MZ-80B - intern Nr. 16

Dieses MZ-80B-intern stellt den Benutzern von BASIC SB-5510 und BASIC SB-6510 einige neue Tricks und Hilfsprogramme vor. Zuerst etwas für diejenigen, die auch noch das letzte Byte aus Ihrem Speicher holen wollen. Beide Interpreter prüfen nur bis $FF00 auf vorhandenen Speicher, der darüber liegende Raum von 255 Byte bleibt in jedem Falle ungenutzt. Das ist natürlich ideal, um dort kleine Maschinenprogramme abzulegen, die nicht überschrieben werden können. Andererseits kann man diesen Bereich mit POKE $49C8,$FF: POKE $49C9,$FF: LIMIT MAX in BASIC SB-5510 und mit POKE $6008,$FF: POKE $6009,$FF: LIMIT MAX in BASIC SB-6510 dem BASIC-Interpreter zugänglich machen. Die POKE-Befehle verändern die absolute obere Speichergrenze, die von LIMIT MAX benutzt wird.

In einer früheren Ausgabe der MZ-80B intern (Nr. 10) war eine Möglichkeit zur Abschaltung der BREAK-Taste angegeben, die in bestimmten Fällen zu Problemen führte. Eine bessere Lösung sind folgende Änderungen: POKE $0569,$C9 ($E6): POKE $0776,$0D ($0B): POKE $0FE3,$00 ($D8). Sie gelten für beide Interpreter, die Werte in den Klammern stellen den alten Zustand wieder her.

Ein anderer Fehler findet sich im BASIC SB-6510. Dieses führt ja nach dem Laden automatisch das Programm AUTO RUN aus. Wenn dieses nicht vorhanden ist, soll natürlich keine Fehlermeldung erzeugt werden, also wurde am ON ERROR GOTO etwas geändert. Dies führt dazu, daß ON ERROR GOTO im AUTO RUN-Programm verwendet nicht richtig funktioniert. Abhilfe schaffen die Befehle POKE $14DC,$00: POKE 14DD,$00: POKE $14DE,$00 am Anfang des AUTO RUN-Programm. Damit funktioniert dann ON ERROR GOTO wie gewohnt.

Leider führen beide Interpreter beim Vergleich von Strings zuerst eine Längenprüfung durch, die bei ungleicher Länge den Ausschlag für den Vergleich gibt. Dies ist aber z. B. in Sortierprogrammen unerwünscht. Mit POKE $2358,$AF ($B8) in BASIC SB-5510 und POKE $2509,$AF ($B8) in BASIC SB-6510 wird die Längenprüfung unwirksam gemacht. Damit ändern sich Stringvergleiche in folgender Weise:

vor der Änderung	nach der Änderung	
"BB"<"AAA"	"BB">"AAA"	
"AA"<"AAA"	"AA"<"AAA"	
"AB"<"AAA"	"AB">"AAA"	
"1.11">"1.2"	"1.11"<"1.2"	(auch numerisch richtig!)

Beide Interpreter warten leider sehr lange auf den Drucker, bevor sie die Fehlermeldung ERROR 65 (Drucker nicht bereit) erzeugen. Diese Wartezeit läßt sich durch Ändern der Speicherstelle $4707 (BASIC SB-5510) bzw. $533C (BASIC SB-6510) verkürzen. Normalerweise steht dort $0F, jede kleinere Zahl verkürzt die Wartezeit. Dabei sollte allerdings kein zu kleiner Wert gewählt werden, weil dann eventuell während der Zeit, in der der Drucker wegen des gefüllten Buffers keine Zeichen mehr annimmt, die genannte Fehlermeldung erzeugt wird. Beim Drucker NEC PC-8023B-N hat sich $02 als kleinstmöglicher Wert herausgestellt.

In BASIC SB-6510 kann das aktuelle Disketten-Laufwerk nur mit DIR geändert werden, festgestellt kann es gar nicht werden. Dies ist mit einem POKE- oder PEEK-Befehl aber kein Problem. Mit PEEK($45F3) erhält man das aktuelle Laufwerk (0=1. Laufwerk, ..., 3=4. Laufwerk), mit POKE $45F3,0-3 kann man es verändern.

Nun zu einigen Hilfsprogrammen. Leicht passiert es unter BASIC SB-6510 leicht, daß ein File versehentlich gelöscht wird. Das folgende Programm stellt solch ein File wieder her. Es zeigt alle gelöschten Files der Diskette des angegebenen Laufwerks an und fragt, ob die Datei wiederhergestellt werden soll. Falls ja, wird nach der Dateiart gefragt, weil diese nach dem Löschen durch die Löschmarkierung überschrieben wurde. Damit ist die Datei wieder verfügbar.

Die Anwendung dieses Programm verlangt einige unbedingt einzuhaltende Regeln. Erstens sollte es ausschließlich auf eine Kopie (mit BACKUP des Disk-Handling, s. Katalog, herstellen) der zu bearbeitenden Diskette angewandt werden, weil die Verwaltung der belegten Sektoren zerstört wird. Zweitens sollte es nur auf Disketten angewandt werden, auf die nach dem Löschen des Files nicht mehr geschrieben wurde, weil dann eventuell das gelöschte File mit den danach aufgezeichneten Daten schon überschrieben ist. Aber dies kann man ja ausprobieren (an der Kopie selbstverständlich!). Drittens sollten mit dem Programm bearbeitete Disketten hinterher auf keinen Fall weiter benutzt werden (die Kopie am besten Löschen!), weil wie schon erwähnt die Verwaltung des Diskettenraums nicht mehr funktioniert. Dies stört beim Lesen nicht, sehr wohl aber beim Schreiben!

```
100 PRINT "Wiederholen gelöschter Dateien"
110 INPUT "Laufwerk-Nummer (1-4): ";A
120 C=16:K=61696: G=0: V=0: LIMIT $EFE0
130 POKE $EFF0,$DD:POKE $EFF1,$21:POKE $EFF2,$00:POKE $EFF3,$F0:POKE $EFF4,$C3:
POKE $EFF5,$00:POKE $EFF6,$5D
140 POKE $EFE0,$DD:POKE $EFE1,$21:POKE $EFE2,$00:POKE $EFE3,$F0:POKE $EFE4,$C3:
POKE $EFE5,$03:POKE $EFE6,$5D
150 POKE $F000,A-1:POKE $F001,C:POKE $F002,0:POKE $F003,0:POKE $F004,1:POKE $F0
05,0:POKE $F006,$F1
160 USR($EFF0)
170 FOR I=0 TO 7
180 IF (C=16)*(I=0) GOTO 340
190 X=K+I*32
200 IF (PEEK(X)<5)*(PEEK(X)>0) GOTO 340
210 IF (PEEK(X+1)=0)+(PEEK(X+1)=229) GOTO 340
220 Y=X: G=1: N$=""
230 X=X+1:IF PEEK(X)<>13 THEN N$=N$+CHR$(PEEK(X)): GOTO 230
240 PRINT "Datei "+N$+" wiederherstellen (J/N)?";: INPUT " ";A$
250 IF A$<>"J" GOTO 340
260 V=1
270 PRINT "Handelt es sich um eine OBJ-, BTX-, BSD- oder BRD-Datei?";
280 INPUT " ";A$
290 IF A$="OBJ" THEN POKE Y,1: GOTO 340
300 IF A$="BTX" THEN POKE Y,2: GOTO 340
310 IF A$="BSD" THEN POKE Y,3: GOTO 340
320 IF A$="BRD" THEN POKE Y,4: GOTO 340
330 GOTO 260
340 NEXT I
350 IF V=0 GOTO 380
360 V=0
370 USR($EFE0)
380 C=C+1: IF C<48 GOTO 150
390 USR($5D06)
400 IF G=0 THEN PRINT "Keine gelöschten Dateien vorhanden!"
```

Ausführliche Programmnamen unterstützen zwar die Verständlichkeit der Direktory, machen aber beim Eintippen unnötig viel Arbeit. Die folgenden POKE-Befehle ändern das BASIC SB-6510 so ab, daß bei allen Diskettenbefehlen die erste mit den beim Befehl angegebenen Buchstaben übereinstimmende Datei benutzt wird. Zum Beispiel lädt der Befehl LOAD "M" das erste mit M beginnende Programm in den Speicher (z. B. 'Musikprogramm'), DELETE "Mys" löscht die erste Datei der Direktory, deren Namen mit 'Mys' beginnt (Vorsicht!!!). In den Klammern die Werte zum Wiederherstellen des Urzustandes.

```
POKE $4D2F,$FE ($BE)     POKE $4D32,$A8 ($23)     POKE $4D35,$05 ($0D)
POKE $4D30,$0D ($20)     POKE $4D33,$BE ($13)     POKE $4D36,$23 ($28)
POKE $4D31,$28 ($09)     POKE $4D34,$20 ($FE)     POKE $4D37,$13 ($03)
```

Programme zum Kopieren von Maschinen- oder BASIC-Programmen sowie von ganzen Disketten gibt es schon lange, aber sequentielle Dateien können einzeln bisher nicht kopiert werden. Dies besorgen die beiden folgenden Programme, eines für Kassette und eines für Diskette. Beide Programme machen sich zunutze, daß der Interpreter aufgezeichnete Zahlen auch in Stringvariablen einliest und als Strings aufgezeichnete Zahlen auch in Zahlenvariablen einliest. Die Programme lesen also die angegebene Datei in ein Stringfeld, unabhängig von den Bestandteilen der Datei, und speichern dieses Stringfeld wieder ab, identisch zur Quelldatei. Da BASIC SB-5510 keine Möglichkeit bietet, das Dateiende zu erkennen, muß auf die entsprechende Fehlermeldung gewartet werden. Da man auf Kassette auch nicht gleichzeitig zwei Dateien bearbeiten kann, ist die Datenmenge auf 255 Elemente bzw. den freien RAM-Bereich beschränkt, die Diskettenversion hat in dieser Hinsicht keine Grenzen.

Kassettenversion

```
100 DIM D$(255)
110 PRINT "Kopieren von Dateien auf Kassette"
120 PRINT "Quellkassette einlegen!"
130 INPUT "Quelldateiname: ";Q$
140 PRINT "Wenn '*ERROR 63 IN 160' erscheint, RUN 170 eingeben!"
150 ROPEN/T Q$: I=0
160 I=I+1: INPUT/T D$(I): GOTO 160
170 CLOSE/T
180 PRINT "Zielkassette einlegen!"
190 INPUT "Zieldateiname: ";Z$
200 WOPEN/T Z$
210 FOR J=1 TO I
220 PRINT/T D$(J)
230 NEXT J
240 CLOSE/T
250 END
```

Diskettenversion

```
100 DIM D$(255): F=0
110 PRINT "Kopieren von BSD-Dateien auf Diskette"
120 PRINT "Quelldiskette in Laufwerk 1 und Zieldiskette in Laufwerk 2 legen!"
130 INPUT "Name der Quelldatei: ";Q$
140 INPUT "Name der Zieldatei: ";Z$
150 ROPEN #1,FD1,Q$: WOPEN #2,FD2,Z$
160 I=-1
170 IF EOF(#1) THEN F=1: GOTO 200
180 I=I+1: INPUT #1 D$(I)
190 IF (SIZE>300)*(I<255) GOTO 170
200 IF I=-1 GOTO 240
210 FOR J=0 TO I
220 PRINT #2 D$(J)
230 NEXT J
240 IF F=0 GOTO 160
250 CLOSE #1: CLOSE #2
260 END
```

Zum Schluß zur Systemuhr des MZ-80B. Die Uhr wird von drei hintereinander geschalteten Zählern gebildet. Der erste wird mit 31250 Hz gespeist, der zweite vom ersten und der dritten vom zweiten. Dabei sieht es so aus, daß jeder Zähler mit einer Zahl zwischen 2 und 65535 geladen werden kann. Diese Zahl wird mit jedem Impuls am Zählereingang heruntergezählt, bis 0 erreicht ist. In diesem Moment wird ein Impuls am Ausgang erzeugt und der Startwert des Zählers automatisch wieder geladen. Man kann dies

wie einen programmierbaren Frequenzteiler verstehen. Um nun eine Uhr zu realisieren, wird der erste Zähler mit 31250 geladen Dadurch erzeugt er jede Sekunde einen Impuls, der vom zweiten Zähler aufgenommen wird. Dieser wiederum wurde mit 43200 geladen (entspricht der Anzahl der Sekunden in 12 Stunden, nämlich 60*60*12). Der zweite Zähler erzeugt also alle 12 Stunden einen Impuls, der vom dritten Zähler aufgenommen wird und dort als AM/PM-Flag dient (Vormittag/Nachmittag). Sie werden sich Fragen, wozu dies. Ganz einfach, die Anzahl der Sekunden in 24 Stunden paßt mit 86400 nicht in einen Zähler! Der Monitor SB-1510 stellt eine Routine zum Setzen und eine zum Lesen der Uhr zur Verfügung (TIMST, TIMRD), die die aktuelle Sekundenzahl und das AM/PM-Flag liefern. Die BASIC-Interpreter benutzen diese Routinen auch für die Variable TI$, nur das sie vorher aus der Anzahl der Sekunden und dem AM/PM-Flag die Uhrzeit in Stunden, Minuten und Sekunden errechnen.

Systemuhr!

Nun kann man die Zähler auch anders programmieren. Dabei ist nur zu beachten, daß neue Werte für die einzelnen Zähler von diesen erst nach dem Nulldurchgang übernommen werden, und daß vor dem Lesen der Zähler der aktuelle Zählerstand zwischengespeichert werden muß, damit er sich nicht gerade beim Lesen ändert. Zähler 1 wird über den I/O-Port 228, Zähler 2 über 229 und Zähler 3 über 230 angesprochen, alle drei werden über den I/O-Port 231 kontrolliert. Folgende Befehle sind brauchbar:

	Zwischenspeichern zum Lesen	Setzen
Zähler 1	4	52
Zähler 2	68	116
Zähler 3	132	180

Das folgende Programm zeigt die Zeit in 1/100 Sekunden an. An diesem Beispiel soll die Programmierung erklärt werden. Zeile 100 gibt die Befehle Zähler 1, 2 und 3 Setzen aus, Zeile 110 gibt die Werte für die drei Zähler aus, alle werden mit 2 beschrieben. Wie man sieht, wird wie bei 2-Byte-Maschinenzahlen zuerst das untere Byte der Zahl (Zahl-256*INT(Zahl/256) und dann das obere (INT(Zahl/256)) ausgegeben. Dies erfolgt deshalb, weil die Zähler ja so gesetzt gewesen sein können, daß der nächste Nulldurchgang erst in sehr langer Zeit erfolgt (im Extremfall, falls alle Zähler mit 65535 geladen waren und kurz zuvor den Nulldurchgang hatten, beim dritten Zähler erst nach 272 Jahren!). Durch diese Methode aber sind spätestens nach ca. 64 Sekunden (für den eben genannten Fall) alle Zähler mit 2 initialisiert, dies testet Zeile 120, in dem es Zähler 3 einliest. Dazu wird erst der Zwischenspeicher und Lesebefehl gegeben und dann beide Bytes eingelesen. Ergibt sich 2, sind alle Zähler mit der kleinsten möglichen Zahl, eben der 2, gesetzt.

Nun können sie mit den Zeilen 130-160 mit den wirklichen Werten gesetzt werden. Der erste Zähler wird mit 312 geladen, er erzeugt damit ungefähr alle 1/100 Sekunden einen Impuls, der zweite Zähler wird mit 100 geladen, er erzeugt jede Sekunden einen Impuls, der dritte wird mit 65535 geladen, er soll die Sekunden zählen, deshalb wird er mit dem größtmöglichen Wert geladen. Zeile 170 gibt die Befehle zum Lesen von Zähler 2 und 3, Zeile 180 liest sie ein, Zeile 190 ermittelt die seit dem Setzen des Zählers vergangene Zeit in 1/100 Sekunden. Da die Zähler ja rückwärts zählen, muß dazu die Differenz zu 100 bzw. 65535 gebildet werden. Zeile 200 gibt die Zahl aus. Dann geht es mit Zeile 170 wieder von vorn los. Da dieser Teil länger als 1/100 Sekunden braucht, werden nicht alle 100 Werte pro Sekunde angezeigt, was Sie feststellen können, wenn Sie den CURSOR-Befehl einmal weglassen.

Zur Zeitmessung brauchen Sie den Zähler nun nur vor und nach dem auszumessendem Ereignis lesen, beide Zeiten berechnen und die Differenz ermitteln. Um den alten Zustand, um also TI$ wieder im Sekundentakt zu haben, dient das zweite Programm. Seine Funktionsweise entspricht dem ersten.

```
100 OUT$ 231,52: OUT$ 231,116: OUT$ 231,180
110 OUT$ 228,2: OUT$ 228,0: OUT$ 229,2: OUT$ 229,0: OUT$ 230,2: OUT$ 230,0
120 OUT$ 231,132: INP$ 230,L3: INP$ 230,H3: IF L3+256*H3<>2 GOTO 120
130 OUT$ 231,52: OUT$ 231,116: OUT$ 231,180
140 OUT$ 228,56: OUT$ 228,1    ≙ 312
150 OUT$ 229,100: OUT$ 229,0   ≙ 100
160 OUT$ 230,255: OUT$ 230,255 ≙ 65535
170 OUT$ 231,68: OUT$ 231,132
180 INP$ 229,L2: INP$ 229,H2: INP$ 230,L3: INP$230,H3
190 Z=100-L2+100*(65535-L3-256*H3)
200 CURSOR 0,0: PRINT Z;"    ": GOTO 170
```

```
100 OUT$ 231,52: OUT$ 231,116: OUT$ 231,180
110 OUT$ 228,2: OUT$ 228,0: OUT$ 229,2: OUT$ 229,0: OUT$ 230,2: OUT$ 230,0
120 OUT$ 231,132: INP$ 230,L3: INP$ 230,H3: IF L3+256*H3<>2 GOTO 120
130 OUT$ 231,52: OUT$ 231,116: OUT$ 231,180
140 OUT$ 228,18: OUT$ 228,122   ≙ 31250
150 OUT$ 229,192: OUT$ 229,168  ≙ 43200
160 OUT$ 230,2: OUT$ 230,0      = 2
```

* * *