Matthias Köfferlein

Disk-Editor für CP/M-68k

Das hier beschriebene Programm ermöglicht es, einzelne Sektoren einer Diskette in den Speicher zu laden und auszugeben, Änderungen am Inhalt dieser Sektoren vorzunehmen und den geänderten Inhalt auch wieder auf die Diskette zurückzuschreiben. Er gibt außerdem die Disketten-Parameter an, also die Zahl der reservierten Systemspuren oder der Spuren insgesamt.

Das Programm besteht aus zwei Teilen. Der größte Teil des Programms ist in C geschrieben (*Bild 1*). Lediglich die Einbindung des Programms in das Betriebssystem erfolgt über Assemblerroutinen, die den anderen Teil bilden (*Bild 2*). Beide Programmteile müssen mit einem Editor (z. B. ED.68k) eingegeben werden. Dabei können die Kommentare natürlich weggelassen werden. Nachdem man beide Teile eingegeben hat, müssen sie übersetzt werden. Hat man z.B. das C-Programm unter dem Namen DISKED.C abgespeichert, so erfolgt das Übersetzen dieses Programmteils mit der Sequenz

C DISKED

C.SUB ist die Submit-Datei, die bei CP/M-68k zum C-Compiler mitgeliefert wird. Der Assemblerteil wird, wenn er unter dem Namen CPMC.S abgespeichert wurde, mit der Sequenz

AS68 -L CPMC.S

übersetzt. Danach müssen noch beide Teile mit dem Linker gebunden werden. Dies geschieht mit dem Kommando

LO68 -R -U_NOFLOAT -O DISKED.REL S.O DISKED.O CPMC.O CLIB

Man erhält damit die direkt ausführbare relokatible Datei DISKED.REL.

Anpassung

Um das Programm an die eigene Hardware anzupassen, müssen eventuell die Steuercodes für den Bildschirm geändert werden. Die Steuercodes sind alle am Anfang des Programms als Makros mit der #define-Anweisung des C-Präprozessors definiert. Auch die Funktion für 'Cursor direkt setzen' ist direkt als Funktion per #define-Anweisung definiert. In der abgedruckten Version arbeitet das Programm mit den Steuerzeichen des mc-68000-Computers.
Nachdem das Programm gestartet wurde, wird zuerst der Bildschirm gelöscht und die Bildschirm-Maske ausgegeben. Danach stellt das Programm fest, welches Laufwerk aktiv war, als es gestartet wurde, liest Sektor 0, Spur 0 von der Diskette in den Speicher und gibt den Inhalt dieses Bereichs aus. Gleichzeitig liest das Programm die Parameter des angesprochenen Laufwerks ein und zeigt diese im rechten unteren Bildschirmviertel an.

Die Kommandos

Nach der Initialisierungsphase wartet das Programm auf ein Kommando von der Tastatur. Der Speicherinhalt wird als ein Feld von 16 mal 8 Bytes dargestellt; mit den Cursortasten kann nun der Cursor innerhalb der Darstellung verscho-

```
#define CLS 12                          /* Definitionen: Schirm loeschen */
#define UP 30                           /*   Cursor nach oben */
#define DOWN 31                         /*   Cursor nach unten */
#define LEFT 29                         /*   Cursor nach links */
#define RIGHT 28                        /*   Cursor nach rechts */
#define ESC 27                          /*   Escape */
#define BS 8                            /*   Backspace */
#define CR 13                           /*   Carriage Return */
#define HOME 11                         /*   Code fuer Taste: Home */
#define END 17                          /*                    Linke u. Ecke */

#define scan() bios(3, 0)               /* Keyboard abfragen: BIOS Fkt. #3 */
#define goxy(xp,yp) printf("\033y%c%c",(xp)+32,(yp)+32)
                                        /* Sequenz fuer Cursor absolut setzen */
#define clrel() printf("\033k")         /* Sequenz fuer Zeilenende loeschen */

char buffer[128];                       /* Puffer fuer Sektor */
int sect = 0;                           /* Sektornummer (physikalisch) */
int track = 0;                          /* Tracknummer */
int cursx = 0;                          /* Position des Cursors */
int cursy = 0;
int drive;                              /* Nummer des Drives (0:A usw.) */
char hexl[] = "0123456789abcdef";       /* ASCII-Codes der Hex-Zahlen */
char hexu[] = "0123456789ABCDEF";       /* Auch als upper-case Buchstaben */

struct dpb {                            /* Disk parameter block */
        int spt;                        /* Zahl der records (128 Byte)/track */
        char bsh;                       /* 3: 1024, 4: 2048, 5: 4096 usw ... */
        char r1;                        /* 3 unbenuetzte Bytes */
        char r2;
        char r3;
        int dsm;                        /* Diskgroesse in Blocks minus 1 */
        int drm;                        /* Directory-Eintraege minus 1 */
        int r4;                         /* 2 unbenuetzte Worte */
        int r5;
        int off;                        /* Zahl der reservierten Tracks */
};
struct dph {                            /* Disk Parameter Header */
        char *xlt;                      /* Addresse der translation table */
        int r1;                         /* 5 unebenuetzte Woerter */
        int r2;
        int r3;
        long r4;
        struct dpb *diskp;              /* Addresse des DPB */
};
char *xtable;                           /* Zeiger auf translation table */
struct dpb *diskpar;                    /* diskpar: Enthaelt Diskparameter */
int tracks;                             /* Anzahl der Tracks */

extern long bdos();                     /* Funkion zur Einbindung des BDOS */
extern long bios();                     /* Funktionen zur Einbindung des BIOS */
extern long biosl();                    /* Fuer Longword-Parameter */
extern int sectxlt();                   /* Sector-Translation */
```

Bild 1. Der Disk-Editor ist überwiegend in C geschrieben

```
main()                          /* Hauptprogramm */
{
char c = '\0';

drive = bdos(25, 0);            /* BDOS Fkt. #25 (Get current disk) */
title();                        /* Titelbild */
login();                        /* Diskparameter holen */
reads();                        /* Sektor 0,0 lesen */
display();                      /* Pufferinhalt ausgeben */
while(c != ESC) {               /* ESC beendet Hauptprogramm */
        goxy(cursx * 3 + 5, cursy + 5);
                                /* Cursor setzen */
        c = scan();             /* Keyboard abfragen */
        switch (c) {
        case DOWN:
                if (++cursy < 8)        /* Cursor nach unten bewegen */
                        break;
        case UP:
                if (cursy)              /* Cursor nach oben bewegen */
                        cursy--;
                break;
        case 'e':
        case 'E':
                editby();               /* E: Byte aendern */
                displn(cursy);          /* Zeile ausgeben und weiter */
        case RIGHT:
                if (++cursx < 16)       /* Cursor nach rechts bewegen */
                        break;
                if (++cursy < 8) {
                        cursx = 0;
                        break;
                }
                cursy--;                /* Kein break: Cursor links */
        case LEFT:
                if (cursx) {            /* Cursor nach links bewegen */
                        cursx--;
                        break;
                }
                if (cursy) {
                        cursy--;
                        cursx = 15;
                }
                break;
        case 's':
        case 'S':
                setts();                /* S: Track und Sektor setzen */
                break;
        case 'r':
        case 'R':
                reads();                /* R: Sektor lesen */
                display();              /* Pufferinhalt ausgeben */
                break;
        case 'w':
        case 'W':
                if (ask())              /* Sicherheitsabfrage fuer W: */
                        writes();       /* Sektor schreiben */
                break;
        case ESC:
                if (!ask())             /* Sicherheitsabfrage bei ESC */
                        c = 0;          /* Bei Nein: ESC entschaerfen */
                break;
        case HOME:
                cursy = cursx = 0;      /* HOME: Cursor n. rechts o. */
                break;
        case END:
                cursy = 7;              /* END: Cursor n. links u. */
                cursx = 0;
                break;
        case 'l':
        case 'L':
                seldsk();               /* L: Laufwerk anwaehlen */
                dispts();               /* Statuszeile aktualisieren */
                break;
        }
        putchar(CLS);                   /* Zurueck ins System */
        bdos(0, 0);                     /* BDOS Fkt. #0 (System reset) */
}

title()                         /* Titelbild ausgeben */
{
        int i;

        putchar(CLS);
        printhl();
        puts("\t\t     SEKTOR <T:  ,S:  > LAUFWERK <  : >");
        puts("\n     x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xA xB xC xD xE xF\n");
        puts("\n\n\n\n\n\n\n\n\n\n\n\t    <S> TRACK UND SEKTOR SETZEN");
        puts("\t    <R> SEKTOR LESEN\t\t       RECORDS PRO TRACK <  >");
        puts("\t    <W> SEKTOR SCHREIBEN\t\t    BLOCKGROESSE <    >");
        puts("\t    <E> ANGEWAEHLTES BYTE AENDERN\tDISKETTENGROESSE <    >");
        puts("\t    <L> LAUFWERK ANWAEHLEN\t     DIRECTORY-EINTRAEGE <    >");
        puts("\t  <ESC> EDITOR VERLASSEN\t      RESERVIERTE TRACKS <  >");
        puts(" <CURSORTASTEN> CURSOR BEWEGEN\t\t\t\t  TRACKS <    >");
}

display()                       /* Pufferinhalt als Hexdump ausgeben */
{
        int i;

        for (i = 0; i < 8; i++)         /* 8 Zeilen Hexdump ausgeben */
                displn(i);
        dispts();                       /* Statuszeile aktualisieren */
}

dispts()                        /* Statuszeile aktualisieren */
{
        goxy(28, 1);                    /* Track & Sektor ausgeben */
        printf("<T:%02x,S:%02x>", track, sect);
                                        /* T & S als zweistellige Hexzahlen */
        goxy(49, 1);                    /* Drivenummer und Drivecode ausgeben */
        printf("<%02x:%c>", drive, drive + 'A');
}

displn(line)                    /* Eine Zeile Hexdump ausgeben */
int line;                       /* Zeile: 0..7 */
{
        int i;
        char *p, ascii[17];             /* ascii[]: Puffer fuer ASCII-Dump */

        goxy(0, line + 5);              /* Cursor positionieren */
        printf("%cx   ",line + 48);     /* Zeilennummer ausgeben (0x .. 7x) */
        p = buffer + line * 16;         /* p: Zeiger auf auszugebendes Byte */
        for (i = 0; i < 16; i++) {
                printf("%02x ", (int)*p & 0xff);
                                        /* Byte als 2-st. Hexzahl ausgeben */
                ascii[i] = (*p & 0x60) ? *p & 0x7f : '.';
                                        /* ASCII-Dump-String aufbauen: */
                                        /* ' ' falls Zeichen kein ASCII-Code */
                p++;
        }
        ascii[16] = '\0';               /* ASCII-Dump-String abschliessen */
        printf("  %s", ascii);          /* Und ausgeben */
}

readby(def)                     /* Byte als 2-stellige Hex-Zahl eingeben */
int def;                        /* def: default-Wert */
{
        int c, i;
        int pos = 0;                    /* pos: Eingabeposition (0 .. 2) */
        int by = 0;                     /* by: Puffer fuer Byte */

        printf("  %c%c", LEFT, LEFT);   /* Feld loeschen */
        while ((c = scan()) != CR) {    /* CR beendet Eingabe */
                for (i = 0; i < 16; i++) {
                        if (hexu[i] == c || hexl[i] == c) {
```

```
                                        /* Falls Hex-Ziffer: */
                        if (pos != 2) {
                                by = 16 * by + i;
                                        /* Nibble abspeichern */
                                pos++;
                                putchar(c);
                        }
                }
        }
        if (c == BS) {
                if (pos) {              /* Backspace: Nibble loeschen */
                        putchar('\b');
                        by /= 16;
                        pos--;
                }
        }
        if (c == ESC) {
                while (pos) {           /* ESC: default-Werte holen */
                        pos--;
                        putchar(LEFT);
                }
                printf("%02x", by = def);
                                        /* Und als Byte abspeichern */
                pos = 2;
        }
    }
    return(by);                         /* Byte zurueckgeben */
}

editby()                                /* Byte editieren */
{
    char *p;                            /* Addresse des aktuellen Bytes */

    p = buffer + cursx + cursy * 16;
    *p = readby((int)*p & 0xff);        /* Byte editieren */
}

setts()                                 /* Track und Sektor setzen */
{
    goxy(31, 1);                        /* Cursor aus Track-Feld */
    track = readby(track) % tracks;     /* Track eingeben */
    dispts();                           /* Eingegebenen Wert anzeigen */
    goxy(36, 1);                        /* Dasselbe mit der Sektornummer */
    sect = readby(sect) % (diskpar -> spt);
    dispts();
}

reads()                                 /* Sektor lesen */
{
    int stat;                           /* stat: Status beim Lesen */

    setsek();                           /* Sektorparameter setzen */
    stat = bios(13, 0);                 /* BIOS Fkt. #13 (Read sector) */
    if (stat) {                         /* Falls stat == 1: Fehler */
        putchar(HOME);
        clrel();
        printf("READ ERROR AUF <T:%02x,S:%02x>", track, sect);
    }
    return(stat);
}

setsek()                                /* Sektorparameter setzen */
{
    bios(9, drive);                     /* BIOS Fkt. #9 (Select disk drive) */
    bios(10, track);                    /* BIOS Fkt. #10 (Set track number) */
    bios(11, sectxlt(xtable, sect));
                                        /* BIOS Fkt. #11 (Set sector number) */
    biosl(12, buffer);                  /* BIOS Fkt. #12 (Set DMA address) */
}

writes()                                /* Sektor schreiben */
{
    int stat;                           /* stat: Status beim Schreiben */

    setsek();                           /* Sektorparameter setzen */
    stat = bios(14, 1);                 /* BIOS Fkt. #14 (Write sector) */
                                        /* Directory-Sektor schreiben (1) */
    if (stat) {                         /* stat == 1: Schreibfehler */
        putchar(HOME);
        clrel();
        printf("WRITE ERROR AUF <T:%02x,S:%02x>", track, sect);
    }
}

ask()                                   /* Sicherheitsabfrage */
{
    int c;

    putchar(HOME);                      /* Ueberschrift loeschen */
    clrel();
    printf("SIND SIE SICHER <J/N> ? ");
    c = scan();                         /* Keyboard abfragen */
    printhl();                          /* Ueberschrift wieder schreiben */
    return(c == 'j' || c == 'J');
                                        /* Rueckgabe: 1: JA, 0: Sonst */
}

printhl()                               /* Ueberschrift ausgeben */
{
    putchar(HOME);
    puts("DISKEDITOR CP/M 68k, MATTHIAS KOEFFERLEIN 1986");
}

seldsk()                                /* Drive auswaehlen */
{
    goxy(50, 1);                        /* Cursor auf Feld positionieren */
    drive = readby(drive) & 0xf;        /* Drivenummer einlesen (0 .. 15) */
    login();
}

login()                                 /* Login fuer aktuelle Disk */
{
    int stat;
    struct dph *p;                      /* Zeiger auf DPH */

    track = 0;                          /* Track und Sektor auf 0 setzen */
    sect = 0;
    dispts();                           /* Und anzeigen */
    p = bios(9, drive);                 /* BIOS Fkt. #9 (Select disk drive) */
    xtable = p -> xlt;                  /* Addresse der translation table */
    diskpar = p -> diskp;               /* Addresse des DPB */
    dispdpb();                          /* DPB anzeigen */
}

dispdpb()                               /* DPB anzeigen */
{
    int bls;                            /* bls: Blockgroesse */

    bls = 128 << (diskpar -> bsh);      /* Blockgroesse berechnen */
    goxy(66, 17);                       /* SPT ausgeben */
    printf("%02x", diskpar -> spt);
    goxy(66, 18);                       /* BSH ausgeben */
    printf("%04x", bls);
    goxy(66, 19);                       /* DSM ausgeben (in kBytes) */
    printf("%04x", ((diskpar -> dsm) + 1) * (bls >> 10));
    goxy(66, 20);                       /* DRM ausgeben (Dir.-entries) */
    printf("%04x", (diskpar -> drm) + 1);
    goxy(66, 21);                       /* OFF ausgeben (resvd tracks) */
    printf("%02x", diskpar -> off);
    tracks = (diskpar -> off) +
             ((long)((diskpar -> dsm) + 1) * bls) / (128 * (diskpar -> spt));
                                        /* Anzahl der Tracks berechnen */
    goxy(66, 22);                       /* Und ausgeben */
    printf("%04x", tracks);
}
```

ben werden. Alle Kommandos werden zur Erinnerung im linken unteren Bildschirmviertel angezeigt. Als Kommandos stehen außer den Cursortasten zur Verfügung:

Byte ändern (E)
Mit diesem Kommando kann das Byte, auf das der Cursor zeigt, verändert werden. Die Eingabe wird hexadezimal erwartet. Wird keine Ziffer eingegeben, so wird der Wert 0 angenommen. Mit der ESC-Taste kann der alte Wert in das Eingabefeld übernommen werden. Return beendet die Eingabe.

Spur-/Sektornummer setzen (S)
Damit kann die Spur- und Sektornummer für einen folgenden Schreib- bzw. Lesezugriff gesetzt werden. Zuerst wird die Eingabe der Spurnummer erwartet. Dabei gelten die selben Regeln, wie sie eben beim E-Befehl erläutert wurden. Nach Druck auf die CR-Taste muß die Sektornummer eingegeben werden, die Eingabe erfolgt wie bei der Spurnummer. Werden bei der Eingabe die zulässigen Grenzen für Spur- bzw. Sektornummer überschritten, so werden diese modulo der jeweils höchsten zulässigen Nummer genommen.

Die Sektornummer entspricht der logischen Sektornummer auf der Disk. Damit entfallen lästige Berechnungen, wenn bei dem verwendeten Format die Sektoren nicht hintereinander auf der Disk liegen, sondern mit Zwischenräumen auf der Diskette untergebracht sind (Sector translation).

Sektor lesen (R)
Der Sektor, dessen Adresse mit dem S-Kommando eingegeben wurde, wird in den Speicher eingelesen. Ein eventuell auftretender Lesefehler wird in der obersten Zeile signalisiert.

Sektor schreiben (W)
Damit diese Funktion nicht versehentlich ausgelöst werden kann, ist zusätzlich eine Sicherheitsabfrage eingebaut, die mit J beantwortet werden muß.

Trotzdem hindert das Programm den Benutzer nicht daran, wertvolle Daten zu zerstören. Deshalb sollte man den Befehl äußerst vorsichtig benutzen.

Laufwerk ändern (L)
Mit diesem Kommando kann das aktuelle Laufwerk gewechselt werden. Die Eingabe der Laufwerksnummer erfolgt hexadezimal. Für Laufwerk A muß 0 eingegeben werden, für B 1, C 2 usw. Die Laufwerksnummer wird immer modulo 16 genommen. Der Buchstabe des Laufwerks wird hinter der Nummer angegeben. Für die Eingabe gelten dieselben Regeln, wie für das S- oder E-Kommando. Nachdem die Laufwerksnummer eingegeben wurde, liest das Programm die Parameter des angewählten Laufwerks ein und stellt sie im rechten unteren Bildschirmviertel dar. Dabei sind alle Angaben hexadezimal. Die Blockgröße wird im Bytes, die Diskettengröße in KBytes angegeben. Weiterhin wird die Anzahl der Spuren, der Sektoren pro Spur (ein Sektor entspricht 128 Bytes), die Zahl der Einträge im Inhaltsverzeichnis und die Anzahl der reservierten Systemspuren angezeigt.

Programm beenden (ESC)
Mit ESC kehrt das Programm ins Betriebsystem zurück. Vorher wird jedoch noch eine Sicherheitsabfrage durchgeführt, die mit J beantwortet werden muß.

Das Inhaltsverzeichnis

Bild 3 zeigt eine Hardcopy des Bildschirms nach dem Start des Programms. Das Diskettenformat auf diesem Bild ist das Format, das der mc-68000-Computer benützt. Bei diesem besteht eine Spur aus 40 (Hexadezimal $28) Sektoren. Vier Systemspuren sind für Ladeprogramm und Betriebsystem reserviert. Beim mc-68000-Computer liegt das CP/M jedoch im EPROM, so daß sich auf diesen Spu-

```
* ------------------------------------------------------------------------
*
*       Einbindung der BDOS- und BIOS-Funktionen in das C-Programm:
*
*               _sectxtl:       Sector translation
*               _bdos:          BDOS-Aufruf
*               _bios:          BIOS-Aufruf
*               _biosl:         BIOS-Aufruf mit Langwort-Argument
*
* ------------------------------------------------------------------------
*
*       Beim Aufruf der Funktionen wird der Wert im D0-Register als
*       Funktionswert zurueckgegeben.
*       Defaultmaessig wird ein Word-Parameter zurueckgegeben. Werden die
*       Funktionen mit
*
*               extern long bios(), biosl(), bdos();
*
*       definiert, so wird D0 als Langwort von C uebernommen.
*       Aufruf der Funktionen:
*
*               ret = biosl((int)function#, (long)arg);
*               ret = bios((int)function#, (int)arg);
*               ret = bdos((int)function#, (int)arg);
*               xsect = sectxtl((char *)table, (int)sect);
*
* ------------------------------------------------------------------------
        .globl _biosl                   * Als globale Funktionen definieren
        .globl _bios
        .globl _sectxlt
        .globl _bdos
* ------------------------------------------------------------------------
        .text

_bios:
        move.w 4(sp),d0                 * Funktionsnummer vom Stack holen
        move.w 6(sp),d1                 * Argument vom Stack holen
        clr.w d2                        * D2 loeschen
        trap #3                         * BIOS-Aufruf
        rts                             * Zurueck ins C-Programm

_biosl
        move.w 4(sp),d0                 * Dasselbe mit Long-Parameter
        move.l 6(sp),d1
        clr.w d2
        trap #3
        rts

_bdos:
        move.w 4(sp),d0                 * Einbindung des BDOS in C
        move.w 6(sp),d1                 * (wie BIOS)
        trap #2
        rts

_sectxlt:                               * Sector translation
        move.l 4(sp),d2                 * Addresse der Tabelle vom Stack
        move.w 8(sp),d1                 * Sektornummer vom Stack
        move.w #16,d0                   * BIOS Fkt. #16 (Sector translate)
        trap #3                         * BIOS-Aufruf
        rts
* ------------------------------------------------------------------------
        end
```

Bild 2. Diese Assemblerprogramme stellen die Verbindung zwischen dem C-Programm und dem Betriebssystem her

```
DISKEDITOR CP/M 68k, MATTHIAS KOEFFERLEIN 1986
                     SEKTOR <T:04,S:04> LAUFWERK <00:A>

      x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xA xB xC xD xE xF

0x    00 4A 4F 59 20 20 20 20 20 4F 20 20 00 00 00 06   .JOY      O  ....
1x    73 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   s...............
2x    00 43 50 36 38 20 20 20 20 36 38 4B 00 00 00 80   .CP68    68K....
3x    5C 00 5D 00 5E 00 5F 00 60 00 61 00 62 00 63 00   \.].^._.'.a.b.c.
4x    00 43 50 36 38 20 20 20 20 36 38 4B 01 00 00 47   .CP68    68K...G
5x    64 00 65 00 66 00 67 00 68 00 00 00 00 00 00 00   d.e.f.g.h.......
6x    00 53 54 44 49 4F 20 20 20 48 20 20 00 00 00 1D   .STDIO   H  ....
7x    19 00 1A 00 00 00 00 00 00 00 00 00 00 00 00 00   ................


          <S> TRACK UND SEKTOR SETZEN
          <R> SEKTOR LESEN                    RECORDS PRO TRACK <28>
          <W> SEKTOR SCHREIBEN                     BLOCKGROESSE <0800>
          <E> ANGEWAEHLTES BYTE AENDERN       DISKETTENGROESSE <030C>
          <L> LAUFWERK ANWAEHLEN             DIRECTORY-EINTRAEGE <0100>
        <ESC> EDITOR VERLASSEN                RESERVIERTE TRACKS <04>
<CURSORTASTEN> CURSOR BEWEGEN                           TRACKS <00A0>
```

Bild 3. So präsentiert sich der Disk-Editor dem Benutzer auf dem Bildschirm

ren nur Konfigurations-Informationen befinden. Die Systemspuren liegen auf Spur 0...3, in Spur 4 schließt sich das Inhaltsverzeichnis an. Es umfaßt bei diesem Format 256 ($100) Einträge. Jeder Eintrag besteht aus 32 Bytes. Das gesamte Inhaltsverzeichnis umfaßt also 8192 Bytes, dies entspricht, bei der verwendeten Blockgröße von 16 Sektoren und somit 2048 (= $800) Bytes pro Block, vier Blöcken. Die Blöcke sind sequentiell angeordnet, d. h. Block 0 beginnt auf Spur 4, Sektor 0. Block 1 beginnt auf Spur 4, Sektor 16, und Block 2 auf Spur 4, Sektor 32. Da dieser Block über das Ende der Spur hinausreicht, werden auch noch die ersten 8 Sektoren der nachfolgenden Spur zu diesem Block hinzugenommen, so daß Block 3 bei Spur 5, Sektor 8 beginnt. Da das Inhaltsverzeichnis 4 Blöcke umfaßt, beginnen die eigentlichen Daten bei Block 4, also bei Spur 5, Sektor 24. Das Bild zeigt den Inhalt der Spur 4, Sektor 4, also einen Ausschnitt aus dem Inhaltsverzeichnis. Dies besteht aus Einträgen von je 32 Byte Länge. Jeder Eintrag adressiert 8 Blöcke, so daß ein Eintrag 8 × 2048 = 16384 Bytes adressieren kann. Ist eine Datei größer, so nimmt sie entsprechend viele Einträge in Anspruch (Extensions). Jeder Eintrag im Inhaltsverzeichnis entspricht funktionsmäßig den ersten 32 Byte eines FCB (File control block), und ist folgendermaßen aufgebaut:

Byte 0: Ist dieses Byte mit $E5 beschrieben, so gilt der Eintrag als gelöscht.

Durch Überschreiben mit $00 können also gelöschte Dateien wieder hergestellt werden. Dabei ist zu beachten, daß eine Datei auch aus mehreren Extensions bestehen kann, die alle mit $00 gekennzeichnet werden müssen. Diese Methode funktioniert außerdem auch nur dann einwandfrei, wenn nach dem Löschen keine Schreib-Operationen auf der Diskette durchgeführt wurden. Hat dieses Byte einen Wert ungleich $E5, so gibt es die Usernummer an, unter der die Datei angesprochen werden kann. Dieses Byte unterscheidet sich in der Bedeutung somit von der Bedeutung im FCB, wo es die Drivenummer angibt.

Byte 1...11: Diese Bytes enthalten den Dateinamen: Die ersten acht Bytes enthalten den Namen, die folgenden drei Bytes die Kennung. Unbenützte Bytes werden mit $20 (Leerzeichen) aufgefüllt. Die drei Bytes der Kennung enthalten zusätzlich Informationen über den Status der Datei in den höchstwertigen Bits. Nähere Informationen darüber gibt [1].

Byte 12: Dieses Byte enthält die Extensionnummer. $00 bedeutet, daß dieser Eintrag der erste Eintrag der Datei ist.

Byte 13...14: Diese Bytes haben keine Bedeutung.

Byte 15: Dieses Byte gibt an, wieviele Sektoren (128 Bytes je Sektor) durch diesen Eintrag adressiert werden. Ist der Eintrag voll aufgefüllt, so enthält dieses Byte bei einer Blockgröße von 16 Sektoren, also den Wert 16 × 8 = 128 ($80).

Byte 16...31: Diese Bytes stellen eine Liste dar, in der die Blöcke angegeben werden, die durch diesen Eintrag belegt werden. Dabei müssen zwei grundsätzliche Fälle betrachtet werden. Sind auf der Diskette weniger als 256 Blöcke vorhanden, so wird jedem Block ein Byte dieser Liste zugeordnet, wobei nur die Bytes 16...23 benützt werden. Sind auf der Diskette mehr als 256 Blöcke vorhanden, so werden jedem Block der Liste 2 Bytes zugeordnet. Das erste der beiden enthält das Low-Byte der Blocknummer, das zweite Byte das dazugehörige High-Byte.
Auf eine Möglichkeit der Manipulation wurde schon hingewiesen. Man kann mit dem Diskeditor natürlich z. B. den Status einer Datei nachträglich ändern oder Blöcke aus der Datei löschen. Man sollte diese Manipulation nur an Sicherheitskopien durchführen, da zerstörte Daten sonst kaum wiederbeschafft werden können.

Literatur

[1] Digital Research: CP/M 68k Operating System Programmer's Guide, 1983.
[2] Digital Research: The C-Language Programming Guide, 1983.
[3] Digital Research: CP/M 68k Operating System Guide, 1983.
[4] *Kernighan, Brian W.:* Programmieren in C, Hanser-Verlag, Wien; 1983.

Programmieren in Assembler

Während der Entwicklung eines Programms in Assembler oder einer Hochsprache muß unzählige Male der Editor gestartet, die Quelldatei geladen, die eingegebenen Daten abgespeichert, der Assembler/Compiler und der Linker gestartet werden. Diese mühseligen und zeitraubenden Arbeiten können mit dem von Lauer & Wallwitz entwickelten SAL-System schneller durchgeführt werden. SAL (Structured and Assembly Language) bietet eine komfortable Benutzeroberfläche für den Programmierer mit den Komponenten:

- eingebauter Editor
- interaktive Schnittstelle zu Assembler, Compiler, Linker, Debugger und Library Manager
- komfortable DOS-Schnittstelle.

Da ähnliche Steueranweisungen wie in Turbo-Pascal verwendet werden können, ist eine einfachere Programmierung in Assembler möglich. Repeat..Until, IF-..Then..Else, Case, While..Do werden von SAL verstanden und in MASM-Quelltext übersetzt. Der Assembler-Programmierer kann damit Programme in kürzerer Zeit erstellen. *Dieter Strauß*