In die Geheimnisse der Floppy eingetaucht (Teil 5)

Wie funktioniert ein Kopierschutz, und wie kann man eigene Programme selbst schützen? Warum »rattert« die Floppy bei manchen Programmen? Und wissen Sie, was »Killertracks« sind? Die Antworten finden Sie im folgenden Artikel.

Die fortgeschrittenen Programmierer unter Ihnen werden sicher schon mit Ungeduld auf den Beginn des folgenden Abschnitts gewartet haben. Jetzt wird unser Kurs seinem Titel nämlich endlich voll gerecht werden, und wir wollen einmal sehen, was sich so alles mit einer Diskette anstellen läßt. Selbstverständlich sollen dabei Errors (Diskettenfehler) und »Killertracks« auch nicht zu kurz kommen.

Damit wir uns aber wieder an wichtige Tatsachen erinnern, noch einmal eine kurze Zusammenfassung einiger wichtiger Einzelheiten.

Wie im letzten Teil unseres Kurses ausführlich beschrieben, besteht ein Sektor auf Diskette aus zwei Teilen, nämlich dem Header und dem eigentlichen Datenblock. Beide Teile des Sektors werden auf Diskette durch eine SYNC-Markierung angekündigt, der dann das Kennzeichen (ob Header oder Datenblock) zur Identifikation folgt.

Der Blockheader enthält Track- und Sektornummer des Blocks, die beiden Bytes der Diskettenidentifikation (ID) und schließlich noch eine Prüfsumme, die dem Disk-Controller (DC) mitteilt, ob alle Daten einwandfrei gelesen wurden. Wurde der Blockheader richtig eingelesen, so wartet der DC auf den nachfolgenden Datenblock, der die Zeiger auf den nächsten Block im File, die Datenbytes und schließlich ebenfalls eine Prüfsumme enthält.

Zwischen Blockheader und Datenblock und zwischen Datenblock und Header des darauffolgenden Sektors befindet sich jeweils eine Lücke, die dem DC Zeit zum Umschalten seiner Modi (Lesen und Schreiben) läßt und außerdem für eine symmetrische Verteilung der Sektoren auf Diskette sorgt.

So, und jetzt genug der Wiederholung. Wir werden uns auf ein paar grundsätzliche Programmbeispiele stürzen, die Sie später in eigene Anwendungen einbauen können.

Wie wir schon wissen, werden alle Schreib-/Lesevorgänge des Disk-Controllers interruptgesteuert vorgenommen. Es ist also zum direkten Eingriff auf Diskette notwendig, daß wir uns die Regeln der Interruptsteuerung genau einprägen, da uns die Floppy bei unseren Experimenten sonst mit Sicherheit »abstürzt«.

Da wir in unserem Kurs verständlicherweise kein DOS-Listing abdrucken können, habe ich die wichtigsten Adressen, die wir benötigen, in Tabelle 1 zusammengefaßt und mit einer kurzen Erläuterung versehen, damit Sie sich mit der Anwendung der DOS-Routinen vertraut machen können.

Ein weiteres »Werkzeug« ist die RAM-Belegung der wichtigsten Speicherstellen, die in der letzten Folge 3 (64'er Ausgabe 1/1985) abgedruckt worden ist.

Um den Einstieg zu finden, fangen wir gleich einmal mit der Übergabe der Kommandos an den DC an. Wie bewerkstelligt es das Hauptprogramm, die unterschiedlichsten Befehle wie Lesen, Schreiben, Suchen, Kopf bewegen, Laufwerksmotor an, Formatieren und so weiter an den Disk-Controller zu übergeben?

Um eine Antwort auf diese Frage zu finden, betrachten Sie sich bitte die Tabelle 2. Sie enthält eine Aufstellung aller Jobcodes der Floppy 1541. Mit Jobcodes sind hierbei die Kommandos gemeint, die dafür sorgen, daß ein bestimmter Job zur Ausführung kommt.

Nehmen Sie jetzt einmal die Belegung der Zeropage zur Hand. Wenn Sie sich die Speicherstellen $0000 bis $0005 betrachten, so merken Sie schon am Namen, daß diese Adressen etwas mit unserer Sache zu tun haben. Es handelt sich hierbei um die Jobspeicher, die die Aufgabe haben, für den Dialog zwischen Hauptprogramm und DC zu sorgen.

Habe ich eben etwas von Dialog (nicht etwa Monolog) gesagt? Genau! Die Jobspeicher dienen nicht nur der Übergabe der Kommandos vom Hauptprogramm an den Disk-Controller; sie enthalten nach der Ausführung des Jobs auch die Rückmeldung des DC, an der das Hauptprogramm erkennen kann, ob der Job erfolgreich, das heißt fehlerlos durchgeführt worden ist.

Die Rückmeldungen des Disk-Controllers sind komplett in Tabelle 3 aufgeführt. Wenn Sie sich einmal die Bitmuster der Jobcodes und der Rückmeldungen ansehen und beide Typen miteinander vergleichen, so werden Sie sehr schnell einen Unterschied feststellen, der von entscheidender Bedeutung ist:

Die Befehlscodes sind ausschließlich negative Werte, das heißt Werte, die größer als $80 (128) sind. Das Kennzeichen solcher Zahlen ist das gesetzte Bit 7 im Byte, das deshalb auch als »negative bit« bezeichnet wird und bei jeder Befehlsausführung in Maschinensprache direkt in das Prozessorstatusregister übernommen wird (N-Flag).

Die Rückmeldungen sind fast ausschließlich Zahlen, die kleiner als $0F (16) sind (bis auf eine Ausnahme). Diese Größe spielt zwar nicht direkt eine Rolle; das

Wichtige Adressen des DOS:

$FD9E	— Rücksprung in die Jobschleife
$F556	— Sync-Signal auf Diskette abwarten
$FE00	— PCR auf lesen umschalten
$FE0E	— Track mit $55 vollschreiben
$FDA3	— Track mit Sync vollschreiben
$F510	— Blockheader lesen: + Diskette muß initialisiert sein + $32/33 muß die Adresse der Track- und Sektornummer enthalten (L/H); zum Beispiel $00/03, wenn die Nummern in $0300/0301 abgespeichert sind. + Rückkehr nur bei fehlerfreier Durchführung des Lesens
$F527	— Blockheader lesen: + Diskette muß initialisiert sein + zuvor muß $12 nach $16 und $13 nach $17 gebracht Werden + Track- und Sektornummer in $18 und $19 + Rückkehr nur bei fehlerfreier Durchführung des Lesens
$F50A	— Datenblockanfang suchen: + Parameter siehe $F510

Tabelle 1. Einige Unterprogramme des DOS

Jobcodes des DOS:

$80	— Lesen eines Blocks in einen Puffer
$90	— Schreiben eines Blocks aus einem Puffer
$A0	— Verify eines Sektors mit einem Pufferinhalt
$B0	— Testen eines Sektors auf Vorhandensein
$C0	— 'Bump' des Tonkopfes
$D0	— Maschinenprogramm im Puffer ausführen
$E0	— Programm im Puffer ausführen, nachdem das Laufwerk hochgefahren ist

Tabelle 2. Zeigt alle Jobcodes mit deren Aufgaben

Rückmeldungen der Jobschleife:

$01	— Fehlerfreie Durchführung (00, OK)
$02	— Blockheader wurde nicht gefunden (20, Read Error)
$03	— Sync-Markierung nicht gefunden (21, Read Error)
$04	— Datenblock wurde nicht gefunden (22, Read Error)
$05	— Datenprüfsumme ist falsch (23, Read Error)
$07	— Fehler nach einem Verify (25, Write Error)
$08	— Diskette ist Schreibgeschützt (26, Write Peotect on)
$09	— Prüfsumme im Header falsch (27, Read Error)
$0A	— Datenblock auf Diskette zu lang (28, Write Error)
$0B	— Falsche ID im Blockheader (29, Disk ID Mismatch)
$0F	— Keine Diskette im Laufwerk (74, Drive not ready)
$10	— Fehler bei Dekodierung (24, Read Error)

Tabelle 3. Zeigt alle Rückmeldungen des DC, wobei in Klammern die zugehörige Fehlermeldung steht.

Wichtigste ist jedoch, daß bei diesen Werten keiner größer als $7F (127) ist. Zu der Begründung für diese Einteilung werde ich im folgenden noch kommen.

Wie Sie aus der Belegung der Zeropage ersehen, existiert für jeden Puffer der Floppy ein eigener Jobspeicher. Das ermöglicht einen sehr dynamischen Einsatz der Floppystation, der es zum Beispiel erlaubt, mit mehreren Puffern gleichzeitig zu arbeiten.

Eine wichtige Regel sollten Sie sich gleich einprägen, damit später keine Pannen passieren: Wenn Sie einen Jobcode an den DC übergeben, sollten Sie darauf achten, daß der DC für die Ausführung dieses Jobs meistens einen Puffer benötigt. Den Puffer, der dabei zum Beispiel beschrieben wird, wählen Sie durch die Übergabe des Jobcodes in der entsprechenden Speicherstelle aus.

Achtung: Verwenden Sie dabei niemals den Puffer, in dem Sie Ihr Programm abgelegt haben, da dieses sonst unter Umständen gelöscht wird und sich die Floppy auf »mysteriöse« Weise verabschiedet.

Haben Sie also beispielsweise ein Programm ab $0300 (Puffer 0) abgelegt, so sollten Sie sich davor hüten, die Zeropageadresse $0000 als Jobspeicher zu benutzen. Auch als Zwischenspeicher sind die Adressen $0000 bis $0005 nicht unbedingt zu empfehlen, da es sonst zu einer kleinen Katastrophe kommen kann.

Die Kommandos an den Disk-Controller

Haben Sie sich die Speicherbelegung der Floppy schon etwas genauer betrachtet, so werden Ihnen auch die Speicherstellen $0006 bis $0011 nicht entgangen sein.

Wie wir wissen, gibt es verschiedene Jobcodes, die bestimmte Aktionen hervorrufen (die ausführliche Erläuterung der Jobcodes folgt gleich). Nun ist es aber in der Regel notwendig, einem Befehl auch ein paar Parameter mitzugeben, die dann in entsprechender Weise abgearbeitet werden.

In unserem Fall sind das sicherlich die Track- und Sektornummern, auf die sich unser jeweiliger Befehl beziehen soll. Wie Sie aus der Tabelle 2 nämlich ersehen können, existiert zum Beispiel ein Jobcode, der das Lesen eines Blocks veranlaßt. Hier ist es natürlich nötig, die Blockparameter mit anzugeben.

Wollen Sie also ein Kommando $80 an den DC für Puffer 1 übergeben, so schreiben Sie zunächst in die Speicherstelle $0008 die Tracknummer und in Speicherstelle $0009 die Sektornummer des Blocks, der in Puffer 1 gelesen werden soll. Anschließend erhält die Speicherstelle $0001 den Jobcode und auf geht's... Das klingt alles recht einfach. Stimmt, recht viel komplizierter wird es auch nicht mehr.

Unser einziges Problem besteht jetzt nur noch in der Tatsache, daß der DC für die Ausführung der Befehle eine gewisse Zeit benötigt, die je nach Kommando mehrere Interruptaufrufe erforderlich macht. Woher wissen wir also jetzt, wann ein Block vollständig in den Puffer gelesen ist und wir dessen Inhalt übernehmen können?

Die Lösung dieses Problems liegt in der unterschiedlichen Wertigkeit der Befehlsbytes und der Rückmeldungen des DC, die ich vorhin schon angesprochen habe. Sie können sich noch erinnern: Alle Jobcodes bestehen aus Werten größer als $80 und alle Rückmeldungen aus Werten kleiner als $80.

Da der DC aber nach **jedem** Job seine Rückmeldung in der gleichen Speicherstelle hinterläßt, in die wir vorher das Kommando geschrieben hatten, ist es uns nun ein leichtes, diese Speicherstelle zu überprüfen und das Ende des Jobs anhand der Rückmeldung abzufragen. Anhand der noch folgenden Beispiele wird diese Technik gründlich erläutert.

Jetzt wollen wir uns aber mit den eigentlichen Jobcodes und deren Aufgaben beschäftigen.

1) Lesen eines Sektors in einen Puffer:

Wenn wir einen Sektor in einen Puffer lesen wollen, so stellen wir fest, daß diese Aktion auf der Ebene der Jobschleife fast genauso einfach ist, wie von Basic aus mit dem »B-R«- beziehungsweise »U1«-Befehl. Zum Lesen eines Sektors geben Sie dessen Track- und Sektornummer in den entsprechenden Speicherstellen für den gewünschten Puffer an. Anschließend senden Sie den Code $80 an den DC, und das Laufwerk startet sofort und liest den Sektor ein.

Diese Befehlsübergabe können Sie sogar von Basic aus, mit den MEMORY- und BLOCK-Befehlen, realisieren und dann den Pufferinhalt auslesen, um sich zu überzeugen, daß der Block auch wirklich eingelesen wurde.

Achtung: Die Diskette muß beim Arbeiten in der Jobschleife von Hand initialisiert werden, da wir uns auf dieser unteren Programmierebene im Rücken der automatischen Initialisierung befinden, die hier deshalb nicht mehr von alleine erfolgt. Merken Sie, daß der Inhalt im Puffer nicht mit dem auf der Diskette übereinstimmt, so kann das mit großer Wahrscheinlichkeit an der fehlenden Initialisierung liegen; doch auch dazu später noch mehr.

Jetzt wollen wir die Jobcodes anhand kleiner Beispiele genauer kennenlernen; dabei wollen wir uns auch gleichzeitig mit den Rückmeldungen des DC vertraut machen, anhand derer sich Fehler in der Ausführung des Jobs erkennen lassen.

Wir werden jetzt den Jobcode für Lesen des Blocks 18,1 in Puffer 0 übergeben und uns dann die Rückmeldung und den Inhalt des Blocks ansehen. Mit dem POKE-Befehl im Programm schreiben wir den Inhalt des Puffers direkt in den Bildschirmspeicher, was für unsere Kontrolle langen soll:

```
 1 OPEN1,8,15,"I"
 2 PRINT#1,"M-W"CHR$(6)CHR$(0)CHR$(2)CHR$(18)CHR$(1)
 3 PRINT#1,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(128)
 4 FORX=0TO2000: NEXT X
 5 PRINT#1,"M-R"CHR$(0)CHR$(0)CHR$(1)
 6 GET#1,A$: PRINT ASC(A$+CHR$(0))
 7 FORX=0TO255
 8 PRINT#1,"M-R"CHR$(X)CHR$(3)CHR$(1)
 9 GET#1,A$: POKE1024+X,ASC(A$+CHR$(0))
10 NEXT X
11 CLOSE 1
```

Dieses kleine Programm initialisiert die Diskette im Laufwerk. Anschließend werden Track und Sektor (18,1) übergeben und schließlich der Jobcode in Adresse $0000 geschrieben, der dafür sorgt, daß unser Block in Puffer 0 geladen wird. Nach einer kleinen Warteschleife, in der die Floppy Zeit zur Befehlsausführung hat, wird der Jobspeicher wieder ausgelesen. Anhand von Tabelle 2 können Sie erkennen, daß der Job ordnungsgemäß ausgeführt wurde, wenn Sie als Rückmeldung eine »1« bekommen.

Auf dem Bildschirm erscheint der Inhalt des Puffers, wobei unter anderem auch Teile des Directory der Diskette zum Vorschein kommen sollten.

2) Schreiben eines Blocks auf Diskette:

Analog zum Lesen eines Blocks erfolgt das Schreiben. Hier übergeben Sie die gleichen Parameter; nur muß sich der zu schreibende Block schon im Puffer der Floppy befinden. Durch die Auswahl des Jobspeichers können Sie jeden x-beliebigen Puffer der Floppy (0 bis 4) in jeden Block der Diskette schreiben.

3) Verifizieren eines Blocks von Diskette:

Dieser Vorgang erfolgt in der Floppy bei einem SAVE normalerweise automatisch. Aus diesem Grund dauert das Speichern eines Programms auch um einiges länger als das Wiedereinladen in den Computer. Mit Hilfe des entsprechenden Jobcodes ($A0) können wir ein VERIFY aber nach unserem Belieben starten, um den Inhalt in einem Pufferspeicher mit einem Block auf Diskette zu vergleichen.

Entspricht der Inhalt des Puffers nicht dem Inhalt auf Diskette, so erhalten wir als Rückmeldung die Nummer 7. Beim LOAD-Befehl entspräche das einem »VERIFY ERROR«.

Übrigens: Ich habe ja schon auf die Notwendigkeit des Initialisierens hingewiesen. Unterbleibt dieser Vorgang, so können Sie anhand der Tabelle 3 schon erkennen, was für eine Meldung Sie bekommen werden. Richtig! Die Nummer 11 wird auf Ihrem Bildschirm erscheinen.

4) Suchen eines Sektors:

Dieses Kommando dient nicht dem Lesen eines Blocks von Diskette. Hier wird lediglich untersucht, ob sich der von Ihnen angegebene Block überhaupt auf Diskette befindet. Ist das nicht der Fall, so erhalten Sie eine »2« als Antwort.

Ihnen ist vielleicht auch schon ein weiterer Vorteil der Jobschleife aufgefallen: Es erfolgt keine Kontrolle auf »legale« Angaben mehr; das heißt, wenn Sie an den Disk-Controller das Kommando geben, daß er Block 2 auf Track 38 lesen soll, dann tut er dies auch.

Versuchen Sie das einmal mit dem U1-Befehl; hier bekommen Sie als Antwort: »ILLEGAL TRACK OR SEKTOR«, da Track 38 gar nicht existiert.

So groß der Vorteil dieser Nichtkontrolle auch sein mag; sie sollten sich dessen immer bewußt sein, daß der DC auch versuchen würde, auf Track 100 zuzugreifen, wenn dies verlangt werden sollte.

Die Folge wäre hierbei ein Anschlagen des Kopfes an die vordere Laufschienenbegrenzung der Mechanik; eine sicherlich nicht sehr schonende Angelegenheit.

5. Kopf neu positionieren (BUMP):
Dieser Befehl hat eine nützliche Funktion, die jedoch auch für eine sicher nicht unerhebliche Menge an verstellten Tonköpfen verantwortlich ist. Kann der DC einen Track nicht identifizieren, so besteht die Möglichkeit, daß der Kopf sich auf einer illegalen Spur befindet. In diesem Fall kann der DC die Position des Kopfes nicht mehr anhand der Blockheader auf jedem Track bestimmen.

Aus diesem Grund passiert folgendes: Der DC fährt den Kopf zurück an den Anschlag, und nach einem »Rattern« erfolgt eine neue Ansteuerung des gewünschten Tracks.

Mit dem Kommando $C0 können Sie ein solches Bump ausführen lassen. Nach dem BUMP können Sie den Kopf neu positionieren lassen; der Tonkopf steht ansonsten immer auf Track 1.

6) Maschinenprogramm im Puffer starten:
Mit dem Jobcode $D0 machen Sie intern genau das, was extern mit dem M-E-Befehl funktioniert. Der Unterschied zum M-E-Befehl besteht nur in der Tatsache, daß das Programm, das durch $D0 aufgerufen wird, als Interruptprogramm arbeitet, das heißt es wird in die Jobschleife mit eingebaut und darf deshalb nicht mit einem RTS enden, da ein JMP zurück in die Jobschleife erfolgen muß.

Wie Sie aus einem solchen Programm zurückspringen, wird später noch erläutert.

7) Programm im Puffer starten, nachdem das Laufwerk hochgefahren ist:
Den letzten Befehl werden wir kaum benutzen, da ihm eine Eigenschaft fehlt, die wir dringend benötigen. Wollen wir nämlich ein Programm in der Jobschleife starten, so werden wir meistens Schreib- oder Lesezugriffe auf die Diskette ausführen. Dies ist jedoch mit $D0 nicht möglich, da das Laufwerk stillsteht.

Der Befehl $E0 hat nun folgende Auswirkungen: Erkennt der DC den Jobcode, so wird das Laufwerk angefahren und die Hardware auf Diskettenzugriff vorbereitet. Mit Hilfe dieses Befehls ist es also möglich, direkt auf die Diskette zuzugreifen, was in einem eigenen Maschinenprogramm erfolgt.

Auch hier muß das Programm mit einem JMP-Befehl beendet werden, da ein Rücksprung in die Jobschleife erfolgen soll.

Wichtig ist noch, daß das Programm, das mit $D0 oder $E0 gestartet werden soll, immer am Anfang des entsprechenden Puffers stehen muß. Sollen also Programmteile aufgerufen werden, die an höheren Adressen, als $xx00 (xx = 03 bis 07) stehen, so müssen diese über Sprungbefehle aufgerufen werden.

Mit $E0 werden wir uns in unserem Kurs noch öfters beschäftigen, da er die Grundlage der Diskettenzugriffe darstellt (er wird auch vom DOS für das Formatieren angewendet).

Eine Sache dürfen Sie aber auch beim Jobcode $E0 nicht vergessen, nämlich Track- und Sektornummer anzugeben. Es wird, wie schon erwähnt, das Laufwerk betriebsbereit gemacht. Dazu gehört aber auch das Positionieren des Tonkopfes auf die richtige Spur.

```
0500        JSR $FE0E  ; TRACK LOESCHEN
0503        JMP $FD9E  ; ZUR JOBSCHLEIFE
0506        LDA #$01   ; TRACKNUMMER
0508        STA $0A    ; IN JOBSPEICHER
050A        LDA #$E0   ; JOBCODE
050C WAIT   STA $02    ; UEBERGEBEN
050E        LDA $02    ; RUECKMELDUNG
0510        BMI WAIT   ; ENDE ABWARTEN
0512        RTS        ; PROGRAMMENDE
```

Listing 1. Herstellen eines »21, READ ERROR« auf einer Spur. Startadresse bei $0506.

Wie schreibt das DOS auf Diskette?

Wir haben jetzt die Möglichkeit, ein Maschinenprogramm im Pufferspeicher der Floppy abzulegen und dort zu starten. Unsere Jobcodes erlauben es uns außerdem, direkt in den Ablauf der Jobschleife einzugreifen und die Diskette sozusagen »von Hand« zu manipulieren.

Als letztes fehlen uns jetzt nur noch die Kenntnisse über den direkten Zugriff auf den Schreib-/Lesekopf der Floppy, so daß wir einzelne Bits ohne Umwege und ohne irgendeine Einschränkung durch die Blockstruktur der Diskette direkt auf die Magnetschicht schreiben können. Mit diesem Problem, das eigentlich gar keines ist, wollen wir uns jetzt beschäftigen. Dazu ein paar Bemerkungen zur Organisation der Schreib-/Leseelektronik der 1541.

Die Bytes werden zwar auf Diskette in serieller Bitfolge abgelegt; dieses Problem braucht uns jedoch gar nicht weiter zu beschäftigen. Der VIA 6522, der für uns die Elektronik steuert, kann nämlich von uns wie eine Speicherstelle behandelt werden. Senden wir also ein Byte an den VIA 6522, so geschieht das Schreiben auf Diskette vollautomatisch, so daß uns diese Sache nicht weiter beschäftigen soll.

Das einzige Problem, das sich bei der ganzen Angelegenheit stellt, ist die Frage des Timing. Immerhin benötigt der Schreib- oder Lesevorgang eine gewisse Zeit, das heißt, wenn wir beispielsweise Daten vom Tonkopf lesen wollen, muß uns der DC erst mitteilen, wann das nächste Byte fertig eingelesen ist und zur Ausgabe bereitsteht.

Zur Steuerung dieses Timings wird in der 1541 das V-(Overflow-)Flag des Prozessorstatusregisters benutzt. Der Mikroprozessor 6502 hat nämlich den Vorteil, daß dieses Flag extern beeinflußt werden kann.

Die Regel sieht also folgendermaßen aus: Hat die Lese-Elektronik ein Byte vollständig eingelesen, so wird das V-Flag auf »1« gesetzt. Genauso verhält es sich mit dem Schreiben: Wurde das gegebene Byte komplett auf Diskette geschrieben, so erfolgt ebenfalls ein Setzen des V-Flags.

Das einzige, das der Programmierer nie vergessen darf, ist, daß das V-Flag nach dem Erkennen »von Hand« wieder auf »0« gesetzt werden muß, damit keine Fehlinformation erfolgen kann.

Die Speicherstelle, die für Schreib- und Lesebetrieb zuständig ist, ist Port A des Disk-Controllers mit der Adresse $1C01.

Endlich kommt die Praxis

So, nachdem wir nun so ziemlich alle Voraussetzungen zum Programmieren haben, soll es jetzt endlich mit der praktischen Anwendung unseres Wissens losgehen. Das Werkzeug, das wir jetzt benötigen, besteht aus einem komfortablen Monitor mit »Miniassembler«. Da die Floppyprogramme, die zum Beispiel Fehler auf Diskette bringen, relativ kurz sind, ist es am besten, wenn wir einen Monitor in den Bereich ab $C000 laden und uns anschließend den Bereich ab $8000 für unsere Anwendungen sichern:

POKE 56,127: POKE 52,127: NEW (oder CLR)

Wir legen also unsere kleinen Maschinenprogramme ab $8000 ab und senden diese jeweils mit einem Basic-Programm zur Floppy, wo wir sie dann ausführen.

Achtung: Bei einem RESET wird der Speicher der Floppy gelöscht. Es ist also empfehlenswert, die Programme vor jedem Neustart wieder in den Pufferspeicher der 1541 zu schreiben.

Error Nummer 21 auf Diskette

Ein früher beliebter Programmschutz war das Aufbringen von Errors auf Diskette. Diese konnten von den »alten« Kopierprogrammen nicht übernommen werden. Das geschützte Programm brauchte also nur einen definierten Fehler auf Diskette abzufragen und bei Nichtvorhandensein »auszusteigen«. Wenn Sie sich die Tabelle der Fehlermeldungen im Commodore-Handbuch zur 1541 ansehen, werden Sie sehr schnell erkennen, daß es für jeden kleinen Defekt eine eigene Fehlernummer gibt. Betrachten Sie jetzt Tabelle 3 dieser Folge, so können Sie dort ablesen, welche Rückmeldung des DC welche Fehlermeldung an den Computer zur Folge hat.

Wir wollen uns einmal den Fehler mit der Nummer 21 ansehen. Er tritt dann auf, wenn die Floppy versucht, einen Track zu lesen, auf diesem jedoch keine SYNC-Markierungen findet. Das ist zum Beispiel bei einer unformatierten oder beschädigten Diskette der Fall.

Unser kleines Programm in Listing 1 werden Sie vom Prinzip sehr schnell durchschauen. Es macht nichts weiter, als einen bestimmten Track auf Diskette mit lauter $55 (binär: 01010101) zu überschreiben. Das hat zur Folge, das alle SYNC-Markierungen gelöscht werden und ein Fehler »21« ist die Folge, wenn ein Zugriff stattfinden soll.

Für unsere Versuche sollten Sie eine leere, neuformatierte Diskette verwenden, die Sie sich speziell für unsere Experimente aufheben. Geben Sie also einmal das Programm in Listing 1 ein und starten Sie es anschließend (leere Diskette einlegen!).

Versuchen Sie nun den Track 5 Ihrer Diskette später einmal zu lesen, so wird sich die Floppy mit einem »21, READ ERROR« dafür bedanken.

Wie Sie sehen, ist ein Fehler 21 recht einfach zu erzeugen, da sich dieser über einen gesamten Track erstreckt (alle Informationen werden gelöscht).

Schwieriger wird es bei anderen Fehlern, die beispielsweise nur in einzelnen Blöcken vorkommen, wobei einige davon (20, 22) auch auf einen gesamten Track geschrieben werden kön-

```
AUF SCHREIBEN UMSCHALTEN:

      LDA $1C0C   ; PCR
      AND #$1F    ; AUF SCHREIBMODUS
      ORA #$C0    ;
      STA $1C0C   ; UMSCHALTEN
      LDA #$FF    ;
      STA $1C03   ; PORT A AUF AUSGANG
AUF LESEN UMSCHALTEN:

      LDA $1C0C   ; PCR
      ORA #$E0    ; AUF LESEMODUS
      STA $1C0C   ; UMSCHALTEN
      LDA #$00    ;
      STA $1C03   ; PORT A AUF EINGANG
```

Bild 1. Diese Programme können für das Umschalten zwischen Schreiben und Lesen verwendet werden.

nen. Es sind dies die Fehler mit den Nummern 23, 24, 27, 28 und 29.

Um solche Fehler zu erzeugen, muß jeweils der zu zerstörende Sektor abgetastet werden, bis die richtige Stelle für den Eingriff gefunden wird. Damit Sie die wichtigen Routinen zur Arbeit innerhalb der Jobschleife ebenfalls aufrufen können, sind in Tabelle 1 ein paar wichtige Unterprogramme des DOS mit den geforderten Parametern aufgeführt.

Einen Error 22 beispielsweise

```
0 REM   PROGRAMM ZUM ERZEUGEN EINES          <201>
1 REM         23, READ ERROR                 <199>
2 REM      IN BELIEBIGEM SEKTOR              <197>
3 REM                                        <146>
4 REM  VON KARSTEN SCHRAMM 09.01.1985        <166>
5 REM                                        <148>
6 REM                                        <149>
7 REM  PROGRAMM WIRD AUCH IN SPEICHER        <060>
8 REM       AB $8000 ABGELEGT.               <111>
9 REM                                        <152>
10 POKE 56,31:POKE 52,31:CLR:OPEN 1,8,15,"
   I"                                        <243>
20 FOR X=0 TO 80:READ A:POKE 32768+X,A:NEX
   T                                         <227>
30 INPUT"CLR,DOWN,SPACETRACK FUER ERROR
   22";T                                     <175>
40 INPUT"DOWNSEKTOR FUER ERROR 22";S      <136>
50 POKE 32777,T:POKE 32834,T:POKE 32781,S  <253>
60 RESTORE                                 <200>
70 FOR X=0 TO 80:READ A:PRINT#1,"M-W"CHR$(
   X)CHR$(5)CHR$(1)CHR$(A):NEXT X          <198>
80 PRINT:PRINT:PRINT"PROGRAMM STARTET"     <095>
90 PRINT#1,"M-E"CHR$(64)CHR$(5):CLOSE 1:EN
   D                                         <230>
100 DATA 165,18,133,22,165,19,133,23,169,3
    5,133,24,169,0,133,25,32,39              <121>
110 DATA 245,32,86,245,162,0,202,208,253   <235>
120 DATA 173,12,28,41,31,9,192,141,12,28,1
    69,255,141,3,28,169,85,141,1             <191>
130 DATA 28,80,254,184,80,254,184,80,254,1
    84,32,0,254,76,158,253,234,234           <066>
140 DATA 234,169,35,133,10,169,224,133,2,1
    65,2,48,252,96,0,0,0                     <059>
```

Listing 3a. Ein READ ERROR 23 wird erzeugt

würden Sie dadurch herstellen, daß Sie die Routine zum Finden des Datenblocks aufrufen. Diese kehrt bei gefundenem Datenblock mit RTS zurück. Jetzt schalten Sie auf Schreiben um (in Bild 1 dargestellt) und bringen ein paar Bytes ohne Konzept auf die Diskette. Versucht der DC diesen Datenblock später einmal zu lesen, so erfolgt ein Fehler 22, da sie die Datenblockkennung, die direkt hinter der SYNC-Markierung steht, zerstört haben.

Wollen Sie einen Fehler mit der Nummer 23, dann ist es erforderlich, daß Sie den Vorspann des Datenblocks überspringen und erst inmitten der gespeicherten Daten einen Schreibzugriff durchführen. Durch diesen Zugriff, der in der Prüfsumme am Blockende natürlich nicht verzeichnet wird, folgt die Meldung »23, READ ERROR«, als Zeichen eines Prüfsummenfehlers.

Listing 2a und 3a zeigen Ihnen Programme, die einen Error 22 und einen Error 23 erzeugen (Listing 2 b und 3 b sind die zugehörigen Quellprogramme).

Der Vorteil eines Fehlers mit der Nummer 23 ist, daß die Daten in der Regel schon im Puffer stehen, bevor der Fehler erkannt wird, das heißt, Sie können einen Datenblock auf Diskette gezielt mit einem Fehler versehen, obwohl dieser noch lesbare Inhalte enthält.

```
0 REM   PROGRAMM ZUM ERZEUGEN EINES          <201>
1 REM         22, READ ERROR                 <198>
2 REM      IN BELIEBIGEM SEKTOR              <197>
3 REM                                        <146>
4 REM  VON KARSTEN SCHRAMM 09.01.1985        <166>
5 REM                                        <148>
6 REM                                        <149>
7 REM  PROGRAMM WIRD AUCH IN SPEICHER        <060>
8 REM       AB $8000 ABGELEGT.               <111>
9 REM                                        <152>
10 POKE 56,31:POKE 52,31:CLR:OPEN 1,8,15,"
   I"                                        <243>
20 FOR X=0 TO 80:READ A:POKE 32768+X,A:NEX
   T                                         <227>
30 INPUT"CLR,DOWN,SPACETRACK FUER ERROR
   22";T                                     <175>
40 INPUT"DOWNSEKTOR FUER ERROR 22";S      <136>
50 POKE 32777,T:POKE 32834,T:POKE 32781,S  <253>
60 RESTORE                                 <200>
70 FOR X=0 TO 80:READ A:PRINT#1,"M-W"CHR$(
   X)CHR$(5)CHR$(1)CHR$(A):NEXT X          <198>
80 PRINT:PRINT:PRINT"PROGRAMM STARTET"     <095>
90 PRINT#1,"M-E"CHR$(64)CHR$(5):CLOSE 1:EN
   D                                         <230>
100 DATA 165,18,133,22,165,19,133,23,169,3
    5,133,24,169,1,133,25,32,39              <122>
110 DATA 245,32,86,245,173,12,28,41,31,9,1
    92,141,12,28,169,255,141,3,28            <234>
120 DATA 169,85,141,1,28,80,254,184,80     <161>
130 DATA 254,184,80,254,184,32,0,254,76    <218>
140 DATA 158,253,234,234,234,234,234,234   <017>
150 DATA 234,234,169,35,133,10,169,224,133
    ,2,165,2,48,252,96,0,0,0                 <010>
```

Listing 2a. Ein READ ERROR 22 wird erzeugt (in einem beliebigen Sektor)

```
0500        LDA $12      ; ID 1 HOLEN
0502        STA $16      ; UND UEBERNEHMEN
0504        LDA $13      ; ID 2 HOLEN
0506        STA $17      ; UND UEBERNEHMEN
0508        LDA #$23     ; TRACKNUMMER
050A        STA $18      ; UEBERNEHMEN
050C        LDA 00       ; SEKTORNUMMER
050E        STA $19      ; UEBERNEHMEN
0510        JSR $F527    ; BLOCKHEADER HOLE
                           N
0513        JSR $F556    ; AUF 'SYNC' WARTE
                           N
0516        LDX #$00     ; WARTEN, UM IN
0518 LOOP   DEX          ; DEN DATENBLOCK
0519        BNE LOOP     ; ZU KOMMEN
051B        LDA $1C0C    ;
051E        AND #$1F     ; PCR AUF SCHREIBE
                           N
0520        ORA #$C0     ; UMSCHALTEN
0522        STA $1C0C    ;
0525        LDA #$FF     ; PORT A AUF
0527        STA $1C03    ; AUSGANG STELLEN
052A        LDA #$55     ; FALSCHWERT
052C        STA $1C01    ; IN PUFFER SCHREI
                           BEN
052F W1     BVC W1       ; WARTEN AUF READY
0531        CLV          ; FLAG LOESCHEN
0532 W2     BVC W2       ; WARTEN AUF READY
0534        CLV          ; FLAG LOESCHEN
0535 W3     BVC W3       ; WARTEN AUF READY
0537        CLV          ; FLAG LOESCHEN
0538        JSR $FE00    ; AUF LESEN SCHALT
                           EN
053B        JMP $FD9E    ; ZUR JOBSCHLEIFE
053E        NOP          ;
053F        NOP          ;
0540        NOP          ;
0541        LDA #$23     ; TRACKNUMMER
0543        STA $0A      ; IN JOBSPEICHER
0545        LDA #$E0     ; JOBCODE
0547        STA $02      ; UEBERGEBEN
0549 WAIT   LDA $02      ; RUECKMELDUNG
054B        BMI WAIT     ; WARTEN AUF ENDE
054D        RTS          ; PROGRAMMENDE
```

Listing 2b. Herstellen eines »22, READ ERROR« (Assemblerprogramm)

```
0500        LDA $12      ; ID 1 HOLEN
0502        STA $16      ; UND UEBERNEHMEN
0504        LDA $13      ; ID 2 HOLEN
0506        STA $17      ; UND UEBERNEHMEN
0508        LDA #$23     ; TRACKNUMMER
050A        STA $18      ; UEBERNEHMEN
050C        LDA 01       ; SEKTORNUMMER
050E        STA $19      ; UEBERNEHMEN
0510        JSR $F527    ; BLOCKHEADER HOLE
                           N
0513        JSR $F556    ; AUF 'SYNC' WARTE
                           N
0516        LDA $1C0C    ; PCR
0519        AND #$1F     ; AUF SCHREIBEN
051B        ORA #$C0     ; UMSCHALTEN
051D        STA $1C0C    ;
0520        LDA #$FF     ; PORT A AUF AUSGA
                           NG
0522        STA $1C03    ; UMSCHALTEN
0525        LDA #$55     ; FALSCHWERT
0527        STA $1C01    ;
052A W1     BVC W1       ; SCHREIBEN
052C        CLV          ;
052D W2     BVC W2       ; SCHREIBEN
052F        CLV          ;
0530 W3     BVC W3       ; SCHREIBEN
0532        CLV          ;
0533        JSR $FE00    ; PCR AUF LESEN
0536        JMP $FD9E    ; ZUR JOBSCHLEIFE
0539        NOP          ;
053A        NOP          ;
053B        NOP          ;
053C        NOP          ;
053D        NOP          ;
053E        NOP          ;
053F        NOP          ;
0540        NOP          ;
0541        LDA #$23     ; TRACKNUMMER
0543        STA $0A      ; IN JOBSPEICHER
0545        LDA #$E0     ; JOBCODE
0547        STA $02      ; UEBERGEBEN
0549 WAIT   LDA $02      ; RUECKMELDUNG
054B        BMI WAIT     ; WARTEN AUF ENDE
054D        RTS          ; PROGRAMMENDE
```

Listing 3b. Herstellen eines »23, READ ERROR« (Assemblerprogramm)

Die eben besprochenen Fehler auf Diskette eignen sich hervorragend für einen Kopierschutz. Am wirkungsvollsten sind dabei mit Sicherheit solche Fehler, die zusätzlich noch Daten enthalten. Es gibt nämlich schon eine ganze Menge von Programmen, die Fehler übernehmen und auf der Kopie wieder simulieren.

Soweit zu Fehlern. Haben Sie schon einmal etwas von »Killertracks« gehört? Dieses anschauliche Wort steht für die Manipulation eines Tracks, der sämtliche Sicherheitseinrichtungen des DOS durcheinanderbringt.

Vielleicht hatten Sie schon einmal eine Diskette in Ihren Händen, die folgendes »Phänomen« aufzeigte: Wenn Sie versuchten, einen Block auf einer bestimmten Spur zu lesen, ist der Schreib/Lesekopf der Floppy ordnungsgemäß auf den Track positioniert worden. Danach hat der DC mit dem Lesen des Blocks angefangen und — nicht mehr aufgehört. Anders ausgedrückt: Die 1541 las und las ...

Die Spur, die Sie da versucht haben zu lesen, hat grundsätzlich dafür gesorgt, daß sich die Floppy »aufhängte«. Das es sich hier um schon angesprochenen »Killertrack« handelte, brauche ich kaum noch zu erwähnen. Aber, wie stellt man eine solche »Falle« her? Was ist mit dem Track passiert, daß der DC völlig »aus dem Häuschen« gerät? Die Antwort sehen Sie in Listing 4. Dieses kleine Programm stellt einen solchen »Killertrack« her. Des Rätsels Lösung ist eigentlich ganz einfach: Die gesamte Spur besteht aus einer einzigen SYNC-Markierung. Da die SYNC-Markierung von der Lese/Schreibelektronik speziell verarbeitet wird, verzögert sich die Arbeit des DC gewaltig, wenn eine solche »Dauer-SYNC-Markierung« auftritt.

Da die Floppy bei Fehlern bis zu über 200 mal versucht, einen Block zu lesen, dehnt sich der Zeitraum, den sie bei Verzögerungen benötigt, stark aus. Bei »Killertracks« braucht die Floppy pro Leseversuch eine Unmenge an Zeit, was sich auch im langsamen Blinkrhythmus der LED am Laufwerk zeigt.

Allein schon an den kleinen Anwendungen können Sie erkennen, wie vielseitig und vielfältig die Möglichkeiten sind, die einem in der Programmierung offenstehen. Wenn Sie intensiv mit der Floppy arbeiten, werden Sie bald schon neue Anwendungsmöglichkeiten kennenlernen. Aus der 1541 läßt sich noch eine Menge herausholen, wie wir noch feststellen werden, wobei der Kopierschutz von Disketten sicher nur einen kleinen Teil der vielfältigen Möglichkeiten darstellt.

(Karsten Schramm/gk)

```
0500        JSR $FDA3 ; TRACK LOESCHEN
0503        JMP $FD9E ; ZUR JOBSCHLEIFE
0506        LDA #$01  ; TRACKNUMMER
0508        STA $0A   ; IN JOBSPEICHER
050A        LDA #$E0  ; JOBCODE
050C WAIT   STA $02   ; UEBERGEBEN
050E        LDA $02   ; RUECKMELDUNG
0510        BMI WAIT  ; ENDE ABWARTEN
0512        RTS       ; PROGRAMMENDE
```

Listing 4. Ein »Killertrack« wird erzeugt. Startadresse bei $0506.