Rolf-Dieter Klein

Winchester-Anschluß

Für den NDR-Klein-Computer, Teil 3

Im zweiten Teil wurden die BIOS-Routinen und Unterprogramme für den Z80 besprochen. In diesem Teil werden die Programmteile für den 68008 vorgestellt. Dabei werden auch die Besonderheiten des BIOS beim CP/M-68k besprochen.

Das Grundprogramm des 68008 enthält keine Unterprogramme für den Betrieb der Winchester. Also müssen ein Formatierprogramm und das BIOS neu hinzukommen. Beides soll an dieser Stelle behandelt werden, insbesondere um Unterschiede zum Z80-BIOS zu verdeutlichen.

Das Formatierprogramm

Jede Winchester muß vor Gebrauch formatiert werden. Das Formatierprogramm arbeitet ohne CP/M, nur mit dem Grundprogramm. Es enthält neben den Unterprogrammen zur Formatierung auch ein einfaches Schreib- und Leseprogramm. Damit kann man die Winchester zum Beispiel vortesten.

Unterschiedliche Winchester-Laufwerke benötigen unterschiedliche Formatierungen. Wir verwenden hier das Laufwerk BASF6188 oder das RO352, die zueinander soweit kompatibel sind. Für andere Laufwerke muß man die Anpaßtabelle ändern. *Bild 1* zeigt das Listing. Die Tabelle steht auf Adresse DRVTBL. Die Bedeutung der Tabelle wurde schon in Teil 2 erklärt und muß daher nicht wiederholt werden.

Nun zu den Besonderheiten beim 680xx-System. Zum einen kann man beim

```
 1: ********************************
 2: * kleines Formatier-Programm   *
 3: * fuer Winchester-Laufwerke    *
 4: * mit XEBEC Controler          *
 5: * Mit ASSRDK oder Grundprog.   *
 6: * uebersetzen und bei START    *
 7: * starten.                     *
 8: * Rolf-Dieter Klein  860118    *
 9: * Version 1.0                  *
10: ********************************
11:
12:  org $10000  * hier freien Platz einsetzen.
13:
14: cpu equ 4     * 1=68008 2=68000/10 4=68020(32bit)
15:
16: start: bra main     * Hauptprogramm starten
17:
18: * IO - Definitionen
19: hdbase equ $fffffcc
20: hddata equ (hdbase)*4
21: hdstat equ (hdbase+1)*4
22: hdrst equ (hdbase+1)*4
23: hdsel equ (hdbase+2)*4
24:
25: * Bitmasken
26: hdreq equ 1   * Request Bit Position
27: hdbsy equ 2   * Busy Bit
28: hdmsg equ 4   * MSG Bit
29: hdcd equ 8    * C/D Bit
30: hdio equ $10  * I/O Bit
31:
32: errmsk equ 2 * Fehlermaske
33:
34: * Rueckmeldewerte
35: comfhst equ 8    * Command From Host
36: datfhst equ 0    * Data From Host
37: datthst equ $10  * Data To Host
38: errthst equ $18  * Error Status to Host
39: comrdy equ $1c   * Command Completed
40:
41: * Befehle
42: drvrdyc equ 0    * Test for Ready
43: formatc equ 4    * Format Code
44: readc equ 8      * Read Code
45: writec equ $a    * Write Code
46: sensec equ 3     * Status Sense
47: initlc equ $c    * Init Disk Size
48: seekc equ $b     * Seek Command
49: recalc equ 1     * Recalibrate
50: ramdiag equ $e0 * Ram Diagnosic
51: ctldiag equ $e4 * Controler Diagnostic
52: drvdiag equ $e3 * Drive Diagnostic
53:
54:
55: * Laufwerk BASF 6188 , RO352
56: * Wegen Kompatibilitaet zum
57: * RO352 Laufwerk wird die Kapazitaet
58: * des BASF-Laufwerks 6188 nicht voll genutzt
59: *
60: * sekanz =  cylinder * koepfe * 32
61: * bei 256 Bytes pro Sektor
62:
63: sekanz equ 39168
64: *
65: drvtbl:
66:  dc.b 1    * msb cylinders
67:  dc.b 50   * lsb cylinders 306 = 50 +256
68:  dc.b 4    * heads = 4
69:  dc.b 0    * msb reduced write cyl. 128
70:  dc.b 128  * lsb reduced write cyl. 128
71:  dc.b 0    * msb write precomp. cyl.
72:  dc.b 128  * lsb write precomp. cyl.
73:  dc.b 11   * ecc length.
74: *
75: wiini:     * Initialisieren mit Drive-Tabelle
76:  move.b #0,hdrst    * Achtung, nicht CLR verwenden.
77:  move #initlc,d1     * Parameter Tabelle
78:  bsr taskout        * Ausfuehren dann.
79:  move #8-1,d5       * Und uebertragen
80:  lea drvtbl(pc),a0
81: deflop:
82:  bsr reqwait
83:  move.b (a0)+,hddata
84:  dbra d5,deflop
85:  bsr getstat        * Status laden
86:  move #recalc,d1    * Und Kopf zurueckfahren
87:  bsr taskout
88:  bsr getstat        * Status laden
89:  rts
90: *
91: hdwrite:            * Sektor schreiben
92:  move #writec,d1    * D2.L=Sektoradresse
93:  bsr taskout
94:  move #256-1,d5     * Byte-Anzahl
95:  lea hstbuf,a0      * Speicherbereich.
96:  bsr reqwait
97:  cmp.b #datfhst,d0
98:  bne hderror
99: hdwlp:              * nur bei XEBEC so schnell.
100:  move.b (a0)+,hddata  * nicht schneller als 1.7 ys
101:  dbra d5,hdwlp
102:  bsr getstat
103:  tst d0
104:  bne hderror
105:  rts
106:
107:
108: hdread:            * Sektor lesen
109:  move #readc,d1    * D2.L=Sektoradresse
110:  bsr taskout
111:  move #256-1,d5     * Byte Anzahl
112:  lea hstbuf,a0      * Speicherbereich
```

Bild 1. Das Formatierprogramm für 68000-CPUs

```
113:  bsr reqwait
114:  cmp.b #datthst,d0
115:  bne hderror
116: hdrlp:
117:  move.b hddata,(a0)+ * nicht schneller als 1.7 ys
118:  dbra d5,hdrlp
119:  bsr getstat
120:  tst d0
121:  bne hderror
122:  rts
123:
124: *
125: hderror:
126:  move #1,d0
127:  rts
128:
129: taskout:            * d1=Befehl. d2.l = Sektornr.
130:  lea task,a0        * Zieladresse
131:  move.b d1,(a0)+    * Befehl ausgeben
132:  swap d2
133:  move.b d2,(a0)+    * MSB Sektor ausgeben
134:  swap d2
135:  ror #8,d2
136:  move.b d2,(a0)+    * Middle Sektor ausgeben
137:  rol #8,d2
138:  move.b d2,(a0)+    * LSB Sektor ausgeben
139:  move.b #1,(a0)+    * Max einen Sektor
140:  move.b #0,(a0)+    * 3ms Steprate
141:  lea task,a0        * Dann ausgeben
142: taskhl:             * hier bei beliebigem Block
143:  move #6-1,d5
144:  bsr selcntlr       * Select Controller
145: tskolp:
146:  bsr reqwait        * Warten auf Request
147:  cmp.b #comfhst,d0  * Status pruefen
148:  bne hderror        * Fehler da
149:  move.b (a0)+,hddata  * sonst Daten ausgeben
150:  dbra d5,tskolp     * bis alle Daten ausgegeben
151:  clr d0             * kein Fehler
152:  rts
153:
154: reqwait:
155:  move.b hdstat,d0
156:  btst  #0,d0
157:  beq reqwait        * Warten bis bereit
158:  and #$1c,d0        * Status Teil c/d i/o
159:  rts
160:
161: getstat:
162:  bsr reqwait
163:  cmp.b #errthst,d0
164:  bne hderror
165:  move.b hddata,d1   * Ergebnis
166:  bsr reqwait
167:  cmp.b #comrdy,d0
168:  bne hderror
169:  move.b hddata,d0   * Dummy
170:  move.b d1,d0       * d0=Ergebnis
171:  and #errmsk,d0     *
172:  rts
173:
174: selcntlr:
175:  move.b #1,hddata   * Selekt 1
176:  move.b #1,hdsel    * Controler
177: sel1:
178:  move.b hdstat,d0
179:  and #hdbsy,d0
180:  beq sel1           * Bis Ready
181:  rts
182:
183: ********************************************
184: * Hauptprogramm   Formatierer mit Prueflesen *
185: ********************************************
186:
187: main:
188:  jsr @clrscreen
189:  lea txt1(pc),a0
190:  bsr print
191:  jsr @ci
192:  cmp.b #'j',d0
193:  beq ok
194:  cmp.b #'J',d0
195:  beq ok
196:  bra final
197:
198: ok:     * jetzt wird formatiert.
199:  bsr wiini    * Recalibrate
200:  move #ctldiag,d1
201:  bsr taskout * Controler testen
202:  bsr getstat
203:  bne main     * Fehler, dann zurueck
204:  lea txt2(pc),a0
205:  bsr print
206:  move #ramdiag,d1
207:  bsr taskout * RAM testen
208:  bsr getstat
209:  bne main
210:  lea txt3(pc),a0
211:  bsr print
212:
213:  lea taskform,a0
214:  bsr taskhl
215:  bsr getstat
216:  beq ok2
217: err:
218:  lea txt4(pc),a0
219:  bsr print
220:  bra final
221:
222: ok2:
223:  lea txt5(pc),a0
224:  bsr print
225:  move #drvdiag,d0
226:  bsr taskout
227:  bsr getstat
228:  bne err
229:  lea txt6(pc),a0
230:  bsr print
231:  move.l #0,d2  * Startsektor
232:  move #0,d3    * Testcounter
233: loop:
234:  bsr hdread    * Lesen nach hstbuf
235:  bne err       * Fehler
236:  addq.l #1,d2  * Neuer Sektor
237:  add #1,d3     * Counter
238:  and #$ff,d3
239:  bne skip1
240:  move.b #'.',d0
241:  jsr @co2
242: skip1:
243:  cmp.l #sekanz,d2
244:  bne loop
245:  lea hstbuf,a0   * E5 ablegen
246:  move #256-1,d5
247: wlp1:
248:  move.b #$e5,(a0)+
249:  dbra d5,wlp1
250: *
251:  lea txt7(pc),a0
252:  bsr print
253:  move.l #0,d2
254:  move #256-1,d5   * Sektorenzahl 64K Byte
255: wlp2:
256:  movem.l d2/d5,-(a7)
257:  bsr hdwrite      * Schreiben
258:  movem.l (a7)+,d2/d5
259:  bne err
260:  addq.l #1,d2
261:  dbra d5,wlp2     * ablegen
262: *
263:  lea txt8(pc),a0
264:  bsr print
265: final:
266:  jsr @ci
267:  rts
268: *
269:
270: print:    * A0->Text
271:  move.b (a0)+,d0
272:  beq finprt
273:  jsr @co2
274:  bra.s print
275: finprt:
276:  rts
277:
278: txt1:
279:  dc.b 'Winchester-Formatier-Programm 68k 1.0',$d,$a
280:  dc.b 'Achtung, die gesamte Platte wird geloescht',$d,$a
281:  dc.b 'Start = j ',0
282:
283: txt2:
284:  dc.b $d,$a,'Controler ok.',$d,$a,0
285:
286: txt3:
287:  dc.b 'RAM ok, Formatieren beginnt',$d,$a,0
288:
289: txt4:
290:  dc.b $d,$a,'++++ Formatier Fehler ++++',$d,$a,0
291:
292: txt5:
293:  dc.b 'Sektor-0-Test',$d,$a,0
294:
295: txt6:
296:  dc.b 'Nun folgt Prueflesen jedes Sektors',$d,$a,0
297:
298: txt7:
299:  dc.b $d,$a,'Direktory wird geloescht - Schreibtest',$d,$a,0
300:
301: txt8:
302:  dc.b 'Ok. Laufwerk formatiert. Ende',$d,$a,0
303:
304: * Interleave=5, aber nur bei
305: * Multisektor-Transfer wirksam.
306: taskform: dc.b formatc,0,0,0,5,0
307:
308: task: ds.b 6
309: hstbuf: ds.b 256
310: end
311:
312:
```

NDR-Klein-Computer verschiedene CPUs verwenden, nämlich den 68008, den 68000 und demnächst den 68020. Alle diese CPUs verwenden einen unterschiedlich breiten Datenbus. Daher ist die I/O-Adressierung unterschiedlich. Um das Programm kompatibel zu halten, gibt es am Anfang eine Definition, die hier CPU EQU 4 lautet. Wenn man den 68008 verwendet, wie die Mehrzahl der Leser, so muß man hier CPU EQU 1 schreiben, beim 68000 schreibt man CPU EQU 2 und beim 68020 CPU EQU 4. Alle I/O-Adressen werden nämlich mit diesem Wert multipliziert. Die FESTCON-Baugruppe liegt beim 68008 auf Adresse $FFFFFFCC und folgende. Dies entspricht auch der DIL-Schalter-Einstellung beim Z80.

Die Unterprogramme des Formatierungsprogrammes

WIINI hat die Aufgabe die Diskparameter in DRVTBL an den XEBEC-Controller zu übertragen. Ferner wird der Schreib-Lese-Kopf auf Spur Null positioniert.

Das Unterprogramm HDWRITE schreibt einen Sektor auf die Winchester. Dazu wird im Register D2.L die Adresse des Sektors angegeben. Diese Adresse kann maximal 32 Bit lang sein. Es wird dabei der logische Sektor angegeben, der bei Adresse 0 beginnt und bis zu MAXIMALKAPAZITÄT / 256 – 1 reicht. Jeder Sektor ist bei uns 256 Byte lang. Bei erfolgreichem Aufruf der Routine wird im Register D0 der Wert 0 übergeben, im Fehlerfall der Wert 1.

Die Routine HDREAD liest einen Sektor. Auch hier werden 256 Byte übertragen. Das Register D0 liefert wieder den Fehlercode.

In den beiden Unterprogrammen wird eine Besonderheit des Xebec-Controllers ausgenützt. Er kann bei der Datenübertragung mit maximal 1,7 Mikrosekunden Abstand zwischen den Daten arbeiten. Man braucht dann keine Warteroutine aufrufen, die auf das REQUEST-Signal wartet, sondern kann die Daten gleich direkt übertragen. Aber Achtung, beim 68020 darf der Cache-Speicher nicht eingeschaltet sein, da die Übertragung dann weniger Zeit als 1,7 Mikrosekunden pro Byte benötigt. Das ergibt Fehler.

Das Unterprogramm TASKOUT transportiert einen 6-Byte-Befehl an den Controller. Im Register D0 steht anschließend bei erfolgreicher Übertragung der Wert 0. Der Befehlscode wird im Register D1 angegeben und eine etwaige Sektoradresse im Register D2.L. Es wird eine Schrittrate von 3 ms eingestellt (siehe Kommentar in Listing).

Wenn man will, kann man beim BASF-6188-Laufwerk auch den Buffered-Step-Modus verwenden und den Wert 8 als letztes Byte eintragen, dann wird mit 12 Mikrosekunden pro Step gearbeitet und das Winchester-Laufwerk positioniert dann so schnell es kann. Das Unterprogramm TASKOUT besitzt noch einen zweiten Einsprung, TASKHL, bei dem man im Register A0 die Adresse eines Taskblocks übergeben kann.

Das Unterprogramm REQWAIT wartet auf das Vorhandensein eines Request-Signals, wie es für die normale Datenübertragung vom Controller geliefert wird.

Mit GETSTAT wird der Rückmeldestatus eingelesen, so wie er vom Xebec-Controller geliefert wird. Der Fehlercode steht anschließend in Register D0. Er muß normalerweise 0 sein. Mit SELCNTLR wird der Xebec-Controller 1 angesprochen. Im Prinzip ist es nämlich möglich, auf dem SASI-Bus mehrere Controller anzuschließen.

Das Hauptprogramm beginnt bei der Marke MAIN. Es löscht zunächst den Bildschirm und fragt dann an, ob man es wirklich ernst meint mit dem Formatieren, denn es werden dabei alle etwa vorhandenen Daten vernichtet.

Bei der Marke OK beginnt der Formatiervorgang. Der Controller wird durch Aufruf von WIINI zunächst initialisiert und der Kopf auf Spur 0 gefahren. Wenn sich das Programm hier schon aufhängt, sollte man die Verbindung zum Xebec-Controller, und die Baugruppe FESTCON genau kontrollieren. Anschließend wird noch die Controller-Diagnose aufgerufen, erst dann wird der Text „Controller ok." ausgegeben. Wenn dieser Text erscheint, ist das meiste bereits getestet. Danach folgt noch eine RAM-Diagnose auf dem Xebec-Controller und dann das Formatieren, sobald der Text „RAM ok, Formatieren beginnt" ausgegeben wird.

Die LED am Winchesterlaufwerk muß nun grün aufleuchten und wenn man die Antriebsscheibe beim Schrittmotor betrachten kann, so wird man sehen, daß sie sich ganz langsam dreht. Das Formatieren braucht einige Minuten. Danach wird eine Laufwerkdiagnose durchgeführt, „Sektor-0-Test" genannt, die aber das Laufwerk nur grob testet. Danach wird jeder Sektor gelesen. Dies ist der eigentliche Test, ob alles fehlerfrei formatiert wurde. Das Prüflesen dauert einige Minuten, viel länger als das Formatieren, denn nun müssen 10 MByte zum Computer übertragen werden. Pro Byte werden im Schnitt 64 Mikrosekunden benötigt. Alle 256 Sektoren wird auf dem Bildschirm ein Punkt ausgegeben, damit man sieht, ob das Programm noch arbeitet. Im Fehlerfall wird die Meldung „Formatier-Fehler" ausgegeben. Das Programm ist in dieser Hinsicht noch nicht sehr komfortabel, da es dann abbricht. Beim Xebec-Controller ist es aber möglich, im Fehlerfall, also wenn z. B. eine Spur defekt ist, eine Ersatzspur zuzuweisen. Der Controller verwaltet diese Ersatzspuren dann automatisch.

Wenn die Formatierung in Ordnung war, so wird der Text „Directory wird gelöscht – Schreibtest" ausgegeben. Nun werden 256 Sektoren mit dem Datenwert E5 belegt. Damit ist die Platte auch gleich für den Einsatz unter CP/M vorbereitet.

Die Unterprogramme aus Bild 1 kann man natürlich nun auch ohne CP/M einsetzen, um z. B. das Mikro-DOS (siehe Sonderheft 2, NDR-Computer) auf die Winchester umzustellen. Jedoch ist ein komfortableres Betriebssystem auf Dauer später sehr viel bequemer.

Das BIOS-Listing

Bild 2 zeigt dazu das BIOS-Listing für CP/M-68k. Die Winchester-Unterprogramme wurden eingebunden, aber auch einiges verbessert. So erfolgt zum Beispiel die Bestimmung des Speicherplatzes und der Länge für die RAM-Floppy jetzt automatisch. Die verwendete CPU wird automatisch erkannt, so daß das BIOS unverändert auf allen Prozessoren 680xx verwendet werden kann.
Das BIOS für das Betriebssystem CP/M-68k ist im Prinzip genauso aufgebaut, wie beim Z80, jedoch mit einigen bedeutungsvollen Unterschieden. So ist z. B. nicht mehr nötig, das BIOS zu patchen (was soviel wie „im Objekt-Code rumfummeln" heißt), sondern man bindet es mit einer Bibliothek zusammen, ähnlich wie bei CP/M-Plus.

Das CP/M wird außerdem von der Diskette geladen, auf der es sich als normale Datei befindet. Die Systemspuren werden von einem Lade-CP/M belegt, das

das eigentliche CP/M von Diskette holt. Für das Lade-CP/M ist ebenfalls ein BIOS nötig, das im Aufbau schon dem großen BIOS entspricht, jedoch nur die notwendigsten Unterprogramme enthält.

Beim Betrieb mit der Winchester ist bei uns Laufwerk A das konventionelle Floppy-Laufwerk, entweder 8 Zoll oder 5¼ Zoll. Laufwerk H ist die Winchester. Der Boot-Prozeß geschieht vom Floppy-Laufwerk. Daher ist es nicht notwendig, das Lade-CP/M zu ändern. Beim CP/M-68k wird das System nur nach dem Kaltstart gebootet, so daß durch den Start vom langsameren Floppy-Laufwerk beim normalen Arbeiten kein Nachteil entsteht. Wer will, kann aber im Prinzip auch vom Winchester-Laufwerk booten, man muß jedoch dazu das Lade-CP/M neu binden und auf dem Winchester-Laufwerk unterbringen.

Das BIOS ist „gelinkt"

Das BIOS beginnt mit der Definition dreier globaler Marken: „_init", „_ccp" und „cpm". Diese Marken werden verwendet, um beim Linken die notwendigen Verbindungen zum CP/M zu schaffen. Die Marke „cpm" liegt dabei am Anfang des CP/M-Systems.

„_init" ist der Einsprung ins BIOS, den das CP/M nach dem Kaltstart nutzt und „_ccp" ist der Einsprung ins CP/M, den das BIOS nach dem Warmstart verwendet.

„_init" hat die Aufgabe alle hardwarespezifischen Teile des Systems zu initialisieren. Ferner wird der TRAP-Vektor für das BIOS installiert. Das CP/M und insbesondere Benutzerprogramme greifen nämlich ausschließlich über einen Trap-Befehl (TRAP #3) auf das BIOS zu. Damit werden Anwenderprogramme unabhängig von der genauen Lage des BIOS. Bei den Hardware-Initialisierungen wird zunächst die Startadresse des Grundprogramms gesucht. Denn im BIOS werden Grundprogrammaufrufe verwendet, um zum Beispiel auf die Floppy zuzugreifen. Die Suche nach dem Grundprogramm geschieht mit Hilfe eines Datenmusters, das den Anfang des Grundprogramms kennzeichnet, nämlich $55AA8001. Zusätzlich werden dann noch zwei Sprungbefehle kontrolliert, um sicher zu gehen auch wirklich das Grundprogramm vor sich zu haben. Achtung, wenn ein Anwenderprogramm das gleiche Muster verwendet, so ist ein erneuter Kaltstart nicht möglich. Dies ist aber keine große Einschränkung, denn beim ersten Kaltstart befindet sich das Muster nicht im Speicher. Dynamische Speicher und auch die meisten neuen statischen Speicher haben nach dem Einschalten ein Zufallsmuster, das jedoch paradoxerweise recht regelmäßig aussieht. Die Wahrscheinlichkeit unser Muster zu treffen ist also kleiner, als die Auftrittswahrscheinlichkeit des Musters bei gleicher Wahrscheinlichkeit aller möglichen verschiedener Muster.

Nach erfolgreichem Ende der Suche werden die Anfangsadresse des Grundprogrammes und die der Einsprungstelle in den Variablen GRU68K und GRUJ68K festgehalten. Eine andere Größe, die wichtig ist, nämlich die verwendete CPU, wird ebenfalls gespeichert, und zwar in der Variablen „CPU". Sie enthält den Wert 1, wenn die 68008-CPU oder allgemein ein 8-Bit-Datenbus verwendet wird. Der Wert 2 liegt beim 68000 und 68010 an, also bei einem 16-Bit-Datenbus und der Wert 4 beim 68020, wenn er einen 32-Bit-Datenbus verwendet. Diese Information wird im Winchester-Teil verwendet, um die aktuellen I/O-Adressen zu berechnen. Damit ist es wie ge-

```
1: ********************************************
2: * BIOS 68k 1.2 fuer  NDR-Klein-Computer     *
3: * Rolf-Dieter Klein  (C) 1986, Muenchen     *
4: * // 8 " Laufwerke // 5 1/4 80 Spur //      *
5: * RAM-Floppy                                *
6: * Winchester Laufwerk    860120    2.1 rdk  *
7: * CSTS und Ramfloppy verbessert             *
8: * Version fuer AS68      850323    2.0      *
9: * CPM.SYS  erzeugen                         *
10: * dazu  AS68 wi68bios.s                     *
11: * LO68 -r -ucpm -o cpm.rel cpmlib wi68bios.o*
12: * oder beim 68010 oder 68020               *
13: * LO68 -r -um68010 -ucpm -o cpm.rel cpmlib  *
14: *      wi68bios.o                           *
15: * SIZE68 cpm.rel      Groesse bestimmen     *
16: * RELOC -B19000 cpm.rel cpm.sys    bei 128K *
17: * RELOC -B39000 cpm.rel cpm.sys    bei 256K *
18: * dann auf Systemdiskette copieren.         *
19: * bei AS68 und LO68 ggf. Laufwerke angeben  *
20: * ebenfalls mit -s x: die Init-Datei, siehe *
21: * CP/M Manual                               *
22: ********************************************
23:
24: * CPU-Bestimmung erfolgt automatisch
25: * In der Variablen CPU steht dann
26: * 1=68008 2=68000/10 4=68020(32bit)
27:
28:  .globl _init
29:  .globl _ccp
30:  .globl cpm     * Startadresse CP/M (erste Adresse)
31:
32: * Achtung, Aendern wenn size68 cpm.rel einen Wert
33: * groesser $6fff ergibt.
34: ramdispl equ $7000   * Offset zu CPM fuer Freispeicher
35:
36:
37: *                  Damit ist das Bios,
38: *                  und damit CPM.REL verschiebbar.
39: * man muss nur bei RELOC eine neue Adresse angeben
40: * bei 128K  -B19000   , Kontrolle durch SIZE68 moeglich.
41: * bei 256K  -B39000
42:
43: * CP/M Start, bei reloc
44: * angeben, und auch hier ggf.
45: * aendern, sonst reagiert das CP/M nicht
46: * denn es holt sich die Information aus
47: * der Memory region table.
48: * Kontrolle des Wertes durch size68 durchfuehren
49: * wenn das Bios geaendert wird.
50:
51: * 8 Zoll Version  // wird automatisch eingestellt
52: *                // abhaengig von der Diskette
53: * Laufwerk   A,B,C,D       8 Zoll je 243K A,C = Vorderseite
54: *            E,F           5 1/4    je 800K
55: *            G             Ramfloppy
56: *            H             Winchester
57:
58: * 5 1/4 Zoll Version // wird automatisch eingestellt
59: *                   // abhaengig von der Diskette
60: * Laufwerk   A,B           5 1/4 " je 800K
61: *            C,D,E,F       8 " je 243K C,E=Vorderseite
62: *            G             Ramfloppy
63: *            H             Winchester
64:
65: * 8 "    koennen zweiseitig sein,
66: *        aber einfache Dichte, 128 Bytes / Sektor
67: *        mit 26 Sektoren pro Spur und 77 Spuren
68: *
69: * 5 1/4" (auch 3 1/2" oder 3")
70: *        muessen zweiseitig sein, 80 Spuren haben
```

Bild 2. Das BIOS für CP/M-68k

sagt möglich, ein und dasselbe BIOS für alle vorgesehenen CPUs zu verwenden.

Alles automatisch

Nach dieser Aktion wird das verwendete Floppy-Laufwerk bestimmt. Dazu wird ein Unterprogramm im Grundprogramm aufgerufen, das den Namen GETFLOP besitzt. Dieses Unterprogramm liest ein Laufwerk an und bestimmt automatisch den Dichtecode. Im BIOS gibt es zwei Fälle, die bearbeitet werden. Entweder man verwendet ein 8-Zoll-Laufwerk mit einfacher Dichte (manche haben noch so etwas) oder ein 5¼-Zoll-(3½-Zoll-)Laufwerk mit 80 Spuren, doppelter Dichte und zwei Seiten. Die Information, welches Laufwerk zum Booten verwendet wurde, bestimmt nachher die Laufwerkaufteilung. So sind Laufwerk A, B, C, D

```
71: *        und 1024 Bytes pro Sektor mit 5 Sektoren pro Spur
72: *        ( doppelte Dichte)
73: *
74: *
75: * Die Ramfloppy wird dynamisch auf
76: * den groessten Speicherbereich nach
77: * dem BIOS gelegt. Dabei werden
78: * nur die ersten zwei zusammenhaengenden
79: * Speicherbereiche beachtet.
80: * Die RAMFLOPPY wird automatisch geloescht
81: * wenn vorher keine Zugriffe erfolgt sind.
82: * Eine Pruefwert wird gesetzt um dies zu erkennen.
83: * Ein BAD SEKTOR tritt auf, wenn ein Fehler
84: * auf der RAM-Karte festgestellt wurde. Dabei
85: * wird aber nur grob geprueft.
86:
87: * Die Ramfloppyadresse wird automatisch
88: * auf den nach dem BIOS (bss) folgenden groessten RAM-
89: * Teil gelegt. RAMFLOA kontrollieren, muss
90: * die letzte Adresse im System sein, danach darf RAM
91: * folgen. RAMFLOPPY ist maximal 1 Mbyte gross.
92: * auch wenn mehr RAM verfuegbar ist. Wer das
93: * erweitern will, muss ALV6 vergroessern und
94: * Abfrage im Init-Teil aendern.
95:
96: * Das I/O Byte wird ausgewertet
97: * 7     6     5     4     3     2     1     0
98: * -list-      -auxo-      -auxi-      -cons--
99: *
100: *
101: * cons:  *00=key/gdp 01=si/so 10=(auxi,list) 11= key/gdp
102: * auxi:   00=key     *01=si    10=ri          11= key
103: * auxo:   00=gdp     *01=so    10=po          11= gdp
104: * list:   00=gdp      01=so   *10=lo          11= gdp
105: *    * = Default nach Start
106: * andere Anordnungen durch STAT programmieren
107: * siehe CP/M68 Manual.
108:
109:
110: ************************************************
111:
112: _init:
113: *                zuerst Grundprogramm suchen
114: *                aber erst hinter der TPA
115:  lea $20400,a0 * Start der Suche 128K RAM davor annehmen
116: loop:
117:  cmp.l #$5aa58001,(a0)  * Kennung im Grundprogramm
118:  beq.s gefunden
119:  adda.l #$1000,a0       * Alle 4K Suchen nach dem Anfang
120:  bra.s loop
121: gefunden:
122:  cmp #$6000,$20(a0)
123:  bne.s loop             * dort muss Sprung stehen
124:  cmp #$6000,$24(a0)     * dort muss Sprung stehen
125:  bne.s loop             * ok Test scheint ok
126: *                         a0 -> Grundprog + $400
127:  suba.l #$400,a0        * BASIS-Adresse
128:  move.l a0,gru68k       * dort merken
129:  move.l $414(a0),cpu    * CPU-Size merken fuer rev 2.1
130:  adda.l #$420,a0        * Adresse fuer Trap-Sprung
131:  move.l a0,gruj68k      * auch merken
132: * Bildschirm und Size sind im Boot vorbereitet
133: *                       Laufwerks-Dichte bestimmen
134:  move #1,d4             * Laufwerk A
135:  move #76,d7            * GETFLOP
136:  bsr trapexe
137:  move.b d4,drvcode      * $11=Mini DD, $21=Maxi SD
138: *
139:  clr.b mwrtflg          * alles voreinstellen
140:  move.b #$ff,lastsso    * damit wird Steprate zuerst eingest
141:  clr.b mdrvakt         * 0=ungueltiges Laufwerk in Buffer
142:  clr cursor            * Cursor ist jetzt aus
143:  move.b #0,stepmaxi    * PATCH PATCH PATCH maxi-Steprate
144:  move.b #0,stepmini    * PATCH PATCH PATCH mini-Steprate
145: *                       Rest vorbereiten
146:  tst.w serinitd         * Nur EINMAL initialisieren
147:  bne.s init1ser         * Damit WBOOT CTRL-C CTRL-C ok.
148:  move.w #$ff,serinitda  * nur einmal ausfuehren.
149:  move.b #$1e,d0         * 9600 Baud, 8 Bit, 1 Stopbit
150:  move.b #$0b,d1         * keine Paritaet, RTS high, T,R Freigabe.
151:  move #108,d7           * SIINIT
152:  bsr trapexe           * SER-Baugruppe fuer AUX (RDR,PUN).
153: init1ser:
154: * rev 2.1 dynamische RAMFLOPPY
155:  lea cpm,a0            * Suche bei CPM
156:  adda.l #ramdispl,a0    * + Offset (nach dem Speicherteil) starten.
157:  move #59,d7           * GETRAM erster Bereich
158:  bsr trapexe           * a0->naechste nichtram-zelle,a1->ram
159:  bcs ini2wei           * Es folgt kein Ram mehr (Sonderfall)
160:  move.l gru68k,d1      * Nicht groesser als Grundprog. aktzept.
161:  cmpa.l d1,a0          * Next-Adresse muss kleiner gleich sein.
162:  ble ini10w           * Nur moeglich, wenn Grundp. im RAM liegt.
163:  movea.l d1,a0         * Grundp. Anfang nehmen.
164: ini10w:                * Danach Kontrolle ob Anfang ok.
165:  cmp.l a1,d1           * Wenn Grundprogramm-RAM, auch nicht.
166:  ble ini2wei           * keinen  Bereich nehmen
167:  lea cpm,a2            * Wert nochmals berechnen
168:  adda.l #ramdispl,a2    * Start der Suche
169:  cmpa.l a1,a2          * Wenn kein RAM hinter BIOS
170:  ble ini1bwei          * dann naechsten Bereich nehmen.
171:  movea.l a2,a1         * Wenn Ergebnis kleiner, dann
172: ini1bwei:              * Min-Wert nehmen.
173:  move.l a1,ramadr      * Dort beginnt RAM nach dem BIOS.
174:  move.l a0,d0          * Auch fuer Laengenberechnung.
175:  sub.l a1,d0           * Laenge merken 1...n
176:  movem.l d0,-(a7)       * A0 ist neuer Ausgangspunkt.
177:  move #59,d7           * Zweiter Bereich.
178:  bsr trapexe           * a0->naechste nichtram-zelle,a1->ram
179:  movem.l (a7)+,d0      * Vergleichswert vom ersten Bereich
180:  bcs ini2wei           * Kein RAM mehr da
181:  move.l gru68k,d1      * Nicht groesser als Grundprog. aktzept.
182:  cmpa.l d1,a0          * Next-Adresse muss kleiner gleich sein.
183:  ble ini11w           * Nur moeglich, wenn Grundp. im RAM liegt.
184:  movea.l d1,a0         * Grundp. Anfang nehmen.
185: ini11w:                * Danach Kontrolle ob Anfang ok.
186:  cmp.l a1,d1
187:  ble ini1wei           * sonst ersten Bereich nehmen
188:  move.l a0,d1          * nun Vergleichen
189:  sub.l a1,d1           * Laenge zweiter Bereich in d1.
190:  cmp.l d1,d0          * Wenn erster Bereich groesser, dann
191:  bge ini1wei          * diesen bevorzugen.
192:  move.l d1,d0          * sonst zweiten
193:  move.l a1,ramadr     * als Basisadresse und Laenge.
194: ini1wei:              * d0 enthaelt Laenge in Bytes
195:  sub.l #4,d0          * 4 Bytes fuer Erkennungsmuster abziehen.
196:  asr.l #8,d0          * / 256  Blockzahl berechnen
197:  asr.l #3,d0          * / 8    gesamt /2048
198:  cmp.w #10,d0        * min 20K
199:  ble ini2wei
200:  sub.w #1,d0          * n-1 eintragen.
201:  move.w d0,ramkap     * Blockcount eintragen in Biostab.
202:  cmp.w #511,d0       * ACHTUNG hier Begrenzung max. Kap.
203:  blt.s ini3wei         * ok wenn < 1 Myte
204:  move.w #511,ramkap   * Sonst 1 MByte nehmen.
205:  bra.s ini3wei
206: ini2wei:
207:  move.l #0,ramadr      * Error-Flag
208:  move.w #1,ramkap      * RAMFLOPPY zu klein.
209:  bra.s ini5wei
210: ini3wei:
211: *
212:  movea.l ramadr,a0
213:  add.l #4,ramadr       * reserviere die erste vier Bytes
214:  cmp.b #$55,(a0)
215:  bne.s ini4wei
216:  cmp.b #$aa,1(a0)
217:  bne.s ini4wei
218:  cmp.b #$12,2(a0)
219:  bne.s ini4wei
220:  cmp.b #$34,3(a0)
221:  beq.s ini5wei
222: ini4wei:    * muss geloescht werden.
```

8-Zoll-Laufwerke, wenn von dort gebootet wurde und E und F sind dann die 5¼-Zoll-Laufwerke; oder A und B sind die 5¼-Zoll-Laufwerke und C, D, E und F sind die Maxilaufwerke.
Die Steprate wird bei Maxilaufwerken auf 3 ms, bei Minilaufwerken auf 6 ms gestellt. Dann wird die serielle Schnittstelle auf 9600 Baud eingestellt. Sie kann z. B. für Drucker- oder Terminalbetrieb verwendet werden. Die Schnittstelle wird nur einmal initialisiert, auch wenn die „_init"-Routine vom CP/M ein zweites Mal aufgerufen werden sollte (geschieht ggf. bei zweimaligem CTRL-C CTRL-C).

Als nächstes wird die RAM-Floppy initialisiert. Dabei wird, beginnend bei einem Offset von RAMDISPL, bezogen auf den Start des CP/Ms nach freiem RAM

```
223:    move.b #$55,(a0)+
224:    move.b #$aa,(a0)+
225:    move.b #$12,(a0)+
226:    move.b #$34,(a0)+    * 256 DirektoryEintraege * 32 Byte
227:    init4b:
228:    move.b #$e5,(a0)+ * Direktory loeschen.
229:    dbra d3,init4b
230:    *
231:    ini5wei:    * ist schon Initialisiert oder zu klein.
232:    *
233:    move.w #$10010100,iobyte * LO, SI/SO, KEY/GDP einstellen.
234:    move.l #traphndl,$8c     * trap #3 handler
235:    clr.l d0                 * Laufwerk A, User 0 ist Start
236:    rts
237:
238:    traphndl:
239:    cmpi #nfuncs,d0
240:    bcc trapng
241:    lsl #2,d0
242:    ext.l d0
243:    add.l #biosbase,d0
244:    movea.l d0,a0
245:    movea.l (a0),a0
246:    jsr (a0)
247:    trapng:
248:    rte
249:
250:    biosbase:
251:    dc.l _init      * 0
252:    dc.l _wboot     * 1
253:    dc.l constat    * 2
254:    dc.l conin      * 3
255:    dc.l conout     * 4
256:    dc.l lstout     * 5
257:    dc.l pun        * 6
258:    dc.l rdr        * 7
259:    dc.l home       * 8
260:    dc.l seldsk     * 9
261:    dc.l settrk     * 10
262:    dc.l setsec     * 11
263:    dc.l setdma     * 12
264:    dc.l read       * 13
265:    dc.l write      * 14
266:    dc.l listst     * 15
267:    dc.l sectran    * 16
268:    dc.l setdma     * 17
269:    dc.l getseg     * 18
270:    dc.l getiob     * 19
271:    dc.l setiob     * 20
272:    dc.l flush      * 21
273:    dc.l setexc     * 22
274:    dc.l getbasis   * 23    neuer Bios-Eingang. a4 belegen mit BASIS
275:
276:    nfuncs equ (*-biosbase)/4
277:
278:    getbasis:       * BIOS-Eingang, nur fuer Grundprg. Operationen
279:    move.l gru68k,a4 * aus Hilfszelle holen und in A4 als Ergebnis.
280:    rts
281:
282:    wboot:
283:    bsr puttrk      * falls noch aussteht
284:    clr.b mdrvakt   * und Laufwerk ungueltig danach
285:    bsr putwi       * ggf. abspeichern, jedoch ohne Kontrolle.
286:    jmp _ccp
287:
288:    constat:
289:    move iobyte,d0
290:    and #3,d0
291:    beq csts        * 00 = key
292:    cmp #1,d0
293:    beq sists       * 01 = si
294:    cmp #2,d0
295:    beq allfalse    * 10 = batch, auxi, kein Status
296:    bra csts        *
297:
298:
299:    allfalse:       * kein Status bei BATCH
300:    clr d0          * nicht abbrechbar
301:    rts
302:
303:
304:
305:    alltrue:
306:    move #$00ff,d0
307:    rts
308:
309:    conin:
310:    move iobyte,d0
311:    and #3,d0
312:    beq ci          * 00 = key
313:    cmp #1,d0
314:    beq si          * 01 = si
315:    cmp #2,d0
316:    beq rdr         * 10 = batch, auxi
317:    bra ci          *
318:
319:
320:    conout:
321:    move iobyte,d0
322:    and #3,d0
323:    beq co          * 00 = gdp
324:    cmp #1,d0
325:    beq so          * 01 = so
326:    cmp #2,d0
327:    beq lstout      * 10 = batch, auxi
328:    bra co          *
329:
330:
331:    rdr:
332:    move iobyte,d0
333:    and #%00001100,d0
334:    beq ci          * 00 = key
335:    cmp #%00000100,d0
336:    beq si          * 01 = si
337:    cmp #%00001000,d0
338:    beq ri          * 10 = ri
339:    bra ci          *
340:
341:                    * Ausgabe nur dann, wenn DSR auf
342:    pun:    move iobyte,d0
343:    and #%00110000,d0
344:    beq co          * 00 = gdp
345:    cmp #%00010000,d0
346:    beq so          * 01 = so
347:    cmp #%00100000,d0
348:    beq po          * 10 = po
349:    bra co          *
350:
351:
352:
353:    lstout:
354:    move iobyte,d0
355:    and #%11000000,d0
356:    beq co          * 00 = co
357:    cmp #%01000000,d0
358:    beq so          * 01 = so
359:    cmp #%10000000,d0
360:    beq lo          * 10 = lo
361:    bra co
362:
363:
364:    listst:
365:    move iobyte,d0
366:    and #%11000000,d0
367:    beq alltrue     * 00 = co, immer bereit
368:    cmp #%01000000,d0
369:    beq sosts       * 01 = so
370:    cmp #%10000000,d0
371:    beq losts       * 10 = lo
372:    bra alltrue     * Ausgabe immer bereit
373:
374:
```

gesucht. Mindestens 20 KByte müssen vorhanden sein. Es werden zwei nachfolgende Bereiche gesucht, der größere von beiden wird verwendet, aber nur dann, wenn er unterhalb des Grundprogramms liegt, um den Speicher des Grundprogramms nicht zu zerstören. Die maximale Länge der RAM-Floppy ist zur Zeit auf 1 MByte begrenzt, kann aber durch Ändern des reservierten Speicherplatzes für ALV6 vergrößert werden. Die Länge zwischen 20 KByte und 1 MByte wird automatisch berechnet und in die BIOS-Tabellen eingetragen, so daß man mit STAT DSK: später nach der aktuellen Größe fragen kann. Wenn die RAM-Floppy noch nicht verwendet wurde, wird das Directory der RAM-Floppy mit E5 vorgelöscht. Damit man nach einem Kaltstart, z. B. durch RESET den Inhalt der RAM-Floppy dadurch nicht verliert, wird ein Erkennungsmuster verwendet, daß die Löschung beim zweiten Mal unterbindet. Dazu wird das Muster $55AA1234 in die ersten vier Bytes der RAM-Floppy eingetragen. Diese vier Bytes sind reserviert und erst danach beginnt die eigentliche RAMFLOPPY. Die RAMFLOPPY ist das Laufwerk G.

Der Winchester-Teil wird im „_init"-Programm noch nicht angesprochen, sondern erst dann, wenn man Laufwerk H anspricht. Dadurch ist es möglich, das BIOS auch ohne Winchester zu verwenden.

Der TrapHandler

Mit dem Befehle „MOVE.L TRAPHNDL,$8C" wird die Adresse des

```
                        * rev 2.1
csts:
        cmp #100,cursor * Cursor >100
        ble conlstat
        cmp #101,cursor
        bgt con2stat    * nur noch autoflip
        add #1,cursor   * auf 102 schalten.
        move #61,d7     * Dann einschalten
        bsr trapexe     * CURSEIN
        bra.s con2stat
conlstat:
        add #1,cursor   * bei jedem Aufruf erhoehen.
        bra.s con3stat  * aber zunaechst ohne Autoflip.
con2stat:
        move #60,d7
        bsr trapexe     * Autoflip
con3stat:
        move #13,d7
        bsr trapexe     * CSTS
        beq.s noton
        moveq.l #$1,d0
        rts
noton:
        clr.l d0
        rts

ci:
        tst cursor
        beq conlin
        clr cursor
        move #62,d7
        bsr trapexe     * CURSAUS
conlin:
        move #12,d7
        bsr trapexe
        and.l #$7f,d0   * Wort wird ausgewertet
        rts

co:
        cmp #101,cursor
        ble conlout     * war nicht an.
        move #62,d7     * erst wenn >101.
        bsr trapexe     * CURSAUS
conlout:
        clr cursor      * immer ruecksetzen
        move d1,d0      * bei CO-Aufruf.
        move #33,d7     * CO2.
        bsr trapexe
        rts

lo:
        move d1,d0
        move #22,d7     * LO
        bsr trapexe
        rts

losts:
        move #117,d7    * LSTS
        bsr trapexe     * 0=nicht ready, $00ff=ready
        and #$ff,d0     * savety
        rts

so:
        move d1,d0      * HIGH liegt (12V).
        move #105,d7    * SO, Ausgabe seriell
        bsr trapexe     * d0.b bleibt Zeichen
        rts

sosts:
        move #107,d7
        bsr trapexe     * FF = True
        and #$ff,d0
        rts
```

```
si:
        move #104,d7    * SI, Eingabe seriell
        bsr trapexe     * d0.b = Zeichen
        rts

sists:
        move #106,d7    * SISTS, FFFFFFFF=Zeichen da
        bsr trapexe
        and #$ff,d0     * nur FF wenn TRUE
        rts

ri:
        move #14,d7     * RI
        bsr trapexe
        rts

po:
        move d1,d0
        move #15,d7     * PO
        bsr trapexe
        rts

home:
        clr.w track     * rev 2.0 Wortgroesse
        rts

seldsk:                 * je nach Boot-Quelle
        moveq #0,d0
        cmp.b #maxdsk,d1 * max disk
        bpl selrtn
        move.b d1,seldrv
        bsr conwdrv     * nach d0, Langwort 0,1,2,3,4,5,6,7
*                       unabh. von Boot 0..3 = maxi, 4..5=mini
        mulu #dphlen,d0 * v 2.0 alter Fehler behoben.
        add.l #dph0,d0
selrtn:
        rts

settrk:
        move.w d1,track  * rev 2.0 Wort-Groesse
        rts

setsec:
        move.w d1,sector  * 1..256 moegl. 256->0 abgebildet
        rts

sectran:
        tst.l d2
        beq nosect
        movea.l d2,a0
        ext.l d1
        move.b 0(a0,d1),d0
        ext.l d0
        rts
nosect:
        move d1,d0      * 1..256 / 1..40 / besser fuer allg.
        add #1,d0       * Routinen, wie PUTBOOT, da in Manual so.
        rts

setdma:
        move.l d1,dma
        rts

dcode:                  * Codieren Drive + Dense
        bsr conwdrv     * d0= 0,1,2,3  immer Laufwerkscode
        and #$3,d0      * sicherheitshalber
        move.l a0,-(a7)         * unabh. von gewaehltem Format
        lea dctab1,a0
        cmp.b #$21,drvcode      * wenn aber MINI geladen
        bne.s dlcode
        lea dctab2,a0           * dann andere Tabelle
dlcode: move.b 0(a0,d0.l),d4    * Ergebnis in D4
```

Programms „traphndl" in die Trap-Vektor-Tabelle eingetragen. Bei der Ausführung eines TRAP-Befehls gelangt man dann immer zu diesem Programm. Es hat die Aufgabe, die verschiedenen BIOS-Funktionen aufzurufen. Dazu wird im Register D0 der Index der BIOS-Funktion angegeben. In der Tabelle BIOSBASE sind alle Adressen der BIOS-Funktionen aufgeführt.
Die Funktion 23 ist normalerweise im CP/M-68k nicht vorhanden. Sie ist eine Besonderheit, um den Anfang des Grundprogramms zu ermitteln. Man braucht die Routine, wenn man als Anwender das Grundprogramm direkt ansprechen möchte. Dazu muß man die Funktion 23 aufrufen und man erhält die Startadresse des Grundprogramms. Auf der Adresse $420+Basis kann man dann die Grundprogramm-Unterprogramme über einen Sprung erreichen, der wie beim Trap #1 des Grundprogramms im Register D7 die Funktionsnummer des

```
movea.l (a7)+,a0
rts

                * MAXI BOOT erfolgte
dctab1:  dc.b $11,$12,$91,$92
                * MINI BOOT erfolgte
dctab2:  dc.b $14,$18,$94,$98

read:    bsr convdrv    * 0..3 = maxi, 4..5 = min, constant
         cmp.b #4,d0
         bge readmini   * maxi einstellen ggf.
         bsr delsso     * Lesen
         move #1,d1
         clr d2
         clr d3
         move.w sector,d2
         move.w track,d3
         bsr dcode
         movea.l dma,a0
         move #75,d7
         bsr trapexe
         beq.s rdok
         move.w #1,d0
         rts
rdok:    clr.w d0
         rts

                * d1= 0=norm, 1= dirwrt, 2= first
write:   move.b d1,alloc
         bsr convdrv    * 0..3=maxi, 4..5=mini, 6=ramflo 7=Winchester
         cmp.b #4,d0
         bge writemini
         bsr delsso     * Schreiben
         move #2,d1
         clr d2
         clr d3
         move.w sector,d2
         move.w track,d3
         bsr dcode
         movea.l dma,a0
         move #75,d7
         bsr trapexe
         beq.s wrok
         move.w #1,d0
         rts
wrok:    clr.w d0
         rts

readmini:
         cmp.b #6,d0
         beq readram
         cmp.b #7,d0
         beq readwin
         bsr calc
         move.b mdrvakt,d0
         cmp.b d0,d4    * Laufwerk gleich
         bne rload      * nein, dann laden
         move.b mtrkakt,d0
         cmp.b d0,d3
         bne rload
         move.b msekakt,d0
         cmp.b d0,d2
         bne rload
rlrd:    lea buffer,a1  * Quelle, adr. berechnen
         clr.l d1
         move.w sector,d1  * 1..40
         sub.b #1,d1    * 0..39
         and #7,d1      * 0..7
         asl.l #7,d1    * * 128
         adda.l d1,a1   * ist Quelle
         movea.l dma,a0 * Ziel
         move #128-1,d3
rllp1:   move.b (a1)+,(a0)+
         dbra d3,rllp1
         clr d0
         rts
rload:   bsr puttrk     * falls alte Spur da, rueckschreiben
         bcs errflo
         bsr calc
         move.b d4,mdrvakt
         move.b d3,mtrkakt
         move.b d2,msekakt
         bsr gettrk     * und dann transfer
         bcc rlrd       * GLOBAL
errflo:  move #1,d0
         rts

writemini:
         cmp.b #6,d0
         beq writeram
         cmp.b #7,d0
         beq writewin
         bsr calc
         move.b mdrvakt,d0
         cmp.b d0,d4    * Laufwerk gleich
         bne wload      * nein, dann laden
         move.b mtrkakt,d0
         cmp.b d0,d3
         bne wload
         move.b msekakt,d0
         cmp.b d0,d2
         bne wload
wlwr:    lea buffer,a1  * Ziel adr. berechnen
         clr.l d1
         move.w sector,d1 * 1..40
         sub.b #1,d1    * 0..39
         and #7,d1      * 0..7
         asl.l #7,d1    * * 128
         adda.l d1,a1   * ist Ziel
         movea.l dma,a0 * Quelle
         move #128-1,d3
wrllp1:  move.b (a0)+,(a1)+
         dbra d3,wrllp1
         move.b #1,mwrtflg * merken, das Sektor geschrieben
         cmp.b #1,alloc
         bne wr2
         bsr puttrk
         bne errflo
         clr.b mdrvakt  * ungueltiger Buffer
wr2:     clr d0
         rts

wload:   bsr puttrk     * falls alte Spur da, rueckschreiben
         bcs errflo
         bsr calc
         move.b d4,mdrvakt
         move.b d3,mtrkakt
         move.b d2,msekakt
         bsr gettrk     * und dann transfer
         bcs errflo     * und weiter dann
         bra wlwr
```

Unterprogramms erwartet. Leider kann man den Trap #1 nicht verwenden, da er vom CP/M-68k gelöscht wird.

Die BIOS-Unterprogramme

„WBOOT" wird bei einem Warmstart aufgerufen. Hier werden zur Sicherheit alle internen Puffer auf Diskette oder Winchester zurückgeschrieben, falls erforderlich.

Alle I/O-Routinen wie Consol-Eingabe, -Ausgabe usw. sind vom CP/M aus umdefinierbar. Normalerweise werden z. B. als Console die GDP-64-Baugruppe mit KEY verwendet. Man kann aber durch den Befehl „STAT CON: = CRT:" das Terminal auf die serielle Schnittstelle umleiten.
Ebenso beim Reader, Punch und Drucker. Der Drucker ist normalerweise auf die CENT-Baugruppe (Centronics-Schnittstelle) geschaltet.

```
679:
680:
681: * Buffer Verwaltung
682: * fuer 1K Sektoren
683:
684: calc:          * Berechnet Code, LW, SEK, TRK...
685: *               SEK->D2 TRK->D3 LW->D4
686:     bsr convdrv    * 4,5 ist immer code unabh. von Boot
687:     cmp.b #$21,drvcode * pruefen ob Mini-Boot.
688:     beq calmini
689: *
690:     move.b #$24,d4 * LW        transfer    E,F ist nr
691:     sub.b #4,d0    * 0,1 Ergebnis
692:     beq calc1
693:     move.b #$28,d4 * LW
694: calc1:
695:     bra calweiter
696: calmini:
697:     move.b #$21,d4 * dann A,B physikalisches Laufwerk
698:     sub.b #4,d0
699:     beq calweiter
700:     move.b #$22,d4
701: calweiter:
702:     move.w sector,d2 * 1..40
703:     sub.b #1,d2    * 0..39
704:     ror.b #3,d2    * nnn = 0..4
705:     and #$111,d2   * Bereich Sektor
706:     add #1,d2      * 1..5 erlaubt
707:     move.w track,d3
708:     ror.b #1,d3
709:     bcc.s calc2    * Vorderseite
710:     or #$80,d4     * Rueckseite ansprechen// Achtung SSO verwenden.
711: calc2:
712:     and #$7f,d3    * Track gueltig
713:     rts
714: *
715: *                  MINIFLOPPY
716: selsso:          * sso setzen, falls noetig. Und Steprate
717:     cmp.b #$80,lastsso
718:     beq nosels
719:     clr d1        * step sel cmd
720:     move.b stepmini,d3
721:     or #$80,d3
722:     move #75,d7
723:     bsr trapexe
724:     move.b #$80,lastsso
725: nosels:
726:     rts
727:
728:                  MAXIFLOPPY
729: *
730: delsso:         * sso rueckssetzen. Und Steprate
731:     cmp.b #$0,lastsso
732:     beq nosels
733:     clr d1
734:     move.b stepmaxi,d3
735:     move #75,d7
736:     bsr trapexe
737:     clr.b lastsso
738:     rts
739:
740:
741:
742: gettrk:
743:     bsr selsso
744:     clr.b mwrtflg
745:     lea buffer,a0  * Ziel
746:     clr d2
747:     clr d3
748:     move.b msekakt,d2
749:     move.b mtrkakt,d3
750:     move.b mdrvakt,d4  * drive code valid
751:     move #1,d1     * LESEN
752:     move #75,d7
753:     bsr trapexe    * Floppy zugriff
754:     rts            * incl. Fehlercode
755:
756:
757:
758: puttrk:
759:     tst.b mwrtflg
760:     beq noput
761:     bsr selsso
762:     clr.b mwrtflg
763:     lea buffer,a0  * Quelle
764:     clr d2
765:     clr d3
766:     move.b msekakt,d2
767:     move.b mtrkakt,d3
768:     move.b mdrvakt,d4  * drive code valid
769:     move #2,d1     * SCHREIBEN
770:     move #75,d7
771:     bsr trapexe    * Floppy zugriff
772:     rts            * incl. Fehlercode
773:
774: noput:
775:     clr d0
776:     rts            * nicht ablegt, no error
777:
778:
779: readram:           * RAM FLOPPY STARTET dynamisch
780:     movea.l ramadr,a1  * Quelle
781:     move.l a7,d1
782:     beq noram
783:     clr.l d1       * savety
784:     move.w track,d1
785:     asl.w #8,d1    * 0..n
786:     move.w sector,d0   * 1..256(0) ueberlauf reverse
787:     move.b d0,d1   * rev 2.1
788:     sub.b #1,d1    * 0..255
789:     asl.l #7,d1    * * 128 (da Blockgroesse)
790:     adda.l d1,a1   * Adresse fertig
791:     movea.l dma,a0 * Ziel
792:     move #128-1,d3
793: ramget:
794:     move.b (a1)+,(a0)+
795:     dbra d3,ramget     * no errors
796:     clr.w d0
797:     rts
798:
799:
800: writeram:          * RAM FLOPPY  STARTET dynamisch max 8MByte
801:     movea.l ramadr,a1   * Ziel, dynamischer Platz.
802:     movea.l a1,d1
803:     beq noram
804:     clr.l d1       * savety
805:     move.w track,d1
806:     rol.w #8,d1    * 0..n
807:     move.w sector,d0   * 1..256(0) Ueberlauf reverse
808:     move.b d0,d1   * rev 2.1
809:     sub.b #1,d1    * 0..255
810:     asl.l #7,d1    * * 128 (da Blockgroesse) max 8MByte
811:     adda.l d1,a1   * Adresse fertig
812:     movea.l dma,a0 * Quelle
813:     clr.b (a1)     * Speichertest kurz
814:     tst.b (a1)
815:     bne ramerr
816:     move.b #$ff,(a1)
817:     cmp.b #$ff,(a1)
818:     bne ramerr
819:     move #128-1,d3
820: ramput:
821:     move.b (a0)+,(a1)+
822:     dbra d3,ramput     * no errors
823:     clr.w d0
824:     rts
825:
826: ramerr:           * trotzdem copieren
827:     move #128-1,d3
828: ramlput:
829:     move.b (a0)+,(a1)+
830:     dbra d3,ramlput
```

Die genaue Beschreibung ist im Bild 2 als Kommentar abgedruckt.

Eine Besonderheit stellt der Consolstatus bei Verwendung der GDP-64 dar. Es gibt Programme, z. B. das 68000-FORTH, die ausschließlich den Consolstatus aufrufen, bevor sie die Consoleingabe durchführen. Wenn man nun direkt die Unterprogramme vom Grundprogramm verwendet, so blinkt der Cursor dann nicht mehr, denn der Cursor wird nur bei der Consoleingabe eingeschaltet. Um dies zu verhindern ist die Consolstatusroutine hier erweitert. Wenn sie mehr als 100mal hintereinander aufgerufen wurde, ohne daß die Consolausgabe aufgerufen wurde, so wird der Cursor eingeschaltet und der Blinkmodus durch das Programm AUTOFLIP aktiviert. Erst die Consolausgabe desaktiviert das wieder.
Im alten BIOS wurde ohne die Verwendung eines Zählers der Cursor sofort ein-

```
831: noram:           * Fehler kein RAM da
832:  move #1,d0
833:  rts
834:
835: flush:
836:  bsr puttrk      * incl. flags, ggf zurueckschreiben
837:  bne flush1
838:  clr.b mdrvakt   * Buffer ist dann ungueltig
839:  bsr putwi       * rev 2.0 auch Winchester
840:  bne flush1
841:  clr d0          * ok
842:  rts
843: flush1:
844:  move #$ffff,d0
845:  rts
846:
847: getseg:
848:  move.l #memrgn,d0
849:  rts
850:
851: getiob:
852:  move iobyte,d0
853:  rts
854:
855: setiob:
856:  move d1,iobyte
857:  rts
858:
859: setexc:
860:  andi.l #$ff,d1
861:  lsl #2,d1
862:  movea.l d1,a0
863:  move.l (a0),d0
864:  move.l d2,(a0)
865: noset:
866:  rts
867:
868: *
869: convdrv:          * ergibt logisches Laufwerk in D0
870: *                   abhaengig von drvcode
871:  cmp.b #$21,drvcode      * ok Mini
872:  beq.s convldrv          * dann dorthin
873:  cmp.b #$11,drvcode      * Maxi
874: *                          hier Fehlerbehandlung moeglich
875:  clr.l d0                * Langwort gueltig
876:  move.b seldrv,d0        * Laufwerkscode holen 0,1,2,3,4,5,6,7
877:  rts                     * sonst direkt uebernehmen
878: convldrv:               * 0,1,2,3 -> 4,5 und 4,5 -> 0,1
879:  clr.l d0
880:  move.b seldrv,d0
881:  move.l a0,-(a7)
882:  lea convtab,a0
883:  move.b 0(a0,d0.l),d0    * und Wert holen, Langwort gueltig
884:  movea.l (a7)+,a0
885:  rts                     * d0, gueltig
886:
887: convtab:
888:  dc.b 4,5,0,1,2,3,6,7,0 * Konfiguration MINI-Boot.
889:  ds 0
890:
891:
892: trapexe:          * Grundprog ausfuehren ueber Sprung
893:  movem.l a5/a6,-(a7)
894:  move.l gruj68k,a6 * Zieladresse auf Stack
895:  jsr (a6)          * Sprung auf TRAP-Ersatz
896:  movem.l (a7)+,a5/a6
897:  rts
898: *
899: *
900: *
901:
902: ********************************
903: * Winchester Unterprogramme     *
904: * mit XEBEC Controler           *
905: * Rolf-Dieter Klein  860118     *
906: * Version 1.0                   *
907: ********************************
908:
909:
910: * IO - Definitionen
911: hdbase equ $ffffffcc    * Bei CPU=1, wird umgerechnet in wiini
912:
913: * Bitmasken
914: hdreq equ 1  * Request Bit Position
915: hdbsy equ 2  * Busy Bit
916: hdmsg equ 4  * MSG Bit
917: hdcd equ 8   * C/D Bit
918: hdio equ $10 * I/O Bit
919:
920: errmsk equ 2 * Fehlermaske
921:
922: * Rueckmeldewerte
923: comfhst equ 8    * Command From Host
924: datfhst equ 0    * Data From Host
925: datthst equ $10 * Data To Host
926: errthst equ $18 * Error Status to Host
927: comrdy equ $1c  * Command Completed
928:
929: * Befehle
930: drvrdyc equ 0    * Test for Ready
931: formatc equ 4    * Format Code
932: readc equ 8      * Read Code
933: writec equ $a    * Write Code
934: sensec equ 3     * Status Sense
935: initlc equ $c    * Init Disk Size
936: seekc equ $b     * Seek Command
937: recalc equ 1     * Recalibrate
938: ramdiag equ $e0 * Ram Diagnosic
939: ctldiag equ $e4 * Controler Diagnostic
940: drvdiag equ $e3 * Drive Diagnostic
941:
942:
943: * Laufwerk BASF 6188 , RO352
944: * Wegen Kompatibilitaet zum
945: * RO352 Laufwerk wird die Kapazitaet
946: * des BASF-Laufwerks 6188 nicht voll genutzt.
947: *
948: * sekanz =  cylinder * koepfe * 32
949: * bei 256 Bytes pro Sektor
950:
951: sekanz equ 39168
952: *
953: drvtbl:
954:  dc.b 1    * msb cylinders
955:  dc.b 50   * lsb cylinders 306 = 50 +256
956:  dc.b 4    * heads = 4
957:  dc.b 0    * msb reduced write cyl. 128
958:  dc.b 128 * lsb reduced write cyl. 128
959:  dc.b 0    * msb write precomp. cyl.
960:  dc.b 128 * lsb write precomp. cyl.
961:  dc.b 11   * ecc length.
962: *
963: wiini:     * Initialisieren mit Drive-Tabelle
964: * rev 2.1   CPU unabhaengig.
965:  move.l cpu,d1          * 1,2,4
966:  move.l #hdbase,d0      * Basis-IO-Adresse
967:  muls d1,d0             * ergibt hddata
968:  move.l d0,xhddata      *
969:  move.l #hdbase+1,d0
970:  muls d1,d0             * ergibt hdstat und hdrst
971:  move.l d0,xhdstat      * ablegen dort.
972:  move.l d0,xhdrst
973:  move.l #hdbase+2,d0
974:  muls d1,d0             * ergibt hdsel
975:  move.l d0,xhdsel       * ok IO-Ports definiert.
976: *
977:  movea.l xhdrst,a2  * Adresse IO-Port laden
978:  move.b #0,(a2)     * Achtung, nicht CLR verwenden.
979:  move #initlc,d1    * Parameter Tabelle
980:  bsr taskout        * Ausfuehren dann.
981:  move #8-1,d5       * Und uebertragen
982:  lea drvtbl(pc),a0
```

geschaltet. Dies führt aber beim Betrieb des CP/Ms zu einer stark verlangsamten Ausgabe. Durch die neue Methode ist das nicht mehr der Fall.

Beim CP/M-68k können Laufwerke mit mehr als 8 MByte angesprochen werden. Dies wird unter anderem durch Verwendung von in Wortgröße Spur- und Sektorparamtern erreicht. Daher wird in den Unterprogrammen SETTRK und SETSEC auch je ein Wort gespeichert.

Die Lese- und Schreibroutinen

Mit Hilfe des Unterprogramms CONVDRV wird die Umrechnung der logischen Laufwerke A bis H in die physikalischen Laufwerke 0 bis 7 vorgenommen. Denn Sie erinnern sich: Es hängt vom Boot-Laufwerk ab, welches Laufwerk wo liegt.

Die physikalischen Laufwerke 0 bis 3 sind MAXI-Laufwerke. Wenn ein Wert größer als 3 vorkommt, so werden die Routinen READMINI und WRITEMINI aufgerufen, die für die Mini-Laufwerke zuständig sind.

Dort wird dann noch gefragt, ob das Laufwerk 6, also die RAM-Floppy oder Laufwerk 7, also die Winchester angesprochen werden muß.

Bei den Minilaufwerken wird das NDR80-Format verwendet, mit 1024 By-

```
983: deflop:
984:  bsr reqwait
985:  movea.l xhddata,a2 * Ziel
986:  move.b (a0)+,(a2)
987:  dbra d5,deflop
988:  bsr getstat       * Status laden
989:  move #recalc,d1   * Und Kopf zurueckfahren
990:  bsr taskout
991:  bsr getstat       * Status laden
992:  rts
993: *
994: hdwrite:           * Sektor schreiben
995:  move #writec,d1   * D2.L=Sektoradresse
996:  bsr taskout
997:  move #256-1,d5    * Byte-Anzahl
998:  lea hstbuf,a0     * Speicherbereich.
999:  bsr reqwait
1000:  cmp.b #datfhst,d0
1001:  bne hderror
1002: hdwlp:            * nur bei XEBEC so schnell.
1003:  movea.l xhddata,a2 * kleines Delay fuer 68020
1004:  move.b (a0)+,(a2)  * nicht schneller als 1.7 ys
1005:  dbra d5,hdwlp    * Im Cache-Mode ggf. zu schnell.
1006:  bsr getstat
1007:  tst d0
1008:  bne hderror
1009:  rts
1010:
1011:
1012: hdread:           * Sektor lesen
1013:  move #readc,d1   * D2.L=Sektoradresse
1014:  bsr taskout
1015:  move #256-1,d5   * Byte Anzahl
1016:  lea hstbuf,a0    * Speicherbereich
1017:  bsr reqwait
1018:  cmp.b #datfhst,d0
1019:  bne hderror
1020: hdrlp:
1021:  movea.l xhddata,a2 * kleines Delay.
1022:  move.b (a2),(a0)+  * nicht schneller als 1.7 ys
1023:  dbra d5,hdrlp
1024:  bsr getstat
1025:  tst d0
1026:  bne hderror
1027:  rts
1028: *
1029: hderror:
1030:  move #1,d0
1031:  rts
1032:
1033:
1034: taskout:           * d1=Befehl. d2.l = Sektornr.
1035:  lea task,a0       * Zieladresse
1036:  move.b d1,(a0)+   * Befehl ausgeben
1037:  swap d2
1038:  move.b d2,(a0)+   * MSB Sektor ausgeben
1039:  swap d2
1040:  ror #8,d2
1041:  move.b d2,(a0)+   * Middle Sektor ausgeben
1042:  rol #8,d2
1043:  move.b d2,(a0)+   * LSB Sektor ausgeben
1044:  move.b #1,(a0)+   * Max einen Sektor
1045:  move.b #0,(a0)+   * 3ms Steprate
1046:  lea task,a0       * Dann ausgeben
1047: taskhl:            * hier bei beliebigem Block
1048:  move #6-1,d5
1049:  bsr selcntlr      * Select Controller
1050: tskolp:
1051:  bsr reqwait       * Warten auf Request
1052:  cmp.b #comfhst,d0 * Status pruefen
1053:  bne hderror       * Fehler da
1054:  movea.l xhddata,a2   * Ziel IO
1055:  move.b (a0)+,(a2)    * sonst Daten ausgeben
1056:  dbra d5,tskolp     * bis alle Daten ausgegeben
1057:  clr d0            * kein Fehler
1058:  rts

1059:
1060: reqwait:
1061:  movea.l xhdstat,a2
1062:  move.b (a2),d0
1063:  btst #0,d0            * Warten bis bereit
1064:  beq reqwait           * Status Teil c/d i/o
1065:  and #$1c,d0
1066:  rts
1067:
1068: getstat:
1069:  bsr reqwait
1070:  cmp.b #errthst,d0
1071:  bne hderror
1072:  movea.l xhddata,a2
1073:  move.b (a2),d1        * Ergebnis
1074:  bsr reqwait
1075:  cmp.b #comrdy,d0
1076:  bne hderror
1077:  movea.l xhddata,a2
1078:  move.b (a2),d0        * Dummy Lesen.
1079:  move.b d1,d0          * d0=Ergebnis
1080:  and #errmsk,d0       *
1081:  rts
1082:
1083: selcntlr:
1084:  movea.l xhddata,a2
1085:  move.b #1,(a2)        * Selekt 1
1086:  movea.l xhdsel,a2     * Controler
1087:  move.b #1,(a2)
1088: sel1:
1089:  movea.l xhdstat,a2
1090:  move.b (a2),d0
1091:  and #hdbsy,d0         * Bis Ready
1092:  beq sel1
1093:  rts
1094:
1095: readwin:
1096:  bsr setupwi     * d3=log. Sektor, d2=phys.
1097:  cmp.l wisektor,d2  * wenn phys. schon da, nc. merken
1098:  beq readlwi
1099:  bsr getwi       * sonst ggf. alten zurueck, neuen laden.
1100:  bne errwi       * Ende im Fehlerfall
1101: readlwi:         * d3=log. Sektor
1102:  lea hstbuf,a0   * Quelle
1103:  btst #0,d3      * =0, dann even
1104:  beq read2wi
1105:  lea hstbuf+128,a0
1106: read2wi:
1107:  move #128-1,d4
1108:  movea.l dma,a1
1109: read3wi:
1110:  move.b (a0)+,(a1)+
1111:  dbra d4,read3wi
1112:  clr d0
1113:  rts
1114:
1115:
1116: writewin:         * alloc.b = 0,1=dirwrite,2
1117:  bsr setupwi     * d3=log. Sektor, d2=phys.
1118:  cmp.l wisektor,d2  * phys da
1119:  beq writelwi
1120:  bsr getwi       * neuen Sektor laden zuvor
1121:  bne errwi       * Fehler sonst
1122: writelwi:
1123:  lea hstbuf,a0   * Ziel
1124:  btst #0,d3      * =0, dann even
1125:  beq write2wi
1126:  lea hstbuf+128,a0
1127: write2wi:
1128:  move #128-1,d4
1129:  movea.l dma,a1
1130: write3wi:
1131:  move.b (a1)+,(a0)+
1132:  dbra d4,write3wi
1133:  move #$ffff,imflag   * Schreiben noetig
1134: *
```

te pro Sektor. Ein entsprechender Puffer befindet sich im BIOS. Dieser Puffer wird immer dann auf die Diskette zurückgeschrieben, wenn entweder ein neuer Sektor gebraucht wird, und der im Puffer vorhandene schreibend genutzt wurde, oder wenn ein Direktory-Schreibzugriff erfolgt. Die Information, daß ein schreibender Zugriff auf das Direktory erfolgt, wird vom CPM an das BIOS geliefert und von diesem in der Speicherzelle ALLOC festgehalten.

Die Unterprogramme für die RAM-Floppy gestalten sich dagegen recht schlicht. Bei READRAM wird ein Sektor von der RAM-Floppy gelesen und bei WRITE-RAM wird einer geschrieben. Der Vorgang geschieht in einer kleinen Schleife recht rasch. Beim Schreiben eines Wertes wird zusätzlich geprüft, ob überhaupt noch Speicher da sind, um damit eine zusätzliche Sicherheit zu schaffen. Im Fehlerfall meldet das CP/M einen de-

```
1135:  cmp.b #1,alloc * Bei Direktory Write
1136:  beq write4wi
1137:  bsr putwi       * Gleich schreiben.
1138:  bne errwi
1139: write4wi:
1140:  clr d0
1141:  rts
1142:
1143:
1144: putwi:     * Alten Sektor ablegen
1145:  tst imflag
1146:  beq nputwi
1147:  movem.l d2/d3/a0,-(a7)
1148:  move.l wisektor,d2
1149:  bsr hdwrite * und ablegen
1150:  movem.l (a7)+,d2/d3/a0
1151:  bne errwi
1152:  clr imflag  * jetzt geschrieben und damit ok.
1153: nputwi:
1154:  clr d0    * ok, kein Fehler
1155:  rts
1156:
1157: errwi:     * Fehler aufgetreten.
1158:  move #1,d0
1159:  rts
1160:
1161: getwi:    * Neuen Sektor holen d3=log.
1162:  movem.l d2/d3,-(a7) * Neuen Sektor merken.
1163:  bsr putwi  * alten Sektor ggf. ablegen zuvor
1164:  movem.l (a7)+,d2/d3
1165:  bne errwi  * Fehler bei Ablage
1166:  movem.l d2/d3/a0,-(a7)
1167:  bsr hdread * und Lesen dann.
1168:  movem.l (a7)+,d2/d3/a0
1169:  bne errwi
1170:  move.l d2,wisektor * neuer Sektor
1171:  clr d0
1172:  rts
1173:
1174:
1175: setupwi:
1176:  tst imwr
1177:  beq setlup
1178:  bsr wiini     * Aufrufen Init der Winchester
1179:  clr imwr      * Nur nach Kaltstart.
1180: setlup:
1181:  move.w sector,d0 * 1..256
1182:  sub #1,d0     * 0..255
1183:  clr.l d3
1184:  move.w track,d3 * 0...n-1
1185:  rol.l #8,d3
1186:  move.b d0,d3  * d3=logischer Sektor zu je 128 Byte.
1187:  move.l d3,d2
1188:  asr.l #1,d2    * d2=phys. Sektor zu je 256 Byte.
1189:  rts
1190:
1191: *
1192:  .data
1193:
1194: stepmaxi: dc.b 0 * 0=schnellste ,1,2,3
1195: stepmini: dc.b 0 * 0=schnellste ,1,2,3
1196: iobyte:   dc.w 0 * nur Byte gueltig.
1197:
1198: * rev 2.1
1199: imwr: dc.w $ffff    * Winchester noch nicht aufgerufen.
1200: imflag: dc.w 0      * unbeschriebener Sektor
1201: wisektor: dc.l $ffffffff  * Winchester-Sektor, undef.
1202: * rev 2.1
1203: serinitd: dc.w 0    * 0=nicht initialisiert. Fuer INIT.
1204:
1205: * drvcode bestimmt das gebootete Laufwerk
1206: * er entspricht der internen Codierung,
1207: * also $11 bei Maxi, SD
1208: * und  $21 bei Mini, DD
1209: * nur diese beiden Formate werden z.Z. ausgewertet
1210:
```

```
1211: drvcode: dc.b 0
1212:          dc.b 0    * angleich EVEN
1213:
1214: **
1215: cursor:   dc.w 0    * Merker Cursor 1=ein, 0=aus
1216: dma:      dc.l 0
1217: track:    dc.w 0    * rev 2.0 Wortgross
1218: sector:   dc.w 0    * rev 2.0 Wortgross.
1219:
1220: seldrv:   dc.b 0
1221: lastsso:  dc.b $FF * Merker letzter SSO-Zustand (0,$80) FF=unguelt.
1222: *                    0=ohne SSO, $80=SSO war gesetzt
1223: mdrvakt:  dc.b 0    * Intern code 0=unguelt.
1224: mtrkakt:  dc.b 0    * fuer Bufferverwaltung
1225: msekakt:  dc.b 0
1226: alloc:    dc.b 0
1227:
1228: mwrtflg: dc.b 0    * <> 0 dann schreiben noch noetig
1229:
1230:  ds 0
1231:
1232: gru68k:  dc.l 0          * Basis-Adresse Grundprogramm
1233: gruj68k: dc.l $420       * Basis-Adresse + $420 fuer Sprung
1234: cpu:     dc.l 1          * CPU, 1=68008, 2=68000, 4=68020 (32er)
1235:
1236: * Adressen der IO-Ports. Sie werden in wiini
1237: * berechnet, da sie CPU-abhaengig sind.
1238: xhddata: dc.l 0          * hddata = (hdbase)*cpu
1239: xhdstat: dc.l 0          * hdstat = (hdbase+1)*cpu
1240: xhdrst:  dc.l 0          * hdrst  = (hdbase+1)*cpu
1241: xhdsel:  dc.l 0          * hdsel  = (hdbase+2)*cpu
1242:
1243: ramadr:  dc.l 0          * Start RAMFLOPPY 0=undef.
1244:
1245:
1246: memrgn:
1247:  dc.w 1              * nur eine TPA vorhanden (immer so.)
1248:  dc.l $400           * Start der TPA
1249:  dc.l cpm-$400       * 1024 Bytes gehen durch Vektoren verloren
1250:
1251: * disk parameters
1252:
1253: maxdsk equ 8    * rev 2.0
1254: dphlen equ 26
1255:
1256: dph0:     * 8 "  a
1257:  dc.l xlt
1258:  dc.w 0
1259:  dc.w 0
1260:  dc.w 0
1261:  dc.l dirbuf
1262:  dc.l dpb
1263:  dc.l ckv0
1264:  dc.l alv0
1265:
1266: dph1:     * 8 " b
1267:  dc.l xlt
1268:  dc.w 0
1269:  dc.w 0
1270:  dc.w 0
1271:  dc.l dirbuf
1272:  dc.l dpb
1273:  dc.l ckv1
1274:  dc.l alv1
1275:
1276: dph2:     * 8 " c
1277:  dc.l xlt
1278:  dc.w 0
1279:  dc.w 0
1280:  dc.w 0
1281:  dc.l dirbuf
1282:  dc.l dpb
1283:  dc.l ckv2
1284:  dc.l alv2
1285:
1286: dph3:     * 8 " d
```

fekten Sektor. Am Anfang wird außerdem geprüft, ob die RAM-Floppy überhaupt vorhanden ist. Wenn nämlich RAMADR den Wert 0 hat, so gibt es überhaupt keine RAM-Floppy im System.

Die Winchester-Routinen

Die Unterprogramme sind ähnlich zu denen aus Bild 1, auch hier gibt es eine DRVTBL mit der Information über die Winchesterparameter, und die Routinen WIINI, HDWRITE und HDREAD. Dabei sind alle I/O-Adressen Speicherzellen entnommen, deren Werte durch WIINI berechnet wurden. Damit wird das BIOS unabhängig von der verwendeten CPU.

Die eigentlichen Programme für das BIOS tragen den Namen READWIN und WRITEWIN. Dabei enthält das Register D2.L den logischen Sektor von 0...n-1, wobei hier 128 Byte pro Sektor angenommen werden. Die Umrechnung auf 256 Byte erfolgt in den Routinen selbst, ähnlich wie bei den Minilaufwerken. Der Sektor wird erst dann zurückgeschrieben, wenn entweder ein neuer benötigt wird, und der Sektor auch vorher beschrieben wurde, oder wenn ein Direktory-Schreib-Zugriff erfolgt. Schlußendlich folgen die Tabellen für die Laufwerksbestimmung durch das BDOS.

Sie entsprechen in etwa denen des CP/M-80, wobei in den meisten Fällen lediglich Langworte anstelle von Worten verwendet wurden.

```
1287:  dc.l xlt
1288:  dc.w 0
1289:  dc.w 0
1290:  dc.w 0
1291:  dc.l dirbuf
1292:  dc.l dpb
1293:  dc.l ckv3
1294:  dc.l alv3
1295:
1296:  dph4:   * 5 1/4 " e
1297:  dc.l 0        * no table
1298:  dc.w 0
1299:  dc.w 0
1300:  dc.w 0
1301:  dc.l dirbuf
1302:  dc.l dpbmini
1303:  dc.l ckv4
1304:  dc.l alv4
1305:
1306:
1307:  dph5:   * 5 1/4 " f
1308:  dc.l 0  * no table
1309:  dc.w 0
1310:  dc.w 0
1311:  dc.w 0
1312:  dc.l dirbuf
1313:  dc.l dpbmini
1314:  dc.l ckv5
1315:  dc.l alv5
1316:
1317:
1318:  dph6:    * Ramfloppy g  rev 2.1
1319:  dc.l 0   * no table
1320:  dc.w 0
1321:  dc.w 0
1322:  dc.w 0
1323:  dc.l dirbuf
1324:  dc.l dpbram
1325:  dc.l 0   * no check
1326:  dc.l alv6
1327:
1328:  dph7:    * Winchester 6188
1329:  dc.l 0   * no table
1330:  dc.w 0
1331:  dc.w 0
1332:  dc.w 0
1333:  dc.l dirbuf
1334:  dc.l dpbwin
1335:  dc.l 0   * no check
1336:  dc.l alv7
1337:
1338:  dpb:
1339:  dc.w 26
1340:  dc.b 3
1341:  dc.b 7
1342:  dc.b 0
1343:  dc.b 0
1344:  dc.w 242
1345:  dc.w 63
1346:  dc.w $c000
1347:  dc.w 16
1348:  dc.w 2
1349:
1350:  dpbmini:
1351:  dc.w 40          * 40 log. 128 Byte Sektoren
1352:  dc.b 4           * BSH    2048 Bytes / Block
1353:  dc.b 15          * BLM
1354:  dc.b 0           * EXTNT MASK
1355:  dc.b 0
1356:  dc.w 388         * Kapazitaet * 2048
1357:  dc.w 255         * Direktory Eintraege-1
1358:  dc.w $f000       * Direktory mask (reserved )
1359:  dc.w 64          * check size
1360:  dc.w 4           * offset
1361:
1362:
```

```
1363:  dpbram:
1364:  dc.w 256         * 256 log. 128 Byte Sektoren pro Track.
1365:  dc.b 4           * BSH    2048 Bytes / Block
1366:  dc.b 15          * BLM
1367:  dc.b 0           * EXTNT MASK
1368:  dc.b 0
1369:  ramkap:          * Wird berechnet
1370:  dc.w 255         * (Kapazitaet+1) * 2048 , z.B. 512K
1371:  dc.w 255         * Direktory Eintraege-1
1372:  dc.w $f000      * Direktory mask (reserved )
1373:  dc.w 0           * no check
1374:  dc.w 0           * offset
1375:
1376:
1377:  dpbwin:          * rev 2.0
1378:  dc.w 256         * 256 log. 128 Byte Sektoren
1379:  dc.b 5           * BSH 4096 Bytes pro Block
1380:  dc.b 31          * BLM
1381:  dc.b 1           * EXTNT MASK
1382:  dc.w 2399        * 9.830400 MByte Disksize-1 (mit reserve)
1383:  dc.w 1023        * 1024 Direktory Eintraege
1384:  dc.w $ff00       * 8 Bloecke reserviert bei 1024 Dirs.
1385:  dc.w 0           * no check
1386:  dc.w 1           * 1 Track Offset Reserve. 32KByte.
1387:  *
1388:
1389:
1390:  xlt:
1391:  dc.b 1,7,13,19
1392:  dc.b 25,5,11,17
1393:  dc.b 23,3,9,15
1394:  dc.b 21,2,8,14
1395:  dc.b 20,26,6,12
1396:  dc.b 18,24,4,10
1397:  dc.b 16,22
1398:
1399:  ds 0
1400:
1401:  .bss
1402:
1403:  dirbuf:
1404:  ds.b 128
1405:  ckv0: ds.b 16
1406:  ckv1: ds.b 16
1407:  ckv2: ds.b 16
1408:  ckv3: ds.b 16
1409:  ckv4: ds.b 64
1410:  ckv5: ds.b 64
1411:
1412:  alv0: ds.b 32
1413:  alv1: ds.b 32
1414:  alv2: ds.b 32
1415:  alv3: ds.b 32
1416:  alv4: ds.b 50        * (49+1), Achtung, alvs auf even
1417:  alv5: ds.b 50        * sonst Fehler in STAT
1418:  alv6: ds.b 64        * Dynamisches Ende.( fuer max. 1MByte)
1419:  alv7: ds.b 302       * Blockcount / 8 + savety
1420:
1421:  buffer: ds.b 1024    * Sektor Speicher fuer Mini-Laufwerk
1422:
1423:  * Winchester Speicherbereiche.
1424:  task:   ds.b 6
1425:  hstbuf: ds.b 256     * Sektor Speicher fuer Winchester.
1426:
1427:  ds 0
1428:
1429:  * Achtung diesen Bereich nicht fuer dynamische
1430:  * Strukturen verwenden, da beim Linken weitere
1431:  * Bereiche hier zu liegen kommen.
1432:
1433:  .end
1434:
1435:
1436:
1437:
1438:
```