# howto write a bios

**howto write a bios for cp/m 2.2**
**V. Pohlers, 2012**

Zu CP/M 2.2 gibt es eine Datei cbios.asm mit, die als Ausgang für ein BIOS dienen kann. Diesen Code habe ich hier genutzt, an Z80 angepasst und ggf. weiter vereinfacht.

Das BIOS folgt unmittelbar auf CCP und BDOS. Am Anfang des BIOS steht ein Sprungverteiler zu den 17 BIOS-Funktionen.

Die BIOS-Funktionen müssen keine Register retten.

## Kalt- und Warmstart

```
        .Z80
;   Skeletal CBIOS for first level of CP/M 2.0 alteration
;
CCP     EQU    xx00H        ;base of ccp
BDOS        EQU    CCP+806H    ;base of bdos
BIOS        EQU    CCP+1600H    ;base of bios

CDISK       EQU    0004H        ;current disk number 0=A,...,15=P
IOBYTE      EQU    0003H        ;intel i/o byte
;
        ORG    BIOS        ;origin of this program

;   jump vector for individual subroutines
        JP    BOOT        ;cold start
WBOOTE:     JP    WBOOT        ;warm start
        JP    CONST        ;console status
        JP    CONIN        ;console character in
        JP    CONOUT        ;console character out
        JP    LIST        ;list character out
        JP    PUNCH        ;punch character out
        JP    READER        ;reader character out
        JP    HOME        ;move head to home position
        JP    SELDSK        ;select disk
        JP    SETTRK        ;set track number
        JP    SETSEC        ;set sector number
        JP    SETDMA        ;set dma address
        JP    READ        ;read disk
        JP    WRITE        ;write disk
        JP    LISTST        ;return list status
        JP    SECTRAN        ;sector translate
```

**BOOT** ist der Eintrittspunkt, der nach dem BOOT-Vorgang einmalig angesprungen werden sollte. Vom

CP/M aus wird niemals BOOT aufgerufen. Im einfachsten Fall kann BOOT im BIOS-Bereich komplett leer bleiben, wenn die Systeminitialisierung im BOOT-Loader erfolgt.

Der Kaltstart wird nur nach dem erstmaligen Laden des Betriebsystems benötigt. Der Aufruf an den Kaltstart erfolgt meist von einem speziellen Ladeprogramm, das nach dem Einschalten des Rechners das BIOS von der Systemdiskette geladen hat.

Aufgabe des Kaltstart ist es, die einzelnen Systemkomponenten zu initialisieren und eine Meldung über den erfolgten Systemstart auf der Console auszugeben.

Normalerweise wird nach einem Kaltstart ein Warmstart ausgeführt, der den CCP und das BDOS von der Diskette lädt und die Sprungbefehle einsetzt. Dann braucht nur die zuerst anzuwählende Disk- und Usernummer in die Speicheradresse 0004h eingetragen zu werden.

**WBOOT** wird beim Warmstart (z.b. ^C oder JP 0000) aufgerufen. Die BDOS- Funktion 0 geht direkt zur BIOS-Funktion WBOOT weiter.

WBOOT muss CCP und BDOS im Speicher restaurieren. Für ein erstes BIOS kann dies übergangen werden, solange keine Programe gestartet werden, die das CCP oder sogar CCP und BDOS überschreiben. Zum Neuladen von CCP und BDOS sind folgende Verfahren üblich:

- Laden aus Systemspuren der BOOT-Diskette (meist A:)
- Laden aus einer Kopie im Speicher (z.B. bei Vorhandensein von Schattenspeicher)
- Erstellen einer Kopie während des BOOT-Vorgangs in eine RAM-Disk und Laden von dieser (üblicherweise auch hier aus Systemspuren)
- Laden aus einer @OS-Datei von der Diskette. Dazu muss aber ein Mini-BDOS verfügbar sein, was das logische Lesen von Diskette unterstützt. Die Records dieser @OS-Datei könnten ja beliebig auf der Diskette verstreut liegen.

Nach dem Neuladen von CCP und BDOS werden die Systemsprünge für JP 0000 und CALL 5 eingerichtet, das aktuelle Laufwerk wieder selektiert und die Steuerung ans CCP übergeben.

```
;
BOOT:       ;simplest case is to just perform parameter initialization
        XOR    A        ;zero in the accum
        LD    (IOBYTE),A    ;clear the iobyte
        LD    (CDISK),A    ;select disk zero
        JP    GOCPM        ;initialize and go to cp/m
;
WBOOT:      ;simplest case is to read the disk until all sectors loaded
        LD    SP,80H        ;use space below buffer for stack
        ;...
        ..reread ccp+bdos into memory
        ;...
;
;   end of load operation, set parameters and go to cp/m
GOCPM:
        LD    A,0C3H        ;c3 is a jmp instruction
        LD    (0),A        ;for jmp to wboot
        LD    HL,WBOOTE    ;wboot entry point
        LD    (1),HL        ;set address field for jmp at 0
;
```

```
        LD    (5),A        ;for jmp to bdos
        LD    HL,BDOS        ;bdos entry point
        LD    (6),HL        ;address field of jump at 5 to bdos
;
        LD    BC,80H        ;default dma address is 80h
        CALL    SETDMA
;
        EI            ;enable the interrupt system
        LD    A,(CDISK)    ;get current disk number
        LD    C,A        ;send to the ccp
        JP    CCP        ;go to cp/m for further processing
```

# Zeichen-I/O

Im BIOS sind die grundlegenden I/O-Funktionen zur zeichenweisen Ein- und Ausgabe für Konsole, Drucker, Reader, Punch enthalten.

Ein einfaches BIOS braucht nur Ein- und Ausgabe für Konsole, die restl. Funktionen sind Leerfunktionen.

Das **I/O-Byte** wird von CP/M selbst nicht genutzt. Die BDOS-Funktionen Nr. 7 und 8 lesen bzw. beschreiben direkt die Speicherzelle IOBYTE. Das Systemprogramm STAT nutzt wiederum nur diese BDOS-Funktionen.

Ein I/O-Byte-Unterstützung muss deshalb vollständig im BIOS erfolgen. Man kann problemlos darauf verzichten; wenn man nicht verschiedene Geräte für die 4 I/O- Kanäle Konsole, Drucker, Reader, Punch unterstützen muss/will.

Die BIOS-Funktion **CONIN** darf das eingetippte Zeichen nicht auf dem Bildschirm ausgeben.

```
;
;
;   simple i/o handlers (must be filled in by user)
;   in each case, the entry point is provided, with space reserved
;   to insert your own code
;
CONST:      ;console status, return 0ffh if character ready, 00h if not
        ..status subroutine
        LD    A,00H
        RET
;
CONIN:      ;console character into register a
        ..input routine
        AND    7FH    ;strip parity bit
        RET
;
CONOUT:     ;console character output from register c
        LD    A,C    ;get to accumulator
        ..output routine
        RET
```

```
;
LIST:       ;list character from register c
        LD    A,C    ;character to register a
        RET         ;null subroutine
;
LISTST:     ;return list status (0 if not ready, 1 if ready)
        XOR    A    ;0 is always ok to return
        RET
;
PUNCH:      ;punch character from register c
        LD    A,C    ;character to register a
        RET    ;null subroutine
;
;
READER:     ;read character into register a from reader device
        LD    A,1AH    ;enter end of file for now (replace later)
        AND    7FH    ;remember to strip parity bit
        RET
```

# Diskettenfunktionen

Der größte Teil des BIOS besteht in Funktionen zur Diskettenarbeit.

Eine Diskette besteht physikalisch aus **Spuren (Tracks)**. Diese sind in **physische Sektoren** unterteilt. Das BDOS greift immer über Track- und logische Sektornummer auf Laufwerke zu. Ein **logischer Sektor (sector, auch record)** ist immer 128 Byte lang.

Eine Spur kann Werte von 0..FFFFh annehmen, ein logischer Sektor von 0..FFFFh. Üblich sind bei Disketten aber Werte von 0..80 für die Spur und 0..255 für den logischen Sektor. Viele BIOSe arbeiten deshalb auch nur mit 8-Bit-Registern für diese Werte. Damit sind immerhin 256 Tracks * 256 logische Sektoren * 128 Byte = 8 MByte adressierbar.

Es ist Aufgabe des BIOS, die beiden Werte (Tracks und logischer Sektor) in

- Diskettenseite
- Spur
- und physische Sektornummer

umzusetzen. Wie das erfolgt, ist dem BIOS-Schreiber überlassen (und hängt von der Hardware ab). Beispiele folgen später.

Die physischen Sektoren sind beispielsweise 1 KByte groß. Das BIOS muss einen Puffer bereitstellen, um einen kompletten physischen Sektor einzulesen und diesen in logische Sektoren aufzuteilen. Dieser Mechanismuss heiß Blocking/Deblocking. Das dies nicht trivial ist, gibt es zu CP/M 2.2 eine Datei deblock.asm als Vorlage.

Die Routinen SETDMA, SETTRK, SETSEC speichern einfach die übergebenen Werte. Wenn die Laufwerke nur mit max 256 Spuren zu max. 256 log. Sektoren arbeiten, kann man TRACK und SECTOR als Byte belassen und nur Register C übernehmen. Das BDOS übergibt nur gültige Werte, so dass Register B in diesem Fall immer 0 ist und nicht beachtet werden muss.

**HOME** selektiert Spur 0. Ein phys. Zugriff aufs Laufwerk ist nicht erforderlich. Mam spart auch CALL SETTRK und RET, wenn HOME direkt vor SETTRK steht. Diese Funktion war bei älteren Laufwerken zur exakten Positionierung des Schreib/Lesekopfes gedacht. Da das BDOS vor jedem Diskzugriff die Spurnummer über SETTRK anwählt, ist HOME bei neueren Laufwerken überflüssig.

**SETTRK** bezieht sich auf die im Registerpaar BC übergebene Spur. Diese Spurnummer errechnet sich immer aus der BDOS-internen (logischen) Spurnummer plus dem OFF- Wert im DPB. Wie auch beim SELDSK-Aufruf ist ein tatsächlicher Diskzugriff nicht garantiert.

**SETSEC** bezieht sich auf den im Registerpaar BC übergebenen Sektor. Die so gesetzte Sektornummer ist immer das Ergebnis der SECTRAN-Funktion (s.u.). Auch hier ist ein tatsächlicher Diskzugriff auf diesen Sektor nicht garantiert.

**SETDMA**: Alle nachfolgenden Diskzugriffe müssen die DMA-Adresse als Quell- (bei Schreibzugriffen) bzw. Zieladresse (bei Lesezugriffen) benutzen. Die DMA-Adresse zeigt immer auf einen 128-Byte großen Buffer, weshalb Diskzugriffe immer in Recordgröße erfolgen.

```
;
;   i/o drivers for the disk
;
HOME:       ;move to the track 00 position of current drive
        LD    BC,0        ;select track 0
        ;CALL    SETTRK
        ;RET
;
SETTRK      ;set track given by register bc
        LD    (TRACK),BC
        RET
;
SETSEC      ;set sector given by register bc
        LD    (SECTOR),BC
        RET

SETDMA      ;set dma address given by registers b and c
        LD    (DMAAD),BC
        RET

;
```

**SECTRAN** übernahm ursprünglich eine Sektornummertransformation bei hardsektorierten Disketten (8„) anhand einer Sektorverschränkungstabelle (X- lation table XLT). Dies ist bei moderneren Laufwerken nicht mehr üblich bzw. wegen größerer physischer Sektorlänge als 128 Byte auch nicht möglich. Eine Sektorverschränkung physischer Sektoren erfolgt deshalb meist im phys. Diskettentreiber (s. unten).

Im allgemeinen genügt es, einfach die im Registerpaar BC übergebene Sektornummer ins Registerpaar HL zu kopieren.

im CP/A wird einfach der übergebene Wert genommen und um 1 erhöht (die Sektoren zählen in CP/A bzw. auf Diskette ab 1). Nutzt man eine allgemeine SECTRAN- Routine für alle Laufwerke, muss dies bei den phys. Laufwerkstreibern f. Read und Write beachtet werden.

```
SECTRAN: (allg)
;translate the sector given by BC using the
;translate table given by DE
        EX    DE,HL         ;HL=.trans
        ADD    HL,BC         ;HL=.trans(sector)
        LD    L,(HL)         ;L = trans(sector)
        LD    H,0         ;HL= trans(sector)
        RET    ;with value in HL
;
SECTRAN: (CP/A)
;translate the sector given by BC without translate table
        LD    L,C         ;L = trans(sector)
        LD    H,B         ;HL= trans(sector)
        inc    HL
        RET            ;with value in HL
;
```

**SELDSK**: Das BIOS muß die im C-Register übergebene Laufwerksnummer überprüfen und, falls ein Laufwerk mit dieser Nummer existiert, in HL die Adresse des zugehörigen disk parameter header DPH zurückgeben, Im CP/M ist nicht garantiert, daß nach einem SELDSK-Aufruf auch tatsächlich auf dieses Laufwerk zugegriffen wird. Vielmehr hat der SELDSK-Aufruf nur eine 'Anmeldefunktion', damit sich das BDOS auf das Laufwerk einstellen kann. Das BIOS muß die Laufwerksnummer aber intern speichern, da sich nachfolgende Diskzugriffe immer auf das zuletzt selektierte Laufwerk beziehen.

In einem aufwendigen BIOS kann bei SELDSK eine Analyse der Diskette erfolgen, um das konkrete **Diskettenformat automatisch zu ermitteln**. Im CP/A-BIOS gibt es eine Liste von Formaten, die hier getestet werden (z.B. 624k, 780k, 800k). Als Resultat dieser Analyse wird ein passender DPB ausgewählt (oder dynamisch zusammengestellt) und im DPH eingetragen.

Die Laufwerke müssen nicht in alphabetischer Reihenfolge und durchlaufend angelegt sein. Man kann die Laufwerksbuchstaben A..P willkürlich den Laufwerken zuordnen. SELDSK muss für ein existierenes Laufwerk den passenden DPH und andernfalls 0000 zurückgeben.

Für jeden Laufwerksbuchstabe, der vom BIOS angesprochen werden kann, muss es einen eigenen disk parameter header DPH geben.

Ein disk parameter header DPH ist 16 Byte lang und muss im RAM stehen. BDOS beschreibt die freien Felder des DPH mit eigenen Werten.

Ein DPH umfasst 8 Einträge zu je 16 Bit und hat folgende Struktur: <ditaa>

```
  +-------------------------------------------------+
  | XLT | NHDE | CLTK | FSCT | DIRBUF | DPB | CSV | ALV |
  +-------------------------------------------------+
```

Byte 0/1 2/3 4/5 6/7 8/9 A/B C/D E/F </ditaa>

XLT (s.o.), NHDE, CLTK, FSCT sind mit 0 vorbelegt, DIRBUF ist ein 128 Byte großer Puffer (für alle Laufwerke derselbe), DPB ist die Adresse des Disk Parameter Blocks, CSV die Adresse des Prüfsummenvektors (Check Sum Vector) und ALV die Adresse des Belegungsvektors (Allocation Vector).

Der DPB enthält die Laufwerkseigenschaften und wird weiter unten beschrieben. Ein DPB kann für mehrere DPH genommen werden, wenn die Laufwerkseigenschaften gleich sind (z.B. für 2 gleiche Diskettenlaufwerke).

CSV und ALV sind Speicherbereiche im RAM, deren Größe von den Laufwerkseigenschaften abhängen (s. DPB).

```
;   fixed data tables for four drives

DPBASE:
;   disk parameter header for disk 00
DPH0:       DW    0000H,0000H
        DW    0000H,0000H
        DW    DIRBF,DPB0
        DW    CHK00,ALL00
;   disk parameter header for disk 01
DPH1:       DW    0000H,0000H
        DW    0000H,0000H
        DW    DIRBF,DPB1
        DW    CHK01,ALL01
;   disk parameter header for disk 02
DPH2:       DW    0000H,0000H
        DW    0000H,0000H
        DW    DIRBF,DPB2
        DW    CHK02,ALL02
;   disk parameter header for disk 03
DPH3:       DW    0000H,0000H
        DW    0000H,0000H
        DW    DIRBF,DPB3
        DW    CHK03,ALL03
;

; werden die Laufwerksbuchstaben durchgehend vergeben (A:..D:) und
; folgen die DPH direkt aufeinander, kann der DPH durch DPBASE + 16*DISKNO
berechnet werden
SELDSK:     ;select disk given by register C
        LD    HL,0000H    ;error return code
        LD    A,C
        LD    (DISKNO),A
        CP    4    ;must be between 0 and 3
        RET    NC    ;no carry if 4,5,...
;   compute proper disk parameter header address
        LD    A,(DISKNO)
        LD    L,A    ;L=disk number 0,1,2,3
        LD    H,0    ;high order zero
        ADD    HL,HL    ;*2
        ADD    HL,HL    ;*4
        ADD    HL,HL    ;*8
        ADD    HL,HL    ;*16 (size of each header)
        LD    DE,DPBASE
        ADD    HL,DE    ;HL=.dpbase(diskno*16)
```

```
        RET

; Alternativ: sind die Laufwerksbuchstaben nicht durchgehend vergeben, kann
die Ermittlung auch direkt
; erfolgen, hier Laufwerke A:, B:, F:, P:
SELDSK:     ;select disk given by register C
        LD    A,C
        LD    (DISKNO),A
        LD    HL, DPH0
        CP    'A'-'A'        ; Laufwerk A
        RET    Z
        LD    HL, DPH1
        CP    'B'-'A'        ; Laufwerk B
        RET    Z
        LD    HL, DPH2
        CP    'F'-'A'        ; Laufwerk F
        RET    Z
        LD    HL, DPH3
        CP    'P'-'A'        ; Laufwerk P
        RET    Z
        ;
        LD    HL,0000H    ;error return code
        RET
```

Die **READ**-Funktion liest einen (logischen) Sektor von der Diskette in den DMA-Buffer. Die Disknummer, Spurnummer und Sektornummer sind jeweils durch die letzten SELDSK-, SETTRK- und SETSEC-Aufrufe festgelegt.

Bei physikalischen Sektorlängen vom mehr als 128 Bytes muß das BIOS einen Sektorbuffer entsprechender Große selbst bereitstellen und aus diesem Buffer 128 Bytes zum zuletzt definierten DMA-Buffer kopieren. Falls ein Lesefehler auftritt, sollte das BIOS den Diskzugriff ein paar Mal wiederholen und, falls der Fehler bestehen bleibt, den Fehlercode 1 im A-Register zurückgeben.

Die **WRITE**-Funktion schreibt einen (logischen) Sektor vom DMA-Buffer auf die Diskette. Die Disknummer, Spurnummer und Sektornummer sind jeweils durch die letzten SELDSK-, SETTRK- und SETSEC-Aufrufe festgelegt.

Bei physikalischen Sektorlängen von mehr als 128 Bytes kann das Record-Flag zur Realisierung eines 'Blocking'-Algorithmus verwendet werden. Bei einem normalen Schreibzugriff reicht es, den logischen Sektor nur in den BIOS-internen Sektorbuffer zu übernehmen. Dies hat den Vorteil, daß nachfolgende Schreibzugriffe auf den selben physikalischen Sektor keinen Diskettenzugriff verlangen. Erst wenn der neue logische Sektor in einem anderen physikalsichen Sektor liegt, muß der Sektorbuffer auf die Diskette geschrieben werden. Directory-Schreibzugriffe sollten immer direkt auf die Diskette geleitet werden.

Je nach Laufwerkstyp (Diskette, RAM-Floppy etc.) können Read und Write völlig unterschiedlich implementiert sein. Die BIOS-Routinen READ und WRITE müssen in diesem Fall je nach Laufwerk DISKNO auf spezielle Routinen READx und WRITEx verzweigen.

Man spricht von physischen Laufwerkstreibern für Read und Write, wenn diese den physischen Transfer eines (physischen) Sektors von/zum Laufwerk übernehmen. Die logischen Laufwerkstreiber übernehmen das Blocking/Deblocking und andere Aufgaben zur Laufwerksverwaltung wie

Optimierung der Zugriffe auf verschiedene Laufwerke etc. In einem einfachen BIOS z.B. für eine RAM-Disk mit 128 Byte großen Sektoren braucht man diese Unterteilung nicht.

Ganz einfache Routinen:

```
;
READ:       CALL    calcadr
        ..read log. Sektor nach (DMAAD)
        ld    a,0        ; keine Fehler
        ret
;
WRITE:      CALL    calcadr
        ..schreibe log. Sektor von (DMAAD)
        ld    a,0        ; keine Fehler
        ret
;
calcadr:    ..aus log. Track TRACK und log. Sektor SECTOR
          die physikalische Position berechnen
        ret
```

Es verbleiben die RAM-Speicherbereiche, die nicht vorbelegt sind und deshalb am Ende des BIOS stehen sollten, damit das BIOS in den Systemspuren nicht zu groß wird.

```
;   the remainder of the CBIOS is reserved uninitialized
;   data area, and does not need to be a part of the
;   system memory image (the space must be available,
;   however, between "begdat" and "enddat").
;
TRACK:      DS    2    ;two bytes for expansion
SECTOR:     DS    2    ;two bytes for expansion
DMAAD:      DS    2    ;direct memory address
DISKNO:     DS    1    ;disk number 0-15
;
;   scratch ram area for BDOS use
DIRBF:      DS    128    ;scratch directory area
ALL00:      DS    xx    ;allocation vector 0         benötigte Größe siehe
DPB
ALL01:      DS    xx    ;allocation vector 1
ALL02:      DS    xx    ;allocation vector 2
ALL03:      DS    xx    ;allocation vector 3
CHK00:      DS    xx    ;check vector 0              benötigte Größe siehe
DPB
CHK01:      DS    xx    ;check vector 1
CHK02:      DS    xx    ;check vector 2
CHK03:      DS    xx    ;check vector 3

BUFFER:     DS    xxK    ;Puffer für physischen Sektor, bei einfachen BIOS
pro Laufwerk!
;
    END
```

# Der Disk Parameter Block

Ein DPB umfasst 15 Bytes in folgender Aufteilung: <ditaa> +————————————————————-+

| SPT | BSH | BLM | EXM | DSM | DRM | AL0 | AL1 | CKS | OFF |
|---|---|---|---|---|---|---|---|---|---|

+————————————————————-+

```
0/1     2     3     4   5/6   7/8    9     A   B/C   D/E
 16     8     8     8    16    16    8     8    16    16
```

</ditaa>

Beschreibung s.a. der_disk_parameter_block

Zur Erstellung eines DPB braucht man folgende Angaben:

1. Gesamtkapazität der Diskette in KByte
2. physischer Aufbau der Diskette
    1. Anzahl der Spuren (Tracks)
    2. Anzahl physischer Sektoren pro Spur
    3. Anzahl der beschreibbaren Diskettenseiten
3. max. Anzahl der Direktory-Einträge
4. Anzahl der Systemspuren
5. gewünschte Blockgröße

CP/M kennt keine Diskettenseiten, man muss im physischen Diskettentreiber die Seite aus TRACK oder SECTOR ermitteln. Möglich ist z.B. bei einer Diskette mit 2 Seiten, 80 Spuren, 5 phys. Sektoren pro Spur:

- gerader Track Vorderseite (0,2,4,..), ungerader Track Rückseite (1,3,5,..)
- Track 0..79 Vorderseite, Track 80..159 Rückseite
- Sektor 1..5 Vorderseite, 6..10 Rückseite
- ...

CP/A nutzt die erste Variante.

Unabhängig von der physischen Sektorlänge gibt es im CP/M noch die **Blockgröße**. Das BDOS teilt jede Diskette in Blöcke (engl. Blocks) auf, um damit den Verwaltungs- und Speicheraufwand für die Belegungstabelle zu verkleinern. Die Länge eines Blocks ist 1, 2, 4, 8 oder 16 kByte.

Die Disketten-Belegungstabelle wird in Blöcken geführt, somit kann das BDOS Diskettenplatz auch nur blockweise vergeben. Nachteil dieser Aufteilung ist, das ein File immer ganze Blöcke belegt, auch wenn die tatsächliche Filelänge kleiner ist.

Beispiel 1: eine 800K Diskette mit 2 Seiten, 80 Spuren, 5 phys. Sektoren pro Spur

2 Seiten * 80 Spuren → 160 TRACKs
800 KByte / 160 → 5 KByte / Spur
5 phys. Sektoren pro Spur → 1 KByte großer phys. Sektor
SPT = 5 KByte / Spur → 5K/128 = 40 logische Sektoren (records) pro Spur

Man muss sich für eine Blockgröße entscheiden. Nimmt man 2 KByte große Blöcke, gibt es insgesamt 800k/2k = 400 Blöcke. Damit braucht man 16 bit große Blocknummern. Nimmt man 4 KByte große Blöcke, gibt es insgesamt 800k/4k = 200 Blöcke. Damit braucht man nur 8 bit große Blocknummern. Aber man bekommt weniger kleine Dateien auf der Diskette unter.

Allgemein sollte die Blockgröße mit der Diskettengröße wachsen.

Aus Blockgröße und Blocknummeranzahl ergeben sich die Felder BSH, BLM und EXM des disk parameter blocks:

**OFS** := Anzahl der Systemspuren (logische TRACKs!)
**SPT** := Anz.Phys.Sektoren*Größe.Phys.Sektor / 128

block size BLS in Byte (1024, 2048, .. ,16384)

**DSM** := Gesamtkapazität - 1 = Anz.Blöcke * (block size) - 1

**BSH** := log2 (block size / 128)
**BLM** := (block size / 128) - 1
**EXM** := (block size / 1024) - 1 bei 8 bit-Blocknummern ( DSM/block size ⇐ 255)
**EXM** := (block size / 2048) - 1 bei 16 bit-Blocknummern ( DSM/block size > 255)

Die Anzahl der Directory-Einträge ist frei wählbar. Es werden immer ganze Blöcke vergeben, diese gilt auch für das Directory. Maximal können 16 Blöcke genutzt werden. Ein Directory-Eintrag ist 32 Byte lang, damit sind BLS/32 Directory-Einträge pro Block möglich. Die Maximal-Zahl ergibt sich zu

**DRM** := Anz.Dir.Blöcke * (block size / 32) - 1

Das Verhältnis zwischen Anzahl der Directory-Einträge und Anzahl der Blöcke sollte gewahrt bleiben. Es ist wenig sinnvoll, mehr Directory-Einträge als Blöcke zu haben. DSM/(durchschnittliche Dateigröße) kann ein Anhaltspunkt sein.

Im Beispiel ergibt sich für drei 2K-Blöcke 3 * 2048/32 = maximal mögliche 192 Einträge, das sollte man auch nutzen und DRM auf 191 setzen.

CP/M erkennt Diskettenwechsel, indem ein Prüfvektor über die Directory-Records gebildet wird.

**CKS** := (DRM + 1) / (128 / 32) = (DRM + 1) / 4

Diese Maximalgröße muss nicht immer genommen werden; gerade bei großen Disketten oder Festplatten würde eine Prüfung zu lange dauern. Bei nicht wechselbaren Laufwerken wie Festplatten oder RAM-Disketten kann CKS auch auf 0 gesetzt werden. Damit wird nicht auf Diskettenwechsel geprüft.

Im DPH (s.o.) wird für den Prüfsummenvektor Speicherplatz definiert (CHKxx). Dieser muss CKS Byte groß sein:

CHK00: DS xx ;check vector 0 = CKS Byte

Der Allocation Vektor (ALV) bildet die Belegungstabelle (besser: Belegungsvektor) der Diskette. Für jeden Block der Diskette ist im ALV ein Bit vorhanden, das entsprechend auf 0 (Block frei) oder 1 (Block belegt) gesetzt wird. Die Zuordnung der Blöcke zu den Bits geschieht in absteigender Bitnummernfolge (höchstes Bit eines Bytes zuerst) und aufsteigender Bytefolge (erstes Byte des ALV

zuerst). Für ALLxx muss man deshalb (DSM+1)/8 Byte groß sein:

ALL00: DS xx ;allocation vector 0 = (DSM+1)/8 Byte

Achtung: Bei automatischer Formaterkennung müssen die Speicherplätze CHKxx und ALLxx für die größtmöglichen Werte ausgelegt sein!

Im disk parameter block sind die ersten beiden Byte des Allocation Vektors einzutragen. Je ein Bit ist für einen genutzten Directory-Block zu setzen:

AL0,AL1 = 11..100..00b

Im Beispiel gilt für 3 Directory-Blöcke

AL0 = 1110000b = E0h, AL1 = 00000000b = 00h

insgesamt ergibt sich für Beispiel 1

```
DPB00:      DW    40    ;SPT sectors per track
        DB    4     ;BSF block shift factor
        DB    15     ;BLM block mask
        DB    0     ;EXM null mask
        DW    224     ;DSM disk size-1
        DW    191     ;DRM directory max
        DB    0E0h     ;alloc 0
        DB    00h     ;alloc 1
        DW    48     ;check size
        DW    0     ;track offset

CHK00:      DS    48    ;check vector 0
ALL00:      DS    50    ;allocation vector 0
```

From:
https://hc-ddr.hucki.net/wiki/ - **Homecomputer DDR**

Permanent link:
**https://hc-ddr.hucki.net/wiki/doku.php/cpm/write_a_bios?rev=1331134022**

Last update: **2012/03/07 15:27**