```
       1/       0 :                             cpu Z8601
       2/       0 :                             page 0
       3/       0 :                             include stddefZ8.inc
(1)    1/       0 :                                     save
(1)   55/       0 :                                     restore
(1)   56/       0 :
(1)   57/       0 :
       4/       0 :
       5/       0 :                     ;*****************************************************************
       6/       0 :                     ;
       7/       0 :                     ;              Apple "Rene" HD20 Controller - Z8 firmware
       8/       0 :                     ;                   Version 342-0343-B (3372)
       9/       0 :                     ;
      10/       0 :                     ;
      11/       0 :                     ;  commented and converted into as source by Patrick Schaefer, 2013
      12/       0 :                     ;      (some blocks were re-used from the Widget source code)
      13/       0 :                     ;
      14/       0 :                     ;    use as build 2011-08-02 or later. Last changes: 2013-09-30 PS
      15/       0 :                     ;
      16/       0 :                     ;*****************************************************************
      17/       0 :
      18/       0 :
      19/       0 : =1H                 DontListIncls   SET 1
      20/       0 :
      21/       0 : =2000H              ROMsize         SET 8192
      22/       0 : =98C6H              Checksum0       SET 098C6h              ; for ROM test bank 0
      23/       0 : =B74DH              Checksum1       SET 0B74Dh              ;              bank 1
      24/       0 : =F078H              Passwrd1        SET 0F078h              ; upper nibble high shifted right
      25/       0 : =3C1EH              Passwrd2        SET 03C1Eh
      26/       0 : =33H                HiRevNumber     SET 033h
      27/       0 : =72H                LoRevNumber     SET 072h
      28/       0 :
      29/       0 :
      30/       0 :                     ; external calls into Z8 internal memory 341-0339-A
      31/       0 : =E9H                EpromTest       EQU 000E9h
      32/       0 : =124H               MsWait          EQU 00124h
      33/       0 : =198H               L0198           EQU 00198h
      34/       0 : =1E2H               L01e2           EQU 001E2h
      35/       0 : =1E7H               L01e7           EQU 001E7h
      36/       0 : =1ECH               L01ec           EQU 001ECh
      37/       0 : =1F0H               L01f0           EQU 001F0h
      38/       0 : =1F8H               L01f8           EQU 001F8h
      39/       0 : =1FFH               L01FF           EQU 001FFh
      40/       0 : =201H               L0201           EQU 00201h
      41/       0 : =203H               L0203           EQU 00203h
      42/       0 : =205H               L0205           EQU 00205h
      43/       0 : =265H               L0265           EQU 00265h
      44/       0 : =28EH               L028e           EQU 0028Eh
      45/       0 : =29BH               L029b           EQU 0029Bh
      46/       0 : =2ABH               L02ab           EQU 002ABh
      47/       0 : =2C9H               L02c9           EQU 002C9h
      48/       0 : =2CBH               Bank_Call       EQU 002CBh
      49/       0 : =2F6H               Bank_Ret        EQU 002F6h
      50/       0 : =357H               Abort           EQU 00357h
      51/       0 : =39FH               SS_OpFail       EQU 0039Fh
      52/       0 : =3A7H               L03a7           EQU 003A7h
      53/       0 : =3ADH               SS_ReadErr      EQU 003ADh
      54/       0 : =3B6H               SS_NoHdr        EQU 003B6h
      55/       0 : =3BCH               SS_SprWarn      EQU 003BCh
      56/       0 : =3C2H               SetStatus       EQU 003C2h
      57/       0 : =3C8H               Reset_StMach    EQU 003C8h
      58/       0 : =3F0H               Load_Header     EQU 003F0h
      59/       0 : =418H               Load_Logical    EQU 00418h
      60/       0 : =428H               Chk_Chk_Byte    EQU 00428h
      61/       0 : =439H               Gen_Chk_Byte    EQU 00439h
      62/       0 : =450H               L0450           EQU 00450h
      63/       0 : =459H               L0459           EQU 00459h
      64/       0 : =46CH               LoadStatus      EQU 0046Ch
      65/       0 : =47CH               L047c           EQU 0047Ch
      66/       0 : =489H               L0489           EQU 00489h
      67/       0 : =48FH               L048f           EQU 0048Fh
      68/       0 : =49BH               L049b           EQU 0049Bh
      69/       0 : =4A3H               L04a3           EQU 004A3h
      70/       0 : =4D0H               L04d0           EQU 004D0h
      71/       0 : =4DBH               L04db           EQU 004DBh
      72/       0 : =4E6H               L04e6           EQU 004E6h
      73/       0 : =4EDH               L04ed           EQU 004EDh
      74/       0 : =501H               L0501           EQU 00501h
      75/       0 : =51BH               L051b           EQU 0051Bh
      76/       0 : =522H               Ext_Push        EQU 00522h
      77/       0 : =54FH               Ext_Pop         EQU 0054Fh
      78/       0 : =586H               L0586           EQU 00586h
      79/       0 : =58EH               ZeroHeader      EQU 0058Eh
      80/       0 : =59DH               L059d           EQU 0059Dh
      81/       0 : =5A3H               L05a3           EQU 005A3h
      82/       0 : =5BAH               L05ba           EQU 005BAh
      83/       0 : =5E7H               Load_PassWord   EQU 005E7h
      84/       0 : =651H               L0651           EQU 00651h
      85/       0 : =65BH               AdjustGeometry  EQU 0065Bh
      86/       0 : =69DH               L069d           EQU 0069Dh
      87/       0 :
      88/       0 :
      89/       0 :                             include DefsHD20.inc
(1)    1/       0 :                                     save
(1)    2/       0 : =>TRUE                      if DontListIncls
(1)  603/    16AA :                                     restore
(1)  604/    16AA :
(1)  605/    16AA :
(1)  606/    16AA :
      90/    16AA :
      91/    16AA :                     ;*****************************************************************
      92/    16AA :                     ; Module Bank0.Assem
      93/    16AA :                     ;
      94/    16AA :                     ;*****************************************************************
      95/    16AA :
      96/       0 :                             org 0000h
      97/    1000 :                             phase 1000h                     ; EPROM will start at 4k
      98/    1000 :                             assume RP:Wrk_Sys
      99/    1000 : 98 C6                               DW Checksum0
     100/    1002 : 00                                  DB 0                    ; bank 0
     101/    1003 : F0 78               Password        DW Passwrd1
     102/    1005 : 3C 1E                               DW Passwrd2
     103/    1007 :
     104/    1007 :                     ; Vector Table
     105/    1007 :                     B0_VctTab
```

```
106/    1007 : 8D 10 CE           Start_Vector    jp Start_Command
107/    100A : 8D 11 46           RdL_Vector      jp Rd_Leave
108/    100D : 8D 10 AE           Free_Vector     jp Strt_FreeProc1
109/    1010 : 8D 10 A8           SlfTst_Vector   jp SelfTest1
110/    1013 : 8D 10 A2           SprTbl_Vector   jp Load_SprTbl1
111/    1016 : 8D 10 99           IScan_SprChk    jp Chk_SprCnt1
112/    1019 : 8D 10 6E           L1019           jp L106e
113/    101C : 8D 10 6A           L101c           jp L106a
114/    101F : 8D 10 B4           L101f           jp L10b4
115/    1022 : 8D 1C F4           L1022           jp L1cf4
116/    1025 : 8D 1C C3           L1025           jp L1cc3
117/    1028 : 8D 1C AF           L1028           jp L1caf
118/    102B : 8D 1C 80           L102b           jp L1c80
119/    102E : 8D 1C FA           L102e           jp L1cfa
120/    1031 : 8D 1C 77           L1031           jp L1c77
121/    1034 :
122/    1034 :                    ; Command Table
123/    1034 : 17 F8              Cmnd_Ptrs       DW Sys_Read             ;  0 MultiBlock Read
124/    1036 : 19 08                              DW Sys_Write            ;  1 MultiBlock Write
125/    1038 : 19 72                              DW Sys_WrVer            ;  2 MultiBlock WriteVerify
126/    103A : 15 87                              DW L1587                ;  3
127/    103C : 15 39                              DW Read_ID              ;  4 Read Device ID
128/    103E : 16 28                              DW Read_CStatus         ;  5 Read Controller Status
129/    1040 : 16 8B                              DW Read_SStatus         ;  6 Read Servo Status
130/    1042 : 16 B6                              DW Send_ServoCmnd       ;  7 Send Servo Command
131/    1044 : 16 D4                              DW Send_Seek            ;  8 Seek
132/    1046 : 16 F4                              DW Send_Restore         ;  9 Data Recal / Brake Release
133/    1048 : 17 10                              DW Set_Recovery         ;  A Set Recovery
134/    104A : 00 0C                              DW 0000Ch               ;  B Soft Reset
135/    104C : 17 37                              DW Send_Park            ;  C Park
136/    104E : 15 91                              DW D_Read               ;  D Diag Read
137/    1050 : 15 C6                              DW D_ReadHdr            ;  E Diag Read Header
138/    1052 : 16 0F                              DW D_Write              ;  F Diag Write
139/    1054 : 17 41                              DW Set_AutoOffset       ; 10 Auto Offset
140/    1056 : 15 7D                              DW Read_SprTbl          ; 11 Read Spare Table
141/    1058 : 17 4B                              DW Wr_SprTbl            ; 12 Write Spare Table
142/    105A : 17 71                              DW Format               ; 13 Format Track
143/    105C : 11 39                              DW Abort_10             ; 14 (Abort with code 10)
144/    105E : 17 95                              DW Read_Abort           ; 15 Read Abort Status
145/    1060 : 17 AC                              DW D_RstSrvo            ; 16 Reset Servo
146/    1062 : 17 BC                              DW D_RdTrack            ; 17 Read Track
147/    1064 : 17 B6                              DW D_WrTrack            ; 18 Write Track
148/    1066 : 17 E4                              DW L17e4                ; 19
149/    1068 : 17 EE                              DW L17ee                ; 20
150/    106A :
151/    106A : D6 11 9D           L106a           call L119d
152/    106D : AF                                 ret
153/    106E :
154/    106E : D6 05 1B           L106e           call L051b
155/    1071 : EC 12                              ld R14, #WEndGap /256
156/    1073 : FC 3B                              ld R15, #WEndGap #256
157/    1075 : D6 05 E7                           call Load_PassWord
158/    1078 : D6 05 BA                           call L05ba
159/    107B : 0C 00                              ld R0, #0
160/    107D : 1C 00                              ld R1, #0
161/    107F : D6 04 D0                           call L04d0
162/    1082 : D6 04 ED                           call L04ed
163/    1085 : B0 E0                              clr R0
164/    1087 : D6 04 E6                           call L04e6
165/    108A : 46 3E 04                           or 3Eh, #004h
166/    108D : AF                                 ret
167/    108E :
168/    108E : D6 19 B9           WrBlk_Vector    call WriteBlock
169/    1091 : 8D 02 F6           VctrB0_Ret      jp Bank_Ret
170/    1094 : D6 1A 4B           RdBlk_Vector    call ReadBlock
171/    1097 : 8B F8                              jr VctrB0_Ret
172/    1099 :
173/    1099 : 2C 23              Chk_SprCnt1     ld R2, #Chk_SprCnt /256
174/    109B : 3C 76                              ld R3, #Chk_SprCnt #256
175/    109D : D6 02 CB           VctrB0_BC       call Bank_Call
176/    10A0 : 8B EF                              jr VctrB0_Ret
177/    10A2 :
178/    10A2 : 2C 21              Load_SprTbl1    ld R2, #Load_SprTbl /256
179/    10A4 : 3C 69                              ld R3, #Load_SprTbl #256
180/    10A6 : 8B F5                              jr VctrB0_BC
181/    10A8 :
182/    10A8 : 2C 29              SelfTest1       ld R2, #SelfTest /256
183/    10AA : 3C 8A                              ld R3, #SelfTest #256
184/    10AC : 8B EF                              jr VctrB0_BC
185/    10AE :
186/    10AE : 2C 2B              Strt_FreeProc1  ld R2, #Strt_FreeProcess /256
187/    10B0 : 3C 5D                              ld R3, #Strt_FreeProcess #256
188/    10B2 : 8B E9                              jr VctrB0_BC
189/    10B4 :
190/    10B4 : 8D 11 69           L10b4           jp L1169
191/    10B7 :
192/    10B7 :
193/    10B7 :                    ; Device ID block
194/    10B7 :                    ;
195/    10B7 : 52 65 6E 65 2D 31  DeviceParams    DB "Rene-1 RM MH "      ; device name
                20 52 4D 20 4D 48
                20
196/    10C4 : 00 02 10                           DB 000h, 002h, 010h     ; device type
197/    10C7 : 33 72                              DB HiRevNumber, LoRevNumber
198/    10C9 : 00 98 35                           DB 000h, 098h, 035h     ; number of logical blocks
199/    10CC : 02 14                              DW 532                  ; number of bytes per block
200/    10CE : =17H               Dev_Parm_Length EQU $ - DeviceParams
201/    10CE :
202/    10CE :
203/    10CE :
204/    10CE :                    ;*****************************************************************
205/    10CE :                    ; Module Cmnd.Assem
206/    10CE :                    ;
207/    10CE :                    ; This module controls the way in which commands received by
208/    10CE :                    ; the Host are executed. Its main responsibility is to determine
209/    10CE :                    ; whether a command string is protected by a check byte (the
210/    10CE :                    ; original ProFile commands are not) and then decode the command
211/    10CE :                    ; and pass control over to the correct driver routine to
212/    10CE :                    ; execute the command. All exception handling at this level is
213/    10CE :                    ; controlled by the individual command routines.
214/    10CE :                    ;
215/    10CE :                    ;       PROCEDURE Start_Command
216/    10CE :                    ;       PROCEDURE Pro_Read
217/    10CE :                    ;       FUNCTION Read_Common : BOOLEAN
218/    10CE :                    ;
219/    10CE :                    ;*****************************************************************
220/    10CE :
221/    10CE :                    ;*****************************************************************
```

```
222/    10CE :                  ;
223/    10CE :                  ; Procedure: StartCommand
224/    10CE :                  ;
225/    10CE :                  ;  Assumes a command string in external memory; the
226/    10CE :                  ;  command byte must be in location $0000 (external)
227/    10CE :                  ;
228/    10CE :                  ; Algorithm:
229/    10CE :                  ;
230/    10CE :                  ;  Begin
231/    10CE :                  ;    Initialize internal and external stack ptrs
232/    10CE :                  ;    ZeroHeader
233/    10CE :                  ;    Excpt_Stat.Buf_Damage:=false
234/    10CE :                  ;    Excpt_Stat.Nzero_Stat:=false
235/    10CE :                  ;    If Cache_Index = Cahe_Length
236/    10CE :                  ;      Then Cache_Index:=0
237/    10CE :                  ;    OpCode:=ExtMem[0]
238/    10CE :                  ;    If OpCode.CommandType is FreeProcess type
239/    10CE :                  ;      Then goto Strt_FreeProcess
240/    10CE :                  ;    If OpCode.CommandType is protected by a checkbyte
241/    10CE :                  ;      Then
242/    10CE :                  ;        If not(Check_Command_String)
243/    10CE :                  ;          Then Abort(Cmnd_Driver_Exception,
244/    10CE :                  ;                     Start_Command, CheckByte_Mismatch)
245/    10CE :                  ;    If OpCode.Instruction > CommandLimit[CommandType]
246/    10CE :                  ;      Then Abort(Cmnd_Driver_Exception,
247/    10CE :                  ;                 Start_Command, IllegalOpCode, OpCode)
248/    10CE :                  ;    JMP @Command^[CommandType].RoutineTable[Instruction]
249/    10CE :                  ;  End
250/    10CE :                  ;
251/    10CE :                  ;*****************************************************************
252/    10CE :
253/    10CE : 8F               Start_Command   di
254/    10CF : 31 10                            srp #Wrk_Sys            ; get into a reasonable state
255/    10D1 :                          assume RP:Wrk_Sys
256/    10D1 : E6 FF 80                         ld SPL, #Stack_Top
257/    10D4 : 46 32 04                         or Excpt_Stat, #UseECC
258/    10D7 : 56 32 F5                         and Excpt_Stat, #0FFh-NZero_Stat-002h   ; ???
259/    10DA : 56 3E 9D                         and 3Eh, #0FFh-S_Block-B_Block-002h
260/    10DD : D6 05 1B                         call L051b
261/    10E0 : D6 02 8E                         call L028e
262/    10E3 : 2C 10                            ld R2, #Cmnd_Ptr /256   ; host passes command here
263/    10E5 : 3C 1B                            ld R3, #Cmnd_Ptr #256
264/    10E7 : 82 42                            lde R4, @RR2            ; command byte into Status0
265/    10E9 : 49 5F                            ld Status0, R4
266/    10EB : 46 5F 80                         or Status0, #080h       ; set bit 7
267/    10EE : E6 5E 00                         ld Status1, #0          ; clear Status1
268/    10F1 :                  ; copy host command into HostCmndBuf
269/    10F1 : EC 14                            ld R14, #(HostCmndBuf) /256
270/    10F3 : FC 83                            ld R15, #(HostCmndBuf) #256
271/    10F5 : 1C 08                            ld R1, #8               ; copy 8 bytes
272/    10F7 : 82 02            St_L_Lp         lde R0, @RR2
273/    10F9 : 92 0E                            lde @RR14, R0
274/    10FB : A0 E2                            incw RR2
275/    10FD : A0 EE                            incw RR14
276/    10FF : 1A F6                            djnz R1, St_L_Lp
277/    1101 :                  ;
278/    1101 : 56 50 EF                         and 50h, #0FFh-010h
279/    1104 : A6 E4 02                         cp R4, #2               ; cmd 0..2?
280/    1107 : 2D 11 1E                         jp LE, Chk_Inst         ;  --> these need passed selftest!
281/    110A : A6 E4 05                         cp R4, #5               ; cmd 5 (controller status)?
282/    110D : 6D 11 28                         jp Z, Ok_Inst           ;  -->
283/    1110 : 2C 14            Chk_Inst1       ld R2, #014h
284/    1112 : 3C 7B                            ld R3, #07Bh
285/    1114 : 1C 04                            ld R1, #4
286/    1116 : 0C 00                            ld R0, #0
287/    1118 : D6 04 50                         call L0450
288/    111B : 8D 11 28                         jp Ok_Inst
289/    111E :                  ; cmd 0..2
290/    111E : 44 33 33         Chk_Inst        or SlfTst_Result,SlfTst_Result  ; check if we're not healthy
291/    1121 : EB 1F                            jr nz, Inst_Abort1
292/    1123 : 46 50 10                         or 50h, #010h
293/    1126 : 8B E8                            jr Chk_Inst1
294/    1128 :
295/    1128 :                  ; jump to command subroutine, command number in R4
296/    1128 : EC 10            Ok_Inst         ld R14, #Cmnd_Ptrs /256 ; offset to command vector table
297/    112A : FC 34                            ld R15, #Cmnd_Ptrs #256
298/    112C : CF                               rcf
299/    112D : 90 E4                            rl R4                   ; *2
300/    112F : 02 F4                            add R15, R4             ; get offset into table
301/    1131 : C2 CE                            ldc R12, @RR14          ; get address to routine
302/    1133 : A0 EE                            incw RR14
303/    1135 : C2 DE                            ldc R13, @RR14
304/    1137 : 30 EC                            jp @RR12                ; jump to routine
305/    1139 :
306/    1139 :
307/    1139 :                  ; Cmd 14: Abort with code 10
308/    1139 : FC 0A            Abort_10        ld R15, #10
309/    113B : 8D 03 57         Inst_Abort      jp Abort
310/    113E :                  ;
311/    113E : FC 08            Abort_8         ld R15, #8
312/    1140 : 8B F9                            jr Inst_Abort
313/    1142 :                  ;
314/    1142 : FC 09            Inst_Abort1     ld R15, #9              ; command rejected because selftest failed
315/    1144 : 8B F5                            jr Inst_Abort
316/    1146 :
317/    1146 :
318/    1146 :                  ; we arrive here after completion of any command
319/    1146 : 56 3E E7         Rd_Leave        and 3Eh, #0FFh-010h-008h
320/    1149 : D6 11 69                         call L1169              ; assemble status word
321/    114C : 8D 01 98         L114c           jp L0198
322/    114F :
323/    114F :
324/    114F : 2C 20            Pro_Rd_BB       ld R2, #RBuf_To_Buf2 /256
325/    1151 : 3C 8A                            ld R3, #RBuf_To_Buf2 #256
326/    1153 : D6 02 CB                         call Bank_Call
327/    1156 : 0C 82                            ld R0, #Error+Ex_SprBlock       ; Spare this block!
328/    1158 : 66 E0 80                         tcm R0, #Error          ; set Z flag
329/    115B : AF                               ret
330/    115C :
331/    115C :
332/    115C : 2C 20            Wr_BBlock       ld R2, #WrBuf_To_Buf2 /256
333/    115E : 3C 56                            ld R3, #WrBuf_To_Buf2 #256
334/    1160 : D6 02 CB                         call Bank_Call
335/    1163 : 0C 82                            ld R0, #Error+Ex_SprBlock       ; Spare this block!
336/    1165 : D6 13 BD                         call Data_Ex_Handler
337/    1168 : AF                               ret
338/    1169 :
339/    1169 :
```

```
340/    1169 :                      ; send completion status
341/    1169 : 76 32 40             L1169           tm Excpt_Stat, #SprTbl_Warn
342/    116C : 6B 07                                jr Z, L1175
343/    116E : 0C 00                                ld R0, #0
344/    1170 : 1C 80                                ld R1, #080h
345/    1172 : D6 02 9B                             call L029b
346/    1175 :
347/    1175 : 0C 02                L1175           ld R0, #2
348/    1177 : 18 5D                                ld R1, 5Dh
349/    1179 : D6 02 9B                             call L029b
350/    117C : EC 10                                ld R14, #(RBuffer1-6) /256      ; RAM 1013h
351/    117E : FC 13                                ld R15, #(RBuffer1-6) #256
352/    1180 : 2C 14                                ld R2, #014h
353/    1182 : 3C 7B                                ld R3, #07Bh            ; 147Bh
354/    1184 : 08 5F                                ld R0,Status0
355/    1186 : 92 0E                                lde @RR14, R0           ; first Status0
356/    1188 : A0 EE                                incw RR14
357/    118A : 08 5E                                ld R0,Status1
358/    118C : 92 0E                                lde @RR14, R0           ; then Status1
359/    118E : A0 EE                                incw RR14
360/    1190 : 1C 04                                ld R1, #4
361/    1192 : 82 02                Move4_Lp        lde R0, @RR2            ; then 147B..147Eh
362/    1194 : 92 0E                                lde @RR14, R0
363/    1196 : A0 EE                                incw RR14
364/    1198 : A0 E2                                incw RR2
365/    119A : 1A F6                                djnz R1,Move4_Lp
366/    119C : AF                                   ret
367/    119D :
368/    119D : D6 02 8E             L119d           call L028e
369/    11A0 : E6 5D 00                             ld 5Dh, #0
370/    11A3 : AF                                   ret
371/    11A4 :
372/    11A4 : 46 35 2C             WrVer_Common    or DiskStat, #Wr_Op+Offset_On+User_Type
373/    11A7 : D6 11 D3                             call RW_Common
374/    11AA : 6B 22                                jr Z, WrVer_Ret
375/    11AC : 2C 20                                ld R2, #WrBuf_To_Buf2 /256
376/    11AE : 3C 56                                ld R3, #WrBuf_To_Buf2 #256
377/    11B0 : D6 02 CB                             call Bank_Call
378/    11B3 : 56 35 DF                             and DiskStat, #0FFh-Wr_Op
379/    11B6 : D6 11 D3                             call RW_Common
380/    11B9 : 6B 03                                jr Z, WrVer_Ret1
381/    11BB : 8D 02 F6             L11bb           jp Bank_Ret
382/    11BE : A6 E0 88             WrVer_Ret1      cp R0, #088h
383/    11C1 : EB 04                                jr nz, L11c7
384/    11C3 : 0C 8A                                ld R0, #08Ah
385/    11C5 : 8B 07                                jr WrVer_Ret
386/    11C7 : A6 E0 84             L11c7           cp R0, #084h
387/    11CA : EB 02                                jr nz, WrVer_Ret
388/    11CC : 0C 82                                ld R0, #082h
389/    11CE : D6 13 BD             WrVer_Ret       call Data_Ex_Handler
390/    11D1 : 8B E8                                jr L11bb
391/    11D3 :
392/    11D3 : 76 35 20             RW_Common       tm DiskStat, #Wr_Op     ; are we writing?
393/    11D6 : 6B 07                                jr Z, RW_Cmd_Rd
394/    11D8 : D6 19 B9                             call WriteBlock
395/    11DB : EB 7C                                jr NZ, L1259
396/    11DD : 8B 05                                jr L11e4
397/    11DF :
398/    11DF : D6 1A 4B             RW_Cmd_Rd       call ReadBlock
399/    11E2 : EB 75                                jr NZ, L1259
400/    11E4 :
401/    11E4 : 70 E4                L11e4           push R4
402/    11E6 : 76 32 08                             tm Excpt_Stat, #NZero_Stat
403/    11E9 : EB 06                                jr NZ, L11f1
404/    11EB : D6 03 A7                             call L03a7
405/    11EE : 46 32 08                             or Excpt_Stat, #NZero_Stat
406/    11F1 : 76 35 20             L11f1           tm DiskStat, #Wr_Op
407/    11F4 : 6B 07                                jr Z, L11fd
408/    11F6 : 2C 20                                ld R2, #WrBuf_To_Buf2 /256
409/    11F8 : 3C 56                                ld R3, #WrBuf_To_Buf2 #256
410/    11FA : D6 02 CB                             call Bank_Call
411/    11FD : 4C 80                L11fd           ld R4, #080h
412/    11FF : 76 32 80                             tm Excpt_Stat, #Recovery
413/    1202 : 6B 4E                                jr Z, L1252
414/    1204 :
415/    1204 : 4C 86                L1204           ld R4, #086h
416/    1206 : 6C 04                                ld R6, #004h
417/    1208 : 76 34 68             L1208           tm RWStat, #RdSrvoErr+RdNoHdrFnd+RdSMErr
418/    120B : 6B 66                                jr Z, L1273
419/    120D : 70 35                L120d           push DiskStat
420/    120F : 2C 28                                ld R2, #SrvoRcvry /256
421/    1211 : 3C 3A                                ld R3, #SrvoRcvry #256
422/    1213 : D6 02 CB                             call Bank_Call
423/    1216 : 50 35                                pop DiskStat
424/    1218 : 76 35 20                             tm DiskStat, #Wr_Op
425/    121B : 6B 0C                                jr Z, L1229
426/    121D : 2C 20                L121d           ld R2, #Buf2_To_WrBuf /256
427/    121F : 3C 61                                ld R3, #Buf2_To_WrBuf #256
428/    1221 : D6 02 CB                             call Bank_Call
429/    1224 : D6 19 B9                             call WriteBlock
430/    1227 : 8B 0E                                jr L1237
431/    1229 :
432/    1229 : D6 1A 4B             L1229           call ReadBlock
433/    122C : 6B 09                                jr Z, L1237
434/    122E : 2C 20                                ld R2, #RBuf_To_Buf2 /256
435/    1230 : 3C 8A                                ld R3, #RBuf_To_Buf2 #256
436/    1232 : D6 02 CB                             call Bank_Call
437/    1235 : 8B 1B                                jr L1252
438/    1237 :
439/    1237 : 6A CF                L1237           djnz R6, L1208
440/    1239 : 4C 80                                ld R4, #080h
441/    123B : 76 34 40                             tm RWStat, #RdSrvoErr
442/    123E : 6B 08                                jr Z, L1248
443/    1240 : D6 03 C2                             call SetStatus
444/    1243 : FC 0B                                ld R15, #11
445/    1245 : 8D 03 57                             jp Abort
446/    1248 :
447/    1248 : 76 34 08             L1248           tm RWStat, #RdSMErr
448/    124B : 6B 0F                                jr Z, L125c
449/    124D : FC 1A                                ld R15, #26
450/    124F : 8D 03 57                             jp Abort
451/    1252 :
452/    1252 : 08 E4                L1252           ld R0, R4
453/    1254 : 50 E4                                pop R4
454/    1256 : 66 E0 80                             tcm R0, #080h
455/    1259 : 8D 02 F6             L1259           jp Bank_Ret
456/    125C :
457/    125C : 76 34 20             L125c           tm RWStat, #RdNoHdrFnd
```

```
458/    125F : 6B 0D                              jr Z, L126e
459/    1261 : 76 35 20                           tm DiskStat, #Wr_Op
460/    1264 : 6B 04                              jr Z, L126a
461/    1266 : 4C 8A                              ld R4, #08Ah
462/    1268 : 8B E8                              jr L1252
463/    126A :
464/    126A : 4C 88                 L126a        ld R4, #88h
465/    126C : 8B E4                              jr L1252
466/    126E :
467/    126E : 76 35 20              L126e        tm DiskStat, #Wr_Op
468/    1271 : EB DF                              jr NZ, L1252
469/    1273 : 08 E9                 L1273        ld R0, R9
470/    1275 : 56 E0 0F                           and R0, #00Fh
471/    1278 : A6 E0 03                           cp R0, #3
472/    127B : 2B 09                              jr LE, L1286
473/    127D : A6 E0 0A                           cp R0, #00Ah
474/    1280 : 6B 08                              jr Z, L128a
475/    1282 : 4C 82                              ld R4, #082h
476/    1284 : 8B CC                              jr L1252
477/    1286 :
478/    1286 : 4C 86                 L1286        ld R4, #086h
479/    1288 : 8B C8                              jr L1252
480/    128A :
481/    128A : 4C 84                 L128a        ld R4, #084h
482/    128C : 8B C4                              jr L1252
483/    128E :
484/    128E :
485/    128E :
486/    128E :                       ;*****************************************************************
487/    128E :                       ; Module Spare.Assem
488/    128E :                       ;
489/    128E :                       ; This module contains all the routines that pertain
490/    128E :                       ; to the management of the spare table.
491/    128E :                       ;
492/    128E :                       ;       PROCEDURE SprBlock( SpareType : BIT {R10/bit 7} )
493/    128E :                       ;       PROCEDURE SpareBlock( BlockIsSpare : BOOLEAN {R10/bit 4} )
494/    128E :                       ;
495/    128E :                       ;*****************************************************************
496/    128E :
497/    128E :                       ;*****************************************************************
498/    128E :                       ;
499/    128E :                       ; Procedure: SprBlock {spare a block}
500/    128E :                       ;
501/    128E :                       ;  This procedure is responsible for relocating a logical
502/    128E :                       ;  block (note that the block type can be USER or SPARETABLE)
503/    128E :                       ;  from either the user data area or the spare area to
504/    128E :                       ;  some location within the spare area.
505/    128E :                       ;
506/    128E :                       ;  This procedure is capable of gobbling up ALL available spare
507/    128E :                       ;  block space and will ABORT if no space is available.
508/    128E :                       ;
509/    128E :                       ; Inputs: SpareType: BIT {R10/Bit 4}
510/    128E :                       ;
511/    128E :                       ; Outputs: none
512/    128E :                       ;
513/    128E :                       ; Algorithm:
514/    128E :                       ;
515/    128E :                       ;  Begin
516/    128E :                       ;    If SpareType=Spare
517/    128E :                       ;      Then
518/    128E :                       ;        If BlkStat.SpareCode=BadBlock
519/    128E :                       ;          Then
520/    128E :                       ;            DeleteSpare
521/    128E :                       ;            SpareCount(Dec_BadCnt)
522/    128E :                       ;        BlockMove(WBuffer1, Buffer2)
523/    128E :                       ;        If not(WrVer_Common)
524/    128E :                       ;          Then SpareBlock(SpareType)
525/    128E :                       ;      Else SpareBlock(BadBlockType)
526/    128E :                       ;    UpDate_SprTbl
527/    128E :                       ;    BlockMove(Buffer2, RBuffer1)
528/    128E :                       ;  End
529/    128E :                       ;
530/    128E :                       ;*****************************************************************
531/    128E :
532/    128E : 76 EA 10              SprBlock     tm R10, #Spare          ; If SpareType=Spare
533/    1291 : 6B 65                              jr Z, SprBlk_1
534/    1293 :                       ;
535/    1293 : 76 3E 20                           tm 3Eh, #B_Block
536/    1296 : 6B 10                              jr Z, SprBlk_2
537/    1298 :                       ;
538/    1298 : 2C 24                              ld R2, #DeleteSpare /256
539/    129A : 3C 7C                              ld R3, #DeleteSpare #256
540/    129C : D6 02 CB                           call Bank_Call
541/    129F : 0C 02                              ld R0, #Dec_BadCnt
542/    12A1 : 2C 23                              ld R2, #SpareCount /256
543/    12A3 : 3C 42                              ld R3, #SpareCount #256
544/    12A5 : D6 02 CB                           call Bank_Call
545/    12A8 : 2C 25                 SprBlk_2     ld R2, #ReSeek /256
546/    12AA : 3C 31                              ld R3, #ReSeek #256
547/    12AC : D6 02 CB                           call Bank_Call
548/    12AF : 4C 0A                              ld R4, #10              ; bang on this block for a while
549/    12B1 : 6C 00                              ld R6, #0               ; init error count
550/    12B3 : 2C 20                 Spr_WrVer    ld R2, #Buf2_To_WrBuf /256
551/    12B5 : 3C 61                              ld R3, #Buf2_To_WrBuf #256
552/    12B7 : D6 02 CB                           call Bank_Call
553/    12BA : 46 35 20                           or DiskStat, #Wr_Op
554/    12BD : D6 05 22                           call Ext_Push
555/    12C0 : D6 11 D3                           call RW_Common
556/    12C3 : 70 FC                              push FLAGS
557/    12C5 : D6 05 4F                           call Ext_Pop
558/    12C8 : 50 FC                              pop FLAGS
559/    12CA : 6B 2C                              jr Z, SprBlk_1          ; check for write error
560/    12CC : 56 35 DF                           and DiskStat, #0FFh-Wr_Op
561/    12CF : D6 05 22                           call Ext_Push
562/    12D2 : D6 11 D3                           call RW_Common
563/    12D5 : 70 FC                              push FLAGS
564/    12D7 : D6 05 4F                           call Ext_Pop
565/    12DA : 50 FC                              pop FLAGS
566/    12DC : EB 0E                              jr NZ, SprWrV_1
567/    12DE : 08 E9                              ld R0, R9               ; get read error count
568/    12E0 : 56 E0 0F                           and R0, #00Fh           ; mask off status info
569/    12E3 : 6B 13                              jr Z, SprBlk_1          ; check for header failure
570/    12E5 : A6 E0 03                           cp R0, #SprThresh
571/    12E8 : AB 0E                              jr GT, SprBlk_1         ; spare the block if over threshold
572/    12EA : 02 60                              add  R6, R0             ; bump cumulative error count
573/    12EC : 4A C5                 SprWrV_1     djnz R4, Spr_WrVer
574/    12EE : A6 E6 03                           cp R6, #SprThresh       ; take a percentage of total reads
575/    12F1 : AB 05                              jr GT, SprBlk_1
```

```
576/    12F3 :                      ;
577/    12F3 : 76 3E 20                         tm 3Eh, #B_Block
578/    12F6 : EB 03                            jr NZ, SprBlk_3
579/    12F8 : D6 13 0A             SprBlk_1    call SpareBlock
580/    12FB :                      ;
581/    12FB : 2C 22                SprBlk_3    ld R2, #UpDate_SprTbl /256
582/    12FD : 3C 80                            ld R3, #UpDate_SprTbl #256
583/    12FF : D6 02 CB                         call Bank_Call
584/    1302 : 2C 20                            ld R2, #Buf2_To_RBuf /256
585/    1304 : 3C 4C                            ld R3, #Buf2_To_RBuf #256
586/    1306 : D6 02 CB                         call Bank_Call
587/    1309 : AF                               ret
588/    130A :
589/    130A :
590/    130A :                      ;*****************************************************************
591/    130A :                      ;
592/    130A :                      ; Procedure: SpareBlock
593/    130A :                      ;
594/    130A :                      ;  This procedure performs the actual sparing.
595/    130A :                      ;
596/    130A :                      ; Inputs: SpareType: BIT {R10/Bit 4}
597/    130A :                      ;
598/    130A :                      ; Outputs: none
599/    130A :                      ;
600/    130A :                      ; Local Variables: SparingASpare: BOOLEAN {R9}
601/    130A :                      ;                  SpareLocation: BYTE {R4}
602/    130A :                      ;                  Error:         BOOLEAN {R5}
603/    130A :                      ;
604/    130A :                      ; Algorithm:
605/    130A :                      ;
606/    130A :                      ;  Begin
607/    130A :                      ;    If BlkStat.SpareCode=BadBlock
608/    130A :                      ;      Then
609/    130A :                      ;        AddSpare(GetNewSpare(Load_Logical), Load_Logical, BadBlock)
610/    130A :                      ;        SpareCount(Inc_BadCnt)
611/    130A :                      ;      Else
612/    130A :                      ;        Repeat
613/    130A :                      ;          If BlkStat.SpareCode=Spare
614/    130A :                      ;            Then
615/    130A :                      ;              Location:=CnvrtLogical(Load_Logical)
616/    130A :                      ;              Ptr:=Get_Ptr(Location)
617/    130A :                      ;              Ptr^.Useable:=false
618/    130A :                      ;          SpareCount(Inc_SprCnt)
619/    130A :                      ;          BlkStat.SpareCode:=SpareBlock
620/    130A :                      ;          Location:=GetNewSpare(Load_Logical)
621/    130A :                      ;          AddSpare(Location, Load_Logical, SpareBlock)
622/    130A :                      ;          Seek_Type:=Access_Offset
623/    130A :                      ;          Seek(Get_Cyl_H_S(MulR0_m(Location)))
624/    130A :                      ;        Until WrVer_Common
625/    130A :                      ;  End
626/    130A :                      ;
627/    130A :                      ;*****************************************************************
628/    130A :
629/    130A : 0C 01                SpareBlock      ld R0, #1               ; byte 1
630/    130C : 1C 04                                ld R1, #Stat_Spare
631/    130E : D6 02 9B                             call L029b              ; SetStatus
632/    1311 : 76 EA 10                             tm R10, #Spare
633/    1314 : EB 25                                jr NZ, S_Blk_Rpt
634/    1316 :                      ;
635/    1316 : 76 3E 20                             tm 3Eh, #B_Block
636/    1319 : EB 2D                                jr NZ, S_Blk_End
637/    131B :                      ;
638/    131B : D6 04 18                             call Load_Logical
639/    131E : 2C 23                                ld R2, #GetNewSpare /256
640/    1320 : 3C 90                                ld R3, #GetNewSpare #256
641/    1322 : D6 02 CB                             call Bank_Call
642/    1325 : F8 E0                                ld R15, R0
643/    1327 : 8C 00                                ld R8, #BadBlock
644/    1329 : 2C 24                                ld R2, #AddSpare /256
645/    132B : 3C 1F                                ld R3, #AddSpare #256
646/    132D : D6 02 CB                             call Bank_Call
647/    1330 : 0C 01                                ld R0, #Inc_BadCnt
648/    1332 : 2C 23                                ld R2, #SpareCount /256
649/    1334 : 3C 42                                ld R3, #SpareCount #256
650/    1336 : D6 02 CB                             call Bank_Call
651/    1339 : 8B 0D                                jr S_Blk_End
652/    133B :                      ;
653/    133B : D6 04 18             S_Blk_Rpt       call Load_Logical
654/    133E : D6 13 4B                             call Spr_Enter
655/    1341 : EB 05                                jr NZ, S_Blk_End
656/    1343 : A6 E0 86                             cp R0, #Error+Ex_ReadErr        ; check for sparing threshold
657/    1346 : EB F3                                jr NZ, S_Blk_Rpt
658/    1348 : 8D 02 F6             S_Blk_End       jp Bank_Ret
659/    134B :                      ;
660/    134B : 76 3E 40             Spr_Enter       tm 3Eh, #S_Block
661/    134E : 6B 16                                jr Z, S_Blk_New
662/    1350 : D6 1B 04                             call CnvrtLogical
663/    1353 : EB 05                                jr NZ, S_Blk_Unuse
664/    1355 : FC 0C                                ld R15, #12
665/    1357 : 8D 03 57                             jp Abort
666/    135A :                      ;
667/    135A : 08 E1                S_Blk_Unuse     ld R0, R1
668/    135C : D6 15 1F                             call Get_Ptr
669/    135F : 82 02                                lde R0, @RR2            ; element useable:=false
670/    1361 : 56 E0 DF                             and R0, #0FFh-Useable
671/    1364 : 92 02                                lde @RR2, R0
672/    1366 :                      ;
673/    1366 : 0C 00                S_Blk_New       ld R0, #Inc_SprCnt
674/    1368 : 2C 23                                ld R2, #SpareCount /256
675/    136A : 3C 42                                ld R3, #SpareCount #256
676/    136C : D6 02 CB                             call Bank_Call
677/    136F : 56 3E 9F                             and 3Eh, #0FFh-B_Block-S_Block  ; mask out the SpareCode stuff
678/    1372 : 46 3E 40                             or 3Eh, #S_Block
679/    1375 : D6 04 18                             call Load_Logical
680/    1378 : 2C 23                                ld R2, #GetNewSpare /256
681/    137A : 3C 90                                ld R3, #GetNewSpare #256
682/    137C : D6 02 CB                             call Bank_Call
683/    137F : F8 E0                                ld R15, R0
684/    1381 : 8C 10                                ld R8, #Spare
685/    1383 : 2C 24                                ld R2, #AddSpare /256
686/    1385 : 3C 1F                                ld R3, #AddSpare #256
687/    1387 : D6 02 CB                             call Bank_Call
688/    138A : 46 35 08                             or DiskStat, #Offset_On
689/    138D : D6 04 18                             call Load_Logical
690/    1390 : D6 1C 24                             call OverLap
691/    1393 : 76 3E 40                             tm 3Eh, #S_Block
692/    1396 : ED 13 9E                             jp NZ, S_Blk_New1
693/    1399 : F8 33                                ld R15,SlfTst_Result
```

```
694/    139B : 8D 03 57                         jp Abort
695/    139E :                          ;
696/    139E : 76 35 04                 S_Blk_New1      tm DiskStat, #User_Type
697/    13A1 : EB 0D                                    jr NZ, Spr_LdBuf2
698/    13A3 : 2C 22                                    ld R2, #SprChkSum /256  ; calculate new checkbyte
699/    13A5 : 3C 24                                    ld R3, #SprChkSum #256
700/    13A7 : D6 02 CB                                 call Bank_Call
701/    13AA : 2C 20                                    ld R2, #Spr_To_WrBuf /256
702/    13AC : 3C 9D                                    ld R3, #Spr_To_WrBuf #256
703/    13AE : 8B 04                                    jr Spr_LdWrBuf
704/    13B0 : 2C 20                    Spr_LdBuf2      ld R2, #Buf2_To_WrBuf /256
705/    13B2 : 3C 61                                    ld R3, #Buf2_To_WrBuf #256
706/    13B4 : D6 02 CB                 Spr_LdWrBuf     call Bank_Call
707/    13B7 : D6 11 A4                                 call WrVer_Common
708/    13BA : 8D 02 F6                                 jp Bank_Ret
709/    13BD :
710/    13BD :
711/    13BD :
712/    13BD :                          ;*****************************************************************
713/    13BD :                          ; Module Data.X.Assem
714/    13BD :                          ;
715/    13BD :                          ; Data Exception Handling
716/    13BD :                          ;
717/    13BD :                          ;       PROCEDURE: Data_Ex_Handler( ErrorCode : 7 BITS {R0/Bits 6:0} )
718/    13BD :                          ;
719/    13BD :                          ;*****************************************************************
720/    13BD :
721/    13BD :                          ;*****************************************************************
722/    13BD :                          ;
723/    13BD :                          ; Procedure: Data_Ex_Handler  {data exception handler}
724/    13BD :                          ;
725/    13BD :                          ;  This procedure is responsible for cleaning up after some sort
726/    13BD :                          ;  of data error (spareing, bad-blocking, etc). It is
727/    13BD :                          ;  assumed that if spareing is going to occur that correct data must
728/    13BD :                          ;  be residing in Buffer2.
729/    13BD :                          ;
730/    13BD :                          ; Inputs: ErrorCode: 7 BITS {R0/Bits 6:0}
731/    13BD :                          ;
732/    13BD :                          ; Outputs: none
733/    13BD :                          ;
734/    13BD :                          ; Algorithm:
735/    13BD :                          ;
736/    13BD :                          ;  Begin
737/    13BD :                          ;    If not(DiskStatus.Wr_Op) Then SetStatus(ReadError)
738/    13BD :                          ;    Case ErrorCode Of
739/    13BD :                          ;      0: SetStatus(OperationFailed)
740/    13BD :                          ;      2: Spare(Spare)
741/    13BD :                          ;      4: Spare(BadBlock)
742/    13BD :                          ;         SetStatus(OperationFailed)
743/    13BD :                          ;      6: SetStatus(ErrorCount)
744/    13BD :                          ;      8: SetStatus(NoHeaderFound)
745/    13BD :                          ;         Spare(Spare)
746/    13BD :                          ;      Otherwise Abort
747/    13BD :                          ;  End
748/    13BD :                          ;
749/    13BD :                          ;*****************************************************************
750/    13BD :
751/    13BD : 70 E0                    Data_Ex_Handler push R0                 ; save error code
752/    13BF : D6 05 22                                 call Ext_Push           ; save state
753/    13C2 : 76 35 20                                 tm DiskStat, #Wr_Op
754/    13C5 : EB 03                                    jr NZ, Data_Ex_RdErr
755/    13C7 : D6 03 AD                                 call SS_ReadErr         ; SetStatus ???
756/    13CA : 50 E1                    Data_Ex_RdErr   pop R1                  ; get error code back
757/    13CC : 56 E1 7F                                 and R1, #07Fh           ; mask off error flag
758/    13CF : 76 E1 01                                 tm  R1, #1              ; see if error code is odd
759/    13D2 : EB 05                                    jr NZ, Data_Ex_Abort
760/    13D4 : A6 E1 0A                                 cp R1, #Ex_Case_Max     ; bounds check
761/    13D7 : 2B 05                                    jr LE, Data_Ex_Cmnd
762/    13D9 : FC 0F                    Data_Ex_Abort   ld R15, #15
763/    13DB : 8D 03 57                                 jp Abort
764/    13DE :                          ;
765/    13DE : 2C 13                    Data_Ex_Cmnd    ld R2, #Ex_Jp_Tbl /256
766/    13E0 : 3C EF                                    ld R3, #Ex_Jp_Tbl #256
767/    13E2 : 02 31                                    add R3, R1              ; add offset
768/    13E4 : 16 E2 00                                 adc R2, #0
769/    13E7 : C2 E2                                    ldc R14, @RR2           ; get routine to execute
770/    13E9 : A0 E2                                    incw RR2
771/    13EB : C2 F2                                    ldc R15, @RR2
772/    13ED : 30 EE                                    jp @RR14                ; go to routine
773/    13EF :
774/    13EF : 13 FB                    Ex_Jp_Tbl       DW X_Undeter
775/    13F1 : 14 00                                    DW X_Spare
776/    13F3 : 14 07                                    DW X_BadBlock
777/    13F5 : 14 1B                                    DW X_ReadErr
778/    13F7 : 14 2B                                    DW X_HdrBadBlock
779/    13F9 : 14 5A                                    DW X_HdrSpare
780/    13FB :
781/    13FB : D6 03 9F                 X_Undeter       call SS_OpFail
782/    13FE : 8B 25                                    jr Except_Return
783/    1400 :                          ;
784/    1400 : AC 10                    X_Spare         ld R10, #Spare
785/    1402 : D6 12 8E                 X_Spr_Call      call SprBlock
786/    1405 : 8B 1E                                    jr Except_Return
787/    1407 :                          ;
788/    1407 : 4C 02                    X_BadBlock      ld R4, #2               ; retry 2 times
789/    1409 : 70 E4                    X_BB_Lp         push R4
790/    140B : D6 14 7B                                 call Bad_Recover
791/    140E : 50 E4                                    pop R4
792/    1410 : EB EE                                    jr NZ, X_Spare
793/    1412 : 4A F5                                    djnz R4, X_BB_Lp
794/    1414 : D6 03 9F                                 call SS_OpFail
795/    1417 : AC 00                                    ld R10, #BadBlock
796/    1419 : 8B E7                                    jr X_Spr_Call
797/    141B :                          ;
798/    141B : D6 03 AD                 X_ReadErr       call SS_ReadErr
799/    141E : 2C 20                                    ld R2, #Buf2_To_RBuf /256
800/    1420 : 3C 4C                                    ld R3, #Buf2_To_RBuf #256
801/    1422 : D6 02 CB                                 call Bank_Call
802/    1425 :                          ;
803/    1425 : D6 05 4F                 Except_Return   call Ext_Pop
804/    1428 : 8D 02 F6                                 jp Bank_Ret
805/    142B :                          ;
806/    142B : D6 14 7B                 X_HdrBadBlock   call Bad_Recover
807/    142E : EB D0                                    jr NZ, X_Spare
808/    1430 : D6 03 B6                                 call SS_NoHdr
809/    1433 : 76 3E 01                                 tm 3Eh, #HDA_Type       ; do we have a Rodime HDA?
810/    1436 : EB 05                                    jr NZ, X_HdrBadBk1      ;  yes -->
811/    1438 : D6 06 5B                                 call AdjustGeometry
```

```
812/    143B : 8B 02                        jr X_HdrBadBk2
813/    143D : 8B C8                X_HdrBadBk1     jr X_BadBlock
814/    143F : 2C 20                X_HdrBadBk2     ld R2, #ReadHdr /256
815/    1441 : 3C 09                                ld R3, #ReadHdr #256
816/    1443 : D6 02 CB                             call Bank_Call
817/    1446 : 6B 09                                jr Z, X_Hdr_Crct
818/    1448 : 2C 20                                ld R2, #RBuf_To_Buf2 /256
819/    144A : 3C 8A                                ld R3, #RBuf_To_Buf2 #256
820/    144C : D6 02 CB                             call Bank_Call
821/    144F : 8B AF                                jr X_Spare
822/    1451 :                      ;
823/    1451 : 2C 2C                X_Hdr_Crct      ld R2, #ECC /256
824/    1453 : 3C 03                                ld R3, #ECC #256
825/    1455 : D6 02 CB                             call Bank_Call
826/    1458 : 6B AD                                jr Z, X_BadBlock
827/    145A :                      ;
828/    145A : D6 03 B6             X_HdrSpare      call SS_NoHdr
829/    145D : 8B A1                                jr X_Spare
830/    145F :
831/    145F :
832/    145F :                      ; this belongs to D_ReadHdr and is called with a Rodime HDA installed
833/    145F : 44 3D 3D             D_RdH_3         or Sector,Sector
834/    1462 : 6B 05                                jr Z, L1469
835/    1464 : D6 06 5B                             call AdjustGeometry
836/    1467 : 8B 11                                jr L147a
837/    1469 : E6 3D 1F             L1469           ld Sector, #31
838/    146C : D6 06 5B                             call AdjustGeometry
839/    146F : 56 FA FB                             and IRQ, #0FFh-Irq_Sector       ; clear any old sector marks
840/    1472 : 2C 01                                ld R2, #1
841/    1474 : D6 06 9D                             call L069d
842/    1477 : E6 3D 00                             ld Sector, #0
843/    147A : AF                   L147a           ret
844/    147B :
845/    147B :
846/    147B :                      ;*****************************************************************
847/    147B :                      ;
848/    147B :                      ; Function: Bad_Recover
849/    147B :                      ;
850/    147B :                      ;  This function is called when a block can not be read, either the
851/    147B :                      ;  header can not be found or there is a hard data error. The purpose
852/    147B :                      ;  here is to use the auto offset capabilities of the servo controller
853/    147B :                      ;  and reread the block.
854/    147B :                      ;
855/    147B :                      ; Inputs: none
856/    147B :                      ;
857/    147B :                      ; Outputs: Bad_Recover: BOOLEAN {zero flag set if failure on retry}
858/    147B :                      ;
859/    147B :                      ;*****************************************************************
860/    147B :
861/    147B : 76 35 01             Bad_Recover     tm DiskStat, #Offset_Set        ; check for auto_offset
862/    147E : EB 03                                jr NZ, B_Rcvr_Read
863/    1480 : D6 25 31                             call ReSeek
864/    1483 : 56 35 DF             B_Rcvr_Read     and DiskStat, #0FFh-Wr_Op
865/    1486 : D6 11 D3                             call RW_Common
866/    1489 : EB 20                                jr NZ, Bad_Rcvr_Ret
867/    148B : A6 E0 86                             cp R0, #Error+Ex_ReadErr
868/    148E : 6B 10                                jr Z, Bad_Rc_Spr
869/    1490 : A6 E0 84                             cp R0, #Error+Ex_BadBlock
870/    1493 : 6B 0F                                jr Z, Bad_Rc_Ecc
871/    1495 : A6 E0 82                             cp R0, #Error+Ex_SprBlock
872/    1498 : 6B 06                                jr Z, Bad_Rc_Spr
873/    149A : 0C 00                                ld R0, #0
874/    149C : 42 00                Bad_Rc_Set      or R0, R0
875/    149E : 8B 0B                                jr Bad_Rcvr_Ret
876/    14A0 : 0C 01                Bad_Rc_Spr      ld R0, #1
877/    14A2 : 8B F8                                jr Bad_Rc_Set
878/    14A4 : 2C 2C                Bad_Rc_Ecc      ld R2, #ECC /256
879/    14A6 : 3C 03                                ld R3, #ECC #256
880/    14A8 : D6 02 CB                             call Bank_Call
881/    14AB : AF                   Bad_Rcvr_Ret    ret
882/    14AC :
883/    14AC :
884/    14AC :
885/    14AC :                      ;*****************************************************************
886/    14AC :                      ; Module SprUtils.Assem
887/    14AC :                      ;
888/    14AC :                      ; This module contains all the primitive utility procedures used by
889/    14AC :                      ; the module: Spare
890/    14AC :                      ;
891/    14AC :                      ;       FUNCTION Get_HeadPtr( LogicalBlock : 3 BYTES {R12:14} ) :
892/    14AC :                      ;               BOOLEAN
893/    14AC :                      ;               HeadPtr : BYTE {R0}
894/    14AC :                      ;               ArrayPtr : PTR {RR2}
895/    14AC :                      ;               FUNCTION Get_EoList( Start_Ptr : BYTE {R0} ) :
896/    14AC :                      ;               SparePtr : BYTE {R0}
897/    14AC :                      ;               ElementStatus : BYTE {R1}
898/    14AC :                      ;               Ptr : PTR {RR2}
899/    14AC :                      ;               PreviousPtr : PTR {ScrReg0:1}
900/    14AC :                      ;       FUNCTION TSC_BitMap( TestBit : BOOLEAN {R12/bit 7}
901/    14AC :                      ;               SetBit : BOOLEAN {R12/bit 6}
902/    14AC :                      ;               ClearBit : BOOLEAN {R12/bit 5}
903/    14AC :                      ;               Location : BYTE {R13} ) : BOOLEAN
904/    14AC :                      ;       FUNCTION Get_Ptr( SparePtr : BYTE {R0} ) : Ptr : PTR {R2}
905/    14AC :                      ;       FUNCTION Get_Spr_Code : 2 BITS {R0/Bits 1:0}
906/    14AC :                      ;
907/    14AC :                      ;*****************************************************************
908/    14AC :
909/    14AC :                      ;*****************************************************************
910/    14AC :                      ;
911/    14AC :                      ; Function: Get_HeadPtr
912/    14AC :                      ;
913/    14AC :                      ;  This function searches the SegPtrArray (that array for the spare
914/    14AC :                      ;  table that contains the various head ptrs for the spare table) and
915/    14AC :                      ;  returns the HeadPtr that corresponds to the list whose elements all
916/    14AC :                      ;  have the first 7 bits of the (passed in) LogicalBlockNumber.
917/    14AC :                      ;
918/    14AC :                      ; Inputs: LogicalBlockNumber : 3 BYTES {R12:14}
919/    14AC :                      ;
920/    14AC :                      ; Outputs: Get_HeadPtr : BOOLEAN {zero is set if HeadPtr.Nil is true}
921/    14AC :                      ;          HeadPtr     : BYTE {R0}
922/    14AC :                      ;          ArrayPtr    : Ptr {RR2} {ptr to location in array}
923/    14AC :                      ;
924/    14AC :                      ; Algorithm:
925/    14AC :                      ;
926/    14AC :                      ;  BEGIN
927/    14AC :                      ;    HeadPtr :=  SegPtrArray[ LogicalBlockNumber/bits 10:16 ]
928/    14AC :                      ;    Get_HeadPtr := NOT( HeadPtr.Nil )
929/    14AC :                      ;  END
```

```
 930/    14AC :                     ;
 931/    14AC :                     ;*********************************************************************
 932/    14AC :
 933/    14AC : 18 ED               Get_HeadPtr     ld R1, R13              ; bits 8:15 of logical block #
 934/    14AE : 56 E1 FC                            and R1, #0FCh
 935/    14B1 : E0 E1                               rr R1                   ; divide by 4
 936/    14B3 : E0 E1                               rr R1
 937/    14B5 : 2C 14                               ld R2, #SegPtrArray /256
 938/    14B7 : 3C 9F                               ld R3, #SegPtrArray #256
 939/    14B9 : 02 31                               add R3, R1              ; add in offset to array
 940/    14BB : 16 E2 00                            adc R2, #0
 941/    14BE : 82 02                               lde R0, @RR2            ; load HeadPtr
 942/    14C0 : 66 E0 80                            tcm R0, #Nil            ; test for Nil Ptr
 943/    14C3 : 8D 02 F6                            jp Bank_Ret
 944/    14C6 :
 945/    14C6 :
 946/    14C6 :                     ;*********************************************************************
 947/    14C6 :                     ;
 948/    14C6 :                     ; Function: Get_EoList (Get End-Of-List)
 949/    14C6 :                     ;
 950/    14C6 :                     ;  This function takes a starting ptr as an input parameter and
 951/    14C6 :                     ;  follows the list specified by the ptr until the list terminates.
 952/    14C6 :                     ;  A pointer to the last element of the array is returned to the
 953/    14C6 :                     ;  caller.
 954/    14C6 :                     ;
 955/    14C6 :                     ; Inputs: Start_Ptr : BYTE {R0}
 956/    14C6 :                     ;
 957/    14C6 :                     ; Outputs: Ptr : PTR {RR2}
 958/    14C6 :                     ;          SparePtr : BYTE {R0}
 959/    14C6 :                     ;          Element.Status : BYTE {R1}
 960/    14C6 :                     ;          PreviousPtr : PTR {ScrReg6,7}
 961/    14C6 :                     ;
 962/    14C6 :                     ; Algorithm:
 963/    14C6 :                     ;
 964/    14C6 :                     ;  BEGIN
 965/    14C6 :                     ;    Ptr := Get_Ptr( Start_Ptr )
 966/    14C6 :                     ;    WHILE NOT( Get_Ptr( Temp )^.Nil ) DO
 967/    14C6 :                     ;      PreviousPtr := Get_Ptr( Temp )
 968/    14C6 :                     ;      Temp := PreviuosPtr^.Ptr
 969/    14C6 :                     ;      Ptr := Get_Ptr( Temp )
 970/    14C6 :                     ;      Element.Status := Ptr^.Status
 971/    14C6 :                     ;  END
 972/    14C6 :                     ;
 973/    14C6 :                     ;*********************************************************************
 974/    14C6 :
 975/    14C6 : 09 43               Get_EoList      ld ScrReg3, R0
 976/    14C8 : D6 15 1F                            call Get_Ptr            ; create ptr into Spare Table
 977/    14CB : 82 12                               lde R1, @RR2            ; get element status
 978/    14CD : 76 E1 80                            tm R1, #Nil             ; IF Nil THEN Exit Loop
 979/    14D0 : EB 0E                               jr NZ, Get_EL_Done
 980/    14D2 : 29 46                               ld ScrReg6, R2          ; save Ptr to element i - 1
 981/    14D4 : 39 47                               ld ScrReg7, R3
 982/    14D6 : 06 E3 03                            add R3, #3              ; get ptr to next
 983/    14D9 : 16 E2 00                            adc R2, #0
 984/    14DC : 82 02                               lde R0, @RR2
 985/    14DE : 8B E6                               jr Get_EoList
 986/    14E0 : 8D 02 F6            Get_EL_Done     jp Bank_Ret
 987/    14E3 :
 988/    14E3 :
 989/    14E3 :                     ;*********************************************************************
 990/    14E3 :                     ;
 991/    14E3 :                     ; Function: TSC_BitMap  (Test, Set, Or Clear BitMap Location)
 992/    14E3 :                     ;
 993/    14E3 :                     ;  This function performs one of 3 functions: Set a bit location in
 994/    14E3 :                     ;  the Spare Table bit map; Clear a location in the Spare Table bit
 995/    14E3 :                     ;  map; or Test a location in the spare table bit map. The location
 996/    14E3 :                     ;  to test, set, or clear is an integer value between 0 and 75.
 997/    14E3 :                     ;
 998/    14E3 :                     ; Inputs: TestBit : BOOLEAN {R12/bit 7}
 999/    14E3 :                     ;         SetBit  : BOOLEAN {R12/bit 6}
1000/    14E3 :                     ;         ClearBit : BOOLEAN {R12/bit 5}
1001/    14E3 :                     ;         Location : BYTE {R13}
1002/    14E3 :                     ;
1003/    14E3 :                     ; Outputs: TSC_BitMap : BOOLEAN {zero is set if location is zero}
1004/    14E3 :                     ;
1005/    14E3 :                     ; Global Variables Changed: SpareBitMap
1006/    14E3 :                     ;
1007/    14E3 :                     ;*********************************************************************
1008/    14E3 :
1009/    14E3 : 08 ED               TSC_BitMap      ld R0, R13
1010/    14E5 : 3C 03                               ld R3, #3               ; right hand justify the byte index
1011/    14E7 : CF                  TSC_Lp1         rcf
1012/    14E8 : C0 E0                               rrc R0
1013/    14EA : 3A FB                               djnz R3, TSC_Lp1
1014/    14EC : 2C 14                               ld R2, #SpareBitMap /256
1015/    14EE : 3C E1                               ld R3, #SpareBitMap #256
1016/    14F0 : 02 30                               add R3, R0
1017/    14F2 : 16 E2 00                            adc R2, #0              ; inc second byte of address if needed
1018/    14F5 : 82 12                               lde R1, @RR2            ; load in the byte from bit map
1019/    14F7 : 56 ED 07                            and R13, #007h          ; mask off all but MOD 8 of original
1020/    14FA : 0C 80                               ld R0, #TestBitMap      ; convert index to mask
1021/    14FC : 6B 04                               jr Z, TSC_Map
1022/    14FE : E0 E0               TSC_Lp2         rr R0
1023/    1500 : DA FC                               djnz R13, TSC_Lp2
1024/    1502 : 70 E0               TSC_Map         push R0                 ; save mask
1025/    1504 : 76 EC 80                            tm R12, #TestBitMap
1026/    1507 : EB 0F                               jr NZ, TSC_End
1027/    1509 : 76 EC 40                            tm R12, #SetBitMap
1028/    150C : EB 06                               jr NZ, TSC_Set
1029/    150E : 60 E0               TSC_Clear       com R0
1030/    1510 : 52 10                               and R1, R0              ; mask out bit
1031/    1512 : 8B 02                               jr TSC_SCEnd
1032/    1514 :                     ;
1033/    1514 : 42 10               TSC_Set         or R1, R0               ; merge in bit
1034/    1516 : 92 12               TSC_SCEnd       lde @RR2, R1            ; update the bit map
1035/    1518 : 50 E0               TSC_End         pop R0
1036/    151A : 72 10                               tm R1, R0               ; test the bit
1037/    151C : 8D 02 F6                            jp Bank_Ret
1038/    151F :
1039/    151F :
1040/    151F :                     ;*********************************************************************
1041/    151F :                     ;
1042/    151F :                     ; Function: Get_Ptr
1043/    151F :                     ;
1044/    151F :                     ;  This function accepts a Spare Table ptr structure and creates a
1045/    151F :                     ;  real ptr into the Spare Table array.
1046/    151F :                     ;
1047/    151F :                     ; Inputs: SparePtr : BYTE {R0}
```

```
1048/    151F :                    ;
1049/    151F :                    ; Outputs: SParePtr : BYTE {R0} {input is unchanged}
1050/    151F :                    ;          Ptr : PTR {RR2}
1051/    151F :                    ;
1052/    151F :                    ;*****************************************************************
1053/    151F :
1054/    151F : 2C 14              Get_Ptr         ld R2, #SpareTable /256
1055/    1521 : 3C EB                              ld R3, #SpareTable #256
1056/    1523 : 70 E0                              push R0                 ; save structure ptr
1057/    1525 : 18 E0                              ld R1, R0
1058/    1527 : B0 E0                              clr R0
1059/    1529 : 02 11                              add R1, R1              ;  RR0 := 4 * R1
1060/    152B : 02 11                              add R1, R1
1061/    152D : 16 E0 00                           adc R0, #0
1062/    1530 : 02 31                              add R3, R1              ; add offset into table
1063/    1532 : 12 20                              adc R2, R0
1064/    1534 : 50 E0                              pop R0                  ; get original structure ptr back
1065/    1536 : 8D 02 F6                           jp Bank_Ret
1066/    1539 :
1067/    1539 :
1068/    1539 :
1069/    1539 :                    ;*****************************************************************
1070/    1539 :                    ; Module Cmnd0.Assem   {Command Driver 0}
1071/    1539 :                    ;
1072/    1539 :                    ;  This module controls the way in which commands received by the
1073/    1539 :                    ;  Host are executed. It's main responsibility is to determine whether
1074/    1539 :                    ;  a command string is protected by a check byte (the original Profile
1075/    1539 :                    ;  commands are not) and then decode the command and pass control over
1076/    1539 :                    ;  to the correct driver routine to execute the command. All exception
1077/    1539 :                    ;  handling at this level is controlled by the individual command
1078/    1539 :                    ;  routines.
1079/    1539 :                    ;
1080/    1539 :                    ;       PROCEDURE Read_ID
1081/    1539 :                    ;       PROCEDURE Read_SprTbl
1082/    1539 :                    ;       PROCEDURE Pro_Write
1083/    1539 :                    ;       FUNCTION RW_Common : BOOLEAN
1084/    1539 :                    ;       PROCEDURE Pro_WrVer
1085/    1539 :                    ;       FUNCTION WrVer_Common : BOOLEAN
1086/    1539 :                    ;
1087/    1539 :                    ;*****************************************************************
1088/    1539 :
1089/    1539 :                    ;*****************************************************************
1090/    1539 :                    ;
1091/    1539 :                    ; Procedure: Read_ID
1092/    1539 :                    ;
1093/    1539 :                    ;  This procedure is responsible for letting the host system
1094/    1539 :                    ;  know what type of device it is talking to
1095/    1539 :                    ;
1096/    1539 :                    ; Inputs: none
1097/    1539 :                    ;
1098/    1539 :                    ; Outputs: none
1099/    1539 :                    ;
1100/    1539 :                    ; Algorithm:
1101/    1539 :                    ;
1102/    1539 :                    ;  Begin
1103/    1539 :                    ;    ClrNormStat
1104/    1539 :                    ;    ZeroRdBuf
1105/    1539 :                    ;    ID.DeviceName:=DeviceName
1106/    1539 :                    ;    ID.DeviceNumber:=DeviceNumber
1107/    1539 :                    ;    ID.Revision:=RevisionNumber
1108/    1539 :                    ;    ID.Capacity:=Capacity
1109/    1539 :                    ;    ID.BlockSize:=BlockSize
1110/    1539 :                    ;    Move4(StatusArray, Status)
1111/    1539 :                    ;    Goto Rd_Leave
1112/    1539 :                    ;  End
1113/    1539 :                    ;
1114/    1539 :                    ;*****************************************************************
1115/    1539 :
1116/    1539 : CC 10              Read_ID         ld R12, #RBuffer1 /256
1117/    153B : DC 19                              ld R13, #RBuffer1 #256
1118/    153D : EC 10                              ld R14, #DeviceParams /256      ; ID block
1119/    153F : FC B7                              ld R15, #DeviceParams #256
1120/    1541 : 1C 17                              ld R1, #Dev_Parm_Length ; copy 23 bytes
1121/    1543 : C2 0E              Read_ID_Lp1     ldc R0, @RR14
1122/    1545 : 92 0C                              lde @RR12, R0
1123/    1547 : A0 EC                              incw RR12
1124/    1549 : A0 EE                              incw RR14
1125/    154B : 1A F6                              djnz R1, Read_ID_Lp1
1126/    154D : 76 3E 01                           tm 3Eh, #HDA_Type       ; do we have a Nisha HDA?
1127/    1550 : 6B 08                              jr Z, Read_ID1          ;  yes -->
1128/    1552 : 2C 01                              ld R2, #NbrTracks_A /256        ; set geometry for Rodime
1129/    1554 : 3C 31                              ld R3, #NbrTracks_A #256
1130/    1556 : 4C 04                              ld R4, #NbrHds_A
1131/    1558 : 8B 06                              jr Read_ID2
1132/    155A : 2C 02              Read_ID1        ld R2, #NbrTracks_B /256        ; set geometry for Nisha
1133/    155C : 3C 62                              ld R3, #NbrTracks_B #256
1134/    155E : 4C 02                              ld R4, #NbrHds_B
1135/    1560 : 5C 20              Read_ID2        ld R5, #NbrSctrs        ; number of sectors
1136/    1562 : 6C 00                              ld R6, #0               ; number of possible spare locations
1137/    1564 : 7C 00                              ld R7, #0
1138/    1566 : 8C 4C                              ld R8, #76
1139/    1568 : EC 14                              ld R14, #SprCount /256
1140/    156A : FC DF                              ld R15, #SprCount #256
1141/    156C : 82 9E                              lde R9, @RR14           ; get spare count
1142/    156E : A0 EE                              incw RR14
1143/    1570 : 82 AE                              lde R10, @RR14          ; point to bad block count
1144/    1572 : 0C 12                              ld R0, #012h            ; start with R2 in Wrk_Sys
1145/    1574 : 1C 09                              ld R1, #9
1146/    1576 : 93 0C              Read_ID_Lp2     ldei @RR12, @R0
1147/    1578 : 1A FC                              djnz R1, Read_ID_Lp2
1148/    157A : 8D 11 46                           jp Rd_Leave
1149/    157D :
1150/    157D :
1151/    157D :                    ;*****************************************************************
1152/    157D :                    ;
1153/    157D :                    ; Procedure: Read_SprTbl
1154/    157D :                    ;
1155/    157D :                    ;  This procedure is responsible for passing the spare table on to
1156/    157D :                    ;  the host. Note that this information is sent in it's raw form and
1157/    157D :                    ;  that it is up to the host to correctly interpret it.
1158/    157D :                    ;
1159/    157D :                    ; Inputs: none
1160/    157D :                    ;
1161/    157D :                    ; Outputs: none
1162/    157D :                    ;
1163/    157D :                    ; Algorithm:
1164/    157D :                    ;
1165/    157D :                    ;  Begin
```

```
1166/    157D :                     ;    RBuffer1:=SpareArray
1167/    157D :                     ;    Move4(StatusArray, Status)
1168/    157D :                     ;    Clr_Bsy(StatusArray)
1169/    157D :                     ;  End
1170/    157D :                     ;
1171/    157D :                     ;*****************************************************************
1172/    157D :
1173/    157D : 2C 20               Read_SprTbl     ld R2, #Spr_To_RBuf /256
1174/    157F : 3C 70                               ld R3, #Spr_To_RBuf #256
1175/    1581 : D6 02 CB                            call Bank_Call
1176/    1584 : 8D 11 46                            jp Rd_Leave
1177/    1587 :
1178/    1587 :
1179/    1587 :                     ; Cmd 3
1180/    1587 : 2C 2C               L1587           ld R2, #L2CEC /256
1181/    1589 : 3C EC                               ld R3, #L2CEC #256
1182/    158B : D6 02 CB                            call Bank_Call
1183/    158E : 8D 11 46                            jp Rd_Leave
1184/    1591 :
1185/    1591 :
1186/    1591 :
1187/    1591 :                     ;*****************************************************************
1188/    1591 :                     ; Module Cmnd1.Assem  {continuation of the command processor module}
1189/    1591 :                     ;
1190/    1591 :                     ;       PROCEDURE D_Read
1191/    1591 :                     ;       PROCEDURE D_ReadHdr
1192/    1591 :                     ;       PROCEDURE D_Write
1193/    1591 :                     ;       PROCEDURE Read_CStatus
1194/    1591 :                     ;       PROCEDURE Read_SStatus
1195/    1591 :                     ;       PROCEDURE Send_ServoCmnd
1196/    1591 :                     ;       PROCEDURE Send_Seek
1197/    1591 :                     ;       PROCEDURE Send_Restore
1198/    1591 :                     ;       PROCEDURE Send_Park
1199/    1591 :                     ;       PROCEDURE Set_Ram_Addr
1200/    1591 :                     ;       PROCEDURE Wr_SprTbl
1201/    1591 :                     ;       PROCEDURE Format
1202/    1591 :                     ;       PROCEDURE Read_Abort
1203/    1591 :                     ;       PROCEDURE D_RstSrvo
1204/    1591 :                     ;       PROCEDURE D_InitSprTbl
1205/    1591 :                     ;
1206/    1591 :                     ;*****************************************************************
1207/    1591 :
1208/    1591 :                     ;*****************************************************************
1209/    1591 :                     ;
1210/    1591 :                     ; Procedure: D_Read
1211/    1591 :                     ;
1212/    1591 :                     ;  The Diag_Read conmand is used to read the block on the disk pointed
1213/    1591 :                     ;  to by the last seek address. The form of the returned data is
1214/    1591 :                     ;  exactly the same as that of ProFile_Read or Sys_Read in that 4
1215/    1591 :                     ;  bytes of Standard_Status precede the block of data.
1216/    1591 :                     ;
1217/    1591 :                     ; Inputs: <Sector><SeqValue>
1218/    1591 :                     ;         Sector = 0..31
1219/    1591 :                     ;         SeqValue = NewSector/IncSector/Long
1220/    1591 :                     ;
1221/    1591 :                     ;   NewSector = $80 {selects 'Sector' as the sector to be operated on}
1222/    1591 :                     ;   IncSector = $40 {increments the last sector value}
1223/    1591 :                     ;   Long = $20 {if Long then the ECC syndrome will be ignored and the
1224/    1591 :                     ;                 checkbytes will be included at the end of the data}
1225/    1591 :                     ;
1226/    1591 :                     ; Outputs: none
1227/    1591 :                     ;
1228/    1591 :                     ; Algorithm:
1229/    1591 :                     ;
1230/    1591 :                     ;  Begin
1231/    1591 :                     ;    ClrNormStat
1232/    1591 :                     ;    Ack_Read(D_Read_Response)
1233/    1591 :                     ;    If not(Read_Common)
1234/    1591 :                     ;      Then Data_Ex_Handler(Read_Common.ErrorCode)
1235/    1591 :                     ;    Rd_Leave
1236/    1591 :                     ;  End
1237/    1591 :                     ;
1238/    1591 :                     ;*****************************************************************
1239/    1591 :
1240/    1591 :                     ; Cmd D 09 Diag Read
1241/    1591 : D6 15 AA            D_Read          call Get_SeqValue
1242/    1594 : 46 35 04                            or DiskStat, #User_Type
1243/    1597 : 56 35 DF                            and DiskStat, #0FFh-Wr_Op
1244/    159A : D6 11 D3                            call RW_Common
1245/    159D : EB 08                               jr NZ, D_Read_End
1246/    159F : D6 03 A7                            call L03a7
1247/    15A2 : 0C 80                               ld R0, #Error+Ex_Undetermined
1248/    15A4 : D6 13 BD                            call Data_Ex_Handler
1249/    15A7 : 8D 11 46            D_Read_End      jp Rd_Leave
1250/    15AA :                     ;
1251/    15AA : D6 16 A5            Get_SeqValue    call Get_HostParms
1252/    15AD : 76 E1 80                            tm R1, #080h            ; NewSector?
1253/    15B0 : 6B 04                               jr Z, Get_SeqVal1       ;  no, try next
1254/    15B2 : 09 3D                               ld Sector, R0           ; get sector from R0
1255/    15B4 : 8B 07                               jr Get_SeqVal2
1256/    15B6 : 76 E1 40            Get_SeqVal1     tm R1, #040h            ; IncSector?
1257/    15B9 : 6B 02                               jr Z, Get_SeqVal2       ;  no, try next
1258/    15BB : 20 3D                               inc Sector              ; increment sector number
1259/    15BD : 76 E1 20            Get_SeqVal2     tm R1, #020h            ; Long?
1260/    15C0 : 6B 03                               jr Z, Get_SeqVal3       ;  no, exit
1261/    15C2 : 56 32 FB                            and Excpt_Stat, #0FFh-UseECC
1262/    15C5 : AF                  Get_SeqVal3     ret
1263/    15C6 :
1264/    15C6 :
1265/    15C6 :                     ;*****************************************************************
1266/    15C6 :                     ;
1267/    15C6 :                     ; Procedure: D_ReadHdr
1268/    15C6 :                     ;
1269/    15C6 :                     ;  This diagnostic command is used to read the header (and whatever
1270/    15C6 :                     ;  the contents may be) of the block pointed to by the last seek
1271/    15C6 :                     ;  address. The host should be cautioned that the buffer that the
1272/    15C6 :                     ;  controller points it to at the completion of the read is longer
1273/    15C6 :                     ;  than a normal read buffer by the length of the header and the
1274/    15C6 :                     ;  data gap.
1275/    15C6 :                     ;
1276/    15C6 :                     ; Inputs: none
1277/    15C6 :                     ;
1278/    15C6 :                     ; Outputs: none
1279/    15C6 :                     ;
1280/    15C6 :                     ; Algorithm:
1281/    15C6 :                     ;
1282/    15C6 :                     ;  Begin
1283/    15C6 :                     ;    ClrNormStat
```

```
1284/    15C6 :                     ;    Ack_Read(D_RdHdr_Response)
1285/    15C6 :                     ;    Sector:=LdParam1.First
1286/    15C6 :                     ;    LocateSector
1287/    15C6 :                     ;    ReadHdr
1288/    15C6 :                     ;    If ReadHdr.ServoErr Then SetStatus(Byte0, ServoErr)
1289/    15C6 :                     ;    If ReadHdr.CrcErr Then SetStatus(Byte0, ReadError)
1290/    15C6 :                     ;    Move4(RdHdrStatusArray, CStatus0)
1291/    15C6 :                     ;    Clr_Bsy(RdHdrStatusArray)
1292/    15C6 :                     ;  End
1293/    15C6 :                     ;
1294/    15C6 :                     ;*****************************************************************
1295/    15C6 :
1296/    15C6 :                     ; Cmd E 0A Diag Read Header
1297/    15C6 : D6 15 AA            D_ReadHdr       call Get_SeqValue
1298/    15C9 : 76 3E 01                            tm 3Eh, #HDA_Type       ; Rodime mechanism?
1299/    15CC : EB 05                               jr NZ, D_RdH_1          ;  yes -->
1300/    15CE : D6 06 5B                            call AdjustGeometry
1301/    15D1 : 8B 03                               jr D_RdH_2
1302/    15D3 : D6 14 5F            D_RdH_1         call D_RdH_3
1303/    15D6 : 46 35 02            D_RdH_2         or DiskStat, #SeekComplete
1304/    15D9 : 56 35 DF                            and DiskStat, #0FFh-Wr_Op
1305/    15DC : 2C 20                               ld R2, #ReadHdr /256
1306/    15DE : 3C 09                               ld R3, #ReadHdr #256
1307/    15E0 : D6 02 CB                            call Bank_Call
1308/    15E3 : EB 13                               jr NZ, D_RdH_End
1309/    15E5 : D6 03 A7                            call L03a7
1310/    15E8 : 76 34 40                            tm RWStat, #RdHSrvoErr  ; If ReadHdr.ServoErr
1311/    15EB : 6B 03                               jr Z, D_RdH_Crc
1312/    15ED : D6 03 C2                            call SetStatus
1313/    15F0 :                     ;
1314/    15F0 : 76 34 10            D_RdH_Crc       tm RWStat, #RdCrcErr
1315/    15F3 : 6B 03                               jr Z, D_RdH_End
1316/    15F5 : D6 03 AD                            call SS_ReadErr
1317/    15F8 :                     ;
1318/    15F8 : EC 10               D_RdH_End       ld R14, #RHHeader /256
1319/    15FA : FC 0C                               ld R15, #RHHeader #256
1320/    15FC : 2C 12                               ld R2, #RHEndGap /256
1321/    15FE : 3C 33                               ld R3, #RHEndGap #256
1322/    1600 : 1C 06                               ld R1, #6
1323/    1602 : 82 0E               D_RdH_Lp        lde R0, @RR14
1324/    1604 : 92 02                               lde @RR2, R0
1325/    1606 : A0 E2                               incw RR2
1326/    1608 : A0 EE                               incw RR14
1327/    160A : 1A F6                               djnz R1, D_RdH_Lp
1328/    160C : 8D 11 46                            jp Rd_Leave
1329/    160F :
1330/    160F :
1331/    160F :                     ;*****************************************************************
1332/    160F :                     ;
1333/    160F :                     ; Procedure: D_Write
1334/    160F :                     ;
1335/    160F :                     ;  This procedure, like D_Read, allows the host to perform a single
1336/    160F :                     ;  write to the last seek address. KEEP IN MIND THAT THIS ROUTINE
1337/    160F :                     ;  WILL ALLOW YOU TO WRITE ANYWHERE ON THE DISK THAT YOU CARE TO
1338/    160F :                     ;  WRITE!!!!
1339/    160F :                     ;
1340/    160F :                     ; Inputs: none
1341/    160F :                     ;
1342/    160F :                     ; Outputs: none
1343/    160F :                     ;
1344/    160F :                     ; Algorithm:
1345/    160F :                     ;
1346/    160F :                     ;  Begin
1347/    160F :                     ;    ClrNormStat
1348/    160F :                     ;    Get_Wr_Data(D_Write_Response)
1349/    160F :                     ;    If not(Write_Common)
1350/    160F :                     ;      Then Data_Ex_Handler(Wr_Common.ErrorCode)
1351/    160F :                     ;    Rd_Leave
1352/    160F :                     ;  End
1353/    160F :                     ;
1354/    160F :                     ;*****************************************************************
1355/    160F :
1356/    160F :                     ; Cmd F 0B Diag Write
1357/    160F : D6 01 EC            D_Write         call L01ec
1358/    1612 : D6 15 AA                            call Get_SeqValue
1359/    1615 : 46 35 04                            or DiskStat, #User_Type
1360/    1618 : 46 35 20                            or DiskStat, #Wr_Op
1361/    161B : D6 11 D3                            call RW_Common
1362/    161E : EB 05                               jr NZ, D_Write_End
1363/    1620 : 0C 80                               ld R0, #Error+Ex_Undetermined
1364/    1622 : D6 13 BD                            call Data_Ex_Handler
1365/    1625 : 8D 11 46            D_Write_End     jp Rd_Leave
1366/    1628 :
1367/    1628 :
1368/    1628 :                     ;*****************************************************************
1369/    1628 :                     ;
1370/    1628 :                     ; Procedure: Read_CStatus
1371/    1628 :                     ;
1372/    1628 :                     ;  This procedure allows the host to read the controller's status.
1373/    1628 :                     ;
1374/    1628 :                     ;*****************************************************************
1375/    1628 :
1376/    1628 :                     ; Cmd 5 01 Read Controller Status
1377/    1628 : EC 10               Read_CStatus    ld R14, #(RBuffer1-6) /256      ; RAM buffer 1013h
1378/    162A : FC 13                               ld R15, #(RBuffer1-6) #256
1379/    162C : 08 5F                               ld R0, Status0
1380/    162E : 18 5E                               ld R1, Status1
1381/    1630 : 2C 00                               ld R2, #0
1382/    1632 : 3C 00                               ld R3, #0
1383/    1634 : 4C 00                               ld R4, #0
1384/    1636 : 5C 00                               ld R5, #0
1385/    1638 : CC 10                               ld R12, #Wrk_Sys
1386/    163A : DC 06                               ld R13, #6              ; copy 6 bytes standard status
1387/    163C : 93 CE               Read_CStat_Lp1  ldei @RR14, @R12
1388/    163E : DA FC                               djnz R13, Read_CStat_Lp1
1389/    1640 : EC 10                               ld R14, #RBuffer1 /256  ; RAM buffer 1019h
1390/    1642 : FC 19                               ld R15, #RBuffer1 #256
1391/    1644 : 2C 14                               ld R2, #014h
1392/    1646 : 3C 7B                               ld R3, #07Bh
1393/    1648 : D6 05 01                            call L0501
1394/    164B : 2C 14                               ld R2, #014h
1395/    164D : 3C 7F                               ld R3, #07Fh
1396/    164F : D6 05 01                            call L0501
1397/    1652 : 4C 3A                               ld R4, #Cylinder        ; register 3A..3Dh
1398/    1654 : D6 16 84                            call Read_CStat_Cp2
1399/    1657 : 4C 38                               ld R4, #Cur_Cyl         ; register 38..3Bh
1400/    1659 : D6 16 84                            call Read_CStat_Cp2
1401/    165C : 08 32                               ld R0, Excpt_Stat
```

```
1402/    165E : 18 35                              ld R1, DiskStat
1403/    1660 : 28 3E                              ld R2, 3Eh
1404/    1662 : 38 34                              ld R3, RWStat
1405/    1664 : D6 16 82                           call Read_CStat_Cp1     ; system variables
1406/    1667 : D6 02 8E                           call L028e
1407/    166A : 38 E0                              ld R3, R0
1408/    166C : 08 5D                              ld R0, 5Dh
1409/    166E : 18 33                              ld R1, SlfTst_Result
1410/    1670 : 28 02                              ld R2, P2
1411/    1672 : D6 16 82                           call Read_CStat_Cp1     ; more system variables
1412/    1675 : 2C 12                              ld R2, #LastSeek_Stat /256
1413/    1677 : 3C 4F                              ld R3, #LastSeek_Stat #256
1414/    1679 : D6 05 01                           call L0501              ; last seek address
1415/    167C : 56 3E E7                L167c      and 3Eh, #0FFh-010h-008h
1416/    167F : 8D 11 4C                           jp L114c
1417/    1682 :                         ;
1418/    1682 : 4C 10                   Read_CStat_Cp1  ld R4, #Wrk_Sys         ; start at 10h
1419/    1684 : 5C 04                   Read_CStat_Cp2  ld R5, #4               ; copy four registers
1420/    1686 : 93 4E                   Read_CStat_Lp2  ldei @RR14, @R4
1421/    1688 : 5A FC                              djnz R5, Read_CStat_Lp2
1422/    168A : AF                                 ret
1423/    168B :
1424/    168B :
1425/    168B :                         ;*****************************************************************
1426/    168B :                         ;
1427/    168B :                         ; Procedure: Read_SStatus
1428/    168B :                         ;
1429/    168B :                         ;  This procedure is used by the host to examine the servo processor's
1430/    168B :                         ;  status information. The controller makes NO assumptions concerning
1431/    168B :                         ;  knowledge or meaning of the status bits, nor does the controller
1432/    168B :                         ;  interpret the status requests sent by the host. In other words,
1433/    168B :                         ;  when the host issues a request for status to the servo, it is
1434/    168B :                         ;  his responsibility to make certain that this request is meaningful
1435/    168B :                         ;
1436/    168B :                         ;  The returned status is preceded by Standard Status.
1437/    168B :                         ;
1438/    168B :                         ;*****************************************************************
1439/    168B :
1440/    168B :                         ; Cmd 6 02 Send Servo Status
1441/    168B : D6 16 A5                Read_SStatus    call Get_HostParms
1442/    168E : 2C 27                              ld R2, #ServoStatus /256
1443/    1690 : 3C A1                              ld R3, #ServoStatus #256
1444/    1692 : D6 02 CB                           call Bank_Call
1445/    1695 : D6 11 69                           call L1169
1446/    1698 : EC 10                              ld R14, #RBuffer1 /256
1447/    169A : FC 19                              ld R15, #RBuffer1 #256
1448/    169C : 2C 14                              ld R2, #SStatus0 /256
1449/    169E : 3C 90                              ld R3, #SStatus0 #256
1450/    16A0 : D6 05 01                           call L0501
1451/    16A3 : 8B D7                              jr L167c
1452/    16A5 :
1453/    16A5 :
1454/    16A5 :                         ; get host command parameters
1455/    16A5 : 31 40                   Get_HostParms   srp #Wrk_Scr
1456/    16A7 :                                 assume RP:Wrk_Scr
1457/    16A7 : EC 14                              ld R14, #(HostCmndBuf+1) /256   ; first parameter
1458/    16A9 : FC 84                              ld R15, #(HostCmndBuf+1) #256
1459/    16AB : 4C 10                              ld R4, #Wrk_Sys                 ; into R0
1460/    16AD : 5C 05                              ld R5, #5                       ; copy five bytes
1461/    16AF : 83 4E                   Get_HostP1      ldei @R4, @RR14
1462/    16B1 : 5A FC                              djnz R5, Get_HostP1
1463/    16B3 : 31 10                              srp #Wrk_Sys
1464/    16B5 :                                 assume RP:Wrk_Sys
1465/    16B5 : AF                                 ret
1466/    16B6 :
1467/    16B6 :
1468/    16B6 :                         ;*****************************************************************
1469/    16B6 :                         ;
1470/    16B6 :                         ; Procedure: Send_ServoCmnd
1471/    16B6 :                         ;
1472/    16B6 :                         ;  This procedure is used by the host to send the servo a command
1473/    16B6 :                         ;  of any type EXCEPT a status request (use Read_SStatus for that).
1474/    16B6 :                         ;  The controller will Abort the operation if this servo command
1475/    16B6 :                         ;  is sent to it.
1476/    16B6 :                         ;
1477/    16B6 :                         ;  Also, as in Read_SStatus, the host is responsible for the
1478/    16B6 :                         ;  command string that it sends to the servo - all the Widget
1479/    16B6 :                         ;  controller will do is pass along the command string.
1480/    16B6 :                         ;
1481/    16B6 :                         ;*****************************************************************
1482/    16B6 :
1483/    16B6 :                         ; Cmd 7 03 Send Servo Command
1484/    16B6 : D6 16 A5                Send_ServoCmnd  call Get_HostParms
1485/    16B9 : 42 00                              or R0, R0               ; check for Status Command
1486/    16BB : EB 05                              jr nz, S_Scmnd
1487/    16BD : FC 12                              ld R15, #18
1488/    16BF : 8D 03 57                           jp Abort
1489/    16C2 : C8 E0                   S_Scmnd         ld R12, R0
1490/    16C4 : D8 E1                              ld R13, R1
1491/    16C6 : E8 E2                              ld R14, R2
1492/    16C8 : F8 E3                              ld R15, R3
1493/    16CA : 2C 27                              ld R2, #ServoCmnd /256
1494/    16CC : 3C 69                              ld R3, #ServoCmnd #256
1495/    16CE : D6 02 CB                           call Bank_Call
1496/    16D1 : 8D 11 46                           jp Rd_Leave
1497/    16D4 :
1498/    16D4 :
1499/    16D4 :                         ;*****************************************************************
1500/    16D4 :                         ;
1501/    16D4 :                         ; Procedure: Send_Seek
1502/    16D4 :                         ;
1503/    16D4 :                         ;  This procedure allows the host to position the heads ANYWHERE
1504/    16D4 :                         ;  on the disk service that it pleases - the only consideration
1505/    16D4 :                         ;  is to be aware of the crash stop positions!
1506/    16D4 :                         ;
1507/    16D4 :                         ;  The form of the parameters passed into the controller are
1508/    16D4 :                         ;  HiCylinder, LowCylinder, Head, Sector. Keep in mind that these
1509/    16D4 :                         ;  values will be used to check for a valid header if and when
1510/    16D4 :                         ;  a state machine operation is initiated.
1511/    16D4 :                         ;
1512/    16D4 :                         ;*****************************************************************
1513/    16D4 :
1514/    16D4 :                         ; Cmd 8 04 Send Seek
1515/    16D4 : D6 16 A5                Send_Seek       call Get_HostParms      ; get parameters from command processor
1516/    16D7 : 46 35 08                           or DiskStat, #Offset_On
1517/    16DA : 76 E4 01                           tm R4, #1               ; auto offset on?
1518/    16DD : EB 03                              jr NZ, Send_Sk1         ;  no -->
1519/    16DF : 56 35 F7                           and DiskStat, #0FFh-Offset_On
```

```
1520/    16E2 : C8 E0              Send_Sk1        ld R12, R0              ; pass params to Seek
1521/    16E4 : D8 E1                              ld R13, R1
1522/    16E6 : E8 E2                              ld R14, R2
1523/    16E8 : F8 E3                              ld R15, R3
1524/    16EA : 2C 25                              ld R2, #Seek /256
1525/    16EC : 3C 47                              ld R3, #Seek #256
1526/    16EE : D6 02 CB                           call Bank_Call
1527/    16F1 : 8D 11 46                           jp Rd_Leave
1528/    16F4 :
1529/    16F4 :
1530/    16F4 :                    ;*****************************************************************
1531/    16F4 :                    ;
1532/    16F4 :                    ; Procedure: Send_Restore
1533/    16F4 :                    ;
1534/    16F4 :                    ;  This procedure allows the host to issue either a Format Recal or
1535/    16F4 :                    ;  a Data Recal to the servo without having to build a primitive
1536/    16F4 :                    ;  servo command. The value of which type of restore is to be
1537/    16F4 :                    ;  performed is passed in with the command string.
1538/    16F4 :                    ;
1539/    16F4 :                    ;*****************************************************************
1540/    16F4 :
1541/    16F4 :                    ; Cmd 9 05 Send Restore
1542/    16F4 : D6 16 A5           Send_Restore    call Get_HostParms      ; get recal type
1543/    16F7 : A6 E0 40                           cp R0, #DataRecal       ; Data Recal
1544/    16FA : 6B 0A                              jr Z, S_Restore
1545/    16FC : A6 E0 70                           cp R0, #FrmtRecal       ; Brake Release
1546/    16FF : 6B 05                              jr Z, S_Restore
1547/    1701 : FC 13                              ld R15, #19
1548/    1703 : 8D 03 57                           jp Abort
1549/    1706 : 2C 27              S_Restore       ld R2, #Restore /256
1550/    1708 : 3C CE                              ld R3, #Restore #256
1551/    170A : D6 02 CB                           call Bank_Call
1552/    170D : 8D 11 46                           jp Rd_Leave
1553/    1710 :
1554/    1710 :
1555/    1710 :                    ;*****************************************************************
1556/    1710 :                    ;
1557/    1710 :                    ; Procedure: Set_Recovery
1558/    1710 :                    ;
1559/    1710 :                    ;  This procedure is used by the host to set the global
1560/    1710 :                    ;  variable Recovery either on or off. This variable controls
1561/    1710 :                    ;  the behaviour of Widget when an exception occurs; in the
1562/    1710 :                    ;  case of Recovery = true, the controller tries everything it
1563/    1710 :                    ;  can to correct the exception and continue about it's day.
1564/    1710 :                    ;  When Recovery = false, however, the controller will do nothing
1565/    1710 :                    ;  about the problem and report the error to the host.
1566/    1710 :                    ;
1567/    1710 :                    ;*****************************************************************
1568/    1710 :
1569/    1710 :                    ; Cmd A 06 Set Recovery
1570/    1710 : D6 16 A5           Set_Recovery    call Get_HostParms
1571/    1713 : 56 3E FB                           and 3Eh, #0FFh-004h
1572/    1716 : 56 32 DF                           and Excpt_Stat, #0FFh-Buf_Damage
1573/    1719 : 76 E0 02                           tm R0, #2               ; ??? when bit 1 set
1574/    171C : 6B 03                              jr Z, L1721
1575/    171E : 46 3E 04                           or 3Eh, #004h
1576/    1721 : 56 32 7F           L1721           and Excpt_Stat, #0FFh-Recovery
1577/    1724 : 76 E0 01                           tm R0, #1               ; turn on recovery when bit 0 set
1578/    1727 : 6B 03                              jr Z, L172c
1579/    1729 : 46 32 80                           or Excpt_Stat, #Recovery
1580/    172C : 76 E0 04           L172c           tm R0, #4               ; ??? when bit 2 set
1581/    172F : EB 03                              jr NZ, L1734
1582/    1731 : 46 32 20                           or Excpt_Stat, #Buf_Damage      ; ???
1583/    1734 : 8D 11 46           L1734           jp Rd_Leave
1584/    1737 :
1585/    1737 :
1586/    1737 :                    ;*****************************************************************
1587/    1737 :                    ;
1588/    1737 :                    ; Procedure: Send_Park
1589/    1737 :                    ;
1590/    1737 :                    ;  This diagnostic command allows the host to issue a Park
1591/    1737 :                    ;  command to the drive. This command has the effect of
1592/    1737 :                    ;  sending the drive heads off the data surface.
1593/    1737 :                    ;
1594/    1737 :                    ; Inputs: none
1595/    1737 :                    ;
1596/    1737 :                    ; Outputs: none
1597/    1737 :                    ;
1598/    1737 :                    ; Algorithm:
1599/    1737 :                    ;
1600/    1737 :                    ;  Begin
1601/    1737 :                    ;    ClrNormStat
1602/    1737 :                    ;    Ack_Read(S_Park_Response)
1603/    1737 :                    ;    Park_Heads
1604/    1737 :                    ;    Rd_Leave
1605/    1737 :                    ;  End
1606/    1737 :                    ;
1607/    1737 :                    ;*****************************************************************
1608/    1737 :
1609/    1737 :                    ; Cmd C 08 Send Park
1610/    1737 : 2C 28              Send_Park       ld R2, #Park_Heads /256
1611/    1739 : 3C 5D                              ld R3, #Park_Heads #256
1612/    173B : D6 02 CB                           call Bank_Call
1613/    173E : 8D 11 46                           jp Rd_Leave
1614/    1741 :
1615/    1741 :
1616/    1741 :                    ;*****************************************************************
1617/    1741 :                    ;
1618/    1741 :                    ; Procedure: Set_AutoOffset
1619/    1741 :                    ;
1620/    1741 :                    ;  This command allows the user to set the auto offset capabilities
1621/    1741 :                    ;  of the drive without using the Servo Command function.
1622/    1741 :                    ;
1623/    1741 :                    ; Inputs: {none}
1624/    1741 :                    ;
1625/    1741 :                    ; Outputs: {none}
1626/    1741 :                    ;
1627/    1741 :                    ; Algorithm:
1628/    1741 :                    ;
1629/    1741 :                    ;  BEGIN
1630/    1741 :                    ;    Auto_Offset
1631/    1741 :                    ;  END
1632/    1741 :                    ;
1633/    1741 :                    ;*****************************************************************
1634/    1741 :
1635/    1741 :                    ; Cmd 10 0C Auto Offset
1636/    1741 : 2C 26              Set_AutoOffset  ld R2, #Auto_Offset /256
1637/    1743 : 3C DF                              ld R3, #Auto_Offset #256
```

```
1638/    1745 : D6 02 CB                        call Bank_Call
1639/    1748 : 8D 11 46                        jp Rd_Leave
1640/    174B :
1641/    174B :
1642/    174B :                    ;*****************************************************************
1643/    174B :                    ;
1644/    174B :                    ; Procedure: Wr_SprTbl
1645/    174B :                    ;
1646/    174B :                    ;  This command allows the host to update the spare table
1647/    174B :                    ;  without the controller's intervention. BE CAREFUL WITH
1648/    174B :                    ;  THIS ONE!
1649/    174B :                    ;
1650/    174B :                    ; Inputs: none
1651/    174B :                    ;
1652/    174B :                    ; Outputs: none
1653/    174B :                    ;
1654/    174B :                    ; Algorithm:
1655/    174B :                    ;
1656/    174B :                    ;  Begin
1657/    174B :                    ;    If not(Chk_PassWord(CommandString#1)) Then Abort
1658/    174B :                    ;    Ack_Read(Wr_Spr_Response)
1659/    174B :                    ;    Clr_Bsy(SpareArray)
1660/    174B :                    ;  End
1661/    174B :                    ;
1662/    174B :                    ;*****************************************************************
1663/    174B :
1664/    174B :                    ; Cmd 12 0E Write Spare Table
1665/    174B : D6 01 EC           Wr_SprTbl       call L01ec
1666/    174E : 0C 14                              ld R0, #(HostCmndBuf+1) /256
1667/    1750 : 1C 84                              ld R1, #(HostCmndBuf+1) #256
1668/    1752 : 2C 23                              ld R2, #Chk_PassWord /256
1669/    1754 : 3C 1D                              ld R3, #Chk_PassWord #256
1670/    1756 : D6 02 CB                           call Bank_Call
1671/    1759 : EB 05                              jr NZ, Wr_Spr1
1672/    175B : FC 14                              ld R15, #20
1673/    175D : 8D 03 57                           jp Abort
1674/    1760 :                    ;
1675/    1760 : 2C 20              Wr_Spr1         ld R2, #WrBuf_To_Spr /256
1676/    1762 : 3C 84                              ld R3, #WrBuf_To_Spr #256
1677/    1764 : D6 02 CB                           call Bank_Call
1678/    1767 : 2C 22                              ld R2, #UpDate_SprTbl /256
1679/    1769 : 3C 80                              ld R3, #UpDate_SprTbl #256
1680/    176B : D6 02 CB                           call Bank_Call
1681/    176E : 8D 11 46                           jp Rd_Leave
1682/    1771 :
1683/    1771 :
1684/    1771 :                    ;*****************************************************************
1685/    1771 :                    ;
1686/    1771 :                    ; Procedure: Format
1687/    1771 :                    ;
1688/    1771 :                    ;  This procedure allows the host to format the track that the heads
1689/    1771 :                    ;  are currently positioned over.
1690/    1771 :                    ;
1691/    1771 :                    ;  BE ULTRA CAREFUL HERE!!!! THIS COMMAND DESTROYS ALL DATA ON THE
1692/    1771 :                    ;  TRACK.... DATA RECOVERY WILL BE IMPOSSIBLE AFTER A TRACK HAS BEEN
1693/    1771 :                    ;  FORMATTED.
1694/    1771 :                    ;
1695/    1771 :                    ;  The two parameters that are passed into the controller by the host
1696/    1771 :                    ;  allow the host to specify the offset from index mark (in sectors)
1697/    1771 :                    ;  that Sector 0 will be located and the interleave factor.
1698/    1771 :                    ;
1699/    1771 :                    ; Inputs: none
1700/    1771 :                    ;
1701/    1771 :                    ; Outputs: none
1702/    1771 :                    ;
1703/    1771 :                    ; Algorithm:
1704/    1771 :                    ;
1705/    1771 :                    ;  Begin
1706/    1771 :                    ;    ClrNormStat
1707/    1771 :                    ;    If not(Chk_PassWord(CStatus4+3)) Then Abort
1708/    1771 :                    ;    Ack_Read(Fmt_Response)
1709/    1771 :                    ;    If not(FormatTrack(CStatus4+1, CStatus4+2)
1710/    1771 :                    ;      Then Abort
1711/    1771 :                    ;    Rd_Leave
1712/    1771 :                    ;  End
1713/    1771 :                    ;
1714/    1771 :                    ;*****************************************************************
1715/    1771 :
1716/    1771 : D6 17 82           Format          call Chk_FmtParms
1717/    1774 : 70 3D                              push Sector
1718/    1776 : 2C 2D                              ld R2, #FormatTrack /256
1719/    1778 : 3C 86                              ld R3, #FormatTrack #256
1720/    177A : D6 02 CB                           call Bank_Call
1721/    177D : 50 3D                              pop Sector
1722/    177F : 8D 11 46                           jp Rd_Leave
1723/    1782 :
1724/    1782 : 0C 14              Chk_FmtParms    ld R0, #(HostCmndBuf+1) /256
1725/    1784 : 1C 84                              ld R1, #(HostCmndBuf+1) #256
1726/    1786 : 2C 23                              ld R2, #Chk_PassWord /256
1727/    1788 : 3C 1D                              ld R3, #Chk_PassWord #256
1728/    178A : D6 02 CB                           call Bank_Call
1729/    178D : EB 05                              jr NZ, Format1
1730/    178F : FC 15                              ld R15, #21
1731/    1791 : 8D 03 57                           jp Abort
1732/    1794 : AF                 Format1         ret
1733/    1795 :
1734/    1795 :
1735/    1795 :                    ;*****************************************************************
1736/    1795 :                    ;
1737/    1795 :                    ; Procedure: Read_Abort
1738/    1795 :                    ;
1739/    1795 :                    ;  This command allows the host to get a snapshot of the controller's
1740/    1795 :                    ;  register set a the time of the last abort. If there has been no
1741/    1795 :                    ;  abort, the results of this command will be invalid.
1742/    1795 :                    ;
1743/    1795 :                    ; Inputs: none
1744/    1795 :                    ;
1745/    1795 :                    ; Outputs: none
1746/    1795 :                    ;
1747/    1795 :                    ;*****************************************************************
1748/    1795 :
1749/    1795 : 0C 12              Read_Abort      ld R0, #Abort_Stat /256
1750/    1797 : 1C 3F                              ld R1, #Abort_Stat #256
1751/    1799 : 2C 10                              ld R2, #RBuffer1 /256
1752/    179B : 3C 19                              ld R3, #RBuffer1 #256
1753/    179D : 4C 00                              ld R4, #0               ; copy 256 bytes
1754/    179F : 82 50              Rd_Abt_Lp       lde R5, @RR0
1755/    17A1 : 92 52                              lde @RR2, R5
```

```
1756/    17A3 : A0 E0                              incw RR0
1757/    17A5 : A0 E2                              incw RR2
1758/    17A7 : 4A F6                              djnz R4, Rd_Abt_Lp
1759/    17A9 : 8D 11 46                           jp Rd_Leave
1760/    17AC :
1761/    17AC :
1762/    17AC :                    ;*****************************************************************
1763/    17AC :                    ;
1764/    17AC :                    ; Procedure: D_RstSrvo
1765/    17AC :                    ;
1766/    17AC :                    ;  This procedure allows the host to reset the servo processor.
1767/    17AC :                    ;
1768/    17AC :                    ; Inputs: none
1769/    17AC :                    ;
1770/    17AC :                    ; Outputs: none
1771/    17AC :                    ;
1772/    17AC :                    ;*****************************************************************
1773/    17AC :
1774/    17AC : 2C 29              D_RstSrvo       ld R2, #ResetServo /256
1775/    17AE : 3C 22                              ld R3, #ResetServo #256
1776/    17B0 : D6 02 CB                           call Bank_Call
1777/    17B3 : 8D 11 46                           jp Rd_Leave
1778/    17B6 :
1779/    17B6 :
1780/    17B6 :                    ; Cmd 18 14 Write Track (all sectors with the same data)
1781/    17B6 : 46 35 24           D_WrTrack       or DiskStat, #Wr_Op+User_Type
1782/    17B9 : 8D 17 C2                           jp D_RdTrk1
1783/    17BC :
1784/    17BC :
1785/    17BC :                    ; Cmd 17 13 Read Track
1786/    17BC : 46 35 04           D_RdTrack       or DiskStat, #User_Type
1787/    17BF : 56 35 DF                           and DiskStat, #0FFh-Wr_Op
1788/    17C2 : D6 15 AA           D_RdTrk1        call Get_SeqValue
1789/    17C5 : D6 11 D3           D_RdTrk_Next    call RW_Common
1790/    17C8 : 6B 12                              jr Z, D_RdTrk_Err
1791/    17CA : 06 3D 02                           add Sector, #2          ; use 2:1 interleave
1792/    17CD : A6 3D 20                           cp Sector, #NbrSctrs
1793/    17D0 : 1B F3                              jr LT, D_RdTrk_Next
1794/    17D2 : A6 3D 21                           cp Sector, #NbrSctrs+1
1795/    17D5 : 6B 0A                              jr Z, D_RdTrk_Done
1796/    17D7 : E6 3D 01                           ld Sector, #1
1797/    17DA : 8B E9                              jr D_RdTrk_Next
1798/    17DC :                    ;
1799/    17DC : 0C 80              D_RdTrk_Err     ld R0, #Error+Ex_Undetermined
1800/    17DE : D6 13 BD                           call Data_Ex_Handler
1801/    17E1 : 8D 11 46           D_RdTrk_Done    jp Rd_Leave
1802/    17E4 :
1803/    17E4 :
1804/    17E4 :                    ; Cmd 19
1805/    17E4 : 2C 2E              L17e4           ld R2, #L2E44 /256
1806/    17E6 : 3C 44                              ld R3, #L2E44 #256
1807/    17E8 : D6 02 CB                           call Bank_Call
1808/    17EB : 8D 11 46                           jp Rd_Leave
1809/    17EE :
1810/    17EE :
1811/    17EE :                    ; Cmd 20
1812/    17EE : 2C 2E              L17ee           ld R2, #L2E4E /256
1813/    17F0 : 3C 4E                              ld R3, #L2E4E #256
1814/    17F2 : D6 02 CB                           call Bank_Call
1815/    17F5 : 8D 11 46                           jp Rd_Leave
1816/    17F8 :
1817/    17F8 :
1818/    17F8 :
1819/    17F8 :                    ;*****************************************************************
1820/    17F8 :                    ; Module Cmnd2.Assem  {continuation of the command processor module}
1821/    17F8 :                    ;
1822/    17F8 :                    ; This module contains all the code associated with processing
1823/    17F8 :                    ; the System commands.
1824/    17F8 :                    ;
1825/    17F8 :                    ;       PROCEDURE Sys_Read
1826/    17F8 :                    ;       PROCEDURE Sys_Write
1827/    17F8 :                    ;
1828/    17F8 :                    ;*****************************************************************
1829/    17F8 :
1830/    17F8 :                    ;*****************************************************************
1831/    17F8 :                    ;
1832/    17F8 :                    ; Procedure: Sys_Read
1833/    17F8 :                    ;
1834/    17F8 :                    ;  This procedure is the System Read command for Widget. It allows
1835/    17F8 :                    ;  the host to read up to 8 sequential blocks without sending any
1836/    17F8 :                    ;  additional commands. Between each block status is passed back to
1837/    17F8 :                    ;  the host with the data, and the host must set CMD before the next
1838/    17F8 :                    ;  block can be read. In general, the protocol is about the same as
1839/    17F8 :                    ;  in Pro_Read.
1840/    17F8 :                    ;
1841/    17F8 :                    ; Inputs: none
1842/    17F8 :                    ;
1843/    17F8 :                    ; Outputs: none
1844/    17F8 :                    ;
1845/    17F8 :                    ; Algorithm:
1846/    17F8 :                    ;
1847/    17F8 :                    ;  Begin
1848/    17F8 :                    ;    Ld_LgclBlk(Wid_Log_Offset)
1849/    17F8 :                    ;    Count:=Ld_Param1.FirstParam
1850/    17F8 :                    ;    Offset:=0
1851/    17F8 :                    ;    BlockNumber:=Load_Logical
1852/    17F8 :                    ;    ClrNormStat
1853/    17F8 :                    ;    Get_Type_Check(BlockNumber+Count)
1854/    17F8 :                    ;    Seek_Type:=Access
1855/    17F8 :                    ;    DiskStat.Wr_Op:=false
1856/    17F8 :                    ;    OverLap
1857/    17F8 :                    ;    If not(ReadBlock)
1858/    17F8 :                    ;      Then
1859/    17F8 :                    ;        SS_RdErr
1860/    17F8 :                    ;        Goto Pro_Rd_Exit
1861/    17F8 :                    ;    While Always Do
1862/    17F8 :                    ;      BlockNumber:=BlockNumber+1
1863/    17F8 :                    ;      If not(SrchCache)
1864/    17F8 :                    ;        Then OverLap
1865/    17F8 :                    ;      If not(Read_Fast)
1866/    17F8 :                    ;        Then Goto Pro_Rd_Exit
1867/    17F8 :                    ;      If Count>1
1868/    17F8 :                    ;        Then
1869/    17F8 :                    ;          Command_Pending, IBsy:=true, Response:=Sys_Rd_Resp
1870/    17F8 :                    ;          Rd_Leave2
1871/    17F8 :                    ;        Else RdLeave
1872/    17F8 :                    ;      Count:=Count-1
1873/    17F8 :                    ;  End
```

```
1874/    17F8 :                     ;
1875/    17F8 :                     ;*********************************************************************
1876/    17F8 :
1877/    17F8 :                     ; Cmd 0 System Read Block
1878/    17F8 : D6 18 59            Sys_Read        call Sys_SetUp          ; Sys_SetUp ???
1879/    17FB : 56 35 D7            Sys_Rd_Lp       and DiskStat, #0FFh-Wr_Op-Offset_On
1880/    17FE : 46 35 04                            or DiskStat, #User_Type
1881/    1801 : D6 18 B8                            call Sys_Rw_Seek            ; Sys_RW_Seek ???
1882/    1804 : 56 35 DF                            and DiskStat, #0FFh-Wr_Op
1883/    1807 : D6 11 D3                            call RW_Common
1884/    180A : 6B 15                               jr Z, Sys_RdErr
1885/    180C :                     ;
1886/    180C : 76 3E 20                            tm 3Eh, #B_Block        ; check if bad block came back!
1887/    180F : EB 15                               jr NZ, Sys_Rd_BB
1888/    1811 : 46 3E 18                            or 3Eh, #010h+008h
1889/    1814 : 4A 03                               djnz R4, Sys_Rd_More
1890/    1816 : 8D 11 46                            jp Rd_Leave
1891/    1819 : D6 01 98            Sys_Rd_More     call L0198
1892/    181C : D6 18 E1            Sys_Rd_M2       call Sys_Inc_Blk
1893/    181F : 8B DA                               jr Sys_Rd_Lp
1894/    1821 :                     ;
1895/    1821 : D6 13 BD            Sys_RdErr       call Data_Ex_Handler
1896/    1824 : 8B 05                               jr Sys_Chk_Ftl
1897/    1826 :                     ;
1898/    1826 : D6 11 4F            Sys_Rd_BB       call Pro_Rd_BB          ; mark block for sparing and set Z flag
1899/    1829 : 8B F6                               jr Sys_RdErr
1900/    182B :
1901/    182B : D6 18 4A            Sys_Chk_Ftl     call Chk_FatalStat
1902/    182E : ED 11 46                            jp NZ, Rd_Leave         ; get out if fatal error
1903/    1831 : 46 3E 18                            or 3Eh, #010h+008h
1904/    1834 : 4A 03                               djnz R4, Sys_Rd_M1
1905/    1836 : 8D 11 46                            jp Rd_Leave
1906/    1839 : D6 11 4C            Sys_Rd_M1       call L114c
1907/    183C : 1C 04                               ld R1, #4
1908/    183E : 2C 14                               ld R2, #014h
1909/    1840 : 3C 7B                               ld R3, #07Bh
1910/    1842 : D6 04 50                            call L0450
1911/    1845 : D6 04 18                            call Load_Logical
1912/    1848 : 8B D2                               jr Sys_Rd_M2
1913/    184A :                     ;
1914/    184A : 76 32 20            Chk_FatalStat   tm Excpt_Stat, #Buf_Damage      ; check if operation failed
1915/    184D : EB 09                               jr NZ, Chk_FatStat1
1916/    184F : 2C 14                               ld R2, #014h
1917/    1851 : 3C 7B                               ld R3, #07Bh
1918/    1853 : 82 02                               lde R0, @RR2            ; get 1st byte of standard stat ???
1919/    1855 : 76 E0 01                            tm R0, #1               ; check if operation failed
1920/    1858 : AF                  Chk_FatStat1    ret
1921/    1859 :                     ;
1922/    1859 : 2C 10               Sys_SetUp       ld R2, #010h
1923/    185B : 3C 20                               ld R3, #020h
1924/    185D : 82 02                               lde R0, @RR2
1925/    185F : A4 5D E0                            cp R0,>5Dh
1926/    1862 : ED 19 03                            jp NZ, Abort_61
1927/    1865 : D6 16 A5                            call Get_HostParms
1928/    1868 : 42 00                               or R0, R0               ; check for zero count
1929/    186A : 6D 18 F9                            jp Z, Abort_24
1930/    186D : 48 E0                               ld R4, R0               ; Count:=...
1931/    186F : 09 5E                               ld >Status1, R0         ; remember block count
1932/    1871 : 00 E4                               dec R4                  ; account for numbering 1..n
1933/    1873 : 2C 14                               ld R2, #(HostCmndBuf+2) /256
1934/    1875 : 3C 85                               ld R3, #(HostCmndBuf+2) #256
1935/    1877 : EC 14                               ld R14, #LogicalBlock /256
1936/    1879 : FC 80                               ld R15, #LogicalBlock #256
1937/    187B : AC 1C                               ld R10, #Wrk_Sys+12
1938/    187D : 1C 03                               ld R1, #3
1939/    187F : 82 02               Sys_Set1        lde R0, @RR2            ; copy LBA from host command
1940/    1881 : 92 0E                               lde @RR14, R0           ; into LogicalBlock and
1941/    1883 : F3 A0                               ld @R10, R0             ; R12..14
1942/    1885 : AE                                  inc R10
1943/    1886 : A0 E2                               incw RR2
1944/    1888 : A0 EE                               incw RR14
1945/    188A : 1A F3                               djnz R1, Sys_Set1
1946/    188C : 02 E4                               add R14, R4             ; set R12..14 to last block to read
1947/    188E : 16 ED 00                            adc R13, #0
1948/    1891 : 0C 00                               ld R0, #HiMaxLogical
1949/    1893 : 1C 98                               ld R1, #MidMaxLogical
1950/    1895 : 2C 34                               ld R2, #LoMaxLogical
1951/    1897 : D6 04 59                            call L0459
1952/    189A : FB 0C                               jr NC, Sys_Set2
1953/    189C :                     ;
1954/    189C : 0C 02                               ld R0, #002h            ; LBA overflow, abort
1955/    189E : 1C 40                               ld R1, #040h
1956/    18A0 : D6 02 9B                            call L029b
1957/    18A3 : FC 1C                               ld R15, #28
1958/    18A5 : 8D 03 57                            jp Abort
1959/    18A8 :                     ;
1960/    18A8 : 22 E4               Sys_Set2        sub R14, R4             ; restore first block to read
1961/    18AA : 36 ED 00                            sbc R13, #0
1962/    18AD : 46 35 04                            or DiskStat, #User_Type
1963/    18B0 : E6 0C 10                            ld 0Ch, #WBuffer1 /256
1964/    18B3 : E6 0D 21                            ld 0Dh, #WBuffer1 #256
1965/    18B6 : 4E                                  inc R4                  ; and get ready for next one
1966/    18B7 : AF                                  ret
1967/    18B8 :                     ;
1968/    18B8 : 70 E4               Sys_Rw_Seek     push R4                 ; save state
1969/    18BA : 2C 1C                               ld R2, #OverLap /256
1970/    18BC : 3C 24                               ld R3, #OverLap #256
1971/    18BE : D6 02 CB                            call Bank_Call
1972/    18C1 : 50 E4                               pop R4
1973/    18C3 : 76 35 20                            tm DiskStat, #Wr_Op     ; no auto-offset when reading
1974/    18C6 : 6B 0C                               jr Z, Sys_Rw_Sk1
1975/    18C8 : 76 35 01                            tm DiskStat, #Offset_Set        ; or offset already set
1976/    18CB : EB 07                               jr NZ, Sys_Rw_Sk1
1977/    18CD : 2C 25                               ld R2, #ReSeek /256
1978/    18CF : 3C 31                               ld R3, #ReSeek #256
1979/    18D1 : D6 02 CB                            call Bank_Call
1980/    18D4 : 2C 14               Sys_Rw_Sk1      ld R2, #LogicalBlock /256
1981/    18D6 : 3C 80                               ld R3, #LogicalBlock #256
1982/    18D8 : 1C 03                               ld R1, #3
1983/    18DA : 0C 1C                               ld R0, #Wrk_Sys+12
1984/    18DC : 83 02               Sys_Rw_Sk_Lp    ldei @R0, @RR2          ; get block number into R12..14
1985/    18DE : 1A FC                               djnz R1, Sys_Rw_Sk_Lp
1986/    18E0 : AF                                  ret
1987/    18E1 :                     ;
1988/    18E1 : 06 EE 01            Sys_Inc_Blk     add R14, #1             ; BlockNumber:=BlockNumber+1
1989/    18E4 : 16 ED 00                            adc R13, #0
1990/    18E7 : 16 EC 00                            adc R12, #0
1991/    18EA : 2C 14                               ld R2, #LogicalBlock /256
```

```
1992/    18EC : 3C 80                              ld R3, #LogicalBlock #256
1993/    18EE : 1C 03                              ld R1, #3
1994/    18F0 : 0C 1C                              ld R0, #Wrk_Sys+12
1995/    18F2 : 93 02              Sys_Inc_Lp      ldei @RR2, @R0
1996/    18F4 : 1A FC                              djnz R1, Sys_Inc_Lp
1997/    18F6 : 00 5E                              dec Status1
1998/    18F8 : AF                                 ret
1999/    18F9 :                    ;
2000/    18F9 : FC 18              Abort_24        ld R15, #24
2001/    18FB : 8D 03 57                           jp Abort
2002/    18FE :                    ;
2003/    18FE : FC 37              Abort_55        ld R15, #55
2004/    1900 : 8D 03 57                           jp Abort
2005/    1903 :
2006/    1903 : FC 3D              Abort_61        ld R15, #61
2007/    1905 : 8D 03 57                           jp Abort
2008/    1908 :
2009/    1908 :
2010/    1908 :                    ;*****************************************************************
2011/    1908 :                    ;
2012/    1908 :                    ; Procedure: Sys_Write
2013/    1908 :                    ;
2014/    1908 :                    ;  This procedure is the System Write command for Widget. In general
2015/    1908 :                    ;  it is a major departure from the ProFile write protocol: Status is
2016/    1908 :                    ;  not sent back to the host unless there is useful information in
2017/    1908 :                    ;  the status -- the fact that there were no errors in a block
2018/    1908 :                    ;  transfer is communicated to the host via Response Byte that is
2019/    1908 :                    ;  handshaken across at CMD/BSY time. If an error has occurred, then
2020/    1908 :                    ;  the host is to recognize this state and come back to read the
2021/    1908 :                    ;  status from the controller, otherwise it is assumed that the next
2022/    1908 :                    ;  thing that the controller wants is write data. Note that the
2023/    1908 :                    ;  second handshake (data received acknowledgement) has been removed
2024/    1908 :                    ;  from the protocol.
2025/    1908 :                    ;
2026/    1908 :                    ; Inputs: none
2027/    1908 :                    ;
2028/    1908 :                    ; Outputs: none
2029/    1908 :                    ;
2030/    1908 :                    ; Algorithm:
2031/    1908 :                    ;
2032/    1908 :                    ;  Begin
2033/    1908 :                    ;    Ld_LgclBlk(Wid_Log_Offset)
2034/    1908 :                    ;    Count:=Ld_Param1.FirstParam
2035/    1908 :                    ;    BlockNumber:=Load_Logical
2036/    1908 :                    ;    ClrNormStat
2037/    1908 :                    ;    Get_Type_Check(BlockNumber+Count)
2038/    1908 :                    ;    Seek_Type:=Access_Offset
2039/    1908 :                    ;    DiskStat.Wr_Op:=true
2040/    1908 :                    ;    OverLap
2041/    1908 :                    ;    If not(ReadBlock)
2042/    1908 :                    ;      Then
2043/    1908 :                    ;        SS_RdErr
2044/    1908 :                    ;        Goto Pro_Rd_Exit
2045/    1908 :                    ;    While Always Do
2046/    1908 :                    ;      BlockNumber:=BlockNumber+1
2047/    1908 :                    ;      If not(SrchCache)
2048/    1908 :                    ;        Then OverLap
2049/    1908 :                    ;      If not(Write_Common)
2050/    1908 :                    ;        Then Data_Ex_Handler(Wr_Common.Error.Code)
2051/    1908 :                    ;      If not(Read_Fast)
2052/    1908 :                    ;        Then Goto Pro_Rd_Exit
2053/    1908 :                    ;        Else
2054/    1908 :                    ;          If BlkStat.Bad_Block
2055/    1908 :                    ;            Then Goto Pro_Rd_BB
2056/    1908 :                    ;      If Count>1
2057/    1908 :                    ;        Then
2058/    1908 :                    ;          Command_Pending, IBsy:=true, Response:=Sys_Wr_Resp
2059/    1908 :                    ;          Rd_Leave2
2060/    1908 :                    ;        Else RdLeave
2061/    1908 :                    ;      Count:=Count-1
2062/    1908 :                    ;  End
2063/    1908 :                    ;
2064/    1908 :                    ;*****************************************************************
2065/    1908 :
2066/    1908 :                    ; Cmd 1 System Write Block
2067/    1908 : D6 18 59           Sys_Write       call Sys_SetUp
2068/    190B : 46 35 2C           Sys_Wr_Lp       or DiskStat, #Wr_Op+Offset_On+User_Type
2069/    190E : D6 18 B8                           call Sys_Rw_Seek
2070/    1911 : 76 3E 20                           tm 3Eh, #B_Block        ; check if bad block
2071/    1914 : ED 19 63                           jp NZ, Sys_Wr_BB
2072/    1917 : 46 35 2C                           or DiskStat, #Wr_Op+Offset_On+User_Type
2073/    191A : D6 11 D3                           call RW_Common
2074/    191D : 6B 1E                              jr Z, Sys_WrErr
2075/    191F :                    ;
2076/    191F : 46 3E 1A                           or 3Eh, #010h+008h+002h
2077/    1922 : 4A 03                              djnz R4, Sys_Wr_Next    ; more to do?
2078/    1924 : 8D 11 46           Sys_WrE1        jp Rd_Leave
2079/    1927 : E6 06 EF           Sys_Wr_Next     ld 06h, #0EFh
2080/    192A : D6 01 98                           call L0198
2081/    192D : A2 04              Sys_Wr_M1       cp R0, R4
2082/    192F : ED 19 68                           jp NZ, Abort_56
2083/    1932 : A4 5D E1                           cp R1,>5Dh
2084/    1935 : ED 19 6D                           jp NZ, Abort_61a
2085/    1938 : D6 18 E1                           call Sys_Inc_Blk
2086/    193B : 8B CE                              jr Sys_Wr_Lp
2087/    193D :                    ;
2088/    193D : D6 13 BD           Sys_WrErr       call Data_Ex_Handler
2089/    1940 : D6 18 4A           Sys_WrChk       call Chk_FatalStat
2090/    1943 : EB DF                              jr NZ, Sys_WrE1
2091/    1945 : 46 3E 1A                           or 3Eh, #010h+008h+002h
2092/    1948 : 4A 03                              djnz R4, Sys_Wr_More
2093/    194A : 8D 19 24                           jp Sys_WrE1
2094/    194D : E6 06 EF           Sys_Wr_More     ld 06h, #0EFh
2095/    1950 : D6 01 98                           call L0198
2096/    1953 : 0C 00                              ld R0, #0
2097/    1955 : 1C 04                              ld R1, #4
2098/    1957 : 2C 14                              ld R2, #014h
2099/    1959 : 3C 7B                              ld R3, #07Bh
2100/    195B : D6 04 50                           call L0450
2101/    195E : D6 04 18                           call Load_Logical
2102/    1961 : 8B CA                              jr Sys_Wr_M1
2103/    1963 :                    ;
2104/    1963 : D6 11 5C           Sys_Wr_BB       call Wr_BBlock
2105/    1966 : 8B D8                              jr Sys_WrChk
2106/    1968 :                    ;
2107/    1968 : FC 38              Abort_56        ld R15, #56
2108/    196A : 8D 03 57                           jp Abort
2109/    196D :                    ;
```

```
2110/    196D : FC 3D              Abort_61a       ld R15, #61
2111/    196F : 8D 03 57                           jp Abort
2112/    1972 :
2113/    1972 :
2114/    1972 :                    ; Cmd 2 System WriteVerify Block
2115/    1972 : D6 18 59           Sys_WrVer       call Sys_SetUp
2116/    1975 : 46 35 2C           Sys_WV_Lp       or DiskStat, #Wr_Op+Offset_On+User_Type
2117/    1978 : D6 18 B8                           call Sys_Rw_Seek
2118/    197B : 76 3E 20                           tm 3Eh, #B_Block        ; check if bad block
2119/    197E : 6B 05                              jr Z, Sys_WVErr
2120/    1980 : D6 11 5C                           call Wr_BBlock
2121/    1983 : 8B 03                              jr Sys_WVChk
2122/    1985 :                    ;
2123/    1985 : D6 11 A4           Sys_WVErr       call WrVer_Common
2124/    1988 : D6 18 4A           Sys_WVChk       call Chk_FatalStat
2125/    198B : EB 97                              jr NZ, Sys_WrE1
2126/    198D : 46 3E 1A                           or 3Eh, #010h+008h+002h
2127/    1990 : 4A 08                              djnz R4, Sys_WV_More
2128/    1992 : 8D 11 46                           jp Rd_Leave
2129/    1995 : D6 18 E1           Sys_WV_M1       call Sys_Inc_Blk
2130/    1998 : 8B DB                              jr Sys_WV_Lp
2131/    199A :                    ;
2132/    199A : E6 06 EF           Sys_WV_More     ld 06h, #0EFh
2133/    199D : D6 01 98                           call L0198
2134/    19A0 : A2 04                              cp R0, R4
2135/    19A2 : EB C4                              jr NZ, Abort_56
2136/    19A4 : A4 5D E1                           cp R1,>5Dh
2137/    19A7 : EB C4                              jr NZ, Abort_61a
2138/    19A9 : 0C 00                              ld R0, #0
2139/    19AB : 1C 04                              ld R1, #4
2140/    19AD : 2C 14                              ld R2, #014h
2141/    19AF : 3C 7B                              ld R3, #07Bh
2142/    19B1 : D6 04 50                           call L0450
2143/    19B4 : D6 04 18                           call Load_Logical
2144/    19B7 : 8B DC                              jr Sys_WV_M1
2145/    19B9 :
2146/    19B9 :
2147/    19B9 :
2148/    19B9 :                    ;*****************************************************************
2149/    19B9 :                    ; Module Write.Assem
2150/    19B9 :                    ;
2151/    19B9 :                    ;  This module contains all but the very most primitive of procedures
2152/    19B9 :                    ;  (the routines that are resident are listed in the external spec's)
2153/    19B9 :                    ;  that concern themselves with performing a write operation.
2154/    19B9 :                    ;
2155/    19B9 :                    ;       FUNCTION WriteBlock( Parent : BYTE {R8} ) :
2156/    19B9 :                    ;                BOOLEAN
2157/    19B9 :                    ;                Status : BYTE {R0}
2158/    19B9 :                    ;                WrErrCnt : BYTE {R1}
2159/    19B9 :                    ;
2160/    19B9 :                    ;*****************************************************************
2161/    19B9 :
2162/    19B9 :                    ;*****************************************************************
2163/    19B9 :                    ;
2164/    19B9 :                    ; Function: WriteBlock
2165/    19B9 :                    ;
2166/    19B9 :                    ;  This function assumes that the heads are positioned over the
2167/    19B9 :                    ;  correct track and writes the block of data whose header matches
2168/    19B9 :                    ;  the header that is made up of the cylinder, head and sector
2169/    19B9 :                    ;  information that is stored in memory.
2170/    19B9 :                    ;
2171/    19B9 :                    ; Inputs: Parent : BYTE {R8}
2172/    19B9 :                    ;
2173/    19B9 :                    ; Outputs: WriteBlock : BOOLEAN {Zero flag, true if error in ReadBlock}
2174/    19B9 :                    ;          Status     : BYTE {R0}
2175/    19B9 :                    ;          WrErrCnt   : BYTE {R1}
2176/    19B9 :                    ;
2177/    19B9 :                    ; Global Variables Used: Cylinder, Head, Sector, Recovery
2178/    19B9 :                    ;
2179/    19B9 :                    ; Local Variables Used: WrRetryCnt : BYTE {R8}
2180/    19B9 :                    ;                       WrError    : BOOLEAN {R9/bit 7}
2181/    19B9 :                    ;                       WrExcept   : BOOLEAN {R9/bit 6}
2182/    19B9 :                    ;                       WrSuccess  : BOOLEAN {R9/bit 5}
2183/    19B9 :                    ;                       NoHeaderFound : BOOLEAN {R9/bit 4}
2184/    19B9 :                    ;                       SectorsRead: BYTE {R10}
2185/    19B9 :                    ;
2186/    19B9 :                    ; Algorithm:
2187/    19B9 :                    ;
2188/    19B9 :                    ;  BEGIN
2189/    19B9 :                    ;    SetDeadManTimer( WriteBlock, Parent )
2190/    19B9 :                    ;    WrRetryCnt := 10
2191/    19B9 :                    ;    WrErrCnt := 0
2192/    19B9 :                    ;    WrError := False
2193/    19B9 :                    ;    WrExcept := False
2194/    19B9 :                    ;    NoHeaderFound := False
2195/    19B9 :                    ;    REPEAT
2196/    19B9 :                    ;      WHeader[ 1 ] := HiCylinder
2197/    19B9 :                    ;      WHeader[ 2 ] := LoCylinder
2198/    19B9 :                    ;      WHeader[ 3 ]/bits 7:6 := Head
2199/    19B9 :                    ;      WHeader[ 3 ]/bits 5:0 := Sector
2200/    19B9 :                    ;      WHeader[ 4 ] := Invert( WHeader[ 1 ] )
2201/    19B9 :                    ;      WHeader[ 5 ] := Invert( WHeader[ 2 ] )
2202/    19B9 :                    ;      WHeader[ 6 ] := Invert( WHeader[ 3 ] )
2203/    19B9 :                    ;      WDataSync := $0001
2204/    19B9 :                    ;     _
2205/    19B9 :                    ;    /
2206/    19B9 :                    ; R  |  Set-up external ram address counter for WRITE
2207/    19B9 :                    ; E  |  Msel0:1 := Disk <--> Mem
2208/    19B9 :                    ; S  |  WHILE SectorMark DO BEGIN END
2209/    19B9 :                    ; I  |  StartL := True
2210/    19B9 :                    ; D  |  WHILE NOT( SectorDnL ) DO BEGIN END
2211/    19B9 :                    ; E  |  Status := Status_Port
2212/    19B9 :                    ; N  |  StartL := False
2213/    19B9 :                    ; T  |  Msel0:1 := Z8 <--> Mem
2214/    19B9 :                    ;    \_
2215/    19B9 :                    ;
2216/    19B9 :                    ;      CASE Status.State OF
2217/    19B9 :                    ;        NormalEndState        : RWStat.WrSuccessful := True
2218/    19B9 :                    ;
2219/    19B9 :                    ;        NoMatchingHeaderFound : RWStat.WrSuccessful := False
2220/    19B9 :                    ;                                RWStat.WrError := True
2221/    19B9 :                    ;                                RWStat.NoHeaderFound := True
2222/    19B9 :                    ;
2223/    19B9 :                    ;        AbnormalState         : Reset_StateMachine
2224/    19B9 :                    ;                                Abort
2225/    19B9 :                    ;
2226/    19B9 :                    ;      IF Status.ServoErr OR NOT( Status.ServoRdy )
2227/    19B9 :                    ;        THEN
```

```
2228/    19B9 :                     ;          RWStat.WrError := True
2229/    19B9 :                     ;          RWStat.Except := True
2230/    19B9 :                     ;
2231/    19B9 :                     ;      IF Status.WrtNvldL  AND WrSuccessful
2232/    19B9 :                     ;        THEN
2233/    19B9 :                     ;          RWStat.WrError := True
2234/    19B9 :                     ;          RWStat.WrErrCnt := WrErrCnt + 1
2235/    19B9 :                     ;          WrRetryCnt := WrRetryCnt - 1
2236/    19B9 :                     ;        ELSE
2237/    19B9 :                     ;          RdRetryCnt := 0
2238/    19B9 :                     ;
2239/    19B9 :                     ;    UNTIL NOT( Recovery ) OR NOT( WrError ) OR ( WrRetryCnt = 0 ) OR
2240/    19B9 :                     ;                WrExcept OR ( NoHeaderFound )
2241/    19B9 :                     ;    ClearDeadManTimer
2242/    19B9 :                     ;  END
2243/    19B9 :                     ;
2244/    19B9 :                     ;*********************************************************************
2245/    19B9 :
2246/    19B9 : B0 34               WriteBlock      clr RWStat              ; clear booleans
2247/    19BB : 8C 0A                               ld R8, #10              ; WrdRetryCnt:=10
2248/    19BD : B0 E9                               clr R9                  ; ErrCnt:=0
2249/    19BF : 46 35 20                            or DiskStat, #Wr_Op     ; make certain we are writing
2250/    19C2 :                     ;
2251/    19C2 : 2C 10               WrBlk_Rpt       ld R2, #WHeader /256    ; initialize gaps
2252/    19C4 : 3C 0B                               ld R3, #WHeader #256
2253/    19C6 : D6 03 F0                            call Load_Header
2254/    19C9 :                     ;
2255/    19C9 : 0C 00                               ld R0, #0
2256/    19CB : 1C 0D                               ld R1, #(WDataSync - WDataGap - 1)
2257/    19CD : 92 02               WrGap_Lp        lde @RR2, R0            ; write 13x $00
2258/    19CF : 3E                                  inc R3
2259/    19D0 : 1A FB                               djnz R1, WrGap_Lp
2260/    19D2 :                     ;
2261/    19D2 : 0C 00                               ld R0, #0               ; assume $00 for Nisha
2262/    19D4 : 76 3E 01                            tm 3Eh, #HDA_Type       ; do we have a Nisha HDA?
2263/    19D7 : 6B 02                               jr Z, L19db             ;  yes -->
2264/    19D9 : 0C FF                               ld R0, #0FFh            ;  else use $FF for Rodime
2265/    19DB : 92 02               L19db           lde @RR2, R0
2266/    19DD : 3E                                  inc R3
2267/    19DE : 0C 01                               ld R0, #1
2268/    19E0 : 92 02                               lde @RR2, R0            ; then first byte of sync
2269/    19E2 : 3E                                  inc R3
2270/    19E3 : 0C 55                               ld R0, #055h            ; then second byte of sync
2271/    19E5 : 92 02                               lde @RR2, R0
2272/    19E7 : D6 04 89                            call L0489              ; ??? generate header ???
2273/    19EA : D6 01 FF                            call L01ff              ; ??? Wr_Resident ???
2274/    19ED : 42 AA                               or R10, R10
2275/    19EF : 6B 3D                               jr Z, WrBlk_NoHdr
2276/    19F1 : 18 E5                               ld R1, R5               ; CASE Status.State
2277/    19F3 : 56 E1 0F                            and R1, #YMask
2278/    19F6 : A6 E1 02                            cp R1, #Norm_State      ; statemachine healthy?
2279/    19F9 : EB 27                               jr NZ, WrBlk_Abnorm
2280/    19FB :                     ;
2281/    19FB : 76 E5 10            WrBlk_Norm      tm R5, #ServoErr        ; IF ServorErr OR NOT(ServoRdy)
2282/    19FE : EB 40                               jr NZ, WrBlk_ServoErr
2283/    1A00 : 76 E5 20                            tm R5, #ServoRdy
2284/    1A03 : 6B 3B                               jr Z, WrBlk_ServoErr
2285/    1A05 : 76 15 40            Wr_ServoOk      tm >15h, #WrtNvldL      ; IF Status.WrtNtVldL (ECC error)?
2286/    1A08 : 6B 3B                               jr Z, WrBlk_NVld
2287/    1A0A :
2288/    1A0A : 56 34 7F                            and RWStat, #0FFh-WrError
2289/    1A0D : 76 32 80            WrBlk_Until     tm Excpt_Stat, #Recovery
2290/    1A10 : 6B 07                               jr Z, WrBlk_End
2291/    1A12 : 76 34 80                            tm RWStat, #WrError
2292/    1A15 : 6B 02                               jr Z, WrBlk_End
2293/    1A17 : 8A 04                               djnz R8, WrBlk_Rpt1     ; try again if fault occurred
2294/    1A19 :                     ;
2295/    1A19 : 66 34 80            WrBlk_End       tcm RWStat, #WrError    ; set zero flag if error
2296/    1A1C : AF                                  ret
2297/    1A1D :                     ;
2298/    1A1D : D6 05 8E            WrBlk_Rpt1      call ZeroHeader         ; clear up header
2299/    1A20 : 8B A0                               jr WrBlk_Rpt            ; and try again
2300/    1A22 :                     ;
2301/    1A22 : D6 03 C8            WrBlk_Abnorm    call Reset_StMach
2302/    1A25 : 46 34 88                            or RWStat, #WrError+WrSMErr
2303/    1A28 : 9C 10                               ld R9, #010h
2304/    1A2A : 42 91                               or R9, R1
2305/    1A2C : 8B EB                               jr WrBlk_End
2306/    1A2E :                     ;
2307/    1A2E : 76 E5 10            WrBlk_NoHdr     tm R5, #ServoErr
2308/    1A31 : EB 0D                               jr NZ, WrBlk_ServoErr
2309/    1A33 : 76 E5 20                            tm R5, #ServoRdy
2310/    1A36 : 6B 08                               jr Z, WrBlk_ServoErr
2311/    1A38 : 46 34 A0                            or RWStat, #WrError+WrNoHdrFnd
2312/    1A3B : 46 E9 20                            or R9, #020h
2313/    1A3E : 8B D9                               jr WrBlk_End
2314/    1A40 :
2315/    1A40 : 46 34 C0            WrBlk_ServoErr  or RWStat, #WrError+WrSrvoErr   ; THEN WrError AND WrSrvoErr
2316/    1A43 : 8B D4                               jr WrBlk_End
2317/    1A45 :                     ;
2318/    1A45 : 46 34 80            WrBlk_NVld      or RWStat, #WrError             ;  ELSE
2319/    1A48 : 9E                                  inc R9
2320/    1A49 : 8B C2                               jr WrBlk_Until
2321/    1A4B :
2322/    1A4B :
2323/    1A4B :
2324/    1A4B :                     ;*********************************************************************
2325/    1A4B :                     ; Module Read.Assem
2326/    1A4B :                     ;
2327/    1A4B :                     ; This module contains all but the very most primitive of
2328/    1A4B :                     ; procedures (the routines that are resident are listed
2329/    1A4B :                     ; in the external spec's) that concern themselves with performing
2330/    1A4B :                     ; a read operation.
2331/    1A4B :                     ;
2332/    1A4B :                     ;       FUNCTION ReadBlock( Parent : BYTE {R8} ) :
2333/    1A4B :                     ;                BOOLEAN
2334/    1A4B :                     ;                Status : BYTE {R0}
2335/    1A4B :                     ;                 RdErrCnt : BYTE {R1}
2336/    1A4B :                     ;
2337/    1A4B :                     ;*********************************************************************
2338/    1A4B :
2339/    1A4B :                     ;*********************************************************************
2340/    1A4B :                     ;
2341/    1A4B :                     ; Function: ReadBlock
2342/    1A4B :                     ;
2343/    1A4B :                     ;  This function assumes that the heads are positioned over the
2344/    1A4B :                     ;  correct track and reads the block of data whose header matched
2345/    1A4B :                     ;  the header that is made up of the cylinder, head and sector
```

```
2346/    1A4B :                     ;  information that is stored in memory.
2347/    1A4B :                     ;
2348/    1A4B :                     ; Inputs: Parent: BYTE {R8}
2349/    1A4B :                     ;
2350/    1A4B :                     ; Outputs: ReadBlock: BOOLEAN {zero flag, true if error in ReadBlock}
2351/    1A4B :                     ;          Status:    BYTE {R0}
2352/    1A4B :                     ;          RdErrCnt:  BYTE {R1}
2353/    1A4B :                     ;
2354/    1A4B :                     ; Global Variables Used: Cylinder, Head, Sector, Recovery
2355/    1A4B :                     ;
2356/    1A4B :                     ; Local Variables Used: RdRetryCnt:  BYTE {R8}
2357/    1A4B :                     ;                       RdError:     BOOLEAN {R9/bit 7}
2358/    1A4B :                     ;                       RdExcept:    BOOLEAN {R9/bit 6}
2359/    1A4B :                     ;                       RdSuccess:   BOOLEAN {R9/bit 5}
2360/    1A4B :                     ;                       SectorsRead: BOOLEAN {R9/bit 4}
2361/    1A4B :                     ;
2362/    1A4B :                     ; Algorithm:
2363/    1A4B :                     ;
2364/    1A4B :                     ;  Begin
2365/    1A4B :                     ;    SetDeadManTimer(ReadBlock, Parent)
2366/    1A4B :                     ;    RdRetryCnt:=10
2367/    1A4B :                     ;    RdErrCnt:=0
2368/    1A4B :                     ;    RdError:=false
2369/    1A4B :                     ;    RdExcept:=false
2370/    1A4B :                     ;    NoHeaderFound:=false
2371/    1A4B :                     ;    SectorsRead:=2xNbrSctrs {try to find header for two rotations}
2372/    1A4B :                     ;    Repeat
2373/    1A4B :                     ;      RHeader[1]:=HiCylinder
2374/    1A4B :                     ;      RHeader[2]:=LoCylinder
2375/    1A4B :                     ;      RHeader[3]/bits 7:6:=Head
2376/    1A4B :                     ;      RHeader[3]/bits 5:0:=Sector
2377/    1A4B :                     ;      RHeader[4]:=invert(RHeader[1])
2378/    1A4B :                     ;      RHeader[5]:=invert(RHeader[2])
2379/    1A4B :                     ;      RHeader[6]:=invert(RHeader[3])
2380/    1A4B :                     ;      ReadArray[RDummy-1]:=0
2381/    1A4B :                     ;    /-
2382/    1A4B :                     ;  R | Set-up external RAM address counter for read
2383/    1A4B :                     ;  E | Msel0:1:=Disk<-->Mem
2384/    1A4B :                     ;  S | While SectorMark Do Begin End
2385/    1A4B :                     ;  I | StartL:=true
2386/    1A4B :                     ;  D | While not(SectorDnL) Do Begin End
2387/    1A4B :                     ;  E | Status:=StatusPort
2388/    1A4B :                     ;  N | StartL:=false
2389/    1A4B :                     ;  T | Msel0:1:=Z8<-->Mem
2390/    1A4B :                     ;    \-
2391/    1A4B :                     ;     Case Status.State Of
2392/    1A4B :                     ;       NormalEndState:        RdSuccess:=true
2393/    1A4B :                     ;       NoMatchingHeaderFound: RdSuccess:=false
2394/    1A4B :                     ;                              RdError:=true
2395/    1A4B :                     ;                              NoHeaderFound:=true
2396/    1A4B :                     ;       AbnormalState:         Reset_StateMachine
2397/    1A4B :                     ;                              Abort
2398/    1A4B :                     ;     If Status.ServoErr Or not(Status.ServoRdy)
2399/    1A4B :                     ;       Then
2400/    1A4B :                     ;         RdError:=true
2401/    1A4B :                     ;         RdExcept:=true
2402/    1A4B :                     ;     If Status.CrcErr And RdSuccess
2403/    1A4B :                     ;       Then
2404/    1A4B :                     ;         RdError:=true
2405/    1A4B :                     ;         RdErrCnt:=RdErrCnt+1
2406/    1A4B :                     ;         RdRetryCnt:=RdRetryCnt-1
2407/    1A4B :                     ;       Else
2408/    1A4B :                     ;         If RdError
2409/    1A4B :                     ;           Then
2410/    1A4B :                     ;             BlockMove(Buffer2, RBuffer1)
2411/    1A4B :                     ;             RdRetryCnt:=RdRetryCnt-1
2412/    1A4B :                     ;    Until not(Recovery) Or not(RdError) Or RdRetryCnt=0 Or
2413/    1A4B :                     ;            RdExcept Or NoHeaderFound
2414/    1A4B :                     ;    ClearDeadManTimer
2415/    1A4B :                     ;    Status:=R9
2416/    1A4B :                     ;  End
2417/    1A4B :                     ;
2418/    1A4B :                     ;*****************************************************************
2419/    1A4B :                     
2420/    1A4B : B0 34               ReadBlock       clr RWStat              ; clear booleans
2421/    1A4D : 8C 0A                               ld R8, #10              ; RdRetryCnt:=10
2422/    1A4F : B0 E9                               clr R9                  ; ErrCnt:=0
2423/    1A51 : 56 35 DF                            and DiskStat, #0FFh-Wr_Op       ; make sure we are reading
2424/    1A54 : 2C 10               RdBlk_Rpt       ld R2, #RHeader /256    ; initialize gaps
2425/    1A56 : 3C 0B                               ld R3, #RHeader #256
2426/    1A58 : D6 03 F0                            call Load_Header
2427/    1A5B : 0C 00                               ld R0, #0
2428/    1A5D : 92 02                               lde @RR2, R0
2429/    1A5F : 2C 10                               ld R2, #(RDummy-1) /256
2430/    1A61 : 3C 17                               ld R3, #(RDummy-1) #256
2431/    1A63 : 92 02                               lde @RR2, R0
2432/    1A65 : A0 E2                               incw RR2
2433/    1A67 : 92 02                               lde @RR2, R0
2434/    1A69 : D6 04 A3                            call L04a3
2435/    1A6C : D6 02 01                            call L0201
2436/    1A6F : 42 AA                               or R10, R10
2437/    1A71 : 6B 3F                               jr Z, RdBlk_NoHdr
2438/    1A73 : 18 15                               ld R1,>15h              ; R5
2439/    1A75 : 56 E1 0F                            and R1, #YMask
2440/    1A78 : A6 E1 02                            cp R1, #Norm_State
2441/    1A7B : EB 29                               jr NZ, RdBlk_AbNorm
2442/    1A7D :                     ;
2443/    1A7D : 76 E5 10            RdBlk_Norm      tm R5, #ServoErr        ; If ServoErr Or not(ServoRdy)
2444/    1A80 : EB 58                               jr NZ, RD_ServoErr
2445/    1A82 : 76 E5 20                            tm R5, #ServoRdy
2446/    1A85 : 6B 53                               jr Z, RD_ServoErr
2447/    1A87 :                     ;
2448/    1A87 : 76 E5 80                            tm R5, #CrcErrL         ; If Status.CrcErr
2449/    1A8A : 6D 1A DF                            jp Z, RD_BadCrc
2450/    1A8D : 76 34 80                            tm RWStat, #RdError
2451/    1A90 : ED 1A FB                            jp NZ, RdBlk_Rmove
2452/    1A93 : 76 32 80            RdBlk_Until     tm Excpt_Stat, #Recovery
2453/    1A96 : 6B 05                               jr Z, RdBlk_End
2454/    1A98 : 76 34 80                            tm RWStat, #RdError
2455/    1A9B : EB 04                               jr NZ, RdB_Rpt1
2456/    1A9D :                     ;
2457/    1A9D : 66 34 80            RdBlk_End       tcm RWStat, #RdError    ; set zero flag if error
2458/    1AA0 : AF                                  ret
2459/    1AA1 :                     ;
2460/    1AA1 : D6 05 8E            RdB_Rpt1        call ZeroHeader
2461/    1AA4 : 8B AE                               jr RdBlk_Rpt
2462/    1AA6 :                     ;
2463/    1AA6 : D6 03 C8            RdBlk_AbNorm    call Reset_StMach
```

```
2464/    1AA9 : 46 34 88                        or RWStat, #RdError+RdSMErr
2465/    1AAC : 9C 10                           ld R9, #010h
2466/    1AAE : 42 91                           or R9, R1
2467/    1AB0 : 8B EB                           jr RdBlk_End
2468/    1AB2 :                        ;
2469/    1AB2 : 76 E5 10               RdBlk_NoHdr     tm R5, #ServoErr
2470/    1AB5 : EB 23                           jr NZ, RD_ServoErr
2471/    1AB7 : 76 E5 20                        tm R5, #ServoRdy
2472/    1ABA : 6B 1E                           jr Z, RD_ServoErr
2473/    1ABC : 46 34 80                        or RWStat, #RdError
2474/    1ABF : 76 19 20                        tm >19h, #Hdr_MisMatch          ; R9
2475/    1AC2 : EB 11                           jr NZ, Rd_HdrErr
2476/    1AC4 : 46 19 20                        or >19h, #Hdr_MisMatch
2477/    1AC7 : 76 32 80                        tm Excpt_Stat, #Recovery        ; auto offset ONLY if recovery on
2478/    1ACA : 6B C7                           jr Z, RdBlk_Until
2479/    1ACC : 2C 25                           ld R2, #ReSeek /256             ; set auto-offset
2480/    1ACE : 3C 31                           ld R3, #ReSeek #256
2481/    1AD0 : D6 02 CB                        call Bank_Call
2482/    1AD3 : 8B BE                           jr RdBlk_Until
2483/    1AD5 :                        ;
2484/    1AD5 : 46 34 A0               Rd_HdrErr       or RWStat, #RdError+RdNoHdrFnd
2485/    1AD8 : 8B C3                           jr RdBlk_End
2486/    1ADA :                        ;
2487/    1ADA : 46 34 C0               RD_ServoErr     or RWStat, #RdError+RdSrvoErr   ; Then RdError And RdServoErr
2488/    1ADD : 8B BE                           jr RdBlk_End
2489/    1ADF :                        ;
2490/    1ADF : 46 34 90               RD_BadCrc       or RWStat, #RdError+RdCrcErr    ; Else
2491/    1AE2 : 9E                              inc R9
2492/    1AE3 : 46 E9 80                        or R9, #EccStat
2493/    1AE6 : 76 32 80                        tm Excpt_Stat, #Recovery        ; re-seek only if recovery is on
2494/    1AE9 : 6B 0C                           jr Z, Rd_B_Crc_1
2495/    1AEB : 76 35 01                        tm DiskStat, #Offset_Set                ; set auto-offset if needed
2496/    1AEE : EB 07                           jr NZ, Rd_B_Crc_1
2497/    1AF0 : 2C 25                           ld R2, #ReSeek /256
2498/    1AF2 : 3C 31                           ld R3, #ReSeek #256
2499/    1AF4 : D6 02 CB                        call Bank_Call
2500/    1AF7 : 8A 9A                  Rd_B_Crc_1      djnz R8, RdBlk_Until
2501/    1AF9 : 8B A2                           jr RdBlk_End
2502/    1AFB :                        ;
2503/    1AFB : 2C 20                  RdBlk_Rmove     ld R2, #RBuf_To_Buf2 /256
2504/    1AFD : 3C 8A                           ld R3, #RBuf_To_Buf2 #256
2505/    1AFF : D6 02 CB                        call Bank_Call
2506/    1B02 : 8B F3                           jr Rd_B_Crc_1
2507/    1B04 :
2508/    1B04 :
2509/    1B04 :
2510/    1B04 :                        ;*****************************************************************
2511/    1B04 :                        ; Module Cache.Assem
2512/    1B04 :                        ;
2513/    1B04 :                        ; This name of this module is a bit misleading: there is no true
2514/    1B04 :                        ; cache implemented at this time. However, there is an attempt made
2515/    1B04 :                        ; at some primitive look-ahead methods to reduce the access time
2516/    1B04 :                        ; of the disk. This module contains those routines that are
2517/    1B04 :                        ; chiefly involved with the look-ahead algorithm.
2518/    1B04 :                        ;
2519/    1B04 :                        ;       FUNCTION CnvrtLogical(LogicalBlock : 3 BYTES {R12:14}) :
2520/    1B04 :                        ;               CnvrtLogical : BOOLEAN {true if block spared}
2521/    1B04 :                        ;               Cylinder : WORD {RR12}
2522/    1B04 :                        ;               Head : BYTE {R14}
2523/    1B04 :                        ;               Sector : BYTE {R15}
2524/    1B04 :                        ;               Status : BYTE {R0}
2525/    1B04 :                        ;               Ptr : BYTE {R1}
2526/    1B04 :                        ;       FUNCTION SrchCach(LogicalBlock : 3 BYTES {R12:14}
2527/    1B04 :                        ;               Random : BOOLEAN {R7/Bit 7}
2528/    1B04 :                        ;               Offset : 3 BITS {R7/Bits 2:0}) :
2529/    1B04 :                        ;               BOOLEAN
2530/    1B04 :                        ;               HeadSector : BYTE {R0}
2531/    1B04 :                        ;
2532/    1B04 :                        ;*****************************************************************
2533/    1B04 :
2534/    1B04 :                        ;*****************************************************************
2535/    1B04 :                        ;
2536/    1B04 :                        ; Function: CnvrtLogical  {convert logical block}
2537/    1B04 :                        ;
2538/    1B04 :                        ;  This function is responsible for converting a logical block
2539/    1B04 :                        ;  number to a physical block number by first searching the spare
2540/    1B04 :                        ;  table and then doing the appropriate arithmetic to arrive at
2541/    1B04 :                        ;  the cylinder, head, and sector value.
2542/    1B04 :                        ;
2543/    1B04 :                        ; Inputs: LogicalBlockNumber: 3 BYTES {R12:14}
2544/    1B04 :                        ;
2545/    1B04 :                        ; Outputs: Cylinder: WORD {R12:13}
2546/    1B04 :                        ;          Head:     BYTE {R14}
2547/    1B04 :                        ;          Sector:   BYTE {R15}
2548/    1B04 :                        ;          Status:   BYTE {R0, returned from search spare table}
2549/    1B04 :                        ;          Ptr:      BYTE {R1, returned from search spare table}
2550/    1B04 :                        ;
2551/    1B04 :                        ; Local Variables: PhysicalBlock: 3 BYTES {R12:14}
2552/    1B04 :                        ;                  Temp:          3 BYTES {R1:3}
2553/    1B04 :                        ;
2554/    1B04 :                        ; Algorithm:
2555/    1B04 :                        ;
2556/    1B04 :                        ;  Begin
2557/    1B04 :                        ;    If LogicalBlock>MaxLogicalBlock Then Abort
2558/    1B04 :                        ;    PhysicalBlock:=SearchSpareTable(LogicalBlock)
2559/    1B04 :                        ;    Cylinder:=PhysicalBlock div (Heads*Sectors)
2560/    1B04 :                        ;    Temp:=PhysicalBlock mod (Heads*Sectors)
2561/    1B04 :                        ;    Head:=Temp div Sectors
2562/    1B04 :                        ;    Sector:=Temp mod Sectors
2563/    1B04 :                        ;  End
2564/    1B04 :                        ;
2565/    1B04 :                        ;*****************************************************************
2566/    1B04 :
2567/    1B04 : D6 02 AB               CnvrtLogical    call L02ab
2568/    1B07 :                        ;
2569/    1B07 :                        ; *** SrchSpTabl function inline
2570/    1B07 :                        ;  This procedure is responsible for checking to see if the block number
2571/    1B07 :                        ;  that is passed into it is currently in the spare table. If the block
2572/    1B07 :                        ;  is found to be in the spare table then the physical block number of
2573/    1B07 :                        ;  the spare block is passed back to the caller, as well as the PTR to
2574/    1B07 :                        ;  the location of spared block's element within the spare table. In any
2575/    1B07 :                        ;  case, a byte of status is always passed back to the caller describing
2576/    1B07 :                        ;  the state of the logical block within the spare table.
2577/    1B07 :                        ;
2578/    1B07 :                        ; Inputs: LogicalBlockNumber: 3 BYTES {R12:14}
2579/    1B07 :                        ;         ElementType:        BYTE {R15}
2580/    1B07 :                        ;
2581/    1B07 :                        ; Outputs: CnvrtLogical:  BOOLEAN {zero flag set if not found in table}
```

```
2582/    1B07 :                     ;          PhysicalBlockNumber: 3 BYTES {R12:14}
2583/    1B07 :                     ;          Status:              BYTE {ScrReg0}
2584/    1B07 :                     ;          ElementPtr:          BYTE {ScrReg1}
2585/    1B07 :                     ;
2586/    1B07 : B0 E7                           clr R7               ; Found:=false
2587/    1B09 :                     ; *** Get_HeadPtr function inline
2588/    1B09 : 18 ED                           ld R1, R13           ; check if head ptr is NIL
2589/    1B0B : 56 E1 FC                        and R1, #0FCh        ; but first form a headptr structure
2590/    1B0E : E0 E1                           rr R1                ; and index into HdPtr array
2591/    1B10 : E0 E1                           rr R1
2592/    1B12 : 2C 14                           ld R2, #SegPtrArray /256
2593/    1B14 : 3C 9F                           ld R3, #SegPtrArray #256
2594/    1B16 : 02 31                           add R3, R1
2595/    1B18 : 16 E2 00                        adc R2, #0
2596/    1B1B : 82 02                           lde R0, @RR2         ; get HeadPtr and check for NIL
2597/    1B1D : 66 E0 80                        tcm R0, #Nil
2598/    1B20 : 6D 1B A3                        jp Z, NotInTabl      ; do a real search if not NIL
2599/    1B23 :                     ;
2600/    1B23 : A6 E0 4C            SrchLp      cp R0, #304/4        ; pointer out of range?
2601/    1B26 : 1D 1B 2E                        jp LT, SST_GetPtr
2602/    1B29 : FC 32                           ld R15, #50
2603/    1B2B : 8D 03 57                        jp Abort
2604/    1B2E :                     ; *** Get_Ptr function inline
2605/    1B2E : 09 41               SST_GetPtr  ld ScrReg1, R0       ; save current ptr
2606/    1B30 : 2C 14                           ld R2, #SpareTable /256
2607/    1B32 : 3C EB                           ld R3, #SpareTable #256
2608/    1B34 : 18 E0                           ld R1, R0
2609/    1B36 : B0 E0                           clr R0
2610/    1B38 : 02 11                           add R1, R1           ;  RR0 := 4 * R1
2611/    1B3A : 02 11                           add R1, R1
2612/    1B3C : 16 E0 00                        adc R0, #0
2613/    1B3F : 02 31                           add R3, R1           ; add offset into table
2614/    1B41 : 12 20                           adc R2, R0
2615/    1B43 : 82 12                           lde R1, @RR2         ; get element status
2616/    1B45 : 19 40                           ld ScrReg0, R1       ; save element status
2617/    1B47 : 76 E1 20                        tm R1, #Useable      ; If Useable
2618/    1B4A : 6B 43                           jr Z, SrchLpElse
2619/    1B4C : 76 35 04                        tm DiskStat, #User_Type
2620/    1B4F : 6B 04                           jr Z, SST_GetPtr1    ; no user block, set spare table type
2621/    1B51 : 0C 02                           ld R0, #UserBlk_Type
2622/    1B53 : 8B 02                           jr SST_GetPtr2
2623/    1B55 : 0C 08               SST_GetPtr1 ld R0, #SprTbl_Type
2624/    1B57 : 72 10               SST_GetPtr2 tm R1, R0            ; And Type is correct
2625/    1B59 : 6B 34                           jr Z, SrchLpElse
2626/    1B5B : A0 E2                           incw RR2
2627/    1B5D : 82 02                           lde R0, @RR2         ; get token
2628/    1B5F : A0 E2                           incw RR2
2629/    1B61 : 82 12                           lde R1, @RR2
2630/    1B63 : A2 1E                           cp R1, R14           ; check bits 0:7 first
2631/    1B65 : EB 28                           jr NZ, SrchLpElse
2632/    1B67 : 56 E0 03                        and R0, #3
2633/    1B6A : 18 ED                           ld R1, R13
2634/    1B6C : 56 E1 03                        and R1, #3
2635/    1B6F : A2 01                           cp R0, R1            ; then bits 8:9
2636/    1B71 : EB 1C                           jr NZ, SrchLpElse
2637/    1B73 : 76 40 10                        tm ScrReg0, #Spare   ; check if BadBlock
2638/    1B76 : EB 05                           jr NZ, Srch_Spare
2639/    1B78 : 46 E7 01                        or R7, #Found
2640/    1B7B : 8B 26                           jr NotInTabl
2641/    1B7D :                     ;
2642/    1B7D : 08 41               Srch_Spare  ld R0, ScrReg1       ; get back pointer
2643/    1B7F : 0E                              inc R0               ; number of spare blocks 1..76
2644/    1B80 :                     ; ***** INLINE: MulR0_m *****
2645/    1B80 : B0 EE               MulR0_m     clr R14              ; Result:= R0 * 256
2646/    1B82 : D8 E0                           ld R13, R0
2647/    1B84 : B0 EC                           clr R12
2648/    1B86 : 10 EE                           rlc R14              ; Result := Result * 2
2649/    1B88 : 10 ED                           rlc R13
2650/    1B8A :                     ; ***************************
2651/    1B8A : 46 E7 01                        or R7, #Found
2652/    1B8D : 8B 2F                           jr Get_Cyl_H_S
2653/    1B8F :                     ;
2654/    1B8F : 76 40 80            SrchLpElse  tm ScrReg0, #Nil     ; test if element.Ptr=Nil
2655/    1B92 : EB 0F                           jr NZ, NotInTabl
2656/    1B94 :                     ;
2657/    1B94 : 08 41                           ld R0, ScrReg1       ; get address of current element
2658/    1B96 : D6 15 1F                        call Get_Ptr
2659/    1B99 : 06 E3 03                        add R3, #3           ; point to next Ptr
2660/    1B9C : 16 E2 00                        adc R2, #0
2661/    1B9F : 82 02                           lde R0, @RR2
2662/    1BA1 : 8B 80                           jr SrchLp
2663/    1BA3 :
2664/    1BA3 : 28 ED               NotInTabl   ld R2, R13           ; shift right 1 byte
2665/    1BA5 : CF                              rcf
2666/    1BA6 : C0 E2                           rrc R2
2667/    1BA8 : 02 E2                           add R14, R2          ; PhysicalBlock:=LogicalBlock +
2668/    1BAA : 16 ED 00                        adc R13, #0          ;              LogicalBlock DIV k
2669/    1BAD : 08 ED                           ld R0, R13           ; save rollover byte
2670/    1BAF : 56 E0 FE                        and R0, #0FEh        ; mask off MOD 512 bits
2671/    1BB2 : 90 E2                           rl R2
2672/    1BB4 : A2 02                           cp R0, R2            ; check for rollover
2673/    1BB6 : 6B 06                           jr Z, Get_Cyl_H_S
2674/    1BB8 : 06 EE 01                        add R14, #1          ; otherwise account for rollover
2675/    1BBB : 16 ED 00                        adc R13, #0
2676/    1BBE :                     ;
2677/    1BBE :                     ; *** Get_Cyl_H_S function inline
2678/    1BBE :                     ;  This routine extracts the cylinder, head, and sector information
2679/    1BBE :                     ;  from a physical block number.
2680/    1BBE :                     ;
2681/    1BBE :                     ; Inputs: PhysicalBlockNumber: 3 BYTES {R12:14}
2682/    1BBE :                     ;
2683/    1BBE :                     ; Outputs: HiCylinder: BYTE {R12}
2684/    1BBE :                     ;          LoCylinder: BYTE {R13}
2685/    1BBE :                     ;          Head:       BYTE {R14}
2686/    1BBE :                     ;          Sector:     BYTE {R15}
2687/    1BBE :                     ;
2688/    1BBE : 08 ED               Get_Cyl_H_S ld R0, R13           ; physical LBA
2689/    1BC0 : 18 EE                           ld R1, R14
2690/    1BC2 : CF                              rcf                  ; calculate cylinder number
2691/    1BC3 : C0 E0                           rrc R0               ; /(32 sectors * 2 heads)
2692/    1BC5 : C0 E1                           rrc R1
2693/    1BC7 : C0 E0                           rrc R0
2694/    1BC9 : C0 E1                           rrc R1
2695/    1BCB : C0 E0                           rrc R0
2696/    1BCD : C0 E1                           rrc R1
2697/    1BCF : C0 E0                           rrc R0
2698/    1BD1 : C0 E1                           rrc R1
2699/    1BD3 : C0 E0                           rrc R0
```

```
2700/    1BD5 : C0 E1                              rrc R1
2701/    1BD7 : C0 E0                              rrc R0
2702/    1BD9 : C0 E1                              rrc R1
2703/    1BDB : 76 3E 01                           tm 3Eh, #HDA_Type       ; Nisha HDA?
2704/    1BDE : 6B 04                              jr Z, Get_CHS1          ;  yes --> done
2705/    1BE0 : C0 E0                              rrc R0                  ; otherwise /(32 sectors * 4 heads)
2706/    1BE2 : C0 E1                              rrc R1
2707/    1BE4 : 56 E0 03           Get_CHS1        and R0, #3              ; clip HiCyl
2708/    1BE7 : F8 EE                              ld R15, R14
2709/    1BE9 : 56 EF 1F                           and R15, #NbrSctrs-1    ; get sector
2710/    1BEC : 28 EE                              ld R2, R14
2711/    1BEE : EC 00                              ld R14, #0              ; set head 0
2712/    1BF0 : 56 E2 7F                           and R2, #(NbrHds_A*NbrSctrs)-1  ; Rodime values
2713/    1BF3 : 76 3E 01                           tm 3Eh, #HDA_Type       ; Nisha HDA?
2714/    1BF6 : EB 03                              jr NZ, Get_CHS2         ;  no -->
2715/    1BF8 : 56 E2 3F                           and R2, #(NbrHds_B*NbrSctrs)-1  ; Nisha values
2716/    1BFB : 76 E2 20           Get_CHS2        tm R2, #020h            ; head 1 ?
2717/    1BFE : 6B 03                              jr Z, Get_CHS3
2718/    1C00 : 46 EE 01                           or R14, #001h           ; set Hs0 bit
2719/    1C03 : 76 E2 40           Get_CHS3        tm R2, #040h            ; head 2 or 3 ?
2720/    1C06 : 6B 03                              jr Z, Get_CHS4
2721/    1C08 : 46 EE 02                           or R14, #002h           ; set Hs1 bit
2722/    1C0B : D8 E1              Get_CHS4        ld R13, R1              ; HiCyl
2723/    1C0D : C8 E0                              ld R12, R0              ; LoCyl
2724/    1C0F : 2C 16                              ld R2, #Map_Table /256
2725/    1C11 : 3C 1B                              ld R3, #Map_Table #256
2726/    1C13 : 02 3F                              add R3, R15             ; index into interleave table
2727/    1C15 : 16 E2 00                           adc R2, #0
2728/    1C18 : 82 F2                              lde R15, @RR2           ; get remapped sector
2729/    1C1A : 8F                                 di
2730/    1C1B : 42 77                              or R7, R7               ; set zero flag if not found in spare table
2731/    1C1D : 08 40                              ld R0, ScrReg0          ; Search Status
2732/    1C1F : 18 41                              ld R1, ScrReg1          ; Element.Ptr
2733/    1C21 : 8D 02 F6                           jp Bank_Ret
2734/    1C24 :
2735/    1C24 :
2736/    1C24 :                    ;*****************************************************************
2737/    1C24 :                    ;
2738/    1C24 :                    ; Function: OverLap  {overlapped seek}
2739/    1C24 :                    ;
2740/    1C24 :                    ;  Convert logical LBA into physical CHS and perform seek.
2741/    1C24 :                    ;  This fuction allows the drive to begin a seek operation (if
2742/    1C24 :                    ;  one is needed) before getting tied up with the SOS driver.
2743/    1C24 :                    ;
2744/    1C24 :                    ; Inputs: BlockNumber: 3 BYTES {R12:14}
2745/    1C24 :                    ;
2746/    1C24 :                    ; Outputs: Block Status: BYTE {3Eh}
2747/    1C24 :                    ;
2748/    1C24 :                    ;*****************************************************************
2749/    1C24 :
2750/    1C24 : 56 3E 9F           OverLap         and 3Eh, #0FFh-S_Block-B_Block
2751/    1C27 : 2C 1B                              ld R2, #CnvrtLogical /256
2752/    1C29 : 3C 04                              ld R3, #CnvrtLogical #256
2753/    1C2B : D6 02 CB                           call Bank_Call
2754/    1C2E : 6B 0D                              jr Z, Ld_BlkStat        ; LBA not in spare table
2755/    1C30 : 76 E0 10                           tm R0, #Spare           ; check if block is a spare
2756/    1C33 : 6B 05                              jr Z, Ld_Blk_BB
2757/    1C35 : 46 3E 40                           or 3Eh, #S_Block        ; set Spare Block status
2758/    1C38 : 8B 03                              jr Ld_BlkStat
2759/    1C3A : 46 3E 20           Ld_Blk_BB       or 3Eh, #B_Block        ; otherwise it must be a Bad Block
2760/    1C3D :                    ;
2761/    1C3D : 08 EC              Ld_BlkStat      ld R0, R12
2762/    1C3F : 18 ED                              ld R1, R13
2763/    1C41 : B4 3A E0                           xor R0, Cylinder
2764/    1C44 : B4 3B E1                           xor R1, Cylinder+1
2765/    1C47 : 42 01                              or R0, R1
2766/    1C49 : EB 0C                              jr NZ, Ld_Blk_Seek      ; seek needed
2767/    1C4B : F9 3D                              ld Sector, R15
2768/    1C4D : A4 3C EE                           cp R14, Head
2769/    1C50 : 6B 0C                              jr Z, Ld_Blk_Done       ; no head change needed
2770/    1C52 : D6 1C 61                           call Ld_Blk_Hs          ;  else do head change
2771/    1C55 : 8B 07                              jr Ld_Blk_Done
2772/    1C57 :                    ;
2773/    1C57 : 2C 25              Ld_Blk_Seek     ld R2, #Seek /256
2774/    1C59 : 3C 47                              ld R3, #Seek #256
2775/    1C5B : D6 02 CB                           call Bank_Call
2776/    1C5E : 8D 02 F6           Ld_Blk_Done     jp Bank_Ret
2777/    1C61 :                    ;
2778/    1C61 : E9 3C              Ld_Blk_Hs       ld Head, R14
2779/    1C63 : 56 00 CF                           and P0, #0FFh-Hs0-Hs1
2780/    1C66 : 76 3C 01                           tm Head, #001h
2781/    1C69 : 6B 03                              jr Z, Ld_Blk_Hs1
2782/    1C6B : 46 00 10                           or P0, #Hs0
2783/    1C6E : 76 3C 02           Ld_Blk_Hs1      tm Head, #002h
2784/    1C71 : 6B 03                              jr Z, Ld_Blk_Hs2
2785/    1C73 : 46 00 20                           or P0, #Hs1
2786/    1C76 : AF                 Ld_Blk_Hs2      ret
2787/    1C77 :
2788/    1C77 :
2789/    1C77 :
2790/    1C77 : D6 1D 20           L1c77           call L1d20
2791/    1C7A : D6 1D 28                           call L1d28
2792/    1C7D : 46 02 08                           or P2, #008h
2793/    1C80 : 56 50 10           L1c80           and >50h, #010h
2794/    1C83 : B0 51                              clr >51h
2795/    1C85 : D6 1D 80                           call L1d80
2796/    1C88 : 76 50 40                           tm >50h, #040h
2797/    1C8B : EB 19                              jr NZ, L1ca6
2798/    1C8D : 76 51 10                           tm >51h, #010h
2799/    1C90 : 6B 06                              jr Z, L1c98
2800/    1C92 : D6 1D 12                           call L1d12
2801/    1C95 : D6 1E AD                           call L1ead
2802/    1C98 : 76 50 10           L1c98           tm >50h, #010h
2803/    1C9B : 6B E3                              jr Z, L1c80
2804/    1C9D : 31 10                              srp #Wrk_Sys
2805/    1C9F :                                  assume RP:Wrk_Sys
2806/    1C9F : 2C 2B                              ld R2, #Strt_FreeProcess /256
2807/    1CA1 : 3C 5D                              ld R3, #Strt_FreeProcess #256
2808/    1CA3 : D6 02 CB                           call Bank_Call
2809/    1CA6 : 31 10              L1ca6           srp #Wrk_Sys
2810/    1CA8 :                                  assume RP:Wrk_Sys
2811/    1CA8 : 2C 10                              ld R2, #Start_Command /256
2812/    1CAA : 3C CE                              ld R3, #Start_Command #256
2813/    1CAC : D6 02 CB                           call Bank_Call
2814/    1CAF : 06 FF 02           L1caf           add SPL, #2
2815/    1CB2 : D6 1E AD                           call L1ead
2816/    1CB5 : 76 50 20                           tm 50h, #020h
2817/    1CB8 : 6B C6                              jr Z, L1c80
```

```
2818/    1CBA : 31 10                        srp #Wrk_Sys
2819/    1CBC :                      assume RP:Wrk_Sys
2820/    1CBC : 2C 1F                        ld R2, #L1FD1 /256
2821/    1CBE : 3C D1                        ld R3, #L1FD1 #256
2822/    1CC0 : D6 02 CB                     call Bank_Call
2823/    1CC3 : 06 FF 02             L1cc3   add SPL, #2
2824/    1CC6 : D6 1E AD                     call L1ead
2825/    1CC9 : 76 50 20                     tm 50h, #020h
2826/    1CCC : 6B B2                        jr Z, L1c80
2827/    1CCE : B0 51                        clr 51h
2828/    1CD0 : D6 1D 80             L1cd0   call L1d80
2829/    1CD3 : 76 51 04                     tm 51h, #004h
2830/    1CD6 : EB A8                        jr NZ, L1c80
2831/    1CD8 : 76 50 40                     tm 50h, #040h
2832/    1CDB : EB 0E                        jr NZ, L1ceb
2833/    1CDD : 76 51 10                     tm 51h, #010h
2834/    1CE0 : 6B EE                        jr Z, L1cd0
2835/    1CE2 : D6 1D 12                     call L1d12
2836/    1CE5 : D6 1E AD                     call L1ead
2837/    1CE8 : 8D 1C 80                     jp L1c80
2838/    1CEB : 31 10                L1ceb   srp #Wrk_Sys
2839/    1CED :                      assume RP:Wrk_Sys
2840/    1CED : 2C 1F                        ld R2, #L1fd1 /256
2841/    1CEF : 3C D1                        ld R3, #L1fd1 #256
2842/    1CF1 : D6 02 CB                     call Bank_Call
2843/    1CF4 : D6 1E AD             L1cf4   call L1ead
2844/    1CF7 : 8D 1C 80                     jp L1c80
2845/    1CFA :
2846/    1CFA : 06 FF 02             L1cfa   add SPL, #2
2847/    1CFD : 76 50 80                     tm 50h, #080h
2848/    1D00 : ED 1C 77                     jp NZ, L1c77
2849/    1D03 : 76 50 40                     tm 50h, #040h
2850/    1D06 : 6D 1C 77                     jp Z, L1c77
2851/    1D09 : FF                           nop
2852/    1D0A : FF                           nop
2853/    1D0B : FF                           nop
2854/    1D0C : D6 1E AD                     call L1ead
2855/    1D0F : 8D 1C 80                     jp L1c80
2856/    1D12 :
2857/    1D12 : 31 10                L1d12   srp #Wrk_Sys
2858/    1D14 :                      assume RP:Wrk_Sys
2859/    1D14 : E6 53 01                     ld 53h, #1
2860/    1D17 : 6C 10                        ld R6, #010h
2861/    1D19 : 7C 13                        ld R7, #013h
2862/    1D1B : 0C 7F                        ld R0, #07Fh
2863/    1D1D : 92 06                        lde @RR6, R0
2864/    1D1F : AF                           ret
2865/    1D20 :
2866/    1D20 : E6 50 10             L1d20   ld 50h, #010h
2867/    1D23 : B0 51                        clr 51h
2868/    1D25 : 31 50                        srp #050h
2869/    1D27 :                      assume RP:050h
2870/    1D27 : AF                           ret
2871/    1D28 :
2872/    1D28 : 56 02 EF             L1d28   and P2, #0FFh-010h
2873/    1D2B : 46 02 10                     or P2, #010h
2874/    1D2E : 31 10                        srp #Wrk_Sys
2875/    1D30 :                      assume RP:Wrk_Sys
2876/    1D30 : 46 02 02                     or P2, #002h
2877/    1D33 : 0C 1F                        ld R0, #01Fh
2878/    1D35 : 1C 70                        ld R1, #070h
2879/    1D37 : 82 20                        lde R2, @RR0
2880/    1D39 : 1C 72                        ld R1, #072h
2881/    1D3B : 82 20                        lde R2, @RR0
2882/    1D3D : 1C 74                        ld R1, #074h
2883/    1D3F : 82 20                        lde R2, @RR0
2884/    1D41 : 1C 76                        ld R1, #076h
2885/    1D43 : 82 20                        lde R2, @RR0
2886/    1D45 : 1C 7A                        ld R1, #07Ah
2887/    1D47 : 82 20                        lde R2, @RR0
2888/    1D49 : 1C 79                        ld R1, #079h
2889/    1D4B : 82 20                        lde R2, @RR0
2890/    1D4D : 1C 78                        ld R1, #078h
2891/    1D4F : 82 20                        lde R2, @RR0
2892/    1D51 : 1C 7D                        ld R1, #07Dh
2893/    1D53 : 82 20                        lde R2, @RR0
2894/    1D55 : B0 E4                        clr R4
2895/    1D57 : B0 E5                        clr R5
2896/    1D59 : 1C 7F                L1d59   ld R1, #07Fh
2897/    1D5B : 2C 1F                        ld R2, #01Fh
2898/    1D5D : 92 20                        lde @RR0, R2
2899/    1D5F : 1C 7E                        ld R1, #07Eh
2900/    1D61 : 82 20                        lde R2, @RR0
2901/    1D63 : 56 E2 1F                     and R2, #01Fh
2902/    1D66 : A6 E2 1F                     cp R2, #01Fh
2903/    1D69 : 6B 07                        jr Z, L1d72
2904/    1D6B : 80 E4                        decw RR4
2905/    1D6D : EB EA                        jr NZ, L1d59
2906/    1D6F : 46 51 80                     or 51h, #080h
2907/    1D72 : 1C 7C                L1d72   ld R1, #07Ch
2908/    1D74 : 82 20                        lde R2, @RR0
2909/    1D76 : 1C 79                        ld R1, #079h
2910/    1D78 : 82 20                        lde R2, @RR0
2911/    1D7A : 56 02 FD                     and P2, #0FFh-002h
2912/    1D7D : 31 50                        srp #050h
2913/    1D7F :                      assume RP:050h
2914/    1D7F : AF                           ret
2915/    1D80 :
2916/    1D80 : 31 10                L1d80   srp #Wrk_Sys
2917/    1D82 :                      assume RP:Wrk_Sys
2918/    1D82 : 46 02 02                     or P2, #002h
2919/    1D85 : 56 50 BF                     and 50h, #0BFh
2920/    1D88 : B0 E0                        clr R0
2921/    1D8A : B0 E1                        clr R1
2922/    1D8C : 2C 14                        ld R2, #014h
2923/    1D8E : 76 02 04             L1d8e   tm P2, #004h
2924/    1D91 : 6B 0F                        jr Z, L1da2
2925/    1D93 : 80 E0                        decw  RR0
2926/    1D95 : EB F7                        jr NZ, L1d8e
2927/    1D97 : 2A F5                        djnz R2, L1d8e
2928/    1D99 : 46 02 08             L1d99   or P2, #008h
2929/    1D9C : 56 02 FD                     and P2, #0FFh-002h
2930/    1D9F : 31 50                        srp #050h
2931/    1DA1 :                      assume RP:050h
2932/    1DA1 : AF                           ret
2933/    1DA2 :
2934/    1DA2 : 0C 1F                L1da2   ld R0, #01Fh
2935/    1DA4 : 1C 7C                        ld R1, #07Ch
```

```
2936/    1DA6 : 6C 10                        ld R6, #010h
2937/    1DA8 : 7C 1B                        ld R7, #01Bh
2938/    1DAA : 2C 80                        ld R2, #080h
2939/    1DAC : B0 E4                        clr R4
2940/    1DAE : 5C 19                        ld R5, #019h
2941/    1DB0 : 9C AA                        ld R9, #0AAh
2942/    1DB2 : B0 EF                        clr R15
2943/    1DB4 : E6 52 7F                     ld >52h, #07Fh
2944/    1DB7 : E6 53 7F                     ld >53h, #07Fh
2945/    1DBA : 56 02 F7                     and P2, #0FFh-008h
2946/    1DBD : 82 80                        lde R8, @RR0
2947/    1DBF : 82 80                        lde R8, @RR0
2948/    1DC1 : 82 80            L1dc1       lde R8, @RR0
2949/    1DC3 : A2 89                        cp R8, R9
2950/    1DC5 : 6B 07                        jr Z, L1dce
2951/    1DC7 : 76 02 04                     tm P2, #004h
2952/    1DCA : 6B F5                        jr Z, L1dc1
2953/    1DCC : 8B CB                        jr L1d99
2954/    1DCE : 82 80            L1dce       lde R8, @RR0
2955/    1DD0 : 42 88                        or R8, R8
2956/    1DD2 : DB FA                        jr PL, L1dce
2957/    1DD4 : 54 E8 52                     and >52h, R8
2958/    1DD7 : 82 80            L1dd7       lde R8, @RR0
2959/    1DD9 : 42 88                        or R8, R8
2960/    1DDB : DB FA                        jr PL, L1dd7
2961/    1DDD : 54 E8 53                     and >53h, R8
2962/    1DE0 : 82 80            L1de0       lde R8, @RR0
2963/    1DE2 : 42 88                        or R8, R8
2964/    1DE4 : DB FA                        jr PL, L1de0
2965/    1DE6 : FF                           nop
2966/    1DE7 : FF                           nop
2967/    1DE8 : FF                           nop
2968/    1DE9 : FF                           nop
2969/    1DEA : FF                           nop
2970/    1DEB : 8B 0F                        jr L1dfc
2971/    1DED : 82 80            L1ded       lde R8, @RR0
2972/    1DEF : 42 88                        or R8, R8
2973/    1DF1 : DB FA                        jr PL, L1ded
2974/    1DF3 : 10 EF                        rlc R15
2975/    1DF5 : 93 56                        ldei @RR6, @R5
2976/    1DF7 : 5C 19                        ld R5, #019h
2977/    1DF9 : FF                           nop
2978/    1DFA : 02 4F                        add R4, R15
2979/    1DFC : 82 90            L1dfc       lde R9, @RR0
2980/    1DFE : C0 E8                        rrc R8
2981/    1E00 : 10 E9                        rlc R9
2982/    1E02 : 02 49                        add R4, R9
2983/    1E04 : FF                           nop
2984/    1E05 : 93 56                        ldei @RR6, @R5
2985/    1E07 : C0 E8                        rrc R8
2986/    1E09 : 82 A0                        lde R10, @RR0
2987/    1E0B : 10 EA                        rlc R10
2988/    1E0D : 02 4A                        add R4, R10
2989/    1E0F : 93 56                        ldei @RR6, @R5
2990/    1E11 : FF                           nop
2991/    1E12 : F0 54                        swap 54h
2992/    1E14 : C0 E8                        rrc R8
2993/    1E16 : 82 B0                        lde R11, @RR0
2994/    1E18 : 10 EB                        rlc R11
2995/    1E1A : 02 4B                        add R4, R11
2996/    1E1C : 93 56                        ldei @RR6, @R5
2997/    1E1E : F0 54                        swap 54h
2998/    1E20 : A8 52                        ld R10,>52h
2999/    1E22 : C0 E8                        rrc R8
3000/    1E24 : 82 C0                        lde R12, @RR0
3001/    1E26 : 10 EC                        rlc R12
3002/    1E28 : 02 4C                        add R4, R12
3003/    1E2A : 93 56                        ldei @RR6, @R5
3004/    1E2C : F0 54                        swap 54h
3005/    1E2E : 00 52                        dec 52h
3006/    1E30 : C0 E8                        rrc R8
3007/    1E32 : 82 D0                        lde R13, @RR0
3008/    1E34 : 10 ED                        rlc R13
3009/    1E36 : 02 4D                        add R4, R13
3010/    1E38 : 93 56                        ldei @RR6, @R5
3011/    1E3A : AA 17                        djnz R10, L1e53
3012/    1E3C : F0 54                        swap 54h
3013/    1E3E : 82 E0                        lde R14, @RR0
3014/    1E40 : C0 E8                        rrc R8
3015/    1E42 : 10 EE                        rlc R14
3016/    1E44 : 02 4E                        add R4, R14
3017/    1E46 : 93 56                        ldei @RR6, @R5
3018/    1E48 : 3C 02                        ld R3, #2
3019/    1E4A : 73 23                        tm R2, @R3
3020/    1E4C : 82 F0                        lde R15, @RR0
3021/    1E4E : 46 02 08                     or P2, #008h
3022/    1E51 : 8B 3C                        jr L1e8f
3023/    1E53 : C0 E8            L1e53       rrc R8
3024/    1E55 : 82 E0                        lde R14, @RR0
3025/    1E57 : 10 EE                        rlc R14
3026/    1E59 : 02 4E                        add R4, R14
3027/    1E5B : 93 56                        ldei @RR6, @R5
3028/    1E5D : 3C 02                        ld R3, #002h
3029/    1E5F : C0 E8                        rrc R8
3030/    1E61 : 73 23                        tm R2, @R3
3031/    1E63 : 82 F0                        lde R15, @RR0
3032/    1E65 : 6D 1D ED                     jp Z, L1ded
3033/    1E68 : 46 02 08                     or P2, #008h
3034/    1E6B : 10 E8                        rlc R8
3035/    1E6D : 9C AA                        ld R9, #0AAh
3036/    1E6F : 73 23            L1e6f       tm R2, @R3
3037/    1E71 : 6B 0A                        jr Z, L1e7d
3038/    1E73 : 76 02 04                     tm P2, #004h
3039/    1E76 : 6B F7                        jr Z, L1e6f
3040/    1E78 : 46 51 04                     or >51h, #004h
3041/    1E7B : 8B 22                        jr L1e9f
3042/    1E7D : 56 02 F7         L1e7d       and P2, #0F7h
3043/    1E80 : 82 A0                        lde R10, @RR0
3044/    1E82 : 82 A0                        lde R10, @RR0
3045/    1E84 : 82 A0            L1e84       lde R10, @RR0
3046/    1E86 : A2 A9                        cp R10, R9
3047/    1E88 : EB FA                        jr NZ, L1e84
3048/    1E8A : C0 E8                        rrc R8
3049/    1E8C : 8D 1D ED                     jp L1ded
3050/    1E8F : C0 E8            L1e8f       rrc R8
3051/    1E91 : 10 EF                        rlc R15
3052/    1E93 : 02 4F                        add R4, R15
3053/    1E95 : 6B 05                        jr Z, L1e9c
```

```
3054/    1E97 : 46 51 10                        or >51h, #010h
3055/    1E9A : 8B 03                           jr L1e9f
3056/    1E9C : 46 50 40           L1e9c        or >50h, #040h
3057/    1E9F : 76 02 04           L1e9f        tm P2, #004h
3058/    1EA2 : 6B FB                           jr Z, L1e9f
3059/    1EA4 : 46 02 08                        or P2, #008h
3060/    1EA7 : 56 02 FD                        and P2, #0FFh-002h
3061/    1EAA : 31 50                           srp #050h
3062/    1EAC :                          assume RP:050h
3063/    1EAC : AF                              ret
3064/    1EAD :
3065/    1EAD : 31 10              L1ead        srp #Wrk_Sys
3066/    1EAF :                          assume RP:Wrk_Sys
3067/    1EAF : 56 50 DF                        and 50h, #0FFh-020h
3068/    1EB2 : 46 02 02                        or P2, #002h
3069/    1EB5 : 56 02 F7                        and P2, #0FFh-008h
3070/    1EB8 : 76 02 04           L1eb8        tm P2, #004h
3071/    1EBB : EB FB                           jr NZ, L1eb8
3072/    1EBD : 0C 1F                           ld R0, #01Fh
3073/    1EBF : 1C 7D                           ld R1, #07Dh
3074/    1EC1 : 82 30                           lde R3, @RR0
3075/    1EC3 : 1C 7F                           ld R1, #07Fh
3076/    1EC5 : 6C 10                           ld R6, #010h
3077/    1EC7 : 7C 13                           ld R7, #013h
3078/    1EC9 : 28 53                           ld R2, 53h
3079/    1ECB : 3C 02                           ld R3, #002h
3080/    1ECD : B0 E4                           clr R4
3081/    1ECF : 5C 18                           ld R5, #018h
3082/    1ED1 : 83 56                           ldei @R5, @RR6
3083/    1ED3 : 9C AA                           ld R9, #0AAh
3084/    1ED5 : 92 90                           lde @RR0, R9
3085/    1ED7 : 8B 03                           jr L1edc
3086/    1ED9 : 46 E4 00           L1ed9        or R4, #0
3087/    1EDC : 02 48              L1edc        add R4, R8
3088/    1EDE : DF                              scf
3089/    1EDF : C0 E8                           rrc R8
3090/    1EE1 : C0 EF                           rrc R15
3091/    1EE3 : 92 80                           lde @RR0, R8
3092/    1EE5 : 46 E4 00                        or R4, #0
3093/    1EE8 : 83 56                           ldei @R5, @RR6
3094/    1EEA : 02 49                           add R4, R9
3095/    1EEC : DF                              scf
3096/    1EED : C0 E9                           rrc R9
3097/    1EEF : C0 EF                           rrc R15
3098/    1EF1 : 92 90                           lde @RR0, R9
3099/    1EF3 : 46 E4 00                        or R4, #0
3100/    1EF6 : 83 56                           ldei @R5, @RR6
3101/    1EF8 : 02 4A                           add R4, R10
3102/    1EFA : DF                              scf
3103/    1EFB : C0 EA                           rrc R10
3104/    1EFD : C0 EF                           rrc R15
3105/    1EFF : 92 A0                           lde @RR0, R10
3106/    1F01 : 46 E4 00                        or R4, #0
3107/    1F04 : 83 56                           ldei @R5, @RR6
3108/    1F06 : 02 4B                           add R4, R11
3109/    1F08 : DF                              scf
3110/    1F09 : C0 EB                           rrc R11
3111/    1F0B : C0 EF                           rrc R15
3112/    1F0D : 92 B0                           lde @RR0, R11
3113/    1F0F : FF                              nop
3114/    1F10 : 83 56                           ldei @R5, @RR6
3115/    1F12 : 02 4C                           add R4, R12
3116/    1F14 : DF                              scf
3117/    1F15 : C0 EC                           rrc R12
3118/    1F17 : C0 EF                           rrc R15
3119/    1F19 : 92 C0                           lde @RR0, R12
3120/    1F1B : FF                              nop
3121/    1F1C : 83 56                           ldei @R5, @RR6
3122/    1F1E : 02 4D                           add R4, R13
3123/    1F20 : DF                              scf
3124/    1F21 : C0 ED                           rrc R13
3125/    1F23 : C0 EF                           rrc R15
3126/    1F25 : 92 D0                           lde @RR0, R13
3127/    1F27 : 00 E2                           dec R2
3128/    1F29 : EB 21                           jr NZ, L1f4c
3129/    1F2B : F0 EE                           swap R14
3130/    1F2D : E8 E4                           ld R14, R4
3131/    1F2F : 60 E4                           com R4
3132/    1F31 : 4E                              inc R4
3133/    1F32 : DF                              scf
3134/    1F33 : C0 E4                           rrc R4
3135/    1F35 : C0 EF                           rrc R15
3136/    1F37 : 92 40                           lde @RR0, R4
3137/    1F39 : C0 EF                           rrc R15
3138/    1F3B : 46 EF 80                        or R15, #080h
3139/    1F3E : 46 E4 00                        or R4, #0
3140/    1F41 : 46 E4 00                        or R4, #0
3141/    1F44 : 46 E4 00                        or R4, #0
3142/    1F47 : FF                              nop
3143/    1F48 : 92 F0                           lde @RR0, R15
3144/    1F4A : 8B 60                           jr L1fac
3145/    1F4C : 83 56              L1f4c        ldei @R5, @RR6
3146/    1F4E : 02 4E                           add R4, R14
3147/    1F50 : DF                              scf
3148/    1F51 : C0 EE                           rrc R14
3149/    1F53 : C0 EF                           rrc R15
3150/    1F55 : 92 E0                           lde @RR0, R14
3151/    1F57 : 5C 18                           ld R5, #018h
3152/    1F59 : 83 56                           ldei @R5, @RR6
3153/    1F5B : C0 EF                           rrc R15
3154/    1F5D : 46 EF 80                        or R15, #080h
3155/    1F60 : FF                              nop
3156/    1F61 : 9C 80                           ld R9, #080h
3157/    1F63 : 92 F0                           lde @RR0, R15
3158/    1F65 : 73 93                           tm R9, @R3
3159/    1F67 : 6D 1E D9                        jp Z, L1ed9
3160/    1F6A : 46 02 08                        or P2, #008h
3161/    1F6D : 46 02 08                        or P2, #008h
3162/    1F70 : 46 02 08                        or P2, #008h
3163/    1F73 : B0 EA                           clr R10
3164/    1F75 : 92 A0                           lde @RR0, R10
3165/    1F77 : 46 02 08                        or P2, #008h
3166/    1F7A : 46 02 08                        or P2, #008h
3167/    1F7D : 46 02 08                        or P2, #008h
3168/    1F80 : 46 02 08                        or P2, #008h
3169/    1F83 : 1C 7E                           ld R1, #07Eh
3170/    1F85 : 82 A0                           lde R10, @RR0
3171/    1F87 : 1C 7C                           ld R1, #07Ch
```

```
3172/    1F89 : 82 A0                                    lde R10, @RR0
3173/    1F8B : 73 93                    L1f8b           tm R9, @R3
3174/    1F8D : 6B 0A                                    jr Z, L1f99
3175/    1F8F : 76 02 04                                 tm P2, #004h
3176/    1F92 : 6B F7                                    jr Z, L1f8b
3177/    1F94 : 46 51 04                                 or 51h, #004h
3178/    1F97 : 8B 2D                                    jr L1fc6
3179/    1F99 : 56 02 F7                 L1f99           and P2, #0F7h
3180/    1F9C : 1C 7D                                    ld R1, #07Dh
3181/    1F9E : 82 A0                                    lde R10, @RR0
3182/    1FA0 : 1C 7F                                    ld R1, #07Fh
3183/    1FA2 : FF                                       nop
3184/    1FA3 : FF                                       nop
3185/    1FA4 : FF                                       nop
3186/    1FA5 : AC AA                                    ld R10, #0AAh
3187/    1FA7 : 92 A0                                    lde @RR0, R10
3188/    1FA9 : 8D 1E DC                                 jp L1edc
3189/    1FAC : 46 02 08                 L1fac           or P2, #008h
3190/    1FAF : B0 EA                                    clr R10
3191/    1FB1 : FF                                       nop
3192/    1FB2 : FF                                       nop
3193/    1FB3 : FF                                       nop
3194/    1FB4 : FF                                       nop
3195/    1FB5 : 92 A0                                    lde @RR0, R10
3196/    1FB7 : 1C 7C                                    ld R1, #07Ch
3197/    1FB9 : 82 A0                                    lde R10, @RR0
3198/    1FBB : 46 50 20                                 or 50h, #020h
3199/    1FBE : B0 E2                                    clr R2
3200/    1FC0 : FF                                       nop
3201/    1FC1 : FF                                       nop
3202/    1FC2 : 1C 7E                                    ld R1, #07Eh
3203/    1FC4 : 82 A0                                    lde R10, @RR0
3204/    1FC6 : 76 02 04                 L1fc6           tm P2, #004h
3205/    1FC9 : 6B FB                                    jr Z, L1fc6
3206/    1FCB : 56 02 FD                                 and P2, #0FFh-002h
3207/    1FCE : 31 50                                    srp #050h
3208/    1FD0 :                                  assume RP:050h
3209/    1FD0 : AF                                       ret
3210/    1FD1 :
3211/    1FD1 : D6 05 4F                 L1fd1           call Ext_Pop
3212/    1FD4 : 76 3E 10                                 tm 3Eh, #010h
3213/    1FD7 : 6D 01 E2                                 jp Z, L01e2
3214/    1FDA : 76 3E 02                                 tm 3Eh, #002h
3215/    1FDD : 6B 15                                    jr Z, L1ff4
3216/    1FDF : 2C 10                                    ld R2, #010h
3217/    1FE1 : 3C 1B                                    ld R3, #01Bh
3218/    1FE3 : 82 02                                    lde R0, @RR2
3219/    1FE5 : 76 E0 40                                 tm R0, #040h
3220/    1FE8 : 6D 01 E7                                 jp Z, L01e7
3221/    1FEB : A0 E2                                    incw RR2
3222/    1FED : 82 02                                    lde R0, @RR2
3223/    1FEF : 06 E3 04                                 add R3, #004h
3224/    1FF2 : 82 12                                    lde R1, @RR2
3225/    1FF4 : 50 E2                    L1ff4           pop R2
3226/    1FF6 : 50 E2                                    pop R2
3227/    1FF8 : AF                                       ret
3228/    1FF9 :
3229/    1FF9 : FF FF FF FF FF FF                        DB 02000h-$ dup(0FFh)   ; pad with $FF's
                FF
3230/    2000 :                          ;===================================================================
3231/    2000 :                          ;===================================================================
3232/    2000 :
3233/    2000 :                          ;*******************************************************************
3234/    2000 :                          ; Module Bank1.Assem
3235/    2000 :                          ;
3236/    2000 :                          ;*******************************************************************
3237/    2000 :                                  org 1000h
3238/    2000 :                                  phase 2000h             ; EPROM will start at 4k so 4k 01000h offset
3239/    2000 :
3240/    2000 : B7 4D                                    DW Checksum1
3241/    2002 : 01                                       DB 1
3242/    2003 : F0 78                                    DW Passwrd1
3243/    2005 : 3C 1E                                    DW Passwrd2
3244/    2007 : 01 31                                    DB 001h, 031h
3245/    2009 :
3246/    2009 :
3247/    2009 :                          ;*******************************************************************
3248/    2009 :                          ; Module RdHdr.B1.Assem
3249/    2009 :                          ;
3250/    2009 :                          ;  This module contains all but the very most primitive of of
3251/    2009 :                          ;  procedures (the routines that are resident are listed in the
3252/    2009 :                          ;  external spec's) that concern themselves with performing a read
3253/    2009 :                          ;  operation.
3254/    2009 :                          ;
3255/    2009 :                          ;       FUNCTION ReadHdr( Parent : BYTE {R8} ) :
3256/    2009 :                          ;               BOOLEAN
3257/    2009 :                          ;               Status : BYTE {R0}
3258/    2009 :                          ;
3259/    2009 :                          ;*******************************************************************
3260/    2009 :
3261/    2009 :                          ;*******************************************************************
3262/    2009 :                          ;
3263/    2009 :                          ; Function: ReadHdr
3264/    2009 :                          ;
3265/    2009 :                          ;  This function assumes that the heads are positioned over the
3266/    2009 :                          ;  correct track and reads the block of data following the
3267/    2009 :                          ;  next sector mark. This routine is used for two reasons: 1) to
3268/    2009 :                          ;  verify the header that is laid out on a block, and 2) to try
3269/    2009 :                          ;  to recover the data within a block if the header is munched.
3270/    2009 :                          ;
3271/    2009 :                          ; Inputs: Parent: BYTE {R8}
3272/    2009 :                          ;
3273/    2009 :                          ; Outputs: ReadHdr: BOOLEAN {zero flag, true if error in ReadHdr}
3274/    2009 :                          ;          Status:  BYTE {R0}
3275/    2009 :                          ;
3276/    2009 :                          ; Global Variables Used: Cylinder, Head, Sector
3277/    2009 :                          ;
3278/    2009 :                          ; Local Variables Used: RdHError:  BOOLEAN {R9/bit 7}
3279/    2009 :                          ;                       RdHExcept: BOOLEAN {R9/bit 6}
3280/    2009 :                          ;
3281/    2009 :                          ; Algorithm:
3282/    2009 :                          ;
3283/    2009 :                          ;  Begin
3284/    2009 :                          ;    SetDeadManTimer(ReadBlock, Parent)
3285/    2009 :                          ;    RdHError:=false
3286/    2009 :                          ;    RdHExcept:=false
3287/    2009 :                          ;    RdHSctrGap:=0
3288/    2009 :                          ;    RdHHdrGap:=0
```

```
3289/    2009 :                     ;   /-
3290/    2009 :                     ; R | Set-up external RAM address counter for read header
3291/    2009 :                     ; E | Msel0:1:=Disk<-->Mem
3292/    2009 :                     ; S | While SectorMark Do Begin End
3293/    2009 :                     ; I | StartL:=true
3294/    2009 :                     ; D | While not(SectorDnL) Do Begin End
3295/    2009 :                     ; E | Status:=StatusPort
3296/    2009 :                     ; N | StartL:=false
3297/    2009 :                     ; T | Msel0:1:=Z8<-->Mem
3298/    2009 :                     ;   \-
3299/    2009 :                     ;    If Status.State<>NormalEndState
3300/    2009 :                     ;      Then
3301/    2009 :                     ;        Reset_StateMachine
3302/    2009 :                     ;        Abort
3303/    2009 :                     ;    If Status.ServoErr or not(Status.ServoRdy)
3304/    2009 :                     ;      Then
3305/    2009 :                     ;        RdHError:=true
3306/    2009 :                     ;        RdHExcept:=true
3307/    2009 :                     ;    If Status.CrcErrL then RdHError:=true
3308/    2009 :                     ;    ClearDeadManTimer
3309/    2009 :                     ;    Status/bit7:=RdHError
3310/    2009 :                     ;    Status/bit7:=RdHExcept
3311/    2009 :                     ;  End
3312/    2009 :                     ;
3313/    2009 :                     ;*****************************************************************
3314/    2009 :
3315/    2009 : 56 35 DF            ReadHdr         and DiskStat, #0FFh-Wr_Op       ; make sure we are reading
3316/    200C : B0 34                               clr RWStat              ; clear booleans
3317/    200E : B0 19                               clr 19h
3318/    2010 : D6 04 9B                            call L049b
3319/    2013 : D6 02 03                            call L0203
3320/    2016 : 18 E5                               ld R1, R5               ; If Status.State
3321/    2018 : 56 E1 0F                            and R1, #YMask
3322/    201B : A6 E1 02                            cp R1, #Norm_State
3323/    201E : 6B 0D                               jr Z, RdHdr_Norm
3324/    2020 :                     ;
3325/    2020 : 98 E1                               ld R9, R1
3326/    2022 : 46 E9 10                            or R9, #010h
3327/    2025 : D6 03 C8                            call Reset_StMach
3328/    2028 : 46 34 88                            or RWStat, #RdHError+RdSMErr
3329/    202B : 8B 19                               jr RdH_End
3330/    202D :                     ;
3331/    202D : 76 E5 10            RdHdr_Norm      tm R5, #ServoErr        ; If ServoErr or not(ServoRdy)
3332/    2030 : EB 05                               jr nz, RdHd_ServoErr
3333/    2032 : 76 E5 20                            tm R5, #ServoRdy
3334/    2035 : EB 05                               jr nz, RdHd_SrvoOk
3335/    2037 :                     ;
3336/    2037 : 46 34 C0            RdHd_ServoErr   or RWStat, #RdHError+RdHSrvoErr ; Then RdHError and RdHServoErr
3337/    203A : 8B 0A                               jr RdH_End
3338/    203C :                     ;
3339/    203C : 76 E5 80            RdHd_SrvoOk     tm R5, #CrcErrL         ; If Status.CrcErr
3340/    203F : EB 05                               jr nz, RdH_End
3341/    2041 : 46 34 90                            or RWStat, #RdHError+RdCrcErr
3342/    2044 : 8B 00                               jr RdH_End
3343/    2046 :                     ;
3344/    2046 : 66 34 80            RdH_End         tcm RWStat, #RdHError   ; set zero flag if error
3345/    2049 : 8D 02 F6                            jp Bank_Ret
3346/    204C :
3347/    204C :
3348/    204C :
3349/    204C :                     ;*****************************************************************
3350/    204C :                     ; Module Utils.B1
3351/    204C :                     ;
3352/    204C :                     ; This module contains routines that are part of the main utility
3353/    204C :                     ; module and that must be kept in Bank 1.
3354/    204C :                     ;
3355/    204C :                     ;       PROCEDURE RBuf_To_Buf2
3356/    204C :                     ;       PROCEDURE Buf2_To_RBuf
3357/    204C :                     ;       PROCEDURE WrBuf_To_Buf2
3358/    204C :                     ;       PROCEDURE Buf2_To_WrBuf
3359/    204C :                     ;       PROCEDURE RBuf_To_Spr
3360/    204C :                     ;       PROCEDURE Spr_To_RBuf
3361/    204C :                     ;       PROCEDURE WrBuf_To_Spr
3362/    204C :                     ;       PROCEDURE Spr_To_WrBuf
3363/    204C :                     ;       PROCEDURE Zero_RdBuf
3364/    204C :                     ;       PROCEDURE Zero_Fmt
3365/    204C :                     ;       FUNCTION Get_Cyl_H_S( PhysicalBlock : 3 BYTES {RR12} ) :
3366/    204C :                     ;               Cylinder : WORD {RR12}
3367/    204C :                     ;               Head : BYTE {R14}
3368/    204C :                     ;               Sector : BYTE {R15}
3369/    204C :                     ;       FUNCTION GoodHdr : BOOLEAN
3370/    204C :                     ;               Cylinder : WORD {RR0}
3371/    204C :                     ;               Head : BYTE {R2}
3372/    204C :                     ;               Sector : BYTE {R3}
3373/    204C :                     ;       PROCEDURE BlockMove( Destination : PTR {RR2}
3374/    204C :                     ;               Source : PTR {RR0})
3375/    204C :                     ;               FUNCTION UpDate_Hdr : BOOLEAN
3376/    204C :                     ;               FUNCTION Get_Type( DriverType : BYTE {R8} ) :
3377/    204C :                     ;>                      BlockType : 3 BITS {R8/bits 3:1}
3378/    204C :                     ;>                      BlockNumber : 3 BYTES {R12:14}
3379/    204C :                     ;>
3380/    204C :                     ;*****************************************************************
3381/    204C :
3382/    204C :                     ;*****************************************************************
3383/    204C :                     ;
3384/    204C :                     ; Procedures:   Ld_Stand_Stat  {move standard status to output buffer}
3385/    204C :                     ;               RBuf_To_Buf2   {move ReadBuffer to Buffer2}
3386/    204C :                     ;               Buf2_To_RBuf   {move Buffer2 to ReadBuffer}
3387/    204C :                     ;               WrBuf_To_Buf2  {move WriteBuffer to Buffer2}
3388/    204C :                     ;               Buf2_To_WrBuf  {move Buffer2 to WriteBuffer}
3389/    204C :                     ;               RBuf_To_Spr    {move ReadBuffer to SpareArray}
3390/    204C :                     ;               pr_To_RBuf     {move SpareArray to ReadBuffer}
3391/    204C :                     ;               Zero_RdBuf     {move zeros to ReadBuffer}
3392/    204C :                     ;
3393/    204C :                     ; Inputs: none
3394/    204C :                     ;
3395/    204C :                     ; Outputs: none
3396/    204C :                     ;
3397/    204C :                     ;*****************************************************************
3398/    204C :
3399/    204C : 0C 12               Buf2_To_RBuf    ld R0, #Buf2Array /256
3400/    204E : 1C 66                               ld R1, #Buf2Array #256
3401/    2050 : 2C 10                               ld R2, #RDummy /256
3402/    2052 : 3C 18                               ld R3, #RDummy #256
3403/    2054 : 8B 41                               jr B_Move
3404/    2056 :                     ;
3405/    2056 : 76 35 04            WrBuf_To_Buf2   tm DiskStat, #User_Type ; nop if sparetable data
3406/    2059 : 6B 3F                               jr Z, Return_Vector
```

```
3407/    205B : 0C 10                              ld R0, #(WBuffer1-1) /256
3408/    205D : 1C 20                              ld R1, #(WBuffer1-1) #256
3409/    205F : 8B 32                              jr X_To_Buf2
3410/    2061 :                          ;
3411/    2061 : 76 35 04                 Buf2_To_WrBuf   tm DiskStat, #User_Type
3412/    2064 : 6B 37                              jr Z, Spr_To_WrBuf
3413/    2066 : 0C 12                              ld R0, #Buf2Array /256
3414/    2068 : 1C 66                              ld R1, #Buf2Array #256
3415/    206A : 2C 10                              ld R2, #(WBuffer1-1) /256
3416/    206C : 3C 20                              ld R3, #(WBuffer1-1) #256
3417/    206E : 8B 27                              jr B_Move
3418/    2070 :                          ;
3419/    2070 : 0C 14                    Spr_To_RBuf     ld R0, #SpareArray /256
3420/    2072 : 1C 95                              ld R1, #SpareArray #256
3421/    2074 : 2C 10                              ld R2, #RBuffer1 /256
3422/    2076 : 3C 19                              ld R3, #RBuffer1 #256
3423/    2078 : 8B 1D                              jr B_Move
3424/    207A :                          ;
3425/    207A : 0C 10                    RBuf_To_Spr     ld R0, #RBuffer1 /256
3426/    207C : 1C 19                              ld R1, #RBuffer1 #256
3427/    207E : 2C 14                    X_To_Spr        ld R2, #SpareArray /256
3428/    2080 : 3C 95                              ld R3, #SpareArray #256
3429/    2082 : 8B 13                              jr B_Move
3430/    2084 :                          ;
3431/    2084 : 0C 10                    WrBuf_To_Spr    ld R0, #WBuffer1 /256
3432/    2086 : 1C 21                              ld R1, #WBuffer1 #256
3433/    2088 : 8B F4                              jr X_To_Spr
3434/    208A :                          ;
3435/    208A : 76 35 04                 RBuf_To_Buf2    tm DiskStat, #User_Type ; nop if sparetable data
3436/    208D : 6B 0B                              jr Z, Return_Vector
3437/    208F : 0C 10                              ld R0, #RDummy /256
3438/    2091 : 1C 18                              ld R1, #RDummy #256
3439/    2093 : 2C 12                    X_To_Buf2       ld R2, #Buf2Array /256
3440/    2095 : 3C 66                              ld R3, #Buf2Array #256
3441/    2097 : D6 21 0B                 B_Move          call BlockMove
3442/    209A : 8D 02 F6                 Return_Vector   jp Bank_Ret
3443/    209D :                          ;
3444/    209D : 0C 14                    Spr_To_WrBuf    ld R0, #SpareArray /256
3445/    209F : 1C 95                              ld R1, #SpareArray #256
3446/    20A1 : 2C 10                              ld R2, #WBuffer1 /256
3447/    20A3 : 3C 21                              ld R3, #WBuffer1 #256
3448/    20A5 : 8B F0                              jr B_Move
3449/    20A7 :
3450/    20A7 :
3451/    20A7 :                          ;*********************************************************************
3452/    20A7 :                          ;
3453/    20A7 :                          ; Function: UpDate_Cur_Cyl
3454/    20A7 :                          ;
3455/    20A7 :                          ;  This function is responsible for reading the header off of
3456/    20A7 :                          ;  an arbitrary sector. The cylinder information is extracted
3457/    20A7 :                          ;  from the header and placed into the current cylinder memory
3458/    20A7 :                          ;  locations.
3459/    20A7 :                          ;
3460/    20A7 :                          ; Inputs: none
3461/    20A7 :                          ;
3462/    20A7 :                          ; Outputs: UpDate_Cur_Cyl: BOOLEAN {zero flag is set if not(ReadHdr)}
3463/    20A7 :                          ;
3464/    20A7 :                          ; Global Variables Changed: Cur_Cylinder
3465/    20A7 :                          ;
3466/    20A7 :                          ; Algorithm:
3467/    20A7 :                          ;
3468/    20A7 :                          ;  Begin
3469/    20A7 :                          ;    Temp:=GoodHdr(ReadHdr)
3470/    20A7 :                          ;    If Temp
3471/    20A7 :                          ;      Then Cur_Cylinder:=ReadHdr.Cylinder
3472/    20A7 :                          ;    UpDate_Cur_Cyl:=Temp
3473/    20A7 :                          ;  End
3474/    20A7 :                          ;
3475/    20A7 :                          ;*********************************************************************
3476/    20A7 :
3477/    20A7 : D6 20 B6                 UpDate_Cur_Cyl  call GoodHdr
3478/    20AA : 6B 07                              jr Z, UD_C_C_End
3479/    20AC : 09 38                              ld Cur_Cyl, R0
3480/    20AE : 19 39                              ld Cur_Cyl+1, R1
3481/    20B0 : 46 E1 01                           or R1, #1               ; return non-zero status
3482/    20B3 : 8D 02 F6                 UD_C_C_End      jp Bank_Ret
3483/    20B6 :
3484/    20B6 :
3485/    20B6 :                          ;*********************************************************************
3486/    20B6 :                          ;
3487/    20B6 :                          ; Function: GoodHdr
3488/    20B6 :                          ;
3489/    20B6 :                          ;  This function is responsible for reading the header off of a
3490/    20B6 :                          ;  sector and checking that header for validity, i.e. whether
3491/    20B6 :                          ;  the cylinder, head, and sector (and their complements) look
3492/    20B6 :                          ;  reasonable.
3493/    20B6 :                          ;
3494/    20B6 :                          ; Inputs: none
3495/    20B6 :                          ;
3496/    20B6 :                          ; Outputs: GoodHdr: BOOLEAN {zero flag is set if header invalid}
3497/    20B6 :                          ;          GoodHdr.Cylinder: WORD {RR0}
3498/    20B6 :                          ;          GoodHdr.Head:     BYTE {R2}
3499/    20B6 :                          ;          GoodHdr.Sector:   BYTE {R3}
3500/    20B6 :                          ;
3501/    20B6 :                          ; Algorithm:
3502/    20B6 :                          ;
3503/    20B6 :                          ;  Begin
3504/    20B6 :                          ;    ReadHdr
3505/    20B6 :                          ;    If inverse(ReadHdr.Cylinder)=ReadHdr.InverseCylinder and
3506/    20B6 :                          ;       inverse(ReadHdr.HdSctr)=ReadHdr.InverseHdSctr
3507/    20B6 :                          ;     Then
3508/    20B6 :                          ;       GoodHdr:=true
3509/    20B6 :                          ;       GoodHdr.Cylinder:=ReadHdr.Cylinder
3510/    20B6 :                          ;       GoodHdr.Head:=ReadHdr.HdSctr/bits 7:6
3511/    20B6 :                          ;       GoodHdr.Sector:=ReadHdr.HdSctr/bits 5:0
3512/    20B6 :                          ;     Else
3513/    20B6 :                          ;       GoodHdr:=false
3514/    20B6 :                          ;  End
3515/    20B6 :                          ;
3516/    20B6 :                          ;*********************************************************************
3517/    20B6 :
3518/    20B6 : 4C 08                    GoodHdr         ld R4, #8               ; try hard to find a good header
3519/    20B8 : D6 20 09                 GdHdr_1         call ReadHdr
3520/    20BB : 2C 10                              ld R2, #RHHeader /256
3521/    20BD : 3C 0C                              ld R3, #RHHeader #256
3522/    20BF : 0C 40                              ld R0, #ScrReg0
3523/    20C1 : 1C 06                              ld R1, #6               ; load 6 bytes
3524/    20C3 : 83 02                    GdHdr_Lp        ldei @R0, @RR2
```

```
3525/    20C5 : 1A FC                           djnz R1, GdHdr_Lp
3526/    20C7 : 31 40                           srp #Wrk_Scr            ; context switch
3527/    20C9 :                         assume RP:Wrk_Scr
3528/    20C9 : B2 30                           xor R3, R0              ; check for valid header
3529/    20CB : B2 41                           xor R4, R1
3530/    20CD : B2 52                           xor R5, R2
3531/    20CF : 52 34                           and R3, R4
3532/    20D1 : 52 35                           and R3, R5
3533/    20D3 : 60 E3                           com R3
3534/    20D5 : 31 10                           srp #Wrk_Sys            ; context switch
3535/    20D7 :                         assume RP:Wrk_Sys
3536/    20D7 : B0 46                           clr ScrReg6             ; assume bad header
3537/    20D9 : 6B 13                           jr Z, GdHdr_2
3538/    20DB : 76 32 80                        tm Excpt_Stat, #Recovery        ; check for Recovery on
3539/    20DE : 6B 25                           jr Z, GdHdr_End
3540/    20E0 : D6 27 30                        call ChkOff_NoOff       ; set auto-offset if not already on
3541/    20E3 : 2C 00                           ld R2, #10 /256         ; wait 100ms before retrying
3542/    20E5 : 3C 0A                           ld R3, #10 #256
3543/    20E7 : D6 01 24                        call MsWait
3544/    20EA : 4A CC                           djnz R4, GdHdr_1        ; retry if bad header
3545/    20EC : 8B 17                           jr GdHdr_End
3546/    20EE :                         ; found a good one
3547/    20EE : 08 40                   GdHdr_2         ld R0, ScrReg0          ; GoodHdr.Cylinder:=ReadHdr.Cylinder
3548/    20F0 : 18 41                           ld R1, ScrReg1
3549/    20F2 : 28 42                           ld R2, ScrReg2          ; GoodHdr.Head:=ReadHdr.HdSctr/bits 7:6
3550/    20F4 : F0 E2                           swap R2
3551/    20F6 : E0 E2                           rr R2
3552/    20F8 : E0 E2                           rr R2
3553/    20FA : 56 E2 03                        and R2, #3              ; just leave head info in register
3554/    20FD : 38 42                           ld R3, ScrReg2          ; GoodHdr.Sectror:=ReadHdr.HdSctr/bits 5:0
3555/    20FF : 56 E3 3F                        and R3, #3Fh
3556/    2102 : E6 46 01                        ld ScrReg6, #1
3557/    2105 :                         ;
3558/    2105 : 44 46 46                GdHdr_End       or ScrReg6, ScrReg6     ; set zero flag
3559/    2108 : 8D 02 F6                        jp Bank_Ret
3560/    210B :
3561/    210B :
3562/    210B :                         ;*********************************************************************
3563/    210B :                         ;
3564/    210B :                         ; Procedure: BlockMove
3565/    210B :                         ;
3566/    210B :                         ;  This procedure performs the following:
3567/    210B :                         ;
3568/    210B :                         ;  (Destination) <-- (Source)
3569/    210B :                         ;
3570/    210B :                         ;  where destination and source are both the same size (namely
3571/    210B :                         ;  exactly 524 bytes plus CRC and ECC).
3572/    210B :                         ;
3573/    210B :                         ; Inputs: Destination: PTR {RR2}
3574/    210B :                         ;         Source:      PTR {RR0}
3575/    210B :                         ;
3576/    210B :                         ; Outputs: none
3577/    210B :                         ;
3578/    210B :                         ;*********************************************************************
3579/    210B :
3580/    210B : D6 05 22                BlockMove       call Ext_Push
3581/    210E : 4C 02                           ld R4, #(BlockLength+1) /256
3582/    2110 : 5C 15                           ld R5, #(BlockLength+1) #256
3583/    2112 : 82 60                   Blk_Move        lde R6, @RR0
3584/    2114 : 92 62                           lde @RR2, R6
3585/    2116 : A0 E0                           incw RR0
3586/    2118 : A0 E2                           incw RR2
3587/    211A : 80 E4                           decw RR4
3588/    211C : EB F4                           jr NZ, Blk_Move
3589/    211E : D6 05 4F                        call Ext_Pop
3590/    2121 : AF                              ret
3591/    2122 :
3592/    2122 :
3593/    2122 :                         ;*********************************************************************
3594/    2122 :                         ;
3595/    2122 :                         ; Function: UpDate_Hdr  {update header}
3596/    2122 :                         ;
3597/    2122 :                         ;  This function is responsible for comparing the header information
3598/    2122 :                         ;  in the buffer (it is assumed that a ReadHdr operation has just
3599/    2122 :                         ;  been performed and that the header info is currently residing in
3600/    2122 :                         ;  the ReadHdr buffer space) to that of the global cylinder variable.
3601/    2122 :                         ;  If the two do not match, then Cur_Cyl is updated to reflect the
3602/    2122 :                         ;  ReadHdr operation's findings and the function returns a False
3603/    2122 :                         ;  value indicating that a seek is in order.
3604/    2122 :                         ;
3605/    2122 :                         ; Inputs: none
3606/    2122 :                         ;
3607/    2122 :                         ; Outputs: UpDate_Hdr: BOOLEAN {zero flag is true if off-track}
3608/    2122 :                         ;
3609/    2122 :                         ; Global Variables Used: Cylinder
3610/    2122 :                         ;
3611/    2122 :                         ; Global Variables Changed: On_Track, Cur_Cyl
3612/    2122 :                         ;
3613/    2122 :                         ; Algorithm:
3614/    2122 :                         ;
3615/    2122 :                         ;  Begin
3616/    2122 :                         ;    DiskStatus.On_Track:=false
3617/    2122 :                         ;    If GoodHdr
3618/    2122 :                         ;      Then
3619/    2122 :                         ;       If RHHeader.Cylinder<>Cylinder
3620/    2122 :                         ;        Then
3621/    2122 :                         ;          Cur_Cyl:=RHHeader.Cylinder
3622/    2122 :                         ;          On_Track:=false
3623/    2122 :                         ;          UpDate_Hdr:=false
3624/    2122 :                         ;        Else
3625/    2122 :                         ;          UpDate_Hdr:=true
3626/    2122 :                         ;          DiskStatus.On_Track:=true
3627/    2122 :                         ;  End
3628/    2122 :                         ;
3629/    2122 :                         ;*********************************************************************
3630/    2122 :
3631/    2122 : D6 05 22                UpDate_Hdr      call Ext_Push
3632/    2125 : 56 35 7F                        and DiskStat, #0FFh-On_Track
3633/    2128 : D6 20 B6                        call GoodHdr
3634/    212B : 2C 00                           ld R2, #0
3635/    212D : 6B 15                           jr Z, UpDate_Err
3636/    212F : 09 38                           ld Cur_Cyl, R0          ; Cur_Cyl:=RHHeader.Cylinder
3637/    2131 : 19 39                           ld Cur_Cyl+1, R1
3638/    2133 : B4 3A E0                        xor R0, Cylinder        ; check for correct cylinder address
3639/    2136 : B4 3B E1                        xor R1, Cylinder+1
3640/    2139 : 42 01                           or R0, R1
3641/    213B : 2C 00                           ld R2, #0               ; assume failure
3642/    213D : EB 05                           jr NZ, UpDate_Err
```

```
3643/    213F : 46 35 80                        or DiskStat, #On_Track
3644/    2142 : 2C 01                           ld R2, #1
3645/    2144 :                         ;
3646/    2144 : 42 22                   UpDate_Err      or R2, R2
3647/    2146 : 70 FC                           push FLAGS
3648/    2148 : D6 05 4F                        call Ext_Pop
3649/    214B : 50 FC                           pop FLAGS
3650/    214D : 8D 02 F6                        jp Bank_Ret
3651/    2150 :
3652/    2150 :
3653/    2150 : 3C 04                   L2150           ld R3, #4
3654/    2152 : D6 21 5B                        call L215b
3655/    2155 : 56 46 0F                        and ScrReg6, #00Fh
3656/    2158 : 8D 02 F6                        jp Bank_Ret
3657/    215B :
3658/    215B : 18 ED                   L215b           ld R1, R13
3659/    215D : 28 EE                           ld R2, R14
3660/    215F : 29 46                           ld ScrReg6, R2
3661/    2161 : CF                      L2161           rcf
3662/    2162 : C0 E1                           rrc R1
3663/    2164 : C0 E2                           rrc R2
3664/    2166 : 3A F9                           djnz R3, L2161
3665/    2168 : AF                              ret
3666/    2169 :
3667/    2169 :
3668/    2169 :                         ;*********************************************************************
3669/    2169 :                         ;
3670/    2169 :                         ; Procedure: Load_SprTbl
3671/    2169 :                         ;
3672/    2169 :                         ;  This procedure loads the spare table from disk. The table is
3673/    2169 :                         ;  found by a linear search of all spare blocks until a block
3674/    2169 :                         ;  is found where both the Spare Table Identifier field is present
3675/    2169 :                         ;  and the internal passwords and check byte are valid.
3676/    2169 :                         ;
3677/    2169 :                         ; Inputs: none
3678/    2169 :                         ;
3679/    2169 :                         ; Outputs: none
3680/    2169 :                         ;
3681/    2169 :                         ; Algorithm:
3682/    2169 :                         ;
3683/    2169 :                         ;  Begin
3684/    2169 :                         ;    Seek_Type:=Access_Offset
3685/    2169 :                         ;    InterLeaveFactor:=Find_InterLeave
3686/    2169 :                         ;    Found:=false
3687/    2169 :                         ;    Count:=0
3688/    2169 :                         ;    i:=NumberOfSpareBlocks
3689/    2169 :                         ;    While i>0 and Count<2 Do
3690/    2169 :                         ;      Seek(CnvrtLogical(MulR0_m(i)))
3691/    2169 :                         ;      j:=0
3692/    2169 :                         ;      Sector:=0
3693/    2169 :                         ;      While not(Found) And (j<NbrSctrs) Do
3694/    2169 :                         ;        If ReadCommon
3695/    2169 :                         ;          Then
3696/    2169 :                         ;            If BlockID=SprTblID and
3697/    2169 :                         ;                 PassWord1=PassWord2=PassWord and
3698/    2169 :                         ;                 CheckByte is valid
3699/    2169 :                         ;             Then
3700/    2169 :                         ;               Found:=true
3701/    2169 :                         ;               Count:=Count+1
3702/    2169 :                         ;               If Count>1
3703/    2169 :                         ;                 Then
3704/    2169 :                         ;                   If SpareTable.RunNumber<ReadBuffer.RunNumber
3705/    2169 :                         ;                     Then MoveBlock(SpareTable, ReadBuffer)
3706/    2169 :                         ;                     Else MoveBlock(SpareTable, ReadBuffer)
3707/    2169 :                         ;          Else
3708/    2169 :                         ;            If RdErrCnt<10
3709/    2169 :                         ;              Then
3710/    2169 :                         ;                MoveBlock(ReadBuffer, Buffer2)
3711/    2169 :                         ;                If BlockID=SprTblID and
3712/    2169 :                         ;                     PassWord1=PassWord2=PassWord and
3713/    2169 :                         ;                     CheckByte is valid
3714/    2169 :                         ;                  Then
3715/    2169 :                         ;                    Found:=true
3716/    2169 :                         ;                    Count:=Count+1
3717/    2169 :                         ;                    If Count>1
3718/    2169 :                         ;                      Then
3719/    2169 :                         ;                        If SpareTable.RunNumber<ReadBuffer.RunNumber
3720/    2169 :                         ;                          Then MoveBlock(SpareTable, ReadBuffer)
3721/    2169 :                         ;                        Else MoveBlock(SpareTable, ReadBuffer)
3722/    2169 :                         ;        j:=j+1
3723/    2169 :                         ;        Sector:=(Secotr+InterLeaveFactor) mod NbrSctrs
3724/    2169 :                         ;      i:=i+1
3725/    2169 :                         ;    If not(Found)
3726/    2169 :                         ;      Then Abort
3727/    2169 :                         ;      Else
3728/    2169 :                         ;        UpDate_SprTbl
3729/    2169 :                         ;        Park
3730/    2169 :                         ;        SlfTst_Result.NoSpareTable:=False
3731/    2169 :                         ;  End
3732/    2169 :                         ;
3733/    2169 :                         ;*********************************************************************
3734/    2169 :
3735/    2169 : D6 22 0C                Load_SprTbl     call L220c
3736/    216C : 6C 4C                           ld R6, #76              ; i:= NumberOfSpareBlocks
3737/    216E : B0 E7                           clr R7                  ; SprTbl_Found:=false
3738/    2170 : 46 35 04                        or DiskStat, #User_Type
3739/    2173 : D6 05 22                L_SprTbl_Lp     call Ext_Push
3740/    2176 : 08 E6                           ld R0, R6
3741/    2178 : 2C 1B                           ld R2, #MulR0_m /256
3742/    217A : 3C 80                           ld R3, #MulR0_m #256    ; do Get_Cyl_H_S after that
3743/    217C : D6 02 CB                        call Bank_Call
3744/    217F : FC 00                           ld R15, #0              ; set sector 0
3745/    2181 : 56 35 F7                        and DiskStat, #0FFh-Offset_On
3746/    2184 : D6 25 47                        call Seek
3747/    2187 : D6 05 4F                        call Ext_Pop
3748/    218A : 2C 10                           ld R2, #RdBlk_Vector /256
3749/    218C : 3C 94                           ld R3, #RdBlk_Vector #256
3750/    218E : D6 02 CB                        call Bank_Call
3751/    2191 : EB 18                           jr NZ, Chk_SprTbl
3752/    2193 :                         ;
3753/    2193 : 56 E0 0F                        and R0, #00Fh           ; mask unwanted status
3754/    2196 : A6 E0 0A                        cp R0, #10              ; check for any successful reads
3755/    2199 : 9B 05                           jr GE, L_SprTbl_More
3756/    219B : D6 20 4C                        call Buf2_To_RBuf
3757/    219E : 8B 0B                           jr Chk_SprTbl
3758/    21A0 :
3759/    21A0 : 6A D1                   L_SprTbl_More   djnz R6, L_SprTbl_Lp
3760/    21A2 : 42 77                           or R7, R7               ; check if any spare table found
```

```
3761/    21A4 : EB 57                           jr NZ, L_Spr_End
3762/    21A6 : FC 1E                           ld R15, #30             ; no spare table found
3763/    21A8 : 8D 03 57                        jp Abort
3764/    21AB :                         ;
3765/    21AB : 0C 10                   Chk_SprTbl      ld R0, #RBuffer1 /256
3766/    21AD : 1C 19                           ld R1, #RBuffer1 #256
3767/    21AF : D6 23 1D                        call Chk_PassWord
3768/    21B2 : 6B EC                           jr Z, L_SprTbl_More
3769/    21B4 :                         ;
3770/    21B4 : 2C 11                           ld R2, #(RBuffer1+SpareCheck-SpareArray) /256
3771/    21B6 : 3C BF                           ld R3, #(RBuffer1+SpareCheck-SpareArray) #256
3772/    21B8 : 82 02                           lde R0, @RR2            ; check possible check byte
3773/    21BA : A0 E2                           incw RR2
3774/    21BC : 82 12                           lde R1, @RR2
3775/    21BE : 31 40                           srp #Wrk_Scr
3776/    21C0 :                                 assume RP:Wrk_Scr
3777/    21C0 : CC 10                           ld R12, #RBuffer1 /256
3778/    21C2 : DC 19                           ld R13, #RBuffer1 #256
3779/    21C4 : D6 22 28                        call SprChk2
3780/    21C7 : 31 10                           srp #Wrk_Sys
3781/    21C9 :                                 assume RP:Wrk_Sys
3782/    21C9 : D6 22 57                        call Chk_Spr2
3783/    21CC : 6B D2                           jr Z, L_SprTbl_More
3784/    21CE :                         ;
3785/    21CE : 42 77                           or R7, R7               ; check for a SpareTable already found
3786/    21D0 : 6B 22                           jr Z, L_Spr_Move
3787/    21D2 : 31 40                           srp #Wrk_Scr
3788/    21D4 :                                 assume RP:Wrk_Scr
3789/    21D4 : 6C 14                           ld R6, #SpareTmStmp /256
3790/    21D6 : 7C 99                           ld R7, #SpareTmStmp #256
3791/    21D8 : 4C 40                           ld R4, #ScrReg0
3792/    21DA : D6 22 1D                        call Ld_TmStmp
3793/    21DD : 6C 10                           ld R6, #(RBuffer1+SpareTmStmp-SpareArray) /256
3794/    21DF : 7C 1D                           ld R7, #(RBuffer1+SpareTmStmp-SpareArray) #256
3795/    21E1 : 4C 4C                           ld R4, #ScrRegC
3796/    21E3 : D6 22 1D                        call Ld_TmStmp
3797/    21E6 : 22 3F                           sub R3, R15
3798/    21E8 : 32 2E                           sbc R2, R14
3799/    21EA : 32 1D                           sbc R1, R13
3800/    21EC : 32 0C                           sbc R0, R12
3801/    21EE : 31 10                           srp #Wrk_Sys
3802/    21F0 :                                 assume RP:Wrk_Sys
3803/    21F0 : 9B 05                           jr GE, L_Spr_Inc
3804/    21F2 : 00 E7                           dec R7                  ; account for old, bogus spare table
3805/    21F4 : D6 20 7A                L_Spr_Move      call RBuf_To_Spr
3806/    21F7 : 7E                      L_Spr_Inc       inc R7                  ; note the arrval of a SpareTable
3807/    21F8 : A6 E7 02                        cp R7, #2               ; see if that's all there is
3808/    21FB : EB A3                           jr NZ, L_SprTbl_More
3809/    21FD :                         ;
3810/    21FD : D6 22 80                L_Spr_End       call UpDate_SprTbl
3811/    2200 : 56 33 FE                        and SlfTst_Result, #0FFh-No_SprTbl
3812/    2203 : D6 22 0C                        call L220c
3813/    2206 : D6 28 5D                        call Park_Heads
3814/    2209 : 8D 02 F6                        jp Bank_Ret
3815/    220C :                         ;
3816/    220C : 2C 16                   L220c           ld R2, #016h            ; clear 24 bytes from RAM 1641h
3817/    220E : 3C 41                           ld R3, #041h
3818/    2210 : 1C 18                           ld R1, #018h
3819/    2212 : 0C 00                           ld R0, #0
3820/    2214 : 92 02                   L2214           lde @RR2, R0
3821/    2216 : A0 E2                           incw RR2
3822/    2218 : 1A FA                           djnz R1, L2214
3823/    221A : 8D 02 F6                        jp Bank_Ret
3824/    221D :                         ;*********************************************************************
3825/    221D : 5C 04                   Ld_TmStmp       ld R5, #4               ; load 4 bytes
3826/    221F : 83 46                   Ld_Tm_Lp        ldei @R4, @RR6
3827/    2221 : 5A FC                           djnz R5, Ld_Tm_Lp
3828/    2223 : AF                              ret
3829/    2224 :
3830/    2224 :
3831/    2224 :                         ;*********************************************************************
3832/    2224 :                         ;
3833/    2224 :                         ; Procedure: SprChkSum
3834/    2224 :                         ;
3835/    2224 :                         ;  This procedure calculates a 16-bit checksum over the contents
3836/    2224 :                         ;  of the spare table, and stores the sum within the spare table.
3837/    2224 :                         ;
3838/    2224 :                         ; Inputs: none
3839/    2224 :                         ;
3840/    2224 :                         ; Outputs: none
3841/    2224 :                         ;
3842/    2224 :                         ; Side Effect: ScrReg1, ScrReg2 hold the calculated check byte on return
3843/    2224 :                         ;
3844/    2224 :                         ; Algorithm:
3845/    2224 :                         ;
3846/    2224 :                         ;  Begin
3847/    2224 :                         ;    Sum:=0
3848/    2224 :                         ;    SumPtr:=SpareArray
3849/    2224 :                         ;    For i:=1 To Length(SpareArray) Do
3850/    2224 :                         ;      Sum:=Sum+SpareArray[i-1]
3851/    2224 :                         ;    SpareArray.ChkSum:=Sum
3852/    2224 :                         ;  End
3853/    2224 :                         ;
3854/    2224 :                         ;*********************************************************************
3855/    2224 :
3856/    2224 : CC 14                   SprChkSum       ld R12, #SpareArray /256
3857/    2226 : DC 95                           ld R13, #SpareArray #256
3858/    2228 : EC 01                   SprChk2         ld R14, #(SpareCheck-SpareArray) /256
3859/    222A : FC A6                           ld R15, #(SpareCheck-SpareArray) #256
3860/    222C : B0 E4                           clr R4
3861/    222E : B0 E5                           clr R5
3862/    2230 : 82 0C                   SprChk_Lp       lde R0, @RR12
3863/    2232 : 02 50                           add R5, R0
3864/    2234 : 16 E4 00                        adc R4, #0
3865/    2237 : A0 EC                           incw RR12
3866/    2239 : 80 EE                           decw RR14
3867/    223B : EB F3                           jr NZ, SprChk_Lp
3868/    223D : 92 4C                           lde @RR12, R4           ; store high byte of checksum
3869/    223F : A0 EC                           incw RR12
3870/    2241 : 92 5C                           lde @RR12, R5           ; store low byte of checksum
3871/    2243 : 8D 02 F6                        jp Bank_Ret
3872/    2246 :
3873/    2246 :
3874/    2246 :                         ;*********************************************************************
3875/    2246 :                         ;
3876/    2246 :                         ; Function: Chk_SprChk
3877/    2246 :                         ;
3878/    2246 :                         ;  This function is responsible for verifying that the checksum
```

```
3879/    2246 :                     ;  residing in the spare table is correct.
3880/    2246 :                     ;
3881/    2246 :                     ; Inputs: none
3882/    2246 :                     ;
3883/    2246 :                     ; Outputs: Chk_SprChk: BOOLEAN {zero flag}
3884/    2246 :                     ;
3885/    2246 :                     ; Algorithm:
3886/    2246 :                     ;
3887/    2246 :                     ;  Begin
3888/    2246 :                     ;    TempSum:=SpareTable.CheckSum
3889/    2246 :                     ;    SpareTable.CheckSum:=SprChkSum
3890/    2246 :                     ;    If TempSum=SpareTable.CheckSum
3891/    2246 :                     ;      Then Chk_SprChk:=true
3892/    2246 :                     ;      Else Chk_SprChk:=false
3893/    2246 :                     ;  End
3894/    2246 :                     ;
3895/    2246 :                     ;*****************************************************************
3896/    2246 :
3897/    2246 : 2C 16               Chk_SprChk      ld R2, #SpareCheck /256
3898/    2248 : 3C 3B                               ld R3, #SpareCheck #256
3899/    224A : 82 02                               lde R0, @RR2
3900/    224C : A0 E2                               incw RR2
3901/    224E : 82 12                               lde R1, @RR2
3902/    2250 : 31 40                               srp #Wrk_Scr
3903/    2252 :                            assume RP:Wrk_Scr
3904/    2252 : D6 22 24                            call SprChkSum
3905/    2255 : 31 10                               srp #Wrk_Sys
3906/    2257 :                            assume RP:Wrk_Sys
3907/    2257 : B4 44 E0            Chk_Spr2        xor R0, ScrReg4         ; side effect: ScrReg 4:5 hold new checkbyte
3908/    225A : B4 45 E1                            xor R1, ScrReg5
3909/    225D : 42 01                               or R0, R1
3910/    225F : B0 E0                               clr R0
3911/    2261 : EB 02                               jr NZ, Chk_Spr_End
3912/    2263 : 0C 01                               ld R0, #1
3913/    2265 : 42 00               Chk_Spr_End     or R0, R0               ; set zero flag
3914/    2267 : 8D 02 F6                            jp Bank_Ret
3915/    226A :
3916/    226A :
3917/    226A :                     ;*****************************************************************
3918/    226A :                     ;
3919/    226A :                     ; Function: Spr
3920/    226A :                     ;
3921/    226A :                     ;  This function returns the logical block number associated
3922/    226A :                     ;  with the parameter passed in. Because there are only two spare
3923/    226A :                     ;  blocks containing the spare table, this function accepts
3924/    226A :                     ;  only ODD or EVEN input params.
3925/    226A :                     ;
3926/    226A :                     ; Inputs: SpareTableIndex: BYTE {R8}
3927/    226A :                     ;
3928/    226A :                     ; Outputs: Spr: 3 BYTES {R12:14}
3929/    226A :                     ;
3930/    226A :                     ; Algorithm:
3931/    226A :                     ;
3932/    226A :                     ;  Begin
3933/    226A :                     ;    If SpareTableIndex is EVEN
3934/    226A :                     ;      Then Spr:=SprBlk0
3935/    226A :                     ;      Else Spr:=SprBlk1
3936/    226A :                     ;  End
3937/    226A :                     ;
3938/    226A :                     ;*****************************************************************
3939/    226A :
3940/    226A : 76 E8 01            Spr             tm R8, #1               ; If SpareTableIndex is EVEN
3941/    226D : EB 08                               jr NZ, Spr_Odd
3942/    226F :                     ;
3943/    226F : CC 00                               ld R12, #HiSpr0
3944/    2271 : DC 32                               ld R13, #MidSpr0
3945/    2273 : EC AB                               ld R14, #LoSpr0
3946/    2275 : 8B 06                               jr Spr_End
3947/    2277 :                     ;
3948/    2277 : CC 00               Spr_Odd         ld R12, #HiSpr1
3949/    2279 : DC 65                               ld R13, #MidSpr1
3950/    227B : EC 53                               ld R14, #LoSpr1
3951/    227D : 8D 02 F6            Spr_End         jp Bank_Ret
3952/    2280 :
3953/    2280 :
3954/    2280 :                     ;*****************************************************************
3955/    2280 :                     ;
3956/    2280 :                     ; Procedure: UpDate_SprTbl
3957/    2280 :                     ;
3958/    2280 :                     ;  This procedure is responsible for updating the spare table
3959/    2280 :                     ;  to both of its locations on disk after a change has been made
3960/    2280 :                     ;  to the table
3961/    2280 :                     ;
3962/    2280 :                     ; Inputs: none
3963/    2280 :                     ;
3964/    2280 :                     ; Outputs: none
3965/    2280 :                     ;
3966/    2280 :                     ; Algorithm:
3967/    2280 :                     ;
3968/    2280 :                     ;  Begin
3969/    2280 :                     ;    SpareTmStamp:=SpareTmStamp+1
3970/    2280 :                     ;    SprChkSum
3971/    2280 :                     ;    ZeroBlock
3972/    2280 :                     ;    WBuffer1.BlockID:=PassWord
3973/    2280 :                     ;    For i:=0 To 1 Do
3974/    2280 :                     ;      MoveBlock(WriteBuffer, SpareTable)
3975/    2280 :                     ;      TempCyl, TempHead, TempSector:=
3976/    2280 :                     ;                    Get_Cyl_H_S(CnvrtLogical(SpareTable(Spr(i)))
3977/    2280 :                     ;      If table not found in SpareTable Then Abort
3978/    2280 :                     ;      Seek(TemoCyl, TempHead, TempSector)
3979/    2280 :                     ;      If nor(WriteVerify(Conservative))
3980/    2280 :                     ;        Then
3981/    2280 :                     ;          If Recovery And WriteVerify.ErrorCode=Ex_ReadErr
3982/    2280 :                     ;            Then Exit UpDate_SprTbl
3983/    2280 :                     ;            Else
3984/    2280 :                     ;              ZeroBlock
3985/    2280 :                     ;              WriteBlock
3986/    2280 :                     ;              MoveBlock(Buffer2, SpareTable)
3987/    2280 :                     ;              SpareBlock(True, SpareTable, Write_Op, Spr(i))
3988/    2280 :                     ;  End
3989/    2280 :                     ;
3990/    2280 :                     ;*****************************************************************
3991/    2280 :
3992/    2280 : C8 3A               UpDate_SprTbl   ld R12, Cylinder        ; save current seek address
3993/    2282 : D8 3B                               ld R13, Cylinder+1
3994/    2284 : E8 3C                               ld R14, Head
3995/    2286 : F8 3D                               ld R15, Sector
3996/    2288 : D6 05 22                            call Ext_Push
```

```
3997/    228B :                     ;
3998/    228B : 2C 14                           ld R2, #FmtOffset /256  ; one byte after timestamp
3999/    228D : 3C 9D                           ld R3, #FmtOffset #256
4000/    228F : FC 04                           ld R15, #4              ; get 4 bytes
4001/    2291 : 1C 01                           ld R1, #1
4002/    2293 : 80 E2               UpDate_1    decw RR2
4003/    2295 : 82 02                           lde R0, @RR2
4004/    2297 : 02 01                           add R0, R1              ; increment the count
4005/    2299 : 92 02                           lde @RR2, R0
4006/    229B : 7B 02                           jr C, UpDate_2
4007/    229D : 1C 00                           ld R1, #0
4008/    229F : FA F2               UpDate_2    djnz R15, UpDate_1
4009/    22A1 : D6 22 24                        call SprChkSum
4010/    22A4 : 4C 02                           ld R4, #2               ; write the table to the disk twice
4011/    22A6 : B0 E5                           clr R5                  ; spare table index
4012/    22A8 : D6 20 9D            SprB_Lp     call Spr_To_WrBuf
4013/    22AB : EC 12                           ld R14, #(WBuffer1+BlockID) /256
4014/    22AD : FC 21                           ld R15, #(WBuffer1+BlockID) #256
4015/    22AF : D6 05 E7                        call Load_PassWord
4016/    22B2 : 88 E5                           ld R8, R5
4017/    22B4 : D6 22 6A                        call Spr
4018/    22B7 : 56 35 FB                        and DiskStat, #0FFh-User_Type
4019/    22BA : 2C 1B                           ld R2, #CnvrtLogical /256
4020/    22BC : 3C 04                           ld R3, #CnvrtLogical #256
4021/    22BE : D6 02 CB                        call Bank_Call
4022/    22C1 : EB 05                           jr NZ, SprB_Seek
4023/    22C3 : FC 1F                           ld R15, #31
4024/    22C5 : 8D 03 57                        jp Abort
4025/    22C8 :
4026/    22C8 : 46 35 08            SprB_Seek   or DiskStat, #Offset_On
4027/    22CB : D6 25 47                        call Seek
4028/    22CE : 46 3E 40                        or 3Eh, #S_Block        ; say that we've got a spare block
4029/    22D1 : 56 35 FB                        and DiskStat, #0FFh-User_Type
4030/    22D4 : 2C 11                           ld R2, #WrVer_Common /256
4031/    22D6 : 3C A4                           ld R3, #WrVer_Common #256
4032/    22D8 : D6 02 CB                        call Bank_Call
4033/    22DB : EB 34                           jr NZ, Spr_Next
4034/    22DD :                     ;
4035/    22DD : 76 32 80                        tm Excpt_Stat, #Recovery        ; If Recovery Then ...
4036/    22E0 : EB 05                           jr NZ, Rcvr_SprTbl
4037/    22E2 : FC 20                           ld R15, #32
4038/    22E4 : 8D 03 57                        jp Abort
4039/    22E7 :
4040/    22E7 : 76 35 20            Rcvr_SprTbl tm DiskStat, #Wr_Op
4041/    22EA : EB 08                           jr NZ, Rcvr_SprT1
4042/    22EC : 56 E9 0F                        and R9, #00Fh           ; check for noisy read
4043/    22EF : A6 E9 03                        cp R9, #SprThresh
4044/    22F2 : 2B 1D                           jr LE, Spr_Next
4045/    22F4 : 0C A5               Rcvr_SprT1  ld R0, #0A5h
4046/    22F6 : D6 05 A3                        call L05a3
4047/    22F9 : 46 35 20                        or DiskStat, #Wr_Op
4048/    22FC : 2C 11                           ld R2, #RW_Common /256
4049/    22FE : 3C D3                           ld R3, #RW_Common #256
4050/    2300 : D6 02 CB                        call Bank_Call
4051/    2303 : 88 E5                           ld R8, R5
4052/    2305 : D6 22 6A                        call Spr
4053/    2308 : 2C 13                           ld R2, #Spr_Enter /256
4054/    230A : 3C 4B                           ld R3, #Spr_Enter #256
4055/    230C : D6 02 CB                        call Bank_Call
4056/    230F : 6B D6                           jr Z, Rcvr_SprTbl
4057/    2311 :                     ;
4058/    2311 : 5E                  Spr_Next    inc R5                  ; do next table
4059/    2312 : 4A 94                           djnz R4, SprB_Lp
4060/    2314 : D6 05 4F                        call Ext_Pop
4061/    2317 : D6 25 47                        call Seek               ; get back to original seek address
4062/    231A : 8D 02 F6                        jp Bank_Ret
4063/    231D :
4064/    231D :
4065/    231D :                     ;*********************************************************************
4066/    231D :                     ;
4067/    231D :                     ; Function: Chk_PassWord
4068/    231D :                     ;
4069/    231D :                     ;  This function is responsible for checking if the 32 bits
4070/    231D :                     ;  pointed to by the input parameter match with the controller's
4071/    231D :                     ;  32 bit password
4072/    231D :                     ;
4073/    231D :                     ; Inputs: PassWordPtr: PTR {SysReg0:1}
4074/    231D :                     ;
4075/    231D :                     ; Outputs: Chk_PassWord: BOOLEAN {zero flag}
4076/    231D :                     ;
4077/    231D :                     ; Algorithm:
4078/    231D :                     ;
4079/    231D :                     ;  Begin
4080/    231D :                     ;    If @PassWordPtr=PassWord
4081/    231D :                     ;      Then Chk_PassWord:=true
4082/    231D :                     ;      Else Chk_PassWord:=false
4083/    231D :                     ;  End
4084/    231D :                     ;
4085/    231D :                     ;*********************************************************************
4086/    231D :
4087/    231D : 31 40               Chk_PassWord    srp #Wrk_Scr
4088/    231F :                             assume RP:Wrk_Scr
4089/    231F : 4C 04                           ld R4, #4               ; check 4 bytes
4090/    2321 : 28 10                           ld R2,10h
4091/    2323 : 38 11                           ld R3,11h
4092/    2325 : EC 10                           ld R14, #Password /256
4093/    2327 : FC 03                           ld R15, #Password #256
4094/    2329 : C2 0E               Chk_P_Lp    ldc R0, @RR14           ; get a byte of the password
4095/    232B : A0 EE                           incw RR14
4096/    232D : 82 12                           lde R1, @RR2            ; get a byte of test string
4097/    232F : A0 E2                           incw RR2
4098/    2331 : A2 01                           cp R0, R1
4099/    2333 : B0 E0                           clr R0                  ; assume failure
4100/    2335 : EB 04                           jr NZ, Chk_P_End
4101/    2337 : 4A F0                           djnz R4, Chk_P_Lp
4102/    2339 : 0C 01                           ld R0, #1
4103/    233B : 42 00               Chk_P_End   or R0, R0               ; set flags
4104/    233D : 31 10                           srp #Wrk_Sys
4105/    233F :                             assume RP:Wrk_Sys
4106/    233F : 8D 02 F6                        jp Bank_Ret
4107/    2342 :
4108/    2342 :
4109/    2342 :                     ;*********************************************************************
4110/    2342 :                     ;
4111/    2342 :                     ; Procedure: SpareCount
4112/    2342 :                     ;
4113/    2342 :                     ;  This procedure is responsible for incrementing/decrementing
4114/    2342 :                     ;  the Spare/Bad Block count. If the total count exceeds
```

```
4115/    2342 :                     ;  MaxSpares-5 then a status bit is set for this command only.
4116/    2342 :                     ;
4117/    2342 :                     ; Inputs: Command: 2 BITS {R0/Bits 1:0}
4118/    2342 :                     ;
4119/    2342 :                     ; Outputs: none
4120/    2342 :                     ;
4121/    2342 :                     ; Algorithm:
4122/    2342 :                     ;
4123/    2342 :                     ;  Begin
4124/    2342 :                     ;    Case Command Of
4125/    2342 :                     ;      0: Increment the spare count
4126/    2342 :                     ;      1: Increment the bad block count
4127/    2342 :                     ;      2: Decrement the bad block count
4128/    2342 :                     ;      Otherwise Abort
4129/    2342 :                     ;    If SpareCount+BadBlockCount>=MaxSpares-5
4130/    2342 :                     ;      Then
4131/    2342 :                     ;        Excpt_Stat.SprTbl_Warn:=true
4132/    2342 :                     ;        SetStatus(SprBlk_Warn)
4133/    2342 :                     ;  End
4134/    2342 :                     ;
4135/    2342 :                     ;*********************************************************************
4136/    2342 :
4137/    2342 : 18 E0               SpareCount      ld R1, R0               ; get command
4138/    2344 : 56 E1 FC                            and R1, #0FCh           ; check for illegal command
4139/    2347 : 6B 07                               jr Z, SprCnt_1
4140/    2349 : 98 E0                               ld R9, R0
4141/    234B : FC 21                               ld R15, #33
4142/    234D : 8D 03 57                            jp Abort
4143/    2350 :                     ;
4144/    2350 : 2C 14               SprCnt_1        ld R2, #SprCount /256
4145/    2352 : 3C DF                               ld R3, #SprCount #256
4146/    2354 : 82 12                               lde R1, @RR2            ; assume spare count increment
4147/    2356 : 1E                                  inc R1
4148/    2357 : A6 E0 00                            cp R0, #0               ; check for Inc_SpareCount
4149/    235A : 6B 12                               jr Z, S_C_Spare
4150/    235C : A0 E2                               incw RR2                ; get address of BadCount
4151/    235E : 82 12                               lde R1, @RR2            ; get bad block count
4152/    2360 : A6 E0 01                            cp R0, #1               ; check for Inc_BadBlock
4153/    2363 : 6B 04                               jr Z, S_C_BadInc
4154/    2365 : 00 E1                               dec R1                  ; otherwise Decrement bad block count
4155/    2367 : 8B 01                               jr S_C_BadEnd
4156/    2369 :                     ;
4157/    2369 : 1E                  S_C_BadInc      inc R1
4158/    236A : 92 12               S_C_BadEnd      lde @RR2, R1            ; store new count
4159/    236C : 8B 02                               jr SprCnt_End
4160/    236E :                     ;
4161/    236E : 92 12               S_C_Spare       lde @RR2, R1            ; store new count
4162/    2370 : D6 23 76            SprCnt_End      call Chk_SprCnt
4163/    2373 : 8D 02 F6                            jp Bank_Ret
4164/    2376 :                     ;
4165/    2376 : 2C 14               Chk_SprCnt      ld R2, #SprCount /256
4166/    2378 : 3C DF                               ld R3, #SprCount #256
4167/    237A : 82 02                               lde R0, @RR2            ; get spare count
4168/    237C : A0 E2                               incw RR2
4169/    237E : 82 12                               lde R1, @RR2            ; get bd block count
4170/    2380 : 02 01                               add R0, R1
4171/    2382 : A6 E0 47                            cp R0, #76-5            ; check for spare count overflow
4172/    2385 : 1B 06                               jr LT, Chk_Spr_Ret
4173/    2387 : 46 32 40                            or Excpt_Stat, #SprTbl_Warn
4174/    238A : D6 03 BC                            call SS_SprWarn
4175/    238D : 8D 02 F6            Chk_Spr_Ret     jp Bank_Ret
4176/    2390 :
4177/    2390 :
4178/    2390 :
4179/    2390 :                     ;*********************************************************************
4180/    2390 :                     ; Module Spare1.Assem (a continuation of Spare)
4181/    2390 :                     ;
4182/    2390 :                     ;       FUNCTION SrchSpTabl(LogicalBlock : 3 BYTES {R12:14}
4183/    2390 :                     ;               ElementType: BYTE {R15}) :
4184/    2390 :                     ;               PhysicalBlock : 3 BYTES {R12:14}
4185/    2390 :                     ;               Status : BYTE {R0}
4186/    2390 :                     ;               ElementPtr : BYTE {R1}
4187/    2390 :                     ;       FUNCTION GetNewSpare(BlockNumber : 3 BYTES {R12:14}) : BYTE {R0}
4188/    2390 :                     ;       PROCEDURE AddSpare(BlockType : 3 BITS {R8/bits 3:1}
4189/    2390 :                     ;               SpareType : BIT {R8/bit 4}
4190/    2390 :                     ;               Location  : BYTE {R15}
4191/    2390 :                     ;               LogicalBlock : 3 BYTES {R12:14})
4192/    2390 :                     ;       PROCEDURE DeleteSpare(Location : BYTE {R15}
4193/    2390 :                     ;               LogicalBlock : 3 BYTES {R12:14})
4194/    2390 :                     ;
4195/    2390 :                     ;*********************************************************************
4196/    2390 :
4197/    2390 :                     ;*********************************************************************
4198/    2390 :                     ;
4199/    2390 :                     ; Function: GetNewSpare
4200/    2390 :                     ;
4201/    2390 :                     ;  This function accepts either a physical block number or
4202/    2390 :                     ;  a logical block number and returns a one byte index into
4203/    2390 :                     ;  the Spare Table's bit map describing the location of the
4204/    2390 :                     ;  block that can be used as a spare
4205/    2390 :                     ;
4206/    2390 :                     ; Inputs: BlockNumber: 3 BYTES {R12:14}
4207/    2390 :                     ;
4208/    2390 :                     ; Outputs: GetNewSpare BYTE {R0}
4209/    2390 :                     ;
4210/    2390 :                     ; Global Variables Used: SpareBitMap
4211/    2390 :                     ;
4212/    2390 :                     ; Local Variables: Bit:   ScrReg0
4213/    2390 :                     ;                  Temp1: ScrReg1
4214/    2390 :                     ;                  Temp2: ScrReg2
4215/    2390 :                     ;                  NoHis: ScrReg3/bit 7
4216/    2390 :                     ;                  NoLos: ScrReg3/bit 6
4217/    2390 :                     ;
4218/    2390 :                     ; Algorithm:
4219/    2390 :                     ;
4220/    2390 :                     ;  Begin
4221/    2390 :                     ;    Bit:=CnvrtLogical div k {get physical blocknumber divided by
4222/    2390 :                     ;                             the number of blocks between spares }
4223/    2390 :                     ;    If SpareBitMap[Bit]=0
4224/    2390 :                     ;      Then GetNewSpare:=Bit
4225/    2390 :                     ;      Else
4226/    2390 :                     ;        NoHis:=false
4227/    2390 :                     ;        NoLos:=false
4228/    2390 :                     ;        Temp1:=Bit
4229/    2390 :                     ;        While not(NoHis) and SpareBitMap[Temp1]=1 Do
4230/    2390 :                     ;          Temp1:=Temp1+1
4231/    2390 :                     ;          If Temp1>=76 Then NoHis:=True
4232/    2390 :                     ;        Temp2:=Bit
```

```
4233/    2390 :                     ;        While not(NoLos) and SpareBitMap[Temp2]=1 Do
4234/    2390 :                     ;          Temp2:=Temp2-1
4235/    2390 :                     ;          If Temp2<0 Then NoLos:=true
4236/    2390 :                     ;        If NoHis and NoLos
4237/    2390 :                     ;          Then Abort {spare table full}
4238/    2390 :                     ;          Else
4239/    2390 :                     ;            If NoHis
4240/    2390 :                     ;              Then GetNewSpare:=Temp2
4241/    2390 :                     ;              Else
4242/    2390 :                     ;                If Temp1-Bit > Bit-Temp2
4243/    2390 :                     ;                  Then GetNewSpare:=Temp2
4244/    2390 :                     ;                  Else GetNewSpare:=Temp1
4245/    2390 :                     ;  End
4246/    2390 :                     ;
4247/    2390 :                     ;*****************************************************************
4248/    2390 :
4249/    2390 : D6 05 22            GetNewSpare     call Ext_Push           ; save state
4250/    2393 :                     ; do a range check first. Rene's capacity is 38064 (9834h) blocks
4251/    2393 :                     ;  plus 76 spares, which gives a total of 610*2*32 = 39040 (9880h).
4252/    2393 : 48 ED                               ld R4, R13              ; LBA mid byte
4253/    2395 : A6 E4 98                            cp R4, #098h            ; do range check
4254/    2398 : EB 02                               jr NZ, Gns_RC1
4255/    239A : 4C 97                               ld R4, #097h
4256/    239C : CF                  Gns_RC1         rcf
4257/    239D : C0 E4                               rrc R4
4258/    239F : A6 E4 4C                            cp R4, #04Ch
4259/    23A2 : 1B 05                               jr LT, Gns_RCok
4260/    23A4 : FC 35                               ld R15, #53
4261/    23A6 : 8D 03 57                            jp Abort
4262/    23A9 :                     ;
4263/    23A9 : D8 E4               Gns_RCok        ld R13, R4
4264/    23AB : CC 80                               ld R12, #TestBitMap
4265/    23AD : 2C 14                               ld R2, #TSC_BitMap /256
4266/    23AF : 3C E3                               ld R3, #TSC_BitMap #256
4267/    23B1 : D6 02 CB                            call Bank_Call          ; If BitMap[Bit]=0 ...
4268/    23B4 : 08 E4                               ld R0, R4               ; assume Bit is unused
4269/    23B6 : 6B 5D                               jr Z, Gns_End           ; jump if Then
4270/    23B8 :                     ;
4271/    23B8 : 58 E4                               ld R5, R4               ; Else ...
4272/    23BA : B0 E7                               clr R7
4273/    23BC : D8 E5               Gns_Lp1         ld R13, R5              ; test for bit map location = 0
4274/    23BE : CC 80                               ld R12, #TestBitMap
4275/    23C0 : 2C 14                               ld R2, #TSC_BitMap /256
4276/    23C2 : 3C E3                               ld R3, #TSC_BitMap #256
4277/    23C4 : D6 02 CB                            call Bank_Call
4278/    23C7 : 6B 09                               jr Z, Gns_Lp1End
4279/    23C9 : 5E                                  inc R5                  ; bump Temp1
4280/    23CA : A6 E5 4C                            cp R5, #76
4281/    23CD : 1B ED                               jr LT, Gns_Lp1
4282/    23CF : 46 E7 80                            or R7, #080h            ; NoHis:=true
4283/    23D2 : 68 E4               Gns_Lp1End      ld R6, R4
4284/    23D4 : D8 E6               Gns_Lp2         ld R13, R6              ; test for bit map location = 0
4285/    23D6 : CC 80                               ld R12, #TestBitMap
4286/    23D8 : 2C 14                               ld R2, #TSC_BitMap /256
4287/    23DA : 3C E3                               ld R3, #TSC_BitMap #256
4288/    23DC : D6 02 CB                            call Bank_Call
4289/    23DF : 6B 07                               jr Z, Gns_Lp2End
4290/    23E1 : 00 E6                               dec R6
4291/    23E3 : AB EF                               jr GT, Gns_Lp2
4292/    23E5 : 46 E7 40                            or R7, #040h            ; NoLos:=true
4293/    23E8 : 66 E7 C0            Gns_Lp2End      tcm R7, #0C0h           ; If NoHis and NoLos...
4294/    23EB : EB 0C                               jr NZ, Gns_Chk_Hi
4295/    23ED :                     ;
4296/    23ED : 0C 01                               ld R0, #1               ; status byte 1
4297/    23EF : 1C 40                               ld R1, #SprBlk_Hard     ; spare table full
4298/    23F1 : D6 02 9B                            call L029b              ; SetStatus ???
4299/    23F4 : FC 0D                               ld R15, #13
4300/    23F6 : 8D 03 57                            jp Abort
4301/    23F9 :                     ;
4302/    23F9 : 08 E6               Gns_Chk_Hi      ld R0, R6               ; assume NoHis
4303/    23FB : 76 E7 80                            tm R7, #080h            ; test for NoHis
4304/    23FE : EB 15                               jr NZ, Gns_End
4305/    2400 :                     ;
4306/    2400 : 08 E5                               ld R0, R5               ; assume NoLos
4307/    2402 : 76 E7 40                            tm R7, #040h            ; test for NoLos
4308/    2405 : EB 0E                               jr NZ, Gns_End
4309/    2407 :                     ;
4310/    2407 : 18 E5                               ld R1, R5
4311/    2409 : 22 14                               sub R1, R4              ; otherwise find which is closer
4312/    240B : 22 46                               sub R4, R6
4313/    240D : 08 E6                               ld R0, R6               ; assume Temp2 is closer
4314/    240F : A2 14                               cp R1, R4
4315/    2411 : FB 02                               jr NC, Gns_End
4316/    2413 : 08 E5                               ld R0, R5               ; otherwise Temp1 is closer
4317/    2415 : 09 36               Gns_End         ld 36h, R0
4318/    2417 : D6 05 4F                            call Ext_Pop
4319/    241A : 08 36                               ld R0, 36h
4320/    241C : 8D 02 F6                            jp Bank_Ret
4321/    241F :
4322/    241F :
4323/    241F :                     ;*****************************************************************
4324/    241F :                     ;
4325/    241F :                     ; Procedure: AddSpare {add an element to the spare table}
4326/    241F :                     ;
4327/    241F :                     ;  This procedure is responsible for adding an element to the spare
4328/    241F :                     ;  table. It accepts a 1 byte value describing the location within
4329/    241F :                     ;  the spare table (it is assumed that the caller has already used
4330/    241F :                     ;  GetNewSpare) as well as the LogicalBlockNumber and whether the
4331/    241F :                     ;  block being added to the table is a Spare block or a Bad Block.
4332/    241F :                     ;
4333/    241F :                     ; Inputs: BlockType:          3 BITS {R8/bits 3:1}
4334/    241F :                     ;         SpareType:          BOOLEAN {R8/bit4}
4335/    241F :                     ;         Location:           BYTE {R15}
4336/    241F :                     ;         LogicalBlockNumber: 3 BYTES {R12:14}
4337/    241F :                     ;
4338/    241F :                     ; Outputs: none
4339/    241F :                     ;
4340/    241F :                     ; Global Variables Used: SpareCount
4341/    241F :                     ;
4342/    241F :                     ; Algorithm:
4343/    241F :                     ;
4344/    241F :                     ;  Begin
4345/    241F :                     ;    HeadPtr:=Get_HeadPtr(LogicalBlockNumber)
4346/    241F :                     ;    If HeadPtr.Nil
4347/    241F :                     ;      Then
4348/    241F :                     ;        HeadPtr.Nil:=False
4349/    241F :                     ;        HeadPtr.Ptr:=Location
4350/    241F :                     ;        SegPtrArray[LogicalBlockNumber/bits 10:16]:= HeadPtr
```

```
4351/    241F :                     ;      Else
4352/    241F :                     ;        Ptr:=HeadPtr.Ptr
4353/    241F :                     ;        Ptr:=Get_EoList(Ptr)
4354/    241F :                     ;        Ptr^.Nil:=False
4355/    241F :                     ;        Ptr^.Ptr:=Location
4356/    241F :                     ;    Ptr:=Get_Ptr(Location)
4357/    241F :                     ;    Ptr^.Nil:=True
4358/    241F :                     ;    Ptr^.Used:=True
4359/    241F :                     ;    Ptr^.Useable:=True
4360/    241F :                     ;    Ptr^.Spare:=Spare
4361/    241F :                     ;    Ptr^.Type:=SpareType
4362/    241F :                     ;    Ptr^.Token:=LogicalBlockNumber/bits 0:9
4363/    241F :                     ;    TCS_BitMap(Set, Location) {set the bit map location for the add}
4364/    241F :                     ;    If SpareType=Spare
4365/    241F :                     ;      Then SpareCount:=SpareCount+1
4366/    241F :                     ;      Else BadCount:=BadCount+1
4367/    241F :                     ;  End
4368/    241F :                     ;
4369/    241F :                     ;*****************************************************************
4370/    241F :
4371/    241F : 2C 14               AddSpare        ld R2, #Get_HeadPtr /256
4372/    2421 : 3C AC                               ld R3, #Get_HeadPtr #256
4373/    2423 : D6 02 CB                            call Bank_Call          ; If HeadPtr.Nil ...
4374/    2426 : EB 09                               jr NZ, ADS_Else1
4375/    2428 :                     ;
4376/    2428 : 56 E0 7F                            and R0, #0FFh-Nil       ; Then HeadPtr.Nil:=false
4377/    242B : 42 0F                               or R0, R15              ;      HeadPtr.Ptr:=Location
4378/    242D : 92 02                               lde @RR2, R0            ; create link
4379/    242F : 8B 14                               jr ADS_UpDate
4380/    2431 :                     ;
4381/    2431 : 2C 14               ADS_Else1       ld R2, #Get_EoList /256
4382/    2433 : 3C C6                               ld R3, #Get_EoList #256
4383/    2435 : D6 02 CB                            call Bank_Call          ; search till end of list
4384/    2438 : 56 E1 7F                            and R1, #0FFh-Nil       ; Ptr^.Nil:=false
4385/    243B : 92 12                               lde @RR2, R1
4386/    243D : 06 E3 03                            add R3, #3              ; get Ptr^.Ptr
4387/    2440 : 16 E2 00                            adc R2, #0
4388/    2443 : 92 F2                               lde @RR2, R15           ; create link
4389/    2445 : 08 EF               ADS_UpDate      ld R0, R15              ; get structure ptr
4390/    2447 : 2C 15                               ld R2, #Get_Ptr /256
4391/    2449 : 3C 1F                               ld R3, #Get_Ptr #256
4392/    244B : D6 02 CB                            call Bank_Call          ; create a read ptr out of it
4393/    244E : 0C E0                               ld R0, #Nil+Used+Useable
4394/    2450 : 42 08                               or R0, R8               ; merge Spare/Bad Block/Type info
4395/    2452 : 76 35 04                            tm DiskStat, #User_Type
4396/    2455 : 6B 05                               jr Z, ADS_UpDt1
4397/    2457 : 46 E0 02                            or R0, #002h
4398/    245A : 8B 03                               jr ADS_UpDt2
4399/    245C : 46 E0 08            ADS_UpDt1       or R0, #008h
4400/    245F : 92 02               ADS_UpDt2       lde @RR2, R0
4401/    2461 : A0 E2                               incw RR2                ; point to element.HiToken
4402/    2463 : 08 ED                               ld R0, R13              ; get HiToken
4403/    2465 : 56 E0 03                            and R0, #3
4404/    2468 : 92 02                               lde @RR2, R0
4405/    246A : A0 E2                               incw RR2                ; piont to element.LoToken
4406/    246C : 92 E2                               lde @RR2, R14           ; store LoToken
4407/    246E : CC 40                               ld R12, #SetBitMap
4408/    2470 : D8 EF                               ld R13, R15
4409/    2472 : 2C 14                               ld R2, #TSC_BitMap /256
4410/    2474 : 3C E3                               ld R3, #TSC_BitMap #256
4411/    2476 : D6 02 CB                            call Bank_Call          ; update the bit map
4412/    2479 : 8D 02 F6                            jp Bank_Ret
4413/    247C :
4414/    247C :
4415/    247C :                     ;*****************************************************************
4416/    247C :                     ;
4417/    247C :                     ; Procedure: DeleteSpare {delete an element from the spare table}
4418/    247C :                     ;
4419/    247C :                     ;  This procedure is responsible for deleting an element from the spare
4420/    247C :                     ;  table. It accepts a 1 byte value describing the location within
4421/    247C :                     ;  the spare table (it is assumed that the caller has already used
4422/    247C :                     ;  GetNewSpare) as well as the LogicalBlockNumber.
4423/    247C :                     ;
4424/    247C :                     ; Inputs: Location:           BYTE {R15}
4425/    247C :                     ;         LogicalBlockNumber: 3 BYTES {R12:14}
4426/    247C :                     ;
4427/    247C :                     ; Outputs: none
4428/    247C :                     ;
4429/    247C :                     ; Local Variables: Ptr1^.Status: BYTE {ScrReg5}
4430/    247C :                     ;
4431/    247C :                     ; Global Variables Used: SpareCount
4432/    247C :                     ;
4433/    247C :                     ; Algorithm:
4434/    247C :                     ;
4435/    247C :                     ;  Begin
4436/    247C :                     ;    Location:=CnvrtLogical(Load_Logical)
4437/    247C :                     ;    If not(CnvrtLogical.Found) Then Abort
4438/    247C :                     ;    If Get_Head(LogicalBlockNumber).Ptr = Location
4439/    247C :                     ;      Then Head.Nil:=true
4440/    247C :                     ;      Else
4441/    247C :                     ;        Ptr1:=Get_Ptr(Location)
4442/    247C :                     ;        If Ptr1^.Nil=true
4443/    247C :                     ;          Then Ptr(PreviousElement)^.Nil:=true
4444/    247C :                     ;          Else Ptr(PreviousElement)^.Ptr:=Ptr1^.Ptr
4445/    247C :                     ;    zero out the deleted element
4446/    247C :                     ;    TCS_BitMap(Clear, Location) {free up the bit map location}
4447/    247C :                     ;  End
4448/    247C :                     ;
4449/    247C :                     ;*****************************************************************
4450/    247C :
4451/    247C : D6 04 18            DeleteSpare     call Load_Logical
4452/    247F : 2C 1B                               ld R2, #CnvrtLogical /256
4453/    2481 : 3C 04                               ld R3, #CnvrtLogical #256
4454/    2483 : D6 02 CB                            call Bank_Call
4455/    2486 : F8 E1                               ld R15, R1
4456/    2488 : EB 05                               jr NZ, Chk_HdPtr
4457/    248A : FC 0E                               ld R15, #14
4458/    248C : 8D 03 57                            jp Abort
4459/    248F :                     ;
4460/    248F : D6 04 18            Chk_HdPtr       call Load_Logical
4461/    2492 : 2C 14                               ld R2, #Get_HeadPtr /256
4462/    2494 : 3C AC                               ld R3, #Get_HeadPtr #256
4463/    2496 : D6 02 CB                            call Bank_Call          ; If Get_HeadPtr ...
4464/    2499 : 09 44                               ld ScrReg4, R0
4465/    249B : 08 EF                               ld R0, R15
4466/    249D : 2C 15                               ld R2, #Get_Ptr /256
4467/    249F : 3C 1F                               ld R3, #Get_Ptr #256
4468/    24A1 : D6 02 CB                            call Bank_Call
```

```
4469/    24A4 : 82 02                              lde R0, @RR2
4470/    24A6 : 76 E0 80                           tm R0, #Nil
4471/    24A9 : 6B 12                              jr Z, Chk_Chain
4472/    24AB : A4 EF 44                           cp ScrReg4, R15
4473/    24AE : EB 0D                              jr NZ, Chk_Chain        ; Else ...
4474/    24B0 :                    ;
4475/    24B0 : 2C 14                              ld R2, #Get_HeadPtr /256
4476/    24B2 : 3C AC                              ld R3, #Get_HeadPtr #256
4477/    24B4 : D6 02 CB                           call Bank_Call
4478/    24B7 : 0C 80                              ld R0, #Nil             ; Then Head.Nil:=true
4479/    24B9 : 92 02                              lde @RR2, R0
4480/    24BB : 8B 53                              jr Zero_Element
4481/    24BD :                    ;
4482/    24BD : 09 45              Chk_Chain       ld ScrReg5, R0
4483/    24BF : 46 E0 80                           or R0, #Nil
4484/    24C2 : 92 02                              lde @RR2, R0            ; break the chain
4485/    24C4 : 08 44                              ld R0, ScrReg4
4486/    24C6 : 2C 14                              ld R2, #Get_EoList /256
4487/    24C8 : 3C C6                              ld R3, #Get_EoList #256
4488/    24CA : D6 02 CB                           call Bank_Call          ; get Ptr(Previous) in ScrReg0,1
4489/    24CD : 08 45                              ld R0, ScrReg5          ; get the original status back
4490/    24CF : 76 E0 80                           tm R0, #Nil             ; If Ptr1^.Nil
4491/    24D2 : 28 46                              ld R2, ScrReg6          ; get Ptr(Previous)
4492/    24D4 : 38 47                              ld R3, ScrReg7
4493/    24D6 : 6B 09                              jr Z, D_Chk_Else        ; Else ...
4494/    24D8 :                    ;
4495/    24D8 : 82 02                              lde R0, @RR2
4496/    24DA : 46 E0 80                           or R0, #080h
4497/    24DD : 92 02                              lde @RR2, R0
4498/    24DF : 8B 2F                              jr Zero_Element
4499/    24E1 :                    ;
4500/    24E1 : A4 44 EF           D_Chk_Else      cp R15, ScrReg4
4501/    24E4 : EB 09                              jr NZ, Get_Previous
4502/    24E6 : 2C 14                              ld R2, #Get_HeadPtr /256
4503/    24E8 : 3C AC                              ld R3, #Get_HeadPtr #256
4504/    24EA : D6 02 CB                           call Bank_Call
4505/    24ED : 8B 06                              jr Save_Previous
4506/    24EF :                    ;
4507/    24EF : 06 E3 03           Get_Previous    add R3, #3              ; get to Ptr(Previous)^.Ptr
4508/    24F2 : 16 E2 00                           adc R2, #0
4509/    24F5 : 70 E2              Save_Previous   push R2
4510/    24F7 : 70 E3                              push R3
4511/    24F9 : 08 EF                              ld R0, R15
4512/    24FB : 2C 15                              ld R2, #Get_Ptr /256
4513/    24FD : 3C 1F                              ld R3, #Get_Ptr #256
4514/    24FF : D6 02 CB                           call Bank_Call
4515/    2502 : 06 E3 03                           add R3, #3              ; get to Ptr1^.Ptr
4516/    2505 : 16 E2 00                           adc R2, #0
4517/    2508 : 82 02                              lde R0, @RR2
4518/    250A : 50 E3                              pop R3
4519/    250C : 50 E2                              pop R2
4520/    250E : 92 02                              lde @RR2, R0            ; Ptr(Previous)^.Ptr:=Ptr1^.Ptr
4521/    2510 : 08 EF              Zero_Element    ld R0, R15              ; get Ptr1 once more
4522/    2512 : 2C 15                              ld R2, #Get_Ptr /256
4523/    2514 : 3C 1F                              ld R3, #Get_Ptr #256
4524/    2516 : D6 02 CB                           call Bank_Call
4525/    2519 : 0C FF                              ld R0, #0FFh            ; initial element
4526/    251B : 1C 04                              ld R1, #4               ; zero 4 bytes
4527/    251D : 92 02              Zero_E_Lp       lde @RR2, R0
4528/    251F : A0 E2                              incw RR2
4529/    2521 : 1A FA                              djnz R1, Zero_E_Lp
4530/    2523 : CC 20                              ld R12, #ClearBitMap
4531/    2525 : D8 EF                              ld R13, R15
4532/    2527 : 2C 14                              ld R2, #TSC_BitMap /256
4533/    2529 : 3C E3                              ld R3, #TSC_BitMap #256
4534/    252B : D6 02 CB                           call Bank_Call
4535/    252E : 8D 02 F6                           jp Bank_Ret
4536/    2531 :
4537/    2531 :
4538/    2531 : 46 35 08           ReSeek          or DiskStat, #Offset_On
4539/    2534 : 56 35 FE                           and DiskStat, #0FFh-Offset_Set
4540/    2537 : D6 04 DB                           call L04db
4541/    253A : 80 E0                              decw  RR0
4542/    253C : D6 04 D0                           call L04d0
4543/    253F : C8 3A                              ld R12,Cylinder
4544/    2541 : D8 3B                              ld R13,Cylinder+1
4545/    2543 : E8 3C                              ld R14,Head
4546/    2545 : F8 3D                              ld R15,Sector
4547/    2547 :
4548/    2547 :
4549/    2547 :                    ;*********************************************************************
4550/    2547 :                    ;
4551/    2547 :                    ; Procedure: Seek
4552/    2547 :                    ;
4553/    2547 :                    ;  This procedure accepts cylinder, head, and sector values
4554/    2547 :                    ;  and then calls PositionHeads.
4555/    2547 :                    ;
4556/    2547 :                    ; Inputs:  Cylinder: WORD {R12}
4557/    2547 :                    ;          Head:     BYTE {R14}
4558/    2547 :                    ;          Sector:   BYTE {R15}
4559/    2547 :                    ;
4560/    2547 :                    ; Outputs: none
4561/    2547 :                    ;
4562/    2547 :                    ; Global Variables Changed: Cylinder, Head, Sector
4563/    2547 :                    ;
4564/    2547 :                    ; Algorithm:
4565/    2547 :                    ;
4566/    2547 :                    ;  Begin
4567/    2547 :                    ;    LastSeekAddress:=CurrentSeekAddress
4568/    2547 :                    ;    DiskStat.Parked:=false
4569/    2547 :                    ;    If DiskStat.SeekComplete
4570/    2547 :                    ;      Then
4571/    2547 :                    ;        i:=4
4572/    2547 :                    ;        While i>0 and not(PositionHeads(Wait, Dmt_Seek, Cylinder,
4573/    2547 :                    ;                                        Head, Sector) Do
4574/    2547 :                    ;          If not(Recovery) Then Abort
4575/    2547 :                    ;          i:=i-1
4576/    2547 :                    ;        If i=0 Then Abort
4577/    2547 :                    ;      Else wait up to 1 sec for ServoReady, if timeout Abort
4578/    2547 :                    ;    DiskStat.SeekComplete:=true
4579/    2547 :                    ;    DiskStat.OnTrack:=true
4580/    2547 :                    ;    GlobalCylinder:=Cylinder
4581/    2547 :                    ;    SelectHead( Head )
4582/    2547 :                    ;    GlobalSector:=Sector
4583/    2547 :                    ;    SectorCount:=SectorCount+1
4584/    2547 :                    ;    Chk_Offset
4585/    2547 :                    ;  End
4586/    2547 :                    ;
```

```
4587/    2547 :                     ;*********************************************************************
4588/    2547 :
4589/    2547 : D6 05 22            Seek            call Ext_Push
4590/    254A : 2C 12                               ld R2, #LastSeek_Stat /256      ; save last seek address
4591/    254C : 3C 4F                               ld R3, #LastSeek_Stat #256
4592/    254E : 0C 3A                               ld R0, #Cylinder
4593/    2550 : 1C 04                               ld R1, #4
4594/    2552 : 93 02               Seek_LSk        ldei @RR2, @R0          ; copy Cylinder, Head, Sector
4595/    2554 : 1A FC                               djnz R1, Seek_Lsk       ; to RAM 124Fh
4596/    2556 : 4C 04                               ld R4, #4               ; four attempts
4597/    2558 :                     ;
4598/    2558 :                     ; **** PosHeads inline
4599/    2558 : 0C 01               Seek_Lp         ld R0, #1               ; do all servoing off of head 1
4600/    255A : D6 26 6C                            call SelectHead
4601/    255D : D6 26 82                            call ServoOk            ; test if Servo is in a reasonable state
4602/    2560 : 6B 7F                               jr Z, PosHds_6          ; leave if servo can't be made useable
4603/    2562 :                     ; *** CalcMagDir inline
4604/    2562 : A8 38                               ld R10, Cur_Cyl         ; get current cylinder
4605/    2564 : B8 39                               ld R11, Cur_Cyl+1
4606/    2566 : 22 BD                               sub R11, R13            ; subtract target cylinder
4607/    2568 : 32 AC                               sbc R10, R12
4608/    256A : 1B 04                               jr LT, C_MagDir_Else    ; negative --> go backwards
4609/    256C : 0C 00                               ld R0, #Hd_Dir_Rev      ; set direction negative
4610/    256E : 8B 0C                               jr S_Glbl_Cyl
4611/    2570 :                     ;
4612/    2570 : 60 EB               C_MagDir_Else   com R11                 ; 2's complement sutraction result
4613/    2572 : 60 EA                               com R10
4614/    2574 : 06 EB 01                            add R11, #1
4615/    2577 : 16 EA 00                            adc R10, #0
4616/    257A : 0C 08                               ld R0, #Hd_Dir_Frwd     ; set direction positive
4617/    257C :                     ;
4618/    257C : 18 EA               S_Glbl_Cyl      ld R1, R10              ; check for no seek condition
4619/    257E : 42 1B                               or R1, R11
4620/    2580 : 6B 5B                               jr Z, PosHds_5          ; distance is zero!
4621/    2582 : 42 A0                               or R10, R0              ; merge in direction
4622/    2584 :                     ; *** CalcMagDir inline
4623/    2584 : 56 35 6C                            and DiskStat, #0FFh-On_Track-Parked-SeekComplete-Offset_Set
4624/    2587 : 76 35 08                            tm DiskStat, #Offset_On
4625/    258A : EB 12                               jr NZ, GtSk_LdAO
4626/    258C : 76 32 80                            tm Excpt_Stat, #Recovery
4627/    258F : 2C 00                               ld R2, #0               ; default is no offsets
4628/    2591 : 6B 16                               jr Z, GtSk_LdNo
4629/    2593 :                     ;
4630/    2593 : D6 27 39                            call Get_Zone
4631/    2596 : 2C 00                               ld R2, #0               ; default is no offsets
4632/    2598 : 1B 0F                               jr LT, GtSk_LdNo
4633/    259A :                     ;
4634/    259A : 28 E0                               ld R2, R0               ; set amount of offset
4635/    259C : 8B 02                               jr GS_LdAO1
4636/    259E :                     ;
4637/    259E : 2C 40               GtSk_LdAO       ld R2, #Off_Auto        ; select access with auto offset for Nisha
4638/    25A0 : 0C 90               GS_LdAO1        ld R0, #Access_Offset
4639/    25A2 : 76 3E 01                            tm 3Eh, #HDA_Type       ; do we have a Nisha HDA?
4640/    25A5 : 6B 04                               jr Z, GS_LdNo1          ;  yes -->
4641/    25A7 : 2C 00                               ld R2, #0               ; else select simple access for Rodime
4642/    25A9 :                     ;
4643/    25A9 : 0C 80               GtSk_LdNo       ld R0, #Access
4644/    25AB : 42 0A               GS_LdNo1        or R0, R10              ; plus direction & MSB magnitude
4645/    25AD : 18 EB                               ld R1, R11              ; LSB magnitude
4646/    25AF : 3C 00                               ld R3, #0
4647/    25B1 : D6 27 71                            call ServoCmnd1
4648/    25B4 : D6 04 7C                            call L047c
4649/    25B7 : 76 35 08                            tm DiskStat, #Offset_On ; check for auto_offset
4650/    25BA : 6B 03                               jr Z, PosHds_4
4651/    25BC :                     ;
4652/    25BC : 46 35 01                            or DiskStat, #Offset_Set
4653/    25BF : AC 61               PosHds_4        ld R10, #25000 /256     ; intermediate timer of 1sec
4654/    25C1 : BC A8                               ld R11, #25000 #256
4655/    25C3 : 80 EA               PosHds_42       decw RR10
4656/    25C5 : 6B 16                               jr Z, PosHds_5
4657/    25C7 : D6 02 8E                            call L028e
4658/    25CA : 70 E0                               push R0
4659/    25CC : D6 04 7C                            call L047c
4660/    25CF : 0C 80                               ld R0, #128
4661/    25D1 : 0A FE               PosHds_41       djnz R0, PosHds_41
4662/    25D3 : 50 E0                               pop R0
4663/    25D5 : 76 E0 20                            tm R0, #ServoRdy        ; THEN loop until timeout OR ServoRdy
4664/    25D8 : 6B E9                               jr Z, PosHds_42
4665/    25DA :                     ;
4666/    25DA : 46 35 02                            or DiskStat, #SeekComplete
4667/    25DD : C9 38               PosHds_5        ld Cur_Cyl, R12
4668/    25DF : D9 39                               ld Cur_Cyl+1, R13
4669/    25E1 :                     ;
4670/    25E1 : D6 26 51            PosHds_6        call Chk_SStat
4671/    25E4 :                     ; **** PosHeads inline
4672/    25E4 : EB 2A                               jr NZ, Seek_End
4673/    25E6 : 56 35 FE                            and DiskStat, #0FFh-Offset_Set  ; remove any offsets
4674/    25E9 : 76 32 80                            tm Excpt_Stat, #Recovery
4675/    25EC : 6B 17                               jr Z, Seek_Abt
4676/    25EE : 2C 00                               ld R2, #20 /256
4677/    25F0 : 3C 14                               ld R3, #20 #256
4678/    25F2 : D6 01 24                            call MsWait             ; wait 200 ms before retrying
4679/    25F5 : 42 44                               or R4, R4               ; counter expired?
4680/    25F7 : 6D 26 05                            jp Z, Seek_Abt
4681/    25FA : 00 E4                               dec R4
4682/    25FC : ED 25 58                            jp NZ, Seek_Lp          ; go for a next try
4683/    25FF : D6 26 63                            call Seek_Reset         ; do a reset before the last one
4684/    2602 : 8D 25 58                            jp Seek_Lp
4685/    2605 :                     ;
4686/    2605 : D6 03 C2            Seek_Abt        call SetStatus
4687/    2608 : 46 35 02                            or DiskStat, #SeekComplete
4688/    260B : FC 22                               ld R15, #34
4689/    260D : 8D 03 57                            jp Abort
4690/    2610 :                     ;
4691/    2610 : 46 35 82            Seek_End        or DiskStat, #On_Track+SeekComplete
4692/    2613 : C9 3A                               ld Cylinder, R12
4693/    2615 : D9 3B                               ld Cylinder+1, R13
4694/    2617 : 08 EE                               ld R0, R14
4695/    2619 : D6 26 6C                            call SelectHead
4696/    261C : F9 3D                               ld Sector, R15
4697/    261E : 76 3E 01                            tm 3Eh, #HDA_Type       ; Nisha HDA?
4698/    2621 : EB 11                               jr NZ, Seek_End1        ;  no -->
4699/    2623 : 4C 20                               ld R4, #020h
4700/    2625 : 5C 07                               ld R5, #007h
4701/    2627 : 82 04                               lde R0, @RR4            ; get value from RAM 2007 ???
4702/    2629 : A0 E4                               incw RR4
4703/    262B : 82 14                               lde R1, @RR4            ; 2008 ???
4704/    262D : 22 D1                               sub R13, R1
```

```
4705/    262F : 32 C0                              sbc R12, R0
4706/    2631 : 1D 26 3A                           jp LT, L263a
4707/    2634 : 56 00 BF           Seek_End1       and P0, #0FFh-RWI       ; RWI on
4708/    2637 : 8D 26 3D                           jp L263d
4709/    263A : 46 00 40           L263a           or P0, #RWI             ; RWI off
4710/    263D : D6 04 DB           L263d           call L04db
4711/    2640 : A0 E0                              incw RR0                ; seek count ???
4712/    2642 : D6 04 D0                           call L04d0
4713/    2645 : D6 27 2B                           call Chk_Offset
4714/    2648 : D6 02 8E                           call L028e
4715/    264B : D6 05 4F                           call Ext_Pop
4716/    264E : 8D 02 F6                           jp Bank_Ret
4717/    2651 :                    ;
4718/    2651 : D6 02 8E           Chk_SStat       call L028e              ; LoadStatus ???
4719/    2654 : D6 02 65                           call L0265
4720/    2657 : 18 E0                              ld R1, R0               ; get copies of both ServoErr and ServoRdy
4721/    2659 : 60 E0                              com R0                  ; ServoErr := NOT( ServoErr )
4722/    265B : 90 E0                              rl R0                   ; get ServoErr in same position as ServoRdy
4723/    265D : 52 01                              and R0, R1              ; return AND( ServoRdy, NOT( ServoErr ) )
4724/    265F : 56 E0 20                           and R0, #ServoRdy       ; set zero flag
4725/    2662 : AF                                 ret
4726/    2663 :                    ;
4727/    2663 : D6 29 22           Seek_Reset      call ResetServo
4728/    2666 : 0C 40                              ld R0, #DataRecal
4729/    2668 : D6 27 CE                           call Restore
4730/    266B : AF                                 ret
4731/    266C :
4732/    266C :
4733/    266C :                    ;*********************************************************************
4734/    266C :                    ;
4735/    266C :                    ; Procedure: SelectHead
4736/    266C :                    ;
4737/    266C :                    ;  This procedure is responsible for selecting the correct head
4738/    266C :                    ;  on the disk.
4739/    266C :                    ;
4740/    266C :                    ; Inputs: Head: BYTE {R0}
4741/    266C :                    ;
4742/    266C :                    ; Outputs: none
4743/    266C :                    ;
4744/    266C :                    ;*********************************************************************
4745/    266C :
4746/    266C : 09 3C              SelectHead      ld Head, R0
4747/    266E : 56 00 CF                           and P0, #0FFh-Hs0-Hs1   ; select Head 0
4748/    2671 : 76 3C 01                           tm Head, #001h
4749/    2674 : 6B 03                              jr Z, Sel_Head1
4750/    2676 : 46 00 10                           or P0, #Hs0
4751/    2679 : 76 3C 02           Sel_Head1       tm Head, #002h
4752/    267C : 6B 03                              jr Z, SelHd_Ret
4753/    267E : 46 00 20                           or P0, #Hs1
4754/    2681 : AF                 SelHd_Ret       ret
4755/    2682 :
4756/    2682 :
4757/    2682 :                    ;*********************************************************************
4758/    2682 :                    ;
4759/    2682 :                    ; Function: ServoOk
4760/    2682 :                    ;
4761/    2682 :                    ;  This function is responsible for determining if the servo is in a
4762/    2682 :                    ;  reasonable state to perform commands. In other words if ServoReady
4763/    2682 :                    ;  and NOT(ServoError) then the servo is healthy, wealthy and wise.
4764/    2682 :                    ;  If, on the other hand, ServoError is active then the state of
4765/    2682 :                    ;  ServoReady determines the type of action to perform to try to get
4766/    2682 :                    ;  the servo back to a nice state:
4767/    2682 :                    ;
4768/    2682 :                    ;  ServoError and ServoReady: Read Status
4769/    2682 :                    ;  ServorError and Not(ServoReady): do Data Recal
4770/    2682 :                    ;
4771/    2682 :                    ; Inputs: none
4772/    2682 :                    ;
4773/    2682 :                    ; Outputs:      ServOk: BOOLEAN {Zero Flag, True if NOT(ServoOk) }
4774/    2682 :                    ;               RecalcMagDir: BYTE {R0}
4775/    2682 :                    ;
4776/    2682 :                    ; Local Variables:      Retry   : BYTE {R15}
4777/    2682 :                    ;                       SioRetry: BYTE {R8}
4778/    2682 :                    ;                       Done    : BOOLEAN { }
4779/    2682 :                    ;                       ControllerStatusPTR : PTR {RR12}
4780/    2682 :                    ;
4781/    2682 :                    ; Algorithm:
4782/    2682 :                    ;
4783/    2682 :                    ; BEGIN
4784/    2682 :                    ;   IF Recovery
4785/    2682 :                    ;     THEN
4786/    2682 :                    ;       IF ServoError
4787/    2682 :                    ;         THEN
4788/    2682 :                    ;           IF Write_Operation THEN WrBuf_To_Buf2
4789/    2682 :                    ;           Restore( DataRecal )
4790/    2682 :                    ;           Buf2_To_WrBuf
4791/    2682 :                    ;         ELSE
4792/    2682 :                    ;           IF NOT( ServoReady )
4793/    2682 :                    ;             THEN
4794/    2682 :                    ;               IF Write_Operation THEN WrBuf_To_Buf2
4795/    2682 :                    ;               S_Reset
4796/    2682 :                    ;               Buf2_To_WrBuf
4797/    2682 :                    ;   ServoOk := NOT( ServoError )
4798/    2682 :                    ; END
4799/    2682 :                    ;
4800/    2682 :                    ;*********************************************************************
4801/    2682 :
4802/    2682 : 76 32 80           ServoOk         tm Excpt_Stat, #Recovery        ; IF Recovery
4803/    2685 : 6B 38                              jr Z, S_Ok_End
4804/    2687 :                    ;
4805/    2687 : D6 02 8E                           call L028e
4806/    268A : D6 02 65                           call L0265
4807/    268D : 76 E0 10                           tm R0, #ServoErr        ; IF ServoError
4808/    2690 : 6B 1F                              jr Z, SOk_ChkRdy
4809/    2692 :                    ;
4810/    2692 : 76 E0 20                           tm R0, #020h
4811/    2695 : 6B 10                              jr Z, L26a7
4812/    2697 : D6 26 C5                           call Chk_MvWrData
4813/    269A : 0C 00                              ld R0, #0
4814/    269C : 1C 00                              ld R1, #0
4815/    269E : 2C 00                              ld R2, #0
4816/    26A0 : 3C 01                              ld R3, #1
4817/    26A2 : D6 27 A9                           call ServoStatus1
4818/    26A5 : 8B 15                              jr L26bc
4819/    26A7 : D6 26 C5           L26a7           call Chk_MvWrData
4820/    26AA : 0C 40                              ld R0, #DataRecal
4821/    26AC : D6 27 CE                           call Restore
4822/    26AF : 8B 0B                              jr L26bc
```

```
4823/    26B1 : 76 E0 20            SOk_ChkRdy      tm R0, #ServoRdy        ; IF NOT( ServoErr )
4824/    26B4 : EB 09                               jr NZ, S_Ok_End         ; AND ServoRdy
4825/    26B6 :                     ;
4826/    26B6 : D6 26 C5            SOk_Park        call Chk_MvWrData       ; save write data if Write_Op
4827/    26B9 : D6 26 63                            call Seek_Reset
4828/    26BC : D6 26 D1            L26bc           call L26d1
4829/    26BF : D6 26 51            S_Ok_End        call Chk_SStat
4830/    26C2 : 8D 02 F6                            jp Bank_Ret
4831/    26C5 :                     ;
4832/    26C5 :                     ;*******************
4833/    26C5 :                     ;
4834/    26C5 : D6 03 C2            Chk_MvWrData    call SetStatus
4835/    26C8 : 76 35 20                            tm DiskStat, #Wr_Op     ; check for write_op
4836/    26CB : 6B 03                               jr Z, MvWr_End
4837/    26CD : D6 20 56                            call WrBuf_To_Buf2
4838/    26D0 : AF                  MvWr_End        ret
4839/    26D1 :                     ;
4840/    26D1 :                     ;*******************
4841/    26D1 :                     ;
4842/    26D1 : D6 21 22            L26d1           call UpDate_Hdr
4843/    26D4 : EB 05                               jr NZ, L26db
4844/    26D6 : 0C 40                               ld R0, #DataRecal
4845/    26D8 : D6 27 CE                            call Restore
4846/    26DB : D6 20 61            L26db           call Buf2_To_WrBuf
4847/    26DE : AF                                  ret
4848/    26DF :
4849/    26DF :
4850/    26DF :                     ;*********************************************************************
4851/    26DF :                     ;
4852/    26DF :                     ; Procedure: Auto_Offset
4853/    26DF :                     ;
4854/    26DF :                     ;  This procedure is used to set auto_offset to the servo processor.
4855/    26DF :                     ;
4856/    26DF :                     ; Inputs: none
4857/    26DF :                     ;
4858/    26DF :                     ; Outputs: none
4859/    26DF :                     ;
4860/    26DF :                     ; Algorithm:
4861/    26DF :                     ;
4862/    26DF :                     ; BEGIN
4863/    26DF :                     ;   Temp := Head
4864/    26DF :                     ;   SelectHead( 1 )
4865/    26DF :                     ;   ServoCmnd( Offset, 0, Off_Auto, S_Rate_57_6 )
4866/    26DF :                     ;   @Get_Zone := ServoStatus( 1 ).Offset_Value
4867/    26DF :                     ;   DiskStat.Offset_Set := True
4868/    26DF :                     ; END
4869/    26DF :                     ;
4870/    26DF :                     ;*********************************************************************
4871/    26DF :
4872/    26DF : D6 05 22            Auto_Offset     call Ext_Push
4873/    26E2 : 70 3C                               push Head               ; save the current head value
4874/    26E4 : 0C 01                               ld R0, #1               ; select head 1 for offsetting
4875/    26E6 : D6 26 6C                            call SelectHead
4876/    26E9 : 76 3E 01                            tm 3Eh, #HDA_Type       ; do we have aRodime mechanism?
4877/    26EC : EB 2F                               jr NZ, Auto_Off_Ret     ;  yes --> exit
4878/    26EE :                     ;
4879/    26EE : 0C 90                               ld R0, #Access_Offset
4880/    26F0 : 1C 00                               ld R1, #0
4881/    26F2 : 2C 40                               ld R2, #Off_Auto
4882/    26F4 : 3C 00                               ld R3, #0
4883/    26F6 : D6 27 71                            call ServoCmnd1
4884/    26F9 : 0C 00                               ld R0, #ReadStatus
4885/    26FB : 1C 00                               ld R1, #0
4886/    26FD : 2C 00                               ld R2, #0
4887/    26FF : 3C 01                               ld R3, #1               ; read servo stat 1 to find offset value
4888/    2701 : D6 27 A9                            call ServoStatus1
4889/    2704 : 2C 14                               ld R2, #(SStatus0+1) /256       ; get offset amount
4890/    2706 : 3C 91                               ld R3, #(SStatus0+1) #256
4891/    2708 : 82 02                               lde R0, @RR2
4892/    270A : B6 E0 1F                            xor R0, #01Fh           ; get correct polarity
4893/    270D : 56 E0 3F                            and R0, #03Fh           ; mask off unwanted stuff
4894/    2710 : 70 E0                               push R0
4895/    2712 :                     ;
4896/    2712 : C8 3A                               ld R12,Cylinder
4897/    2714 : D8 3B                               ld R13,Cylinder+1
4898/    2716 : D6 27 39                            call Get_Zone
4899/    2719 : 50 E0                               pop R0
4900/    271B : 92 02                               lde @RR2, R0            ; store new value
4901/    271D :                     ;
4902/    271D : 50 E0               Auto_Off_Ret    pop R0                  ; re-select original head
4903/    271F : D6 26 6C                            call SelectHead
4904/    2722 : D6 05 4F                            call Ext_Pop
4905/    2725 : 46 35 01                            or DiskStat, #Offset_Set
4906/    2728 : 8D 02 F6                            jp Bank_Ret
4907/    272B :                     ;
4908/    272B :                     ;*****************
4909/    272B :                     ;
4910/    272B : 76 35 08            Chk_Offset      tm DiskStat, #Offset_On
4911/    272E : 6B 08                               jr Z, Chk_Off_Ret
4912/    2730 : 76 35 01            ChkOff_NoOff    tm DiskStat, #Offset_Set        ; THEN check for offset already on
4913/    2733 : EB 03                               jr NZ, Chk_Off_Ret
4914/    2735 : D6 26 DF                            call Auto_Offset
4915/    2738 : AF                  Chk_Off_Ret     ret
4916/    2739 :
4917/    2739 :
4918/    2739 :                     ;*********************************************************************
4919/    2739 :                     ;
4920/    2739 :                     ; Function: Get_Zone
4921/    2739 :                     ;
4922/    2739 :                     ;  This function is responsible for identifying the zone that the
4923/    2739 :                     ;  heads are currently positioned in; a zone being a section of the
4924/    2739 :                     ;  drive that is assumed to have a homogeneous servo offset
4925/    2739 :                     ;  characteristic.
4926/    2739 :                     ;
4927/    2739 :                     ; Inputs: Cylinder: WORD {RR12}
4928/    2739 :                     ;
4929/    2739 :                     ; Outputs:      Get_Zone : BYTE {R0},
4930/    2739 :                     ;               Get_Zone1 : BOOLEAN {zero flag is set if servo err}
4931/    2739 :                     ;
4932/    2739 :                     ; Algorithm:
4933/    2739 :                     ;
4934/    2739 :                     ; BEGIN
4935/    2739 :                     ;   Get_Zone := Zone_Table[ Cylinder/NbrZones ]
4936/    2739 :                     ;   IF ( Get_Zone <= MinOffset )
4937/    2739 :                     ;     THEN Get_Zone1 := False
4938/    2739 :                     ;     ELSE GEt_Zone1 := True
4939/    2739 :                     ; END
4940/    2739 :                     ;
```

```
4941/    2739 :                     ;*****************************************************************
4942/    2739 :
4943/    2739 : 08 EC               Get_Zone        ld R0, R12              ; use seek address for zone calc
4944/    273B : 18 ED                               ld R1, R13
4945/    273D : 2C 05                               ld R2, #ZoneShift
4946/    273F : CF                                  rcf
4947/    2740 : C0 E0               GtZn_Lp         rrc R0                  ; rotate down the cylinder value
4948/    2742 : C0 E1                               rrc R1
4949/    2744 : 2A FA                               djnz R2, GtZn_Lp
4950/    2746 : 2C 16                               ld R2, #Zone_Table /256
4951/    2748 : 3C 41                               ld R3, #Zone_Table #256
4952/    274A : 02 31                               add R3, R1              ; add offset to table
4953/    274C : 16 E2 00                            adc R2, #0
4954/    274F : 82 02                               lde R0, @RR2            ; get offset value
4955/    2751 : 76 E0 20                            tm R0, #Dir_Frwd        ; mask all but sign bit
4956/    2754 : 1C 00                               ld R1, #Off_Dir_Rev
4957/    2756 : EB 02                               jr NZ, GtZn_Push
4958/    2758 : 1C 80                               ld R1, #Off_Dir_Frwd
4959/    275A : 56 E0 1F            GtZn_Push       and R0, #01Fh
4960/    275D : A6 E0 0A                            cp R0, #MinOffset       ; check for lower bound on offset
4961/    2760 : 70 FC                               push FLAGS
4962/    2762 : 42 01                               or R0, R1               ; merge sign and magnitude
4963/    2764 : 50 FC                               pop FLAGS
4964/    2766 : 8D 02 F6                            jp Bank_Ret
4965/    2769 :
4966/    2769 :
4967/    2769 :
4968/    2769 :                     ;*****************************************************************
4969/    2769 :                     ; Module Srvo2.B1.Assem
4970/    2769 :                     ;
4971/    2769 :                     ; This module holds those routines that make up most of the
4972/    2769 :                     ; auxiliary servo commands.
4973/    2769 :                     ;
4974/    2769 :                     ;       PROCEDURE ServoCmnd( CommandString : 4 BYTES {R0:3} )
4975/    2769 :                     ;       PROCEDURE ServoStatus( CommandString : 4 BYTES {R0:3}
4976/    2769 :                     ;                              BufferPtr : PTR {RR14} )
4977/    2769 :                     ;       FUNCTION ServoStore : BOOLEAN
4978/    2769 :                     ;       FUNCTION ServoLoad( BufferPtr : PTR {RR2} ) : BOOLEAN
4979/    2769 :                     ;       FUNCTION Restore( RecalType : BYTE {R0} ) : BOOLEAN
4980/    2769 :                     ;       PROCEDURE SrvoRcvry
4981/    2769 :                     ;       PROCEDURE Load_SrvoCmnd
4982/    2769 :                     ;       PROCEDURE Park_Heads
4983/    2769 :                     ;       PROCEDURE ResetServo
4984/    2769 :                     ;
4985/    2769 :                     ;*****************************************************************
4986/    2769 :
4987/    2769 :                     ;*****************************************************************
4988/    2769 :                     ;
4989/    2769 :                     ; Procedure: SrvoCmnd  {send the servo a command}
4990/    2769 :                     ;
4991/    2769 :                     ; Inputs:  CommandString: 4 BYTES {R0:3}
4992/    2769 :                     ;
4993/    2769 :                     ; Outputs: none
4994/    2769 :                     ;
4995/    2769 :                     ; Global Variables Changed: SrvoCmndBuffer
4996/    2769 :                     ;
4997/    2769 :                     ; Local Variables: SioRetry: BYTE {R8}
4998/    2769 :                     ;
4999/    2769 :                     ; Algorithm:
5000/    2769 :                     ;
5001/    2769 :                     ;  Begin
5002/    2769 :                     ;    SrvoCmndBuffer[CommandByte]:=R0
5003/    2769 :                     ;    SrvoCmndBuffer[LoDiffByte]:=R1
5004/    2769 :                     ;    SrvoCmndBuffer[OffsetByte]:=R2
5005/    2769 :                     ;    SrvoCmndBuffer[StatusByte]:=R3
5006/    2769 :                     ;    SioRetry:=4
5007/    2769 :                     ;    While not(ServoStore(CommandString) And SioRetry>0 Do
5008/    2769 :                     ;      SioRetry:=SioRetry-1
5009/    2769 :                     ;    If SioRetry=0 Abort
5010/    2769 :                     ;  End
5011/    2769 :                     ;
5012/    2769 :                     ;*****************************************************************
5013/    2769 :
5014/    2769 : 08 EC               ServoCmnd       ld R0, R12
5015/    276B : 18 ED                               ld R1, R13
5016/    276D : 28 EE                               ld R2, R14
5017/    276F : 38 EF                               ld R3, R15
5018/    2771 : D6 05 22            ServoCmnd1      call Ext_Push
5019/    2774 : D6 29 0F                            call Load_SrvoCmnd
5020/    2777 : 4C 0A                               ld R4, #10              ; SioRetry:=10
5021/    2779 :                     ;
5022/    2779 : D6 28 7D            Srv_C_Lp        call ServoStore
5023/    277C : EB 1D                               jr nz, Srv_C_End
5024/    277E : 76 32 80                            tm Excpt_Stat, #Recovery
5025/    2781 : 6B 13                               jr Z, Srv_C_Abort
5026/    2783 : 4A F4                               djnz R4, Srv_C_Lp
5027/    2785 : D6 29 22                            call ResetServo
5028/    2788 : D6 05 4F                            call Ext_Pop
5029/    278B : D6 05 22                            call Ext_Push
5030/    278E : D6 29 0F                            call Load_SrvoCmnd
5031/    2791 : D6 28 7D                            call ServoStore
5032/    2794 : EB 05                               jr nz, Srv_C_End
5033/    2796 :                     ;
5034/    2796 : FC 23               Srv_C_Abort     ld R15, #35
5035/    2798 : 8D 03 57                            jp Abort
5036/    279B :                     ;
5037/    279B : D6 05 4F            Srv_C_End       call Ext_Pop
5038/    279E : 8D 02 F6                            jp Bank_Ret
5039/    27A1 :
5040/    27A1 :
5041/    27A1 :                     ;*****************************************************************
5042/    27A1 :                     ;
5043/    27A1 :                     ; Procedure: ServoStatus  {read a status location from the servo}
5044/    27A1 :                     ;
5045/    27A1 :                     ; Inputs:  CommandString: 4 BYTES {R0:3}
5046/    27A1 :                     ;          BufferPtr:     PTR {RR14}
5047/    27A1 :                     ;
5048/    27A1 :                     ; Outputs: none
5049/    27A1 :                     ;
5050/    27A1 :                     ; Global Variables Changed: SrvoCmndBuffer
5051/    27A1 :                     ;                           @BufferPtr
5052/    27A1 :                     ;
5053/    27A1 :                     ; Local Variables: SioRetry: BYTE {R8}
5054/    27A1 :                     ;
5055/    27A1 :                     ; Algorithm:
5056/    27A1 :                     ;
5057/    27A1 :                     ;  Begin
5058/    27A1 :                     ;    ServoCmnd
```

```
5059/    27A1 :                     ;    If not(ServoLoad(BufferPtr)) Then Abort
5060/    27A1 :                     ;  End
5061/    27A1 :                     ;
5062/    27A1 :                     ;*********************************************************************
5063/    27A1 :
5064/    27A1 : 38 E0               ServoStatus     ld R3, R0
5065/    27A3 : 0C 00                               ld R0, #0
5066/    27A5 : 1C 00                               ld R1, #0
5067/    27A7 : 2C 00                               ld R2, #0
5068/    27A9 : D6 05 22            ServoStatus1    call Ext_Push
5069/    27AC : 4C 04                               ld R4, #4               ; retry 4 times
5070/    27AE : D6 27 71            Servo_StLp1     call ServoCmnd1
5071/    27B1 : D6 28 D0                            call ServoLoad
5072/    27B4 : 6B 12                               jr Z, Srv_St_End
5073/    27B6 : 76 32 80                            tm Excpt_Stat, #Recovery
5074/    27B9 : 6B 08                               jr Z, Servo_St_Abort
5075/    27BB : D6 05 4F                            call Ext_Pop
5076/    27BE : D6 05 22                            call Ext_Push
5077/    27C1 : 4A EB                               djnz R4, Servo_StLp1
5078/    27C3 :                     ;
5079/    27C3 : FC 24               Servo_St_Abort  ld R15, #36
5080/    27C5 : 8D 03 57                            jp Abort
5081/    27C8 :                     ;
5082/    27C8 : D6 05 4F            Srv_St_End      call Ext_Pop
5083/    27CB : 8D 02 F6                            jp Bank_Ret
5084/    27CE :
5085/    27CE :
5086/    27CE :                     ;*********************************************************************
5087/    27CE :                     ;
5088/    27CE :                     ; Function: Restore
5089/    27CE :                     ;
5090/    27CE :                     ;  This function performs a Recal operation on the servo.
5091/    27CE :                     ;  Normally, a ReadHeader operation is to be performed after
5092/    27CE :                     ;  the Servo comes ready, and the current cylinder is to be
5093/    27CE :                     ;  subsequently updated. This mechanism (doing the ReadHeader)
5094/    27CE :                     ;  can be turned off by clearing the RdHdrRecal bit in the
5095/    27CE :                     ;  exception working register set.
5096/    27CE :                     ;
5097/    27CE :                     ; Inputs:  RecalType: BYTE {R0}
5098/    27CE :                     ;
5099/    27CE :                     ; Outputs: Restore: BOOLEAN {zero flag, true if not(ServoRdy)}
5100/    27CE :                     ;
5101/    27CE :                     ; Algorithm:
5102/    27CE :                     ;
5103/    27CE :                     ;  Begin
5104/    27CE :                     ;    ServoCmnd(RecalType, 0, 0, 57.6kBaud)
5105/    27CE :                     ;    While not(ServoRdy) And 2 seconds hasn't gone by Do Begin End
5106/    27CE :                     ;    If DiskStat.RdHdrRecal and ServoRdy
5107/    27CE :                     ;      Then If not(UpDate_Cur_Cyl) Then Abort
5108/    27CE :                     ;      Else
5109/    27CE :                     ;        If RecalType=DataRecal
5110/    27CE :                     ;          Then CurCyl:=Init_Cyl
5111/    27CE :                     ;          Else CurCyl:=Max_Cyl
5112/    27CE :                     ;    Restore:=ServoRdy
5113/    27CE :                     ;  End
5114/    27CE :                     ;
5115/    27CE :                     ;*********************************************************************
5116/    27CE :
5117/    27CE : D6 05 22            Restore         call Ext_Push
5118/    27D1 : B0 E1                               clr R1
5119/    27D3 : B0 E2                               clr R2
5120/    27D5 : B0 E3                               clr R3
5121/    27D7 : D6 27 71                            call ServoCmnd1
5122/    27DA : 56 35 7E                            and DiskStat, #0FFh-On_Track-Offset_Set
5123/    27DD : 68 E0                               ld R6, R0
5124/    27DF : E6 38 02                            ld Cur_Cyl, #HiMaxCyl   ; assume FrmtRecal
5125/    27E2 : E6 39 68                            ld Cur_Cyl+1, #LoMaxCyl
5126/    27E5 : A6 E6 70                            cp R6, #FrmtRecal
5127/    27E8 : 6B 11                               jr Z, Rest_Up2
5128/    27EA :                     ;
5129/    27EA : E6 38 02                            ld Cur_Cyl, #Init_HiCyl_B       ; DataRecal track for Nisha
5130/    27ED : E6 39 30                            ld Cur_Cyl+1, #Init_LoCyl_B
5131/    27F0 : 76 3E 01                            tm 3Eh, #HDA_Type       ; do we have aNisha mechanism?
5132/    27F3 : 6B 06                               jr Z, Rest_Up2          ;  yes -->
5133/    27F5 : E6 38 00                            ld Cur_Cyl, #Init_HiCyl_A       ; else DataRecal track for Rodime
5134/    27F8 : E6 39 00                            ld Cur_Cyl+1, #Init_LoCyl_A
5135/    27FB : B0 E5               Rest_Up2        clr R5
5136/    27FD : 4C D5                               ld R4, #0D5h            ; max time to wait is two seconds
5137/    27FF :                     ;
5138/    27FF : D6 02 8E            Restore_Lp      call L028e              ; ??? LoadStatus ???
5139/    2802 : 76 E0 20                            tm R0, #ServoRdy
5140/    2805 : EB 08                               jr NZ, Rest_UpDate
5141/    2807 : 80 E4                               decw RR4
5142/    2809 : EB F4                               jr NZ, Restore_Lp
5143/    280B : B0 E0                               clr R0                  ; pass error status to exit
5144/    280D : 8B 19                               jr Restore_End
5145/    280F :                     ;
5146/    280F : 76 35 40            Rest_UpDate     tm DiskStat, #RdHdrRecal
5147/    2812 : 6B 12                               jr Z, Rest_Up1
5148/    2814 : A6 E6 70                            cp R6, #FrmtRecal       ; don't try to read headers here!
5149/    2817 : 6B 0D                               jr Z, Rest_Up1
5150/    2819 : D6 20 A7                            call UpDate_Cur_Cyl
5151/    281C : EB 08                               jr NZ, Rest_Up1         ; leave if positioned correctly
5152/    281E :                     ;
5153/    281E : D6 03 B6                            call SS_NoHdr
5154/    2821 : FC 25                               ld R15, #025h
5155/    2823 : D6 03 57                            call Abort
5156/    2826 :                     ;
5157/    2826 : 0C 01               Rest_Up1        ld R0, #1
5158/    2828 : 70 E0               Restore_End     push R0                 ; save result
5159/    282A : D6 05 4F                            call Ext_Pop
5160/    282D : 56 35 6E                            and DiskStat, #0FFh-Offset_Set-Parked-On_Track
5161/    2830 : 46 35 02                            or DiskStat, #SeekComplete
5162/    2833 : 50 E0                               pop R0
5163/    2835 : 42 00                               or R0, R0               ; set zero flag
5164/    2837 : 8D 02 F6                            jp Bank_Ret
5165/    283A :
5166/    283A :
5167/    283A :                     ;*********************************************************************
5168/    283A :                     ;
5169/    283A :                     ; Procedure: SrvoRcvry  {servo recovery}
5170/    283A :                     ;
5171/    283A :                     ;  This procedure's responsibility is to do everything within reason
5172/    283A :                     ;  to make certain that the Servo Processor is Healthy, Wealthy, and
5173/    283A :                     ;  Wise; and if it can't then there is no point in using the Servo.
5174/    283A :                     ;
5175/    283A :                     ; Inputs:  none
5176/    283A :                     ;
```

```
5177/    283A :                     ; Outputs: none
5178/    283A :                     ;
5179/    283A :                     ; Local Variables: i:     BYTE {R8}
5180/    283A :                     ;                  Error: BOOLEAN {R9}
5181/    283A :                     ;
5182/    283A :                     ; Global Variabled Changed: On_Track, Cur_Cyl
5183/    283A :                     ;
5184/    283A :                     ; Global Variables Used: Cylinder, Head, Sector
5185/    283A :                     ;
5186/    283A :                     ; Algorithm:
5187/    283A :                     ;
5188/    283A :                     ;  Begin
5189/    283A :                     ;    ZeroHeader
5190/    283A :                     ;    Error:=false
5191/    283A :                     ;    If ServoError
5192/    283A :                     ;      Then Error:=not(ServoOk)
5193/    283A :                     ;      Else
5194/    283A :                     ;       Error:=not(ReadHdr)
5195/    283A :                     ;       If not(Error)
5196/    283A :                     ;        Then
5197/    283A :                     ;         If ReadHdr.Cylinder<>Cylinder
5198/    283A :                     ;          Then
5199/    283A :                     ;           Cur_Cyl:=ReadHdr.Cylinder
5200/    283A :                     ;           On_Track:=false
5201/    283A :                     ;    If not(On_Track) Then Seek(Cylinder, Head, Sector)
5202/    283A :                     ;  End
5203/    283A :                     ;
5204/    283A :                     ;*****************************************************************
5205/    283A :
5206/    283A : D6 05 22            SrvoRcvry       call Ext_Push           ; save caller's stuff
5207/    283D : D6 05 8E                            call ZeroHeader
5208/    2840 : D6 26 51                            call Chk_SStat
5209/    2843 : EB 03                               jr NZ, Srvo_R_Sok
5210/    2845 : D6 26 82                            call ServoOk
5211/    2848 :                     ;
5212/    2848 : D6 21 22            Srvo_R_Sok      call UpDate_Hdr         ; If ReadHdr.Cylinder<>Cylinder
5213/    284B : EB 0A                               jr NZ, Srvo_R_Leave
5214/    284D : 0C 01                               ld R0, #1               ; status byte 1
5215/    284F : 1C 02                               ld R1, #Stat_Seek
5216/    2851 : D6 02 9B                            call L029b              ; ??? SetStatus ???
5217/    2854 : D6 25 31                            call ReSeek
5218/    2857 : D6 05 4F            Srvo_R_Leave    call Ext_Pop
5219/    285A : 8D 02 F6                            jp Bank_Ret
5220/    285D :
5221/    285D :
5222/    285D :                     ;*****************************************************************
5223/    285D :                     ;
5224/    285D :                     ; Procedure: Park_Heads
5225/    285D :                     ;
5226/    285D :                     ;  This procedure moves the heads in a closed-loop fashion
5227/    285D :                     ;  (access command vs Park command) to a location away from the
5228/    285D :                     ;  user data area. The attempt here is to provide some additional
5229/    285D :                     ;  protection from power failures, and the such, from accidentally
5230/    285D :                     ;  writing bogus data on the disk.
5231/    285D :                     ;
5232/    285D :                     ; Inputs: none
5233/    285D :                     ;
5234/    285D :                     ; Outputs: none
5235/    285D :                     ;
5236/    285D :                     ; Algorithm:
5237/    285D :                     ;
5238/    285D :                     ;  Begin
5239/    285D :                     ;    Seek(HiParkCylinder, LowParkCylinder, 0, 0)
5240/    285D :                     ;    Set_SeekNeeded
5241/    285D :                     ;  End
5242/    285D :                     ;
5243/    285D :                     ;*****************************************************************
5244/    285D :
5245/    285D : 56 35 F7            Park_Heads      and DiskStat, #0FFh-Offset_On
5246/    2860 : CC 02                               ld R12, #ParkCyl_B /256 ; park track for Nisha
5247/    2862 : DC 6C                               ld R13, #ParkCyl_B #256
5248/    2864 : 76 3E 01                            tm 3Eh, #HDA_Type       ; Nisha HDA?
5249/    2867 : 6B 04                               jr Z, Park_Hds1         ;  yes -->
5250/    2869 : CC 01                               ld R12, #ParkCyl_A /256 ;  else select track for Rodime
5251/    286B : DC 30                               ld R13, #ParkCyl_A #256
5252/    286D : B0 EE               Park_Hds1       clr R14
5253/    286F : B0 EF                               clr R15
5254/    2871 : D6 25 47                            call Seek
5255/    2874 : 46 35 10                            or DiskStat, #Parked
5256/    2877 : 56 35 7F                            and DiskStat, #0FFh-On_Track
5257/    287A : 8D 02 F6                            jp Bank_Ret
5258/    287D :
5259/    287D :
5260/    287D :                     ;*****************************************************************
5261/    287D :                     ;
5262/    287D :                     ; Function: ServoStore
5263/    287D :                     ;
5264/    287D :                     ;  The function of this routine is to transmit a command string
5265/    287D :                     ;  to the Servo Processor. The command string is expected to
5266/    287D :                     ;  reside in the global variable SrvoCmndBuffer (thus providing a
5267/    287D :                     ;  record of the last command sent to the servo). ServoStore then
5268/    287D :                     ;  attempts to complete the transmission to the Servo, and returns
5269/    287D :                     ;  a boolean variable indicating whether the transmission was
5270/    287D :                     ;  completed or not.
5271/    287D :                     ;
5272/    287D :                     ; Inputs:  Parent: BYTE {R1}
5273/    287D :                     ;
5274/    287D :                     ; Outputs: ServoStore: BOOLEAN {zero flag, true if transmission failed}
5275/    287D :                     ;
5276/    287D :                     ; Local Variables: Retry: BYTE {R6}
5277/    287D :                     ;                  i:     BYTE {R4}
5278/    287D :                     ;                  j:     BYTE {R5}
5279/    287D :                     ;
5280/    287D :                     ; Algorithm:
5281/    287D :                     ;
5282/    287D :                     ;  Begin
5283/    287D :                     ;    SrvoCmndBuffer[CheckByte]:=GenerateCheckByte(PTR(SrvoCmndBuffer),
5284/    287D :                     ;                                                 Length(SrvoCmndBuffer))
5285/    287D :                     ;    Retry:=10
5286/    287D :                     ;    Repeat
5287/    287D :                     ;      For i:=1 To 5 Do
5288/    287D :                     ;        While IRQ.Serial_Output=false Or not(SioReady) Do
5289/    287D :                     ;          Begin End
5290/    287D :                     ;        Sio.Data:=SrvoCmndBuffer[i]
5291/    287D :                     ;      Retry:=Retry-1
5292/    287D :                     ;      For j:=1 To (value for 250µs wait) Do Begin End
5293/    287D :                     ;    Until not(SioReady) Or Retry=0
5294/    287D :                     ;    For j:=1 To (value for 400µs wait) Do Begin End
```

```
5295/    287D :                     ;      If Retry=0
5296/    287D :                     ;        Then ServoStore:=false
5297/    287D :                     ;        Else ServoStore:=true
5298/    287D :                     ;  End
5299/    287D :                     ;
5300/    287D :                     ;*****************************************************************
5301/    287D :
5302/    287D : D6 05 22            ServoStore      call Ext_Push           ; save context
5303/    2880 : EC 14                               ld R14, #SrvoCmndBuf /256
5304/    2882 : FC 8B                               ld R15, #SrvoCmndBuf #256
5305/    2884 : 8C 04                               ld R8, #S_Cmnd_Len-1
5306/    2886 : D6 04 39                            call Gen_Chk_Byte
5307/    2889 : EC 14                               ld R14, #SrvoCmndBuf /256
5308/    288B : FC 8B                               ld R15, #SrvoCmndBuf #256
5309/    288D : 6C 21                               ld R6, #8500 /256
5310/    288F : 7C 34                               ld R7, #8500 #256
5311/    2891 : 4C 05                               ld R4, #S_Cmnd_Len
5312/    2893 : D6 02 AB                            call L02ab
5313/    2896 :                     ;
5314/    2896 : 76 02 40            Srvo_St_1       tm P2, #SioRdy          ; While not(SioReady)
5315/    2899 : EB 06                               jr NZ, Srvo_St_3
5316/    289B : 80 E6                               decw RR6
5317/    289D : EB F7                               jr NZ, Srvo_St_1
5318/    289F : 8B 20                               jr Srvo_St_Exit
5319/    28A1 :
5320/    28A1 : 56 FA E7            Srvo_St_3       and IRQ, #0FFh-Serial_Out-Serial_In     ; clear old interrupts
5321/    28A4 : 82 0E                               lde R0, @RR14           ; get a command byte
5322/    28A6 : 09 F0                               ld SIO, R0              ; send it to servo
5323/    28A8 : A0 EE                               incw RR14               ; point to next command byte
5324/    28AA :                     ;
5325/    28AA : 76 FA 10            Srvo_St_2       tm IRQ, #Serial_Out     ; While IRQ.Serial_Out=false
5326/    28AD : 6B FB                               jr Z, Srvo_St_2
5327/    28AF :                     ;
5328/    28AF : 4A E5                               djnz R4, Srvo_St_1      ; loop till all command bytes are sent
5329/    28B1 : 56 FA EF                            and IRQ, #0FFh-Serial_Out       ; clear old interrupts
5330/    28B4 : 5C 58                               ld R5, #88
5331/    28B6 : 5A FE               Srvo_Wt_1       djnz R5, Srvo_Wt_1      ; do a 250µs wait
5332/    28B8 : 6C 01                               ld R6, #001h
5333/    28BA : 76 02 40                            tm P2, #SioRdy          ; if not(SioReady) then Servo
5334/    28BD : 6B 02                               jr Z, Srvo_St_Exit      ;  took the bytes and is munchin' on 'em
5335/    28BF : 6C 00                               ld R6, #0
5336/    28C1 : D6 02 C9            Srvo_St_Exit    call L02c9
5337/    28C4 : 42 66                               or R6, R6               ; test for Retry=0
5338/    28C6 : 70 FC                               push FLAGS
5339/    28C8 : D6 05 4F                            call Ext_Pop            ; return to original context
5340/    28CB : 50 FC                               pop FLAGS
5341/    28CD : 8D 02 F6                            jp Bank_Ret
5342/    28D0 :
5343/    28D0 :
5344/    28D0 :                     ;*****************************************************************
5345/    28D0 :                     ;
5346/    28D0 :                     ; Function: ServoLoad
5347/    28D0 :                     ;
5348/    28D0 :                     ;  This function is responsible for reading a status back
5349/    28D0 :                     ;  from the servo. The bytes that it receives are stored into a
5350/    28D0 :                     ;  buffer that is passed into the routine in the form of a PTR.
5351/    28D0 :                     ;  ServoLoad passes back to the caller a boolean variable describing
5352/    28D0 :                     ;  whether the checkbyte was valid (i.e. the transmission was a
5353/    28D0 :                     ;  success).
5354/    28D0 :                     ;
5355/    28D0 :                     ; Inputs:  Parent:    BYTE {R1}
5356/    28D0 :                     ;          BufferPtr: PTR {R2:3}
5357/    28D0 :                     ;
5358/    28D0 :                     ; Outputs: ServoLoad: BOOLEAN {zero flag, true if transmission failed}
5359/    28D0 :                     ;
5360/    28D0 :                     ; Global Variables Changed: Buffer pointed to by RR2
5361/    28D0 :                     ;
5362/    28D0 :                     ; Local Variables: i:          BYTE {R4}
5363/    28D0 :                     ;                  TempBufPtr: PTR {RR10}
5364/    28D0 :                     ;
5365/    28D0 :                     ; Algorithm:
5366/    28D0 :                     ;
5367/    28D0 :                     ;  Begin
5368/    28D0 :                     ;    For i:=1 To 5 Do
5369/    28D0 :                     ;      While not(IRQ.Serial_Input) Do Begin End
5370/    28D0 :                     ;      Buffer[i]:=Sio.Data
5371/    28D0 :                     ;    ServoLoad:=Check_Check_Byte(Ptr(Buffer),5)
5372/    28D0 :                     ;  End
5373/    28D0 :                     ;
5374/    28D0 :                     ;*****************************************************************
5375/    28D0 :
5376/    28D0 : D6 05 22            ServoLoad       call Ext_Push
5377/    28D3 : EC 14                               ld R14, #SStatus0 /256
5378/    28D5 : FC 90                               ld R15, #SStatus0 #256
5379/    28D7 : 6C 21                               ld R6, #8500 /256
5380/    28D9 : 7C 34                               ld R7, #8500 #256
5381/    28DB : 4C 05                               ld R4, #5               ; load i
5382/    28DD : D6 02 AB                            call L02ab
5383/    28E0 : 76 FA 08            Srvo_Ld_Wt      tm IRQ, #Serial_In
5384/    28E3 : EB 09                               jr NZ, Srvo_Ld_Wt1
5385/    28E5 : 80 E6                               decw RR6
5386/    28E7 : EB F7                               jr NZ, Srvo_Ld_Wt
5387/    28E9 : 46 E0 01                            or R0, #1
5388/    28EC : 8B 14                               jr L2902
5389/    28EE : 56 FA F7            Srvo_Ld_Wt1     and IRQ, #0FFh-Serial_In        ; clear old interrupts
5390/    28F1 : 08 F0                               ld R0, SIO              ; read SIO
5391/    28F3 : 92 0E                               lde @RR14, R0           ; and store in buffer
5392/    28F5 : A0 EE                               incw RR14               ; point to next location in buffer
5393/    28F7 : 4A E7                               djnz R4, Srvo_Ld_Wt     ; loop till all bytes are read
5394/    28F9 : EC 14                               ld R14, #SStatus0 /256  ; get original buffer ptr
5395/    28FB : FC 90                               ld R15, #SStatus0 #256
5396/    28FD : 8C 04                               ld R8, #4               ; length of buffer
5397/    28FF : D6 04 28                            call Chk_Chk_Byte
5398/    2902 : 70 FC               L2902           push FLAGS
5399/    2904 : D6 02 C9                            call L02c9
5400/    2907 : D6 05 4F                            call Ext_Pop
5401/    290A : 50 FC                               pop FLAGS
5402/    290C : 8D 02 F6                            jp Bank_Ret
5403/    290F :
5404/    290F :
5405/    290F :                     ;*****************************************************************
5406/    290F :                     ;
5407/    290F :                     ; Procedure: Load_SrvoCmnd  {load servo command buffer}
5408/    290F :                     ;
5409/    290F :                     ;  This procedure loads the global array ServoCmndBuffer
5410/    290F :                     ;  with the information contained within R0:3
5411/    290F :                     ;
5412/    290F :                     ; Inputs: SrvoCmndBuffer[0]: BYTE {R0}
```

```
5413/    290F :                     ;         SrvoCmndBuffer[1]: BYTE {R1}
5414/    290F :                     ;         SrvoCmndBuffer[2]: BYTE {R2}
5415/    290F :                     ;         SrvoCmndBuffer[3]: BYTE {R3}
5416/    290F :                     ;
5417/    290F :                     ; Outputs: none
5418/    290F :                     ;
5419/    290F :                     ;*****************************************************************
5420/    290F :
5421/    290F : 31 40               Load_SrvoCmnd   srp #Wrk_Scr
5422/    2911 :                             assume RP:Wrk_Scr
5423/    2911 : 2C 14                       ld R2, #SrvoCmndBuf /256
5424/    2913 : 3C 8B                       ld R3, #SrvoCmndBuf #256
5425/    2915 : 1C 04                       ld R1, #4               ; load 4 bytes
5426/    2917 : 0C 10                       ld R0, #010h            ; start with what's in SysReg0
5427/    2919 : 93 02               Ld_S_Cmnd_Lp    ldei @RR2, @R0
5428/    291B : 1A FC                       djnz R1, Ld_S_Cmnd_Lp
5429/    291D : 31 10                       srp #Wrk_Sys
5430/    291F :                             assume RP:Wrk_Sys
5431/    291F : 8D 02 F6                    jp Bank_Ret
5432/    2922 :
5433/    2922 :
5434/    2922 :                     ;*****************************************************************
5435/    2922 :                     ;
5436/    2922 :                     ; Procedure: ResetServo
5437/    2922 :                     ;
5438/    2922 :                     ;  This procedure is responsible for performing all the
5439/    2922 :                     ;  necessary tasks in resetting the servo. This includes
5440/    2922 :                     ;  positioning the heads over the data field and keeping
5441/    2922 :                     ;  the world straight about it (cylinder is set to
5442/    2922 :                     ;  MaxDataCylinder because a DataRecal positions the heads
5443/    2922 :                     ;  at the closest data cylinder to the inside)
5444/    2922 :                     ;
5445/    2922 :                     ; Inputs: none
5446/    2922 :                     ;
5447/    2922 :                     ; Outputs: none
5448/    2922 :                     ;
5449/    2922 :                     ; Global Variables Changed: Cylinder
5450/    2922 :                     ;
5451/    2922 :                     ; Algorithm:
5452/    2922 :                     ;
5453/    2922 :                     ;  Begin
5454/    2922 :                     ;    ServoRst:=true
5455/    2922 :                     ;    wait for 50ms to make certain that Reset is valid
5456/    2922 :                     ;    ServoRst:=false
5457/    2922 :                     ;    wait for 1 sec to allow motor and servo to get ready
5458/    2922 :                     ;    If not(ServoRdy) or ServoError Then Abort
5459/    2922 :                     ;    If not(ServoStore(ReadStatus, x, x, 0)
5460/    2922 :                     ;      Then Abort
5461/    2922 :                     ;    If not(ServoLoad(NormalReadStatusBuffer)
5462/    2922 :                     ;      Then Abort
5463/    2922 :                     ;    If not(Restore(DataRecal)) Then Abort
5464/    2922 :                     ;    DiskStatus.On_Track:=false
5465/    2922 :                     ;  End
5466/    2922 :                     ;
5467/    2922 :                     ;*****************************************************************
5468/    2922 :
5469/    2922 : D6 05 22            ResetServo      call Ext_Push
5470/    2925 : 56 F1 FD                    and TMR, #0FFh-T0_CntEn ; halt T0
5471/    2928 : E6 F5 05                    ld PRE0, #5             ; PRE0:=1, continuous run
5472/    292B : E6 F4 01                    ld T0, #1               ; Nisha runs at 58.59kbps (7,5MHz/128)
5473/    292E : 46 F1 03                    or TMR, #T0_CntEn+T0_Load
5474/    2931 : BC 04                       ld R11, #4              ; 4 retries
5475/    2933 :                     ;
5476/    2933 : 56 02 FE            S_Rst_1         and P2, #0FFh-Not_ServoRst
5477/    2936 : 2C 00                       ld R2, #0
5478/    2938 : 3C 80                       ld R3, #080h
5479/    293A : 80 E2               S_Rst_2         decw RR2
5480/    293C : EB FC                       jr NZ, S_Rst_2
5481/    293E : 46 02 01                    or P2, #Not_ServoRst
5482/    2941 : 2C 01                       ld R2, #300 /256
5483/    2943 : 3C 2C                       ld R3, #300 #256
5484/    2945 : D6 01 24                    call MsWait             ; busy wait for 3s
5485/    2948 : 0C 00                       ld R0, #ReadStatus      ; try talking to the Servo
5486/    294A : B0 E1                       clr R1                  ;  (00 00 00 02)
5487/    294C : B0 E2                       clr R2
5488/    294E : 3C 02                       ld R3, #2
5489/    2950 : D6 29 0F                    call Load_SrvoCmnd
5490/    2953 : D6 28 7D                    call ServoStore
5491/    2956 : EB 09                       jr NZ, S_Rst_Ld
5492/    2958 : BA D9                       djnz R11, S_Rst_1
5493/    295A : AC 00                       ld R10, #S_Store
5494/    295C :                     ;
5495/    295C : FC 27               S_Rst_Abort     ld R15, #39
5496/    295E : 8D 03 57                    jp Abort
5497/    2961 :                     ;
5498/    2961 : D6 28 D0            S_Rst_Ld        call ServoLoad
5499/    2964 : AC 01                       ld R10, #S_Load
5500/    2966 : 6B 04                       jr Z, S_Rst_Ld1
5501/    2968 : BA C9                       djnz R11, S_Rst_1
5502/    296A : 8B F0                       jr S_Rst_Abort
5503/    296C :                     ; determine HDA type:
5504/    296C :                     ; Rene works with two different drive mechanisms: Apple's Nisha and Rodime's RO552
5505/    296C :                     ; The current type is indentified from the Servo status response.
5506/    296C : EC 14               S_Rst_Ld1       ld R14, #SStatus0 /256
5507/    296E : FC 90                       ld R15, #SStatus0 #256
5508/    2970 : 82 0E                       lde R0, @RR14
5509/    2972 : A6 E0 01                    cp R0, #1               ; check HDA type
5510/    2975 : EB 0A                       jr NZ, S_Rst_Ld2        ;  01h --> Rodime
5511/    2977 :                     ;
5512/    2977 : 56 3E FE                    and 3Eh, #0FFh-HDA_Type ; set HDA_Type = Nisha
5513/    297A : 0C 70                       ld R0, #FrmtRecal       ; and recalibrate
5514/    297C : D6 27 CE                    call Restore
5515/    297F : 8B 03                       jr S_Rst_Ret
5516/    2981 :                     ;
5517/    2981 : 46 3E 01            S_Rst_Ld2       or 3Eh, #HDA_Type       ; set HDA_Type = Rodime RO552
5518/    2984 : D6 05 4F            S_Rst_Ret       call Ext_Pop
5519/    2987 : 8D 02 F6                    jp Bank_Ret
5520/    298A :
5521/    298A :
5522/    298A :
5523/    298A :                     ;*****************************************************************
5524/    298A :                     ; Module SlfTst.B1.Assem   {Selftest}
5525/    298A :                     ;
5526/    298A :                     ;       FUNCTION: MtrSpd
5527/    298A :                     ;       FUNCTION: TrackCount
5528/    298A :                     ;       FUNCTION: SctrCount
5529/    298A :                     ;       FUNCTION: RwTest
5530/    298A :                     ;
```

```
5531/    298A :                     ;*********************************************************************
5532/    298A :
5533/    298A :
5534/    298A :                     ;*********************************************************************
5535/    298A :                     ;
5536/    298A :                     ; Function: Selftest
5537/    298A :                     ;
5538/    298A :                     ;  This function performs some check to confirm Nisha's integrity.
5539/    298A :                     ;
5540/    298A :                     ;  Inputs: none
5541/    298A :                     ;
5542/    298A :                     ;  Outputs: SlfTst_Result: BYTE
5543/    298A :                     ;
5544/    298A :                     ;*********************************************************************
5545/    298A :
5546/    298A : 46 32 80            SelfTest        or Excpt_Stat, #Recovery        ; try our best to pass this test
5547/    298D : E6 5D 00                            ld 5Dh, #0
5548/    2990 : D6 29 C1                            call MtrSpd
5549/    2993 : 6B 0C                               jr Z, ST_Mtr_Gd
5550/    2995 :                     ;
5551/    2995 : 2C 05                               ld R2, #1500 /256
5552/    2997 : 3C DC                               ld R3, #1500 #256       ; wait another 21 ms
5553/    2999 : D6 01 24                            call MsWait
5554/    299C : D6 29 C1                            call MtrSpd
5555/    299F : EB 1D                               jr NZ, Slf_Leave
5556/    29A1 :                     ;
5557/    29A1 : 56 33 DF            ST_Mtr_Gd       and SlfTst_Result, #0FFh-Disk_Speed
5558/    29A4 : D6 2A 05            Slf_Tracks      call TrackCount
5559/    29A7 : 56 33 EF                            and SlfTst_Result, #0FFh-Servo_Fail
5560/    29AA : D6 2A 1C            Slf_Sectors     call SctrCount
5561/    29AD : 56 33 F7                            and SlfTst_Result, #0FFh-Sector_Cnt
5562/    29B0 : D6 04 6C            Slf_State       call LoadStatus         ; check state machine state
5563/    29B3 : 56 33 FB                            and SlfTst_Result, #0FFh-State_Fail
5564/    29B6 : D6 2A 63            Slf_RwTest      call RwTest             ; see if we can read and write
5565/    29B9 : 6B 03                               jr Z, Slf_Leave
5566/    29BB : 56 33 FD                            and SlfTst_Result, #0FFh-Rw_Fail
5567/    29BE :                     ;
5568/    29BE : 8D 02 F6            Slf_Leave       jp Bank_Ret
5569/    29C1 :
5570/    29C1 :
5571/    29C1 :                     ;*********************************************************************
5572/    29C1 :                     ;
5573/    29C1 :                     ; Function: MtrSpd  {motor speed}
5574/    29C1 :                     ;
5575/    29C1 :                     ;  This function is responsible for measuring the motor
5576/    29C1 :                     ;  speed. This is done by timing the interval between
5577/    29C1 :                     ;  consecutive index marks
5578/    29C1 :                     ;
5579/    29C1 :                     ; Inputs: none
5580/    29C1 :                     ;
5581/    29C1 :                     ; Outputs: MtrSpd: BOOLEAN {zero flag, NOT set if failure}
5582/    29C1 :                     ;
5583/    29C1 :                     ;*********************************************************************
5584/    29C1 :
5585/    29C1 : 56 FA FE            MtrSpd          and IRQ, #0FFh-IRQ_Index        ; clear old event
5586/    29C4 : 0C 01                               ld R0, #1               ; assume failure
5587/    29C6 : 4C 06                               ld R4, #1600 /256       ; load dead man timer
5588/    29C8 : 5C 40                               ld R5, #1600 #256       ;  (22,3ms)
5589/    29CA : 2C 2A                               ld R2, #11000 /256      ; timeout after about 4 rotations
5590/    29CC : 3C F8                               ld R3, #11000 #256      ;  (153,6ms)
5591/    29CE : 76 FA 01            MtrSpd_Lp1      tm IRQ, #Irq_Index      ; wait for index to come around
5592/    29D1 : EB 06                               jr NZ, MtrSpd_Idx       ;  got him
5593/    29D3 : 80 E2                               decw RR2                ; decrement timeout
5594/    29D5 : EB F7                               jr NZ, MtrSpd_Lp1
5595/    29D7 : 8B 29                               jr MtrSpd_Lv
5596/    29D9 :                     ;
5597/    29D9 : 56 FA FA            MtrSpd_Idx      and IRQ, #0FFh-IRQ_Index-Irq_Sector     ; clear events
5598/    29DC : B0 E0                               clr R0                  ; clear counter
5599/    29DE : B0 E1                               clr R1
5600/    29E0 : 76 FA 01            MtrSpd_Lp2      tm IRQ, #Irq_Index      ; wait for one rev
5601/    29E3 : EB 06                               jr NZ, MtrSpd_End       ;  done
5602/    29E5 : A0 E0                               incw RR0                ; 0.0140 ms/lp or 71.6 cnts/ms
5603/    29E7 : 80 E4                               decw RR4                ; decrement timeout
5604/    29E9 : EB F5                               jr NZ, MtrSpd_Lp2
5605/    29EB :                     ;
5606/    29EB : E8 E1               MtrSpd_End      ld R14, R1
5607/    29ED : D8 E0                               ld R13, R0
5608/    29EF : CC 00                               ld R12, #0
5609/    29F1 : D6 21 50                            call L2150              ; get speed within tolerance in R2
5610/    29F4 : 0C 01                               ld R0, #1               ; assume failure
5611/    29F6 : A6 E2 5B                            cp R2, #05Bh            ; accept speeds between 20 and 24 ms
5612/    29F9 : 1B 07                               jr LT, MtrSpd_Lv
5613/    29FB : A6 E2 64                            cp R2, #064h
5614/    29FE : AB 02                               jr GT, MtrSpd_Lv
5615/    2A00 : 0C 00                               ld R0, #0               ; otherwise pass
5616/    2A02 :                     ;
5617/    2A02 : 42 00               MtrSpd_Lv       or R0, R0               ; set flags
5618/    2A04 : AF                                  ret
5619/    2A05 :
5620/    2A05 :
5621/    2A05 :                     ;*********************************************************************
5622/    2A05 :                     ;
5623/    2A05 :                     ; Function: TrackCount
5624/    2A05 :                     ;
5625/    2A05 :                     ;  This function is responsible for counting the number of
5626/    2A05 :                     ;  tracks that are available on the servo.
5627/    2A05 :                     ;
5628/    2A05 :                     ; Inputs: none
5629/    2A05 :                     ;
5630/    2A05 :                     ; Outputs: TrackCount: BOOLEAN {zero flag, NOT set if failure
5631/    2A05 :                     ;
5632/    2A05 :                     ; Algorithm:
5633/    2A05 :                     ;
5634/    2A05 :                     ;  Begin
5635/    2A05 :                     ;   Restore(FrmtRecal)
5636/    2A05 :                     ;   Seek(Cyl=0)
5637/    2A05 :                     ;  End
5638/    2A05 :                     ;
5639/    2A05 :                     ;*********************************************************************
5640/    2A05 :
5641/    2A05 : D6 29 22            TrackCount      call ResetServo         ; try to get the servo in a nice state
5642/    2A08 : 56 35 BF                            and DiskStat, #0FFh-RdHdrRecal  ; don't read any headers!
5643/    2A0B : 0C 40                               ld R0, #DataRecal
5644/    2A0D : D6 27 CE                            call Restore
5645/    2A10 : CC 00                               ld R12, #0              ; cylinder = 0
5646/    2A12 : DC 69                               ld R13, #069h
5647/    2A14 : EC 00                               ld R14, #0              ; head = 0
5648/    2A16 : FC 00                               ld R15, #0              ; sector = 0
```

```
5649/    2A18 : D6 25 47                        call Seek
5650/    2A1B : AF                              ret
5651/    2A1C :
5652/    2A1C :
5653/    2A1C :                     ;*********************************************************************
5654/    2A1C :                     ;
5655/    2A1C :                     ; Function: SctrCount
5656/    2A1C :                     ;
5657/    2A1C :                     ;  This function counts the number of sector marks between
5658/    2A1C :                     ;  successive index marks and returns a true value if
5659/    2A1C :                     ;  the number of sectors counted equals NbrSctrs.
5660/    2A1C :                     ;
5661/    2A1C :                     ; Inputs: none
5662/    2A1C :                     ;
5663/    2A1C :                     ; Outputs: SctrCount: BOOLEAN {zero flag, NOT set if failure}
5664/    2A1C :                     ;
5665/    2A1C :                     ;*********************************************************************
5666/    2A1C :
5667/    2A1C : 8F                  SctrCount       di
5668/    2A1D : 76 3E 01                            tm 3Eh, #HDA_Type       ; Nisha mechanism?
5669/    2A20 : 6B 39                               jr Z, S_Cnt_Exit        ;  yes --> leave
5670/    2A22 : 4C 04                               ld R4, #4               ; try four times
5671/    2A24 :                     ;
5672/    2A24 : B0 E2               S_Cnt_Retry     clr R2                  ; count:=0
5673/    2A26 : 6C 00                               ld R6, #0               ; set timeout counter
5674/    2A28 : 7C 00                               ld R7, #0
5675/    2A2A : 56 FA FE                            and IRQ, #0FFh-Irq_Index        ; clear old index marks
5676/    2A2D : 76 FA 01            S_Cnt_1         tm IRQ, #Irq_Index      ; wait for index mark
5677/    2A30 : EB 06                               jr NZ, S_Cnt_2
5678/    2A32 : 80 E6                               decw RR6
5679/    2A34 : EB F7                               jr NZ, S_Cnt_1
5680/    2A36 : 8B 26                               jr S_Cnt_Abort          ; abort with timeout
5681/    2A38 : 56 FA FA            S_Cnt_2         and IRQ, #0FAh
5682/    2A3B : 76 03 04            S_Cnt_3         tm P3, #IndexMark       ; While not(Index)
5683/    2A3E : EB 0F                               jr NZ, S_Cnt_End
5684/    2A40 : 80 E6                               decw RR6
5685/    2A42 : 6B 1A                               jr Z, S_Cnt_Abort
5686/    2A44 : 76 FA 04                            tm IRQ, #Irq_Sector
5687/    2A47 : 6B F2                               jr Z, S_Cnt_3
5688/    2A49 : 2E                                  inc R2                  ; bump the sector count
5689/    2A4A : 56 FA FB                            and IRQ, #0FFh-Irq_Sector
5690/    2A4D : 8B EC                               jr S_Cnt_3
5691/    2A4F : A6 E2 20            S_Cnt_End       cp R2, #NbrSctrs
5692/    2A52 : 6B 07                               jr Z, S_Cnt_Exit        ; passed, return
5693/    2A54 : 4A CE                               djnz R4, S_Cnt_Retry
5694/    2A56 :                     ;
5695/    2A56 : FC 3B                               ld R15, #59             ; sector count mismatch
5696/    2A58 : 8D 03 57                            jp Abort
5697/    2A5B : 8D 02 F6            S_Cnt_Exit      jp Bank_Ret
5698/    2A5E :                     ;
5699/    2A5E : FC 3C               S_Cnt_Abort     ld R15, #60             ; sector count index mark timeout
5700/    2A60 : 8D 03 57                            jp Abort
5701/    2A63 :
5702/    2A63 :
5703/    2A63 :                     ;*********************************************************************
5704/    2A63 :                     ;
5705/    2A63 :                     ; Function: RwTest
5706/    2A63 :                     ;
5707/    2A63 :                     ;  This test moves the heads over a track away from the data
5708/    2A63 :                     ;  area (useable data area) and performs a write verify
5709/    2A63 :                     ;  on block zero of that track. If a failure occurs, the test
5710/    2A63 :                     ;  will continue on the next sequential sector on that track
5711/    2A63 :                     ;  until all sectors have been written to. If there are no
5712/    2A63 :                     ;  successful transfers on any of the sectors of this track then
5713/    2A63 :                     ;  the test is assumed to have failed.
5714/    2A63 :                     ;
5715/    2A63 :                     ; Inputs: none
5716/    2A63 :                     ;
5717/    2A63 :                     ; Outputs: RwTest: BOOLEAN {zero flag, NOT set if failure
5718/    2A63 :                     ;
5719/    2A63 :                     ; Algorithm:
5720/    2A63 :                     ;
5721/    2A63 :                     ;  Begin
5722/    2A63 :                     ;    Seek(RwTest track)
5723/    2A63 :                     ;    Repeat
5724/    2A63 :                     ;      If not(WrVerCommon) And RdErrCnt=10
5725/    2A63 :                     ;        Then
5726/    2A63 :                     ;          Sector:=Sector+1
5727/    2A63 :                     ;          If Sector>NbrSctrs
5728/    2A63 :                     ;            Then Done:=true
5729/    2A63 :                     ;       Else Done:=true
5730/    2A63 :                     ;    Until Done
5731/    2A63 :                     ;    If Sector>NbrSctrs
5732/    2A63 :                     ;      Then RwTest:=false
5733/    2A63 :                     ;      Else RwTest:=true
5734/    2A63 :                     ;  End
5735/    2A63 :                     ;
5736/    2A63 :                     ;*********************************************************************
5737/    2A63 :
5738/    2A63 : 46 35 40            RwTest          or DiskStat, #RdHdrRecal
5739/    2A66 : 0C 40                               ld R0, #DataRecal
5740/    2A68 : D6 27 CE                            call Restore
5741/    2A6B : 5C 00                               ld R5, #0
5742/    2A6D : 6B 49                               jr Z, RwTest_End
5743/    2A6F :                     ;
5744/    2A6F : CC 02               RwTest1         ld R12, #Tst_HiCyl_B    ; assume values for Nishas
5745/    2A71 : DC 67                               ld R13, #Tst_LoCyl_B
5746/    2A73 : EC 01                               ld R14, #Tst_Head_B
5747/    2A75 : 76 3E 01                            tm 3Eh, #HDA_Type       ; Nisha HDA?
5748/    2A78 : 6B 06                               jr Z, RwTest2           ;  yes -->
5749/    2A7A : CC 01                               ld R12, #Tst_HiCyl_A    ; else use values for Rodime
5750/    2A7C : DC 31                               ld R13, #Tst_LoCyl_A
5751/    2A7E : EC 03                               ld R14, #Tst_Head_A
5752/    2A80 : FC 00               RwTest2         ld R15, #Tst_Sctr
5753/    2A82 : D6 25 47                            call Seek
5754/    2A85 : 5C 20                               ld R5, #NbrSctrs
5755/    2A87 :                     ;
5756/    2A87 : D6 05 8E            RwTest_Lp       call ZeroHeader
5757/    2A8A : 0C 39                               ld R0, #039h
5758/    2A8C : D6 05 A3                            call L05a3
5759/    2A8F : 70 E5                               push R5
5760/    2A91 : 46 35 20                            or DiskStat, #Wr_Op
5761/    2A94 : 2C 10                               ld R2, #WrBlk_Vector /256
5762/    2A96 : 3C 8E                               ld R3, #WrBlk_Vector #256
5763/    2A98 : D6 02 CB                            call Bank_Call
5764/    2A9B : 50 E5                               pop R5
5765/    2A9D : 6B 15                               jr Z, Rw_Next
5766/    2A9F :                     ;
```

```
5767/    2A9F : 0C 00                              ld R0, #0               ; assume a failure
5768/    2AA1 : D6 05 A3                           call L05a3
5769/    2AA4 : 70 E5                              push R5
5770/    2AA6 : 56 35 DF                           and DiskStat, #0FFh-Wr_Op
5771/    2AA9 : 2C 10                              ld R2, #RdBlk_Vector /256
5772/    2AAB : 3C 94                              ld R3, #RdBlk_Vector #256
5773/    2AAD : D6 02 CB                           call Bank_Call
5774/    2AB0 : 50 E5                              pop R5
5775/    2AB2 : EB 04                              jr NZ, RwTest_End
5776/    2AB4 :                          ;
5777/    2AB4 : 20 3D                    Rw_Next   inc Sector
5778/    2AB6 : 5A CF                              djnz R5, RwTest_Lp
5779/    2AB8 : 42 55                    RwTest_End or R5, R5              ; set zero flag
5780/    2ABA : 6B 19                              jr Z, RwTest_Exit       ;  set --> passed!
5781/    2ABC : 1C 39                              ld R1, #039h
5782/    2ABE : 2C 02                              ld R2, #BlockLength /256
5783/    2AC0 : 3C 14                              ld R3, #BlockLength #256
5784/    2AC2 : 4C 10                              ld R4, #RBuffer1 /256   ; RAM 1019h
5785/    2AC4 : 5C 19                              ld R5, #RBuffer1 #256
5786/    2AC6 : 82 04                    L2ac6     lde R0, @RR4
5787/    2AC8 : A2 01                              cp R0, R1
5788/    2ACA : EB 0A                              jr NZ, RwTest_Abort
5789/    2ACC : A0 E4                              incw RR4
5790/    2ACE : 80 E2                              decw  RR2
5791/    2AD0 : EB F4                              jr NZ, L2ac6
5792/    2AD2 : D6 28 5D                           call Park_Heads
5793/    2AD5 : AF                       RwTest_Exit ret
5794/    2AD6 :                          ;
5795/    2AD6 : FC 31                    RwTest_Abort ld R15, #49            ; RW test failed
5796/    2AD8 : 8D 03 57                           jp Abort
5797/    2ADB :
5798/    2ADB :
5799/    2ADB : 44 33 33                 L2adb     or SlfTst_Result,SlfTst_Result
5800/    2ADE : ED 2B 57                           jp NZ, L2b57
5801/    2AE1 : 4C 01                              ld R4, #001h
5802/    2AE3 : 2C 12                    L2ae3     ld R2, #012h
5803/    2AE5 : 3C 55                              ld R3, #055h
5804/    2AE7 : 1C 03                              ld R1, #3
5805/    2AE9 : 0C 1C                              ld R0, #Wrk_Sys+12
5806/    2AEB : 83 02                    L2aeb     ldei @R0, @RR2
5807/    2AED : 1A FC                              djnz R1, L2aeb
5808/    2AEF : 06 EE 01                           add R14, #001h
5809/    2AF2 : 16 ED 00                           adc R13, #0
5810/    2AF5 : CC 00                              ld R12, #0
5811/    2AF7 : 0C 00                              ld R0, #0
5812/    2AF9 : 1C 98                              ld R1, #098h
5813/    2AFB : 2C 34                              ld R2, #034h
5814/    2AFD : D6 04 59                           call L0459
5815/    2B00 : FB 06                              jr NC, L2b08
5816/    2B02 : CC 00                              ld R12, #0
5817/    2B04 : DC 00                              ld R13, #0
5818/    2B06 : EC 00                              ld R14, #0
5819/    2B08 : 2C 12                    L2b08     ld R2, #012h
5820/    2B0A : 3C 55                              ld R3, #055h
5821/    2B0C : 1C 03                              ld R1, #3
5822/    2B0E : 0C 1C                              ld R0, #Wrk_Sys+12
5823/    2B10 : 93 02                    L2b10     ldei @RR2, @R0
5824/    2B12 : 1A FC                              djnz R1, L2b10
5825/    2B14 : 2C 14                              ld R2, #014h
5826/    2B16 : 3C 7F                              ld R3, #07Fh
5827/    2B18 : 1C 03                              ld R1, #3
5828/    2B1A : 0C 1C                              ld R0, #Wrk_Sys+12
5829/    2B1C : 93 02                    L2b1c     ldei @RR2, @R0
5830/    2B1E : 1A FC                              djnz R1, L2b1c
5831/    2B20 : D6 05 22                           call Ext_Push
5832/    2B23 : 56 35 D7                           and DiskStat, #0FFh-Wr_Op-Offset_On
5833/    2B26 : 46 35 04                           or DiskStat, #User_Type
5834/    2B29 : 2C 1C                              ld R2, #OverLap /256
5835/    2B2B : 3C 24                              ld R3, #OverLap #256
5836/    2B2D : D6 02 CB                           call Bank_Call
5837/    2B30 : 56 35 D7                           and DiskStat, #0FFh-Wr_Op-Offset_On
5838/    2B33 : 2C 11                              ld R2, #RW_Common /256
5839/    2B35 : 3C D3                              ld R3, #RW_Common #256
5840/    2B37 : D6 02 CB                           call Bank_Call
5841/    2B3A : 6B 0C                              jr Z, L2b48
5842/    2B3C : 76 3E 20                           tm 3Eh, #B_Block
5843/    2B3F : 6B 0E                              jr Z, L2b4f
5844/    2B41 : 2C 11                              ld R2, #Pro_Rd_BB /256
5845/    2B43 : 3C 4F                              ld R3, #Pro_Rd_BB #256
5846/    2B45 : D6 02 CB                           call Bank_Call
5847/    2B48 : 2C 13                    L2b48     ld R2, #Data_Ex_Handler /256
5848/    2B4A : 3C BD                              ld R3, #Data_Ex_Handler #256
5849/    2B4C : D6 02 CB                           call Bank_Call
5850/    2B4F : D6 05 4F                 L2b4f     call Ext_Pop
5851/    2B52 : 4A 8F                              djnz R4, L2ae3
5852/    2B54 : D6 28 5D                           call Park_Heads
5853/    2B57 : D6 05 BA                 L2b57     call L05ba
5854/    2B5A : 8D 02 F6                           jp Bank_Ret
5855/    2B5D :
5856/    2B5D :
5857/    2B5D :
5858/    2B5D : 8F                       Strt_FreeProcess        di
5859/    2B5E : 31 10                              srp #Wrk_Sys            ; get into a reasonable state
5860/    2B60 :                                  assume RP:Wrk_Sys
5861/    2B60 : B0 FE                              clr SPH
5862/    2B62 : E6 FF 80                           ld SPL, #Stack_Top
5863/    2B65 : 76 3E 04                           tm 3Eh, #004h
5864/    2B68 : 6D 2B CA                           jp Z, Chk_Pk_Jp
5865/    2B6B : D6 04 DB                           call L04db
5866/    2B6E : 76 E0 F8                           tm R0, #0F8h            ; do arm sweep every 2k seeks or so
5867/    2B71 : 6B 20                              jr Z, Chk_Park
5868/    2B73 :                          ;
5869/    2B73 : B0 E0                              clr R0                  ; 10h SeekCount
5870/    2B75 : B0 E1                              clr R1                  ; 11h SeekCount+1
5871/    2B77 : D6 04 D0                           call L04d0
5872/    2B7A : CC 00                              ld R12, #0              ; seek to data recal location
5873/    2B7C : DC 00                              ld R13, #0
5874/    2B7E : EC 00                              ld R14, #0
5875/    2B80 : FC 00                              ld R15, #0
5876/    2B82 : D6 25 47                           call Seek
5877/    2B85 : CC 00                              ld R12, #0
5878/    2B87 : DC 40                              ld R13, #040h
5879/    2B89 : EC 00                              ld R14, #0
5880/    2B8B : FC 00                              ld R15, #0
5881/    2B8D : D6 25 47                           call Seek
5882/    2B90 : 56 35 7F                           and DiskStat, #0FFh-On_Track
5883/    2B93 : 76 35 10                 Chk_Park  tm DiskStat, #Parked
5884/    2B96 : EB 06                              jr NZ, FreeP_NoPark
```

```
5885/    2B98 : D6 2A DB                        call L2adb
5886/    2B9B : 8D 2B CA                        jp Chk_Pk_Jp
5887/    2B9E :                         ;
5888/    2B9E : D6 04 ED                FreeP_NoPark    call L04ed
5889/    2BA1 : 0E                              inc R0
5890/    2BA2 : D6 04 E6                        call L04e6
5891/    2BA5 : 76 E0 0F                        tm R0, #00Fh
5892/    2BA8 : ED 2B CA                        jp NZ, Chk_Pk_Jp
5893/    2BAB : 2C 2B                           ld R2, #SlfTst_Table /256
5894/    2BAD : 3C C0                           ld R3, #SlfTst_Table #256
5895/    2BAF : F0 E0                           swap R0
5896/    2BB1 : 90 E0                           rl R0
5897/    2BB3 : 02 30                           add R3, R0
5898/    2BB5 : 16 E2 00                        adc R2, #0
5899/    2BB8 : C2 E2                           ldc R14, @RR2
5900/    2BBA : A0 E2                           incw RR2
5901/    2BBC : C2 F2                           ldc R15, @RR2
5902/    2BBE : 30 EE                           jp @RR14
5903/    2BC0 :                         ;
5904/    2BC0 : 2B D0                   SlfTst_Table    DW Free_Ram
5905/    2BC2 : 2B DA                           DW Free_Eprom
5906/    2BC4 : 2B EB                           DW Free_MtrSpd
5907/    2BC6 : 2B E6                           DW Free_SctrCnt
5908/    2BC8 : 2B F5                           DW Free_RW
5909/    2BCA :                         ;
5910/    2BCA : D6 02 8E                Chk_Pk_Jp       call L028e
5911/    2BCD : 8D 01 F8                        jp L01f8
5912/    2BD0 :                         ;
5913/    2BD0 : D6 22 46                Free_Ram        call Chk_SprChk         ; verify spare table checksum
5914/    2BD3 : EB F5                           jr NZ, Chk_Pk_Jp
5915/    2BD5 :                         ;
5916/    2BD5 : 46 33 80                        or SlfTst_Result, #Ram_Fail
5917/    2BD8 : 8B F0                           jr Chk_Pk_Jp
5918/    2BDA :                         ;
5919/    2BDA : 0C 02                   Free_Eprom      ld R0, #Eprom2
5920/    2BDC : D6 00 E9                        call EpromTest
5921/    2BDF : 6B E9                           jr Z, Chk_Pk_Jp
5922/    2BE1 :                         ;
5923/    2BE1 : 46 33 40                        or SlfTst_Result, #Eprom_Fail
5924/    2BE4 : 8B E4                           jr Chk_Pk_Jp
5925/    2BE6 :                         ;
5926/    2BE6 : D6 2A 1C                Free_SctrCnt    call SctrCount
5927/    2BE9 : 8B DF                           jr Chk_Pk_Jp
5928/    2BEB :                         ;
5929/    2BEB : D6 29 C1                Free_MtrSpd     call MtrSpd
5930/    2BEE : 6B DA                           jr Z, Chk_Pk_Jp
5931/    2BF0 :                         ;
5932/    2BF0 : 46 33 20                        or SlfTst_Result, #Disk_Speed
5933/    2BF3 : 8B D5                           jr Chk_Pk_Jp
5934/    2BF5 :                         ;
5935/    2BF5 : 46 33 02                Free_RW         or SlfTst_Result, #Rw_Fail      ; assume failure
5936/    2BF8 : D6 2A 6F                        call RwTest1
5937/    2BFB : 6B CD                           jr Z, Chk_Pk_Jp
5938/    2BFD :                         ;
5939/    2BFD : 56 33 FD                        and SlfTst_Result, #0FFh-Rw_Fail
5940/    2C00 : 8D 01 F0                        jp L01f0
5941/    2C03 :
5942/    2C03 :
5943/    2C03 :
5944/    2C03 :                         ;*****************************************************************
5945/    2C03 :                         ; Module ECC.B1.Assem   {Error Check and Correction}
5946/    2C03 :                         ;
5947/    2C03 :                         ; This module contains all the relevant files pertaining
5948/    2C03 :                         ; to the ECC method and algorithm used on Widget/Nisha
5949/    2C03 :                         ;
5950/    2C03 :                         ;       FUNCTION Ecc : BOOLEAN
5951/    2C03 :                         ;       PROCEDURE ShiftAndXor( VAR R1, R2, R3, R4, R5, R6 : BYTE {R1:6} )
5952/    2C03 :                         ;       PROCEDURE TestMod8( J : WORD {RR8} )
5953/    2C03 :                         ;       PROCEDURE Test0( J : WORD {RR8} )
5954/    2C03 :                         ;
5955/    2C03 :                         ;*****************************************************************
5956/    2C03 :
5957/    2C03 :                         ;*****************************************************************
5958/    2C03 :                         ;
5959/    2C03 :                         ; Function: Ecc
5960/    2C03 :                         ;
5961/    2C03 :                         ;  This function is responsible for 1) checking if the data in the
5962/    2C03 :                         ;  ReadBuffer is correctable and 2) correcting that data if it is
5963/    2C03 :                         ;  correctable.
5964/    2C03 :                         ;
5965/    2C03 :                         ;  The method used was prepared by:
5966/    2C03 :                         ;    Neil Glover
5967/    2C03 :                         ;    Daza Systems Technology Corp.
5968/    2C03 :                         ;    1801 Aspen St.
5969/    2C03 :                         ;    Broomfield, Co. 80020
5970/    2C03 :                         ;    (303) 466-5228
5971/    2C03 :                         ;
5972/    2C03 :                         ; Inputs:  none
5973/    2C03 :                         ;
5974/    2C03 :                         ; Outputs: Ecc: BOOLEAN {zero flag is set if not correctable}
5975/    2C03 :                         ;
5976/    2C03 :                         ; K1 = BitLength(DataField)+BitLength(CrcField)+BitLength(EccField)-41
5977/    2C03 :                         ;    = (1+532)*8 + 2*8 + 6*8 -41
5978/    2C03 :                         ;    = 4287
5979/    2C03 :                         ;
5980/    2C03 :                         ; Correction Span = 12 bits
5981/    2C03 :                         ;
5982/    2C03 :                         ;  R2Mask = $00
5983/    2C03 :                         ;  R3Mask = $0F
5984/    2C03 :                         ;
5985/    2C03 :                         ; Syndrome Bytes begin at ReadArray.RBuf1Ecc
5986/    2C03 :                         ;
5987/    2C03 :                         ; Local Variables: R1:          BYTE {R1}
5988/    2C03 :                         ;                  R2:          BYTE {R2}
5989/    2C03 :                         ;                  R3:          BYTE {R3}
5990/    2C03 :                         ;                  R4:          BYTE {R4}
5991/    2C03 :                         ;                  R5:          BYTE {R5}
5992/    2C03 :                         ;                  R6:          BYTE {R6}
5993/    2C03 :                         ;                  Correctable: BOOLEAN {R7/bit 7}
5994/    2C03 :                         ;                  Aligned:     BOOLEAN {R7/bit 6}
5995/    2C03 :                         ;                  Done:        BOOLEAN {R7/bit 5}
5996/    2C03 :                         ;                  J:           WORD {RR8}
5997/    2C03 :                         ;
5998/    2C03 :                         ; Algorithm:
5999/    2C03 :                         ;
6000/    2C03 :                         ;  Begin
6001/    2C03 :                         ;    R1:=SynfromeByte[1] {most significant byte}
6002/    2C03 :                         ;    R2:=SynfromeByte[2]
```

```
6003/    2C03 :                     ;    R3:=SynfromeByte[3]
6004/    2C03 :                     ;    R4:=SynfromeByte[4]
6005/    2C03 :                     ;    R5:=SynfromeByte[5]
6006/    2C03 :                     ;    R6:=SynfromeByte[6]
6007/    2C03 :                     ;    If (R1=R2=R3=R4=R5=R6=0)
6008/    2C03 :                     ;      Then Ecc:=false
6009/    2C03 :                     ;      Else
6010/    2C03 :                     ;        J:=K1
6011/    2C03 :                     ;        Aligned:=false
6012/    2C03 :                     ;        Done:=false
6013/    2C03 :                     ;        Correctable:=false
6014/    2C03 :                     ;        While R1=0 Do
6015/    2C03 :                     ;          ShiftRegsLeft 1 Byte {left-hand justify syndrome}
6016/    2C03 :                     ;          J:=J+8
6017/    2C03 :                     ;        While not(Done) Or not(Aligned) Do
6018/    2C03 :                     ;          ShiftAndXor
6019/    2C03 :                     ;          If R1=0
6020/    2C03 :                     ;            Then
6021/    2C03 :                     ;              If R4=R5=R6=0 And R2*R2Mask=0 And R3*R3Mask
6022/    2C03 :                     ;                Then
6023/    2C03 :                     ;                  Aligned:=true
6024/    2C03 :                     ;                  TestMod8
6025/    2C03 :                     ;            If not(Aligned) Then Test0
6026/    2C03 :                     ;            If not(Done) Then J:=J-1
6027/    2C03 :                     ;        While not(Done) Do
6028/    2C03 :                     ;          ShiftAndXor
6029/    2C03 :                     ;          If R=0
6030/    2C03 :                     ;            Then TestMod8
6031/    2C03 :                     ;            Else Test0
6032/    2C03 :                     ;          If not(Done) Then J:=J-1
6033/    2C03 :                     ;        If Correctable
6034/    2C03 :                     ;          Then
6035/    2C03 :                     ;            J:=J div 8
6036/    2C03 :                     ;            Buffer1[J]:=Buffer1[J] Xor R2
6037/    2C03 :                     ;            Buffer1[J+1]:=Buffer1[J+1] Xor R3
6038/    2C03 :                     ;            Buffer1[J+2]:=Buffer1[J+2] Xor R4
6039/    2C03 :                     ;            BlockMove(Buffer2, RBuffer
6040/    2C03 :                     ;        Ecc:=Correctable
6041/    2C03 :                     ;  End
6042/    2C03 :                     ;
6043/    2C03 :                     ;*******************************************************************
6044/    2C03 :
6045/    2C03 : =10AFH              K1              EQU 4271
6046/    2C03 : =0H                 R2Mask          EQU 0
6047/    2C03 : =FH                 R3Mask          EQU 00Fh
6048/    2C03 : =80H                Ecc_Correctable EQU 080h
6049/    2C03 : =40H                Ecc_Aligned     EQU 040h
6050/    2C03 : =20H                Ecc_Done        EQU 020h
6051/    2C03 :
6052/    2C03 : B0 E7               ECC             clr R7                  ; clear booleans
6053/    2C05 : 8C 10                               ld R8, #K1 /256
6054/    2C07 : 9C AF                               ld R9, #K1 #256
6055/    2C09 : 0C 00                               ld R0, #0               ; get ready to check for all zero syndrome
6056/    2C0B : BC 06                               ld R11, #6              ; load six bytes, R1..R6:=Syndrome Bytes
6057/    2C0D : CC 12                               ld R12, #RBuf1Ecc /256
6058/    2C0F : DC 2D                               ld R13, #RBuf1Ecc #256
6059/    2C11 : AC 11                               ld R10, #Wrk_Sys+1      ; load syndrome bytes into registers
6060/    2C13 :                     ;
6061/    2C13 : 83 AC               Ecc_Ld_Lp       ldei @R10, @RR12
6062/    2C15 : 43 0A                               or R0, @R10
6063/    2C17 : BA FA                               djnz R11, Ecc_Ld_Lp
6064/    2C19 : 6D 2C A8                            jp Z, Ecc_End
6065/    2C1C :                     ;
6066/    2C1C : 42 11               Ecc_LHJ_While   or R1, R1               ; While R1=0 Do
6067/    2C1E : EB 14                               jr NZ, Ecc_Align
6068/    2C20 : 18 E2                               ld R1, R2               ; shift left 1 byte
6069/    2C22 : 28 E3                               ld R2, R3
6070/    2C24 : 38 E4                               ld R3, R4
6071/    2C26 : 48 E5                               ld R4, R5
6072/    2C28 : 58 E6                               ld R5, R6
6073/    2C2A : B0 E6                               clr R6
6074/    2C2C : 06 E9 08                            add R9, #8              ; J:=J+8
6075/    2C2F : 16 E8 00                            adc R8, #0
6076/    2C32 : 8B E8                               jr Ecc_LHJ_While
6077/    2C34 :                     ;
6078/    2C34 : D6 2C AB            Ecc_Align       call ShiftAndXor
6079/    2C37 : EB 15                               jr NZ, Ecc_Al_1
6080/    2C39 : 08 E4                               ld R0, R4               ; If (R4=R5=R6=0)
6081/    2C3B : 42 05                               or R0, R5
6082/    2C3D : 42 06                               or R0, R6
6083/    2C3F : F8 E3                               ld R15, R3              ; And (R3*R3Mask=0)
6084/    2C41 : 56 EF 0F                            and R15, #R3Mask
6085/    2C44 : 42 0F                               or R0, R15
6086/    2C46 : EB 06                               jr NZ, Ecc_Al_1
6087/    2C48 :                     ;
6088/    2C48 : 46 E7 40                            or R7, #Ecc_Aligned
6089/    2C4B : D6 2C CC                            call TestMod8
6090/    2C4E :                     ;
6091/    2C4E : 76 E7 40            Ecc_Al_1        tm R7, #Ecc_Aligned
6092/    2C51 : EB 03                               jr NZ, Ecc_Al_2
6093/    2C53 :                     ;
6094/    2C53 : D6 2C D7                            call Test0
6095/    2C56 :                     ;
6096/    2C56 : 76 E7 20            Ecc_Al_2        tm R7, #20h
6097/    2C59 : EB 1D                               jr NZ, Ecc_Crct
6098/    2C5B :                     ;
6099/    2C5B : 80 E8                               decw RR8                ; J:=J-1
6100/    2C5D : 76 E7 60                            tm R7, #Ecc_Done+Ecc_Aligned
6101/    2C60 : 6B D2                               jr Z, Ecc_Align
6102/    2C62 :                     ;
6103/    2C62 : D6 2C AB            Ecc_Shift       call ShiftAndXor
6104/    2C65 : EB 05                               jr NZ, Ecc_Shft_Else
6105/    2C67 : D6 2C CC                            call TestMod8
6106/    2C6A : 8B 03                               jr Ecc_Shft_2
6107/    2C6C :                     ;
6108/    2C6C : D6 2C D7            Ecc_Shft_Else   call Test0
6109/    2C6F : 76 E7 20            Ecc_Shft_2      tm R7, #Ecc_Done
6110/    2C72 : EB 04                               jr NZ, Ecc_Crct
6111/    2C74 : 80 E8                               decw RR8                ; J:=J-1
6112/    2C76 : 8B EA                               jr Ecc_Shift
6113/    2C78 :                     ;
6114/    2C78 : 76 E7 80            Ecc_Crct        tm R7, #Ecc_Correctable
6115/    2C7B : 6B 2B                               jr Z, Ecc_End
6116/    2C7D :                     ;
6117/    2C7D : AC 03                               ld R10, #3              ; J:=J div 8
6118/    2C7F : C0 E8               Ecc_Div8        rrc R8
6119/    2C81 : C0 E9                               rrc R9
6120/    2C83 : AA FA                               djnz R10, Ecc_Div8
```

```
6121/    2C85 : 56 E8 1F                        and R8, #1Fh            ; mask off unwanted carries
6122/    2C88 : CC 10                           ld R12, #RDummy /256
6123/    2C8A : DC 18                           ld R13, #RDummy #256
6124/    2C8C : 02 D9                           add R13, R9
6125/    2C8E : 12 C8                           adc R12, R8
6126/    2C90 : BC 12                           ld R11, #Wrk_Sys+2      ; start with R2
6127/    2C92 : AC 03                           ld R10, #3              ; correct 3 bytes
6128/    2C94 :                         ;
6129/    2C94 : 82 1C                   Ecc_Crct_Lp     lde R1, @RR12
6130/    2C96 : B3 1B                           xor R1, @R11
6131/    2C98 : F3 B1                           ld @R11, R1
6132/    2C9A : 93 BC                           ldei @RR12, @R11
6133/    2C9C : AA F6                           djnz R10, Ecc_Crct_Lp
6134/    2C9E :                         ;
6135/    2C9E : 2C 20                           ld R2, #RBuf_To_Buf2 /256
6136/    2CA0 : 3C 8A                           ld R3, #RBuf_To_Buf2 #256
6137/    2CA2 : D6 02 CB                        call Bank_Call
6138/    2CA5 : 76 E7 80                        tm R7, #Ecc_Correctable ; set correctable flag
6139/    2CA8 :                         ;
6140/    2CA8 : 8D 02 F6                Ecc_End         jp Bank_Ret
6141/    2CAB :
6142/    2CAB :
6143/    2CAB :                         ;*********************************************************************
6144/    2CAB :                         ;
6145/    2CAB :                         ; Procedure: ShiftAndXor
6146/    2CAB :                         ;
6147/    2CAB :                         ;  This procedure is used to shift the current syndrome bytes
6148/    2CAB :                         ;  (assumed to be located in R1:6) to the right 1 bit and then
6149/    2CAB :                         ;  Xor the syndromes with the reciprocal polynominal if needed.
6150/    2CAB :                         ;
6151/    2CAB :                         ; Inputs: R1: BYTE {R1}
6152/    2CAB :                         ;         R2: BYTE {R2}
6153/    2CAB :                         ;         R3: BYTE {R3}
6154/    2CAB :                         ;         R4: BYTE {R4}
6155/    2CAB :                         ;         R5: BYTE {R5}
6156/    2CAB :                         ;         R6: BYTE {R6}
6157/    2CAB :                         ;
6158/    2CAB :                         ; Outputs: R1: BYTE {R1}
6159/    2CAB :                         ;          R2: BYTE {R2}
6160/    2CAB :                         ;          R3: BYTE {R3}
6161/    2CAB :                         ;          R4: BYTE {R4}
6162/    2CAB :                         ;          R5: BYTE {R5}
6163/    2CAB :                         ;          R6: BYTE {R6}
6164/    2CAB :                         ;
6165/    2CAB :                         ; Algorithm:
6166/    2CAB :                         ;
6167/    2CAB :                         ;  Begin
6168/    2CAB :                         ;    Shift the syndromes right 1 bit with carry
6169/    2CAB :                         ;    If the LSB of R6 was a 1 {if carry}
6170/    2CAB :                         ;      Then
6171/    2CAB :                         ;        R1:=R1 Xor 140
6172/    2CAB :                         ;        R2:=R3 Xor 12
6173/    2CAB :                         ;        R3:=R3 Xor 10
6174/    2CAB :                         ;        R4:=R4 Xor 40
6175/    2CAB :                         ;        R5:=R5 Xor 24
6176/    2CAB :                         ;        R6:=R6 Xor 8
6177/    2CAB :                         ;  End
6178/    2CAB :                         ;
6179/    2CAB :                         ;*********************************************************************
6180/    2CAB :
6181/    2CAB : CC 06                   ShiftAndXor     ld R12, #6              ; shift 6 bytes
6182/    2CAD : DC 11                           ld R13, #Wrk_Sys+1      ; start with R1
6183/    2CAF : CF                              rcf
6184/    2CB0 :                         ;
6185/    2CB0 : C1 ED                   S_A_Xor_Lp      rrc @R13
6186/    2CB2 : DE                              inc R13
6187/    2CB3 : CA FB                           djnz R12, S_A_Xor_Lp
6188/    2CB5 :                         ;
6189/    2CB5 : FB 12                           jr NC, S_A_Xor_End
6190/    2CB7 : B6 E1 8C                        xor R1, #140
6191/    2CBA : B6 E2 0C                        xor R2, #12
6192/    2CBD : B6 E3 0A                        xor R3, #10
6193/    2CC0 : B6 E4 28                        xor R4, #40
6194/    2CC3 : B6 E5 18                        xor R5, #24
6195/    2CC6 : B6 E6 08                        xor R6, #8
6196/    2CC9 : 42 11                   S_A_Xor_End     or R1, R1               ; If R1=0 ...
6197/    2CCB : AF                              ret
6198/    2CCC :
6199/    2CCC :
6200/    2CCC :                         ;*********************************************************************
6201/    2CCC :                         ;
6202/    2CCC :                         ; Procedure: TestMod8
6203/    2CCC :                         ;
6204/    2CCC :                         ;  This procedure is used to test if J mod 8 = 0.
6205/    2CCC :                         ;
6206/    2CCC :                         ; Inputs: J: WORD {RR8}
6207/    2CCC :                         ;
6208/    2CCC :                         ; Outputs: none
6209/    2CCC :                         ;
6210/    2CCC :                         ; Global Variables Changed: Done:        BOOLEAN {R7/bit 5}
6211/    2CCC :                         ;                           Correctable: BOOLEAN {R7/bit 7}
6212/    2CCC :                         ;
6213/    2CCC :                         ; Algorithm:
6214/    2CCC :                         ;
6215/    2CCC :                         ;  Begin
6216/    2CCC :                         ;    If J mod 8 = 0
6217/    2CCC :                         ;      Then
6218/    2CCC :                         ;        Done:=true
6219/    2CCC :                         ;        Correctable:=true
6220/    2CCC :                         ;  End
6221/    2CCC :                         ;
6222/    2CCC :                         ;*********************************************************************
6223/    2CCC :
6224/    2CCC : 08 E9                   TestMod8        ld R0, R9
6225/    2CCE : 56 E0 07                        and R0, #7              ; get remainder from division
6226/    2CD1 : EB 03                           jr NZ,TstMd8_Done
6227/    2CD3 : 46 E7 A0                        or R7, #Ecc_Done+Ecc_Correctable
6228/    2CD6 : AF                      TstMd8_Done     ret
6229/    2CD7 :
6230/    2CD7 :
6231/    2CD7 :                         ;*********************************************************************
6232/    2CD7 :                         ;
6233/    2CD7 :                         ; Procedure: Test0
6234/    2CD7 :                         ;
6235/    2CD7 :                         ;  This procedure is used to test if J=0.
6236/    2CD7 :                         ;
6237/    2CD7 :                         ; Inputs: J: WORD {RR8}
6238/    2CD7 :                         ;
```

```
6239/    2CD7 :                     ; Outputs: none
6240/    2CD7 :                     ;
6241/    2CD7 :                     ; Global Variables Changed: Done:        BOOLEAN {R7/bit 5}
6242/    2CD7 :                     ;                           Correctable: BOOLEAN {R7/bit 7}
6243/    2CD7 :                     ;
6244/    2CD7 :                     ; Algorithm:
6245/    2CD7 :                     ;
6246/    2CD7 :                     ;  Begin
6247/    2CD7 :                     ;    If J=0
6248/    2CD7 :                     ;      Then
6249/    2CD7 :                     ;        Done:=true
6250/    2CD7 :                     ;        Correctable:=true
6251/    2CD7 :                     ;  End
6252/    2CD7 :                     ;
6253/    2CD7 :                     ;*******************************************************************
6254/    2CD7 :
6255/    2CD7 : 08 E8               Test0           ld R0, R8
6256/    2CD9 : 42 09                               or R0, R9               ; If J=0 ...
6257/    2CDB : EB 06                               jr NZ,Test0_Done
6258/    2CDD : 46 E7 20                            or R7, #Ecc_Done
6259/    2CE0 : 56 E7 7F                            and R7, #0FFh-Ecc_Correctable
6260/    2CE3 : AF                  Test0_Done      ret
6261/    2CE4 :
6262/    2CE4 :
6263/    2CE4 : 00 01 00 01         L2CE4           DB 000h, 001h, 000h, 001h
6264/    2CE8 : E6 00 98 35                         DB 0E6h, 000h, 098h, 035h
6265/    2CEC :
6266/    2CEC :                     ; Cmd 3
6267/    2CEC :                     ; build something together inside the RW buffer
6268/    2CEC : 0C 00               L2CEC           ld R0, #0
6269/    2CEE : D6 05 A3                            call L05a3
6270/    2CF1 : 2C 10                               ld R2, #010h
6271/    2CF3 : 3C 20                               ld R3, #020h
6272/    2CF5 : 82 02                               lde R0, @RR2
6273/    2CF7 : 09 5D                               ld 5Dh, R0      ; preserve value from 1020h
6274/    2CF9 : CC 10                               ld R12, #010h
6275/    2CFB : DC 19                               ld R13, #019h
6276/    2CFD : EC 2C                               ld R14, #L2CE4 /256
6277/    2CFF : FC E4                               ld R15, #L2CE4 #256
6278/    2D01 : 1C 08                               ld R1, #8
6279/    2D03 : C2 0E               L2d03           Ldc R0, @RR14   ; copy 8 bytes from table
6280/    2D05 : 92 0C                               lde @RR12, R0   ;  into 1019h ff (RBuffer1 ???)
6281/    2D07 : A0 EC                               incw RR12
6282/    2D09 : A0 EE                               incw RR14
6283/    2D0B : 1A F6                               djnz R1, L2d03
6284/    2D0D : EC 14                               ld R14, #SprCount /256
6285/    2D0F : FC DF                               ld R15, #SprCount #256
6286/    2D11 : 2C 00                               ld R2, #0       ; R2 = 0
6287/    2D13 : 82 3E                               lde R3, @RR14
6288/    2D15 : 5C 4C                               ld R5, #76
6289/    2D17 : 22 53                               sub R5, R3
6290/    2D19 : 38 E5                               ld R3, R5       ; R3 = remaining spares
6291/    2D1B : A0 EE                               incw RR14       ; bad block count
6292/    2D1D : 4C 00                               ld R4, #0       ; R4 = 0
6293/    2D1F : 82 5E                               lde R5, @RR14   ; R5 = bad block count
6294/    2D21 : 68 33                               ld R6,SlfTst_Result     ; R6 = selftest result
6295/    2D23 : 78 72                               ld R7,72h       ; R7 = 72h
6296/    2D25 : 0C 12                               ld R0, #012h    ; start with register R2
6297/    2D27 : 1C 06                               ld R1, #6
6298/    2D29 : 93 0C               L2d29           ldei @RR12, @R0 ; copy six registers
6299/    2D2B : 1A FC                               djnz R1, L2d29  ;  into buffer
6300/    2D2D : 0C 11                               ld R0, #011h
6301/    2D2F : 1C 29                               ld R1, #029h
6302/    2D31 : 2C FF                               ld R2, #0FFh
6303/    2D33 : 3C 24                               ld R3, #36
6304/    2D35 : 92 20               L2d35           lde @RR0, R2    ; copy 36x FFh
6305/    2D37 : A0 E0                               incw RR0        ; into 1129h ff
6306/    2D39 : 3A FA                               djnz R3, L2d35
6307/    2D3B : 8C 80                               ld R8, #080h
6308/    2D3D : B0 E9                               clr R9
6309/    2D3F : B0 EA                               clr R10
6310/    2D41 : BC 01                               ld R11, #1
6311/    2D43 : 0C 10                               ld R0, #010h
6312/    2D45 : 1C A9                               ld R1, #0A9h
6313/    2D47 : 2C 09                               ld R2, #9
6314/    2D49 : D6 2D 7D            L2d49           call L2d7d      ; copy 80 00 00 01 into 10A9h ff
6315/    2D4C : 2A FB                               djnz R2, L2d49
6316/    2D4E : 60 E8                               com R8
6317/    2D50 : 60 E9                               com R9
6318/    2D52 : 60 EA                               com R10
6319/    2D54 : 60 EB                               com R11
6320/    2D56 : 0C 10                               ld R0, #010h
6321/    2D58 : 1C A5                               ld R1, #0A5h
6322/    2D5A : D6 2D 7D                            call L2d7d
6323/    2D5D : 0C 10                               ld R0, #010h
6324/    2D5F : 1C CD                               ld R1, #0CDh
6325/    2D61 : D6 2D 7D                            call L2d7d
6326/    2D64 : 0C 11                               ld R0, #011h
6327/    2D66 : 1C 25                               ld R1, #025h
6328/    2D68 : D6 2D 7D                            call L2d7d
6329/    2D6B : 0C 11                               ld R0, #011h
6330/    2D6D : 1C 4D                               ld R1, #04Dh
6331/    2D6F : D6 2D 7D                            call L2d7d
6332/    2D72 : 0C 10                               ld R0, #010h
6333/    2D74 : 1C C1                               ld R1, #0C1h
6334/    2D76 : 2C 88                               ld R2, #088h
6335/    2D78 : 92 20                               lde @RR0, R2
6336/    2D7A : 8D 02 F6                            jp Bank_Ret
6337/    2D7D :
6338/    2D7D : 4C 04               L2d7d           ld R4, #004h
6339/    2D7F : 3C 18                               ld R3, #018h
6340/    2D81 : 93 30               L2d81           ldei @RR0, @R3
6341/    2D83 : 4A FC                               djnz R4, L2d81
6342/    2D85 : AF                                  ret
6343/    2D86 :
6344/    2D86 : D6 05 86            FormatTrack     call L0586
6345/    2D89 : 0C C6                               ld R0, #0C6h
6346/    2D8B : D6 05 9D                            call L059d
6347/    2D8E : 76 3E 01                            tm 3Eh, #HDA_Type       ; Nisha mechnism?
6348/    2D91 : 6B 14                               jr Z, L2da7             ;  yes --> skip selfsync
6349/    2D93 :                     ; write Rodime selfsync pattern
6350/    2D93 : 2C 10                               ld R2, #FormatArray /256
6351/    2D95 : 3C 20                               ld R3, #FormatArray #256
6352/    2D97 : EC 2E                               ld R14, #L2e12 /256
6353/    2D99 : FC 12                               ld R15, #L2e12 #256
6354/    2D9B : 1C 32                               ld R1, #50
6355/    2D9D : C2 0E               L2d9d           Ldc R0, @RR14
6356/    2D9F : 92 02                               lde @RR2, R0
```

```
6357/    2DA1 : A0 E2                              incw RR2
6358/    2DA3 : A0 EE                              incw RR14
6359/    2DA5 : 1A F6                              djnz R1, L2d9d
6360/    2DA7 :                        ;
6361/    2DA7 : 2C 10                  L2da7       ld R2, #FHdrSync /256
6362/    2DA9 : 3C 3A                              ld R3, #FHdrSync #256
6363/    2DAB : 4C 10                              ld R4, #FDataSync /256
6364/    2DAD : 5C 50                              ld R5, #FDataSync #256
6365/    2DAF : 0C 01                              ld R0, #1
6366/    2DB1 : 92 02                              lde @RR2, R0
6367/    2DB3 : 92 04                              lde @RR4, R0
6368/    2DB5 : A0 E2                              incw RR2
6369/    2DB7 : A0 E4                              incw RR4
6370/    2DB9 : 0C 00                              ld R0, #0
6371/    2DBB : 92 02                              lde @RR2, R0
6372/    2DBD : 92 04                              lde @RR4, R0
6373/    2DBF : 46 35 20                           or DiskStat, #Wr_Op
6374/    2DC2 : E6 3D 00                           ld Sector, #0
6375/    2DC5 : 76 3E 01                           tm 3Eh, #HDA_Type       ; Nisha mechnism?
6376/    2DC8 : 6B 13                              jr Z, L2ddd             ;  yes -->
6377/    2DCA :                        ;
6378/    2DCA : E6 3D 1F                           ld Sector, #31
6379/    2DCD : D6 06 5B                           call AdjustGeometry
6380/    2DD0 : 56 FA FB                           and IRQ, #0FFh-Irq_Sector
6381/    2DD3 : 2C 01                              ld R2, #001h
6382/    2DD5 : D6 06 9D                           call L069d
6383/    2DD8 : E6 3D 00                           ld Sector, #0
6384/    2DDB : 8B 03                              jr L2de0
6385/    2DDD :                        ;
6386/    2DDD : D6 06 5B               L2ddd       call AdjustGeometry
6387/    2DE0 : 4C 20                  L2de0       ld R4, #NbrSctrs
6388/    2DE2 : 2C 10                  L2de2       ld R2, #FHeader /256
6389/    2DE4 : 3C 3C                              ld R3, #FHeader #256
6390/    2DE6 : D6 03 F0                           call Load_Header
6391/    2DE9 : D6 04 8F                           call L048f
6392/    2DEC : D6 02 05                           call L0205
6393/    2DEF : 56 E5 77                           and R5, #077h
6394/    2DF2 : A6 E5 62                           cp R5, #062h
6395/    2DF5 : ED 06 51                           jp NZ, L0651
6396/    2DF8 : 06 3D 02                           add Sector, #2          ; 2:1 interleave
6397/    2DFB : A6 3D 20                           cp Sector, #NbrSctrs
6398/    2DFE : EB 0D                              jr NZ, L2e0d
6399/    2E00 : E6 3D 01                           ld Sector, #1
6400/    2E03 : 76 03 04               FmtT_1      tm P3, #IndexMark       ; while not index
6401/    2E06 : 6B FB                              jr Z, FmtT_1
6402/    2E08 : 76 03 02               L2e08       tm P3, #SectorMark
6403/    2E0B : 6B FB                              jr Z, L2e08
6404/    2E0D : 4A D3                  L2e0d       djnz R4, L2de2
6405/    2E0F : 8D 02 F6                           jp Bank_Ret
6406/    2E12 :
6407/    2E12 : 00 00 00 00            L2e12       DB 000h, 000h, 000h, 000h
6408/    2E16 : 00 00 00 00                        DB 000h, 000h, 000h, 000h
6409/    2E1A : 00 00 00 00                        DB 000h, 000h, 000h, 000h
6410/    2E1E : 00 00 00 00                        DB 000h, 000h, 000h, 000h
6411/    2E22 : 00 FF FF FF                        DB 000h, 0FFh, 0FFh, 0FFh
6412/    2E26 : FF FF FF FF                        DB 0FFh, 0FFh, 0FFh, 0FFh
6413/    2E2A : FF FF 01 00                        DB 0FFh, 0FFh, 001h, 000h
6414/    2E2E : 00 00 00 00                        DB 000h, 000h, 000h, 000h
6415/    2E32 : 00 00 00 49                        DB 000h, 000h, 000h, 049h
6416/    2E36 : 24 92 49 24                        DB 024h, 092h, 049h, 024h
6417/    2E3A : 92 49 24 92                        DB 092h, 049h, 024h, 092h
6418/    2E3E : 49 24 92 FF                        DB 049h, 024h, 092h, 0ffh
6419/    2E42 : 01 00                              DB 001h, 000h
6420/    2E44 :
6421/    2E44 :
6422/    2E44 :                        ; Cmd 19
6423/    2E44 : 46 50 01               L2e44       or 50h, #1
6424/    2E47 : 0C 00                              ld R0, #0
6425/    2E49 : D6 05 A3                           call L05a3
6426/    2E4C : 8B 03                              jr L2e51
6427/    2E4E :
6428/    2E4E :                        ; Cmd 20
6429/    2E4E : 56 50 FE               L2E4E       and 50h, #0FFh-001h
6430/    2E51 : 44 33 33               L2e51       or SlfTst_Result,SlfTst_Result
6431/    2E54 : 6B 05                              jr Z, L2e5b
6432/    2E56 : FC 39                              ld R15, #57
6433/    2E58 : 8D 03 57                           jp Abort
6434/    2E5B :                        ;
6435/    2E5B : 2C 10                  L2e5b       ld R2, #010h
6436/    2E5D : 3C 20                              ld R3, #020h
6437/    2E5F : 82 02                              lde R0, @RR2
6438/    2E61 : A4 5D E0                           cp R0,5Dh
6439/    2E64 : 6B 05                              jr Z, L2e6b
6440/    2E66 : FC 3D                              ld R15, #61
6441/    2E68 : 8D 03 57                           jp Abort
6442/    2E6B :                        ;
6443/    2E6B : CC 00                  L2e6b       ld R12, #0
6444/    2E6D : DC 00                              ld R13, #0
6445/    2E6F : EC 00                              ld R14, #0
6446/    2E71 : FC 00                              ld R15, #0
6447/    2E73 : 8B 0A                              jr L2e7f
6448/    2E75 :                        ;
6449/    2E75 : A4 38 EC               L2e75       cp R12,Cur_Cyl
6450/    2E78 : EB 05                              jr NZ, L2e7f
6451/    2E7A : A4 39 ED                           cp R13,Cur_Cyl+1
6452/    2E7D : 6B 12                              jr Z, L2e91
6453/    2E7F : 46 35 08               L2e7f       or DiskStat, #Offset_On
6454/    2E82 : 56 35 FE                           and DiskStat, #0FFh-Offset_Set
6455/    2E85 : D6 25 47                           call Seek
6456/    2E88 : D6 2F 99                           call L2f99
6457/    2E8B : EB 0E                              jr NZ, L2e9b
6458/    2E8D : FC 02                              ld R15, #2
6459/    2E8F : 8B 0A                              jr L2e9b
6460/    2E91 :
6461/    2E91 : A4 3C EE               L2e91       cp R14,Head
6462/    2E94 : 6B 05                              jr Z, L2e9b
6463/    2E96 : 08 EE                              ld R0, R14
6464/    2E98 : D6 26 6C                           call SelectHead
6465/    2E9B : 46 35 04               L2e9b       or DiskStat, #4
6466/    2E9E : F9 3D                              ld Sector, R15
6467/    2EA0 : 56 35 DF                           and DiskStat, #0DFh
6468/    2EA3 : 76 50 01                           tm 50h, #001h
6469/    2EA6 : 6B 03                              jr Z, L2eab
6470/    2EA8 : 46 35 20                           or DiskStat, #020h
6471/    2EAB : 56 3E 9D               L2eab       and 3Eh, #0FFh-S_Block-B_Block-002h
6472/    2EAE : 2C 11                              ld R2, #RW_Common /256
6473/    2EB0 : 3C D3                              ld R3, #RW_Common #256
6474/    2EB2 : D6 02 CB                           call Bank_Call
```

```
6475/    2EB5 : ED 2F 53                        jp NZ, L2f53
6476/    2EB8 : A6 E0 86                        cp R0, #086h
6477/    2EBB : 6D 2F 53                        jp Z, L2f53
6478/    2EBE : D6 05 22                        call Ext_Push
6479/    2EC1 : 0C 06                           ld R0, #6
6480/    2EC3 : 76 3E 01                        tm 3Eh, #HDA_Type       ; Nisha mechanism?
6481/    2EC6 : 6B 02                           jr Z, L2eca             ;  yes -->
6482/    2EC8 : 0C 07                           ld R0, #7
6483/    2ECA : CF                  L2eca       rcf
6484/    2ECB : 10 ED                           rlc R13
6485/    2ECD : 10 EC                           rlc R12
6486/    2ECF : 0A F9                           djnz R0, L2eca
6487/    2ED1 : 0C 05                           ld R0, #005h
6488/    2ED3 : CF                  L2ed3       rcf
6489/    2ED4 : 10 EE                           rlc R14
6490/    2ED6 : 0A FB                           djnz R0, L2ed3
6491/    2ED8 : 2C 16                           ld R2, #016h
6492/    2EDA : 3C 1B                           ld R3, #01Bh
6493/    2EDC : 1C 20                           ld R1, #020h
6494/    2EDE : 82 02               L2ede       lde R0, @RR2
6495/    2EE0 : A2 0F                           cp R0, R15
6496/    2EE2 : 6B 09                           jr Z, L2eed
6497/    2EE4 : A0 E2                           incw RR2
6498/    2EE6 : 1A F6                           djnz R1, L2ede
6499/    2EE8 : FC 3A                           ld R15, #58
6500/    2EEA : 8D 03 57                        jp Abort
6501/    2EED :
6502/    2EED : FC 20               L2eed       ld R15, #020h
6503/    2EEF : 22 F1                           sub R15, R1
6504/    2EF1 : 02 EF                           add R14, R15
6505/    2EF3 : 16 ED 00                        adc R13, #0
6506/    2EF6 : 16 EC 00                        adc R12, #0
6507/    2EF9 : 02 DE                           add R13, R14
6508/    2EFB : 16 EC 00                        adc R12, #0
6509/    2EFE : E8 ED                           ld R14, R13
6510/    2F00 : D8 EC                           ld R13, R12
6511/    2F02 : CC 00                           ld R12, #0
6512/    2F04 : 42 EE                           or R14, R14
6513/    2F06 : 6B 48                           jr Z, L2f50
6514/    2F08 : 76 ED 01                        tm R13, #1
6515/    2F0B : 6B 43                           jr Z, L2f50
6516/    2F0D : 3C 09                           ld R3, #9
6517/    2F0F : D6 21 5B                        call L215b
6518/    2F12 : CC 00                           ld R12, #0
6519/    2F14 : 22 E2                           sub R14, R2
6520/    2F16 : 32 D1                           sbc R13, R1
6521/    2F18 : 0C 00                           ld R0, #HiMaxLogical
6522/    2F1A : 1C 98                           ld R1, #MidMaxLogical
6523/    2F1C : 2C 34                           ld R2, #LoMaxLogical
6524/    2F1E : D6 04 59                        call L0459
6525/    2F21 : 1B 71                           jr LT, L2f94
6526/    2F23 : 2C 14                           ld R2, #LogicalBlock /256
6527/    2F25 : 3C 80                           ld R3, #LogicalBlock #256
6528/    2F27 : 1C 1C                           ld R1, #01Ch
6529/    2F29 : 93 12                           ldei @RR2, @R1
6530/    2F2B : 93 12                           ldei @RR2, @R1
6531/    2F2D : 93 12                           ldei @RR2, @R1
6532/    2F2F : 2C 1B                           ld R2, #CnvrtLogical /256
6533/    2F31 : 3C 04                           ld R3, #CnvrtLogical #256
6534/    2F33 : D6 02 CB                        call Bank_Call
6535/    2F36 : EB 5C                           jr NZ, L2f94
6536/    2F38 : 0C 82                           ld R0, #082h
6537/    2F3A : 76 50 01                        tm 50h, #1
6538/    2F3D : EB 02                           jr NZ, L2f41
6539/    2F3F : 0C 84                           ld R0, #132
6540/    2F41 : 46 35 04            L2f41       or DiskStat, #User_Type
6541/    2F44 : 2C 13                           ld R2, #Data_Ex_Handler /256
6542/    2F46 : 3C BD                           ld R3, #Data_Ex_Handler #256
6543/    2F48 : D6 02 CB                        call Bank_Call
6544/    2F4B : 0C 00                           ld R0, #0
6545/    2F4D : D6 05 A3                        call L05a3
6546/    2F50 : D6 05 4F            L2f50       call Ext_Pop
6547/    2F53 : 06 EF 02            L2f53       add R15, #2
6548/    2F56 : A6 EF 21                        cp R15, #021h
6549/    2F59 : 6B 0A                           jr Z, L2f65
6550/    2F5B : A6 EF 20                        cp R15, #020h
6551/    2F5E : EB 02                           jr NZ, L2f62
6552/    2F60 : FC 01                           ld R15, #1
6553/    2F62 : 8D 2E 75            L2f62       jp L2e75
6554/    2F65 :
6555/    2F65 : FC 00               L2f65       ld R15, #0
6556/    2F67 : EE                              inc R14
6557/    2F68 : 0C 02                           ld R0, #NbrHds_B        ; assume Nisha
6558/    2F6A : 76 3E 01                        tm 3Eh, #HDA_Type       ; do we have a Nisha HDA?
6559/    2F6D : 6B 02                           jr Z, L2f71             ;  yes -->
6560/    2F6F : 0C 04                           ld R0, #NbrHds_A        ;  else use Rodime value
6561/    2F71 : A2 0E               L2f71       cp R0, R14
6562/    2F73 : ED 2E 75                        jp NZ, L2e75
6563/    2F76 : EC 00                           ld R14, #0
6564/    2F78 : A0 EC                           incw RR12
6565/    2F7A : 0C 02                           ld R0, #NbrTracks_B /256
6566/    2F7C : 1C 62                           ld R1, #NbrTracks_B #256
6567/    2F7E : 76 3E 01                        tm 3Eh, #HDA_Type       ; do we have a Nisha HDA?
6568/    2F81 : 6B 04                           jr Z, L2f87             ;  yes -->
6569/    2F83 : 0C 01                           ld R0, #NbrTracks_A /256
6570/    2F85 : 1C 31                           ld R1, #NbrTracks_A #256
6571/    2F87 : A2 0C               L2f87       cp R0, R12
6572/    2F89 : ED 2E 75                        jp NZ, L2e75
6573/    2F8C : A2 1D                           cp R1, R13
6574/    2F8E : ED 2E 75                        jp NZ, L2e75
6575/    2F91 : 8D 02 F6            L2f91       jp Bank_Ret
6576/    2F94 :
6577/    2F94 : D6 05 4F            L2f94       call Ext_Pop
6578/    2F97 : 8B F8                           jr L2f91
6579/    2F99 :
6580/    2F99 : 4C 02               L2f99       ld R4, #2
6581/    2F9B : D6 05 22                        call Ext_Push
6582/    2F9E : 88 E4               L2f9e       ld R8, R4
6583/    2FA0 : D6 22 6A                        call Spr
6584/    2FA3 : 56 35 FB                        and DiskStat, #251
6585/    2FA6 : 2C 1B                           ld R2, #CnvrtLogical /256
6586/    2FA8 : 3C 04                           ld R3, #CnvrtLogical #256
6587/    2FAA : D6 02 CB                        call Bank_Call
6588/    2FAD : A4 38 EC                        cp R12,Cur_Cyl
6589/    2FB0 : EB 0A                           jr NZ, L2fbc
6590/    2FB2 : A4 39 ED                        cp R13,Cur_Cyl+1
6591/    2FB5 : EB 05                           jr NZ, L2fbc
6592/    2FB7 : A4 3C EE                        cp R14,Head
```

```
6593/    2FBA : 6B 05                                    jr Z, L2fc1
6594/    2FBC : 4A E0                  L2fbc             djnz R4, L2f9e
6595/    2FBE : 46 E0 01                                 or R0, #1
6596/    2FC1 : 70 FC                  L2fc1             push FLAGS
6597/    2FC3 : D6 05 4F                                 call Ext_Pop
6598/    2FC6 : 50 FC                                    pop FLAGS
6599/    2FC8 : 8D 02 F6                                 jp Bank_Ret
6600/    2FCB :
6601/    1FCB :                                  dephase
6602/    1FCB : =>FALSE                          if $>ROMsize
6603/    1FCB :                                          error "\aROM size exceeded !!!"
6604/    1FCB : =>TRUE                           elseif
6605/    1FCB : FF FF FF FF FF FF                        DB ROMsize-$ dup(0FFh)  ; pad with LFF's
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF
6606/    2000 : [6602]                           endif
6607/    2000 :                                  end
```

symbol table (* = unused):

```
 ABORT :                    357 - |  ABORT_10 :                 1139 C |
 ABORT_24 :                18F9 C | *ABORT_55 :                 18FE C |
 ABORT_56 :                1968 C |  ABORT_61 :                 1903 C |
 ABORT_61A :               196D C | *ABORT_8 :                  113E C |
 ABORT_STAT :              123F C |  ACCESS :                     80 - |
 ACCESS_OFFSET :             90 - |  ADDSPARE :                 241F C |
 ADJUSTGEOMETRY :            65B - |  ADS_ELSE1 :                2431 C |
 ADS_UPDATE :              2445 C |  ADS_UPDT1 :                245C C |
 ADS_UPDT2 :               245F C | *ARCHITECTURE :  i386-unknown-win32 - |
 AUTO_OFFSET :             26DF C |  AUTO_OFF_RET :             271D C |
*B0_7_IO :                   0 - | *B0_7_SER :                   40 - |
*B0_VCTTAB :               1007 C | *B1_6_HS :                    20 - |
*B1_6_IO :                   0 - | *B2_5_HS :                     4 - |
*B2_5_IO :                   0 - | *B3_4_HS :                    18 - |
*B3_4_IDM :                 10 - | *B3_4_IO :                     0 - |
 BADBLOCK :                   0 - | *BADCOUNT :                 14E0 C |
 BAD_RCVR_RET :            14AB C |  BAD_RC_ECC :               14A4 C |
 BAD_RC_SET :              149C C |  BAD_RC_SPR :               14A0 C |
 BAD_RECOVER :             147B C |  BANK_CALL :                 2CB - |
 BANK_RET :                 2F6 - | *BIGENDIAN :                   0 - |
 BLK_MOVE :                2112 C |  BLOCKID :                   200 - |
 BLOCKLENGTH :              214 - |  BLOCKMOVE :                210B C |
*BRANCHEXT :                 0 - |  BUF2ARRAY :                1266 C |
*BUF2CRC :                 147B C | *BUF2ECC :                  147D C |
*BUF2PW2 :                 1483 C |  BUF2_TO_RBUF :             204C C |
 BUF2_TO_WRBUF :           2061 C | *BUFDUMMY :                 1266 C |
*BUFFER2 :                 1267 C |  BUF_DAMAGE :                20 - |
 B_BLOCK :                   20 - |  B_MOVE :                   2097 C |
 B_RCVR_READ :             1483 C | *CASESENSITIVE :               0 - |
 CHECKSUM0 :               98C6 - |  CHECKSUM1 :                B74D - |
 CHKOFF_NOOFF :            2730 C |  CHK_CHAIN :                24BD C |
 CHK_CHK_BYTE :             428 - |  CHK_FATALSTAT :            184A C |
 CHK_FATSTAT1 :            1858 C |  CHK_FMTPARMS :             1782 C |
 CHK_HDPTR :               248F C |  CHK_INST :                 111E C |
 CHK_INST1 :               1110 C |  CHK_MVWRDATA :             26C5 C |
 CHK_OFFSET :              272B C |  CHK_OFF_RET :              2738 C |
 CHK_PARK :                2B93 C |  CHK_PASSWORD :             231D C |
 CHK_PK_JP :               2BCA C |  CHK_P_END :                233B C |
 CHK_P_LP :                2329 C |  CHK_SPR2 :                 2257 C |
 CHK_SPRCHK :              2246 C |  CHK_SPRCNT :               2376 C |
 CHK_SPRCNT1 :             1099 C |  CHK_SPRTBL :               21AB C |
 CHK_SPR_END :             2265 C |  CHK_SPR_RET :              238D C |
 CHK_SSTAT :               2651 C |  CLEARBITMAP :                20 - |
 CMND_PTR :                101B - |  CMND_PTRS :                1034 C |
 CNVRTLOGICAL :            1B04 C | *CONSTPI :        3.141592653589793 - |
 CRCERRL :                   80 - | *CRCSTAT :                    40 - |
 CUR_CYL :                   38 - |  CYLINDER :                   3A - |
 C_MAGDIR_ELSE :           2570 C |  DATARECAL :                  40 - |
 DATA_EX_ABORT :           13D9 C |  DATA_EX_CMND :             13DE C |
 DATA_EX_HANDLER :         13BD C |  DATA_EX_RDERR :            13CA C |
*DATE :                10/2/2013 - |  DEC_BADCNT :                  2 - |
 DELETESPARE :             247C C |  DEVICEPARAMS :             10B7 C |
 DEV_PARM_LENGTH :           17 - | *DIAGNOSTIC :                 20 - |
 DIR_FRWD :                  20 - |  DISKSTAT :                   35 - |
 DISK_SPEED :                20 - |  DONTLISTINCLS :               1 - |
 D_CHK_ELSE :              24E1 C |  D_RDH_1 :                  15D3 C |
 D_RDH_2 :                 15D6 C |  D_RDH_3 :                  145F C |
 D_RDH_CRC :               15F0 C |  D_RDH_END :                15F8 C |
 D_RDH_LP :                1602 C |  D_RDTRACK :                17BC C |
 D_RDTRK1 :                17C2 C |  D_RDTRK_DONE :             17E1 C |
 D_RDTRK_ERR :             17DC C |  D_RDTRK_NEXT :             17C5 C |
 D_READ :                  1591 C |  D_READHDR :                15C6 C |
 D_READ_END :              15A7 C |  D_RSTSRVO :                17AC C |
 D_WRITE :                 160F C |  D_WRITE_END :              1625 C |
 D_WRTRACK :               17B6 C |  ECC :                      2C03 C |
 ECCSTAT :                   80 - |  ECC_ALIGN :                2C34 C |
 ECC_ALIGNED :               40 - |  ECC_AL_1 :                 2C4E C |
 ECC_AL_2 :                2C56 C |  ECC_CORRECTABLE :            80 - |
 ECC_CRCT :                2C78 C |  ECC_CRCT_LP :              2C94 C |
 ECC_DIV8 :                2C7F C |  ECC_DONE :                   20 - |
 ECC_END :                 2CA8 C |  ECC_LD_LP :                2C13 C |
 ECC_LHJ_WHILE :           2C1C C |  ECC_SHFT_2 :               2C6F C |
 ECC_SHFT_ELSE :           2C6C C |  ECC_SHIFT :                2C62 C |
 EPROM2 :                    2 - |  EPROMTEST :                  E9 - |
 EPROM_FAIL :                40 - |  ERROR :                      80 - |
 EXCEPT_RETURN :           1425 C |  EXCPT_STAT :                 32 - |
*EXT_CLK :                   0 - |  EXT_POP :                   54F - |
 EXT_PUSH :                 522 - |  EX_BADBLOCK :                 4 - |
 EX_CASE_MAX :                A - | *EX_HDRBAD :                   8 - |
*EX_HDRSPR :                 A - |  EX_JP_TBL :                13EF C |
 EX_READERR :                 6 - |  EX_SPRBLOCK :                 2 - |
 EX_UNDETERMINED :            0 - | *FALSE :                       0 - |
*FBUF1ECC :                1266 C | *FBUFFER1 :                 1052 C |
*FDATAGAP :                1042 C |  FDATASYNC :                1050 C |
*FENDGAP :                 126C C | *FHDRGAP :                  102A C |
 FHDRSYNC :                103A C |  FHEADER :                  103C C |
 FLAGS :                     FC - | *FMTERROR :                   80 - |
*FMTLNTRL :                149E C |  FMTOFFSET :                149D C |
*FMTSRVOERR :                40 - | *FMTSUCCESS :                 20 - |
 FMTT_1 :                  2E03 C |  FORMAT :                   1771 C |
 FORMAT1 :                 1794 C |  FORMATARRAY :              1020 C |
 FORMATTRACK :             2D86 C |  FOUND :                       1 - |
 FREEP_NOPARK :            2B9E C |  FREE_EPROM :               2BDA C |
 FREE_MTRSPD :             2BEB C |  FREE_RAM :                 2BD0 C |
 FREE_RW :                 2BF5 C |  FREE_SCTRCNT :             2BE6 C |
*FREE_VECTOR :             100D C |  FRMTRECAL :                  70 - |
*FSCTRGAP :                1020 C | *FULLPMMU :                    1 - |
*GATE_CLK :                  10 - |  GDHDR_1 :                  20B8 C |
 GDHDR_2 :                 20EE C |  GDHDR_END :                2105 C |
 GDHDR_LP :                20C3 C |  GEN_CHK_BYTE :              439 - |
 GETNEWSPARE :             2390 C |  GET_CHS1 :                 1BE4 C |
 GET_CHS2 :                1BFB C |  GET_CHS3 :                 1C03 C |
 GET_CHS4 :                1C0B C |  GET_CYL_H_S :              1BBE C |
 GET_EL_DONE :             14E0 C |  GET_EOLIST :               14C6 C |
 GET_HEADPTR :             14AC C |  GET_HOSTP1 :               16AF C |
 GET_HOSTPARMS :           16A5 C |  GET_PREVIOUS :             24EF C |
 GET_PTR :                 151F C |  GET_SEQVAL1 :              15B6 C |
 GET_SEQVAL2 :             15BD C |  GET_SEQVAL3 :              15C5 C |
 GET_SEQVALUE :            15AA C |  GET_ZONE :                 2739 C |
 GNS_CHK_HI :              23F9 C |  GNS_END :                  2415 C |
 GNS_LP1 :                 23BC C |  GNS_LP1END :               23D2 C |
 GNS_LP2 :                 23D4 C |  GNS_LP2END :               23E8 C |
```

GNS_RC1 :	239C	C	GNS_RCOK :	23A9	C
GOODHDR :	20B6	C	GS_LDAO1 :	25A0	C
GS_LDNO1 :	25AB	C	GTSK_LDAO :	259E	C
GTSK_LDNO :	25A9	C	GTZN_LP :	2740	C
GTZN_PUSH :	275A	C	*HAS64 :	1	-
*HASDSP :	0	-	*HASFPU :	0	-
*HASPMMU :	0	-	HDA_TYPE :	1	-
HDR_MISMATCH :	20	-	HD_DIR_FRWD :	8	-
HD_DIR_REV :	0	-	HEAD :	3C	-
HIMAXCYL :	2	-	HIMAXLOGICAL :	0	-
HIREVNUMBER :	33	-	HISPR0 :	0	-
HISPR1 :	0	-	*HOME :	C0	-
HOSTCMNDBUF :	1483	C	HS0 :	10	-
HS1 :	20	-	*ID_TYPE :	4	-
*IMR :	FB	-	INC_BADCNT :	1	-
INC_SPRCNT :	0	-	INDEXMARK :	4	-
*INEXTMODE :	0	-	INIT_HICYL_A :	0	-
INIT_HICYL_B :	2	-	INIT_LOCYL_A :	0	-
INIT_LOCYL_B :	30	-	*INLWORDMODE :	0	-
*INMAXMODE :	0	-	*INSRCMODE :	0	-
INST_ABORT :	113B	C	INST_ABORT1 :	1142	C
*INSUPMODE :	0	-	*INT_OUT :	C0	-
*IPR :	F9	-	IRQ :	FA	-
IRQ_INDEX :	1	-	*IRQ_SECDN :	2	-
IRQ_SECTOR :	4	-	*ISCAN_SPRCHK :	1016	C
K1 :	10AF	-	L0198 :	198	-
L01E2 :	1E2	-	L01E7 :	1E7	-
L01EC :	1EC	-	L01F0 :	1F0	-
L01F8 :	1F8	-	L01FF :	1FF	-
L0201 :	201	-	L0203 :	203	-
L0205 :	205	-	L0265 :	265	-
L028E :	28E	-	L029B :	29B	-
L02AB :	2AB	-	L02C9 :	2C9	-
L03A7 :	3A7	-	L0450 :	450	-
L0459 :	459	-	L047C :	47C	-
L0489 :	489	-	L048F :	48F	-
L049B :	49B	-	L04A3 :	4A3	-
L04D0 :	4D0	-	L04DB :	4DB	-
L04E6 :	4E6	-	L04ED :	4ED	-
L0501 :	501	-	L051B :	51B	-
L0586 :	586	-	L059D :	59D	-
L05A3 :	5A3	-	L05BA :	5BA	-
L0651 :	651	-	L069D :	69D	-
*L1019 :	1019	C	*L101C :	101C	C
*L101F :	101F	C	*L1022 :	1022	C
*L1025 :	1025	C	*L1028 :	1028	C
*L102B :	102B	C	*L102E :	102E	C
*L1031 :	1031	C	L106A :	106A	C
L106E :	106E	C	L10B4 :	10B4	C
L114C :	114C	C	L1169 :	1169	C
L1175 :	1175	C	L119D :	119D	C
L11BB :	11BB	C	L11C7 :	11C7	C
L11E4 :	11E4	C	L11F1 :	11F1	C
L11FD :	11FD	C	*L1204 :	1204	C
L1208 :	1208	C	*L120D :	120D	C
*L121D :	121D	C	L1229 :	1229	C
L1237 :	1237	C	L1248 :	1248	C
L1252 :	1252	C	L1259 :	1259	C
L125C :	125C	C	L126A :	126A	C
L126E :	126E	C	L1273 :	1273	C
L1286 :	1286	C	L128A :	128A	C
L1469 :	1469	C	L147A :	147A	C
L1587 :	1587	C	L167C :	167C	C
L1721 :	1721	C	L172C :	172C	C
L1734 :	1734	C	L17E4 :	17E4	C
L17EE :	17EE	C	L19DB :	19DB	C
L1C77 :	1C77	C	L1C80 :	1C80	C
L1C98 :	1C98	C	L1CA6 :	1CA6	C
L1CAF :	1CAF	C	L1CC3 :	1CC3	C
L1CD0 :	1CD0	C	L1CEB :	1CEB	C
L1CF4 :	1CF4	C	L1CFA :	1CFA	C
L1D12 :	1D12	C	L1D20 :	1D20	C
L1D28 :	1D28	C	L1D59 :	1D59	C
L1D72 :	1D72	C	L1D80 :	1D80	C
L1D8E :	1D8E	C	L1D99 :	1D99	C
L1DA2 :	1DA2	C	L1DC1 :	1DC1	C
L1DCE :	1DCE	C	L1DD7 :	1DD7	C
L1DE0 :	1DE0	C	L1DED :	1DED	C
L1DFC :	1DFC	C	L1E53 :	1E53	C
L1E6F :	1E6F	C	L1E7D :	1E7D	C
L1E84 :	1E84	C	L1E8F :	1E8F	C
L1E9C :	1E9C	C	L1E9F :	1E9F	C
L1EAD :	1EAD	C	L1EB8 :	1EB8	C
L1ED9 :	1ED9	C	L1EDC :	1EDC	C
L1F4C :	1F4C	C	L1F8B :	1F8B	C
L1F99 :	1F99	C	L1FAC :	1FAC	C
L1FC6 :	1FC6	C	L1FD1 :	1FD1	C
L1FF4 :	1FF4	C	L2150 :	2150	C
L215B :	215B	C	L2161 :	2161	C
L220C :	220C	C	L2214 :	2214	C
L263A :	263A	C	L263D :	263D	C
L26A7 :	26A7	C	L26BC :	26BC	C
L26D1 :	26D1	C	L26DB :	26DB	C
L2902 :	2902	C	L2AC6 :	2AC6	C
L2ADB :	2ADB	C	L2AE3 :	2AE3	C
L2AEB :	2AEB	C	L2B08 :	2B08	C
L2B10 :	2B10	C	L2B1C :	2B1C	C
L2B48 :	2B48	C	L2B4F :	2B4F	C
L2B57 :	2B57	C	L2CE4 :	2CE4	C
L2CEC :	2CEC	C	L2D03 :	2D03	C
L2D29 :	2D29	C	L2D35 :	2D35	C
L2D49 :	2D49	C	L2D7D :	2D7D	C
L2D81 :	2D81	C	L2D9D :	2D9D	C
L2DA7 :	2DA7	C	L2DDD :	2DDD	C
L2DE0 :	2DE0	C	L2DE2 :	2DE2	C
L2E08 :	2E08	C	L2E0D :	2E0D	C
L2E12 :	2E12	C	L2E44 :	2E44	C
L2E4E :	2E4E	C	L2E51 :	2E51	C
L2E5B :	2E5B	C	L2E6B :	2E6B	C
L2E75 :	2E75	C	L2E7F :	2E7F	C
L2E91 :	2E91	C	L2E9B :	2E9B	C
L2EAB :	2EAB	C	L2ECA :	2ECA	C
L2ED3 :	2ED3	C	L2EDE :	2EDE	C
L2EED :	2EED	C	L2F41 :	2F41	C
L2F50 :	2F50	C	L2F53 :	2F53	C
L2F62 :	2F62	C	L2F65 :	2F65	C
L2F71 :	2F71	C	L2F87 :	2F87	C
L2F91 :	2F91	C	L2F94 :	2F94	C

```
 L2F99 :                    2F99 C |  L2F9E :                    2F9E C |
 L2FBC :                    2FBC C |  L2FC1 :                    2FC1 C |
 LASTSEEK_STAT :            124F C |  LD_BLKSTAT :               1C3D C |
 LD_BLK_BB :                1C3A C |  LD_BLK_DONE :              1C5E C |
 LD_BLK_HS :                1C61 C |  LD_BLK_HS1 :               1C6E C |
 LD_BLK_HS2 :               1C76 C |  LD_BLK_SEEK :              1C57 C |
*LD_OFF_VAL :                 20 - |  LD_S_CMND_LP :             2919 C |
 LD_TMSTMP :                221D C |  LD_TM_LP :                 221F C |
*LEDSTAT :                    1 - | *LISTON :                     1 - |
 LOADSTATUS :                46C - |  LOAD_HEADER :               3F0 - |
 LOAD_LOGICAL :              418 - |  LOAD_PASSWORD :             5E7 - |
 LOAD_SPRTBL :              2169 C |  LOAD_SPRTBL1 :             10A2 C |
 LOAD_SRVOCMND :            290F C |  LOGICALBLOCK :             1480 C |
 LOMAXCYL :                  68 - |  LOMAXLOGICAL :               34 - |
 LOREVNUMBER :               72 - |  LOSPR0 :                     AB - |
 LOSPR1 :                    53 - |  L_SPRTBL_LP :              2173 C |
 L_SPRTBL_MORE :            21A0 C |  L_SPR_END :                21FD C |
 L_SPR_INC :                21F7 C |  L_SPR_MOVE :               21F4 C |
*MACEXP :                     1 - |  MAP_TABLE :                161B C |
*MEM_EXT :                   20 - | *MEM_NORM :                   0 - |
 MIDMAXLOGICAL :             98 - |  MIDSPR0 :                   32 - |
 MIDSPR1 :                   65 - |  MINOFFSET :                  A - |
*MOD_N :                      1 - |  MOMCPU :                   8601 - |
*MOMCPUNAME :              Z8601 - |  MOVE4_LP :                 1192 C |
 MSWAIT :                   124 - |  MTRSPD :                   29C1 C |
 MTRSPD_END :               29EB C |  MTRSPD_IDX :               29D9 C |
 MTRSPD_LP1 :               29CE C |  MTRSPD_LP2 :               29E0 C |
 MTRSPD_LV :                2A02 C |  MULR0_M :                  1B80 C |
 MVWR_END :                 26D0 C |  NBRHDS_A :                   4 - |
 NBRHDS_B :                    2 - |  NBRSCTRS :                  20 - |
 NBRTRACKS_A :              131 - |  NBRTRACKS_B :               262 - |
*NBRZONES :                   10 - | *NESTMAX :                   100 - |
 NIL :                        80 - | *NOHDR_STATE :                0 - |
*NON_RETRIG :                 20 - | *NORMFMT_STATE :              A - |
 NORM_STATE :                  2 - |  NOTINTABL :                1BA3 C |
 NOT_SERVORST :                1 - |  NO_SPRTBL :                  1 - |
 NZERO_STAT :                  8 - | *OFFSET :                    10 - |
 OFFSET_ON :                   8 - |  OFFSET_SET :                 1 - |
 OFF_AUTO :                   40 - |  OFF_DIR_FRWD :              80 - |
 OFF_DIR_REV :                 0 - |  OK_INST :                  1128 C |
 ON_TRACK :                   80 - | *OPEN_DRAIN :                 0 - |
 OVERLAP :                  1C24 C |  P0 :                         0 - |
*P01M :                       F8 - | *P0_03_ADR :                  2 - |
*P0_03_IN :                   1 - | *P0_03_OUT :                  0 - |
*P0_47_ADR :                 80 - | *P0_47_IN :                  40 - |
*P0_47_OUT :                  0 - | *P1 :                         1 - |
*P1_ADR :                    10 - | *P1_IN :                      8 - |
*P1_OUT :                     0 - | *P1_TRI :                    18 - |
 P2 :                         2 - | *P2M :                       F6 - |
 P3 :                         3 - | *P3M :                       F7 - |
*PACKING :                    0 - | *PADDING :                    1 - |
 PARKCYL_A :                130 - |  PARKCYL_B :                26C - |
 PARKED :                    10 - |  PARK_HDS1 :                286D C |
 PARK_HEADS :               285D C | *PAR_OFF :                    0 - |
*PAR_ON :                     80 - |  PASSWORD :                 1003 C |
 PASSWRD1 :                F078 - |  PASSWRD2 :                 3C1E - |
 POSHDS_4 :                 25BF C |  POSHDS_41 :                25D1 C |
 POSHDS_42 :                25C3 C |  POSHDS_5 :                 25DD C |
 POSHDS_6 :                 25E1 C |  PRE0 :                       F5 - |
*PRE1 :                       F3 - |  PRO_RD_BB :                114F C |
*PWRRST :                     10 - | *R2MASK :                     0 - |
 R3MASK :                      F - |  RAM_FAIL :                  80 - |
 RBUF1ECC :                 122D C | *RBUF1PW :                  1234 C |
 RBUFFER1 :                 1019 C |  RBUF_TO_BUF2 :             208A C |
 RBUF_TO_SPR :              207A C |  RCVR_SPRT1 :               22F4 C |
 RCVR_SPRTBL :              22E7 C | *RDATAGAP :                 1011 C |
 RDBLK_ABNORM :             1AA6 C |  RDBLK_END :                1A9D C |
 RDBLK_NOHDR :              1AB2 C | *RDBLK_NORM :               1A7D C |
 RDBLK_RMOVE :              1AFB C |  RDBLK_RPT :                1A54 C |
 RDBLK_UNTIL :              1A93 C |  RDBLK_VECTOR :             1094 C |
 RDB_RPT1 :                 1AA1 C |  RDCRCERR :                   10 - |
 RDERROR :                    80 - |  RDHDRRECAL :                40 - |
 RDHDR_NORM :               202D C |  RDHD_SERVOERR :            2037 C |
 RDHD_SRVOOK :              203C C |  RDHERROR :                  80 - |
 RDHSRVOERR :                 40 - |  RDH_END :                  2046 C |
*RDH_STAT_ARRAY :           1008 - | *RDL_VECTOR :               100A C |
 RDNOHDRFND :                 20 - |  RDSMERR :                    8 - |
 RDSRVOERR :                  40 - |  RDUMMY :                   1018 C |
 RD_ABT_LP :                179F C |  RD_BADCRC :                1ADF C |
 RD_B_CRC_1 :               1AF7 C |  RD_HDRERR :                1AD5 C |
 RD_LEAVE :                 1146 C |  RD_SERVOERR :              1ADA C |
*READARRAY :                1000 C |  READBLOCK :                1A4B C |
 READHDR :                  2009 C | *READHDRARRAY :             1000 C |
 READSTATUS :                  0 - |  READ_ABORT :               1795 C |
 READ_CSTATUS :             1628 C |  READ_CSTAT_CP1 :           1682 C |
 READ_CSTAT_CP2 :           1684 C |  READ_CSTAT_LP1 :           163C C |
 READ_CSTAT_LP2 :           1686 C |  READ_ID :                  1539 C |
 READ_ID1 :                 155A C |  READ_ID2 :                 1560 C |
 READ_ID_LP1 :              1543 C |  READ_ID_LP2 :              1576 C |
 READ_SPRTBL :              157D C |  READ_SSTATUS :             168B C |
 RECOVERY :                   80 - | *RELAXED :                    0 - |
*RENDGAP :                  1233 C |  RESEEK :                   2531 C |
 RESETSERVO :               2922 C |  RESET_STMACH :              3C8 - |
 RESTORE :                  27CE C |  RESTORE_END :              2828 C |
 RESTORE_LP :               27FF C |  REST_UP1 :                 2826 C |
 REST_UP2 :                 27FB C |  REST_UPDATE :              280F C |
*RETRIG :                    30 - |  RETURN_VECTOR :            209A C |
*RHBUF1ECC :                122D C | *RHBUF1PW :                 1234 C |
*RHBUFFER1 :                1019 C | *RHDATAGAP :                1012 C |
*RHDRGAP :                  100A C | *RHDUMMY :                  1018 C |
 RHEADER :                  100B C |  RHENDGAP :                 1233 C |
*RHHDRGAP :                 100A C |  RHHEADER :                 100C C |
*RHSCTRGAP :                1000 C |  ROMSIZE :                  2000 - |
*RP :                         FD - | *RSCTRGAP :                 1000 C |
 RWI :                        40 - |  RWSTAT :                     34 - |
 RWTEST :                   2A63 C |  RWTEST1 :                  2A6F C |
 RWTEST2 :                  2A80 C |  RWTEST_ABORT :             2AD6 C |
 RWTEST_END :               2AB8 C |  RWTEST_EXIT :              2AD5 C |
 RWTEST_LP :                2A87 C |  RW_CMD_RD :                11DF C |
 RW_COMMON :                11D3 C |  RW_FAIL :                    2 - |
 RW_NEXT :                  2AB4 C |  SAVE_PREVIOUS :            24F5 C |
 SCRREG0 :                    40 - |  SCRREG1 :                    41 - |
 SCRREG2 :                    42 - |  SCRREG3 :                    43 - |
 SCRREG4 :                    44 - |  SCRREG5 :                    45 - |
 SCRREG6 :                    46 - |  SCRREG7 :                    47 - |
*SCRREG8 :                    48 - | *SCRREG9 :                    49 - |
*SCRREGA :                    4A - | *SCRREGB :                    4B - |
 SCRREGC :                    4C - | *SCRREGD :                    4D - |
```

```
*SCRREGE :                4E - | *SCRREGF :                4F - |
 SCTRCOUNT :            2A1C C |  SECTOR :                 3D - |
 SECTORMARK :              2 - |  SECTOR_CNT :              8 - |
 SEEK :                 2547 C |  SEEKCOMPLETE :            2 - |
 SEEK_ABT :             2605 C |  SEEK_END :             2610 C |
 SEEK_END1 :            2634 C |  SEEK_LP :              2558 C |
 SEEK_LSK :             2552 C |  SEEK_RESET :           2663 C |
 SEGPTRARRAY :          149F C |  SELECTHEAD :           266C C |
 SELFTEST :             298A C |  SELFTEST1 :            10A8 C |
 SELHD_RET :            2681 C |  SEL_HEAD1 :            2679 C |
 SEND_PARK :            1737 C |  SEND_RESTORE :         16F4 C |
 SEND_SEEK :            16D4 C |  SEND_SERVOCMND :       16B6 C |
 SEND_SK1 :             16E2 C |  SERIAL_IN :               8 - |
 SERIAL_OUT :              10 - |  SERVOCMND :            2769 C |
 SERVOCMND1 :           2771 C |  SERVOERR :               10 - |
 SERVOLOAD :            28D0 C |  SERVOOK :              2682 C |
 SERVORDY :                20 - | *SERVORST :                0 - |
 SERVOSTATUS :          27A1 C |  SERVOSTATUS1 :         27A9 C |
 SERVOSTORE :           287D C |  SERVO_FAIL :             10 - |
 SERVO_STLP1 :          27AE C |  SERVO_ST_ABORT :       27C3 C |
 SETBITMAP :               40 - |  SETSTATUS :            3C2 - |
 SET_AUTOOFFSET :       1741 C |  SET_RECOVERY :         1710 C |
 SHIFTANDXOR :          2CAB C | *SINGLE_PASS :             0 - |
 SIO :                     F0 - |  SIORDY :                 40 - |
 SLFTST_RESULT :           33 - |  SLFTST_TABLE :         2BC0 C |
*SLFTST_VECTOR :        1010 C |  SLF_LEAVE :            29BE C |
*SLF_RWTEST :           29B6 C | *SLF_SECTORS :          29AA C |
*SLF_STATE :            29B0 C | *SLF_TRACKS :           29A4 C |
 SOK_CHKRDY :           26B1 C | *SOK_PARK :             26B6 C |
 SPARE :                   10 - |  SPAREARRAY :           1495 C |
 SPAREBITMAP :          14E1 C |  SPAREBLOCK :           130A C |
 SPARECHECK :           163B C |  SPARECOUNT :           2342 C |
 SPAREEND :             1641 C |  SPARELENGTH :           1AC - |
*SPAREPW1 :             1495 C | *SPAREPW2 :             163D C |
 SPARETABLE :           14EB C |  SPARETMSTMP :          1499 C |
 SPH :                     FE - |  SPL :                    FF - |
 SPR :                  226A C |  SPRBLK_1 :             12F8 C |
 SPRBLK_2 :             12A8 C |  SPRBLK_3 :             12FB C |
 SPRBLK_HARD :             40 - |  SPRBLOCK :             128E C |
 SPRB_LP :              22A8 C |  SPRB_SEEK :            22C8 C |
 SPRCHK2 :              2228 C |  SPRCHKSUM :            2224 C |
 SPRCHK_LP :            2230 C |  SPRCNT_1 :             2350 C |
 SPRCNT_END :           2370 C |  SPRCOUNT :             14DF C |
 SPRTBL_TYPE :             8 - | *SPRTBL_VECTOR :        1013 C |
 SPRTBL_WARN :             40 - |  SPRTHRESH :               3 - |
 SPRWRV_1 :             12EC C |  SPR_END :              227D C |
 SPR_ENTER :            134B C |  SPR_LDBUF2 :           13B0 C |
 SPR_LDWRBUF :          13B4 C |  SPR_NEXT :             2311 C |
 SPR_ODD :              2277 C |  SPR_TO_RBUF :          2070 C |
 SPR_TO_WRBUF :         209D C |  SPR_WRVER :            12B3 C |
 SRCHLP :               1B23 C |  SRCHLPELSE :           1B8F C |
 SRCH_SPARE :           1B7D C |  SRVOCMNDBUF :          148B C |
 SRVORCVRY :            283A C |  SRVO_LD_WT :           28E0 C |
 SRVO_LD_WT1 :          28EE C |  SRVO_R_LEAVE :         2857 C |
 SRVO_R_SOK :           2848 C |  SRVO_ST_1 :            2896 C |
 SRVO_ST_2 :            28AA C |  SRVO_ST_3 :            28A1 C |
 SRVO_ST_EXIT :         28C1 C |  SRVO_WT_1 :            28B6 C |
 SRV_C_ABORT :          2796 C |  SRV_C_END :            279B C |
 SRV_C_LP :             2779 C |  SRV_ST_END :           27C8 C |
 SSTATUS0 :             1490 C |  SST_GETPTR :           1B2E C |
 SST_GETPTR1 :          1B55 C |  SST_GETPTR2 :          1B57 C |
 SS_NOHDR :              3B6 - |  SS_OPFAIL :             39F - |
 SS_READERR :            3AD - |  SS_SPRWARN :            3BC - |
*STACK_EXT :               0 - | *STACK_IN :                4 - |
 STACK_TOP :               80 - |  START_COMMAND :        10CE C |
*START_STATE :             0 - | *START_VECTOR :         1007 C |
 STATE_FAIL :              4 - |  STATUS0 :                 5F - |
 STATUS1 :                 5E - | *STATUSARRAY :          1015 - |
 STAT_SEEK :               2 - |  STAT_SPARE :              4 - |
*STDDEFZ8INC :             1 - |  STRT_FREEPROC1 :       10AE C |
 STRT_FREEPROCESS :     2B5D C |  ST_L_LP :              10F7 C |
 ST_MTR_GD :            29A1 C |  SYS_CHK_FTL :          182B C |
 SYS_INC_BLK :          18E1 C |  SYS_INC_LP :           18F2 C |
 SYS_RDERR :            1821 C |  SYS_RD_BB :            1826 C |
 SYS_RD_LP :            17FB C |  SYS_RD_M1 :            1839 C |
 SYS_RD_M2 :            181C C |  SYS_RD_MORE :          1819 C |
 SYS_READ :             17F8 C |  SYS_RW_SEEK :          18B8 C |
 SYS_RW_SK1 :           18D4 C |  SYS_RW_SK_LP :         18DC C |
 SYS_SET1 :             187F C |  SYS_SET2 :             18A8 C |
 SYS_SETUP :            1859 C |  SYS_WRCHK :            1940 C |
 SYS_WRE1 :             1924 C |  SYS_WRERR :            193D C |
 SYS_WRITE :            1908 C |  SYS_WRVER :            1972 C |
 SYS_WR_BB :            1963 C |  SYS_WR_LP :            190B C |
 SYS_WR_M1 :            192D C |  SYS_WR_MORE :          194D C |
 SYS_WR_NEXT :          1927 C |  SYS_WVCHK :            1988 C |
 SYS_WVERR :            1985 C |  SYS_WV_LP :            1975 C |
 SYS_WV_M1 :            1995 C |  SYS_WV_MORE :          199A C |
 S_A_XOR_END :          2CC9 C |  S_A_XOR_LP :           2CB0 C |
 S_BLK_END :            1348 C |  S_BLK_NEW :            1366 C |
 S_BLK_NEW1 :           139E C |  S_BLK_RPT :            133B C |
 S_BLK_UNUSE :          135A C |  S_BLOCK :                40 - |
*S_CMND_BYTE :             0 - |  S_CMND_LEN :              5 - |
 S_CNT_1 :              2A2D C |  S_CNT_2 :              2A38 C |
 S_CNT_3 :              2A3B C |  S_CNT_ABORT :          2A5E C |
 S_CNT_END :            2A4F C |  S_CNT_EXIT :           2A5B C |
 S_CNT_RETRY :          2A24 C |  S_C_BADEND :           236A C |
 S_C_BADINC :           2369 C |  S_C_SPARE :            236E C |
*S_DIFF_BYTE :             1 - |  S_GLBL_CYL :           257C C |
 S_LOAD :                  1 - | *S_NORM_STATUS :           1 - |
*S_OFF_BYTE :              2 - |  S_OK_END :             26BF C |
 S_RESTORE :            1706 C |  S_RST_1 :              2933 C |
 S_RST_2 :              293A C |  S_RST_ABORT :          295C C |
 S_RST_LD :             2961 C |  S_RST_LD1 :            296C C |
 S_RST_LD2 :            2981 C |  S_RST_RET :            2984 C |
 S_SCMND :              16C2 C |  S_STAT_1 :                1 - |
*S_STAT_2 :                2 - | *S_STAT_3 :                3 - |
*S_STAT_4 :                4 - | *S_STAT_5 :                5 - |
*S_STAT_6 :                6 - | *S_STAT_7 :                7 - |
*S_STAT_BYTE :             3 - | *S_STAT_O :                0 - |
 S_STORE :                 0 - |  T0 :                     F4 - |
*T0_CNTDIS :               0 - |  T0_CNTEN :                2 - |
 T0_LOAD :                 1 - | *T0_OUT :                 40 - |
*T1 :                     F2 - | *T1_CNTDIS :               0 - |
*T1_CNTEN :                8 - | *T1_EXT_CIK :              0 - |
*T1_INT_CLK :              2 - | *T1_LOAD :                 4 - |
*T1_OUT :                 80 - |  TEST0 :                2CD7 C |
 TEST0_DONE :           2CE3 C |  TESTBITMAP :             80 - |
 TESTMOD8 :             2CCC C | *TIME :                15:27:59 - |
```

```
*TIMER0 :                      10 - | *TIMER1 :                      20 - |
 TMR :                         F1 - |  *TOTEM_POL :                  1 - |
 TRACKCOUNT :                2A05 C |  *TRUE :                       1 - |
 TSC_BITMAP :                14E3 C |  *TSC_CLEAR :                150E C |
 TSC_END :                   1518 C |   TSC_LP1 :                  14E7 C |
 TSC_LP2 :                   14FE C |   TSC_MAP :                  1502 C |
 TSC_SCEND :                 1516 C |   TSC_SET :                  1514 C |
 TSTMD8_DONE :               2CD6 C |   TST_HEAD_A :                  3 - |
 TST_HEAD_B :                   1 - |   TST_HICYL_A :                 1 - |
 TST_HICYL_B :                  2 - |   TST_LOCYL_A :                31 - |
 TST_LOCYL_B :                 67 - |   TST_SCTR :                    0 - |
 UD_C_C_END :                20B3 C |   UPDATE_1 :                 2293 C |
 UPDATE_2 :                  229F C |   UPDATE_CUR_CYL :           20A7 C |
 UPDATE_ERR :                2144 C |   UPDATE_HDR :               2122 C |
 UPDATE_SPRTBL :             2280 C |   USEABLE :                    20 - |
 USED :                        40 - |   USEECC :                      4 - |
 USERBLK_TYPE :                 2 - |   USER_TYPE :                   4 - |
 VCTRB0_BC :                 109D C |   VCTRB0_RET :               1091 C |
*VERSION :                   142F - |  *WBUF1ECC :                 1235 C |
*WBUF1PW :                   123D C |   WBUFFER1 :                 1021 C |
 WDATAGAP :                  1011 C |   WDATASYNC :                101F C |
 WENDGAP :                   123B C |  *WHDRGAP :                  100A C |
 WHEADER :                   100B C |   WRBLK_ABNORM :             1A22 C |
 WRBLK_END :                 1A19 C |   WRBLK_NOHDR :              1A2E C |
*WRBLK_NORM :                19FB C |   WRBLK_NVLD :               1A45 C |
 WRBLK_RPT :                 19C2 C |   WRBLK_RPT1 :               1A1D C |
 WRBLK_SERVOERR :            1A40 C |   WRBLK_UNTIL :              1A0D C |
 WRBLK_VECTOR :              108E C |   WRBUF_TO_BUF2 :            2056 C |
 WRBUF_TO_SPR :              2084 C |   WRERROR :                    80 - |
 WRGAP_LP :                  19CD C |  *WRITEARRAY :               1000 C |
 WRITEBLOCK :                19B9 C |  *WRK_CNTRL :                  F0 - |
 WRK_SCR :                     40 - |   WRK_SYS :                    10 - |
 WRNOHDRFND :                  20 - |   WRSMERR :                     8 - |
 WRSRVOERR :                   40 - |   WRTNVLDL :                   40 - |
 WRVER_COMMON :              11A4 C |   WRVER_RET :                11CE C |
 WRVER_RET1 :                11BE C |   WR_BBLOCK :                115C C |
 WR_OP :                       20 - |  *WR_SERVOOK :               1A05 C |
 WR_SPR1 :                   1760 C |   WR_SPRTBL :                174B C |
*WSCTRGAP :                  1000 C |   X_BADBLOCK :               1407 C |
 X_BB_LP :                   1409 C |   X_HDRBADBK1 :              143D C |
 X_HDRBADBK2 :               143F C |   X_HDRBADBLOCK :            142B C |
 X_HDRSPARE :                145A C |   X_HDR_CRCT :               1451 C |
 X_READERR :                 141B C |   X_SPARE :                  1400 C |
 X_SPR_CALL :                1402 C |   X_TO_BUF2 :                2093 C |
 X_TO_SPR :                  207E C |   X_UNDETER :                13FB C |
 YMASK :                        F - |   ZEROHEADER :                58E - |
 ZERO_ELEMENT :              2510 C |   ZERO_E_LP :                251D C |
 ZONESHIFT :                    5 - |   ZONE_TABLE :               1641 - |

   1028 symbols
    187 unused symbols
```

```
  codepages:
  ----------

STANDARD (0 changed characters)


0.39 seconds assembly time

   7272 lines source file
      2 passes
      0 errors
      0 warnings
```