Drive: A
102 File(s)         80 Track D. Side
0 Del. File(s)      Free Sector  122

   File Name        Start Length  Line
Compact  <B>   3    00758 00750   1
Compact  <C>   1    65368 00113
Renumber <B>   1    00152 00152   9000
Renumber <C>   5    60000 01140
Spectrac <B>   6    01481 01481   1
Spectrac <C>   2    64507 00363
Pulldown <B>   6    01478 01478   2000
Pulldown <C>   4    32768 00850
Scroll   <B>   9    02177 02177   1
Scroll   <C>   2    64800 00474
Extend   <B>   1    00154 00154   9995
Extend   <C>   8    61440 01836
Prodos   <B>  31    07771 07771   9980
Prodos   <C>  15    60000 03787
Header   <B>   7    01731 01731   9000
32*42    <B>   4    00948 00948   9800

scroll?
```

Zurück zum Geschehen um den Puffer! Wenn die Erweiterung >$5D25 nicht mehr benötigt wird, können die Systemvariablen für Pufferende ($5D00) wieder mit $5D25 (normales Ende des 112-Zeichenpuffers) und $5D0C mit $FF (Merker für entfernte Erweiterung) gefüllt werden! Zugleich wird natürlich ein im Speicher verschobenes Basic-Programm wieder zurück auf den alten Platz kopiert, sodaß man nichts mehr von alle dem bemerkt.
Für heute soll's genug sein. Dann also bis demnächst!

MfG

Wilhelm

Bild 1

```
5D25 43 6F 6D 70 61 63  Compac
5D2B 74 20 42 F6 02 EE  t B...
5D31 02 03 08 44 43 6F  ...DCo
5D37 6D 70 61 63 74 20  mpact
5D3D 43 58 FF 71 00 01  CX.q..
5D43 0D 44 52 65 6E 75  .DRenu
5D49 6D 62 65 72 42 98  mberB.
5D4F 00 98 00 01 0C 44  .....D
5D55 52 65 6E 75 6D 62  Renumb
5D5B 65 72 43 60 EA 74  erC`.t
5D61 04 05 0D 44 53 70  ...DSp
5D67 65 63 74 72 61 63  ectrac
5D6D 42 C9 05 C9 05 06  B.....
5D73 02 45 53 70 65 63  .ESpec
5D79 74 72 61 63 43 FB  tracC.
5D7F FB 6B 01 02 08 45  .k...E
5D85 50 75 6C 6C 64 6F  Pulldo
5D8B 77 6E 42 C6 05 C6  wnB...
5D91 05 06 0A 45 50 75  ...EPu
5D97 6C 6C 64 6F 77 6E  lldown
5D9D 43 00 80 52 03 04  C..R..
5DA3 00 46 53 63 72 6F  .FScro
5DA9 6C 6C 20 20 42 81  ll  B.
5DAF 08 81 08 09 04 46  .....F
5DB5 53 63 72 6F 6C 6C  Scroll
5DBB 20 20 43 20 FB BA    C ..
5DC1 01 02 0D 46 45 78  ...FEx
5DC7 74 65 6E 64 20 20  tend
5DCD 42 9A 00 9A 00 01  B.....
5DD3 0F 46 45 78 74 65  .FExte
5DD9 6E 64 20 20 43 00  nd  C.
5DDF F0 2C 07 08 00 47  .,...G
5DE5 50 72 6F 64 6F 73  Prodos
5DEB 20 20 42 5B 1E 5B    B[.[
5DF1 1E 1F 08 47 50 72  ...GPr
5DF7 6F 64 6F 73 20 20  odos
5DFD 43 60 EA CB 0E 0F  C`....
5E03 07 49 48 65 61 64  .IHead
5E09 65 72 20 20 42 C3  er  B.
5E0F 06 C3 06 07 06 4A  .....J
5E15 33 32 2A 34 32 20  32*42
5E1B 20 20 42 B4 03 B4    B...
5E21 03 04 0D 4A 00 FF  ...J..

5E51 06 98 16 66 7A 00  ...fz.
5E57 10 00 00 20 20 20  ...
5E5D 20 20 20 20 20 00      .
5E63 00 44 69 73 6B 20  .Disk
5E69 31 31 20 00 00 00  11 ...
5E6F 00 00 6F 64 65 72  ..oder
5E75 20 30 29 20 62 65   0) be
5E7B 6E 75 74 7A 74 20  nutzt
5E81 77 65 72 64 65 6E  werden
5E87 2E 20 46 4C 41 53  . FLAS
5E8D 48 2C 49 4E 4B 75  H,INKu
5E93 73 77 2E 20 73 69  sw. si
5E99 6E 64 20 6E 69 63  nd nic
5E9F 68 74 20 6E 75 74  ht nut
5EA5 7A 62 61 72 21 22  zbar!"
5EAB 0D 00 32 A4 00 F5  ..2...
5EB1 23 34 0E 00 00 04  #4....
5EB7 00 00 27 3B 22 44  ..';"D
5EBD 69 65 20 50 75 6E  ie Pun
5EC3 6B 74 6D 75 73 74  ktmust
5EC9 65 72 20 61 6C 6C  er all
5ECF 65 72 20 64 72 75  er dru
5ED5 63 6B 62 61 72 65  ckbare
5EDB 6E 20 5A 65 69 63  n Zeic
5EE1 68 65 6E 20 20 6D  hen  m
5EE7 69 74 20 64 65 6E  it den
5EED 20 43 6F 64 65 73   Codes
5EF3 20 76 6F 6E 20 33   von 3
5EF9 32 20 62 69 73 20  2 bis
5EFF 31 36 34 20 73 69  164 si
5F05 6E 64 20 61 62 20  nd ab
5F0B 64 65 72 20 20 41  der  A
5F11 64 72 65 73 73 65  dresse
5F17 20 36 34 37 33 32   64732
5F1D 20 61 62 67 65 6C   abgel
5F23 65 67 74 2E 20 44  egt. D
5F29 69 65 20 55 44 47  ie UDG
5F2F 27 73 20 48 20 62  's H b
5F35 69 73 20 55 20 73  is U s
5F3B 69 6E 64 20 6E 6F  ind no
5F41 63 68 20 6E 69 63  ch nic
5F47 68 74 20 62 65 6E  ht ben
5F4D 75 74 7A 74 2E 22  utzt."

5F53 0B 00 3C 04 00 EE  ..<...
5F59 66 24 0B 00 46 28  f$..F(
5F5F 00 FB 3A F5 23 34  ..:.#4
5F65 0E 00 00 04 00 00  ......
5F6B 3B AE 30 0E 00 00  ;.0...
5F71 00 00 00 2C 30 0E  ...,0.
5F77 00 00 00 00 00 3B  .....;
5F7D 3A F0 23 34 0E 00  :.#4..
5F83 00 04 00 00 0B 00  ......
5F89 64 02 00 E2 0B 26  d....&
5F8F 48 0D 00 FD 36 34  H...64
5F95 32 35 39 0E 00 00  259...
5F9B 03 FB 00 0B 26 52  ....&R
5FA1 1A 00 F9 C0 31 35  ....15
5FA7 36 31 39 0E 00 00  619...
5FAD 03 3B 00 3A EA 3A  .;.:.:
5FB3 EF 22 33 32 2A 34  ."32*4
5FB9 32 22 AF 0B 26 5C  2"..&\
5FBF 0E 00 F9 C0 36 34  ....64
5FC5 32 36 30 0E 00 00  260...
5FCB 04 FB 00 0B 26 66  ....&f
5FD1 02 00 F7 0B 27 06  ....'.
5FD7 1E 00 F9 C0 31 35  ....15
5FDD 36 31 39 0E 00 00  619...
5FE3 03 3B 00 3A EA 3A  .;.:.:
5FE9 F8 22 33 32 2A 34  ."32*4
5FEF 32 22 CA 39 38 30  2"J980
5FF5 30 0B 27 07 24 00  0.'.$.
5FFB F9 C0 31 35 36 31  ..1561
6001 39 0E 00 00 03 3B  9....;
6007 00 3A EA 3A F8 22  .:.:.'
600D 33 32 2A 34 32 22  32*42"
6013 AF 36 34 32 36 30  .64260
6019 2C 31 32 37 35 0B  ,1275.
601F 27 0F 02 00 E2 0B  '.....
6025 80 AA 48 26 39 30  ..H&90
602B 0E 00 00 06 27 00  ....'.
6031 0B 80 CA C1 61 05  ....a.
6037 00 00 00 C1 61 05  ....a.
603D 00 00 00 48 26 00  ...H&.
6043 00 00 00 00 00 00  ......
6049 00 00 00 00 00 00  ......
604F 00 00 00 00 00 00  ......
6055 00 00 00 00 00 00  ......
605B 00 00 00 00 00 00  ......
6061 00 00 00 00 00 00  ......
```

Von $5D25-$5E25 sind die zum Bild 1 gehörenden Katalog-Daten zu finden. Die Organisationssektor-Daten der verwendetem Diskette sind hier im Bereich $5E51-$5E6F erkennbar, dahinter folgen die letzten 2 Sektoren des Prog. "32*42" (bis $6070), die nach der erwähnten Byte-Folge durchsucht wurden (hier zu finden von $6025-$6028)!

Hallo Spectrum-User!

Weiter mit der Beschreibung der BETA-Disk-Vers.5.03! Erst geht's erst nochmal um den in der letzten Folge erklärten Puffer für Org.-Sektor- und Katalog-Kopie. Wie gesagt, beginnt er gewöhnlich bei Adresse $5D25 und ist 256 Bytes lang. Da er im Gegensatz zu dem Systemvariablenpuffer nicht ständig benötigt, muß er im Bedarfsfall aufgebaut und hinterher wieder beseitigt werden. 2 der neuen Systemvariablen lassen den Zustand erkennen: **$5D0C -> $00, wenn Erweiterung besteht und $FF, wenn gelöscht!**

Da häufig der erweiterte Bereich für mehrere Operationen hintereinander benötigt wird, gibt eine zweite Variable Auskunft darüber, wenn er erst noch bestehen bleiben soll. Dazu dient: **$5CF8 -> $00, wenn Löschen, $FF, falls Erhalt erforderlich!**

Dadurch wird vermieden, daß der Puffer z.B. beim Suchen im Katalog bei jedem zu ladenden Sektor zeitraubend (es muß ja auch ein eventuell vorhandenes Basic-Prog. im Speicher hin und hergeschoben werden um den nötigen Platz zu schaffen!) neu angelegt und entfernt wird. Durch Abfrage von $5CF8 ist dies dann meist nur noch zu Beginn und Schluß der Operation erforderlich! Hier nun das kommentierte Assemblerlisting:

```
294A           0010      org $294A       ;Aufrufadresse im TRDOS fuer Puffer-Erweiterungsroutine
               0020 ;
294A E5        0030      PUSH HL         ;Zuerst die Registerinhalte retten!
2942 D5        0040      PUSH  DE
2940 C5        0050      PUSH  BC
2945 F5        0060      PUSH  AF
294E 210C5D    0070      LD HL,$5D0C     ;HL-Reg. mit Adr. der Systemvariablen $5D0C laden
2951  7E       0080      LD  A,(HL)      ;und deren Inhalt ins A-Reg. bringen
2952  B7       0090      0R  A           ;Ist  er $00? Dann ist der Puffer  erweitert und Z-Flag gesetzt.
2953  283D     0100      JR  Z,L2992     ;Deshalb nach $2992, zum Beenden der Routine!
2955  E5       0110      PUSH  HL        ;Sonst HL-Reg. (=$5D0C) retten
2956  010101   0120      LD  BC,$0101    ;und BC-Reg. mit 257 (=Laenge der Erweiterung) laden!
2959 C5        0130      PUSH  BC        ;Auch diesen Wert auf den Stapel retten.
              0140   ;
              0150   ;Der folgende CALL-Befehl springt zu einen RST $20-Befehl, der eine
              0160   ;Routine bei $1FO5 in SPECTRUM-ROM aufruft. Dort wird getestet, ob
              0170   ;noch Platz fuer die 257 Zeichen ist. Wenn nicht, wird Fehlermeldung
              0180   ;'Out of menory' ausgegeben, sonst weiter!
              0190   ;
295A 0DFD19 0200        CALL $19FD      ;ROM-Routine $1FO5 (Test auf genug Speicherplatz) aufrufen
295D CI      0210       POP  BC         ;dann Zeichenzahl ($0101)
295E El      0220       POP  HL         ;und $5DCC zurueckholen.
295F 3600    0230       LD   (HL),$00   ;Nun $5D00 mit 0 laden, da Erweiterung vorgenommen wird!
2961 21255D  0240       LD   HL,$5D25   ;Adr. ab der Erweiterung erforderlich  ins HL-Reg. laden.
             0250   ;
             0260   ;Wieder wird eine ROM-Routine ($1655) per RST $20 aufgerufen! Hier
             0270   ;wird der benoetigte Speicherplatz geschaffen. Dazu muss BC die Anzahl
             0280   ;und HL die Adr. ab der Platz geschaffen werden soll, enthalten!
             0290   ;
2964 CD321E  0300       CALL $1E32      ;ROM-Routine $1655 (schafft benoetigten Platz) aufrufen!
2967 2A115D  0310       LD   HL,($5D11) ;Augenblickliche BASIC-Anfangsadr.Ins HL-Reg.bringen
296A 010101  0320       LD   BC,$0101   ;und Pufferlaenge (=257) ins BC-Reg.!
296D 09      0330       ADD  HL,BC      ;Beide Werte addieren. In HL  steht  nun neuer BASIC-Anfang!
296E 181F    0340       JR   L298F      ;Weiter bei dieser Adresse!
             0350   ;
             0360   ;Hier beginnt die Routine zum Loeschen der Erweiterung. Sie wird nur
              0370  ;aufgerufen, wenn der Bereich nicht sehr benoetigt wird. Dazu wurde
              0380  ;vorher die Systemvariable $5CF8 auf 0 getestet (bei $20E6).
              0390  ;
2970   E5    0400       PUSH HL         ;Zuerst wieder die Register retten!
2971   D5    0410       PUSH  DE
2972   C5    0420       PUSH  BC
2973   F5    0430       PUSH  AF
2974   210C5D 0440       LD   HL,$5D0C   ;HL-Reg. Mit Adr. der Systemvariablen $5D0C laden
2977   7E    0450       LD   A,(HL)     ;und deren Inhalt ins A-Reg. bringen!
2978   B7    0460       0R   A          ;Ist er $00? Dann ist Puffer noch erweitert und Z-Flag gesetzt!
2979   2017  0470       JR   NZ,L2992   ;Wenn nicht,nach  $2992, da dann nichts zu entfernen  ist!
297B 36FF    0480       LD   (HL),$FF   ;Nun $5D0C mit $FF laden, da Erweiterung geloescht wird!
297D 21255D  0490       LD   HL,$5D25   ;Adr. ab der Erweiterung entfernt werden soll in HL laden.
2980 010101  0500       LD   BC,$810i   ;Anzahl der zu entfernenden Zeichen ins BC-Reg. laden.
2983 CD2E1E  0510       CALL  $!E2E     ;ROM-Routine $19E8 (entfernt Platz)  per  RST $20 aufrufen.
2986  B7     0520       0R   A          ;Flags besehen!
2987   010101 0530       LD   BC,$C101   ;Nochmal 257 ins BC-Reg. schreiben
298A 2A115D  0540       LD   HL,($5B11) ;und augenblickliche BASIC-Anfangsadr. ins HL-Reg.,
298D ED42    0550       SBC  HL,BC      ;nun BASIC-Anfangsadr. minus 257 = neuer BASIC-Anfang in HL!
298F 22115D  0560 L298F  LD   ($5D11),HL ;Diesen in Systemvariable $5D11 zurueckschreiben.
2992   F1    0570 L2992 POP   AF         ;Jetzt die urspruenglichen Registerinhalte zurueck holen.
2993   C1    0588       POP   BC
2994   Dl    0590       POP   DE
2995   El    06GO       POP   HL
2996   C9    0610       RET             ;Routine ist beendet, daher zurueck!

#      61A8 L298F  298F L2992  2992
```

Noch ein paar Erläuterungen zum Katalog-Inhalt, erklärt am abgedruckten Beispiel aus der letzten Folge! Dort steht für den ersten Eintrag (jeder ist 16 Bytes lang) ab $5D25 folgendes:

```
$5D25 43 6F  6B 70  61  63 Compac
$5D2B 74 20  42 F6  02  EE t B...
$5D31 02 03  08 44      ... D
```

Wie unschwer zu sehen, enthalten die ersten 8 Bytes den Prog.-Namen ($5D25 - $5D2C), der bekanntlich maximal 8 Zeichen lang sein darf. $5D2D enthält den Code für den Progamm-Typ. Hier gibt es 4 Varianten:

```
$42 = "B" = BASIC-Prog.
$43 = "C" = CODE,
$44 = "D" = DATA
$23 = "#" = Sequentielle / Random-Access-Fi1 es
```

$5D2E-$5D2F enthält in der üblichen Low/High-Form bei CODE die Startadresse des gespeicherten MC-Programms, bei DATA die Länge des Feldes und bei BASIC-Prog. die Länge inklusive Variablen!

Berechnung im Beispiel: 2 * 256 + 246 = 758

In $5D30-$5D31 ist bei CODE-Files die Gesamtlänge des Codes verborgen und bei BASIC-Prog. die Länge ohne Variablen.

Berechnung im Beispiel: 2 * 256 + 238 = 750

Die nächste Stelle ($5D32) gibt Auskunft über die Anzahl der vom Programm belegten Sektoren (jeder Sektor 256 Bytes lang)!

Berechnung im Beispiel: 758 / 256 = 2,96 = 3 Sektoren

Dazu muß gesagt werden, daß der letzte benutzte Sektor zwar nicht immer ganz belegt wird (hier sind z.B. 10 Bytes nicht belegt -> 3 * 256 = 768 - 758 = 10), aber der Rest ist für andere Files nicht mehr verfügbar. Es wird also immer auf volle Sektorenzahlen aufgerundet. Die frei bleibenden Stellen auf der Diskette werden mit 0 gefüllt und beim LOAD nicht mit geladen, damit nichts überschrieben wird.
In $5D33 findet man hier die Sektor- und in S5D34 die Track-Nummer in welcher der File beginnt!

Sektor-Nr. → $08 = Sektor 8 Track-Nr. → $44 = Track 68

Wer will, kann an Hand des Beispiels aus der letzten Folge auch die Werte der weiteren Programme errechnen und mit Bild 1 vergleichen oder eigene Kataloge mit Hilfe des mitgelieferten Disk-Doctor-Programms auslesen und die Angaben prüfen!

Genug von diesem Thema und zu einem anderen Kapitel! Auch das TRDOS verwendet ebenso wie das normale SPECTRUM-Betriebssystem (SOS = SPECTRUM-Operation-System) häufig die platzsparenden RST-Befehle. Bekanntlich handelt es sich dabei im Prinzip um eine verkürzte Variante des CALL-Befehls (’CALL’ ist vergleichbar mit dem ’GO SUB’ im Basic)! Der RST-Befehl erlaubt bei nur einem Byte Platzbedarf den Sprung zu einigen vom Z80-Befeh1satz festgelegten Adressen im unteren Adressbereich. Grundsätzlich sind dies folgende Adressen:

Befehl	Code	Hex-Adr.	Dez-Adr.	Bedeutung im SOS
RST $00	$C7	$0000	0	Start des Betriebssystems
RST $08	$CF	$0008	8	Fehlerbehandlung
RST $10	$D7	$0010	16	Ein Zeichen im A-Reg. ausgeben
RST $18	$DF	$0018	24	Aktuelles Zeichen holen/testen
RST $20	$E7	$0020	32	Nächstes Zeichen holen
RST $28	$EF	$0028	40	Sprung in die Rechenroutine
RST $30	$F7	$0030	48	Platz im Workspace reservieren
RST $38	$FF	$0038	56	Interrupt-Routine

Im TRDOS werden nun nicht alle RST-Befehle verwendet und außerdem haben sie größtenteils eine neue Bedeutung!

Befehl — Bedeutung im TRDOS

RST $00 -> Start des TRDOS, wird nur beim Einschalten oder Reset angesprungen. Im TRDOS sonst nicht verwendet!

RST $08 -> Taucht im TRDOS nur bei $3D44 auf und aktiviert das IF 1. Wenn kein IF 1 angeschlossen ist, wird dieser Befehl im TRDOS nie erreicht (siehe Ass.-Listing vom 1.6.89, Zeile 900).

RST $10 -> Bewirkt ein ’JP $3D82’. Dort wird im Wesentlichen auf IF 1 getestet und per RST $20 in die RST $10-Routine des SOS gesprungen und ein Zeichen im A-Reg. ausgegeben. Danach geht’s zurück ins TRDOS zur dem RST $10-Aufruf folgenden Adresse.

RST $18 -> Bewirkt ein ’JP $2707’. Hier werden Texte aus im TRDOS stehenden Tabellen ausgegeben mit Hilfe der obigen RST $10-Routine. Beendet wird die Ausgabe, wenn ein Code >127 (inverse Zeichen) oder $00 erkannt wurde.

RST $20 -> Bewirkt ein 'JP $2F72'. Ein Sprung zu. der dem RST $20-Befehl folgenden SPECTRUM-ROM-Adresse wird durchgeführt und zur nachfolgenden Adresse zurückgekehrt.

RST $28 -> Bewirkt ein 'JP $2323'. Holt die aktuelle I/O-Info- Adresse usw. wird meines Wissens nur in Verbindung mit Sequentiellen- oder Random-Access-Files verwendet.

RST $30 -> Im TRDOS nicht existent!

RST $38 -> Enthält die Befehle 'DI, RET'! Schaltet den Interrupt ein und kehrt zur nächsten Adresse zurück. Da der Code dieses Befehls $FF ist, taucht er immer da im TRDOS auf, wo leere EPROM-Bereiche (enthalten bekanntlich $FF) sind.

Ste11vertretend für alle anderen RST $20-Befehle nun ein Beispiel! Es ist der im obigen Assemblerlisting in Zeile 220 aufgerufene Speicherplatztest:

```
19FD          0000      org  $19FD      ;Anfangsadr. einer RST $20-Routine im TRDOS
              0010 ;
19FD E7       0020      RST  $20        ;Bewirkt Sprung nach $0020, dort steht JP $2F72. Dort weiter!
19FE          0030      DB   $05        ;Gleichzeitig kommt Adr. dieser Bytes als Ruecksprungadr. Auf
19FF          0040      DB   $1F        ;Stapel! (Enthalten Adr. einer SPECTRUN-ROM-Rout. (hier $1FO5))
1A00 C9       0050      RET             ;Rueckkehr zur Adr. hinter dem Aufruf-Befehl!

#    61A8
```

Und jetzt das Listing der eigentlichen RST $20-Routine. In Verbindung mit den beiden anderen läßt sich sehr schön verfolgen, wie das Abarbeiten von SPECTRUM-ROM-Routinen aus dem TRDOS heraus und die Rückkehr dorthin abläuft. Da dies in gleicher Weise bei allen anderen BETA-Versionen und wahrscheinlich auch bei anderen Interfaces verläuft, habe ich jeden Schritt ausführlich kommentiert. So ist es hoffentlich für jeden, der sich etwas mit Maschinensprache auskennt, verständlich!

```
2F72          0010      org  $2F72      ;Anfangsadr. der RST $20-Bearbeitung im TRDOS
              0020 ;
2F72 22025D   0030      LD   ($5D02),HL ;HL-Reg.-Inhalt und
2F75 ED53045D 0040      LD   ($5D04),DE ;DE-Reg.-Inhalt im 112-Zeichenpuffer zwischenspeichern!
              0050 ;
              0060 ;Mit dem folgenden POP-Befehl wird nun die zuletzt auf den Stapel ge-
              0070 ;legte Adresse ins HL-Reg. gebracht. Dabei handelt es sich zwangs-
              0080 ;laeufig um die dem RST-$20-Aufruf folgende, da diese ja zuletzt als
              0090 ;Ruecksprungadr. gerettet wurde (im Beispiel: $19FE)!
              0100 ;
2F79          0110      POP HL          ;Oberste Adr. vom Stapel ins HL-Reg.
2F7A          0120      LD E,(HL)       ;Inhalt dieser Adr. ins E-Reg. umladen (im Beispiel: $05).
2F7B 23       0130      INC HL          ;HL-Reg. um 1 erhoehen (im Beispiel auf: $19FF)
2F7C 56       0140      LD D,(HL)       ;Inhalt dieser Adr. ins D-Reg. umladen (im Beispiel: $1F)
2F7D 23       0150      INC HL          ;HL-Reg. erneut erhoehen (im Beispiel nun auf:$1A00)
2F7E E5       0160      PUSH HL         ;und auf Stapel ablegen (zeigt im Beispiel auf $C9  = RET)
2F7F 212F3D   0170      HL,$3D2F        ;Adr. im Einschaltbereich des TRDOS (zeigt auf RET-Befehl)
2F82 E5       0180      PUSH HL         ;Auch diese Adresse auf den Stapel ablegen!
2F83 D5       0190       PUSH DE        ;Ebenso DE-Reg.! (Enthaelt im Beispiel hier: $1FO5)
2F84 21C25C   0200      LD HL,$5CC2     ;HL mit dieser Adr. laden!
2F87 E5       0210      PUSH HL         ;Auch auf Stapel! $5CC2 zeigt auf einen RET-Befehl im RAM)
2F88 2A025D   0220      LD HL,($5D02)   ;HL- und DE-Reg.
2F8B ED5B045D 0230      LD DE,($5D04)   ;nun wieder mit den ursprünglichen Werten laden.
2F8F C9       0240      RET             ;Ruecksprung!
              0250 ;
              0260 ;Der obige RET-Befehl geht natuerlich aus des TRDOS zur obersten Adr.
              0270 ;des Stapels, ins RAM nach $5CC2. Hier steht erneut RET, was zur
              0280 ;naechsten Stapeladresse fuehrt. Diese enthaelt im Beispiel die ROM-
              0290 ;Adr. $1FO5, also wird dort weitergemacht (auf benoetigten Speicher-
              0300 ;platz testen).Auch diese Routine endet irgendwann mit RET. Nun liegt
              0310 ;die Adr. $3D2F als naechste auf dem, Stapel und wird angesprungen! Da
              0320 ;diese im Einschaltbereich des TRDOS liegt wird dies aktiviert. Diese
              0330 ;zeigt wieder auf ein RET, womit die naechste Adr. vom Stapel erreicht
              0340 ;wird! Im Beispiel ist dies $1A00, die hier auch nur ein RET enthaelt.
              0350 ;Dies fuehrt dann zur naechsten Adr., an der das TRDOS weitermacht!
              0360 ;Im Beispiel koennte dies dann die Adr. S295D sein, die dem Aufruf
              0370 ;der RST $2C-Routine wit CALL $19FD folgt!

#     61A8
```


Wie man sieht, ist etwas Stapelakrobatik erforderlich, um vom TRDOS über den Umweg im RAM endlich ins SPECTRUM-ROM und wieder zurück zu gelangen! Das Original-ROM wird also immer nur vom RAM her eingeblendet, nie direkt vom BETA-ROM aus. Dieses kann dagegen nur durch Aufruf einer Adresse seines Einschaltbereichs (von 14431-15871!) eingeblendet werden. Niemals sind beide ROM's gleichzeitig eingeschaltet!

Zum Schluß noch was auch für Nicht-BETA-User! Wer das Einschaltbild der BETA-Disk kennt, wird sich schon mal gefragt haben, wie der schwarze Balken mit den schrägen farbigen Streifen (siehe nächste Seite) im Titel erzeugt wird. Auch beim SPECTRUM-128 wird dieser Effekt meines Wissens oft verwendet. In BASIC ist so was ja auch

möglich, aber in MC ist es schneller und platzsparender! Deshalb will ich auch noch diese kleine Routine beschreiben. Für den Gebrauch in eigenen Programmen läßt sie sich problemlos für jede Laufadresse anpassen.

```
106E          0010      org $106E       ;Startadr. der Farbbalkenroutine im TRDOS 5.03
              0020 ;
              0030 ;Schwarzen Balken von 10 Zeichen Laenge ab Zeile 7 / Spalte 5 erzeugen!
              0033 ;
106E 21E558   0040      LD   HL,$58E5    ;Adr. im Attribut-Bereich in HL laden (entspr.'AT 7,5'1
1071 060A     0050      LD   B,$0A       ;B-Reg. dient als Zaehler fuer Ausgabe von 10 Attributwerten
1073 3607     0060 loop1LD   (HL),$07    ;Attribut 7 setzen (PAPER 0 (schwarz), INK 7 (weiss)).
1075  23      0070      INC  HL          ;HL auf naechste Attribut-Adresse setzen
1076  10FB    0080      DJNZ  loop1      ;und so weiter bis alle 10 Felder fertig (d.h.B = 0 ist!)
              0090 |
              0100 ;Der Attributwert ist binaer wie folgt codiert:
              0110 ;
              0120 ;Bit 7    : 1 = FLASH ein / 0 = FLASH aus
              0130 ;Bit 6    : 1 = BRIGHT ein/ 0 = BRIGHT aus
              0140 ;Bit 3-5  : PAPER-Farbe
              0150 ;Bit 0-2  : INK-Farbe
              0160 ;
              0170 ;Nun kommt der Bereich, in dem die 4 farbigen schraegen Balken
              0180 ;entstehen sollen (ab 'AT 7,15').Dazu werden zunaechst die naechsten
              0190 ;Attribute gesetzt!
              0200 ;
1078 3602     0210      LD   (HL),$02    ;PAPER 0 (schwarz), INK 2 (rot)
107A 23       0220      INC   HL         ;Naechstes Attribut-Feld!
107B 3616     0230      LD   (HL),$16    ;PAPER 2 (rot),     INK 6 (gelb)
107D 23       0240      INC   HL         ;Weiter!
107E 3634     0250      LD   (HL),$34    ;PAPER 6  (gelb) ,  INK 4(gruen)
1080  23      0260      INC   HL         ;Weiter!
1081  3625    0270      LD   (HL),$25    ;PAPER 4  (gruen),  INK5(cyan)
1083  23      0280      INC   HL         ;Weiter!
1084  3628    0290      LD   (HL),$28    ;PAPER 5  (cyan) ,  INK0(schwarz)
1086  23      0300      INC   HL         ;Weiter zum letzten Feld!
1087  3607    0310      LD   (HL),$07    ;PAPER(schwarz), INK 7 (weiss)
              0320 ;
              0330 ;Jetzt werden die gesetzten farbigen Attribut-Felder mit einem Punkt-
              0340 ;Muster Pixelreihe fuer Pixelreihe beschrieben, wobei die Punkte in
              0350 ;der jeweiligen INK-Farbe erscheinen!
              0360 ;
1089  21EE40  0370       LD  HL,$40EE    ;Screenadr. der 1. Pixelzeile/spalte des  Attr. $02 laden
108C 0608     0380       LD  B,$08       ;B dient als Zaehler fuer  die8 Pixelreiheneines Zeichens
108E AF       0390       XOR  A          ;A-Reg. auf 0 setzen.
108F C5       0400 loop3 PUSH  BC        ;Zaehler retten!
1090  37       0410       SCF            ;Uebertrags-Flag (= C-Flag) auf 1 setzen
              0420 ;
              0430 ;Mit den folgenden RLA-Befehl wird das C-Flag ins Bit 0 und Bit 7 ins
              0440 ;C-Flag rotiert.Da C-Flag bei jedem Durchlauf neu gesetzt wird,
              0450 ;entstehen nacheinander die Bitmuster von 1,3,7,15,31,63,127 und 255!
              0460 ;Ausgangssituation vorm RLA-Befehl: C-Flag = 1, A-Reg. = 00000000
              0472 ;
              0474 ;Situation nach x. Burchlauf jeweils:
              0475 ;
              0480 ;                   C-Flag Bit 7-0 Beziwal
              0482 ;nach 1. RLA-Befehl:  1    00000001   =1
              0484 ; '   2.    '         1    00000011   =3
              0490 ; '   3.    '         1    00000111   =7
              0500 ; '   4.    '         1    00001111   =15
              0510 ; '   5.    '         1    00011111   =31
              0520 ; '   6.    '         1    00111111   =63
              0522 ; '   7.    '         1    01111111   = 127
              0524 ; '   8.    '         1    11111111   = 255
              0530 ;
1091  17      0540       RLA             ;= A-Reg. nach links durchs C-Flag rotieren zum Punktmuster bilden
1092  E5      0550       PUSH  HL        ;Zeilenadresse retten.
1093  F5      0560       PUSH  AF        ;A-Reg. retten (enthaelt Wert des jeweiligen Punktmusters
1094  0605    0570       LD   B,$05      ;Zaehler fuer 5-malige Bitmuster-Ausgabe
1096  23      0580 loop2 INC HL          ;Naechste Screen-Position einstellen.
1097  77      0590       LD   (HL),A     ;Muster auf Bildschirm in der jeweiligen INK-Farbe ausgeben!
1098  10FC    0600       DJNZ  loop2     ;Zurueck, bis das Bitmuster auf allen 5 Positionen steht.
109A  F1      0610       POP AF
109B  E1      0620       POP HL          ;Register-Inhalte zurueckholen vom Stapel
109C  C1      0630       POP BC
109A  110001  0640       LD DE,$0100     ;DE-Register mit 256 laden!
10A0  19      0650       ADD  HL,DE      ;Zeilenanfangsadr. * 256  =Anfang der neuen Zeile!
10A1  10EC    0660       BJNZ  loop3     ;Und weiter, bis alle 8 Bitmuster ausgegeben wurden!
10A3 C9       0670       RET             ;Danach Ruecksprung!

#       61A8 loop1 1073 loop3 108F loop2 1096
```

Hier nun der eben beschriebene Farbbalken, leider nur in Scharz/Weiß-Darstellung, aber ich meine, man kann erkennen worum es geht! Darunter habe ich nun die Punktmuster gedruckt, die auf den entsprechenden Attribut-Feldern dann in der jeweiligen INK- Farbe dargestellt werden (zum Vergleich siehe Assemblerlisting Zeile 482-524). Der Text 'BETA 128' ist hier natürlich Bestandteil des Titeltextes, der vom TRDOS aus einer Tabelle ($0360- $03AB) auf den Screen gebracht worden ist per RST $18-Routine.

Die ganze Tite1darste11ung wird übrigens im TRDOS bei Adresse $0259 gestartet, beginnend mit einem CLS und dem Öffnen von Kanal 2 (beides natürlich mit Hilfe der beschriebenen RST $20-Routine!), dann der Textausgabe und schließlich bei $0263 dem Aufruf der Farbbalken-Erzeugung! Ob der Titeltext erscheint, oder nur die beiden unteren Zeilen gelöscht werden und der Rest des Screens erhalten bleibt, ist vom Inhalt der Systemvariablen $5D17 (-23831) abhängig. Nur wenn sie <> $AA (=170) ist, wird der komplette Titel ausgegeben!

Will man das Ganze an einer anderen Screen-Position ausdrucken, muß man nur die Anfangsadr. der gewünschten Pixel-Zeile/Spalte errechnen und einfügen und natürlich auch die entsprechende Attributadresse. Aber darauf will ich hier nicht näher eingehen, da die Organisation des Bildschirms beim SPECTRUM und die Pixeladressberechnung schon mehrfach im Info erklärt wurde (zuletzt sehr gut im Info 4/89, Seite 15-16)! Ich meine, mit dieser Routine kann man bei geringem Platzbedarf einem Titel oder Menu ein interessanteres Äußeres geben.

Noch etwas anderes! Im Info 8/89 gab es einen sehr positiven Erfahrungsbericht zum Assembler/ Disassembler 'SYS'! Ich kann dem Bericht nur zustimmen! Es ist wirklich ein hervorragendes Programm und jedem, der sich ernsthaft mit Maschinensprache beschäftigen möchte, wärmstens zu empfehlen!

Ich selbst arbeite seit ca. 2 1/2 Jahren damit (allerdings noch eine Version 2.2) und ich muß sagen, es läßt kaum Wünsche offen. Alle hier ausgedruckten Assemblerlistings sind mit dem Assembler 'Micass' entstanden und ohne den hervorragenden Disassembler hätte ich mehr Probleme beim Entwirren und Verstehen von MC-Programmen gehabt! Besonders der sehr gut arbeitende Einzelschrittmodus (TRACE) ist oft eine große Hilfe!

Vielleicht noch zur Ergänzung: als Textverarbeitungs-Programm benutze ich den Oldtimer 'TASWORD II' in einer überarbeiteten und erweiterten Version und bin im Großen und Ganzen zufrieden damit, auch wenn es einen Schönheitsfehler gibt, der gelegentlich stört. Damit meine ich die Lücken am Zeilenende, die beim Gebrauch von **Steuerzeichen** (Grafikzeichen) in Verbindung mit formatiertem Ausdruck entstehen. Ich habe mich bisher damit zwar noch nicht weiter befaßt, aber vielleicht hat jemand eine Lösung parat?

Noch eine Frage! Wer kann mir mal eine Kopie des Betriebssystems einer BETA-Version 4.xx schicken (auf Kassette oder 3.5"-Disk/40 oder 80 Tracks)? Ich möchte nämlich versuchen, auch diese Version zu disassemb1ieren um festzuste11en, welche Gemeinsamkeiten und Unterschiede zur Version 5.03 auftreten. Wenn möglich, möchte ich mir auch gerne die zugehörige Anleitung kopieren. Wozu das Ganze?

Da ich in einer der nächsten Folgen das Ansprechen des TRDOS aus MC-Programmen heraus beschreiben möchte, ist es natürlich ganz gut, wenn man auch gleich Tipps zur weit verbreiteten Version 4.xx einbauen kann. Unter anderem will ich versuchen, zu zeigen, wie man Programme, die für's IF 1 mit Microdrives geschrieben sind und diese aus MC ansprechen, auf BETA-Disk umgeschrieben werden können!

Zur Zeit arbeite ich z.B. noch an einer 'ART- STUDIO'-Version für BETA-Disk 5.03, bei der unter anderem verschiedene Untermenus normalerweise von Microdrive nachgeladen werden. Hierfür ist eine BETA-Lösung schon getestet und auch bei den SAVE- LOAD- und CAT-Befehlen müßte der Umbau gelingen, obwohl es momentan speziell beim CAT noch viel Kleinarbeit gibt!
Schluß für heute! Bis demnächst!

MfG
Wilhelm