Diesmal geht es um die Herstellung von Boot-Disketten.

Auch in der nächsten Assemblerecke werden wir uns noch mit diesem Thema beschäftigen. Dort werden Sie dann ein Programm vorfinden, mit dem Sie beliebige Maschinenprogramme in ein bootfähiges Format bringen können. Das ist eine nützliche Sache, wenn man eigenen Programmen ein wirklich professionelles Aussehen verleihen will.

Doch zunächst sollen die Grundlagen des Boot Vorgangs erläutert werden. Zu diesem Zweck wird ein Assemblerprogramm gezeigt, das ein kleines Programm im bootfähigen Format auf eine Diskette schreibt. Wenn Sie diese ins Laufwerk legen und den Computer einschalten, so wird das Programm ohne jegliches DOS sofort eingelesen und gestartet, mit anderen Worten, es wird gebootet.

Welche Vorteile sind nun damit verbunden? Ein gebootetes Programm hat den ganzen Speicher von $0600 bis $BFFF zur Verfügung, da kein DOS vorhanden ist. Darüber hinaus wirkt es sehr professionell; Zwischentitel (Loading...) können sehr frühzeitig ausgegeben und der Einbau eines Kopierschutzes sehr wirkungsvoll durchgeführt werden. Fast alle kommerziell erhältlichen Spielprogramme nutzen diese Vorteile des Boot-Formates.

Im Grunde tun wir nichts anderes, als das DOS durch ein eigenes Programm zu ersetzen. Normalerweise wird beim Booten einer DOS-Diskette das DOS in den Arbeitsspeicher geladen und im Betriebssystem verankert. Ohne DOS könnten Sie kein Inhaltsverzeichnis anfordern und auch keine Files lesen und schreiben, da das Betriebssystem im ROM des Atari nur auf sehr einfache Weise mit der Diskettenstation kommunizieren kann. Für viele Anwendungen (wie z.B. Spiele) ist dies völlig ausreichend, da in den meisten Fällen nach dem Laden des Programms keine Zugriffe auf die Diskette mehr nötig sind.

Disk-Handler

Die ROM-Routinen erlauben nur, einzelne Sektoren von der Disk zu lesen oder zu schreiben bzw. eine Diskette zu formatieren. Ein DOS kann diese einfachen Funktionen benutzen, um die Verwaltung von Dateien zu erledigen. Jeder Atari-User hat wohl schon die ärgerliche Erfahrung gemacht, dass dies auf sehr unterschiedliche Art geschehen kann, denn sonst wären DOS 2.5 und DOS 3 kaum so wenig kompatibel. Nun erhebt sich aber noch die Frage, wie denn ein DOS (oder ein anderes bootfähiges Programm) in den Speicher geladen werden kann. Schließlich ist zu diesem Zeitpunkt ja noch kein DOS im Computer vorhanden, mit dem dies bequem möglich wäre.

Dieser Vorgang wird im Rahmen der Power-Up-Routine erledigt, die in der Assemblerecke der Ausgabe August/September 1986 abgehandelt wurde. Diese Routine, die nach dem Einschalten des Computers angesprungen wird, lädt den ersten Sektor einer Diskette in den Speicher. Anhand einiger Angaben in diesem Sektor wird erkannt, wie lang das Boot-Programm ist, an welche Adresse es geladen werden soll und wo die Startadresse liegt. Mit Hilfe dieser Informationen werden weitere Sektoren (mit aufsteigender Nummer) geladen und das gebootete Programm schließlich gestartet.

Oftmals wird aber im Rahmen dieses ersten Boot-Vorgangs nicht das Programm an sich geladen, sondern wiederum nur ein Ladeprogramm, das dann das eigentliche Sektor für Sektor in den Speicher lädt. In diesem Fall spricht man von einem mehrstufigen Boot-Vorgang. Für diesen komplizierten Ablauf gibt es mehrere Gründe. Häufig möchte man während des Ladevorgangs einen Titel bzw. eine Grafik anzeigen, wozu man ihn unterbrechen muss. Das ist nur durch einen zweistufigen Boot-Vorgang zu erreichen. Außerdem ist bei kommerziellen Programmen meist ein Kopierschutz nötig, und der muss, um seinen Zweck zu erfüllen, gut versteckt sein. Daher arbeiten solche Boot-Loader meist sogar über mehr als zwei Stufen.

Sektor 1

Wie wir gesehen haben, spielt der erste Sektor eine Schlüsselrolle beim Boot-Vorgang. Betrachten wir daher den Aufbau dieses Sektors einmal genauer. Wesentlich sind dabei die ersten sechs Byte (s. Bild 1). Das erste enthält Flags, die allerdings in der derzeitigen Version des Betriebssystems nicht benutzt werden. Man setzt es gewöhnlich auf Null. Das zweite Byte gibt die Anzahl der Sektoren an, die im Rahmen der ersten Boot-Stufe geladen werden sollen. Hier können Werte von 0 bis 255 angegeben werden, wobei Null für 256 Sektoren steht. Drittes und viertes Byte ergeben zusammen die Adresse im gewohnten LSB-, MSB-Format, an die das Boot-Programm geladen werden soll. Die letzten beiden bezeichnen schließlich die Initialisierungsadresse, an der das Programm gestartet werden kann (zu diesem Thema später noch mehr). Mit dem siebten Byte beginnt der Programm-Code; hier findet sich meist ein Programm, das den Boot-Vorgang fortsetzt.

Bei näherer Betrachtung fällt eine Ungereimtheit sofort ins Auge. Wie kann die Boot-Routine wissen, wohin der erste Sektor geladen werden soll, wenn die Adresse im Sektor selbst steht? Die Lösung ist einfach: Er wird immer zuerst in den Kassettenpuffer (ab $400) geladen, dann die obige Information entnommen und nun der Sektor per Programm an die gewünschte Adresse kopiert. Eventuell nachfolgende Sektoren (je nach zweitem Byte) werden direkt an die entsprechenden Adressen geladen.

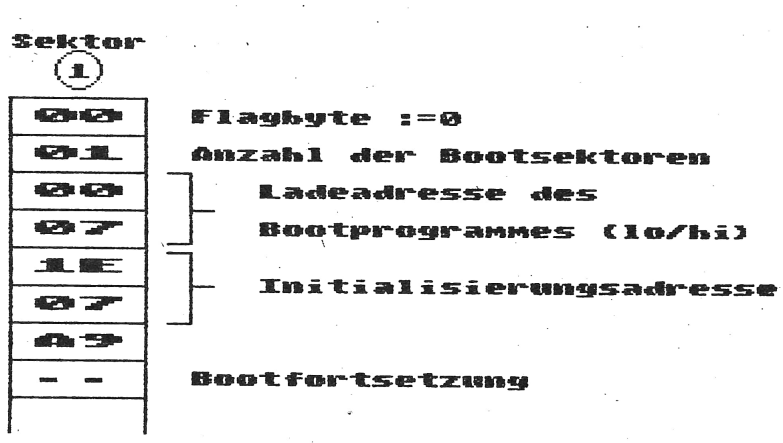

Bild 1

Boot-Ablauf

Hier noch einmal eine Zusammenfassung der Vorgänge, die beim Booten einer Diskette stattfinden :

1. Der erste Sektor wird nach $400 geladen.
2. Die ersten sechs Byte dieses Sektors werden geprüft und in interne Variablen verlagert. Das Flag-Byte kommt mach DFLAGS ($240), die Anzahl der zu ladenden Sektoren nach DBSECT ($241), die Ladeadresse nach BOOTAD ($242, $243) und die Initialisierungsadresse in den bekannten Vektor DOSINI ($0C,$0D).
3. Jetzt wird der gesamte Sektor aus dem Cassettenbuffer an die durch BOOTAD gegebene Adresse verschoben.
4. Die restlichen Sektoren werden gemäß DBSECT anschließend an den ersten Sektor geladen.

5. Nun findet ein Unterprogrammsprung an die Ladeadresse + 6 statt. Somit wird der Code gestartet, der sich an die ersten sechs Byte anschließt. Diese Routine, die normalerweise zur Fortsetzung des Boot-Vorganges benutzt wird, sollte mit einem RTS beendet sein. Ist dabei das Carry-Flag gesetzt, so nimmt das Betriebssystem an, dass ein Fehler aufgetreten ist und gibt die Meldung "BOOT ERROR" aus.
6. Mit einem weiteren Unterprogrammsprung durch den Vektor DOSINI kann das Programm initialisiert werden. Dieser Sprung findet auch statt, wenn später einmal RESET gedrückt wird. Auch diese Routine sollte mit einem RTS abgeschlossen sein.
7. Zuletzt wird durch den Vektor DOSVEC ($0A,$0B) mit einem indirekten JMP gesprungen. Wurde dieser Vektor zuvor auf den Programmstart gerichtet, dann sollte das Programm jetzt ordnungsgemäß gestartet werden.

Die Abfolge der Schritte 5 bis 7, wurde so geschildert, wie sie laut Beschreibung des Atari Betriebssystems stattfinden sollte. Das ist natürlich für eigene Programme nicht zwingend. Man kann durchaus schon ab Schritt 5 die vollständige Kontrolle ergreifen.

Die im ersten Moment etwas verwirrend erscheinenden Unterprogrammsprünge sind nötig, wenn ein DOS installiert werden soll. Sie erlauben, dass das DOS sich mit dem DOSINI -Vektor im Betriebssystem verankern kann und auch dann nicht verlorengeht, wenn RESET gedrückt wird. Im Gegensatz dazu ist der Sprung über DOSVEC nur dann erforderlich, wenn das DOS selbst aufgerufen wird.

Beispiel

Das beste Hilfsmittel zum Verständnis dieser Vorgänge ist sicher ein Beispiel. Sehen Sie sich dazu das Listing 1 an, das ein bootfähiges Programm auf eine Diskette schreibt. Tippen Sie es mit Atmas II (mit Änderungen auch auf anderen Assemblern) ein und assemblieren Sie es. Vor Programmstart muss eine formatierte Diskette eingelegt werden, die keine wertvollen Informationen enthält. Hier ist Vorsicht geboten. Das Programm schreibt auf die Diskette; entfernen Sie daher unbedingt die Atmas-Diskette aus dem Laufwerk.

Nach dem Start des Programms an der Adresse $A800 können Sie an dem charakteristischen Ton hören, wie ein Sektor auf die Diskette geschrieben wird. Wenn Sie nun den Computer ausschalten, etwas warten und mit gedrückter OPTION-Taste wieder einschalten, wird das soeben geschriebene Programm gebootet. Der erste Sektor wird geladen, und das Demo-Programm (das Atari-Farbscrolling) läuft an. Drücken Sie auch ruhig einmal auf RESET; Sie werden sehen, dass nichts passiert.

So funktioniert's

Das Programm besteht aus zwei getrennten Teilen, dem Boot-Generator, der das Programm auf die Diskette schreibt, sowie dem Boot-Programm selbst. Da letzteres sehr kurz ist, wird auch der BootGenerator sehr einfach, denn es muss nur ein einziger Sektor geschrieben werden.

Der Boot-Generator benutzt den eingebauten Disk-Handler, der über den Vektor DSKINV ($E453) angesprochen wird. Zuvor muss der DCB (Device Control Block) mit einigen Informationen wie Sektornummer, Bufferadresse und Laufwerknummer versorgt werden.

Im Boot-Programm wird zunächst der Boot-Header (die beschriebenen sechs Byte) mit DFB- und DFW-Befehlen definiert. Beachten Sie auch den ORG-Befehl, der das Programm zwar ab Adresse $A900 im Speicher ablegt, aber so assembliert, dass es ab Adresse $700 (im Programm BOOTADR) lauffähig ist.

Der erste Einsprung in das Programm erfolgt an der Adresse BOOTPGM+6. Da es im Beispiel keine Notwendigkeit gibt, den Boot-Vorgang fortzusetzen, wird hier nur der Vektor DOSVEC auf den Programmeinsprung (SCRCOL) gerichtet. Mit CLC wird signalisiert, dass kein Fehler aufgetreten ist, und ein RTS gibt die Kontrolle ans Betriebssystem zurück.

Nach der geschilderten Boot-Sequenz folgt nun ein Sprung durch DOSINI, der im Beispiel jedoch nicht gebraucht wird. Er ist daher direkt auf ein RTS gerichtet (Label BOOTINI).

Bevor der Aufruf des Programms über DOSVEC erfolgt, wird bei den XL/XE-Computern das Betriebssystem noch einmal aktiv. Es prüft, ob irgendwelche externen Geräte am parallelen Bus (!) angeschlossen sind und ob sich ein Steckmodul im Schacht befindet. Zu letzteren zählt auch das eingebaute Basic; daher muss beim Booten die OPTION-Taste gedrückt werden.

Versuchen Sie es einmal ohne. Sie werden sehen, dass sich dann Basic mit "READY" meldet. Wie man das verhindern kann, werden wir uns in der nächsten Assemblerecke anschauen. Dort wird dann auch ein Programm vorgestellt, das aus (fast) jedem beliebigen Maschinenprogramm eine Boot-Diskette erzeugen kann.

Peter Finzel

```
**************************************
*
* BEISPIEL: Erzeugen einer boot-
*           faehigen Diskette
*
* P. Finzel                  1986
**************************************

BOOTADR         EQU $700        Anfang Bootpgm.
ABLAGE          EQU $A900       Ablage des Bootpgm.
DRIVE           EQU 1           Aktuelles Laufwerk
RTCLK           EQU $14         VBI-Uhr
DOSVEC          EQU $0A         Reset-Vektor
COLPF2          EQU $D018       Farbregister
WSYNC           EQU $D40A       Synchron.
VCOUNT          EQU $D40B       Rasterzeile
DSKINV          EQU $E453       Disk-Handler-Einsprung
*
* DBC-Kontrollblock
*
DUNIT           EQU $301        Drive-Nummer
DCOMND          EQU $302        Disk-Befehl
DBUFLO          EQU $304        Bufferadresse lo
DBUFHI          EQU $305        ;  -"-          hi
DAUX1           EQU $30A        Sektornummer  lo
DAUX2           EQU $30B        ;  -"-          hi

*-----------------------------------------
* BOOTGENERATOR
*-----------------------------------------

                ORG $A800

                LDA #1          Sektor 1
                LDX #ABLAGE:L   Bufferadresse
                LDY #ABLAGE:H
                JSR PUTSEC      Sektor schreiben
                RTS

*-----------------------------------------
* Sektor schreiben:
*         (A): Sektornummer
*         (X): Buffer LSB
*         (Y): Buffer MSB
*-----------------------------------------

PUTSEC          STA DAUX1       Sektornummer
                LDA #0
                STA DAUX2
                STX DBUFLO
                STY DBUFHI
                LDA #DRIVE      Drive 1
                STA DUNIT
                LDA #'W         Sektor lesen
                STA DCOMND
                JSR DSKINV
                RTS
```

```
*-----------------------------------------
* Bootprogramm
*
* Dieses Programm wird in den Boot-
* sektor geschrieben
*-----------------------------------------

               ORG BOOTADR,ABLAGE

BOOTPGM        DFB 0
               DFB 1
               DFW BOOTADR
               DFW BOOTINI
*
* Einsprung 'Bootfortsetzung'
*
               LDA #SCRCOL:L  Einsprungadresse
               STA DOSVEC     nach DOSVEC
               LDA #SCRCOL:H
               STA DOSVEC+1
               CLC            kein Fehler
               RTS
*
* Einsprung durch DOSVEC
*
SCRCOL         CLC
               LDA VCOUNT     Bild-Zaehler
               ADC RTCLK      plus Raster-Zeile
               STA WSYNC      Warte auf HSYNC
               STA COLPF2     Farbregister
               JMP SCRCOL
*
* Einsprung durch DOSINI
*
BOOTINI        RTS            keine Init.
```